Fractal-Based Models for Internet Traffic and their Application to Secure Data Transmission

by

Rashiq Rafiq Marie

Doctoral Thesis

Submitted in partial fulfillment of the requirements

for the award of PhD

Research School of Informatics/Department of Computer Science
Loughborough University
England

October 2006

This work is dedicated to

My Father

and

The memory of my Mother

Abstract

This thesis studies the application of fractal geometry to the application of covert communications systems. This involves the process of hiding information in background noise; the information being encrypted or otherwise. Models and methods are considered with regard to two communications systems:

(i) Wireless communications;

(ii) Internet communications.

In practice, of course, communication through the Internet cannot be disassociated from wireless communications as Internet traffic is 'piped' through a network that can include wireless communications (e.g. satellite telecommunications). However, in terms of developing models and methods for covert communications in general, points (i) and (ii) above require different approaches and access to different technologies. With regard to (i) above, we develop two methods based on fractal modulation and multi-fractal modulation. With regard to (ii), we implement a practical method and associated software for covert transmission of file attachments based on an analysis of Internet traffic noise. In both cases, however, two fractal models are considered; the first is the standard Random Scaling Fractal model and the second is a generalisation of this model that incorporates a greater range of spectral properties than the first - a Generalised Random Scaling Fractal Model.

The use of encryption techniques for securing information interchange is of course well known and has a long and well established history. However, there is one crucial point that is often not stressed as much as it perhaps should be;

this is, that the act of encrypting information raises a 'flag' to an interceptor of the potential value of the information that is being sent, otherwise why should the sender want to encrypt the information in the first place. In other words, encrypted information is not covert as long as an interceptor realises that the information is encrypted. This consideration leads directly to an investigation of methods of hiding or camouflaging encrypted data before it is transmitted. These include watermarking appropriate data files (e.g. audio and image files) with the encrypted information and/or appropriate keys and embedding data in noise whose characteristics are compatible with the transmission environment, e.g. radio noise and/or Internet traffic noise. This thesis investigates both examples and uses both for the development of the MATLAB prototype software system that is described and provided on the accompanying CD.

Acknowledgments

I thank God for giving me the ability, health and knowledge to complete this project.

I deeply appreciate Professor J.M. Blackledge for spending alot of his precious time offering me guidance and supervision on my work.

I would like to express my gratitude to Dr. Helmut Bez and Dr. Sekharjit Datta for their help throughout this project. Thanks to Mr. Martin Hamilton for providing me data.

Many sincere thanks must go to my father for his support and encouragement through my studying and also to my brothers and sisters.

Last, but by no means least, I wish to thank my wife and my children for their unconditional support.

Contents

List of Figures

List of Tables

Glossary of Terms

ACF	Autocorrelation Function.
AM	Amplitude Modulation.
ASCII	American Standard Code Interchange.
DC	Direct Current.
DFT	Discrete Fourier Transform.
fBM	fractional Brownian Motion.
FT	Fourier Transform.
FFT	Fast Fourier Transform.
FM	Frequency Modulation.
FIR	Finite Impulse Response.
HTTP	HyperText Transport Protocol.
HR	Halls of Residence.
IDFT	Inverse Discrete Fourier Transform.
IFS	Iterated Function System.
IP	Internet Protocol.
IRF	Impulse Response Function.
LANs	Local-Area Networks.
LRD	Long-Range Dependence.
LU	Loughborough University.
PDF	Probability Density Function.
PSDF	Power Spectrum Density Function.
PSM	Power Spectrum Method.

RSF	Random Scaling Fractal.
SFD	*Stochastic Fractional Differentiation.*
SNR	Signal to Noise Ratio.
TCP	Transmission Control Protocol.
WAN	Wide Area Network.
WGN	White Gaussian Noise.
WWW	World Wide Web.

Chapter 1

Introduction

1.1 The Information Society

Our digital age has brought about a number of changes in society but perhaps the most profound is the impact it has had upon basic human activities such as decision making, information processing and communications which are all supported by computer devices.

The information revolution can be compared in its impact to that of the industrial revolution of the nineteenth century. The value of information has changed in that both the constructive and destructive powers of information flow are very different compared with the situation less than a hundred years ago. In the commercial sector, 'know how' contributes considerably to company market value because information provides a primary competitive advantage. Clearly, critical information is vital for national security and financial organizations[1].

Accurate knowledge of public opinion allows political leaders to react rapidly in their policies or programs. Thus, political power is now critically dependent on the flow of information.

The speed of information flow has developed considerably over the past ten years. As with high speed vehicles, skilled control becomes increasingly important as information flow increases in the rapidity of its exchange; the web being an environment in which information flow is now very difficult to control effectively. Access to the global market is a key incentive in society. New economic incentives require new services based on electronic and mobile transactions and both individuals and industries require involvement under conditions that are not only financially advantageous but secure. The continuous update of security infrastructures is absolutely necessary to encourage the further development of the new global economy. Therefore, knowledge management can be considered as a top priority issue in the twenty-first century. Proper information security can ensure the healthy evolution of society and avoid computer terrorism and crime.

1.2 Information Security

Modern information security manifests itself in many ways according to the situation and requirement. It deals with such concepts as confidentiality, data integrity, access control, identification, authentication and authorization. Practical applications, closely related to information security, are private messaging, financial transactions, online services and many others.

A century ago, one could say that cryptography is the science (or art) of secret

writing and reading, which has grown from semiotics, the study of signs and sign-using behavior[1], rather than mathematics. The word is derived from the Greek *kryptos*, meaning 'hidden', and *graphein* meaning 'to write'.

Today cryptography is, essentially, the study of mathematical and computational techniques underlying information security. Cryptography is closely connected to the disciplines of cryptanalysis and cryptology. In simple terms, cryptanalysis is the art of breaking cryptosystems, i.e. retrieving the original message without knowing the proper key or forging an electronic signature. Cryptology is the mathematics, such a number theory, which underpin cryptography and cryptanalysis.

Cryptography has always been shrouded in secrecy itself and remains one of the most secret sciences in the world. Professional cryptographers working for intelligence services and commercial organizations have been limited in their publications. As a result, the freely available literature never fully reflects the state of the art. Nations vary in their reticence: whereas the United States released quite generous information on the situation in the Second World War, the Soviet Union locked itself in silence; but all the time scientists from both countries kept abreast of each other. Claude Shannon, an American electrical engineer, formulated the major criteria and fundamentals of cryptographic techniques in his secret report of 1945 that was declassified in 1949 and published as *Communication theory of secrecy systems.* Unlike Shannon, Vladimir Alexandrovich Kotelnikov, a Russian electrical engineer, is not very well known as a founder of modern cryptography. However, in 1941 Kotelnikov clearly for-

[1] Although the word was used in this sense in the 17th century by the English philosopher John Locke, the idea of semiotics as an interdisciplinary mode for examining phenomena in different fields emerged only in the late 19th and early 20th centuries.

mulated the requirements to a perfect encryption system and mathematically proved its cryptographic resistance. This work together with the Kotelnikov theorem (the sampling theorem) became the foundation of Russian cryptography and provided secure communication during the World War II in 1942-1945.

Since that time, cryptography has absorbed ideas from complexity theory, number theory, group theory, combinatory logic and modern computer science and grown from basic symmetric ciphers to PKI infrastructure and complex cryptosystems. In particular, in 1976 asymmetric public keys were first proposed in a revolutionary article by Whitfield Diffie and Martin Hellman.

1.3 Innovations in Cryptology trough Synergy

In making innovations, we typically take existing models from one science and transplant them into a new subject area. For example, it is quite natural to apply models from nonlinear dynamics (chaos theory) for the purpose of encryption or to apply the principles of fractal geometry to hide information in noise[2]. If we are successful, a new algorithm will emerge. This is a practical benefit of a cross-disciplinary approach. Unlike most new disciplines that appear at the *edge* of existing sciences when a model or a technique from one subject area is applied in another, synergetics studies the *common fundamentals* of these sciences and extends the global collection of ideas and methods.

The term *synergetics* (from Greek: *synergeia* - working together, cooperation) was introduced by the German physicist Haken in the beginning of 1970's. Haken's synergetics treats systems in which cooperation among subsystems creates organized structure on macroscopic scales.

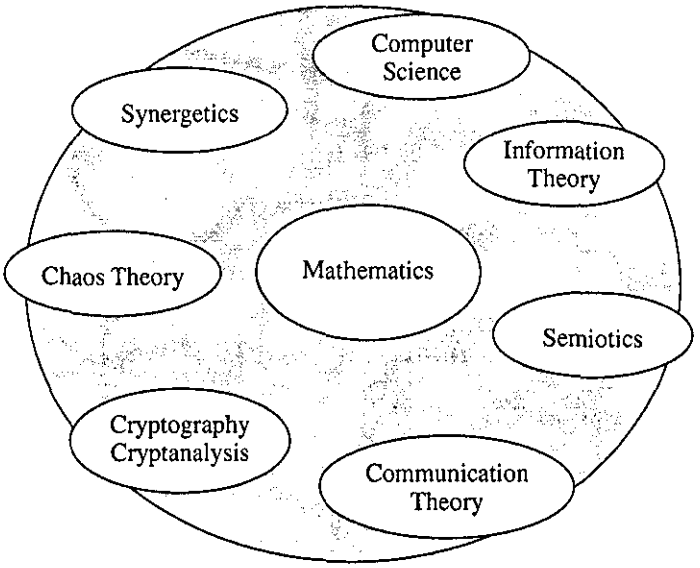

Figure 1.1: The cross-disciplinary approach to cryptology.

Synergetics aims to understand common laws driving processes in different nonlinear systems with a feedback. Examples of problems studied by synergetics are bifurcations, phase transitions in physics, nonlinear oscillations in electrical circuits, population dynamics. From a certain view point, a similar definition is given by the American scientist R. Buckminster Fuller: '*Synergy means behavior of whole systems unpredicted by the behavior of their parts taken separately*'. One can see this property in cryptographic systems, whose strength depends on the integrity of several mathematical constructions.

Today, it would not be easy to find a discipline remaining untouched by synergetics deterministic chaos and fractal geometry. These new sciences have transfused all the scope of human knowledge: not only mathematics, physics, biology, economics, digital imaging, simulation sciences but also many human

studies such as history and sociology. Cryptology, of course, is not an exception - a number of scientists have evaluated encryption techniques based on non-linear dynamic systems and fractals and this thesis represents a continuation of this work.

1.4 Fractal and Chaos in Cryptology

The idea that many simple nonlinear deterministic systems can behave in an apparently unpredictable and chaotic manner was first observed by the great French mathematician Henri Poincaré. Other early pioneering works in the field of chaotic dynamics are to be found in the mathematical literature by such luminaries as Birkhoff, Cartwright, Littlewood, Levinson, Smale, Kolmogorov and his students.

The key feature of chaotic behavior in different systems is mainly related to the high sensitivity to initial conditions due to exponential divergence of all trajectories lying on the attracting structure which is normally bounded in an appropriate phase space. In the 1960's Edward Lorenz, an American meteorologist, discovered a stable chaotic attractor and predicted that: *"...it may happen that small differences in the initial conditions produce very great ones in the final phenomena. A small error in the former will produce an enormous error in the future. Prediction becomes impossible..."*

The practical value of chaos theory is that it attempts to describe mathematically the extreme complexity of the real world such as the process of Brownian motion in physics, weather changes in meteorology, population fluctuations in biology and the geometry of nature for example[2].

On the other hand, chaos theory has caused a major paradigm shift in the philosophy of the universe, providing the first 'scientific' explanation of the coexistence of such concepts as law and disorder, determinism and unpredictability.

Mathematical or deterministic chaos is a dynamic system, characterized with a 'complex' and 'unpredictable' behavior. Intuitively, this property suits the requirements of a digital encryption system — on the one hand computer-based cryptosystems are deterministic; on the other, they must be cryptographically unpredictable. Practically, the last property implies that given certain information on the ciphertext and the plaintext (the message), a cryptanalyst should not be able to predict the cryptographic transformation and recover the key or the message[3].

Since 2000, the potential of chaos- and fractal-based communication, especially spread spectrum modulation, has been recognized worldwide. Many authors have described chaotic modulations and suggested electronic implementations. Again, the emphasis here is put on *information coding* rather then *digital encryption* and *information hiding*, which is the subject of this research.

1.5 Original Contribution

This thesis discusses the theoretical background and practical implementations of chaos-based cryptosystems with a focus on fractal-based information hiding. The following results are considered the most useful and unique and contribute to the primary aspects of the authors original contributions:

- Re-appraisal of the fractal modulation technique originally developed by J M Blackledge and its extension of the technique to multi-fractal modulation[4],[5].

- Analysis of Internet traffic noise in terms of a new generalised random scaling fractal model.

- The development of a new method for transmitting data over the Internet that is based on:

  i) Application of the new model for Internet traffic noise to segment a single file into many files whose sizes and times of transmission are compatible with the Internet traffic that has been sampled at a given time;

  ii) The application of symmetric or asymmetric encryption algorithm for encrypting plaintext before submission of a file for Internet transmission;

  iii) The use of a new watermarking method (based on chirp coding) for covertly embedding the parameter settings required for a receiver to reconstruct the information;

  iv) The development of a prototype software system (based on MATLAB)

- An evaluation of the system developed based on (i)-(iv) above.

- Conclusions with regard to the use of Internet traffic noise for covert transmission.

1.6 References

[1] Ghonaimy M.A., El-Hadidi M. T. and Aslan H. K,"Security in the Information Society: Visions and Perspectives," *IFIP TC11 17th International Conference on Information Security (SEC2002)*, Cairo, Egypt, May 7-9, 2002.

[2] Turner M.J., Blackledge J.M. and Andrews P.R., *Fractal Geometry in Digital Imaging*, Academic Press Ltd., UK, 1998.

[3] Ptitsyn N.V., Blackledge J.M. and Chernenky V.M., "Deterministic Chaos in Digital Cryptography", *Proc. of IMA Conference on Fractal Geometry*, Horwood Publishing, London, pp. 189-222, 2002.

