CORRELATION DIMENSION COMPUTATION OF EEG TIME SERIES

by

Ersin TAŞKIN

BS. in EE, Boğaziçi University, 1994

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Biomedical Engineering

Boğaziçi University

1997

ACKNOWLEDGEMENTS

It is my pleasant duty to express my sincere gratitude to Assoc. Prof. Dr. Halil Özcan GÜLÇÜR, my thesis supervisor, for his precious support, Assoc. Prof. Dr. Yağmur DENİZHAN, my thesis co-supervisor, and Assoc. Prof. Dr. Tamer DEMİRALP, for their inspirations, guidance and reviews, and Prof. Dr. Yorgo ISTEFANOPULOS for his critics. I would also like to thank Mr. Mert İNCEİPLİK, Mr. Mehmet Vefa KARATAY, Mr. Umur ÖZKUL, and other members of the BOĞAZİÇİ UNİVERSİTY CHAOS GROUP, for their valuable discussions.

Finally, I am grateful to my family for their unlimited love, support and patience.

Ersin TAŞKIN

ABSTRACT

In this thesis, a software package, to be used in correlation dimension (D_2) computation of human electroencephalograms (EEG), is developed. The main algorithm calculating the D_2 is based on the Grassberger Procaccia (G-P) theorem. Rigorous algorithms are developed to speed up the process without a loss in accuracy.

The package is developed under Windows 95 in Delphi 2.0 environment, which enables the program to be a natural part of the contemporary 32 bit environments, and provides the user with a user friendly graphical user interface.

The software developed is applied to two groups of signals: 1) Signals, whose D_2's are known a priori- a sinusoidal, a Henon map, and a segment of white noise. 2) EEG samples recorded (whose D_2's have to be calculated from experimental measurements) under various experimental conditions, from various groups of individuals, which comprise steady state responses of flash driven subjects, alpha dominant waves, beta dominant waves, and beta dominant waves from subjects having minor epilepsy. The numerical results obtained are analyzed, and compared with those in the literature.

İNSAN ELEKTROENSEFALOGRAMININ KORRELASYON BOYUTUNUN HESAPLANMASI

ÖZET

Bu tezde, insan elektroensefalogramının (EEG) korrelasyon boyutunun (D_2) hesaplanmasında kullanılmak üzere bir yazılım geliştirilmiştir. D_2 hesaplanmasında kullanılan ana algoritma Grassberger Procaccia (G-P) teoremine dayanmaktadır. Program hızını, kesinlikten ödün vermeden, arttırmak için karmaşık algoritmalar uygulanmıştır.

Söz konusu yazılım Windows 95 işletim sisteminde Delphi 2.0 ortamında geliştirilmiştir. Bu sayede program günümüzün 32 bit ortamlarının doğal bir parçası olarak çalışmakta olup, 32 bit özelliğinin getirdiği güçten tam olarak yararlanmaktadır ve bunun sonucu olarak kullanıcıyla dost bir kullanıcı arabirimi sunmaktadır.

Tez çerçevesinde geliştirilmiş bulunan yazılım iki grup işaretin D_2 hesaplanmasında kullanılmıştır. 1) D_2'u bilinen işaretler -sinüs, Henon haritası, beyaz gürültü. 2) D_2'u deneysel olarak hesaplanmak zorunda olan, çeşitli deney koşullarında çeşitli denek gruplarından kaydedilen EEG işaretleri. Söz konusu işaretler ışık sürümlü deneklerin kararlı durum tepkilerini, alfa bandı ağırlıklı işaretleri, sağlıklı bireylerden ve düşük düzey sara vakalarından alınan beta bandı ağırlıklı işaretleri içermektedir. Sayısal sonuçlar incelenmiş ve literatürdeki örnekleriyle karşılaştırılmıştır.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iii

ABSTRACT iv

KISA ÖZET v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

LIST OF TABLES x

LIST OF SYMBOLS xi

1. INTRODUCTION 1

2. THEORY 3

2.1. DYNAMICAL SYSTEMS 3

2.2. STEADY STATE BEHAVIOR AND LIMIT SETS 3

2.3. DIMENSION 5

2.3.1. Capacity Dimension 5

2.3.2. Information Dimension 6

2.3.3. Correlation Dimension 7

2.4. RECONSTRUCTION OF THE ATTRACTOR 7

3. PRACTICAL CONCERNS 9

3.1. PRACTICAL CONCERNS IN CORRELATION DIMENSION COMPUTATION 9

3.1.1. Choosing Embedding Dimension 9

3.1.2. Choosing Time Delay 10

3.1.2.1. Average Peak to Peak Distance 10

3.1.2.2. Linear Autocorrelation 11

3.1.2.3. Mutual Information 12

3.2. PRACTICAL CONCERNS IN EEG DATA RECORDING 13

4. IMPLEMENTATION - DYNAMICAL STSEMS ANALYZER (DSA)..... 17
4.1. DSA GRAPHICAL USER INTERFACE IN BRIEF..... 18
5. APPLICATION AND RESULTS..... 20
5.1. SINUS..... 20
5.2. HENON MAP..... 21
5.3. EEG TIME SERIES..... 22
6. CONCLUSION..... 33
6.1. ANALYSIS AND COMPARISON OF NUMERICAL RESULTS..... 33
6.2. FUTURE DIRECTIONS..... 36
6.2.1. Future Advancements on DSA..... 36
6.2.2. The Future..... 37
APPENDIX 38
REFERENCES..... 80

LIST OF FIGURES

		Page
FIGURE 6.1	Sin(x). #points = 700	20
FIGURE 6.2	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of Sinus(x). D_2 = 1.03 +/- 0.06	20
FIGURE 6.3	Henon map. #points = 1500	21
FIGURE 6.4	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of Henon map. D_2 = 1.24 +/- 0.02	21
FIGURE 6.5	EEG signal from a flash driven subject (30 flashes). #points = 1280	22
FIGURE 6.6	Steady regime portion of the signal in Figure 6.5. #points=512	22
FIGURE 6.7	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of the signal in Figure 6.6. D_2 = 1.80 +/- 0.02	22
FIGURE 6.8	EEG signal from a flash driven subject (30 flashes). #points = 1280	23
FIGURE 6.9	Steady regime portion of the signal in Figure 6.8. #points=512	23
FIGURE 6.10	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of the signal in Figure 6.9. D_2 = 3.77 +/- 0.09	23
FIGURE 6.11	An alpha dominant EEG signal. #points = 1024	24
FIGURE 6.12	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.11. D_2 = 3.60 +/- 0.09	24
FIGURE 6.13	An alpha dominant EEG signal. #points = 1024	25
FIGURE 6.14	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.13. D_2 = 3.89 +/- 0.13	25
FIGURE 6.15	An alpha dominant EEG signal. #points = 5120	26
FIGURE 6.16	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.15. D_2 = 5.94 +/- 0.13	26
FIGURE 6.17	A beta dominant EEG signal. #points = 5120	27
FIGURE 6.18	D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.17. D_2 = 7.88 +/- 0.08	27
FIGURE 6.19	A Beta dominant EEG signal. #points = 1024	28

FIGURE 6.20 D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.19. D_2 = 7.78 +/- 0.12 28

FIGURE 6.21 A Beta dominant signal. #points = 1024 29

FIGURE 6.22 D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.21. D_2 = 8.15 +/- 0.24 29

FIGURE 6.23 A beta dominant EEG signal from a subject suspected of minor epilepsy. #points = 1024 30

FIGURE 6.24 D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.23. D_2 = 6.01 +/- 0.22 30

FIGURE 6.25 A beta dominant EEG signal from a subject of minor epilepsy suspect. #points = 1024 31

FIGURE 6.26 D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of signal in Figure 6.25. D_2 = 5.51 +/- 0.52 31

FIGURE 6.27 White noise. #points = 1024 32

FIGURE 6.28 D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of white noise. D_2 : No saturation 32

LIST OF TABLES

Page

Table 6.1 Numerical Results 33

LIST OF SYSMBOLS

D_2	correlation dimension
G-P	Grassberger-Procaccia algorithm
DSA	Dynamical Systems Analyzer
GUI	Graphical User Interface
EEG	electroencephalogram
x_0	initial state
t_0	initial time
f(x)	field vector
x_k	k^{th} state variable of discrete time system
$\{x_k\}_{k=0}^{\infty}$	orbit of the discrete-time system
U	neighborhood
$\phi_t(x)$	trajectory of state variable x in state space
L(x)	limit set
$N(\varepsilon)$	number of grids (volume elements) needed to cover the attractor
ε	radius of grids
D_{cap}	capacity dimension
D_I	information dimension
$S(\varepsilon)$	entropy at a resolution of ε
P_i	relative frequency with which a typical trajectory enters the i^{th} volume element of a covering
$C(\varepsilon)$	correlation integral
τ_w	reconstruction window
τ	time delay
m	embedding dimension

d_E	minimum embedding dimension for safe reconstruction as per Taken
d_A	true dimension
$C_L(\tau)$	linear autocorrelation function of a signal for a given τ
$\bar{x}$	average of a set of x values
$I(\tau)$	mutual information
EMG	electromyographic
EOG	electrooculographic
ECG	electrocardiographic
DCx	dimensional complexity
N	number of data points
#points	number of data points
A/D	Analog to digital conversion
m(r)	slope at radius r in log(C(r))-log(r) plot
Q	quality mesure of goodness fit in linear regression
2D	two dimensional
FNNs	false nearest neighbors
r_{up}	upper grid radius
r_{low}	lower grid radius
$\#_{grids}$	number of grids
AI	Artificial Intelligence

1. INTRODUCTION

Recent progress in the science of nonlinear systems has given birth to new paradigms which in turn gave rise to many new branches of research. This has brought subjects, which had been applied stochastic approaches, into the domain of determinism. It has been shown mathematically that systems where there is chaos, i.e. systems which are unpredictable over a long time scale, can have dynamics with a few degrees of freedom. In 1963, Lorenz applied concepts of nonlinear dynamics to the convection phenomenon in hydrodynamics in order to describe atmospheric turbulence (Navier-Stokes equation). He demonstrated the possibility that the unpredictable or *chaotic* behavior observed in an infinite dimensional system might be caused by a three-dimensional dynamical system. This unpredictability is termed as *sensitive dependence on initial conditions.* It means that even an infinitesimal difference in the measurement of the current state of the system results in an exponential divergence of future behavior bringing a categorical difference in the output. Since measurements with infinite precision are theoretically impossible (the theoretical lower limit is put by the *Heisenberg's Uncertainty Principle*), chaotic systems are unpredictable, although deterministic. Despite the temporal unpredictability of *dissipative* (systems that lose energy with time) chaotic systems, when stationary, their long-term behavior follows a structured pattern, which is revealed by plotting the trajectory of the system in the state space, a mathematical space that captures full dynamical behavior of the system (see Chapter 2). In the state space these trajectories accumulate in a finite region of finite dimension -known as the *strange attractor* (see Chapter 2). The *strange attractor* has some invariant measures, which possess valuable information about the dynamics of the system. These include mutual information, autocorrelation, Kolmogorov entropies, Lyapounov exponents, "attractor dimension", etc. Hence, one can analyze chaotic systems despite their unpredictability.

After Lorenz, Başar [1], and Başar & Röschke [2] described that the electroencephalogram (EEG) might reflect properties of a *strange attractor,* assuming the EEG to be a *chaotic* (deterministic but unpredictable) attractor. Following this

study, recent years have witnessed the increasing application of nonlinear dynamics to the analysis of the human EEG. Since the pioneering analysis by Babloyantz et al. [3], several investigators have undertaken the nonlinear analysis of the EEG using Grassberger and Procaccia's [4] algorithm in order to evaluate the correlation dimension (D_2). The object of this thesis is to develop a software package that can be used in such an undertaking.

This thesis is organized as follows:

Chapter 2 of this thesis forms a theoretical background for correlation dimension computation. The basic terms that are used throughout the thesis are defined in this chapter.

In **Chapter 3,** practical concerns are outlined. The first section of the chapter includes practical concerns in D_2 computation, such as choosing optimum time delay, embedding dimensions, etc., and describes and compares various methods employed. In the second section of the chapter, practical considerations in EEG time series recording are described.

Chapter 4, describes **Dynamical Systems Analyzer (DSA)**, the software package developed in this thesis. Brief information on the technical capacity, and the Graphical User Interface (GUI) of DSA is given.

In **Chapter 5**, D_2 computations, by DSA, of various signals, such as computer generated iterations, white noise, and EEG segments, are presented.

Finally in **Chapter 6**, numerical results found in **Chapter 5** are analyzed and a conclusion of the thesis is made. Some ideas for future studies on D_2 computation of EEG data is given.

2. THEORY

2.1. DYNAMICAL SYSTEMS

An n^{th} order continuous-time dynamical system is defined by the state equation

$$dx/dt = f(x), \quad x(t_0) = x_0 \tag{1}$$

where $x(t) \in R^n$ is the state at time t and f: $R^n \rightarrow R^n$ is called the vector field. Since the vector field does not depend on time the initial time may always be taken as $t_0=0$. The solution to (1) with initial condition x_0 at time t=0 is called the trajectory and is denoted by $\phi_t(x_0)$. The mapping $\phi_t : R^n \rightarrow R^n$ is called the flow of the system.

In the discrete-time dynamical systems the equation becomes

$$x_{k+1} = f(x_k), \quad k = 0,1,2, \ldots \tag{2}$$

where $x_k \in R^n$ is called the state, and f maps the state x_k to the next state x_{k+1}. Starting with an initial condition x_0, repeated applications of the map give rise to a sequence of points $\{x_k\}_{k=0}^{\infty}$ called an orbit of the discrete-time system.

2.2. STEADY STATE BEHAVOUR AND LIMIT SETS

Steady state refers to the asymptotic behavior as $t \rightarrow \infty$. It is required that the steady state be bounded.

A point y is a limit point of x if, for every neighborhood U of y, $\phi_t(x)$ repeatedly enters U as $t \rightarrow \infty$.

The set of all limit points of x is called the *limit set* L(x) of x. Limit sets are closed and invariant under ϕ_t.

A limit set L is *attracting* if there exists an open neighborhood of U of L such that $L(x) = L$ for all $x \in U$.

Attracting limit sets are of special interest since non-attracting sets cannot be observed in physical systems. In conservative systems limit set covers the whole state-space, while in dissipative systems the volume is contracted to a finite space. Physical experiments and computer simulations with dynamical systems usually exhibit transient behavior followed by what seems to be an asymptotic regime. Therefore the point $\phi_t(x)$ representing the system should eventually lie on an attracting set (or near it). However, in practice smaller sets, which are called *attractors*, will be obtained (they should be carefully distinguished from attracting sets). This is because some parts of an attracting set may not be attracting. Instead giving a precise mathematical definition of an attractor, an *operational* definition shall be used in this context, that it is a set on which *experimental points* $\phi_t(x)$ accumulate for large t. An attractor is by definition invariant under a dynamical evolution, and this creates a self- similarity that is often strikingly visible.

In most systems that has been in the domain of engineering so far the attractor can be a point, as in the case of an equilibrium point; a circle, as in the case of a periodic steady state; or an n-torus, as in the case of a quasi-periodic solution (with n base frequencies).

In chaotic systems, however, the attractor is not an object of Euclidean Geometry. Instead it has a fine structure with repetitions on various scales and is said to be *fractal*. Such attractors are referred to as *strange attractors*. The property of being chaotic is actually a more important dynamical concept than being fractal, and therefore Feigenbaum attractor is no longer referred to as a strange attractor by Eckmann and Ruelle [5]. We therefore define strange attractor to be an attractor with *sensitive dependence on initial conditions*. The notion of strangeness refers thus to the

dynamics on the attractor , and not just to its geometry. It applies whether the time is discrete or continuous. This is again an operational definition rather than a mathematical one.

2.3. DIMENSION

One of the invariant measures of dynamical systems is the *dimension* of its attractor. Thus systems can be classified using the concept of dimension. Furthermore, for chaotic systems the dimension of the attractor can be used to explore information about the system. An attractor can be defined to be n-dimensional if, in a neighborhood of every point, it looks like an open set of R^n (i.e., it is diffeomorphic[1] to an open subset of R^n). This is how the dimension of a manifold is defined in differential topology. For, instance a limit cycle is one dimensional since it looks locally like an interval. A 2-torus is two dimensional since, locally, it resembles an open subset of R^2. An equilibrium point is considered to have zero dimension. The neighborhood of any point of a strange attractor, however, has a fine structure and does not resemble any Euclidean space. Strange attractors are not manifolds and do not have integer dimension. There are several ways to generalize dimension to the fractional base, the *capacity dimension, information dimension,* and *correlation dimension* being the most generally applied. There are various algorithms developed to find these dimensions, and development of such algorithms together with noise reduction is an open area in nonlinear time series analysis.

2.3.1. Capacity Dimension

The simplest type of dimension is *capacity*, also referred to as *fractal dimension*. The idea is follows. Cover an attractor with volume elements (spheres, cubes, etc.), each with a diameter ε . Let $N(\varepsilon)$ be the number of volume elements needed to cover the *attractor* A. As ε is made smaller the sum of volume elements approaches the volume of A. If A is a D-dimensional attractor (D is not necessarily an integer), then for ε small, the number of volume elements needed to cover A is

[1] A function f is a diffeomorphism if f^{-1} exists and both Df and Df^{-1} exist and are continuous.

inversely proportional to ε^{D}, that is, $N(\varepsilon) = k\varepsilon^{-D}$ for some constant k. The definition of D_{cap} is obtained by solving this equation for D and taking the ε limit.

$$D_{cap} := \lim_{\varepsilon \to 0} \frac{\ln N(\varepsilon)}{\ln(1/\varepsilon)} \tag{3}$$

If the limits does not exist then D_{cap} is undefined. Clearly, for manifolds, D_{cap} is an integer; however, for objects that are not manifolds, D_{cap} is a noninteger.

2.3.2. Information Dimension

D_{cap} is a purely metric concept and does not utilize information about the time behavior of the dynamical system. *Information dimension* is a probabilistic type of dimension, defined in terms of the relative frequency of visitation of a trajectory. The information dimension is defined in a similar setting as the capacity dimension by

$$D_I := \lim_{\varepsilon \to 0} \frac{\ln S(\varepsilon)}{\ln(1/\varepsilon)} \tag{4}$$

where

$$S(e) := -\sum_{i=1}^{N(\varepsilon)} P_i \ln P_i \tag{5}$$

P_i is the relative frequency with which a typical trajectory enters the i^{th} volume element of the covering.

Readers familiar with information theory will recognize $S(\varepsilon)$ as entropy- the amount of information needed to specify the state of the system to an accuracy of ε if the state is known to be on the attractor.

2.3.3. Correlation Dimension

Another probabilistic type of dimension is the *correlation dimension* (D_2) defined by

$$D_2 := \lim_{\varepsilon \to 0} \frac{\ln \sum_{i}^{N(\varepsilon)} P_i^2}{\ln \varepsilon} \tag{6}$$

To help interpret the numerator of the above equation, suppose N points of a trajectory have been gathered either through simulation or from measurements. Define the correlation integral, $C(\varepsilon)$, as

$$C(\varepsilon) := \lim_{N \to \infty} \frac{1}{N^2} \{\text{the number of pairs of points } x_i, x_j \text{ such that } | x_i - x_j | < \varepsilon \} \tag{7}$$

Then

$$D_2 := \lim_{\varepsilon \to 0} \frac{\ln C(\varepsilon)}{\ln \varepsilon} \tag{8}$$

D_2 can be easily calculated because $C(\varepsilon)$ is easy to estimate. In this thesis an emphasis on correlation dimension has been given due to its computational advantages.

2.4. RECONSTRUCTION OF THE ATTRACTOR

Recent progress in the theory of nonlinear dynamical systems has provided new methods for the study of time series in such fields as hydrodynamics, chemistry, climatic variability and human brain activity. The study of such complex systems may be performed by analyzing experimental data recorded as a series of measurements in time of a pertinent and easily accessible variable of the system. In most cases, such variables describe a global or averaged property of the systems. For example, a time

series may be obtained by recording at regular time intervals the mean electrical activity of a portion of the mammalian cortex. Although it may seem that such data offer only one dimensional view of the brain, this is not the case: it can be shown that a time series may provide information about a large number of pertinent variables, which may subsequently be used to explore and characterize the system's dynamics.

Thus it is possible to reconstruct and compute the dimension of the chaotic attractor in the brain by analysis of the time series of EEG data. Let us assume that the dynamics of the brain activity is described by a set of $\{X_0(t), X_1(t), \ldots, X_{m-1}(t)\}$ variables satisfying a system of first-order differential equations. A differential equation of order n with a single variable X_0, accessible from experimental data, is equivalent to the original set. Now both X_0 and its derivatives , therefore the ensemble of m variables, can be obtained from a single time series. However, it is more convenient to construct another set of variables$\{ X_0(t), X_0(t+\tau), X_0(t+(m-1)\tau)\}$, which is topologically equivalent to the original set. Here X_0 may represent electrical potential V recorded by EEG. These variables are obtained by shifting the original time series by fixed time lag τ ($\tau = L\Delta t$, where L is an integer and Δt is the time interval between successive samplings).

These vector variables span a state space, which allows the drawing of the state portrait of the system, or more precisely its projection into a low-dimensional subspace of the full state space. If the dynamics are reducible to deterministic laws, the system reaches in time a state of permanent regime. This fact is reflected by the convergence of families of state trajectories toward a subset of the state space. This invariant is the attractor.

3. PRACTICAL CONCERNS

3.1. PRACTICAL CONCERNS IN CORRELATION DIMENSION COMPUTATION

3.1.1. Choosing Embedding Dimension

In the above layout m is referred to as the *embedding dimension*, and τ is referred to as the *time delay* or *lag*. Takens [6] argue that if an applied dimension d, which is an integer, is larger than 2 d_A (d_A being the true dimension), which can be fractional, then the attractor as seen in the space with lagged coordinates will be smoothly related to the true attractor, as viewed in the original coordinates, which we do not know. In practice, their theorem tells us that. If we have chosen d large enough, physical properties of the attractor that we wish to extract from the measurements will be the same when computed in lagged coordinates and when computed in the physical coordinates. The procedure of choosing sufficiently large d is formally known as embedding, and any dimension that works is called an embedding dimension d_E. Once one has achieved a large enough $d=d_E$, then any $d>=d_E$ will also provide an embedding.