[4] Blackledge J.M., Foxon B. and Mikhailov S., "Fractal modulation for digital communications systems," *Proc. of the IEEE Military Communications Conference MILCOM'98*, Boston, USA, Oct. 1998.

[5] Blackledge J.M., *Digital Signal Processing: Mathematical and Computation Methods, Software Development and Applications*, Horwood Publishing Limited, London, 2nd Edition, 2006.

Chapter 2

Background to Fractal Geometry

2.1 Introduction

Euclidean or classical geometry consists of describing physical objects using lines, circles, ellipses, etc. This type of geometry is appropriate for describing man-made objects; however, the patterns found in nature are significantly more complex and this complexity can be best modelled by fractal geometry [1]. There are a number of important differences between Euclidean and Fractal geometry. For example, it is not possible to draw a tangent to a fractal curve because a fractal curve is not smooth. Further, although fractals may be either continuous or fragmented, they are not differentiable [2] at least in the ordinary sense of the term.

2.2 Definition of a Fractal

The term *fractal* was coined by the Polish mathematician B. Mandelbrot in 1975, a pioneer in the field of fractal geometry and some times referred to as the father of fractal geometry. However, many of the mathematical principles started appearing much earlier than this, even in the Nineteenth Century, with the work of mathematicians such George Cantor, Karl Weierstrass, Giuseppe Peano, and others.

Moreover, Mandelbrot studied with the French mathematician, Paul Lévy, in the late 1930s, who was the first to considered the nature of self-affine random walks. In turn, Levy's ideas came from the work of the English civil engineer, E. Hurst, who found that Einstein's model for Brownian motion (first observed by Robert Brown in the 1860s) was relatively accurate when applied to real world problems such as the apparent random behavior of the Nile delta annual flood.

Mandelbrot claimed that fractal geometry would provide a useful tool to explain a variety of naturally occurring phenomena. Fractal objects can be found everywhere in nature such as coastlines, ferns trees, foods, clouds, mountains and bacteria and a range of electronic signals. The word *fractal*, that Mandelbrot introduced, comes from the Latin adjective *fractus* which is derived from the Latin verb *frangere*, which means 'to break' or to create irregular fragments. In addition to 'fragmented'*fractus* can also mean 'irregular', both meanings being preserved in *fragment.* Since fractals are both fragmented and irregular this word is perfectly suited and Mandelbrot originally defined the formal mathematical definition of a fractal as follows: *a fractal is a set*

for which the Hausdorff-Besicovich dimension strictly exceeds the topological dimension, i.e. exceeds an integer dimension.

In simple terms, fractals are geometrical shapes that unlike Euclidean objects are irregular all over, with the same degree of irregularity on all scales. They cannot be described by terms from classical geometry. However, relatively simple iterative rules can be used to describe these shapes in ways for which conventional techniques are non-applicable.

2.3 Background to Fractal Geometry

Fractal geometry is a consequence of the computing revolution and its development has gone hand-in-hand with advances in digital data processing and computer graphics. However, the principles of fractal geometry have been studied for many years and began with the French School of mathematics in the late Nineteenth Century. This included French mathematicians such as Jules Henri Poincaré who was one of the first mathematicians in history to conceive the idea that a dynamical system (and a Newtonian one at that) could not be predicted deterministically. He was in effect describing the principles of chaos as a result of re-evaluating an award winning piece of work he had undertaken early in his career on the orbits of multiple interacting bodies. Chaos is the study of functions which exhibit patterns or fields that are similar at different scales (see, for example, [3]). Today algorithms abound that are used to generate fractals and chaotic fields that ideally depend on:

(i) an understanding of the physical (typically a nonlinear) system; (ii) a clear and concise mathematical definition of the field properties.

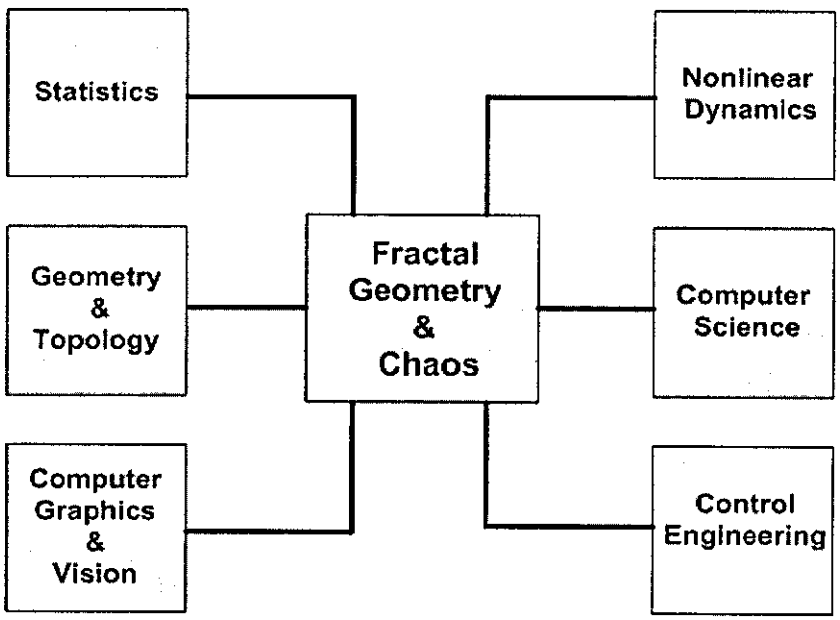

Figure 2.1: The variety disciplines related to fractal geometry and chaos.

Further, there are a range of applications based on fractals and chaos that have been developed including time series analysis, speech processing, data compression, segmentation, terrain modeling, image synthesis, music, financial forecasting, biomedical engineering, digital communications and information technology security. The development of fractal geometry has been undertaken through many different fields of study and it now has an important contribution to make to these fields (see Figure 2.1).

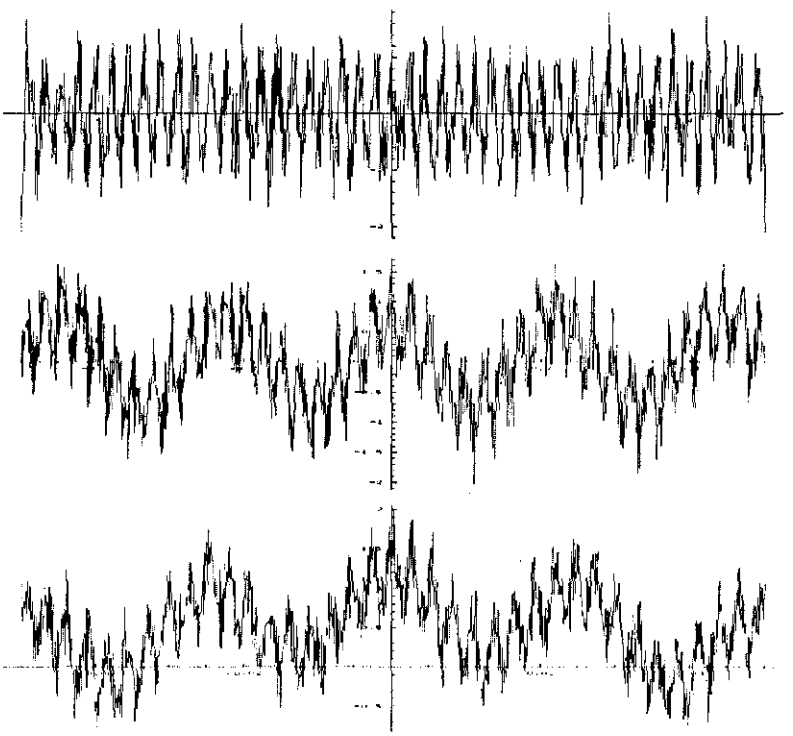

Figure 2.2: Weierstrass function for $a = 9$, $b = 0.7$, with successive 10x zooms on the origin.

During the eighteenth and early nineteenth centuries, it was widely believed that every continuous function must be differentiable at least at one point.

It was well known that a continuous function may not be differentiable at a specific point, for example at $x = 0$ in the function $f(x) =\mid x \mid$. Having established a consistent approach to calculus by introducing the limiting condition for a derivative, i.e. defining a derivative as

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}, \qquad \Delta x \neq 0$$

Karl Wilhelm Weierstrass (1815-1879), was one of the first to create a function that was nowhere differentiable but still continuous. This is a function that has the property that the first derivative $f'(x)$, and all higher order integer derivatives, cannot be found. An example of such a function (invented by Weierstrass) is based on an infinite sum of cosine curves given by

$$f(x) = \sum_{i=1}^{\infty} b^i \cos(a^i \pi x)$$

where a is an odd integer, $b \in (0, 1)$ and $ab > 1 + 3\pi/2$. Figure 2.2 shows a plot of this function when $a = 9$, $b = 0.7$ and with 10 'zooms' on the origin.

After the publication of this result, other mathematicians began to follow Weierstrass example using many 'variations on a theme'.

Giuseppe Peano (1858-1952), for example, introduced the first deterministic space filling function in the 1890s which passes arbitrarily close to any point in the plane. The uses of these functions at the time was not apparent and many mathematicians were alarmed at the loss of differentiation as a 'constant'. Hermite defined them as a '*dreadful plague*', and Poincaré wrote in his collected works (Volume II, page 130), '*Yesterday, if a new function was invented it was to serve some practical end; today they are specially invented only to show up the arguments of our fathers, and they will never have any other use*'. It was not until the 1920's, when nowhere differentiable functions were used to

construct good models for Brownian motion, that such functions started to be appreciated for their practicability. By this time the idea of Weierstrass functions had gone from an interesting peculiarity to become the start of a new field within mathematics.

The growing interest in random motion and stochastic field theory led, in the late 1930s, to Paul Lévy (1886-1971) asking a simple but profound question: *Under what circumstances does the distribution associated with a random walk of a few steps look the same as the distribution after many steps (except for scaling)?* This question is the same as asking under what circumstances do we obtain a random walk that is statistically the same at different scales. One of Paul Lévy's research students was Benoit Mandelbrot who later coined the phrase 'fractal' geometry (e.g. in his most famous book *The Fractal Geometry of Nature*, Freeman, 1982) as the study of geometric objects, either deterministic or stochastic, that are self-similar, i.e. look the same at different scales. The characteristic property of a fractal is that it is *self-similar* for every scale of analysis. This fact implies that any part of a fractal object is a scaled-down copy of the original.

However, for natural objects the self-similarity is observed only for a limited range of scales and it appears in a statistical sense. Moreover, a fractal function is not differentiable in the normal (i.e. integer) sense, but it is differential to fractional order. Further, one way of defining a fractal is in terms of the solution to a fractional differential equation which can be solved by resorting to the applications of the fractional calculus.

Euclidean geometry (a term that derives from Euclid of Alexadria who published his *Elements* around the year 300 BC and provided a systematic devel-

opment of much of Greek mathematics up to that point) is based on ideas, axioms, theorems and results that are associated with simple objects - triangles, squares, circles, lines, etc. Some abstract concepts are required in order to maintain consistency such as defining two parallel lines as those that meet at infinity. However, the underlying philosophy of Euclidean geometry is that we can combine primitive objects to build up and construct complex ones. To do this we first need to analyze a complex object in terms of its 'elements' to construct a simple set of primitives. This is the basis for the construction of most man-made objects and computational Euclidean geometry including computer aided design, solid geometry, etc. It is also the basis we tend to use for analysing a complex problem.

Fractal geometry is based on looking at things in terms of the 'big picture' and observing the fact that the 'smaller pictures' look similar. It uses ideas, axioms, theorems and so on associated with complex objects with repeating patterns, and includes abstract concepts such as infinite repeatability.

Hence, unlike Euclidean geometry, the philosophy of fractal geometry is to construct an object by classifying it in terms of its repetitive underlying structure and repeating this structure again and again. This is the basis upon which many natural object and dynamical systems appear to be based. For example, consider Figure 2.3 which shows a number of gray scale images of natural objects and scenes - a tree, fern, rock, lava, etc. In each case, the image is of an object that, at first sight, appears relatively complex with different textures. However, if we 'look' at the object imaginatively enough in terms of its repeating patterns at different scales, then this complexity starts to be seen for what it is - self-similar simplicity!

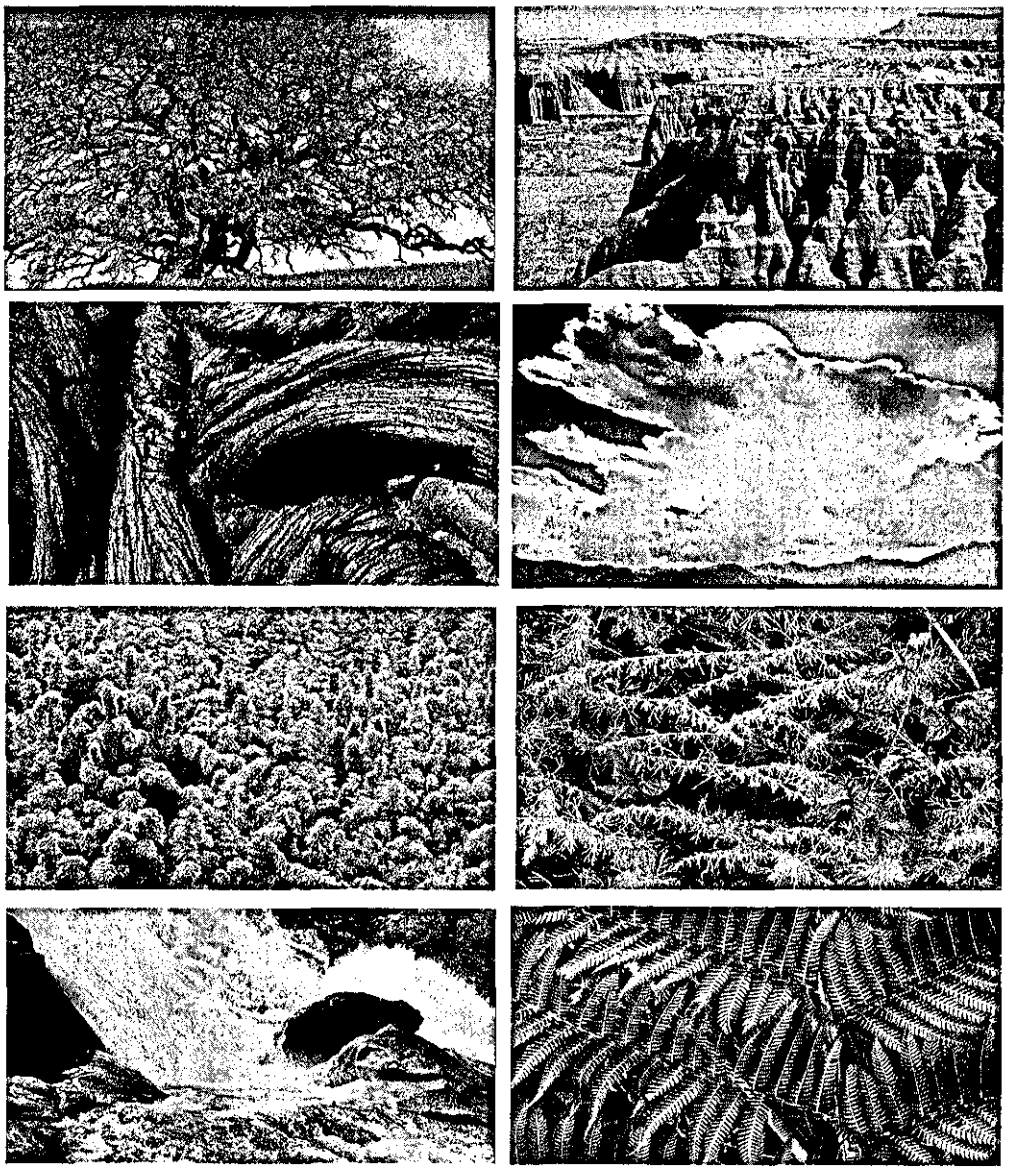

Figure 2.3: Some sample images of natural objects: from top to bottom and from left to right: a tree, rock, lava, a cloud, a forest, sage, a water fall and fern.

This principle is emphasized in Figure 2.4 which shows three images of a fern at different scales to illustrate the principle that *'self-similarity over limited ranges of scale is very common in nature'*. Another example is given in Figure 2.5 which shows another natural fractal. Here, every part of whole cauliflower is similar to another irrespective of the (limited) scale. Another important aspect of fractals is that (unlike Euclidean objects) they exhibit texture. While mathematicians and scientists sometime find texture 'hard to grasp' and define, artists and musicians have understood it for many years. Impressionist paintings are studies in texture, initiated by English artists such as J M W Turner in the mid-Eighteenth Century, extended by the French school of impressionism by artists such as Claude Monet and developed further by modernists such as Jackson Pollack (see Figure 2.6).

Much of the music of composers such as Debussy, Ravel and Scriabin, for example, are studies of musical texture. Indeed, the art, music and languages of most cultures exhibit properties that are fractal, from the stylized versions of self-repeating patterns associated with Islamic art to Japanese art and the work of M C Escher, for example, as illustrated in Figure 2.7.

In this case, a part of the object magnified to the size of the original, exhibits statistical properties similar to those of the original. These 'statistical properties' define the texture of the object (i.e. its roughness). The principal numerical quantification of this texture is defined in terms of the *fractal dimension.*

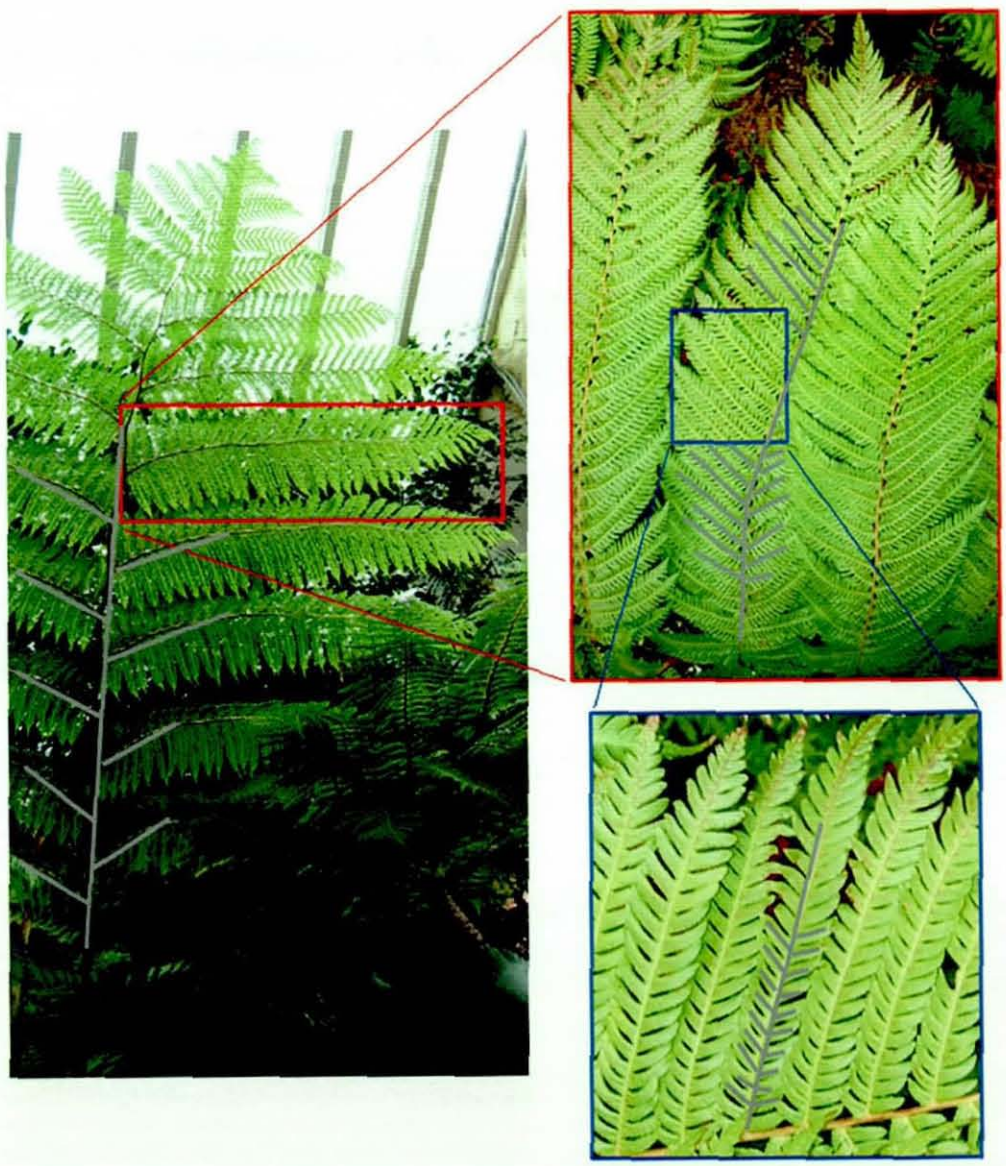

Figure 2.4: Three images of a fern at different scales illustrating the principle of self-similarity.

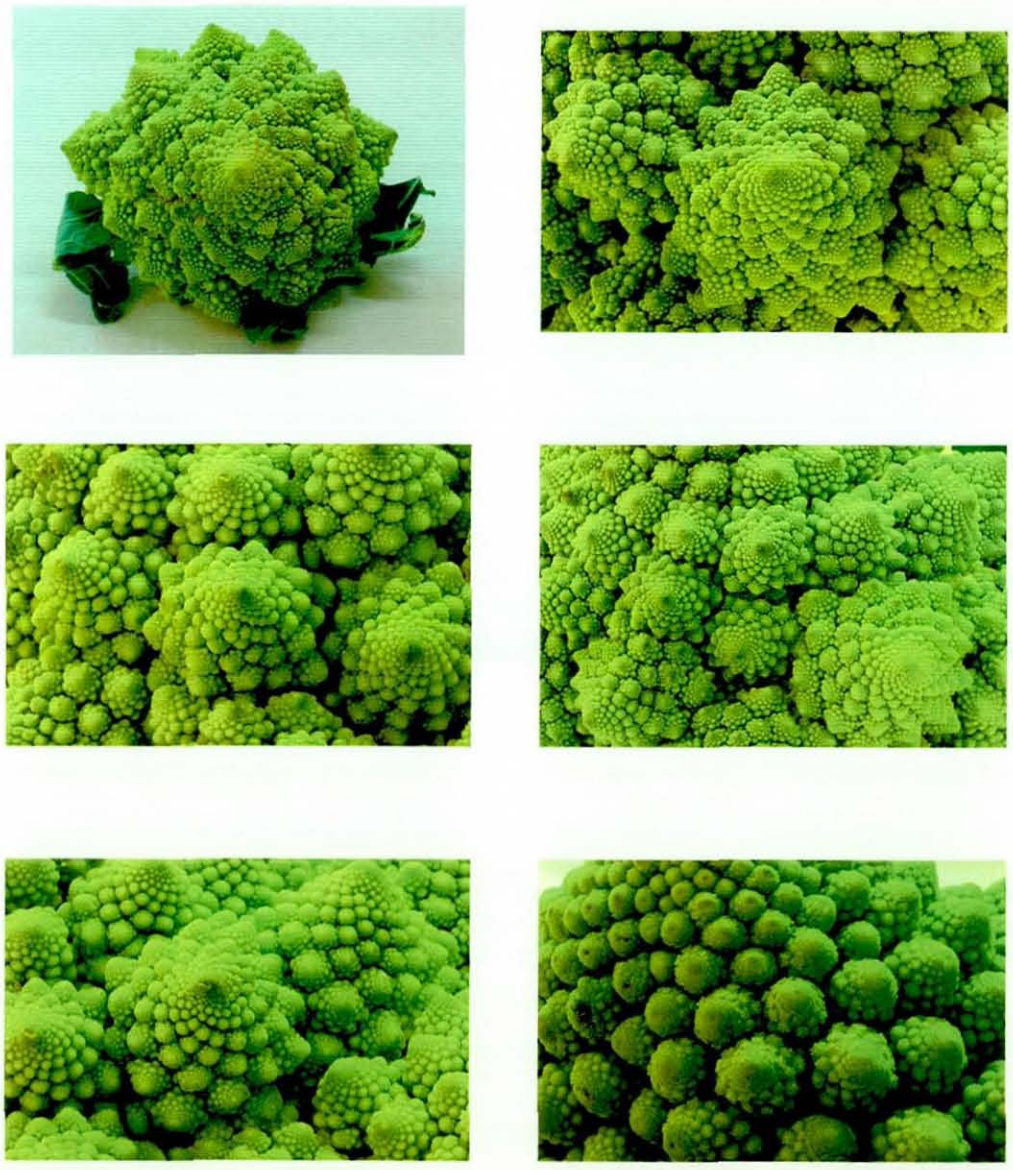

Figure 2.5: Illustration of a self-similar object - a Cauliflower.

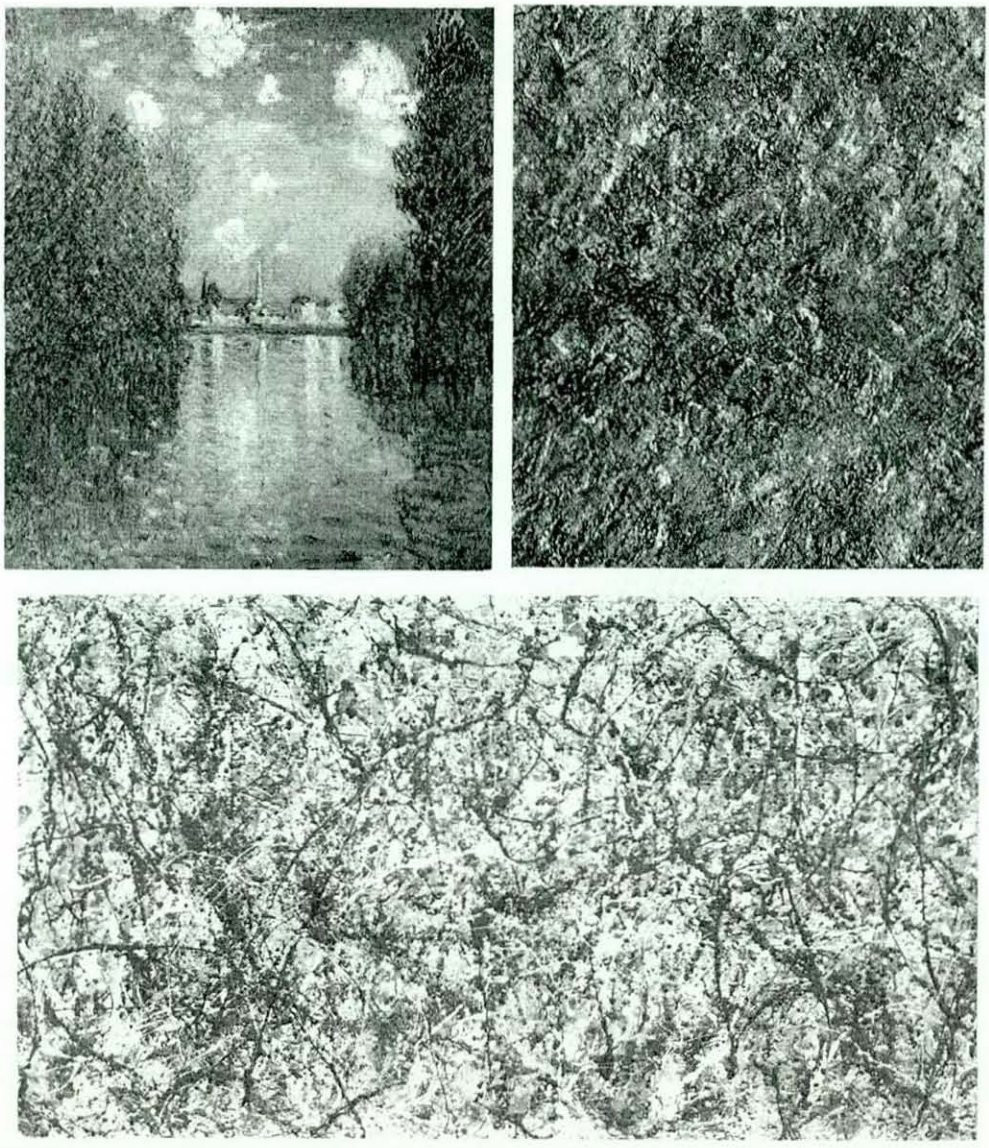

Figure 2.6: Texture by Claude Monet (top-right) taken from the painting shown (top-left) and texture by Jackson Pollock (bottom).