One way of determining the minimal embedding dimension necessary is the method of *false nearest neighbors* (FFNs). State-space points that are close together because of under embedding and not because they are really close on the attractor are FFNs. When the percentage of FFNs drops below some criterion, the attractor is taken to be fully unfolded, and further increase in d is unnecessary- the reader is referred to [7] for details.

The usual practice is starting from a small enough dimension, which is increased until system invariants reach a saturation, which is employed in this project.

3.1.2. Choosing Time Delay

Unfortunately, Takens' theorem assumes the availability of an infinite amount of noise-free data. (In this ideal case, one may choose any value of τ that does not lead to a degenerate reconstruction). In real experiments, i.e. experiments with noisy, finite data sets a value of τ that is too small results in little information gain, i.e. *redundance* , between successive delay coordinates, and the reconstructed trajectory becomes compressed along the main diagonal, or identity line, of the embedding space. With chaotic systems and large value of τ, successive delay coordinates may become causally unrelated, and the reconstruction is no longer representative of the true dynamics. Casdagli et. al. [8] calls this phenomenon *irrelevance*.

Previous studies suggest that it may be more appropriate to fix the *reconstruction window* τ_w, rather than τ alone, where τ_w is the length of the interval spanned by the first and the last delay coordinates:

$$\tau_w = \tau(m\text{-}1). \tag{9}$$

For example, Martinerie et al. [9] showed that the correlation integral is sensitive to τ_w, but not to τ, and m individually. Hence, the problem addressed in this context is reformulated here to that of choosing the proper τ once the embedding dimension is fixed.

Unfortunately, there is no mathematically well established theoretical background for determination of *time delay* values for a specific time series data. There are various methods each of which has advantages and drawbacks. Some of the most popular methods employed are as follows.

3.1.2.1. Average Peak to Peak Distance

The simplest and fastest approach for choosing τ is referring average peak to peak distance as an upper limit. It does not provide one with a good enough τ all the

time but for simple and fast applications it is a good way to take a guess. Heuristically, points that are almost average peak to peak distance apart are not so close to each other to be redundant, and not so far from each other to be irrelevant.

3.1.2.2. Linear Autocorrelation

The second easiest and fastest approach is to consider the values of x(n) as chosen from some unknown distribution. Then computing the *linear autocorrelation function* $C_L(\tau)$

$$C_L(\tau) = \frac{1/N\sum_{m=1}^{N}[x(m+\tau)-\bar{x}][x(m)-\bar{x}]}{1/N\sum_{m=1}^{N}[x(m)-\bar{x}]^2} \tag{10}$$

where

$$\bar{x} = 1/N\sum_{m=1}^{N}x(m) \tag{11}$$

and looking for that time delay where $C_L(\tau)$ first passes through 0, would give us a good hint of a choice for τ. Indeed this does give a good hint. It tells us, however, the independence of the coordinates in a linear fashion. To see this, note that if you want to know whether two measurements x(n) and x(n+τ) depend linearly on each other on the average over the observations, it is found that their connection, in a least-squares sense, is through the correlation matrix just given.

That is if we assume that, the values of x(n) and x(n+τ) are connected by

$$x(n+\tau)-\bar{x} = C_L(\tau)[x(n)-\bar{x}], \tag{12}$$

then minimizing

$$\sum_{n=1}^{N}\{x(n+\tau)-\bar{x}-C_L(\tau)[x(n)-\bar{x}]\}^2, \qquad (13)$$

with respect to $C_L(\tau)$, immediately leads to the definition of $C_L(\tau)$ above [10].

Choosing τ to be the first zero of $C_L(\tau)$ would then, on average over the observations, make x(n) and x(n+τ) linearly independent. What this may have to do with their nonlinear dependence or their utility as coordinates for a nonlinear system is not addressed by all this. Since we are looking for a *prescription* for choosing τ, and this prescription must come from considerations beyond those in the embedding theorem, linear independence of coordinates may serve, but we prefer another point of view, one that stresses an important aspect of chaotic behavior- namely the viewpoint of information theory [11] - and leads to a nonlinear notion of independence.

3.1.2.3. Mutual Information

In contrast to the linear dependence measured by autocorrelation, mutual information I(τ), supplies a measure of general dependence. Therefore, I(τ) is expected to provide a better measure of the shift from redundance to irrelevance with nonlinear systems. Mutual information answers the following question: Given the observation of x(n), how accurately can one predict x(n+τ)? Thus, successive delay coordinates are interpreted as relatively independent when the mutual information is small. The greatest independence, i.e., the lowest I(τ), is associated with the least redundance and, therefore, the best attractor reconstruction. For this reason, they selected τ as the lag that produces a local minimum I(τ). Typically, the first local minimum is the preferred choice. (Note that Fraser and Swinney originally considered this method as establishing the value of τ, not τ_w. However, in light of the work by Martinerie et al. [9] it seems more appropriate to interpret the result as τ_w.).

3.2. PRACTICAL CONSIDERATIONS IN EEG DATA RECORDING

The unavailability of unlimited , noise-free EEG data. Technically, Grassberger, Procaccia (G-P) theorem requires an unlimited amount of noise free data. In the context of EEG psychophysiology, these conditions are of course never met. In fact, the length of EEG segments typically employed is relatively short in order to avoid gross EEG artifacts, as well as increase the probability of *system stationarity* (constant dynamics, an assumption of all dimension estimation methods). However, even the cleanest looking EEG segment probably still contains some residual, non-neural signals in the form of slight electrooculographic (EOG), electromyographic (EMG), electrocardiographic (ECG) and movement artifacts, 50 Hz interference, etc. [12]. Further optimal values for recording parameters, such as rate of digitization, precision of digitization, filter settings, etc., are still somewhat unclear. (For more in depth discussion, see [13].) All these factors virtually ensure that the output of G-P applied to EEG data is not an estimate of *Correlation Dimension (D_2)* that is necessarily accurate in an absolute sense. However, the literature indicates that the measure provided by these methods is useful for making relative comparisons among experimental conditions, and/or groups of interest. In this sense, the measure serves as a relative index of complexity of EEG dynamics regardless of the source of the complexity. In recognition of this relative utility Pritchard & Duke [14] calls this measure *Dimensional Complexity* , when applied to EEG data.

Data Length. In order to get a good evaluation of D_2 ,ideally one must consider several hundred thousand points. However, this is not always possible, especially, in the case of the EEG where the dynamics do not remain stationary for very long. If the phenomenon is of short duration and if the dynamics are characterized by a rather low dimension, acceptable results may be found with several thousand points. In other cases , short time series may seriously underestimate the value of D_2. For example when analyzing the alpha waves according to the length of the time series, values as low as D_2=2.6 and as high as D_2=6.6 may be found. Here

also one must use a data length such that a very substantial increase in the time series does not change the value D_2. Of course the time does not exceed the duration of the stationarity of the phenomena. Otherwise, one would be trying to estimate a D_2 value from time series data obtained from different attractors. In this thesis we used EEG data varying in length from 5100 points with a sampling frequency of 256 Hz (20 seconds) down to 1024 points (5 seconds).

Sampling Rate. In general, it is not possible to compensate for a relatively short data length by increasing the sampling rate. Although this does increase the number of data points, it simultaneously decreases the amount of new information provided by the successive reconstruction state vectors. As sampling rate is increased, the vectors tend to become increasingly similar and therefore when viewed as points in the state space, represent increasingly less movement along the reconstructed attractor, and thus decreasing the amount of information about the attractors' geometrical structure. Besides, due to limited precision, and presence of noise, the little information theoretically provided by too close points are not received in practice and these points are measured to be the same in value. Since the computation time increases by N x (N-1), N being the number of points, one should avoid oversampling which does not provide one with much enough information. This brings an upper bound to the *sampling frequency.*

On the other hand, given adequate data length, one should prefer high sampling frequency to prevent aliasing as per Nyquist theorem which states that the sampling frequency must be more than the highest frequency component present in the data, in order to capture the signal truly. This brings the lower bound to the sampling rate. A tradeoff between the two factors is reached around sampling frequencies of 200 Hz for EEG analysis, since most EEG analysis involves frequency ranges within 100 Hz. A sampling frequency of 256 Hz is used in the computations in this thesis.

Digital Filtering. Lo and Principe [15] have found that digital low-pass filtering EEG segments at 80, 50, and 20 Hz decreases dimension estimates,

presumably by filtering out high-frequency white or near-white noise that of very high dimension. If any filtering is done, it should be *acausal*, where the filtered value of a given data point is a function of the entire data set rather than of preceding points only [16] (for a review of digital filtering see [17]).

A/D precision. As discussed by Theiler [18], any degree digitization results in a finite resolution level, which in turn results in the distances among all reconstructed points in state space being multiples of the digitization level (call the level r_L). The result is the introduction of a stair step into the log[C(r)] versus log(r) relation that is most prominent for small r as seen in [18]. Theiler suggests that log[C(r)] be scaled versus log[r + (r_L/2)] rather than versus log(r) to correct for a finite level of A-to-D precision. Theiler indicates that this eliminates the stair step in the log[C(r)]versus log(r) curve , producing a smoother curve. I am aware of no EEG studies that have employed this correction. I do not note that the Takens-Ellner method systematically estimates D_2 for a region of larger r than G-P and thus may be less susceptible to the stair step problem. Pritchar and Duke also note that significant statistical effects related to locus and experimental condition can be obtained with only 8 bits of precision yielding a resolution level of around 1 μ V [14]. In this project EEG samples have a 12 bit A/D precision, which is good enough for all practical purposes. And in practice no precision above the level of 16 bits is met.

Determination of linear scaling region. Although computation of the correlation integral is straightforward once the delay time τ and the embedding dimension m are fixed, determination of the linear scaling region in log[C(r)] versus log(r) is a sophisticated problem in pattern recognition. One method is simply to visually identify the linear scaling region by viewing the log[C(r)] versus log(r) plot. This method is simply not feasible for large data sets. However, besides its subjectivity, this method is unavoidably applied together with various algorithms, and still human brain together with the human eye is the best pattern recognizer, and interpreter. The most widely applied class of procedures for finding linear scaling region is based on linear regression. The one that is used in this project is based on general procedures recommended by Albano et al [19]. In the procedure applied here,

a linear regression is fit to successive, overlapping local windows of 4 consecutive points in the log[C(r)] versus log(r) plot, and a corresponding set of slopes m(i) are computed. Next an expanded window in log(r) for which no value of the slope of the plot at r, m(r), deviates more than 10% from the average value of m(i) is determined. A linear regression is then fit to the expanded window, and the Q measure of goodness of fit computed. The length of the expanded window is then systematically varied until the value of Q is between 0.3 and 0.2, indicating a linear scaling region that is as wide as possible together with an acceptable quality of fit.

4. IMPLEMENTATION - DYNAMICAL SYSTEMS ANALYZER (DSA)

In this project Correlation Dimension of EEG is computed using rigorous algorithms to speed up the computation by finding shortcuts to calculations, arranging them as per the computer architecture, and to increase precision with considerations stated above.

The software, named ***Dynamical Systems Analyzer (DSA)*** is developed in Delphi 2.0 environment for various reasons:

- *A user friendly and visual Graphical User Interface (GUI)*: The project was developed in Turbo Pascal 7.0 in the beginning. However, in the course of time it was seen that many computations and procedures that are necessary to increase the speed and accuracy of the D_2 computation are required. Secondly, D_2 computation is a signal analysis method, and it requires graphical interpretation and display of signals. Other procedures regarding file handling, file format conversions, printing etc. necessitated a user friendly Graphical User Interface, which, in turn necessitated a contemporary fourth generation (visual) programming environment, which speeds up visual, user friendly code generation.
- *32 bit structure:* Today's computers are 32 bits. Going from 16 bits to 32 bits computers became much faster and more powerful. Software environments were ahead of this transition in hardware environments. Contemporary Operating Systems such as Windows 95, Windows NT, OS/2 Warp 4.0 are all 32 bit environments. In order to let DSA make use of advantages of 32 bit environments fully, a 32 bit programming environment was necessary. Delphi 2.0 is among the best alternatives for 32 bit structure, which makes the software developed be a natural part of the most popular contemporary operating systems such as Windows 95.
- Compared to other fourth generation visual programming environments (such as Visual Basic), Delphi 2.0, being a descendant of Pascal, is a language that is

more appropriate for low level programming, which is required for speeding up the rigorous computations mentioned above.

4.1. DSA GRAPHICAL USER INTERFACE IN BRIEF

The Graphical User Interface (GUI) of DSA consists of a main menu bar at the top, a graph section used for graphical representation of the active signal, a status panel, used for displaying helps, hints and small messages depending on the position of the mouse pointer, and a signal buttons panel where buttons representing signals open are displayed. Users can activate any open signal by clicking the corresponding button. There is no limit to the number and length of signals analyzed by DSA as long as the hardware and Operating System permit. The menus available at main menu bar are as follows.

1.File: Menu for file handling procedures.

1.a. Open: Opens a signal file. The file can be binary, ASCII, any other text file, or a DSA file. DSA files have a special format which provides DSA with information to be used in their full manipulation.

1.b. Convert: Provides procedures for file format conversions. (Binary to DSA, DSA to Binary, Binary to ASCII etc.).

1.c. Save: Saves the active signal with the current path and file name.

1.d. Save As: Saves the active signal with a different path and file name.

1.e. Print to File: Saves the graphical representation of the active signal in MS bitmap, or MS metafile format so that it can be pasted to MS Word, Excel etc. documents.

1.f. Print: Prints the active signal to the active printer.

1.g. Exit: Exits DSA and disposes all the associated memory.

2. Basic: Menu for basic signal manipulation. Some procedures such as cutting, clipping, appending etc. are not implemented, yet.

2.a. Signal Information: Provides basic information such as median, medium, standard deviation, etc. about the selected signal.

2.b. Peak to Peak Distance: Plots the peak to peak distance versus x-axis.

2.c. Autocorrelaiton: Plots the Autocorrelation function of the selected signal according to the delay value entered.

3. Spectral: This menu is not implemented, yet. It includes procedures for spectral analysis such as Fourier transform, wavelet transform, and filtering.

4. Nonlinear: This is the most important menu of DSA. It is what makes DSA such a special signal analyzer. As the name suggests it includes popular procedures of the nonlinear science.

4.a. Mutual Information: Prompts for a *time lag* value, and computes and plots the mutual information graph with respect to the given *time lag*.

4.b. Correlation Dimension: Prompts for correlation dimension computation parameters such as time lag, minimum embedding dimension, maximum embedding dimension, upper radius, lower radius (upper and lower limits of the radii of the grids), and computes the correlation dimension of the selected signal, and finally plots log[C(r)] versus log(r) and D_2 versus log(r) graphs.

5. Plot: Includes menus for drawing 2D and 3D delay plots of the selected signal, and 2D and 3D state space plots of 2 or 3 distinct signals forming vectors of corresponding dimensions.

6. Help: Includes a help browser for DSA (not implemented yet) and an About menu.

Readers are referred to the DSA manual for further information about DSA.

5. APPLICATION AND RESULTS

The software developed for correlation dimension computation is applied to 13 distinct signals during its development under this thesis. 10 of these signals are EEG time series data obtained from occipital electrodes of various subjects under various conditions. The sampling frequency is 256 Hz and the A/D conversion is 12 bits for all the EEG data. The other three signals include a sinusoidal, a Henon map, and a sample of white noise signal, to justify that correlation dimension estimates by DSA are reliable if correct calculation parameters are set and above mentioned practical considerations are carefully taken into account. The application results are as follows.

5.1. SINUS

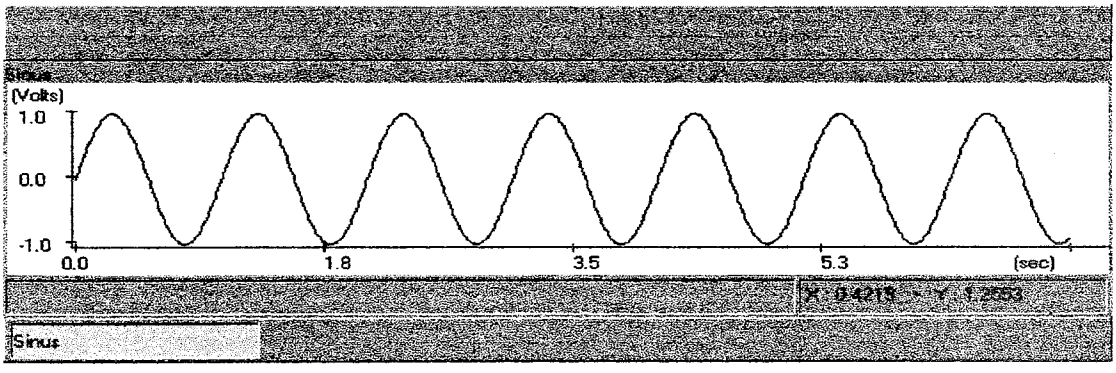

Figure 6.1. Sin(x). Number of points = 700.

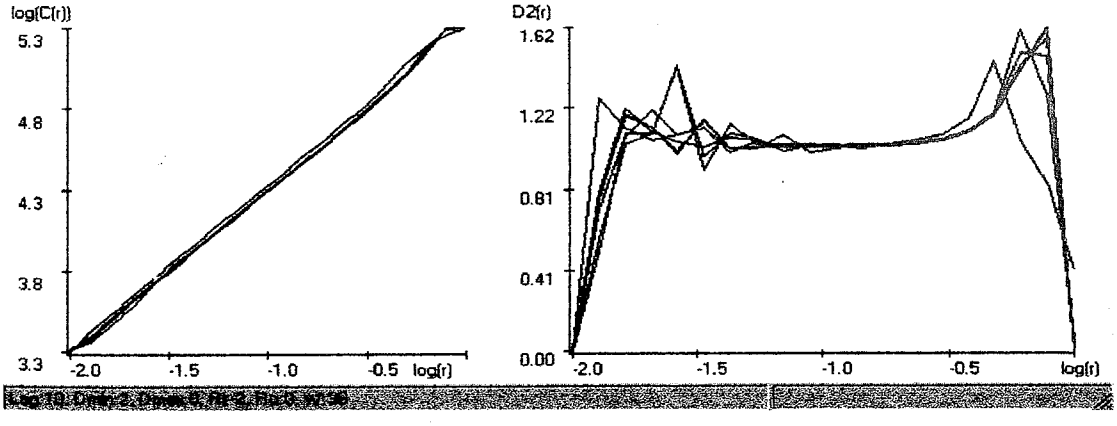

Figure 6.2. D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of Sinus(x). D_2=1.03+/-0.06

5.2. HENON MAP

As an example of a strange attractor, whose dimension is known, the attractor obtained for the 2 dimensional Henon map, is studied. The equation of the Henon map is as follows.

$$x_{n+1} = A - x_n^2 + By_n^2 ,$$

$$y_{n+1} = x_n , \qquad (14)$$

A is taken as 1.4 and B is taken as 0.3. Numerical computations show that the fractal dimension D_0 of the attractor is approximately 1.26.

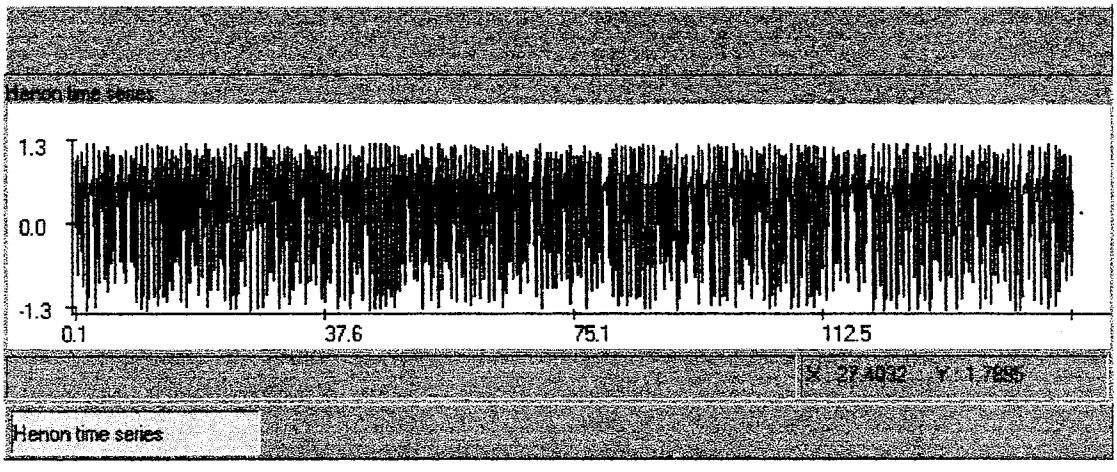

Figure 6.3. Henon map. Number of points = 1500.

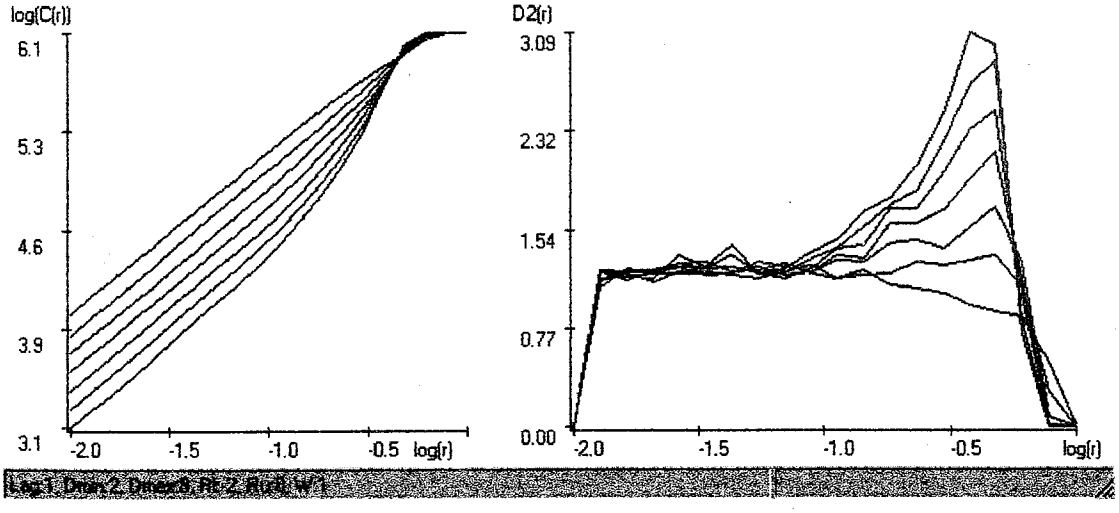

Figure 6.4. D_2 computation plots (log[C(r)] - log(r), and D_2 - log(r)) of Henon map. D_2=1.24+/-0.02

5.3. EEG TIME SERIES

The following shows the results of applications to EEG signals from the simplest to the most complex. All the signals are obtained from occipital channel, sampled at a rate of 256 Hz, and A/D converted with a precision of 12 bits, and low pass filtered at 30Hz.