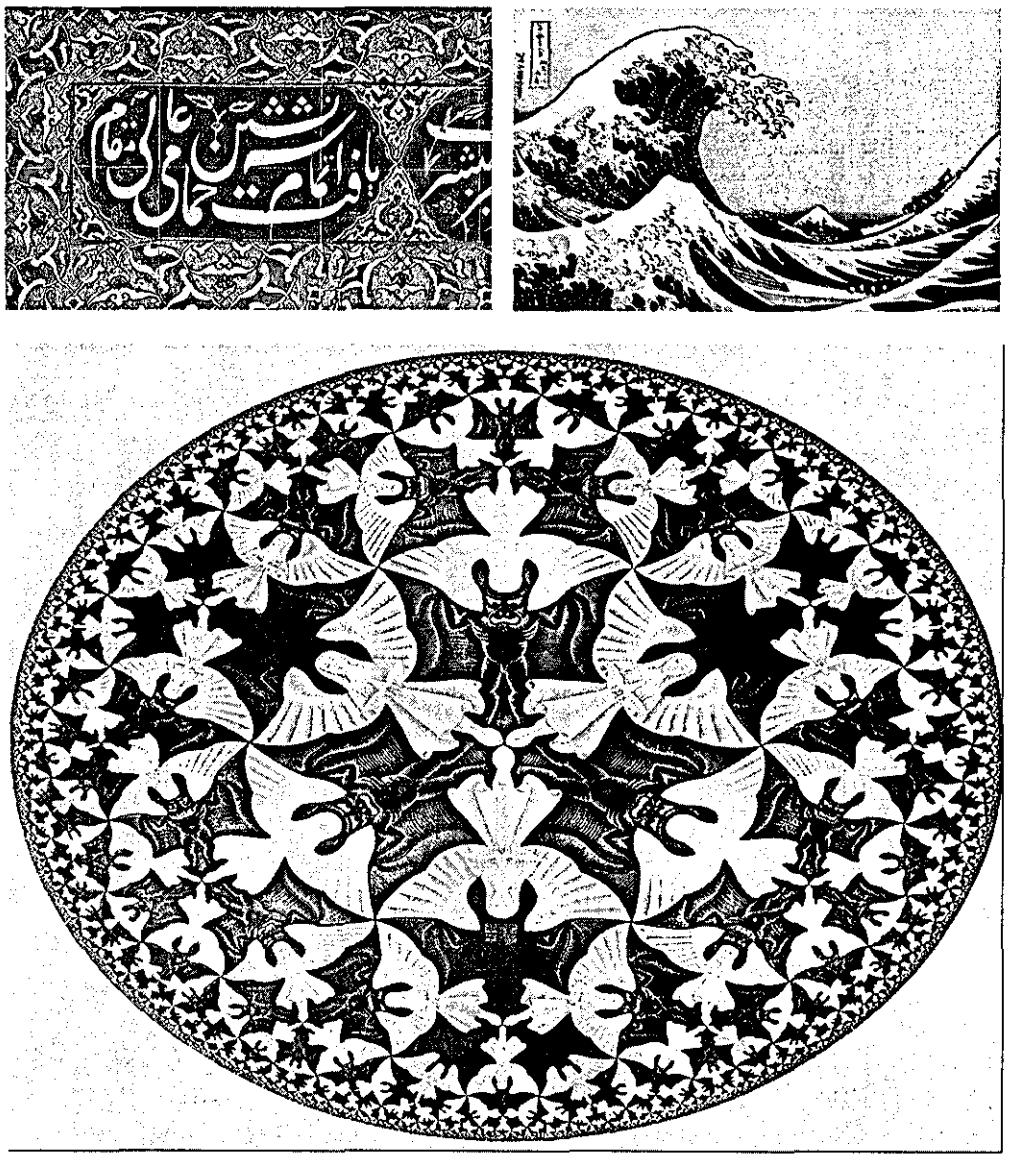

Figure 2.7: Examples of self-similarity in Islamic art (top-left), self-similarity by the Japanese artist K Hokusai from the 1800s (top-right), and an example of deterministic self-similarity by the Dutch graphic artist M C Escher (bottom).

2.4 The Concept of Self-Similarity

The concept of self-similarity is central to fractal geometry. Fractal objects can be classified according to their self-similar properties in terms of the following:

(1) **Deterministic self-similar fractals** in which the fractal object is composed of distinct features which resemble each other in some way at different scales (feature scale invariance). These fractals normally occur in mathematically defined fractals whose realities/constraints on fractal structures by the physical world rarely apply.

(2) **Statistical (Random) self-similarity fractals** in which the features of the fractal object may change at different scales but whose statistical properties at all scales are the same (statistical scale invariance).

Deterministic fractals associated with (1) above are usually generated through some Iterative Function System (IFS) and are remarkable for the complexity that can be derived through the simplest of such systems. The way in which the output from these systems is viewed graphically and interpreted geometrically changes substantially from one fractal to another but the overall principle remains the same. Random self-similar fractals are those used to model a variety of naturally occurring objects (clouds, landscapes, coastlines, etc.). They can be generated by a variety of different stochastic modelling techniques. They can also be considered to be the solution to certain classes of stochastic differential equations of fractional order. Consequently, random fractals have become increasingly important. What began as a purely mathematical concept has now found many applications in the physical and bio-sciences where fractal models arise frequently in a variety of disciplines such as physics, chemistry,

biology and computer science. In the field of digital signal processing, fractal models have proved useful for applications such as data network analysis, texture analysis and image compression.

2.5 Some Examples of Deterministic Fractals

2.5.1 The Von Koch Curve and Island

One of the most famous deterministic fractals was invented by the mathematician Niels Von Koch in 1904 and is called the Von Koch Curve. Here, by deterministic we mean that the Von Koch curve is uniquely determined by a simple production rule. The limiting curve is made from a series of deterministic Euclidean line segments. There is no element of randomness in the production of the curve.

The construction of the curve starts with a line segment of unit length called the *initiator*. The construction of the Koch curve proceeds by replacing the initiator by the *generator* or production rule, that states: Take the initiator, scale it down by a factor $r = 1/3$, make $N = 4$ copies and replace the initiator by these scaled down copies, oriented as shown in Figure 2.8. After iterating this procedure with each segment an infinite number of times, the resulting Koch curve becomes infinitely long and is nowhere differentiable because of the infinitely many discontinuous changes of slope. At the n^{th} step, each line segment has length $\epsilon_n = 3^{-n}$ and there are $N(\epsilon_n) = 4^n$ segments. Alternative, to create the Von Koch Island (also called the Von Koch Snowflake) we join the three lines together to form a triangle (the *initiator*) and repeat the above

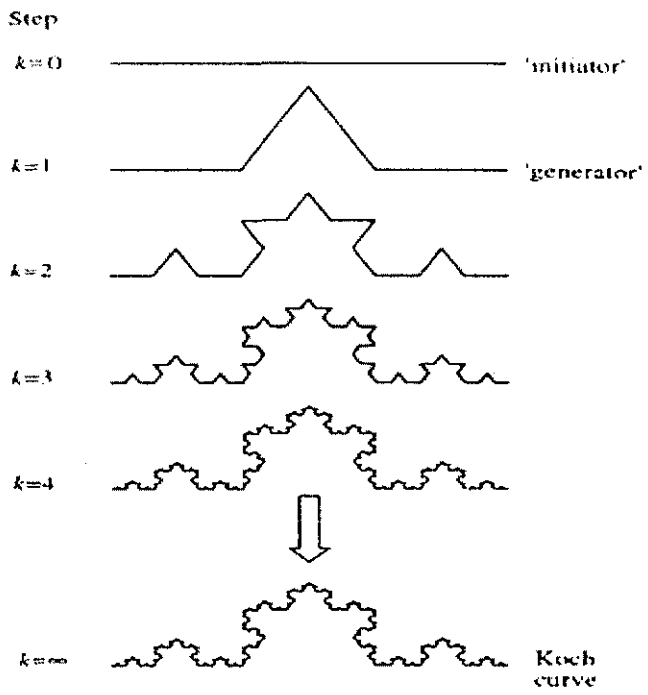

Figure 2.8: The initiator and the first four steps in the construction of the Koch Curve.

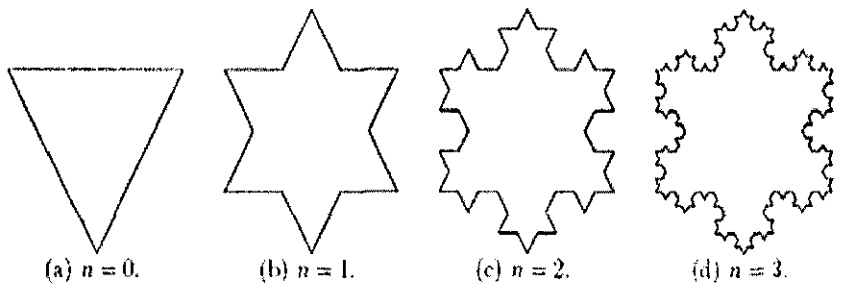

Figure 2.9: The initiator and the first three steps in the construction of the Koch Snowflake.

process on all three sides (see Figure 2.9). Surprisingly it is not too difficult to calculate the area enclosed by each level of the Island. We may define the area of the original triangle to be $A_0 = \Delta$. At each level the small triangles added to the Island are one ninth the size of the last triangles added. At level one we add three of these triangles, so that $A_1 = \Delta + 3/9\Delta = [1 + 3/9]\Delta$. The number of small triangles added at each level is then four times the number added at the last level. Hence at level two $A_2 = [1 + 1/3(1 + 4/9)]\Delta$ and at level three, $A_3 = [1 + 1/3(1 + 4/9 + (4/9)^2)]\Delta$. Repeating these steps, then at n^{th} step

$$A_n = [1 + \frac{1}{3}\sum_{i=0}^{n-1}(\frac{4}{9})^i]\Delta$$

and as $n \rightarrow \infty$,

$$A_\infty = [1 + 1/3(1 - 4/9)^{-1}]\Delta = \frac{8}{5}\Delta$$

Thus although the Island has a fractal boundary, and therefore one of infinite length, the area enclosed is finite and very well defined! This is an example of objects that have come to be known a 'pathological monsters' - geometric objects that do not conform to the principle of conventional (Euclidean) geometry.

However, as one successively zooms in the resulting shape of Figure 2.8 is exactly the same no matter how far in the zoom is applied, (see Figure 2.10).

2.5.2 The Cantor Set

One of the simplest and best known fractals is the so called triadic Cantor Set, named after the Nineteenth Century German mathematician George Cantor. The *initiator* is a line of length L. The production rule is as follows:

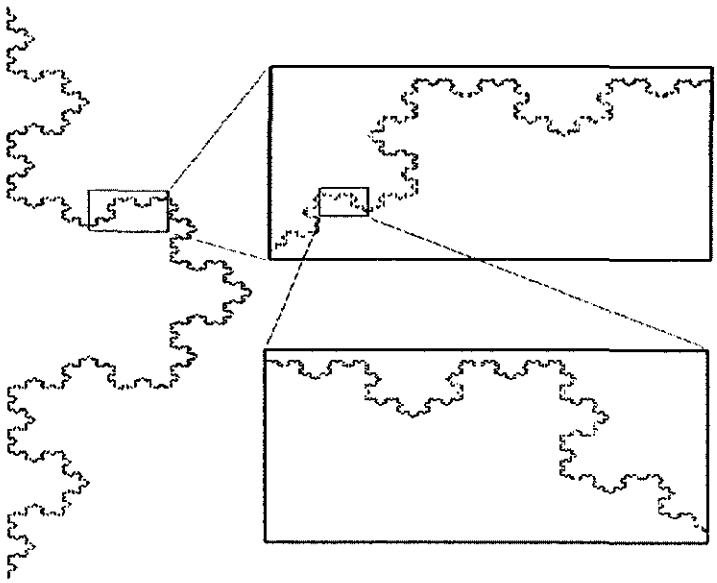

Figure 2.10: The exact self-similarity in Koch Curve.

Divide the line by 3 parts; remove the middle third; repeat the procedure for each of the two remaining parts; repeat this process indefinitely. In each step the same structure is obtained but to a smaller scale [8]. Figure 2.11 visualizes the construction of this 'fractal dust' from the initiator (the first row) to the sixth step.

Figure 2.11: The initiator and the first six steps in the construction of the Cantor Set.

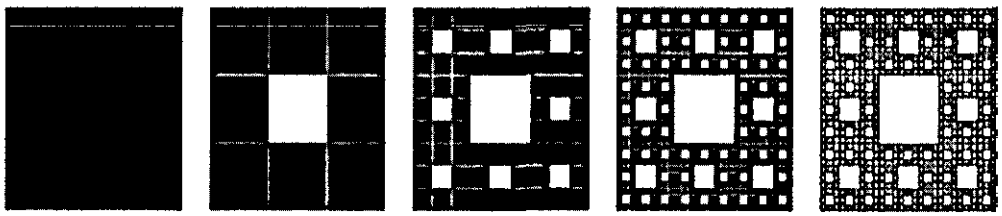

Figure 2.12: The initiator and the the first four steps in the construction of the Sierpinski Carpet.

2.5.3 The Sierpinski Carpet

The Sierpinski carpet is a plane fractal first described by W. Seirpinski in 1916. The carpet is one generalisation of the Cantor dust to two dimensions. The initiator is a rectangle - see Figure 2.12 (top-left). The production rule is: Take the initiator; divide each side by 3; remove the middle part; repeat the procedure for each of the remaining parts.

There are many more examples of deterministic fractals that can be formed through application of some initiator and production rule(s). Indeed, in principle the number of process that can be invented are unlimited. However, in each case, it is desirable to attempt to quantify the fractal object in terms of a suitable parameter which classifies the fractal object. The most important parameter with respect to this requirement is called the *Fractal Dimension.*

2.6 Fractal Dimension

The measure most commonly associated with a self-similar object is its fractal (or similarity) dimension, usually denoted by D. For regular objects (not fractal) the fractal dimension is an integer and is the same as the usual notation of a topological dimension - i.e. an integer dimension of 1, 2 and 3. However, fractal objects can have fractal dimensions that are non-integer and one of the most common definitions (first introduced by Mandelbrot) of a fractal as that it is an object whose fractal dimension is NOT an integer.

The fractal dimension measures the complexity or irregularity of a fractal. An object with a higher fractal dimension is more complicated or 'rough' than one with a lower fractal dimension. The higher the fractal dimension the more the fractal fills space. The fractal dimension is a fractional dimension and an object with a fractal dimension of $D = 1.2$, for example, fills more space than a one-dimensional curve, but less space than a two-dimensional plane. Thus, the fractal dimension succinctly provides information about the geometry of an abject. It *'attempts to quantify a subjective feeling we have about how densely the fractal occupies the metric space in which it lies'* [2].

The fractal dimension is embedded in the original theory of fractals (in [1] for example) that was, in part, based on the work of Hausdorff and Besicovitch [7] who introduced the Hausdorff-Besicovich dimension D_H which is defined as:

$$D_H = \lim_{\epsilon \to 0} \frac{\ln N(\epsilon)}{\ln(1/\epsilon)}$$

where $N(\epsilon)$ is the number of elements of ϵ required to cover the object.

2.6.1 Self-Similarity and Self-Affinity

A bounded set A in a Euclidean n-dimensional space is said to be self-similar if A is the union of N distinct (non-overlapping) copies of itself, each of which has been scaled down by a ratio $r < 1$ in all coordinates. The following relationship between N and r is then valid (see, for example [1])

$$Nr^D = 1 \tag{2.1}$$

so that the similarity (fractal) dimension is given by

$$D = -\frac{\ln N}{\ln r} \tag{2.2}$$

Thus, for example, from equation (2.1) the fractal dimension for the Cantor Set is

$$D = -\frac{\ln 2}{\ln(1/3)} \cong 0.631$$

since this structure is generated from $N = 2$ parts (lines) repeated over the scale $r = 1/3$. This value is to be compared with the initiator for the Cantor Set which has a topological dimension of 1. The structures of the Cantor Set is thus quasi-point-like (fractal dust) having fractal (similarity) dimension between zero and unity.

For the Koch Curve, generated by the procedure illustrated in Figure 2.8, $N = 4$ and $r = 1/3$, so that the similarity dimension is

$$D = -\frac{\ln(4)}{\ln(1/3)} \cong 1.262$$

In this case, instead of the line segment (initiator), which has a topological dimension of 1, the successive structures partially occupy a two-dimensional

space and thus, the dimension of the fractal is more than a curve but less than a surface. Similarly, for the Sierpinski Carpet the similarity dimension is

$$D = -\frac{\ln(8)}{\ln(1/3)} \cong 1.893$$

The range in the value of D characterizes the types of fractals as shown in Table 2.1

Fractal Type	Fractal Dimension
Fractal Dust	$0 < D < 1$
Fractal Curve(Fractal Signal)	$1 < D < 2$
Fractal Surface(Fractal Image)	$2 < D < 3$
Fractal Volume	$3 < D < 4$
Fractal Time	$4 < D < 5$
Hyper Fractals	$D > 5$

Table 2.1: Fractal types and range of fractal dimension, D.

In each case, the fractal may be deterministic or random. When the fractal is random then it is taken to be composed of N distinct subsets, each of which is scaled down by a ratio $r < 1$ from the original and the same in all statistical respects to the scaled original. Table 2.2 gives some examples of common Euclidean and fractal objects together with their fractal dimension. The fractal dimension in this case is given by equation (2.2).

Object	Dimension
point	0
Cantor set	0.6309
Koch curve	1.2619
plane	2
Sierpinski sponge	2.7268
solid	3

Table 2.2: Common Euclidean and fractal objects and their fractal dimension.

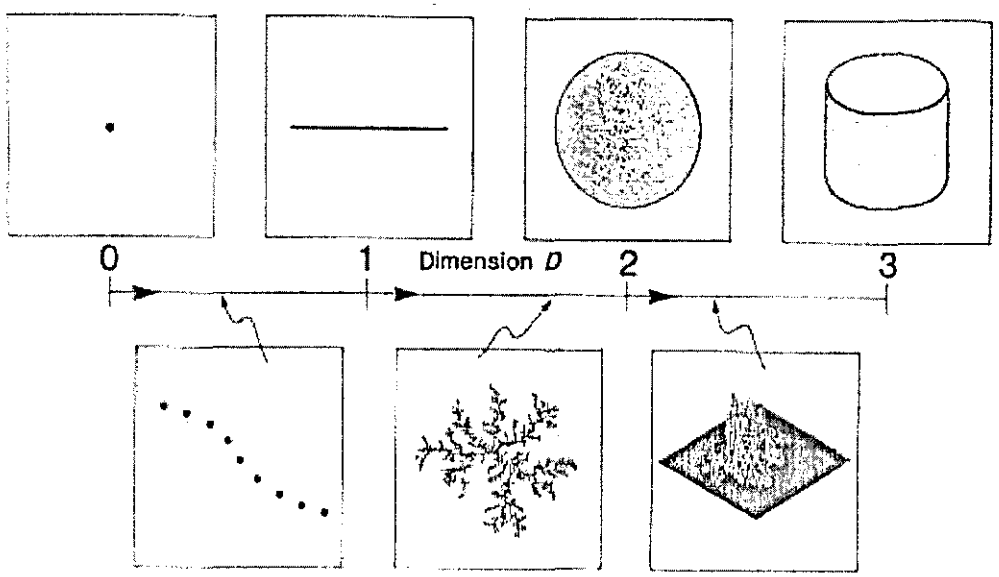

Figure 2.13: The continuum of fractal dimensions.

The scaling ratios need not be the same for all the scaled down copies. Certain fractal sets are composed of the union of N distinct subsets each of which is scaled down by a ratio $r_i, i = 1, 2, 3...., N$ from the original in all coordinates.

Thus the fractal dimension is given by a generalization of equation (2.1), namely

$$\sum_{i=1}^{N} r_i^D = 1 \tag{2.3}$$

Further, there are self-affine fractal sets which are scaled by different ratios in different coordinates. For example, consider the curve $f(x)$ which satisfies

$$f(\lambda x) = \lambda^{\alpha} f(x) \tag{2.4}$$

where λ is a scaling factor and α is the scaling exponent. Equation (2.4) implies that a scaling of the x -coordinate by λ gives a scaling of the f -coordinate by a factor λ^{α} which is an example of self-affinity.

A special case occurs when $\alpha = 1$ as we have a scaling of x by λ producing a scaling of f by λ which is an example self-similarity. Self-affine curves repeat themselves only when the different axes are magnified by different factors, whereas self-similar curves have the same factor for each axis. Random fractal signals and images are, in general, examples of self-affine records.

Naturally occurring fractals do not have such regular structures as deterministic fractals. For instance, in a coastline there is an irregularity over a broad range of spatial scales. A magnified view of one part of the coastline will not precisely reproduce the full picture, but it will have the same qualitative appearance. The time series of many natural phenomena are self-affine fractals. Small sections taken from these series scaled by the appropriate factor, cannot be distinguished from the whole signal. Being able to recognize a time series as fractals provides a way of 'linking' the information time series convey at different time scales. Many time series exhibit statistical self-affine structures. For such structures, the similarity dimension, as given by Equation (2.2), is not appropriate because the data is not based on a precise production rule. They also differ from the strictly mathematically defined fractals in that they do not display statistical or exact self-similarity over all scales. Rather, they display fractal properties over a limited range of scales.

2.7 Least Squares Principle

In the next section, we outline some of the many different methods to estimate the fractal dimension which often depend on application of the least squares criterion. The least squares method is therefore briefly reviewed.

Let $f_i, i = 1, 2, 3, ..., N$ be a real digital function and let $\hat{f}_i$ be an approximation to this function. We assume that $\hat{f}_i$ is the expected form of the data f_i. The least squares error, E, is then defined as

$$E = \sum_{i=1}^{N} (f_i - \hat{f}_i)^2$$

In most cases, algorithms for computing the fractal dimension use logarithmic or semi-logarithmic plots to fit the results of a given algorithm to a best straight line. In these cases, we are interested in finding the slope B and, in certain cases, the intercept A of the line

$$\hat{f}_i = Bx_i + A$$

To find the best fit, we are required to minimize the error, E, which is taken to be a function of A and B. This is achieved by finding the solutions to the equations $\frac{\partial E}{\partial B} = 0$ and $\frac{\partial E}{\partial A} = 0$

Differentiating E with respect to A and B yields

$$\frac{\partial E}{\partial B} = \sum_{i}^{N} x_i(f_i - Bx_i - A) = 0$$

and

$$\frac{\partial E}{\partial A} = \sum_{i}^{N} (f_i - Ax_i - B) = 0.$$

Solving for A and B we obtain

$$B = \frac{N \sum_{i=1}^{N} f_i x_i - (\sum_{i=1}^{N} f_i)(\sum_{i=1}^{N} x_i)}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2}$$

and

$$A = \frac{1}{N}(\sum_{i=1}^{N} f_i - B \sum_{i=1}^{N} x_i)$$

2.8 Survey of Some Methods for Estimating Fractal Dimension

In general, there is no unique and general rule for estimating the fractal dimension of a data set. A wide spectrum of techniques has been developed to estimate the fractal dimension. As with many other techniques for digital signal processing, the computation of the fractal dimension can be undertaken in 'real space' (processing the data directly) or in 'transform space' (processing the data after taking an appropriate transform). The techniques of estimating the fractal dimension can be broadly categorized into two classes:

1. **Size-measure relationships** that are based on recursive length or area measurements of a curve or surface using different measuring scales.

2. **Application of relationships** which are based on approximating or fitting a curve or surface to a known fractal function or statistical property such as the variance.

The following subsections give a brief explanation of some salient methods as applied to fractal signals and/or images. Note that in the following sections, the algorithms for computing the data used to estimate the fractal dimension are based on the least squares method discussed in the previous section. In particular, some of these algorithms are based on the following relationship:

$$Length = c \ \ Step^{\beta}$$

which can be linearized as, $\ln(Length)=\ln(c)+\beta\ln(Step)$

where $A = ln(c)$ and $B = \beta$. Here, the Length represents the measurement of the curve using a 'ruler' of size *Step*. β is the slope of the the bi-logarithmic

plot which has a simple algebraic relationship with the fractal dimension D, depending on the algorithm used.

2.8.1 The Line Divider Method

This method was introduced by Shelberg [5], in which for a given step *Step*, measures the number of chord lengths *Length* required to cover a fractal curve. The technique is based on the principle of taking smaller and smaller rulers of size *Step* to cover the curve and counting the number of rulers required in each case. This approach is a recursive process in which the *Step* is decreased (typically halved) and the new Length calculated. Here, the input signals are taken to be of size N, where N is a power of 2 due to the recursive nature of the method. The process is repeated until there are enough points to reasonably fined the line of the best fit for the relationship:

$$Log[total\ Length]\ Vs.\ Log[Step\ size]$$

The fractal dimension is then given by

$$D = -\beta$$

, where $\beta = \frac{Log[total\ Length]}{Log[Step\ size]}$

In the Line-divider method the initial Step must be carefully chosen. Shelberg specified an appropriate starting value as half of the average distance between the points. Clearly the computation of the initial value and the procedure required to count the number of Steps, makes this algorithm time consuming. Figure 2.14 shows an illustration of this method for the computation of the fractal dimension.

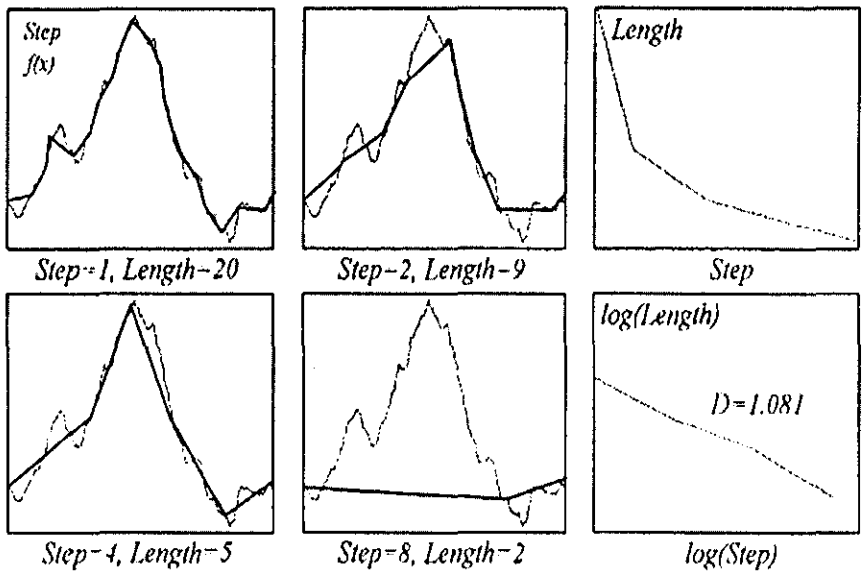

Figure 2.14: Illustration of the Line-Divider method for computing the fractal dimension D of a signal showing four iterations and the least squares fit.