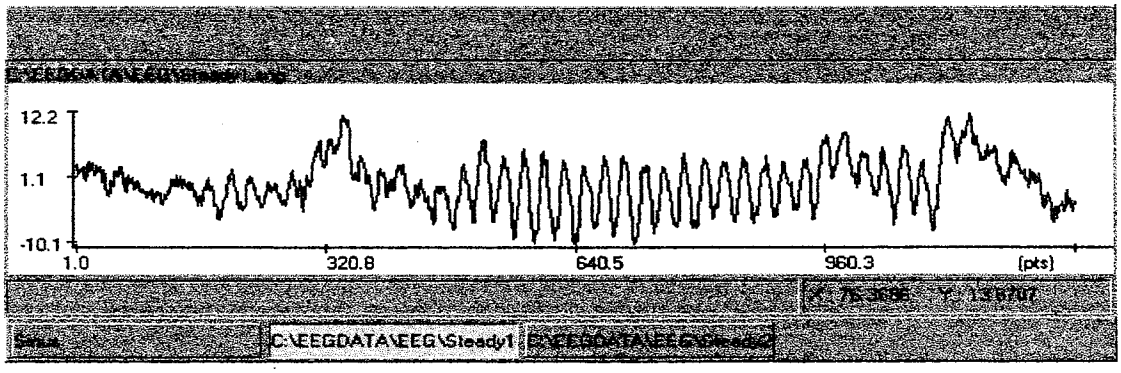

Figure 6.5. EEG signal from a flash driven subject (30 flashes). # points = 1280

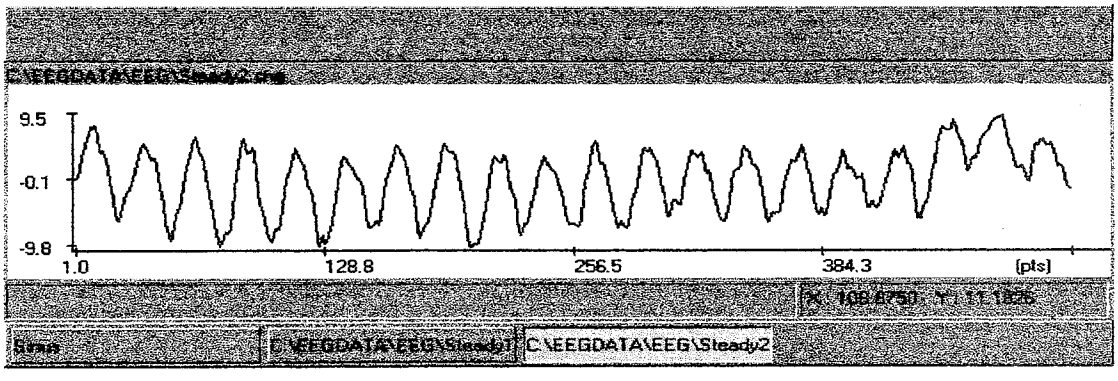

Figure 6.6. Steady regime portion of the above signal. #points=512.

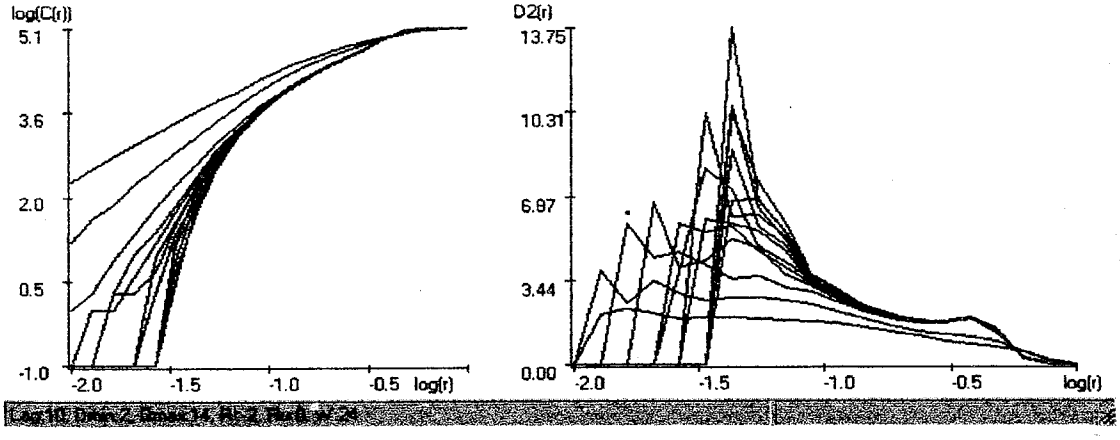

Figure 6.7. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.6.D_2=1.80+/-0.02

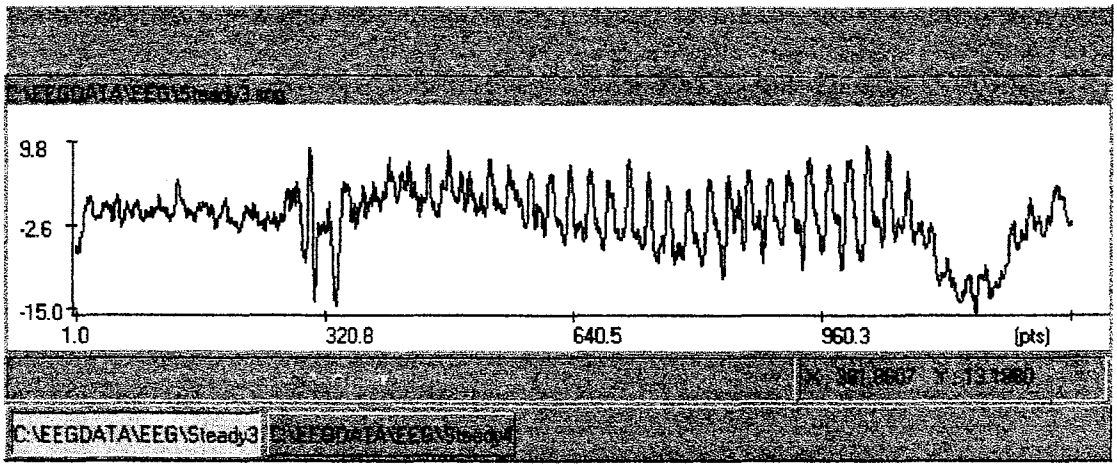

Figure 6.8. EEG signal from a flash driven subject (30 flashes). # points = 1280

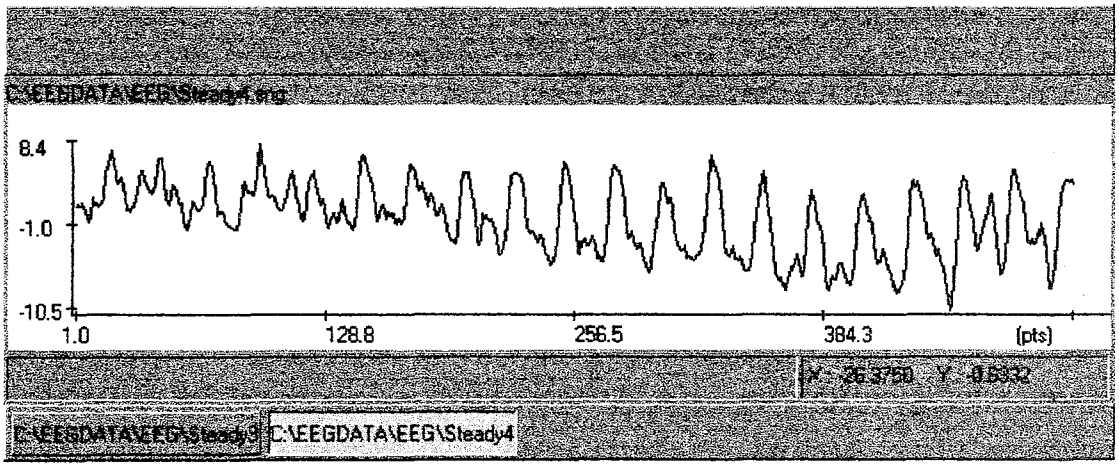

Figure 6.9. Steady regime portion of the above signal. #points=512.

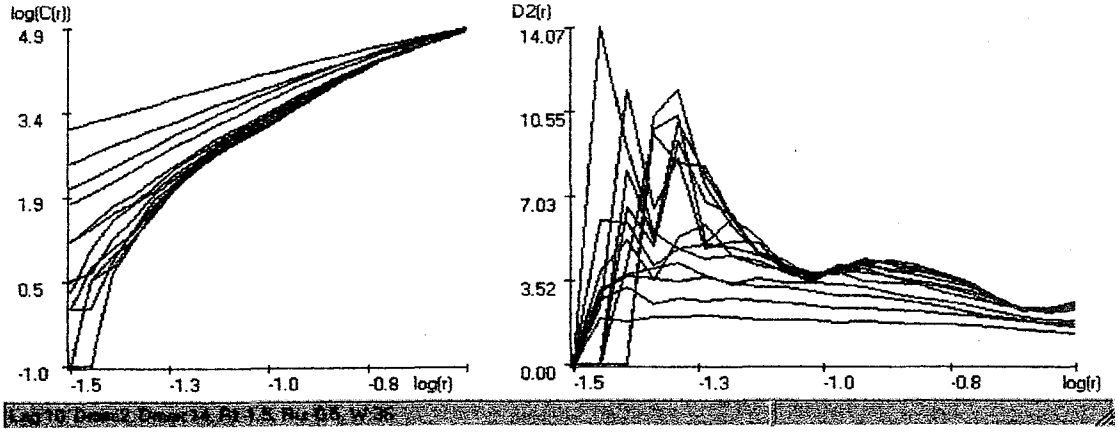

Figure 6.10. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.9. D_2=3.77+/-0.09

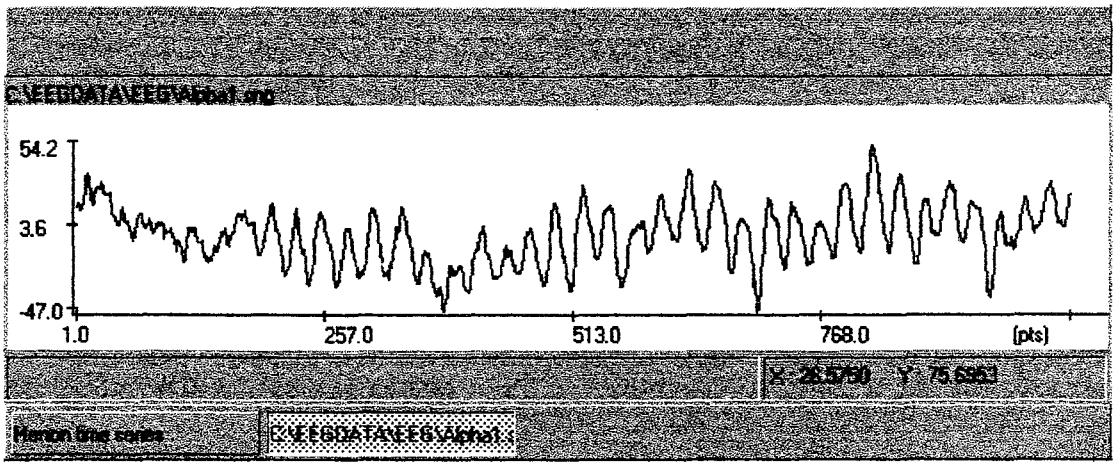

Figure 6.11. An alpha dominant EEG signal. # points = 1024.

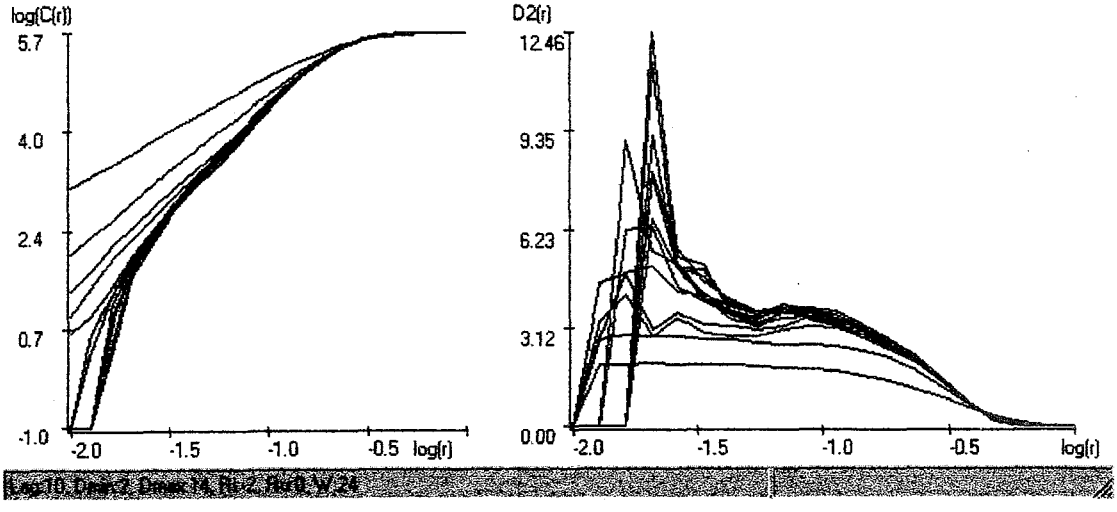

Figure 6.12. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.11.D_2=3.60+/-0.09

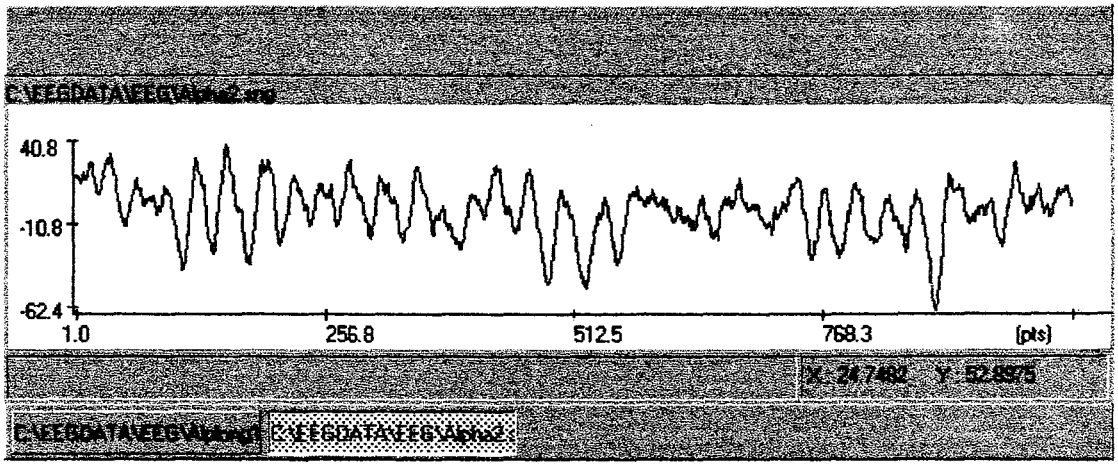

Figure 6.13. An alpha dominant EEG signal. # points = 1024.

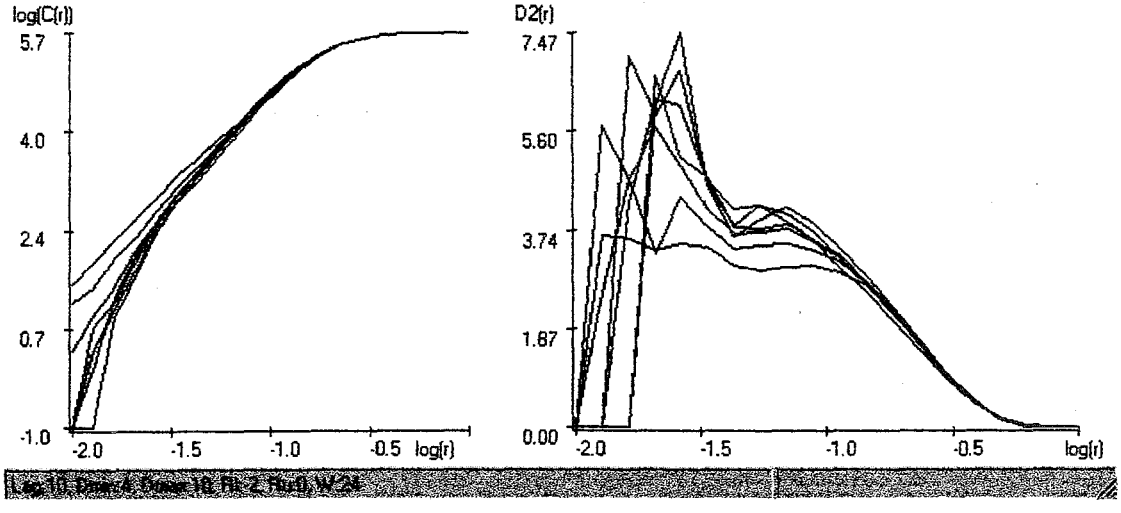

Figure 6.14. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.13. D_2=3.89+/-0.13

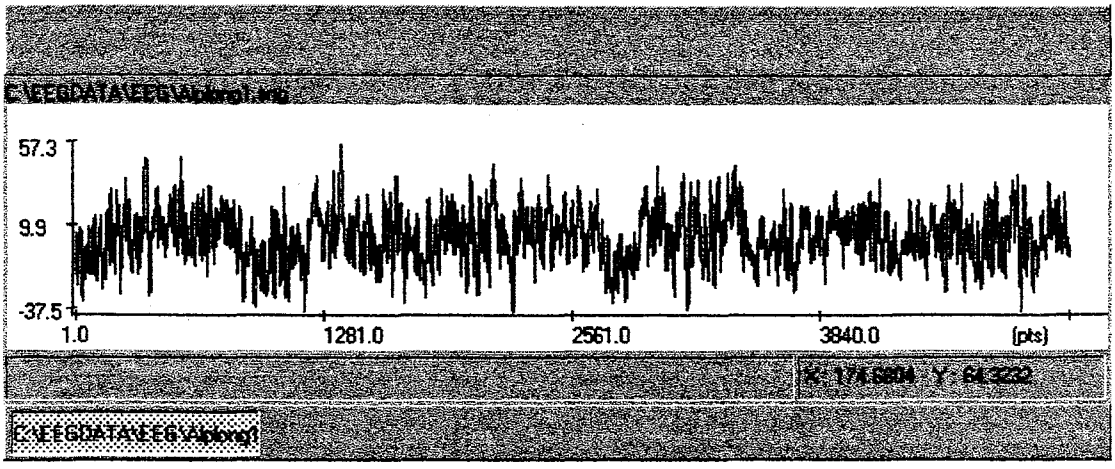

Figure 6.15. An alpha dominant EEG signal. #points = 5120.

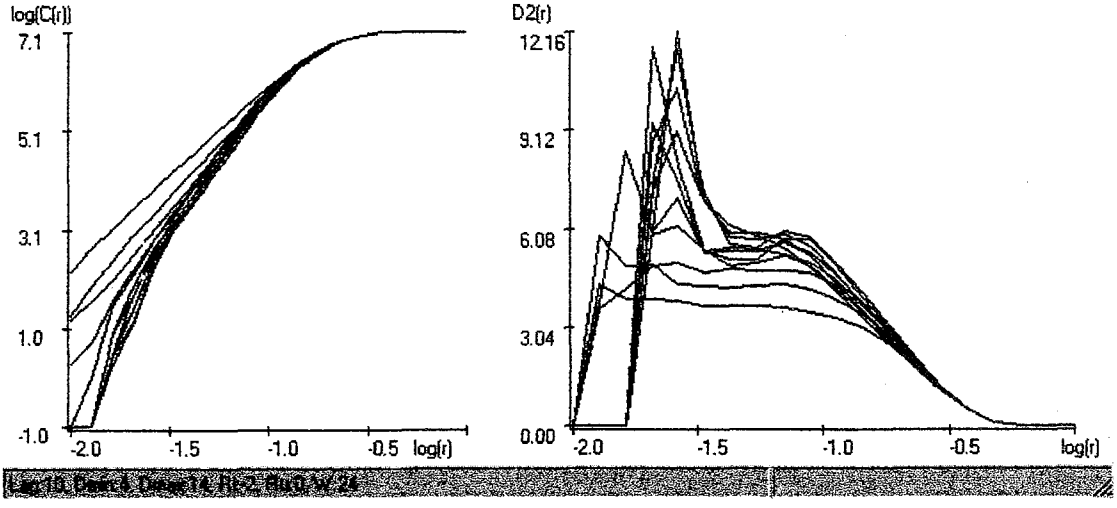

Figure 6.16. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.15. D_2=5.94+/-0.13

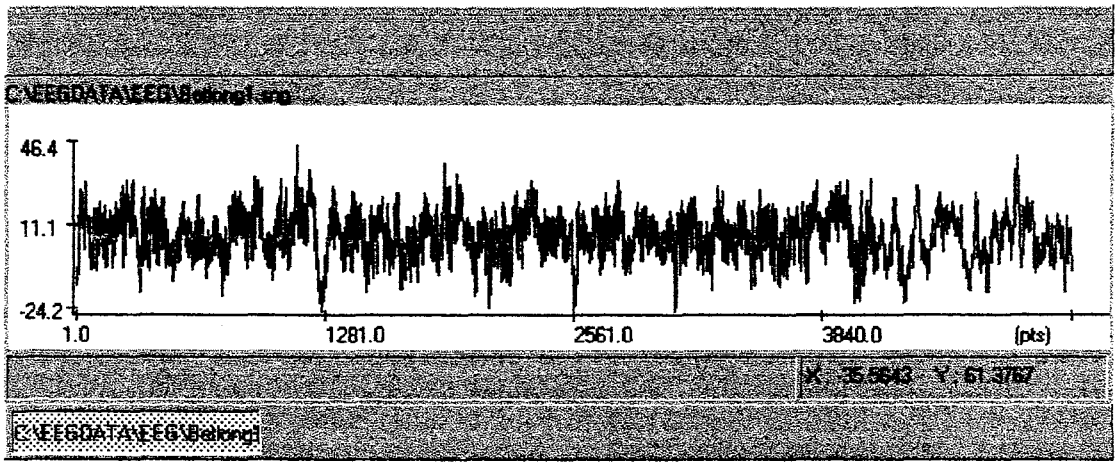

Figure 6. 17. A beta dominant EEG signal. # points = 5120.