2.8.2 The Box-Counting Method (BCM)

One of the most popular algorithms for estimating the fractal dimension of fractal signals and images is the box-counting method originally developed in [6] but modified by others to develop a reasonably fast and accurate algorithm. Its popularity is largely due to its relative ease of mathematical calculation and empirical estimation [7]. In general, BCM involves covering a fractal with a grid of n-dimensional boxes or hyper-cubes with side length ϵ and counting the number of non-empty boxes $N(\epsilon)$. The number of $N(\epsilon)$ depends on the choice for ϵ. For signals, the grid is one of squares and for images a grid of cubes. Boxes of recursively different sizes are used to cover the fractal. However, an input signal with N elements is used where N is of power 2.

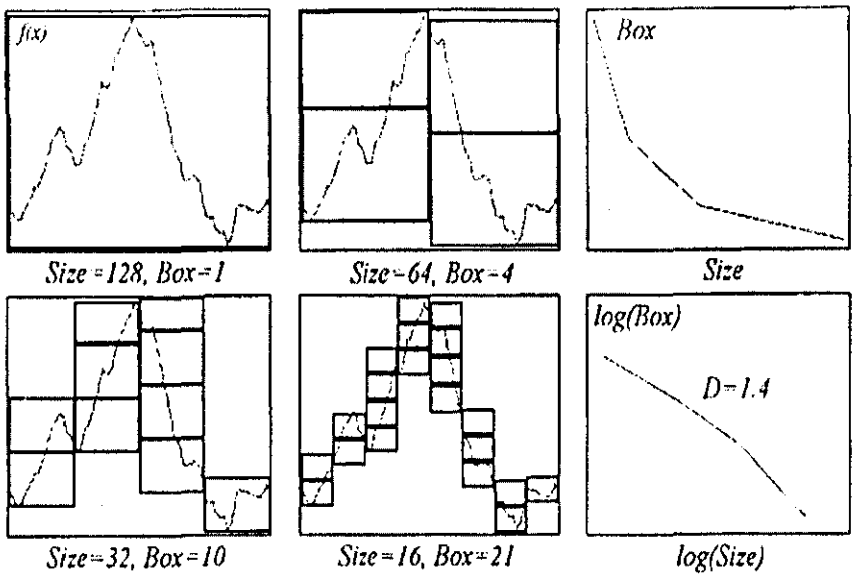

Figure 2.15: Illustration of the Box-Counting Method for computing the fractal dimension of a signal showing four iterations and the least squares fit.

The slope β obtained in a bi-logarithmic plot of the number of boxes used against their sizes then gives the fractal dimension D where $D = -\beta$ [8]. Figure 2.15 illustrates the principle of this method.

For a one-dimensional smooth curve of length L, we would expect

$$N(\epsilon) \approx \frac{L}{\epsilon}.$$

Thus the box counting measure gives us the generalization

$$N(\epsilon) \propto \frac{1}{\epsilon^D}$$

Then,

$$D_B = \lim_{\epsilon \to 0} -\frac{\ln N(\epsilon)}{\ln \epsilon} \tag{2.5}$$

Successive divisions by a factor of 2 are used for the box Size to give a regular spacing in the bi-logarithmic plot and least squares fit.

The behavior of this algorithm is such that the greater the number of points used for the least squares fit, the better the estimate of the fractal dimension. However, it must be remembered that the box counting dimension is a distinct definition in its own right and is not just an algorithmic method for estimating the Hausdorff dimension proper.

In general, box counting algorithms behave well and produce accurate estimates for fractal dimension between 1 and 1.5 for digital signals and between 2 and 2.5 for digital images and are easy to code and fast to compute. Outside this range (i.e. for higher fractal dimension) they tend to give less accurate results [8].

2.8.3 The Prism Counting Method

Clark [10], defines an algorithm based on the idea of box counting, in which instead of counting the number of boxes in a region for given size, the area based on four triangles defined by the corner points is computed and summed over a gray level surface.

The triangles define a prism based on the elevated corners and a central point computed in terms of the average of the four corners. A bi-logarithmic plot of the sum of the prism areas for a given base area gives a fit to a line whose slope is β in which $D = 2 - \beta$. The basic engine for this algorithm is similar to the box counting method but is slower due to the number of multiplications implied by the calculation of the areas.

2.8.4 The Perimeter-Area Relationship Method

For non-fractal closed curves in the plane, the perimeter L is related to the enclosed area A by the relation $L = C\sqrt{A}$, where C is a constant for given type of shape. For a square $C = 1$ and for circles $C = 2\sqrt{\pi}$. Mandelbrot [1], generalizes this equation for the case of closed fractal curves to give

$$L = C(\sqrt{A})^D$$

where $1 < D < 2$. In this case, $D = \ln L / \ln C\sqrt{A}$.

2.8.5 Fractional Brownian Motion (fBm)

Fractional Brownian motion, also known as the 'Random Walk Process', is a function denoted by $B_H(t)$ whose increments $\Delta B_H(x) = B_H(t+x) - B_H(t)$ are Gaussian distributed with zero mean and variance given by

$$\left\langle |B_H(t+x) - B_H(t)|^2 \right\rangle \propto |x|^{2H}.$$

The parameter $0 < H < 1$ specifies the statistical scaling behavior of fBm. We have

$$\left\langle B_H(rx)^2 \right\rangle \propto \left\langle \Delta B_H(x)^2 \right\rangle .$$

For the case when $H = \frac{1}{2}$ the function B_H reduces to the ordinary Brownian motion and $\Delta B_H(t) \propto \Delta t$. If we consider the trace of a fBm covering a time period $\Delta t = 1$, then, without loss of generality, we define the vertical range $\Delta B_H = 1$. We know that B_H is statistically self-similar so if we divide the time span into $N = 1/\Delta t$ equal intervals, the vertical range within these intervals

will be

$$\Delta B_H = \Delta t^H = \frac{1}{N^H} = N^{-H}$$

Using the box counting method, with boxes of length $\epsilon = \frac{1}{N}$, the number of boxes required to cover each interval is

$$\Delta B_H \Delta t = \frac{N^{-H}}{N^{-1}} = N^{1-H}$$

This means that,

$$N(\epsilon) = N N^{1-H} = N^{2-H}$$

and thus from equation (2.2) we have

$$D = 2 - H \tag{2.6}$$

Thus, since for ordinary Brownian motion $H = 1/2$ the fractal dimension is $D = 3/2$.

2.8.6 The Power Spectrum Method (PSM)

The PSM consider here is both generalizable and computationally accurate. The PSM is implemented by applying the Fast Fourier Transformation (FFT) to the real-space signal in order to obtain the power spectrum $P(\omega)$ where

$$P(\omega_i) = Re(\omega_i)^2 + Im(\omega_i)^2$$

We consider the power spectrum of an ideal one-dimensional fractal signal with dimension D. Then $\hat{P}(\omega_i) = c|\ \omega_i\ |^{-\beta}$, where c is a scaling constant and β is the spectral exponent, which is related to the fractal dimension, D, as described later in Section 2.10. Thus, by fitting a least squares error line to the data, the value of the spectral exponent β and hence D can be obtained.

One of the main advantages of this approach is that the computation of D is based on an explicit formula rather than on some iterative process. In the following section the full details for the discrete case are given.

2.9 Computation of Power Spectrum Parameter, β

Consider a digital fractal signal f_i, $i = 1, 2, 3, ..., N$ (N being a power of 2), and suppose the digital power spectrum $P(\omega_i)$ is obtained by applying FFT to f_i. This data can be approximated by(where c is a generic constant)

$$F(\omega_i) = \frac{c}{\mid \omega_i \mid^{\frac{\beta}{2}}}$$

or

$$\mid F(\omega_i) \mid^2 = \hat{P}(\omega_i) = \frac{c}{\mid \omega_i \mid^{\beta}}$$

and by using the least squares method we can estimate β and c as follows:

$$E(\beta, c) = \sum_{i=1}^{N} [\ln P(\omega_i) - \ln \hat{P}(\omega_i)]^2$$

$$= \sum_{i=1}^{N} [\ln P(\omega_i) - C + \beta \ln \omega_i]^2$$

where $C = \ln c$ and it is assumed that $\omega_i > 0$ and $P(\omega_i) > 0$, $\forall i$.

Differentiating E with respect to β and C gives

$$\frac{\partial E}{\partial \beta} = 2 \sum_{i=1}^{N} [\ln P(\omega_i) - C + \beta \ln \omega_i] \ln \omega_i$$

$$\frac{\partial E}{\partial C} = -2\sum_{i=1}^{N}[\ln P(k_i) - C + \beta \ln \omega_i]$$

Solving $\frac{\partial E}{\partial \beta} = 0$ and $\frac{\partial E}{\partial C} = 0$ we obtain the following formulas for β and C, [3]:

$$\beta = \frac{N\sum_{i=1}^{N}(\ln \omega_i)\ln P(\omega_i) - (\sum_{i=1}^{N}\ln \omega_i)(\sum_{i=1}^{N}\ln P(\omega_i)}{(\sum_{i=1}^{N}\ln \omega_i)^2 - N\sum_{i=1}^{N}(\ln \omega_i)^2}$$

$$C = \frac{1}{N}\sum_{i=1}^{N}\ln P(\omega_i) + \frac{\beta}{N}\sum_{i=1}^{N}\ln \omega_i$$

where $c = \exp C$. Here, $P(\omega_i)$ is the measured power spectrum of the signal and ω_i its spatial (or temporal) frequency. Since the power spectrum of real signals of size N is symmetric about the DC level, where the DC level is taken to the mid point $\frac{N}{2} + 1$ of the array, so in practice we consider only $\frac{N}{2}$ data points that lie to the left (or right) of the DC.

2.10 Computing the Fractal Dimension of a Fractal Signal

Herein, we extract the relationship between the spectral exponent β and the fractal dimension D. Suppose $f(t)$ be a fractal signal over an infinite support and a finite sample $f_T(t)$ given by

$$f_T(t) = \begin{cases} f(t), & 0 < t < T \\ 0, & o.w \end{cases}$$

In reality $f(t)$ ia a random function and for any experiment or computer simulation we should consider a finite sample. Let $F_T(\omega)$ be the Fourier transform of $f_T(t)$, then

$$f_T(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F_T(\omega)e^{i\omega t}d\omega$$

and

$$P_T(\omega) = \frac{1}{T}|F_T(\omega)|^2$$

$$P(\omega) = \lim_{T\to\infty} P_T(\omega)$$

where $P_T(\omega)$ and $P(\omega)$ are the Power Spectrum of $f_T(t)$ and $f(t)$ respectively. The power spectrum gives an expression for the power of a signal for particular harmonics. Suppose that $g(t)$ is the function obtained from $f(t)$ through scaling the t-coordinate by some scale $r > 0$, and f-coordinate by $1/r^H$. Taking a finite sample as before,

$$g_T(t) = \begin{cases} g(t), & 0 < t < T \\ 0, & otherwise \end{cases}$$

$$g_T(t) = \begin{cases} \frac{1}{r^H} f(rt), & 0 < t < T \\ 0, & otherwise \end{cases}$$

Let $G_T(\omega)$ and $P'_T(\omega)$ be the Fourier transform and the power spectrum of $g_T(t)$, respectively. We can then obtain an expression of $G_T(\omega)$ in terms of $F_T(\omega)$ as follows,

$$G_T(\omega) = \int_0^T g_T(t)e^{-i\omega t}dt$$

$$= \int_0^T \frac{1}{r^H} f(rt)e^{-i\omega t}dt$$

Substitute $s = rt$ we obtain,

$$\frac{1}{r^{H+1}} \int_0^T f(s) e^{-\frac{i\omega s}{r}} ds$$

and hence

$$G_T(\omega) = \frac{1}{r^{H+1}} F_{rT}(\frac{\omega}{r})$$

and the power spectrum of $g_T(t)$ is

$$P'_T(\omega) = \frac{1}{T}|G_T(\omega)|^2$$

$$= \frac{1}{r^{2H+1}} \frac{1}{rT} |F_{rT}(\frac{\omega}{r})|^2$$

When $T \longrightarrow \infty$, we obtain the power spectrum of $g(t)$ as

$$P'(\omega) = \frac{1}{r^{2H+1}} P(\frac{\omega}{r})$$

Since $g(t)$ is a properly scaled version of $f(t)$, their power spectra are equal, and so

$$P(\omega) = P'(\omega) = \frac{1}{r^{2H+1}} P(\frac{\omega}{r}).$$

Now, formally if we set $\omega = 1$ and then replace $1/r$ by ω we get

$$P(\omega) \propto \frac{1}{\omega^{2H+1}} = \frac{1}{\omega^{\beta}}.$$

The signal that we have produced is statistically self-similar and it is defined by the parameter H as introduced it in the previous section for fBm.

From the above equation we have $\beta = 2H + 1$, and by using the equation (2.6), we can compute the Fractal dimension D as

$$D = 2 - H = 2 - \frac{\beta - 1}{2} = \frac{5 - \beta}{2}.$$

This relationship provides a simple formula for computing the fractal dimension from the power spectrum of a signal [8].

2.11 References

[1] Mandelbrot B.B., *The Fractal Geometry of Nature*, Freeman, 1983.

[2] Barnsley M.F., *Fractals Everywhere*, John Wiley & Sons, New York, 1995.

[3] Blackledge J.M., *Digital Signal Processing*, Horwood, London, 2003.

[4] Peitgen H., Jurgens H. and Saupe D., *Chaos and Fractals*, Springer Verlage, Berlin, 1992.

[5] Shelberg M., *The Development of a Curve and Surface Algorithm to Measure Fractal Dimensions*, Ms Thesis, Ohio State University, 1982.

[6] Voss R., *Random Fractal: Characterization and Measurement.Scaling Phenomena in Disorder Systems*, R.Pynn and A. Skjeltorps, USA, 1982.

[7] Falconer K., *Fractal Geometry: Mathematical Foundations and Applications*, John Wiley& Sons Ltd., UK, 1990.