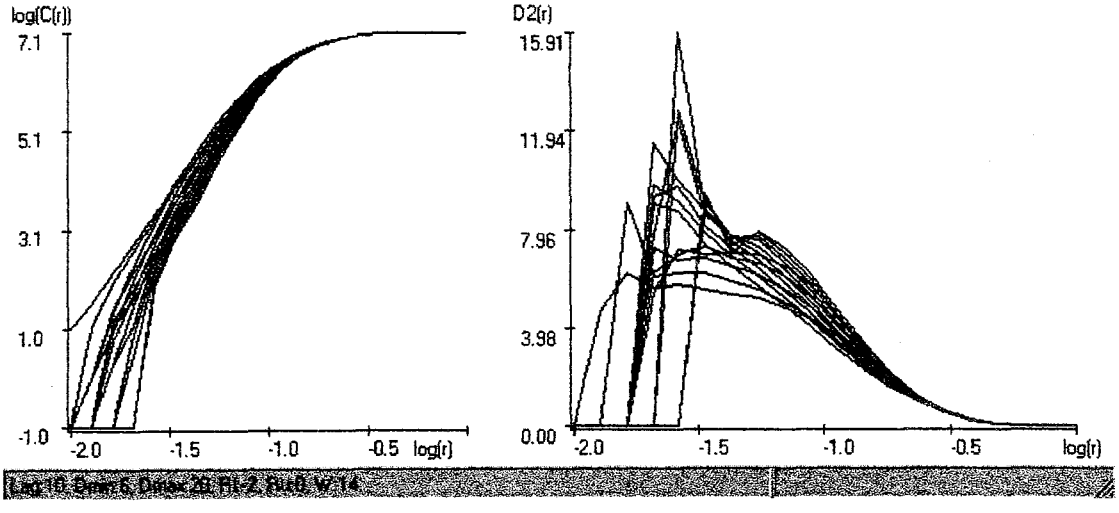

Figure 6.18. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.17. D_2=7.88+/-0.08

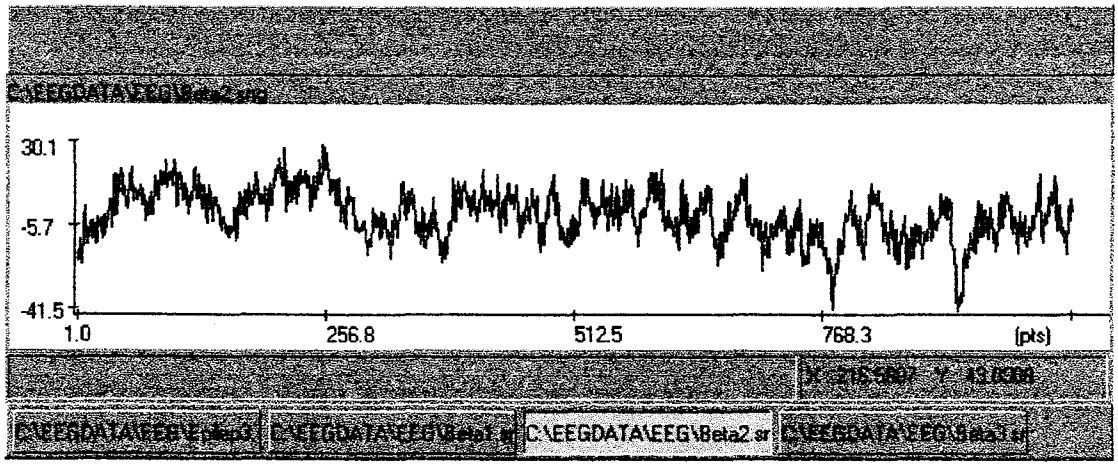

Figure 6.19. A Beta dominant EEG signal. # points = 1024

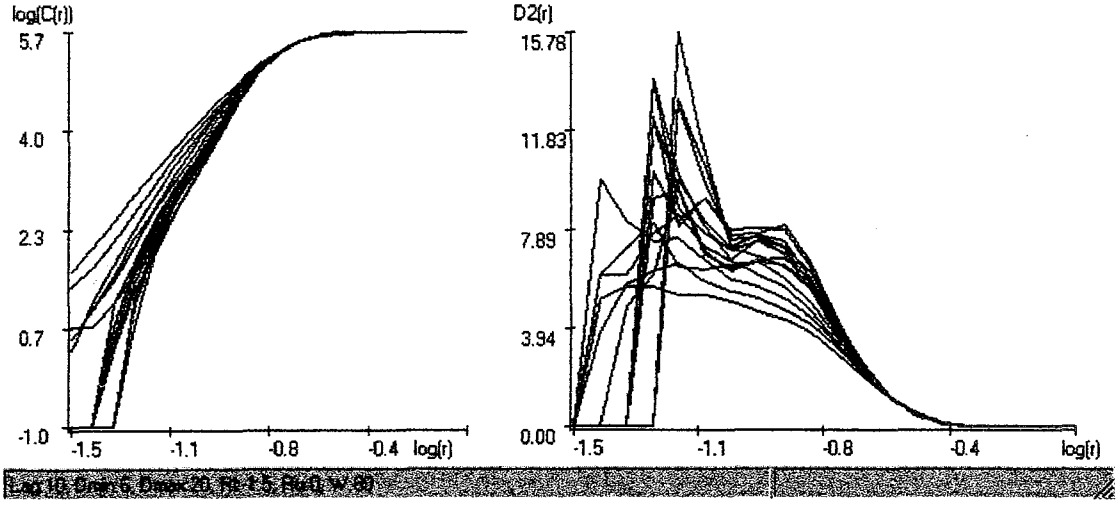

Figure 6.20. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.19. D_2=7.78+/-0.12

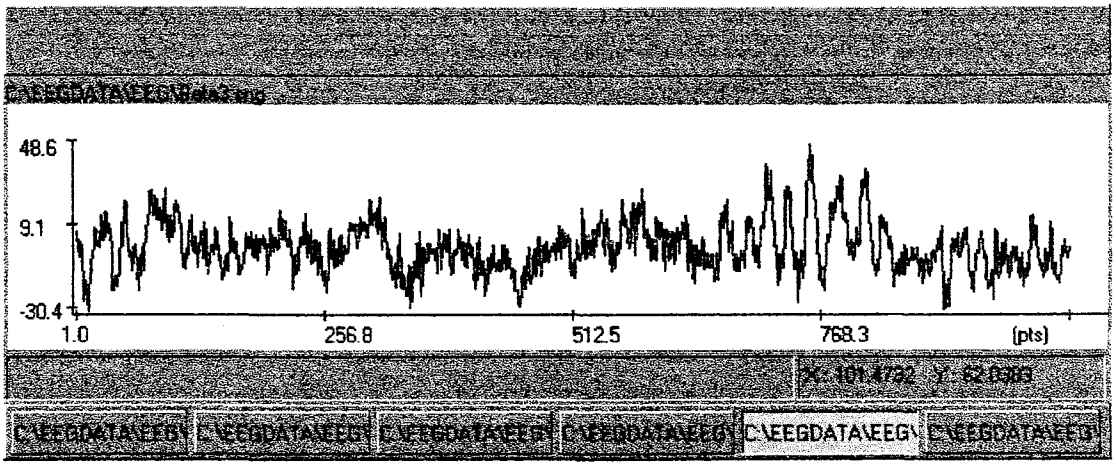

Figure 6.21. A Beta dominant signal. # points = 1024

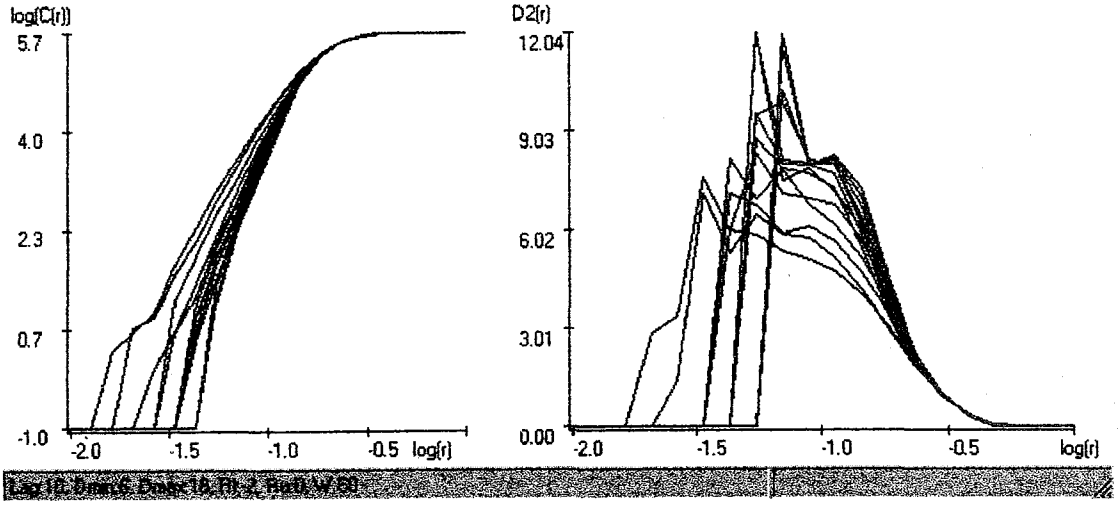

Figure 6.22. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.21. D_2=8.15+/-0.24

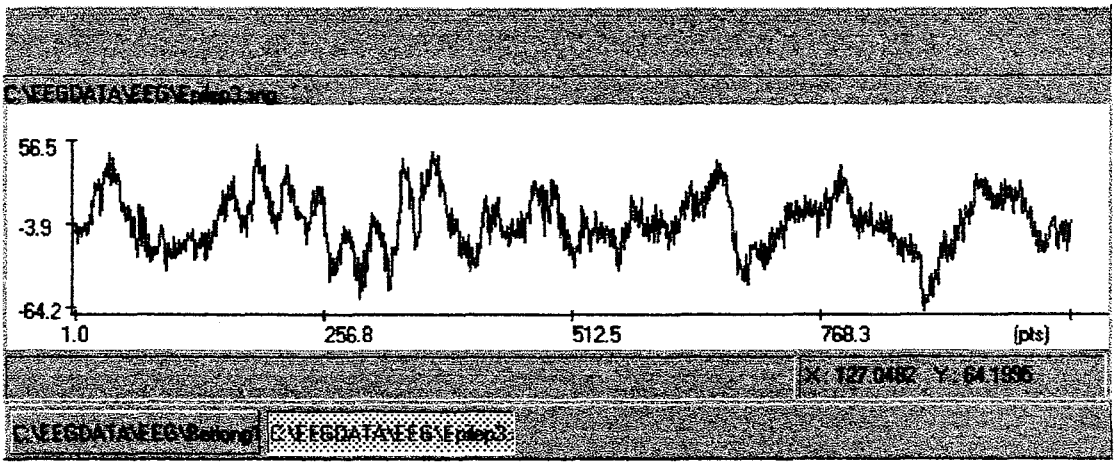

Figure 6.23. A beta dominant EEG signal from a subject suspected of minor epilepsy. # points = 1024.

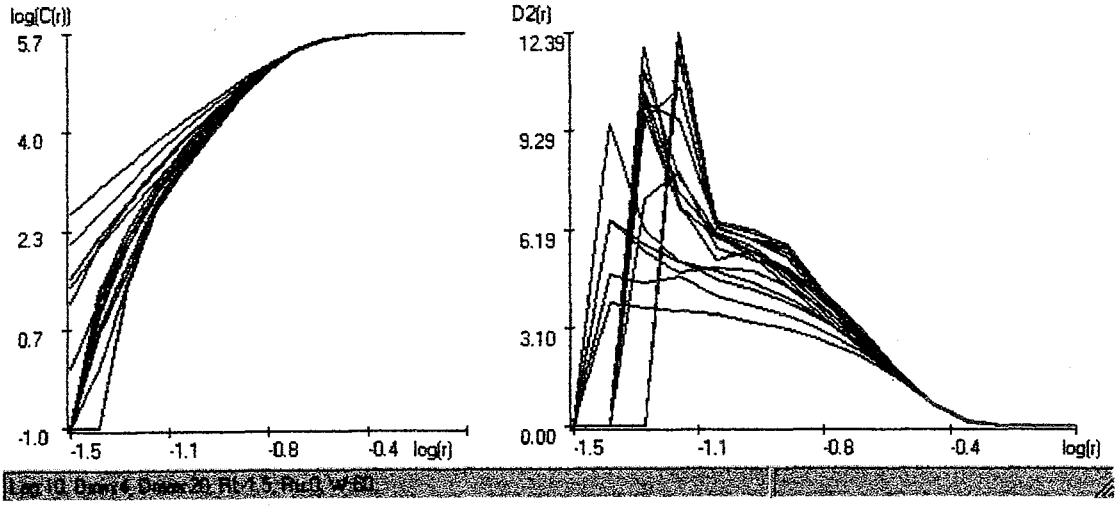

Figure 6.24. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.23. D_2=6.01+/-0.22

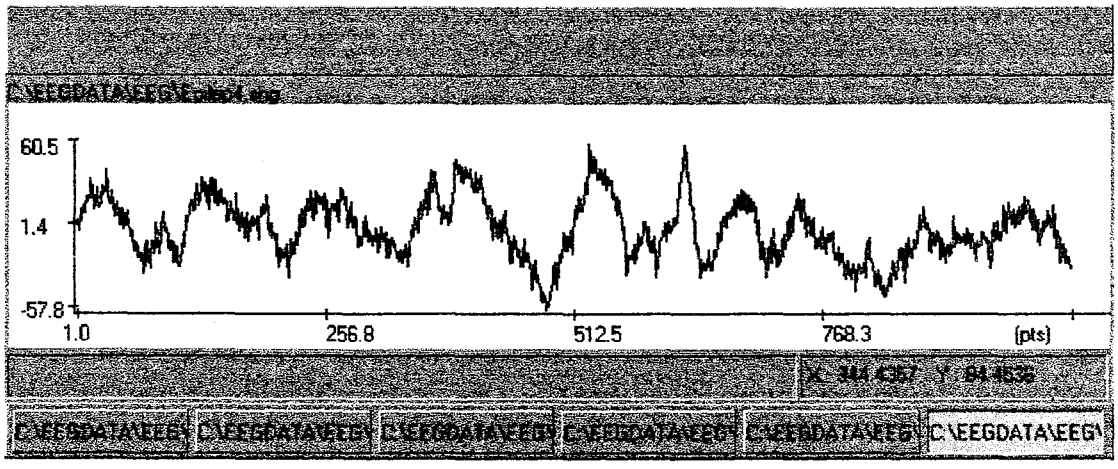

Figure 6.25. A beta dominant EEG signal from a subject of minor epilepsy suspect. # points = 1024

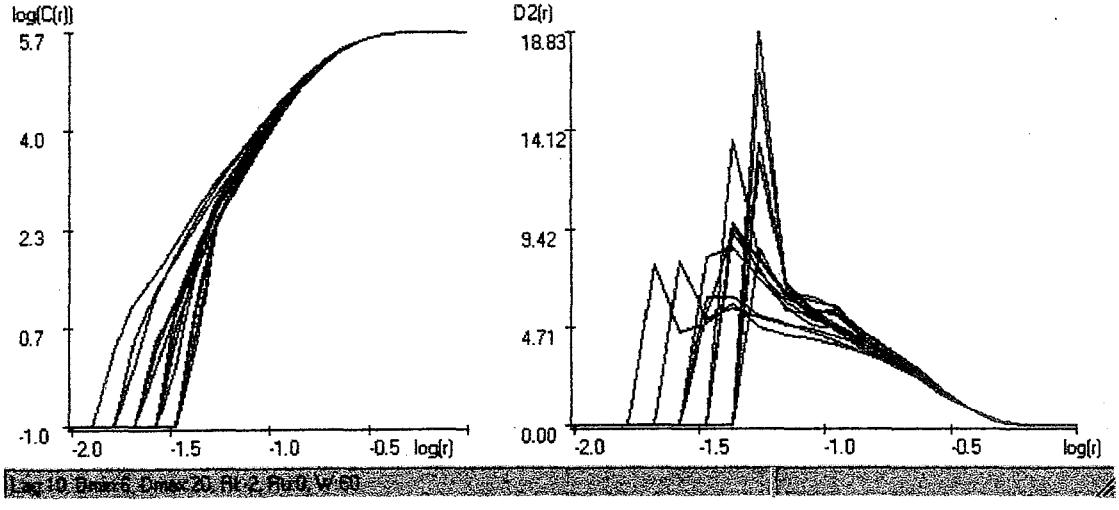

Figure 6.26. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.25. D_2=5.51+/-0.52

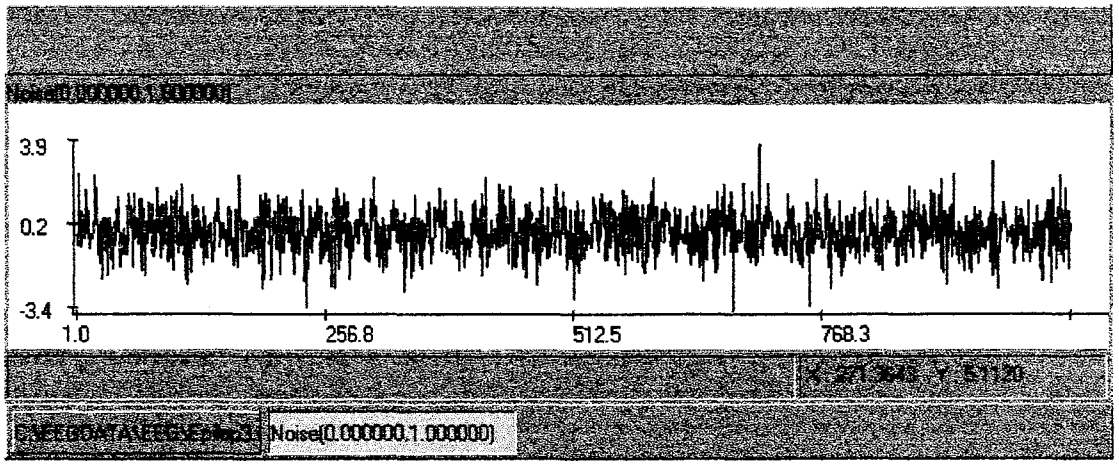

Figure 6.27. White noise. #points = 1024.

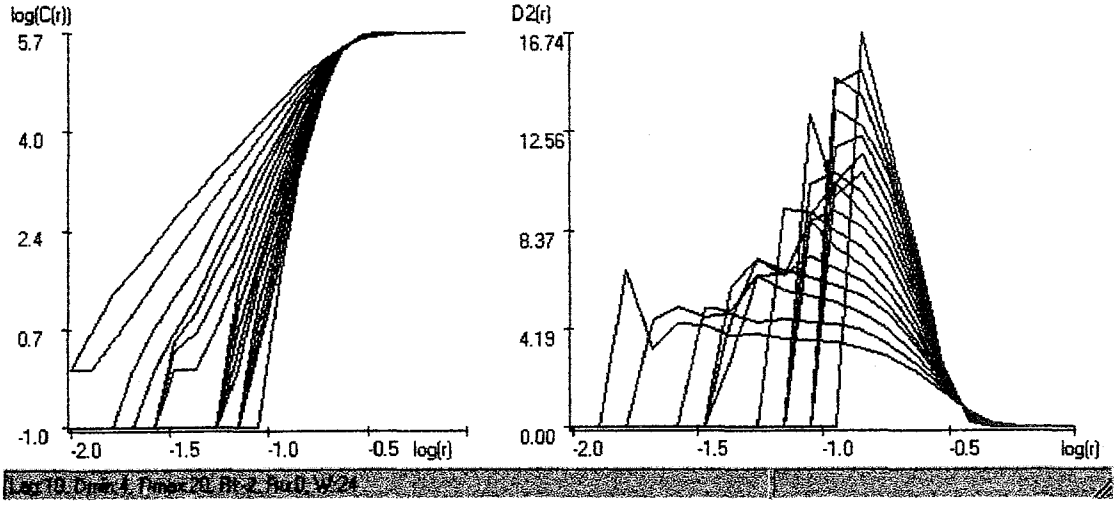

Figure 6.28. D_2 computation plots (log[C(r)] - log(r), and D2 - log(r)) of signal in Figure 6.27. D_2:No saturation

6. CONCLUSION

6.1. ANALYSIS AND COMPARISON OF NUMERICAL RESULTS

Signal	Description	#points	D_2
Sinus	1 dimensional sinusoidal	700	1.03+/-0.06
Henon Map	Henon map	1500	1.24+/-0.02
Steady1	EEG signal from a flash driven subject	512	1.80+/-0.02
Steady2	EEG signal from a flash driven subject	512	3.77+/-0.09
Alpha1	Alpha dominant EEG signal	1024	3.60+/-0.09
Alpha2	Alpha dominant EEG signal	1024	3.89+/-0.13
Alpha_long	Alpha dominant EEG signal	5120	5.94+/-0.13
Beta1	Beta dominant EEG signal	1024	7.88+/-0.08
Beta2	Beta dominant EEG signal	1024	7.78+/-0.12
Beta_long	Beta dominant EEG signal	5120	8.15+/-0.24
Epilepsy1	Beta dominant EEG signal from a subject with minor epilepsy	1024	6.01+/-0.22
Epilepsy2	Beta dominant EEG signal from a subject with minor epilepsy	1024	5.51+/-0.52
White Noise	White noise	1024	no saturation

Table 6.1. Numerical results.