[8] Turner M.J., Blackledge J.M. and Andrews P.R., *Fractal Geometry in Digital Imaging*, Academic Press Ltd., UK, 1998.

[9] Giordano A.A., Hsu F.M., *Least Squares Estimation with Application to Digital Signal Processing*, John Wiley& Sons Ltd., UK, 1985.

[10] Clarke K. C., 'Scale-based simulation of topographic relie," *The American Cartographer*, vol. 15, no. 2, pp. 173-181, 1988.

[11] Crownover R.M., *Introduction to Fractals and Chaos*, Jones and Barlett, Boston, 1995.

[12] Tatom F.B., *The Application of Fractional Calculus to the Simulation of Stochastic Processes*, Engineering Analysis Inc., Hunstville, Alabama, AIAA-89/0792, 1989.

[13] Tricot C., *Curves and Fractal Dimension*, New York, London: Springer-Verlag, 1995.

[14] Crownovre R.M., *Introduction to Fractals and Chaos*, Boston: Jones&Bartlett, 1995.

[15] Falconer K.J., *Techniques in Fractal Geometry*, John Wiley& Sons Ltd., 1997.

[16] Schepers H.E., Beek J.H. and Bassingthwaighte J.B., "Four methods to estimate the fractal dimension from self-affine signals [medical application]," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 11, no. 2, pp. 57-64, 71, Jun 1992.

[17] Reljin I. and Reljin B., "Fractal geometry and multifractals in analyzing and processing medical data and images," Archive of Oncology, vol. 10, no. 4, pp. 283-293, 2002.

[18] Kaye B.H., *A Random Walk through Fractal Dimensions*, 2nd edition, John Wiley& Sons Ltd., 1994.

[19] Theiler J., "Estimating fractal dimension," *Optical Society of America*, vol. 7, no. 6, pp. 1055-1073, June 1990.

[20] Mandelbrot B.B. and Frame M., "Fractals," *Encyclopedia of Physical Science and Technology*, vol. 20, no.1 , pp. 185-199, June 2001.

Chapter 3

Multi-Fractal Models and Fractal Modulation

3.1 Random Scaling Fractal Signals

Many signals observed in nature are random fractals. Random fractals are signals that exhibit the same statistics at different scales. In other words, they are signals whose frequency distribution (Probability Distribution Function or PDF) has the same 'shape' irrespective of the scale over which they are observed. Thus, random fractal signals are (statistically) self-similar; they 'look the same' (in a stochastic sense) at different scales. We can define this property as follows: Suppose $s(t)$ is a signal, with $\Pr[s(t)]$ its PDF and λ is a scaling parameter, then the signal $s(t)$ exhibits statistical self-similarity if

$$\Pr[s(\lambda t)] = \lambda \Pr[s(t)] \tag{3.1}$$

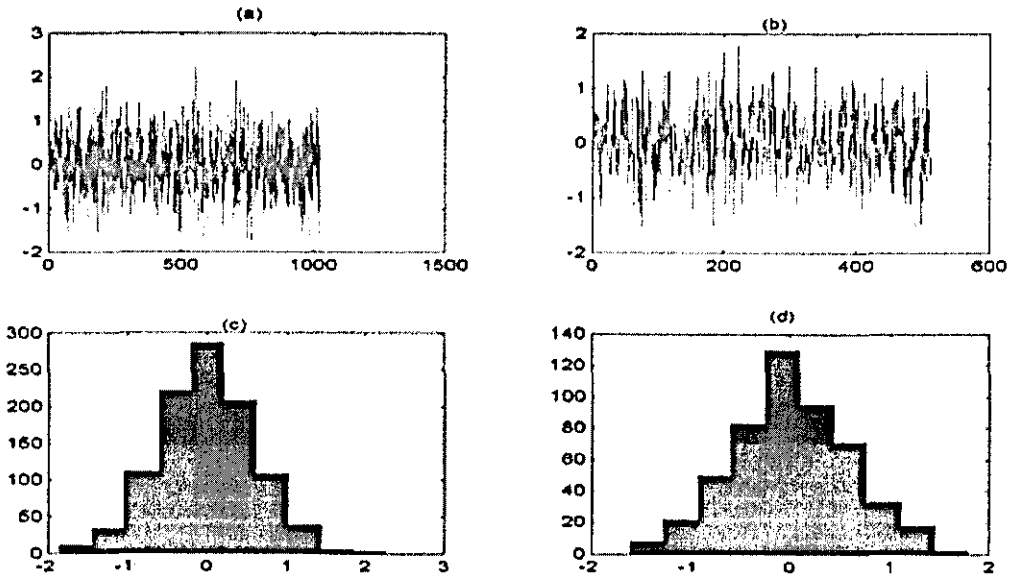

Figure 3.1: (a)&(c) Fractal signal of size N=1024 and its PDF, (b)&(d)Fractal signal of size N=512 and its PDF.

The interpretation of this result is that as we zoom into a fractal signal to observe the detailed structure of the signal, changing the scale by a factor of λ, the distribution of amplitudes remains the same (subject to scaling λ) even though the signal itself 'looks different at a different scale', i.e. $s(t) \neq s(\lambda t)$. In a more general sense, random fractal signals are statistically self-affine, i.e. when they conform with the following property:

$$\Pr[s(\lambda t)] = \lambda^q \Pr[s(t)], \qquad q > 0. \tag{3.2}$$

Figure 3.1 illustrates the idea of the of statistical self-similarity, where the signal (b) is scaled down version of signal (a). The two signals have the same PDF form for the scaling $\lambda = 1/2$.

3.2 Mathematical Modeling of Transmission Noise

Developing mathematical models to simulate and analyse noisy signals has an important role in digital signal processing. The ideal approach for developing a model for transmission noise is to analyse the physics of a transmission system. However, there are a number of problems with this approach. First, the physical origins of many noise types are not well understood. Secondly, conventional approaches for modeling noise fields usually fail to accurately predict these characteristics [1].

The application of fractal geometry to modeling naturally occurring signals is well known. This is due to the fact that the statistics and spectral characteristic of Random Scaling Fractals (RSFs) are consistent with many objects found in nature, a characteristics that is compounded in the term 'statistical self-affinity'. This term refers to random processes that are scale invariant.

A RSF signal is one whose PDF remains the same irrespective of the scale over which the signal is sampled. Thus, as we zoom into a RSF signal, although the pattern of amplitude fluctuations will change across the field of view, the distribution of these amplitudes remains the same (a scaled down version of the 'original'), see Figure 3.1. Many noise signals found in nature are statistically self-affine including transmission noise. Also, speech signals, financial time series and Internet traffic time series tend to exhibit the characteristics of RSFs which is discussed further in Chapter 5. The incredible range of vastly different systems which exhibit random fractal behavior is leading the scientific community to consider statistical self-affinity to be a universal law, a law that

is particularly evident in systems which are undergoing a phase transition. In a stable state, the behavior of the elements from which a system is composed depends primarily on their local neighbors and the statistics of the system is not self-affine.

In a critical state, the elements become connected, propagating 'order' throughout the system in the sense that the statistical characteristics of the system are self-affine with 'system wide' correlations. This is more to do with the connectivity of the elements than the elements themselves. Of course, critical states can be stable in the dynamical sense. Moreover, critical states appear to be governed by the power law

$$\text{System(size)} \propto \frac{1}{(\text{size})^q},$$

where $q > 0$ is a non-integer. Herein, the term 'system' is a generic term representative of some definable parameter that can be measured experimentally over different scales of a certain 'size'. This power law is the principle 'signature' that the system is behaving in a statistically self-affine way.

The published literature shows that there are a wide variety of examples which demonstrate this power law. In particular, the distribution of Internet traffic data is statistically self-affine, i.e. the frequencies of the flow of bytes in a time unit interval is the same at different scales. In Chapter 5 we demonstrate this property for Internet traffic by considering the traffic of an Ethernet network at Loughborough University.

The power law given above, which governs so many of natural signals and systems, is a universal law. However, to date, there is no general mechanism or deeper understanding through which this law can be derived.

RSF signals are characterized by power spectra whose frequency distribution is proportional to $\frac{1}{\omega^q}$ where ω is the frequency and $q > 0$ is the 'Fourier dimension', a value that is related to the Fractal Dimension D. The above power law describes convenential RSF models which are based on stationary processes in which the 'statistics' of the RSF signals are invariant of time and the value of q is constant. However, when the signal is governed by a non-stationary process, the value of q may change. Such signals are common; they are examples of *multi-fractal* behavior and finding a theoretical basis for modeling and interpreting such signals is important in many areas of science and engineering.

There are two principal approaches to characterizing a stochastic field:

(i) Analysis of the Probability Density Function (PDF) or the Characteristic Function (i.e., the Fourier transform of the PDF) - the shape or envelope of the distribution of amplitudes of the field.

(ii) Analysis of the Power Spectral Density Function (PSDF)of the noise signal.

The PSDF is the function that describes the envelop or shape of the power spectrum of the field and is related to the autocorrelation function of a signal through the autocorrelation theorem. In this case, the PSDF is a measure of the time correlations of a signal. For example, a white Gaussian noise signal is characterized by a PSDF which is effectively constant over all frequencies and has a PDF with a Gaussian profile whose mean is zero (for a bi-polar signal). In Chapter 5, we investigate the relation between the power spectrum function and the autocorrelation function of the time series signal for Internet traffic data.

Many noise (stochastic) fields observed in nature have two fundamental characteristics:

(i) The field is statistical self-affine.

(ii)The PSDF is characterized by irrational power laws, i.e. $\frac{1}{|\omega|^q}$, where ω is the frequency and $q > 0$.

A good stochastic model is one that accurately predicts both the PDF and the PSDF of the data under analysis. It is also one which takes into account the fact that a stochastic field may be non-stationary. Note that there is no connection between the theoretical prediction and/or the experimental determination of the PDF and the PSDF, i.e. there is no direct relationship between the characteristics of the Fourier transform of stochastic field and the Fourier transform of the PDF of the same field.

In other words, for a given stochastic field, each of the PDF and PSDF cannot be computed directly one from each of other. There are two traditional approaches that are usually adopted in developing a stochastic noise model(to simulate the stochastic noise field):

(i) An approach which is based on modeling the PDF (or characteristic function) of the system (i.e. predicting the PDF theoretically if possible) so that we can design a pseudo random number generator that provide us with a stochastic field that is characteristic of the PDF.

(ii) An approach that is based on modelling the PSDF; the stochastic field is then simulated by filtering white Gaussian noise for example according to the PSDF model. In this work we consider the second approach.

3.3 RSF Signals as Solutions to Stochastic Fractional Differential Equations

Suppose that $s(t)$ is a signal that is given by the solution of the following Stochastic Fractional Differential (SFD) equation of the q^{th} order:

$$\frac{d^q}{dt^q}s(t) = n(t) \tag{3.3}$$

where q is a real (or complex) number such that $1 < q < 2$, and $n(t)$ is a noise function whose PSDF is constant. Note that when $q = 1$ and assuming that the function $n(t)$ is dimensionless, then the solution of this equation has dimension 1 (t being taken to be length), and when $q = 2$ the solution has dimension 2. Thus for $1 < q < 2$ the solution of this equation can be regarded in terms of a fractal signal with a fractional dimension between 1 and 2.

In what follows, we aim to show that the solution of equation (3.3) is one that is consistent with the fractal power law. As a consequence of this, we are able to quantify the PSDF of a fractal signal.

The first problem to consider is the method of solution required to solve equation (3.3). Fractional Calculus is not new [6, 12], but in general the use of fractional calculus to define RSF signals and stochastic processes has not been deeply explored. The principal arguments with regard to this issue have been explored by Blackledge [2].

There are many techniques to working with fractional derivatives but they all depend on the generalization of results associated with derivatives of integer order.

Here we consider the following definition of the q^{th} fractional derivative of the function $s(t)$, which is particular useful:

$$\frac{d^q}{dt^q}s(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}(i\omega)^q S(\omega)e^{i\omega t}d\omega \tag{3.4}$$

where $S(\omega)$ is the Fourier transform of $s(t)$ given by

$$S(\omega) = \int_{-\infty}^{\infty} s(t)e^{-i\omega t}dt$$

and ω is the frequency. By using this definition, the solution of equation (3.3) becomes

$$s(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}(i\omega)^{-q}N(\omega)e^{i\omega t}d\omega$$

where $N(\omega)$ is the Fourier transform of $n(t)$. Note that the solution is expressed in terms of the inverse Fourier transform of the two functions $(i\omega)^{-q}$ and $N(\omega)$. Application of the convolution theorem allows us to write this result in the form

$$s(t) = \int h(t-\tau)n(\tau)d\tau$$

where h is given by

$$h(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}(i\omega)^{-q}e^{i\omega t}d\omega$$

Substituting r for $i\omega$, $h(t)$ can be written in terms of the inverse Laplace transform of r^q. Since

$$\hat{L}(t^q) = \frac{\Gamma(q+1)}{r^{q+1}}, \qquad q > -1$$

where $\hat{L}$ is taken to denote the Laplace transform and Γ is the Gamma function

$$\Gamma(q) = \int_0^{\infty} x^{q-1}e^{-x}dx$$

we can write

$$t^q = \hat{L}^{-1}[\frac{\Gamma(q+1)}{r^{q+1}}]$$

or

$$\hat{L}^{-1}[\frac{1}{\tau^q}] = \frac{1}{\Gamma(q)} t^{q-1}$$

Thus, the solution to equation (3.3) can be written in the form

$$s(t) = \frac{1}{\Gamma(q)} \int_{-\infty}^{t} \frac{n(\tau)}{(t-\tau)^{1-q}} d\tau$$

This is the Liouville-Rieman transform and is an example of a fractional integral. The question now is whether this transform is consistent with the concept of statistical self-affinity. Consider the case when

$$\hat{s}(t) = \frac{1}{\Gamma(q)} \int_{-\infty}^{t} \frac{n(\lambda\tau)}{(t-\tau)^{1-q}} d\tau$$

where λ is a scaling parameter. Now, with $Z = \lambda\tau$, we obtain

$$\hat{s}(t) = \frac{1}{\lambda^q} \frac{1}{\Gamma(q)} \int_{-\infty}^{\lambda t} \frac{n(z)}{(\lambda t - z)^{1-q}} dz = \frac{1}{\lambda^q} s(\lambda t)$$

Both $\hat{s}(t)$ and $s(\lambda t)$ are stochastic signals but over different scales, ($n(t)$ being white noise at any scale) and so, although $\hat{s}(t) \neq s(\lambda t)$ we have

$$\Pr[s(\lambda t)] = \lambda^q \Pr[\hat{s}(t)]$$

which describes a statistical self-affine property of RSF signals.