The application results are shown in the above table. The first thing to notice is that DSA computes D_2 of the sinusoidal and the Henon map correctly with an acceptable tolerance, i.e. the true values are within the tolerance ranges of the computed values. It also finds that white noise fills the whole state space, and thus does not saturate with increasing embedding dimension. These consistent results suggest that DSA gives reliable results if correct parameters are set.

All the above EEG signals are taken from the occipital channel, sampled at a rate of 256 Hz, A/D converted with a precision of 12 bits, and low pass filtered at 30 Hz. The signals are divided into 4 categories according to the state of the subjects during recordings.

The first category signals are recorded from flash driven subjects (30 flashes). In Figure 6.5 and Figure 6.8 one can see the time when brain responds to the first flash. During the sequence of flashes the brain gets adapted to the stimuli and gives a steady state response. It can be stated that there is a single attractor during this steady state (stationarity), whose dimension is expected to be very low due to its periodic like nature. Please note that the lowest dimensions found are in this category.

The second category signals are recorded when the subjects relax with eyes closed and no task given. This is when the low frequency alpha band signals (8-13 Hz.) become dominant. The estimated D_2 of the first two signals which are of short duration (4 sec) are very close to each other (3.60+/-0.09, 3.89+/-0.13), while that of the third one, which is of much longer duration (20 sec) is considerably higher (5.94+/-0.13). The first question that arises is whether this is due to the big difference in their data length. It might be argued that the brain might have moved to another attractor (loss of stationarity) in the longer signal making the whole time series decorrelated and increasing the dimension. More analysis must be employed in order to answer this question. It is a usual practice to apply EEG dimension computation to data lengths of up to 30 seconds. However, one can never safely guarantee stationarity without applying other tools such as wavelet transform, Lyapunov exponents, etc. Secondly, there is a great variety of alpha band characteristics of individuals. Some people even do not have an alpha band component. So it is not so surprising to find differing dimension estimates of alpha dominant EEG signals from different subjects. However, alpha dominating signals are still expected to have lower dimensions compared to signals dominated by higher frequency bands such as beta waves.

The third category includes beta dominant EEG signals. The beta band is slightly above the alpha band (13-22Hz.). These are expected to have dimensions higher than those of alpha waves, which is observed in the applications in this thesis. The correlation dimensions found for the three beta samples are very close to each other (7.88+/-0.08, 7.78+/-0.12, and 8.15+/-0.24).

The fourth category comprises two beta samples from subjects of minor epilepsy. These signals are found to have dimensions lower than those of the beta samples from healthy subjects, and higher than those of the alpha samples (6.01+/-0.22, 5.51+/-0.52). Which are again consistent with expectations. Previous practice on D_2 computation of epileptic subjects show that there is a significant decrease in the dimension of the attractor of the brain. Interested readers are referred to [20] and [21].

In literature different values are found for similar signals, while on the other hand there is a strong relationship between relative correlation dimensions of various signals and their other characteristics. The Grassberger-Procaccia theorem requires unlimited, noise-free data, which is never met in neuroscience. In practice, EEG segments are of short duration to decrease artifacts and increase the probability of system stationarity. However, even the cleanest EEG recordings may contain electrooculographic (EOG), electromyographic (EMG), and electrocardiographic (ECG), and movement artifacts, etc. Furthermore there is no well established means of being sure about the optimal values of the D_2 computation parameters, such as time delay, sampling rate, digitization precision, filter settings, etc[16]. All these factors ensure that the D_2 computation algorithms do not yield an absolute value of correlation dimension. However, literature indicates that the measure provided by these methods is useful for making relative comparisons among experimental conditions and/or groups of subjects. In this sense the measure serves as a relative index of the complexity of EEG dynamics regardless of the source of complexity. Many scientist (Albano et. al., [22]; Dvorak & Siska, [23]; Layne et.al. [24]; Mayer-Kress &Layne, [25]; Nan &Jinghua, [26]; Rapp et. al. [27]) who have dealt with this phenomenon have noticed this fact. In recognition of this reality Pritchard & Duke,

1992, preferred to call this measure, when applied to EEG data, *dimensional complexity (DCx)* rather than "the correlation dimension (D_2)".

6.2. FUTURE DIRECTIONS

6.2.1. Further Advancements on DSA

The time delay, τ, used in D_2 computation is determined by studying the peak to peak distances, linear autocorrelation function, or mutual information. This approach has yielded satisfactory results in the thirteen D_2 computations made in this thesis. However, this does not guarantee that they will be sufficient in future applications when thousands of signals of great variety are subjected. There are numerous methods for determining the optimum value of τ, each of which has its own advantages and drawbacks, and there is, unfortunately, no well established mathematical basis for finding the optimum τ. This is an open research area in mathematics. Other methods for choosing τ can be studied and incorporated to DSA.

In this thesis embedding dimension was found by simply taking a guess and increasing the dimension until D_2 saturated with increasing embedding dimension, which is the common practice. This is the safest method for finding optimum embedding dimensions. It is, however, practically not feasible in case of large data sets. Various methods for estimating optimum embedding dimensions, such as *singular value decomposition*, and *method of false nearest neighbors* shall be implemented for future applications involving large data sets.

Finally, the *linearizer* (the code that finds the linear scaling region) cannot find the linear region, in case improper values of *upper grid radius* (r_{up}), *lower grid radius* (r_{low}), and *number of grids* ($\#_{grids}$) are entered (this has happened very rarely in this thesis). It is the user's task to find the proper values for these parameters, by trial and error, and by visualization of the corresponding plots. A smart linearizer can be developed to do this automatically. It should be noted, however, that this is not an easy task, and I have not met any such application in literature.

6.2. The Future

Application of nonlinear dynamics to the brain -the organic computational system- provides us with new paradigms and valuable information on its dynamics, which, in turn, feeds the progress in models of artificial computational systems (neural networks). This makes such studies pivotal for the future of Artificial Intelligence (AI), which seems to be one of the most important targets of the human brain itself.

I close with Başar's guiding words [28]:

> Although the new approaches have opened important conceptual new windows ...., we must keep our eyes open not to damage this important new branch of neuroscience. Chaotic dynamics should not be developed to be a target of critics based on papers that contain no experimental information, unevaluated data, or data describing the behavior of only one subject ... in neuroscience quiet revolutions are needed. I think, despite a number of difficulties, the new area treating chaos in the brain function is one of the necessary quiet revolutions in neuroscience. (p.26).

APPENDIX

SOURCE CODES OF DSA:

All the software given in this appendix is developed entirely by Ersin Taşkın in Delphi 2.0. Necessary acknowledgements are made where necessary throughout the thesis.

Outline of the Software:

Name	Description
DSA.dpr	Delphi project file. This is the file that manages the whole software, i.e. links all the units and runs the application
Main.pas	Delphi unit file. Manages the main form of DSA
UnitPlot.pas	Delphi unit file. Responsible for external window plots.
UnitList.pas	Delphi unit file. Called each time the user wants to apply a procedure to a signal. It displays a list of the available signals and prompts for a selection
About.pas	Delphi unit file. Displays the about window
EditSignal.pas	Delphi unit file. Edits the selected signal
Unit2.pas	Delphi unit file. Called when "File | Open" menu is clicked. Displays information about the file to be opened, asks the size of the portion to be opened, and prompts for confirmation
Unit3.pas	Delphi unit file. Displays the values of points of the selected signal
LagPlot2D.pas	Delphi unit file. Draws 2D delay plot of the selected signal to an external window
DelayPlot2D.pas	Delphi unit file. Prompts for τ value to be used by LagPlot2D.pas
Correlation.pas	Delphi unit file. Computes the correlation dimension of the selected signal, and draws the relevant plots
Corparams.pas	Delphi unit file. Prompts for parameter values to be used by Correlation.pas

DSA.dpr:

```
program DSA;

uses
  Forms,
  Main in 'Main.pas' {DSAForm1},
  DelayPlot2D in 'DelayPlot2D.pas' {Form2DDelay},
  UnitPlot in 'UnitPlot.pas' {Form4},
  UnitList in 'UnitList.pas' {FormList},
  About in 'About.pas' {AboutBox},
  EditSignal in 'EditSignal.pas' {FormEdit},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3},
  LagPlot2D in 'LagPlot2D.pas' {FormLag2D},
  Correlation in 'Correlation.pas' {FormCorDim},
  Corparams in 'Corparams.pas' {FormCorParams};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TDSAForm1, DSAForm1);
  Application.CreateForm(TFormList, FormList);
  Application.Run;
end.
```

Main.pas :

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ExtCtrls, StdCtrls, ComCtrls, Buttons, UnitPlot, DSALib;

const
  NewLine: string[2] = #13#10;
  SamplingRate = 256;
type
  TDSAForm1 = class(TForm4)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Open1: TMenuItem;
    Convert: TMenuItem;
    SaveAs1: TMenuItem;
    PrinttoFile1: TMenuItem;
    Print1: TMenuItem;
    Exit1: TMenuItem;
    PopupMenu1: TPopupMenu;
    AsciitoDSA: TMenuItem;
    DSAtoBin1: TMenuItem;
    BitMap1: TMenuItem;
    EPS1: TMenuItem;
    OpenDialog1: TOpenDialog;
    Basic1: TMenuItem;
```

```
SignalInfromation1: TMenuItem;
Spectral1: TMenuItem;
FastFourierTransform1: TMenuItem;
WaveletTransform1: TMenuItem;
Surrogate1: TMenuItem;
Nonlinear1: TMenuItem;
Plot1: TMenuItem;
Help1: TMenuItem;
Help2: TMenuItem;
About1: TMenuItem;
MutualInformation1: TMenuItem;
CorrelationIntegral1: TMenuItem;
CorrelationDimension1: TMenuItem;
PointwiseDimension1: TMenuItem;
LyapunovExponents1: TMenuItem;
SingularValueDecomposition1: TMenuItem;
NearestNeighborhood1: TMenuItem;
PeaktoPeakDistance1: TMenuItem;
Autocorrelation1: TMenuItem;
Crosscorrelation1: TMenuItem;
N2DPlot1: TMenuItem;
N3DPlot1: TMenuItem;
Display1: TMenuItem;
Window1: TMenuItem;
Cascade1: TMenuItem;
Tile1: TMenuItem;
Next1: TMenuItem;
Previous1: TMenuItem;
Close1: TMenuItem;
CloseAll1: TMenuItem;
PanelStatus1: TPanel;
PowerSpectrum1: TMenuItem;
AmplitudeSpectrum1: TMenuItem;
PhaseSpectrum1: TMenuItem;
DefineWavelet1: TMenuItem;
ShowFunction1: TMenuItem;
Decompose1: TMenuItem;
Reconstruct1: TMenuItem;
Approximation1: TMenuItem;
Detail1: TMenuItem;
Shrink1: TMenuItem;
Soft1: TMenuItem;
Hard1: TMenuItem;
Sure1: TMenuItem;
Filter1: TMenuItem;
Highpass1: TMenuItem;
Lowwpass1: TMenuItem;
Bandpass1: TMenuItem;
N2DDelay1: TMenuItem;
N3DDelay1: TMenuItem;
Mathematical1: TMenuItem;
PanelStatus2: TPanel;
PanelTop: TPanel;
PanelBottom: TPanel;
Label1: TLabel;
LabelX: TLabel;
Label3: TLabel;
LabelY: TLabel;
Show1: TMenuItem;
```

```
    ExternalWindow1: TMenuItem;
    EditSignal1: TMenuItem;
    SignalInfo1: TMenuItem;
    Close2: TMenuItem;
    LabelTitle: TLabel;
    BINtoSAN1: TMenuItem;
    SaveDialog1: TSaveDialog;
    BintoDSA: TMenuItem;
    Save1: TMenuItem;
    PrintDialog1: TPrintDialog;
    MetaFile1: TMenuItem;
    procedure Open1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Display1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormResize(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure SaveScr(Left, Top, Right, Bottom :Word);
    procedure RestScr(Left, Top, Right, Bottom :Word);
    procedure Show1Click(Sender: TObject);
    procedure ExternalWindow1Click(Sender: TObject);
    procedure Close2Click(Sender: TObject);
    procedure EditSignal1Click(Sender: TObject);
    procedure SaveAs1Click(Sender: TObject);
    procedure BintoDSAClick(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure N2DDelay1Click(Sender: TObject);
    procedure CorrelationDimension1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction); override;
    procedure PeaktoPeakDistance1Click(Sender: TObject);
    procedure Print1Click(Sender: TObject);
    procedure BitMap1Click(Sender: TObject);
    procedure MetaFile1Click(Sender: TObject);
    procedure Autocorrelation1Click(Sender: TObject);
  private
  public
    procedure SpeedPanelClick(Sender:TObject);
    procedure DisplayHint(Sender: TObject);
  end;

var
  DSAForm1 : TDSAForm1;
  Lag: Byte;

implementation

uses About, Unit2, Unit3, UnitList, EditSignal, DelayPlot2D, LagPlot2D,
  Correlation, Corparams;

{$R *.DFM}
```

```
const
    bZoom : boolean = false;
    ModalResult : TModalResult = 2;

var
  i, j: Smallint;
  Screen:array[1..MaxWidth, 0..MaxHeight]of TColor;

procedure TDSAForm1.DisplayHint(Sender: TObject);
begin
 PanelStatus1.Caption := Application.Hint;
end;

{Saves a rectangular region of the screen determined by the coordinates into a 2-d array}
procedure TDSAForm1.SaveScr(Left, Top, Right, Bottom :Word);
begin
 if Left   > Right then  Swap(Left, Right);
 if Bottom > Top   then  Swap(Top, Bottom);
 for i:=Left to Right do
   for j:=Bottom to Top do
     Screen[i,j]:=Canvas.Pixels[i,j];
end; {SaveScr}

{Restores rectangular region of the screen determined by the coordinates from a 2-d array}
procedure TDSAForm1.RestScr(Left, Top, Right, Bottom :Word);
begin
 if Left   > Right then  Swap(Left, Right);
 if Bottom > Top   then  Swap(Top, Bottom);
 for i:=Left to Right do
   for j:=Bottom to Top do
     Canvas.Pixels[i,j]:=Screen[i,j];
end; {RestScr}

{Displays X, Y coordinate values of the mouse pointer}
procedure XYLabels(X, Y:integer);
begin
 with DSAForm1 do
 with Calib, pSignal^ do begin
  LabelX.Caption:=FloatToStrF(((X-X1)/r+First)*Scale, ffFixed, 7, 4);
  LabelY.Caption:=FloatToStrF((BGraph-Y)*Max_Min/(GraphHeight-2)+Min, ffFixed, 7, 4);
 end; {with}
end; {XYLabels}

{Assigns a SpeedPanel to every newly opened signal. User can then use this SpeedPanel to treat the
corresponding signal}
procedure AssignSpeedPanel(var pTheSignal:TSignalPtr; CaptionStr:String);
var
 ASignal: TSignalPtr;
begin
 with pTheSignal^ do begin
  spPanel:=TSpeedPanel.Create(DSAForm1.PanelBottom);
  spPanel.Parent:=DSAForm1.PanelBottom;
  with spPanel do begin
   PopupMenu:=DSAForm1.PopupMenu1;
   OnClick:=DSAForm1.SpeedPanelClick;
   {Hint:=HintStr;}
   Down:=True;
   ShowHint:=True;
   Alignment:=taLeftJustify;
```

```
    Caption:=CaptionStr;
    Height:=25;
    Top:=4;
    Tag:=SignalList.Count;
    SignalList.Add(pTheSignal);
    with SignalList do
     for i:=0 to Count-1 do begin
      ASignal:=Items[i];
      with ASignal^.SpPanel do begin
       if Count>0 then begin
        Width:=Round((DSAForm1.ClientWidth-32)/Count);
        if Width>140 then Width:=140;
       end
       else Width:=140;
       Left:=4+Tag*(Width+4);
      end; {with}
     end; {for}
   end;{with Panel}
  end;
end; {AssignSpeedPanel}

{Opens a DSA signal file
 Returns: The size of the file opened on success, or 0 on error}
function OpenDSA(const FileName:string):integer;
var
  ASignal:TSignalPtr;
  F:System.Text;
  ScaleStr,SizeStr,strTimeLength:string[30];
  EndStr: String;
begin
  {$I-}
  System.AssignFile(F, FileName);
  System.FileMode := 0;  { Set file access to read only }
  System.Reset(F);
try
  if IOResult=0 then begin
    New(pNewSignal);
    with pNewSignal^ do begin
     Path:=FileName;
     readln(F, Header); {title of the file :DSA Data File}
     readln(F, DataType);
     Delete(DataType, 1, 9);
     DataType:=TrimLeft(DataType);
     readln(F, Title);
     Delete(Title, 1, 6);
     Title:=TrimLeft(Title);
     readln(F, SizeStr);
     Delete(SizeStr, 1, 5);
     readln(F, ScaleStr);
     Delete(ScaleStr, 1, 6);
     readln(F, XUnit);
     Delete(XUnit, 1, 6);
     XUnit:=TrimLeft(XUnit);
     readln(F, YUnit);
     Delete(YUnit, 1, 6);
     YUnit:=TrimLeft(YUnit);
     Size:=StrToInt(SizeStr);
     Scale:=StrToFloat(ScaleStr);
     readln(F);
```

```
    {..Read Comment.}
    Comment:=TStringList.Create;
    readln(F, EndStr);
    while EndStr<>'end comment' do begin
     Comment.Add(EndStr);
     readln(F,EndStr);
    end;
    {while, end of Read Comment}
    Form2:=TForm2.Create(Application);
    Form2.Label4.Caption:=IntToStr(Size);
    Str(Size*Scale:5:2, strTimeLength);
    Form2.Label5.Caption:=strTimeLength + ' sec';
    Form2.Edit3.Text:=Form2.Label4.Caption;
    ModalResult:=Form2.ShowModal;
    if ModalResult = mrOk then begin
      Size:=StrToInt(Form2.Edit3.Text);
      FormList.ListBox1.Items.Add(Title);
      i:=1;
      While not(EOF(F) or (i>Size)) do begin
       readln(F,Series[i]);
       inc(i);
      end; {while}
      {First:=1;
      Last:=Size;}
      AssignSpeedPanel(pNewSignal, Title);
      spPanel.Hint:=Title+'|'+Comment.Strings[0];
      with DSAForm1 do begin
       if pSignal<>nil then begin
        pSignal^.spPanel.Down:=False;
       end;
       pSignal:=SignalList.Items[SignalList.Count-1];
      end;
      OpenDSA:=Size;
    end {if}
    else
     begin
      Dispose(pNewSignal);
      OpenDSA:=0;
     end;
    Form2.Free;
   end; {with pNewSignal^ do}
  end;{if}
finally
  System.CloseFile(F);
  {$I+}
end; {try}
end; {OpenDSA}


{Opens a SAN signal file
 Returns: The size of the file opened on success, or 0 on error}
function OpenSAN(const FileName:string):integer;
var
  ASignal:TSignalPtr;
  Datatype:String;
  F:System.Text;
  ScaleStr,SizeStr,EndStr,strTimeLength:string;
  SizeTemp, FilSize:integer;
  ScaleTemp:Single;
```

```
begin
  {$I-}
  System.AssignFile(F, FileName);
  System.FileMode := 0;  { Set file access to read only }
  System.Reset(F);
  if IOResult=0 then begin
    New(pNewSignal);
    with pNewSignal^ do begin
      Path:=FileName;
      readln(F); {title of the file :DSA Data File}
      readln(F, DataType);
      Delete(DataType, 1, 9);
      DataType:=TrimLeft(DataType);
      readln(F, Title);
      Delete(Title, 1, 6);
      Title:=TrimLeft(Title);
      readln(F, SizeStr);
      Delete(SizeStr, 1, 5);
      readln(F, ScaleStr);
      Delete(ScaleStr, 1, 6);
      readln(F);  {readln(F,shift)}
      readln(F, XUnit);
      Delete(XUnit, 1, 6);
      XUnit:=TrimLeft(XUnit);
      readln(F, YUnit);
      Delete(YUnit, 1, 6);
      YUnit:=TrimLeft(YUnit);
      Size:=StrToInt(SizeStr);
      Scale:=StrToFloat(ScaleStr);
      for i:=1 to 5 do
        readln(F);
      {..Read Comment.}
      Comment:=TStringList.Create;
      readln(F, EndStr);
      while EndStr<>'end comment' do begin
        Comment.Add(EndStr);
        readln(F,EndStr);
      end;
      {while, end of Read Comment}
      Form2:=TForm2.Create(Application);
      Form2.Label4.Caption:=IntToStr(Size);
      Str(Size*Scale:5:2, strTimeLength);
      Form2.Label5.Caption:=strTimeLength + ' sec';
      Form2.Edit3.Text:=Form2.Label4.Caption;
      ModalResult:=Form2.ShowModal;
      if ModalResult = mrOk then begin
        Size:=StrToInt(Form2.Edit3.Text);
        FormList.ListBox1.Items.Add(Title);
        i:=1;
        While not(EOF(F) or (i>Size)) do begin
          readln(F,Series[i]);
          inc(i);
        end; {while}
        AssignSpeedPanel(pNewSignal, Title);
        spPanel.Hint:=Title+'|'+Comment.Strings[0];
        with DSAForm1 do begin
          if pSignal<>nil then begin
            pSignal^.spPanel.Down:=False;
          end;
```

```
        pSignal:=SignalList.Items[SignalList.Count-1];
       end;
       OpenSAN:=Size;
     end {if}
     else
      begin
       Dispose(pNewSignal);
       OpenSAN:=0;
      end;
     Form2.Free;
   end; {with pNewSignal^ do}
  end;{if}
  System.CloseFile(F);
  {$I+}
end; {OpenSAN}

{Opens a binary signal file
 Returns: The size of the file opened on success, or 0 on error}
function OpenBinary(const FileName:String):integer;
const
  NByte : Word = 4;
var
  F:File;
  strTimeLength:string;
begin
  {$I-}
  AssignFile(F, FileName);
  FileMode := 0;  { Set file access to read only }
  Reset(F, NByte);
  if IOResult=0 then begin
   New(pNewSignal);
   with pNewSignal^ do begin
    Path:=FileName;
    Size:=FileSize(F);
    Form2:=TForm2.Create(Application);
    Form2.Label4.Caption:=IntToStr(Size);
    Str(Size*NByte/SamplingRate:5:2, strTimeLength);
    Form2.Label5.Caption:=strTimeLength + ' sec';
    Form2.Edit3.Text:=Form2.Label4.Caption;
    ModalResult:=Form2.ShowModal;
    Size:=StrToInt(Form2.Edit3.Text);
    Comment:=TStringList.Create;
    if ModalResult = mrOk then begin
     FormList.ListBox1.Items.Add(FileName);
     Size:=StrToInt(Form2.Edit3.Text);
     BlockRead(F,Series,Size);
     Title:=(FileName);
     AssignSpeedPanel(pNewSignal, FileName);
     spPanel.Hint:=FileName;
     with DSAForm1 do begin
      if pSignal<>nil then begin
       pSignal^.spPanel.Down:=False;
      end;
      pSignal:=SignalList.Items[SignalList.Count-1];
     end;
     OpenBinary:=Size;
    end {if}
    else begin
     Dispose(pNewSignal);
```

```
      OpenBinary:=0;
    end; {else}
   end; {with}
   Form2.Free;
  end;{if}
  CloseFile(F);
  {$I+}
end; {OpenBinary}

procedure SaveAsDSA(const Filename:String);
var
  F: System.Text;
begin
  {$I-}
  System.AssignFile(F, FileName);
  rewrite(F);
 try
  with DSAForm1.pSignal^ do begin
    writeln(F,'DSA data file');
    writeln(F,'datatype: SIGNAL');
    writeln(F,'title: ' + Title);
    writeln(F,'size: ', Size);
    writeln(F,'scale:', Scale:6:3);
    writeln(F,'xunit:' + XUnit);
    writeln(F,'yunit:' + YUnit);
    writeln(F,'begin comment');
    with Comment do
     for i:=0 to Count-1 do
       writeln(F,' ' + Strings[i]);
    writeln(F,'end comment');
    for i:=1 to Size do begin
     append(F);
     writeln(F, Series[i]);
    end; {for}
  end; {with pSignal^ do}
 finally
  CloseFile(F);
  {$I+}
 end; {try}
end; {SaveAsDSA}

procedure SaveAsASCII(const Filename:String);
var
  F: System.Text;
begin
  {$I-}
  System.AssignFile(F, FileName);
  rewrite(F);
 try
  with DSAForm1.pSignal^ do begin
    for i:=1 to Size do begin
     append(F);
     writeln(F, Series[i]);
    end; {for}
  end; {with pSignal^ do}
 finally
  CloseFile(F);
  {$I+}
 end;
```

```
end; {SaveAsASCII}

procedure SaveAsBin(const FileName:String);
var
  F:File;
begin
  {$I-}
  System.AssignFile(F, FileName);
  rewrite(F, 4);
  with DSAForm1.pSignal^do begin
    try
      BlockWrite(F,Series,Size{,result});
    except
      on Exception do MessageDlg('Error saving '+ Path, mtError, [mbOK], 0);
    end;
  end; {with pSignal^ do}
  CloseFile(F);
  {$I+}
end; {SaveAsBin}

procedure TDSAForm1.BintoDSAClick(Sender: TObject);
var
  ExtSize: byte ;
  ExtOpen, ExtSave: string[3];
  FileSize: integer;
  bOpen, bSave: boolean;
begin
  inherited;
  with OpenDialog1 do begin
    Filter:='Single files (*.SNG)|*.SNG|Binary files (*.BIN)|*.BIN';
    repeat
     bOpen:=Execute;
     if bOpen then begin
      i:=length(FileName)-1;
      ExtSize:=1;
      while Filename[i] <> '.' do begin
        dec(i); inc(ExtSize);
      end;
      ExtOpen:=Copy(FileName, i+1, ExtSize);
      if (ExtOpen='bin')or(ExtOpen='sng') then begin
        FileSize:=OpenBinary(FileName);
        if FileSize>0 then begin
          Calibrate(DSAForm1);
          FormPaint(DSAForm1);
          DSAForm1.PopupMenu1.PopupComponent:=pNewSignal^.spPanel;
          EditSignal1Click(Sender);
          with SaveDialog1 do begin
            Filter:='DSA files (*.DSA)|*.DSA';
            DefaultExt:='dsa';
            repeat
              bSave:=Execute;
              if bSave then begin
                i:=length(FileName)-1;
                ExtSize:=1;
                while Filename[i] <> '.' do begin
                  dec(i); inc(ExtSize);
                end;
                ExtSave:=Copy(FileName, i+1, ExtSize);
                if (ExtSave='dsa') then SaveAsDSA(FileName)
```

```
          else MessageDlg('File extension must be dsa', mtError, [mbOK], 0);
         end;
        until (ExtSave='dsa')or not(bSave);
       end;{with}
      end; {if}
     end {if}
     else MessageDlg('File extension must be sng or bin', mtError, [mbOK], 0);
    end {if}
    else
   until (ExtOpen='sng')or(ExtOpen='bin')or not(bOpen);
  end; {with}
end; {BintoAscii1Click}

procedure TDSAForm1.Open1Click(Sender: TObject);
var
 ExtSize:byte ;
 Ext:string;
 FileSize:integer;
begin
 with OpenDialog1 do begin
  Filter:='DSA files (*.DSA)|*.DSA|Santis files (*.SAN)|*.SAN|' +
       'Text files (*.TXT)|*.TXT| Binary files (*.BIN)|*.BIN|' +
       'Any File (*.*)|*.*';
  FilterIndex:=5;
  if Execute then begin
   i:=length(FileName)-1;
   ExtSize:=1;
   while Filename[i] <> '.' do begin
    dec(i); inc(ExtSize);
   end;
   Ext:=Copy(FileName, i+1, ExtSize);
   if (Ext='bin')or(Ext='sng') then
    FileSize:=OpenBinary(FileName)
   else if (Ext='san') then
    FileSize:=OpenSAN(FileName)
   else if (Ext='dsa')then
    FileSize:=OpenDSA(FileName);
   {end
   else begin
    OpenTxt(FielName);}
   if FileSize>0 then begin
    with pSignal^ do begin
     if Scale=0 then begin
      Scale:=1;
      XUnit:='(pts)';
     end; {if}
     Calib.First:=1; Calib.Last:=Size;
     Max := 0.0; Min := 0.0;
     for i:=Calib.First to Calib.Last do begin
      if Series[i] > Max then Max := Series[i];
      if Series[i] < Min then Min := Series[i]
     end; {for}
    end;
    pNewSignal:=nil;
    Calibrate(DSAForm1);
    FormPaint(DSAForm1);
   end;
  end; {if}
 end; {with}
```

```
end; {Open1Click}

procedure TDSAForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TDSAForm1.Display1Click(Sender: TObject);
var
  strXi:string;
begin
  Form3:=TForm3.Create(Application);
  for i:=1 to pSignal^.Size do begin
    str(pSignal^.Series[i]:10:4, strXi);
    Form3.Memo1.Lines.Add(strXi);
  end;
  Form3.ShowModal;
  Form3.Free;
end;

procedure TDSAForm1.About1Click(Sender: TObject);
begin
  AboutBox:=TAboutBox.Create(Application);
  AboutBox.ShowModal;
  AboutBox.Free;
end;

procedure TDSAForm1.FormPaint(Sender: TObject);
var
  NewRect, OldRect:Trect;
  nXAxis:Word;
begin
  {.....ERASE FRAME AREA, DRAW FRAME AND AXES.....}
  with Canvas do begin
    nXAxis:=BGraph+2;
    OldRect := Rect(0, TGraph, ClientWidth, ClientHeight);
    Brush.Color:=clScrollBar;
    FillRect(OldRect);
    Brush.Color:=clWhite;
    Pen.Color:=clBlack;
    NewRect:=Rect(AlmostNone,TGraph-UnitYHeight,ClientWidth-
AlmostNone,nXAxis+UnitXHeight);
    FillRect(NewRect);
    MoveTo(X1-2, TGraph);
    LineTo(X1-2, nXAxis);
    LineTo(ClientWidth-AlmostNone, nXAxis);
  end; {with}
  {.....END OF DRAW FRAME AND AXES.....}
  if (pSignal<>nil) then begin
     LabelTitle.Caption:=pSignal^.Title;
     Plot(DSAForm1);
  end;
end; {FormPaint}

procedure TDSAForm1.FormResize(Sender: TObject);
var
  ASignal: TSignalPtr;
begin
{.....RESIZE.....}
```

```
  with PanelStatus1 do begin
    Top:=PanelBottom.Top-Height-AlmostNone{?};
    Width:=DSAForm1.ClientWidth-PanelStatus2.Width-AlmostNone-4{?};
    PanelStatus2.Left:=Left+Width+AlmostNone;
    PanelStatus2.Top:=Top;
    BGraph := PanelStatus1.Top-AlmostNone-4-UnitXHeight;
  end;
  GraphHeight:=BGraph-TGraph;
  with SignalList do
    for i:=0 to Count-1 do begin
      ASignal:=Items[i];
      with ASignal^.SpPanel do begin
        if Count>0 then begin
          Width:=Round((DSAForm1.ClientWidth-32)/Count);
          if Width>140 then Width:=140;
        end
        else Width:=140;
        Left:=4+Tag*(Width+4);
      end; {with}
    end; {for}
{.....END RESIZE.....}
  if pSignal<>nil then Calibrate(DSAForm1);
  FormPaint(DSAForm1);
end;

procedure TDSAForm1.FormCreate(Sender: TObject);
begin
  Application.OnHint := DisplayHint;
  Application.ShowHint:=True;
  New(pSignal);
  pSignal:=nil;
  SignalList:=TList.Create;
  for i:=X1 to MaxWidth do
    for j:=Top to MaxHeight do
      Screen[i,j]:=clWhite;
  AutoScroll:=False;
  BGraph:=PanelStatus1.Top-AlmostNone-4-UnitXHeight;
  TGraph:=LabelTitle.Top+LabelTitle.Height+UnitYHeight;
  GraphHeight:=BGraph-TGraph;
  {...Align Status bars.....}
  with PanelStatus1 do begin
    Top:=PanelBottom.Top-Height{?};
    Width:=DSAForm1.ClientWidth-PanelStatus2.Width-AlmostNone-4{?};
    PanelStatus2.Left:=Left+Width+AlmostNone;
    PanelStatus2.Top:=PanelStatus1.Top;
  end;
  FormList:=TFormList.Create(FormList);
end;

procedure TDSAForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
  OutCome: Word;
begin
  CanClose := False;
  OutCome := MessageBox(Handle, 'Exit DSA?',
                        'DSA Dynamical Systems Analyzer',
                        mb_YesNo or mb_IconQuestion);
  if OutCome = id_Yes then
    CanClose := True;
```

```
end;

procedure TDSAForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if SignalList.Count>0 then begin
    with (Sender as TForm4).Calib do begin
      if X>X1 then TempFirst:=Round((X-X1)/r)+First
            else TempFirst:=First;
      FirstX:=X; FirstY:=Y;
      LastX:=X; LastY:=Y;
      bZoom:=true;
    end; {with}
  end; {if}
end;

procedure TDSAForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Temp:word;
begin
 if bZoom then begin
  with  (Sender as TForm4).Calib do begin
    TempLast:=Round((X-X1)/r)+First;
    bZoom:=false;
    if TempLast < TempFirst then
      begin {swap(first, last)}
        if TempFirst<Last then Last:=TempFirst;
        First:=TempLast;
      end
    else if TempLast > TempFirst then
          begin
            First:=TempFirst;
            if TempLast<Last then Last:=TempLast;
          end;
  end; {with}
  Canvas.Pen.Style:=psSolid;
  try
    FormPaint(Sender as TDSAForm1);
  except
    on EInvalidCast do inherited FormPaint(Sender);
  end; {try}
 end; {if}
end; {MouseUp}

procedure TDSAForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if (Y<BGraph+UnitXHeight)and(Y>TGraph-UnitYHeight)and(pSignal<>nil) then
    begin
      with (Sender as TForm4),(Sender as TForm4).Canvas do begin
       if bZoom then begin
         RestScr(FirstX,FirstY,LastX,LastY);
         LastX:=X; LastY:=Y;
         SaveScr(FirstX,FirstY,LastX,LastY);
         MoveTo(FirstX, FirstY);
         Pen.Style:=psDot;
         Pen.Color:=clBlack;
         LineTo(X, FirstY); LineTo(X, Y);   {draw.....}
```

```
        MoveTo(FirstX, FirstY);          {zoom.....}
        LineTo(FirstX, Y); LineTo(X, Y);  {rectangle..}
       end; {if}
       Cursor:=crCross;
     end; {with}
     XYLabels(X,Y);
    end
   else
    Cursor:=crDefault;
end;

procedure TDSAForm1.Show1Click(Sender: TObject);
begin
  SpeedPanelClick(PopupMenu1.PopupComponent);
end;

procedure TDSAForm1.SpeedPanelClick(Sender: TObject);
begin
  inherited;
  with (Sender as TSpeedPanel) do begin
    Down:=not(Down);
    if not(Down) then begin
      pSignal:=nil;
      LabelTitle.Caption:='';
      FormPaint(DSAForm1);
    end
    else begin
      if (pSignal<>nil) then
        pSignal^.spPanel.Down:=False;
      pSignal:=SignalList.Items[Tag];
      Calibrate(DSAForm1);
      FormPaint(DSAForm1);
    end;
  end;
end;

procedure TDSAForm1.Close2Click(Sender: TObject);
var
  ASignal:TSignalPtr;
begin
  inherited;
  with SignalList do begin
    ASignal:=Items[PopupMenu1.PopupComponent.Tag];
    with ASignal^ do begin
      if pForm<>nil then begin
        (pForm^ as TForm4).Close;
      end;
      if spPanel.Down then begin
        pSignal:=nil;
        LabelTitle.Caption:='';
        FormPaint(DSAForm1);
      end;
    end; {with}
{   Dispose(Items[PopupMenu1.PopupComponent.Tag]);}
    Remove(Items[PopupMenu1.PopupComponent.Tag]);
    PopupMenu1.PopupComponent.Free;
    with SignalList do begin
      for i:=0 to Count-1 do begin
        ASignal:=Items[i];
```

```
      with ASignal^.SpPanel do begin
       Tag:=i;
       Width:=Round((DSAForm1.ClientWidth-32)/Count);
       if Width>140 then Width:=140;
       Left:=4+Tag*(Width+4);
      end; {with}
     end; {for}
    end; {with}
  end; {with}
end;

procedure TDSAForm1.EditSignal1Click(Sender: TObject);
var
  ASignal:TSignalPtr;
begin
  inherited;
  try
    ASignal:=SignalList.Items[(Sender as TSpeedPanel).Tag];
  except
    on EInvalidCast do ASignal:=SignalList.Items[PopupMenu1.PopupComponent.Tag];
  end; {try}
  FormEdit:=TFormEdit.Create(Application);
  with FormEdit, ASignal^, Calib do begin
    EditHeader.Text:=Header;
    EditDataType.Text:=DataType;
    EditTitle.Text:=Title;
    EditSize.Text:=IntToStr(Size);
    EditScale.Text:=FloatToStrF(Scale, ffFixed, 7, 4);
    EditXUnit.Text:=XUnit;
    EditYUnit.Text:=YUnit;
    MemoComment.Clear;
    with Comment do
      for i:=0 to Count-1 do
        MemoComment.Lines.Add(Strings[i]);
    ModalResult:=FormEdit.ShowModal;
    if ModalResult = mrOk then begin
      Header:=EditHeader.Text;
      DataType:=EditDataType.Text;
      Title:=EditTitle.Text;
      Size:=StrToInt(EditSize.Text);
      First:=1;
      Last:=Size;
      Scale:=StrToFloat(EditScale.Text);
      XUnit:=EditXUnit.Text;
      YUnit:=EditYUnit.Text;
      with Comment do begin
        Clear;
        AddStrings(MemoComment.Lines);
        spPanel.Hint:=Title+'|'+Strings[0];
      end; {with}
      spPanel.Caption:=Title;
      Calibrate(DSAForm1);
      FormPaint(DSAForm1);
    end;
  end; {with}
end;

procedure TDSAForm1.SaveAs1Click(Sender: TObject);
var
```

```
  ExtSize:Byte;
  Ext:String[3];
begin
  with SaveDialog1 do begin
    Filter:='DSA files (*.DSA)|*.dsa|Santis files (*.SAN)|*.san|' +
          'Text files (*.ASC)|*.asc| Binary files (*.BIN)|*.bin|' +
          'Any File (*.*)|*.*';
    DefaultExt:='dsa';
    if Execute then begin
      i:=length(FileName)-1;
      ExtSize:=1;
      while Filename[i] <> '.' do begin
        dec(i); inc(ExtSize);
      end;
      Ext:=Copy(FileName, i+1, ExtSize);
      if (Ext='bin')or(Ext='sng') then
        SaveAsBin(FileName)
      else if (Ext='dsa') then
        SaveAsDSA(FileName)
      else if (Ext='asc') then
        SaveAsASCII(FileName);
    end; {if}
  end; {with}
end;

procedure TDSAForm1.Save1Click(Sender: TObject);
var
  ExtSize:byte ;
  Ext:string;
begin
  with pSignal^ do begin
      i:=Length(Path)-1;
      ExtSize:=1;
      while Path[i] <> '.' do begin
        dec(i); inc(ExtSize);
      end;
      Ext:=Copy(Path, i+1, ExtSize);
      if {CompareText(Ext,'dsa')=0}Ext='dsa' then SaveAsDSA(Path)
      else if (Ext='bin')or(Ext='sng') then SaveAsBin(Path)
      else SaveAs1Click(Sender);
  end; {with}
end;

procedure TDSAForm1.ExternalWindow1Click(Sender: TObject);
var
  {AForm4: ^TForm4;}
  ASignal: TSignalPtr;
begin
  ASignal:=SignalList.Items[DSAForm1.PopupMenu1.PopupComponent.Tag];
  with ASignal^ do begin
    if pForm=nil then begin
      GetMem(pForm, Sizeof(TForm4));
      pForm^:=TForm4.Create(Application);
    end
    else
      pForm^.Show;
  end;
end;
```

```
procedure TDSAForm1.N2DDelay1Click(Sender: TObject);
begin
 Form2DDelay:=TForm2DDelay.Create(Application);
 {ModalResult:=Form2DDelay.ShowModal;
 if ModalResult = mrOk then begin
  Lag:=StrtoInt(Form2DDelay.Edit1.Text);
  TFormLag2D.Create(Application);
 end;
 Form2DDelay.Free;}
end;  {N2DDelay1Click}

procedure TDSAForm1.CorrelationDimension1Click(Sender: TObject);
begin
{  inherited;}
 FormCorParams:=TFormCorParams.Create(Application);
 try
  ModalResult:=FormCorParams.ShowModal;
  if ModalResult = mrOk then begin
   FormCorDim:=TFormCorDim.Create(Application);
   {FormCorDim.Free;}
  end
 finally
  FormCorParams.Free
 end; {try}
end;

procedure TDSAForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 {Action:=caFree;}
end;

procedure TDSAForm1.PeaktoPeakDistance1Click(Sender: TObject);
const
 Sum: Word =0;
 Dist: Word = 1;
 {Weight: Word = 0;}
 Counter: Word = 0;
 nPreviousPeak: Word = 1;
var
 k, nBound: Word;
begin
{  inherited; }
 New(pNewSignal);
 with pNewSignal^ do begin
  Title:='Peak2peak(' + pSignal^.Title + ')';
  Scale:=pSignal^.Scale;
  Size:=pSignal^.Size;
  XUnit:=pSignal^.XUnit;
  YUnit:='Distance(pts)';
  FormList.ListBox1.Items.Add(Title);
  nBound:=2;
  While ((pSignal^.Series[nBound]<pSignal^.Series[nBound-1]) xor {IsPeak?}
       (pSignal^.Series[nBound]<pSignal^.Series[nBound+1])) do Inc(nBound);
  Sum:=0; Dist:= 1; Counter:= 0; nPreviousPeak:= nBound;
  for i:=nBound+1 to Size-1 do begin
   if ((pSignal^.Series[i]<pSignal^.Series[i-1]) xor {IsNotPeak?}
      (pSignal^.Series[i]<pSignal^.Series[i+1]))then Inc(Dist)
   else begin
    Inc(Counter);
```

```
      Sum:=Sum+Dist;
      for k:=nPreviousPeak to i-1 do
        Series[k]:=Dist;
      nPreviousPeak:=i;
      Dist:=1;
     end;
    end; {for}
    Comment:=TStringList.Create;
    with Comment do begin
     Add('Peak to peak distance signal of ' + pSignal^.Title +
           ', Average=' + FloatToStrF(Sum/Counter, ffFixed, 4, 4));
     AssignSpeedPanel(pNewSignal, Title);
     spPanel.Hint:=Title+'|'+Strings[0];;
    end; {with}
    if pSignal<>nil then pSignal^.spPanel.Down:=False;
    pSignal:={SignalList.Items[SignalList.Count-1];} pNewSignal;
    with pSignal^ do begin
     DSAForm1.Calib.First:=1; DSAForm1.Calib.Last:=Size;
     Max := 1.0; Min := Size/2;
     for i:=Calib.First to Calib.Last do begin
      if Series[i] > Max then Max := Series[i];
      if Series[i] < Min then Min := Series[i]
     end; {for}
     pNewSignal:=nil;
     Calibrate(DSAForm1);
     FormPaint(DSAForm1);
    end;
  end; {with pNewSignal}
end;

procedure TDSAForm1.Print1Click(Sender: TObject);
begin
  inherited;
  with PrintDialog1 do
    if Execute then
      Print;
end;

procedure TDSAForm1.BitMap1Click(Sender: TObject);
var
  ABitMap: TBitMap;
begin
  {inherited;}
  ABitMap:=TBitMap.Create;
  ABitMap.Height:=ClientHeight;
  ABitMap.Width:=ClientWidth;
  ABitMap.Canvas.CopyRect(ClientRect, Canvas, ClientRect);
  with SaveDialog1 do begin
    Filter:='bitmap files (*.bmp)|*.bmp';
    DefaultExt:='bmp';
    if Execute then begin
      ABitMap.SavetoFile(FileName);
    end;
  end;
  ABitMap.Free;
end;

procedure TDSAForm1.MetaFile1Click(Sender: TObject);
var
```

```
  AMetaFile: TMetaFile;
begin
  {inherited;}
  AMetaFile:=TMetaFile.Create;
  with TMetafileCanvas.Create(AMetafile, 0) do
    try
      CopyRect(ClientRect, Canvas, ClientRect);
    finally
      Free;
    end;
  with SaveDialog1 do begin
    Filter:='MetaFile files (*.emf)|*.emf';
    DefaultExt:='emf';
    if Execute then begin
      AMetaFile.SavetoFile(FileName);
    end;
  end;
  AMetaFile.Free;
end;

procedure TDSAForm1.Autocorrelation1Click(Sender: TObject);
var
  L: Word;
  Limit, Sum, Av, Num, Den: Single;
begin
  New(pNewSignal);
  with pNewSignal^ do begin
    Title:='AutoCorrelation(' + pSignal^.Title + ')';
    Scale:=pSignal^.Scale;
    Size:=pSignal^.Size-1;
    XUnit:='Lag - ' + pSignal^.XUnit;
    YUnit:='C(Lag) - ' + pSignal^.YUnit;
    FormList.ListBox1.Items.Add(Title);
    {...Begin Computation.....}
    Sum:=0;
    for i:=1 to pSignal^.Size do
      Sum:=Sum+pSignal^.Series[i];
    Av:=Sum/Size;
    for L:=1 to Size do begin
      Num:=0; Den:=0;
      for i:=1 to pSignal^.Size-L do begin
        Num:=Num + (pSignal^.Series[i+L]-Av)*(pSignal^.Series[i]-Av);
        Den:=Den + Sqr(pSignal^.Series[i]-Av);
      end;
      Series[L]:=Num/Den;
    end; {for}
    Limit:=Series[1]/Exp(1);  {find 1/e crossing}
    L:=2;
    while Series[L]>Limit do Inc(L);
    Dec(L);
    {if Series[L]<>0 then begin     {find first zero crossing crossing
      if Series[L]>0 then
        while Series[L]>0 do Inc(L)
      else
        while Series[L]<0 do Inc(L);
      Dec(L);
    end; }
    {...End Computation.....}
    Comment:=TStringList.Create;
```

```
    with Comment do begin
      Add('Linear Autocorrelation Function of ' + pSignal^.Title +
        ', 1/e crossed at Lag=' + InttoStr(L));
      AssignSpeedPanel(pNewSignal, Title);
      spPanel.Hint:=Title+'|'+Strings[0];;
    end; {with}
    if pSignal<>nil then pSignal^.spPanel.Down:=False;
    pSignal:={SignalList.Items[SignalList.Count-1];} pNewSignal;
    with pSignal^ do begin
      DSAForm1.Calib.First:=1; DSAForm1.Calib.Last:=Size;
      Max := 1.0; Min := Size/2;
      for i:=Calib.First to Calib.Last do begin
        if Series[i] > Max then Max := Series[i];
        if Series[i] < Min then Min := Series[i]
      end; {for}
      pNewSignal:=nil;
      Calibrate(DSAForm1);
      FormPaint(DSAForm1);
    end;
  end; {with pNewSignal}
end;

end. {end of main}
```

UnitPlot.pas:

```
unit UnitPlot;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, DSALib, Unitlist, StdCtrls, Menus;
const
  MaxNoPts = 8000;
  AlmostNone = 2; {height difference between form objects that are in contact}
  X1 = 40; {left indentation for y-axis of the graph}
  MaxHeight = 550;
  MaxWidth = 800;
  MinWidth = 300;
  UnitYHeight = 20;
  UnitXHeight = 20;
type
  TFormPtr = ^TForm;
  TSeries = array [1..MaxNoPts] of single;
  TSignal = record
      Series:TSeries;
      Path:String[150];
      Header:String[100];
      DataType:String[80];
      Title:String[80];
      XUnit:String[40];
      YUnit:String[40];
      Size:Word;
      Scale:Single;
      Comment:TStringList;
      Max:Single;
      Min:Single;
      spPanel:TSpeedPanel;
```

```
    pFormLag,pForm:TFormPtr;
    {First:Word;
    Last:Word;}
  end; {TSignal}

  TPlot = record
    Series:TSeries;
    First:Word;
    Last:Word;
    {XUnit:String[40];
    YUnit:String[40];
    Size:Word;
    Scale:Single;
    Max:Single;
    Min:Single;}
  end;

  TSignalPtr = ^TSignal;

  TForm4 = class(TForm)
   MainMenu2: TMainMenu;
   File2: TMenuItem;
   Save2: TMenuItem;
   SaveAs2: TMenuItem;
   PrinttoFile2: TMenuItem;
   Print2: TMenuItem;
   Close3: TMenuItem;
   PrintDialogPlot: TPrintDialog;
   Bitmap2: TMenuItem;
   Metafile2: TMenuItem;
   procedure FormPaint(Sender: TObject); virtual;
   procedure FormCreate(Sender: TObject); virtual;
   procedure FormResize(Sender: TObject); virtual;
   procedure FormClose(Sender: TObject; var Action: TCloseAction); virtual;
   procedure Print2Click(Sender: TObject);
   procedure Bitmap2Click(Sender: TObject);
  private
  public
   pSignal:TSignalPtr;
   Calib:TPlot;
   BGraph, TGraph, GraphHeight:Word;
   FirstX,FirstY, LastX,LastY:Word;
  end;

var
  SignalList: TList;
  pNewSignal: TSignalPtr;
  Form4: TForm4;
  max_min: single;
  r: single;
  TempFirst, TempLast: integer;
  i, j: Word;

procedure Plot(Sender:TObject);
procedure Calibrate(Sender:Tobject);

implementation

uses Main;
```

```
{$R *.DFM}

procedure Plot(Sender:TObject);
var
  RectWidth : word;
begin
 with (Sender as TForm4) do
 with Canvas, Calib, pSignal^ do begin
  {..Calibrate wrt x axis....begin}
  RectWidth:=ClientWidth-X1-24;
  r:=RectWidth/(Last-First);
  {..Calibrate wrt x axis....end}
  TextOut(4, TGraph-UnitYHeight+2, YUnit);
  MoveTo(X1, BGraph);
  LineTo(X1, BGraph+6);
  TextOut(X1-8, BGraph+7, FloatToStrF(First*Scale, ffFixed, 4, 1));
  MoveTo(X1, BGraph-2);
  LineTo(X1-6, BGraph-2);
  TextOut(8, BGraph-6, FloatToStrF(Min, ffFixed, 3, 1));
  MoveTo(X1, BGraph-1-GraphHeight div 2);
  LineTo(X1-6, BGraph-1-GraphHeight div 2);
  TextOut(8, BGraph-5-GraphHeight div 2, FloatToStrF((Max+Min)/2, ffFixed, 4, 1));
  MoveTo(X1, BGraph-GraphHeight);
  LineTo(X1-6, BGraph-GraphHeight);
  TextOut(8, BGraph-GraphHeight-2, FloatToStrF(Max, ffFixed, 4, 1));
  MoveTo(X1+round((Last-First)*r/4), BGraph);
  LineTo(X1+round((Last-First)*r/4), BGraph+6);
  TextOut(X1+round((Last-First)*r/4), BGraph+7,
       FloatToStrF((First+(Last-First)/4)*Scale, ffFixed, 4, 1));
  MoveTo(X1+round((Last-First)*r/2), BGraph);
  LineTo(X1+round((Last-First)*r/2), BGraph+6);
  TextOut(X1+round((Last-First)*r/2), BGraph+7,
       FloatToStrF((First+Last)*Scale/2, ffFixed, 4, 1));
  MoveTo((X1+Round((Last-First)*r*3/4)), BGraph);
  LineTo((X1+Round((Last-First)*r*3/4)), BGraph+6);
  TextOut((X1+Round((Last-First)*r*3/4)), BGraph+7,
       FloatToStrF((Last-(Last-First)/4)*Scale, ffFixed, 4, 1));
  MoveTo(X1+Round((Last-First)*r), BGraph);
  LineTo(X1+Round((Last-First)*r), BGraph+6);
  TextOut(X1+Round((Last-First)*r)-32, BGraph+7, XUnit);
  Pen.Color:=clHighlight;
  MoveTo(X1,BGraph-round(Calib.Series[First]));
  for i:=First to Last do
   LineTo(X1+Round((i-First)*r), BGraph-round(Calib.Series[i]));
 end; {with}
end; {DSAForm1.Plot}

Procedure Calibrate(Sender:TObject);
begin
 with (Sender as TForm4) do
 if pSignal<>nil then
 with pSignal^ do begin
   {Max := 0.0; Min := 0.0;
   for i:=First to Last do begin
    if Series[i] > Max then Max := Series[i];
    if Series[i] < Min then Min := Series[i]
   end;}
   Max_Min := Max - Min;
```

```
    {Calib.XUnit:=XUnit;
    Calib.YUnit:=YUnit;
    Calib.Max:=Max;
    Calib.Min:=Min;
    Calib.Size:=Size;
    Calib.Scale:=Scale;}
    Calib.First:=1;
    Calib.Last:=Size;
    for i:=1 to Size do
      Calib.Series[i]:=(Series[i]-Min)*(GraphHeight-2)/Max_Min;
  end; {with}
end; {calibrate}

procedure TForm4.FormPaint(Sender: TObject);
var
  R:TRect;
begin
  with (Sender as TForm4) do begin
    R:=Rect(0,0,ClientWidth,ClientHeight);
    Canvas.FillRect(R);
    Canvas.Pen.Color:=clBlack;
    Canvas.MoveTo(X1-2, Tgraph);
    Canvas.LineTo(X1-2, BGraph+2);
    Canvas.LineTo(ClientWidth-2, BGraph+2);
    Plot(Sender);
  end;
end;  {TForm4.FormPaint}


procedure TForm4.FormCreate(Sender: TObject);
{var
  ASignal: TSignalPtr;}
begin
  with (Sender as TForm4) do begin
    {New(pSignal);
    ASignal:=SignalList.Items[DSAForm1.PopupMenu1.PopupComponent.Tag];
    with ASignal^ do begin
      pSignal.XUnit:=XUnit;
      pSignal.YUnit:=YUnit;
      pSignal.Max:=Max;
      pSignal.Min:=Min;
      pSignal.First:=First;
      pSignal.Last:=Last;
      pSignal.Size:=Size;
      pSignal.Scale:=Scale;
      pSignal.Title:=Title;
      for i:=1 to Size do
       pSignal^.Series[i]:=Series[i];
    end;}
    pSignal:=SignalList.Items[DSAForm1.PopupMenu1.PopupComponent.Tag];
    with pSignal^ do begin
      Caption:=Title;
      TGraph:=UnitYHeight+4;
      BGraph:=ClientHeight-UnitXHeight-4;
      GraphHeight:=BGraph-TGraph;
      Calibrate(Sender);
      FormPaint(Sender);
    end;
    Show;
```

```
  end;
end;

procedure TForm4.FormResize(Sender: TObject);
begin
  with (Sender as TForm4) do begin
    TGraph:=UnitYHeight+4;
    BGraph:=ClientHeight-UnitXHeight-4;
    GraphHeight:=BGraph-TGraph;
  end;{with}
  Calibrate(Sender);
  FormPaint(Sender);
end;

procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  {Dispose(pSignal^.pForm);}
  FreeMem(pSignal^.pForm, SizeOf(TForm4));
  pSignal^.pForm:=nil;
end;

procedure TForm4.Print2Click(Sender: TObject);
begin
  with PrintDialogPlot do
    if Execute then
      Print;
end;

procedure TForm4.Bitmap2Click(Sender: TObject);
var
  ABitMap: TBitMap;
begin
  {inherited;}
  ABitMap:=TBitMap.Create;
  ABitMap.Height:=ClientHeight;
  ABitMap.Width:=ClientWidth;
  ABitMap.Canvas.CopyRect(ClientRect, Canvas, ClientRect);
  with DSAForm1.SaveDialog1 do begin
    Filter:='bitmap files (*.bmp)|*.bmp';
    DefaultExt:='bmp';
    if Execute then begin
      ABitMap.SavetoFile(FileName);
    end;
  end;
  ABitMap.Free;
end;

end.
```

UnitList.pas:

```
unit UnitList;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormList = class(TForm)
    ListBox1: TListBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
  private
    { Private declarations }
  public
    { Public declarations }
  end;


var
  FormList: TFormList;

implementation

{$R *.DFM}

end.
```

About.pas:

```
unit About;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TAboutBox = class(TForm)
    Panel1: TPanel;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
    Comments: TLabel;
    OKButton: TButton;
    Label1: TLabel;
    Label2: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  AboutBox: TAboutBox;

implementation

{$R *.DFM}

end.
```

EditSignal.pas:

```
unit EditSignal;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormEdit = class(TForm)
    EditHeader: TEdit;
    EditDataType: TEdit;
    EditTitle: TEdit;
    EditSize: TEdit;
    EditScale: TEdit;
    EditXUnit: TEdit;
    EditYUnit: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    MemoComment: TMemo;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormEdit: TFormEdit;

implementation

{$R *.DFM}

procedure TFormEdit.BitBtn1Click(Sender: TObject);
begin
  ModalResult:=mrOK;
end;
```

```
procedure TFormEdit.BitBtn2Click(Sender: TObject);
begin
  ModalResult:=mrCancel;
end;

end.
```

Unit2.pas:

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm2 = class(TForm)
    Edit3: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label4: TLabel;
    Label5: TLabel;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.DFM}

procedure TForm2.BitBtn1Click(Sender: TObject);
begin
    ModalResult:=mrOK;
end;

procedure TForm2.BitBtn2Click(Sender: TObject);
begin
    ModalResult:=mrCancel;
end;

end.
```

Unit3.pas:

```
unit Unit3;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 StdCtrls;

type
 TForm3 = class(TForm)
  Memo1: TMemo;
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form3: TForm3;

implementation

{$R *.DFM}

procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Action:=caFree;
end;

end.
```

LagPlot2D.pas:

```
unit LagPlot2D;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 UnitPlot, ComCtrls;

type
 TFormLag2D = class(TForm4)
  StatusBar1: TStatusBar;
  procedure FormCreate(Sender: TObject);
  procedure FormResize(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormPaint(Sender: TObject);
  procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
 private
 public
  nLag: Byte;
```

```
 end;

const
 X2 = 24;

var
 FormLag2D: TFormLag2D;
 i:Word;

implementation

uses Main;

{$R *.DFM}

procedure Plot(Sender:TObject);
var
  RectWidth : word;
begin
 with (Sender as TFormLag2D) do
 with Canvas, Calib, pSignal^ do begin
  {..Calibrate wrt x axis....begin}
  RectWidth:=ClientWidth-X1-24;
  r:=RectWidth/(Last-First);
  {..Calibrate wrt x axis....end}
  TextOut(4, TGraph-UnitYHeight+2, YUnit);

  MoveTo(X1, BGraph);
  LineTo(X1-6, BGraph);
  TextOut(8, BGraph-6, FloatToStrF(Min, ffFixed, 3, 1));

  MoveTo(X1, BGraph-GraphHeight div 4);
  LineTo(X1-6, BGraph-GraphHeight div 4);
  TextOut(8, BGraph-6-GraphHeight div 4, FloatToStrF(Min+(Max-Min)/4, ffFixed, 3, 1));

  MoveTo(X1, BGraph-GraphHeight div 2);
  LineTo(X1-6, BGraph-GraphHeight div 2);
  TextOut(8, BGraph-6-GraphHeight div 2, FloatToStrF(Min+(Max-Min)/2, ffFixed, 3, 1));

  MoveTo(X1, TGraph+GraphHeight div 4);
  LineTo(X1-6, TGraph+GraphHeight div 4);
  TextOut(8, TGraph+GraphHeight div 4, FloatToStrF(Max-(Max-Min)/4, ffFixed, 3, 1));

  MoveTo(X1, TGraph);
  LineTo(X1-6, TGraph);
  TextOut(8, TGraph-2, FloatToStrF(Max, ffFixed, 3, 1));

  MoveTo(X1, BGraph);
  LineTo(X1, BGraph+6);
  TextOut(X1-8, BGraph+7, FloatToStrF(Min, ffFixed, 3, 1));
  MoveTo(X1+GraphHeight div 4, BGraph);
  LineTo(X1+GraphHeight div 4, BGraph+6);
  TextOut(X1+GraphHeight div 4, BGraph+7,
       FloatToStrF(Min+(Max-Min)/4, ffFixed, 3, 1));
  MoveTo(X1+GraphHeight div 2, BGraph);
  LineTo(X1+GraphHeight div 2, BGraph+6);
  TextOut(X1+GraphHeight div 2, BGraph+7,
       FloatToStrF(Min+(Max-Min)/2, ffFixed, 3, 1));
  MoveTo(ClientWidth-X2-GraphHeight div 4, BGraph);
```

```
    LineTo(ClientWidth-X2-GraphHeight div 4, BGraph+6);
    TextOut(ClientWidth-X2-GraphHeight div 4, BGraph+7,
          FloatToStrF(Max-(Max-Min)/4, ffFixed, 3, 1));
    MoveTo(X1+GraphHeight, BGraph);
    LineTo(X1+GraphHeight, BGraph+6);
    TextOut(X1+GraphHeight, BGraph+7, YUnit);
    Pen.Color:=clRed;
    MoveTo(X1+Round(Calib.Series[First]), BGraph-round(Calib.Series[First+nLag]));
    for i:=First to Last-nLag do
      LineTo(X1+Round(Calib.Series[i]), BGraph-round(Calib.Series[i+nLag]));
  end; {with}
end; {FormLag2D.Plot}

procedure TFormLag2D.FormCreate(Sender: TObject);
begin
  with (Sender as TFormLag2D) do begin
    nLag:=Lag;
    New(pSignal);
    pSignal:=DSAForm1.pSignal;
    with pSignal^ do begin
      Caption:='2D Delay Plot of '+ Title;
      TGraph:=UnitYHeight+4;
      BGraph:=StatusBar1.Top-UnitXHeight-4;
      GraphHeight:=BGraph-TGraph;
      ClientWidth:=GraphHeight+X1+24;
      Calibrate(Sender);
      FormPaint(Sender);
    end;
    StatusBar1.Panels[0].Text:='Lag: ' + IntToStr(nLag);
    Show;
  end;
end;

procedure TFormLag2D.FormResize(Sender: TObject);
begin
  with (Sender as TForm4) do begin
    BGraph:=StatusBar1.Top-UnitXHeight-4;
    GraphHeight:=BGraph-TGraph;
  end;{with}
  Calibrate(Sender);
  FormPaint(Sender);
end;

procedure TFormLag2D.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;

procedure TFormLag2D.FormPaint(Sender: TObject);
var
  R:TRect;
begin
  with (Sender as TForm4) do begin
    R:=Rect(0,0,ClientWidth,ClientHeight);
    Canvas.FillRect(R);
    Canvas.Pen.Color:=clBlack;
    Canvas.MoveTo(X1-2, Tgraph);
    Canvas.LineTo(X1-2, BGraph+2);
    Canvas.LineTo(ClientWidth-2, BGraph+2);
```

```
   Plot(Sender);
  end;
end;

procedure TFormLag2D.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
  Max_Min:single;
begin
  inherited;
  with pSignal^{Calib} do begin
    Max_Min:=Max-Min;
    StatusBar1.Panels[1].Text:=
    'X: ' + FloatToStrF(Min+(X-X1)*Max_Min/GraphHeight, ffFixed, 7, 4) + ' ' +
    'Y: ' + FloatToStrF(Min+(BGraph-Y)*Max_Min/GraphHeight, ffFixed, 7, 4);
  end; {with}
end; {FormMouseMove}

end.
```

elayPlot2D.pas:

```
unit DelayPlot2D;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm2DDelay = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2DDelay: TForm2DDelay;

implementation

uses Main, LagPlot2D;

{$R *.DFM}

procedure TForm2DDelay.FormCreate(Sender: TObject);
begin
  ShowModal;
```

```
end;

procedure TForm2DDelay.BitBtn1Click(Sender: TObject);
begin
 Lag:=StrtoInt(Edit1.Text);
 ModalResult:=mrOK; {check if this is necessary}
 TFormLag2D.Create(Application);
end;

procedure TForm2DDelay.FormClose(Sender: TObject;
 var Action: TCloseAction);
begin
 Action:=caFree;
end;

procedure TForm2DDelay.BitBtn2Click(Sender: TObject);
begin
 ModalResult:=mrCancel;
end;

end.
```

Correlation.pas:

```
unit Correlation;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 LagPlot2D, StdCtrls,ComCtrls, UnitPlot, Math, Menus;

const
   MaxEmbed = 20;
   MaxNoOfGrids = 39;
   X2: Word = 200;
type
 TRAxis = array [2..MaxEmbed,0..MaxNoOfGrids] of single;
 TCorInt = array [2..MaxEmbed,0..MaxNoOfGrids] of longint;
 TFormCorDim = class(TForm)
   StatusBar1: TStatusBar;
   MainMenuCorDim: TMainMenu;
   File1: TMenuItem;
   Save1: TMenuItem;
   SaveAs1: TMenuItem;
   PrinttoFile1: TMenuItem;
   Print1: TMenuItem;
   Close1: TMenuItem;
   BitmapCorDim: TMenuItem;
   MetafileCorDim: TMenuItem;
   procedure FormCreate(Sender: TObject);
   procedure FormPaint(Sender: TObject);
   procedure FormClose(Sender: TObject; var Action: TCloseAction);
   procedure BitmapCorDimClick(Sender: TObject);
 private
   { Private declarations }
 public
   bPlot: Boolean;
```

```
    L, W: Byte;
    pSignal: TSignalPtr;
    C: TCorInt;
    D2: array[2..MaxEmbed] of single;
    CalibD, D, ClogC :array [2..MaxEmbed,0..MaxNoOfGrids] of single;
    BGraph, TGraph,GraphHeight, GraphWidth, GraphDimWidth: Word;
    Calib: TSeries;
    FirstX, FirstY, LastX, LastY: Word;
    rDim, r, MaxLogC, MinLogC: single;
    NoofGrids: Byte;
    RIncrement: single;
    rsqr:TRAxis;
    Rup,Rlow:real;
    p{,m},DEmbedL,DembedH:byte;
    MaxDimC, MinDimC: Single;
    Dim: Single;
    Qx: Single;
  end;

var
  {bu satýra kadar olanlar TFormCorDim'in icinde}
  FormCorDim: TFormCorDim;

implementation

uses Main, CorParams, Unit3, DimPlot;

{$R *.DFM}

var
   Code,a,i,j,k:integer;
   {RIncrement: real;
   rsqr:RAxis;
   Rup,Rlow:real;
   p,m,L,DEmbedL,DembedH:byte;}

procedure CalibrateSignal(Sender:Tobject);
var
Max_Min:single;
begin
 with (Sender as TFormCorDim) do
  if pSignal<>nil then
   with pSignal^ do begin
     Max_Min := Max - Min;
     for i:=1 to Size do
       Calib[i]:=(Series[i]-Min){*(GraphHeight-2)}/Max_Min;
   end; {with}
end; {CalibrateSignal}

procedure compute(Sender:TObject);
var
   QW, m: Byte;
   Sum, Ave, Ave1, Ave2, Avex, Q1, Q2, Q, {Qx, }DistSqr: single;
   ilk, ilk1, ilk2, radius: Word;
label Zero;
begin
 with (Sender as TFormCorDim) do
 with pSignal^ do begin
   for m:=DEmbedL to DEmbedH do begin
```

```
 for i:=1 to Size-(m-1)*L-1 do begin
  for j:=i+1 to Size-(m-1)*L do begin
   L:=Round(W/(m-1));
   if L=0 then L:=1;
   DistSqr:=0;
   p:=0;   {a new vector couple}
   for a:=0 to m-1 do begin
    DistSqr:=DistSqr+sqr(Calib[i+a*L]-Calib[j+a*L]);
   end;
   if (DistSqr<rSqr[m,p]) and (p>0) then begin
    Dec(p);
    while (DistSqr<rSqr[m,p]) and (p>0) do Dec(p);
   end
   else
    while (DistSqr>rSqr[m,p]) and (p<NoofGrids) do Inc(p);
   if (DistSqr>rSqr[m,p]) then Inc(p);
   {.....UpdateCorInt(m,DistSqr).....}
   for radius:=p to NoOfGrids do
    inc(C[m,radius]);
   {.....end UpdateCorInt.....}
  end {for}
 end; {for}
end; {for}
{.....Find max and min....}
MaxLogC := 0.0; MinLogC := 100.0;
for i:=DembedL to DembedH do
 for j:=0 to NoOfGrids do begin
  if C[i,j] > 0 then ClogC[i,j]:=log10(C[i,j])
  else ClogC[i,j]:=-1;
  if ClogC[i,j] > MaxLogC then MaxLogC := ClogC[i,j];
  if ClogC[i,j] < MinLogC then MinLogC := ClogC[i,j]
 end;
{.....begin Dimension Assignment.....}
for i:=DEmbedL to DEmbedH do
 for j:=1 to NoOfGrids do begin
  if (ClogC[i,j]>0) and (ClogC[i,j-1]>0) then
   D[i,j]:=(ClogC[i,j]-ClogC[i,j-1])/RIncrement
  else D[i,j]:=0;
 end; {for}
{.....end Dimension Assignment.....}
{....Linearizer.....}
QW:=(NoofGrids{+1}) div 5;
for m:=DembedL to DEmbedH do begin
 Ave1:=0; Ave2:=0; Q1:=0.5; Q2:=0.6; ilk1:=NoofGrids; ilk2:=NoofGrids;
 for ilk:=1 to NoofGrids-QW+1 do begin
  Sum:=0;
  for i:=0 to QW-1 do begin
   if D[m,ilk+i]<0.8 then begin
    Goto Zero;
   end;
   Sum:=Sum+D[m,ilk+i];
  end;
  Ave:=Sum/QW;
  Q:=0;
  for i:=0 to QW-1 do
   Q:=Q+Sqr(D[m,ilk+i]-Ave);
  Q:=Sqrt(Q)/QW;
  if Q<Q2 then
   if Q<Q1 then begin
```

```
        Q2:=Q1; ilk2:=ilk1; Ave2:=Ave1;
        Q1:=Q; ilk1:=ilk; Ave1:=Ave;
       end
       else
        Q2:=Q; ilk2:=ilk; Ave2:=Ave;
Zero: ;
    end; {for}
    D2[m]:=Ave1;
  end; {for}
  Dim:=0; Qx:=1.0;
  for ilk:=DEmbedL to DEmbedH-4 do begin
    Sum:=0;
    for i:=0 to 3 do begin
       Sum:=Sum+D2[ilk+i];
    end;
    Ave:=Sum/4;
    Q:=0;
    for i:=0 to 3 do
     Q:=Q+Sqr(D2[ilk+i]-Ave);
    Q:=Sqrt(Q)/4;
    if Q<Qx then begin
     Qx:=Q; Avex:=Ave;
    end;
    Dim:=Avex;
    if exp((ilk1*RIncrement+Rlow)*ln(10))<0.2 then
     Qx:=Qx+exp((ilk1*RIncrement+Rlow)*ln(10));
    {StatusBar1.Panels[1].Text:='D2 = ' + FloatToStrF(Dim, ffFixed, 4, 2) +
                        ' +/- ' + FloatToStrF(Qx, ffFixed, 4, 2);}
  end;
  {....end Linearizer.....}
 end; {with}
end; {Compute}

procedure CalibratePlot(Sender:Tobject);
var
 Max_Min: Single;
begin
 with (Sender as TFormCorDim), pSignal^ do begin
 Max_Min := MaxLogC - MinLogC;
 for i:=DembedL to DembedH do
  for j:=0 to NoOfGrids do
   ClogC[i,j]:=(ClogC[i,j]-MinlogC)*(GraphHeight)/Max_Min;
 end; {with}
end; {CalibratePlot}

procedure CalibrateDimPlot(Sender:Tobject);
var
 Max_Min: Single;
begin
 with (Sender as TFormCorDim) do begin
  {.....Find max and min....}
  MaxDimC := 0.0; MinDimC := 10.0;
  for i:=DembedL to DembedH do
   for j:=0 to NoOfGrids do begin
    if D[i,j] > MaxDimC then MaxDimC := D[i,j];
    if D[i,j] < MinDimC then MinDimC := D[i,j]
   end;
  {....end find max and min....}
  Max_Min := MaxDimC - MinDimC;
```

```
  for i:=DembedL to DembedH do
   for j:=0 to NoOfGrids do
    CalibD[i,j]:=(D[i,j]-MinDimC)*(GraphHeight)/Max_Min;
 end; {with}
end; {CalibrateDimPlot}

procedure Plot(Sender:TObject);
begin
 with (Sender as TFormCorDim) do
 with Canvas, pSignal^ do begin
  for i:=0 to 4 do begin
   MoveTo(X1, BGraph-Round(GraphHeight*i/4));
   LineTo(X1-6, BGraph-Round(GraphHeight*i/4));
   TextOut(8, BGraph-Round((GraphHeight*i/4)-MinlogC),
        FloatToStrF(MinlogC+(MaxlogC-MinlogC)*i/4, ffFixed, 3, 1));
   MoveTo(X2, BGraph-Round(GraphHeight*i/4));
   LineTo(X2-6, BGraph-Round(GraphHeight*i/4));
   TextOut(X2-32, BGraph-Round((GraphHeight*i/4)-MinDimC),
        FloatToStrF(MinDimC+(MaxDimC-MinDimC)*i/4, ffFixed, 4, 2));
  end;
  for i:=0 to 3 do begin
   MoveTo(X1+Round(GraphWidth*i/4), BGraph);
   LineTo(X1+Round(GraphWidth*i/4), BGraph+6);
   TextOut(X1+Round(GraphWidth*i/4), BGraph+7,
        FloatToStrF(Rlow+(Rup-Rlow)*i/4, ffFixed, 3, 1));
   MoveTo(X2+Round(GraphDimWidth*i/4), BGraph);
   LineTo(X2+Round(GraphDimWidth*i/4), BGraph+6);
   TextOut(X2+Round(GraphDimWidth*i/4), BGraph+7,
        FloatToStrF(Rlow+(Rup-Rlow)*i/4, ffFixed, 3, 1));
  end;
  MoveTo(X1+GraphWidth, BGraph);
  LineTo(X1+GraphWidth, BGraph+6);
  TextOut(X1+GraphWidth-32, BGraph+7, 'log(r)');
  TextOut(4, 6, 'log(C(r))');
  MoveTo(X2+GraphDimWidth, BGraph);
  LineTo(X2+GraphDimWidth, BGraph+6);
  TextOut(Clientwidth-32, BGraph+7, 'log(r)');
  TextOut(4+(24+X1+GraphWidth), 6, 'D2(r)');
  Pen.Color:=clRed;
  for i:=DembedL to DembedH do begin
   MoveTo(X1,BGraph-Round(ClogC[i,0]));
   for j:=0 to NoOfGrids do
    LineTo(X1+Round(j*r), BGraph-Round(ClogC[i,j]));
  end;
  for i:=DembedL to DembedH do begin
   MoveTo(X2,BGraph-Round(CalibD[i,0]));
   for j:=0 to NoOfGrids do
    LineTo(X2+Round(j*rDim), BGraph-Round(CalibD[i,j]));
  end;
 end; {with}
end; {Plot}

procedure TFormCorDim.FormPaint(Sender: TObject);
var
 Rec:TRect;
begin
 with (Sender as TFormCorDim) do begin
  Rec:=Rect(0,0,ClientWidth,ClientHeight);
  Canvas.FillRect(Rec);
```

```
    Canvas.Pen.Color:=clBlack;
    Canvas.MoveTo(X1-2, Tgraph);
    Canvas.LineTo(X1-2, BGraph+2);
    Canvas.LineTo(X1+GraphWidth, BGraph+2);
    Canvas.MoveTo(X2-2, Tgraph);
    Canvas.LineTo(X2-2, BGraph+2);
    Canvas.LineTo(X2+GraphDimWidth{ClientWidth-2}, BGraph+2);
    if bPlot then Plot(Sender);
  end;
end;

procedure Display(Sender:TObject);
var
  strXi:string;
begin
 with (Sender as TFormCorDim) do begin
  Form3:=TForm3.Create(Application);
  for i:=DembedL to DembedH do
   with Form3.Memo1.Lines, (Sender as TFormCOrDim)  do begin
    Add('m=' + InttoStr(i) + ': ');
    Add('D2=' + FloattoStrF(D2[i], ffFixed, 4, 2));
    for j:=1 to NoOfGrids do begin
     Add(FloattoStrF(D[i,j], ffFixed, 4, 2) + ',')
     {if (ClogC[i,j]>0) and (ClogC[i,j-1]>0)
     then Add(FloattoStrF((log10(C[i,j])-log10(C[i,j-1]))
                  /(RIncrement), ffFixed, 4, 2) + ',')
     else Add('Inf,');}
    end;
    Add('D2=' + FloatToStrF(Dim, ffFixed, 4, 2) +
      ' +/- ' + FloatToStrF(Qx, ffFixed, 4, 2));
   end; {with}
  Form3.Show;
  {Form3.Free;}
 end; {with Sender do}
end; {display}

procedure TFormCorDim.FormCreate(Sender: TObject);
begin
  bPlot:=False;
  Show;
  with FormCorParams do begin
    StatusBar1.Panels[0].Text:='Lag:'+PanelLag.Caption+
                        ', Dmin:'+PanelMinEmb.Caption +
                        ', Dmax:'+PanelMaxEmb.Caption +
                        ', Rl:'+EditRLow.Text +
                        ', Ru:'+EditRUp.Text +
                        ', W:'+EditW.Text;

    {L:=StrtoInt(PanelLag.Caption);}
    W:=StrtoInt(EditW.Text);
    DEmbedL:=StrtoInt(PanelMinEmb.Caption);
    DEmbedH:=StrtoInt(PanelMaxEmb.Caption);
    Val(EditRlow.Text, Rlow, Code);
    Val(EditRup.Text, Rup, Code);
    Val(EditZoom.Text, NoofGrids, Code);
    Dec(NoofGrids);
  end; {with}
  TGraph:=UnitYHeight+4;
  BGraph:=StatusBar1.Top-UnitXHeight-4;
```

```
  GraphHeight:=BGraph-TGraph;
  GraphWidth:=GraphHeight;
  GraphDimWidth:=GraphWidth+64;
  r:=GraphWidth/NoofGrids;
  rDim:=GraphDimWidth/NoofGrids;
  ClientWidth:=2*(X1+24)+GraphWidth+GraphDimWidth;
  X2:=ClientWidth-GraphDimWidth-24;{2*X1+GraphWidth+24}
  RIncrement:=(Rup-Rlow)/NoOfGrids;
  for j:=DembedL to DEmbedH do
   for i:=0 to NoOfGrids do
    rsqr[j,i]:=j*sqr(exp((i*RIncrement+Rlow)*ln(10))); {gridding log(r) axis}
  pSignal:=DSAForm1.pSignal;
  CalibrateSignal(Sender);
  Compute(Sender);
  CalibratePlot(Sender);
  CalibrateDimPlot(Sender);
  bPlot:=True;
  Plot(Sender);
  Display(Sender);
end;

procedure TFormCorDim.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action:=caFree;
end;

procedure TFormCorDim.BitmapCorDimClick(Sender: TObject);
var
  ABitMap: TBitMap;
begin
  {inherited;}
  ABitMap:=TBitMap.Create;
  try
    ABitMap.Height:=ClientHeight;
    ABitMap.Width:=ClientWidth;
    ABitMap.Canvas.CopyRect(ClientRect, Canvas, ClientRect);
    with DSAForm1.SaveDialog1 do begin
      Filter:='bitmap files (*.bmp)|*.bmp';
      DefaultExt:='bmp';
      if Execute then begin
        ABitMap.SavetoFile(FileName);
      end;
    end;
  finally
   ABitMap.Free;
  end;
end;

end.
```

Corparams.pas:

```
unit Corparams;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls;

type
  TFormCorParams = class(TForm)
    ScrollBar1: TScrollBar;
    PanelLag: TPanel;
    PanelMinEmb: TPanel;
    ScrollBar2: TScrollBar;
    PanelMaxEmb: TPanel;
    ScrollBar3: TScrollBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    EditRLow: TEdit;
    EditRUp: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    EditZoom: TEdit;
    Label13: TLabel;
    EditW: TEdit;
    procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
      var ScrollPos: Integer);
    procedure ScrollBar2Scroll(Sender: TObject; ScrollCode: TScrollCode;
      var ScrollPos: Integer);
    procedure ScrollBar3Scroll(Sender: TObject; ScrollCode: TScrollCode;
      var ScrollPos: Integer);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormCorParams: TFormCorParams;

implementation

{$R *.DFM}
```

```
procedure TFormCorParams.ScrollBar1Scroll(Sender: TObject;
  ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  PanelLag.Caption:=IntToStr(ScrollPos);
  EditW.Text:=PanelLag.Caption;
end;

procedure TFormCorParams.ScrollBar2Scroll(Sender: TObject;
  ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  PanelMinEmb.Caption:=IntToStr(ScrollPos);
end;

procedure TFormCorParams.ScrollBar3Scroll(Sender: TObject;
  ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  PanelMaxEmb.Caption:=IntToStr(ScrollPos);
end;

procedure TFormCorParams.BitBtn1Click(Sender: TObject);
var
  Rl, Ru: Single;
  Code: integer;
  Mess: string;
  i:Byte;
begin
  Mess:='Enter a non positive number for ';
  i:=0;
  Val(EditRlow.Text, Rl, Code);
  if (Rl>0) or (Code<>0) then begin
    Mess:=Mess + 'Lower Radius,';
    Inc(i);
  end;
  Val(EditRup.Text, Ru, Code);
  if (Ru>0) or (Code<>0) then begin
    Mess:=Mess + 'Upper Radius,';
    Inc(i);
  end;
  if i=0 then ModalResult:=mrOK
  else  begin
    ModalResult:=mrNone;
    Delete(Mess, Length(Mess), 1);
    MessageDlg(Mess, mtError, [mbOK], 0);
  end;
end;

procedure TFormCorParams.BitBtn2Click(Sender: TObject);
begin
  ModalResult:=mrCancel;
end;

end.
```

REFERENCES

1. Başar, E., "Toward a physical approach to integrative physiology. Brain dynamics and physical causality," *Am J Physiol*, 245(4), R510-R533, 1989.
2. Başar, E., Röschke, J., "Synergetics of neuronal populations. A survey on experiments," in: Başar. E., Flohr, H., Haken, H., Mandell, A.J. (Eds), *Synergetics of the Brain*, Springer, Berlin Heidelberg New York, pp 199-200 (Springer series in synergetics, Vol.23), 1983.
3. Babloyantz, A., Nicolis, C., Salazar, M., "Evidence of chaotic dynamics," *Phys Lett*, A:152-156, 1985.
4. Grassberger, P., Procaccia, I., "Measuring the strangeness of strange attractors," *Physica*, 9D:189, 1983.
5. Eckmann, J.P., "Ergodic theory of chaos and strange attractors", *Rev of Mod Phys*, Vol.57, No.3, 617-656, 1985.
6. Takens, F., "Detecting strange attractors in turbulence", *Lecture Notes in Mathematics*, Springer Berlin, Vol. 898, p.366, 1981.
7. Kennel, M. B., Brown, R. & Abarbanel, H. D. I., "Determining embedding dimension for phase space reconstruction using a geometrical construction," *Physical Review*, A, 45, 3403-3411, 1992.
8. Casdagli, M., Eubank, S., Farmer, J. D., Gibson, J., "State space reconstruction in the presence of noise," *Physica*, D 51, 1991.
9. Martinerie, J. M., Albano, A. M., Mees, A. I., Rapp, P. E., "Mutual information, strange attractors, and the optimal estimation of dimension," *Phys Rev*, A 45 7058, 1992.
10. Abarbnel, H. D. I., Brown, R., Sidorowich, J. J., Tsimring, L. S., "The analysis of observed chaotic data in physical systems," *Rev of Modern Phys*, 65, No.4, 1331-1392, 1993.
11. Shaw, R., *The Dripping Faucet As a Model Dynamical System*, Aerial Press, Santa Cruz, 1984.
12. Coburn, K. L., Moreno, M. A., "Facts and artifacts in brain electrical activity mapping," *Brain Topology*, 1, 37-45, 1988.

13. Pritchard, W. S., Duke, D. W., "Measuring chaos in the brain: A tutorial review of nonlinear dynamical EEG analysis", *International Journal of Neuroscience*, Vol.67, 31-80, 1992.

14. Pritchard, W. S., Duke, D. W., "Measuring chaos in the brain: A tutorial review of EEG dimension estimation," *Brain and Cognition*, Vol.27, 353-397, 1995.

15. Lo, P. C., Principie, J. C., "The effects of filtering on the EEG correlation dimension: Experimental results," *Proc eleventh annual IEEE Engineering in Medicine and Biology Society Conference*, pp, 638-639, 1989.

16. Mitschke, F., "Acausal filters for chaotic signals," *Physical Review*, A, 41, 1169-1171, 1990.

17. Cook, E. W., Miller, G. A., "Digital filtering: Background and tutorial for psycho-physiologists," *Psychophysiologiy*, 29, 350-367, 1992.

18. Theiler, J., "Estimating fractal dimension," *Journal of the Optical Society of America*, A, 7, 1055-1073 ,1990.

19. Albano, A. M., Muench, J., Schwartz, C., Mees, A. I., Rapp, P. E., "Singular value decomposition and the Grassberger Procaccia algorithm," *Physical Review*, A, 38, 1988.

20. Frank, G. W., Lookman, T., Nerenberg, M. A. H., Essex, C., Lemieux, J., Blume, W., "Chaotic time series analysis of epileptic seizures," *Physica*, D 46, 427-438, 1990.

21. Babloyantz, A., Dextexhe, A., "Low dimensional chaos in an instance of epilepsy," *Proc Natl Acad Sci, USA*, Vol 83, pp 3513-3517, 1986.

22. Albano, A. M., Mees, A. I., de Guzman, G. C., Rapp, P. E., "Data requirements for reliable estimation of correlation dimensions," in Degn, H., Holden, A. V., Olsen, L. F. (eds), *Chaos in Biological Systems*, New York: Plenum, pp 207-220, 1987.

23. Dvorak, I., Siska, J. "On some problems encountered in calculating the correlation dimensionof EEG," *Trieste: International Centre for Theoretical Physics*, 1986.

24. Layne, S. P., Mayer-Kress, G., Holzfuss, J., "Problems associated with dimensional analysis of electroencephalogram data," In Mayer-Kress, G. (Eds), *Dimensions and Entropies in Chaotic Systems*, New York: Springer-Verlag pp 246-256, 1986.

25. Mayer-Kress, G., Layne, S. P., "Dimensionality of the human electroencephalogram," in Koslow, S. H. (Eds.), *Perspectives in Biological Dynamics and Theoretical Medicine*, New York: New York Academy of Sciences, pp 62-87, 1987.

26. Nan, X., Jinghua, X., "The fractal dimension of EEG as a physical measure of conscious human brain activities," *Bulletin of Mathematical Biology*, 50, 559-565, 1988.

27. Rapp, P. E., "Chaos in neurosciences: Cautionary tales from the frontier," *Biologist*, Vol.40, 89-94, 1993.

28. Başar, E., *Chaotic Dynamics and Resonance Phenomena in Brain Function.* New York: Springer - Verlag pp1-30, 1990.