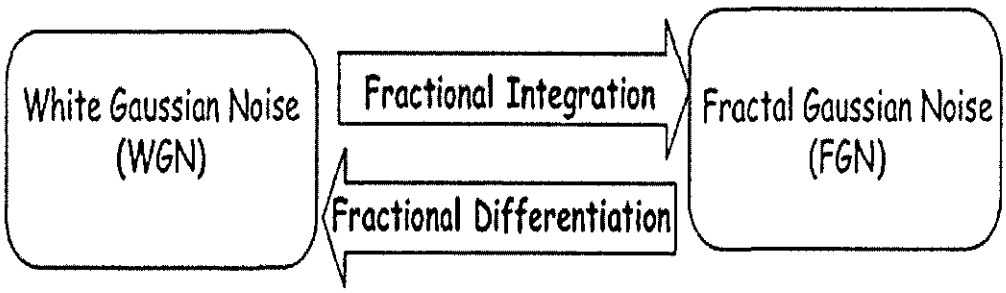

Figure 3.2: The relationship between White Noise and Fractal Noise.

3.3.1 Relationship between White Noise and Fractal noise

From the previous section, we note the relationship between white noise and fractal noise in terms of fractional differentiation and fractional integration as illustrated in Figure 3.2. This result allows us to generate a fractal signal with a given fractal dimension by means of a well-defined fractional integration of white noise. However, in general, a fractal signal is not stationary but varies, i.e. the fractal dimension, D, and also Fourier dimension q, changes with time. This reality leads us to consider a new type of integration/differentiation in which q becomes a function of time. This non-stationary process can be described by the equation

$$\frac{d^{q(t)}}{dt^{q(t)}}s(t) = n(t).$$

We can now define the forward and the inverse problems: the forward problem is, given the function $q(t)$ and a white noise signal $n(t)$, compute the fractal signal $s(t)$; the inverse problem is, given the fractal signal $s(t)$ estimate the function $q(t)$. The inverse problem can be approached in terms of the characteristic PSDF given by $P(\omega) = |S(\omega)|^2$ which is proportional to $|\omega|^{-2q}$ i.e.

$$P(\omega) \propto |\omega|^{-2q}$$

or

$$P(\omega) = \frac{c}{|\omega|^{\beta}}, \qquad \beta = 2q$$

where c is a constant of proportionality. Note, that c provides a measure of the 'energy' of the signal since from Rayleigh's theorem

$$\text{Energy} = \int_{-\infty}^{\infty} |s(t)|^2 dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} |S(\omega)|^2 d\omega = \frac{c}{2\pi}\int_{-\infty}^{\infty} \frac{1}{|\omega|^{2q}} d\omega$$

3.3.2 Digital Algorithms to Generate Fractal Noise and the Fractal Dimension

We introduce two principle algorithms that are required to generate a fractal noise signal by using a fractal noise model and to estimate the fractal dimension.

Algorithm I: Fractal Noise Generation

This algorithm is concerned with the computation of a discrete fractal noise signal s_i given that the parameter q is known and is as follows:

Step1: Generate a White Gaussian Noise (WGN), n_i, $i = 1, 2, 3, ...$

Step2: Calculate the Discrete Fourier Transform (DFT)of n_i to obtain N_i using a Fast Fourier Transform (FFT).

Step3: Apply the Fractal filter $\frac{1}{(i\omega_i)^{\frac{q}{2}}}$ on N_i , where $q = \frac{5-D}{2}$.

Step4: Calculate the Inverse DFT of the result using an (inverse) FFT to obtain the fractal signal s_i (real part).

Algorithm II: Estimation of the Fourier Dimension q

This algorithm is an inverse algorithm in which the fractal signal s_i is given from which we are required to compute q. A suitable approach to do this, which is consistent with the algorithm given above is to estimate q from the Power Spectrum of s_i whose expected form is

$$\hat{P}_i = \frac{c}{\omega_i^q}, \qquad i = 1, 2, 3, ...N/2 - 1$$

where c is a constant.

Here, we consider only the positive half space and exclude the DC component which is singular. The formula for estimating q from a fractal signal is given in Section 2.9 and the required algorithm to implement this inverse solution is as follows [3]:

Step1: Take the fast Fourier transform (FFT) of the fractal signal.

Step2: Calculate the Power Spectrum (PS),
$P_i(\omega) = Re(\omega_i)^2 + Im(\omega_i)^2$, where $Re(\omega_i)$ and $Im(\omega_i)$ are real and imaginary parts of the spectrum, respectively.

Step3: Compute the estimated value of q using the following formula which was derived in Section (2.9)

$$q = \frac{N\sum_{i=1}^{N}(\ln\omega_i)\ln P(\omega_i) - (\sum_{i=1}^{N}\ln\omega_i)\sum_{i=1}^{N}\ln P(\omega_i)}{(\sum_{i=1}^{N}\ln\omega_i)^2 - N\sum_{i=1}^{N}(\ln\omega_i)^2}$$

The mechanisms of generating a fractal signal with a given Fourier dimension, q, and then estimating such parameter from the generated signal are shown in Figure 3.4 and Figure 3.5, respectively.

Figure 3.3 shows the graph of a generated fractal signal with its probability distribution function (a&b), and graph of the theoretical and the empirical power spectrum density function (c&d).

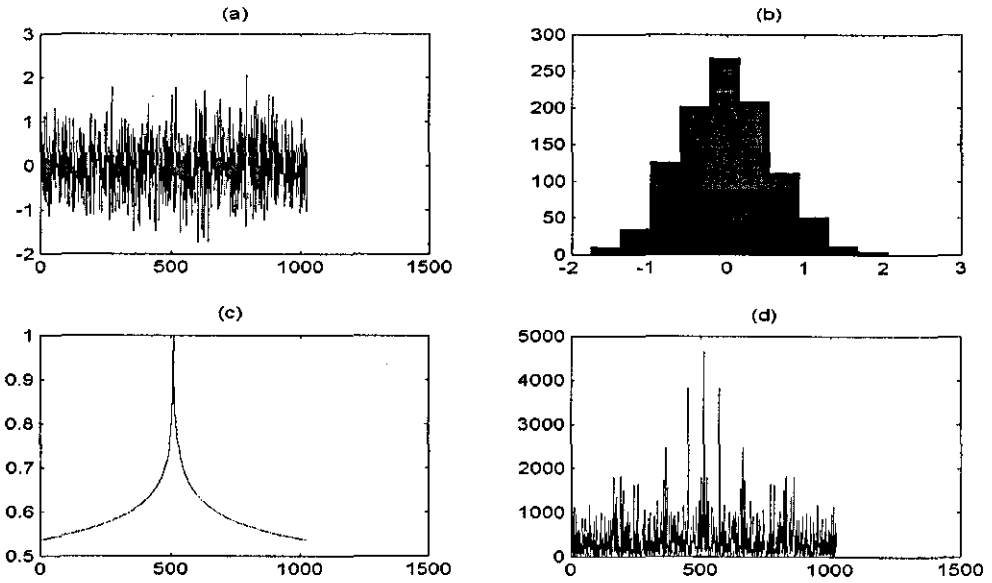

Figure 3.3: (a)&(b) Fractal signal with q=0.1 and its PDF, (c)&(d) its theoretical and empirical PS.

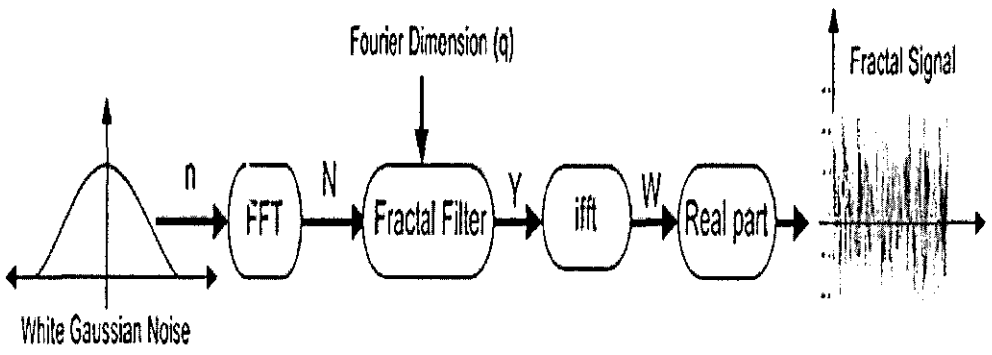

Figure 3.4: Generation of a Fractal Noise Signal.

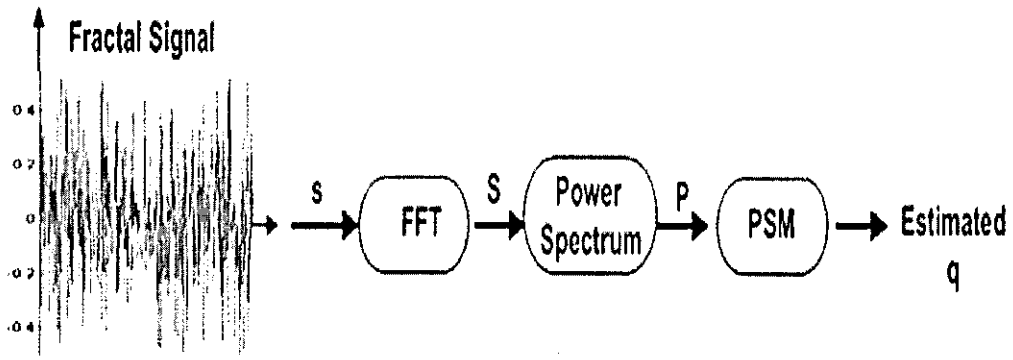

Figure 3.5: Estimation of the Fractal Parameter, q (Fourier Dimension).

3.4 Review of Fractal Modulation

Embedding information in data whose properties and characteristics resemble the background noise of a transmition system is of particular interest in covert digital communications. In this section, we review a technique which was introduced through the work of [6], which is based on embedding a bit stream in fractal noise by modulating the fractal dimension of a fractal noise generator and reconstructing the bit streams in the presence of additive noise assumed to be introduced during the transmission phase.

This form of 'embedding information in noise' is of value in the transmission of information in situations when communication traffic needs to hidden in a covert sense by coupling an increase in the background noise of a given area with appropriate disinformation.

3.4.1 Secure Digital Communications

A digital communications systems is one that is based on transmitting and receiving bit streams, this could include a text file for example. The basic processes involved are as follows: (i) a digital signal is obtained from sampling an analogue signal derived from some speech and/or video system; (ii) this signal (floating point stream) is converted into a binary signal consisting of 0s and 1s (bit stream); (iii) the bit stream is then modulated and transmitted; (iv) at reception, the transmitted signal is demodulated to recover the transmitted bit stream; (v) the (floating point) digital signal is reconstructed. Digital to analogue conversion may then be required depending on the type of technology being used.

In the case of sensitive information, an additional step is required between stages (ii) and (iii) above where the bit stream is coded according to some classified algorithm. Appropriate decoding is then introduced between stages (iv) and (v) with suitable pre-processing to reduce the effects of transmission noise for example which introduces bit errors. The bit stream coding algorithm is typically based on a pseudo random number generator or nonlinear maps in chaotic regions of their phase spaces (chaotic number generation).

The modulation technique is typically either Frequency Modulation or Phase Modulation. Frequency modulation involves assigning a specific frequency to each 0 in the bit stream and another higher (or lower) frequency to each 1 in the stream. The difference between the two frequencies is minimized to provide space for other channels within the available bandwidth. Phase modulation involves assigning a phase value (0, $\pi/2$, π, $3\pi/2$) to one of four possible combinations that occur in a bit stream (i.e. 00, 11, 01 or 10).

Scrambling methods can be introduced before binarization. A conventional approach to this is to distort the digital signal by adding random numbers to the out-of-band components of its spectrum. The original signal is then recovered by lowpass filtering. This approach requires an enhanced bandwidth but is effective in the sense that the signal can be recovered from data with a relatively low signal-to-noise ratio. 'Spread spectrum' or 'frequency hopping' is used to spread the transmitted (e.g. frequency modulated) information over many different frequencies. Although spread spectrum communications use more bandwidth than necessary, by doing so, each communications system avoids interference with another because the transmissions are at such minimal power, with only spurts of data at any one frequency. The emitted signals are so weak that they are almost imperceptible above background noise. This feature results in an added benefit of spread spectrum which is that eavesdropping on a transmission is very difficult and in general, only the intended receiver may ever known that a transmission is taking place, the frequency hopping sequence being known only to the intended party. Direct sequencing, in which the transmitted information is mixed with a coded signal, is based on transmitting each bit of data at several different frequencies simultaneously, with both the transmitter and receiver synchronized to the same coded sequence. More sophisticated spread spectrum techniques include hybrid ones that leverage the best features of frequency hopping and direct sequencing as well as other ways to code data. These methods are particularly resistant to jamming, noise and multipath anomalies, a frequency dependent effect in which the signal is reflected from objects in urban and/or rural environments and from different atmospheric layers, introducing delays in the transmission that can confuse any unauthorized reception of the transmission.

The purpose of Fractal Modulation is to try and make a bit stream 'look like' transmission noise (assumed to be fractal). The technique considered here focuses on the design of algorithms which encode a bit stream in terms of two fractal dimensions that can be combined to produce a fractal signal characteristic of transmission noise. Ultimately, fractal modulation can be considered to be an alternative to frequency modulation although requiring a significantly greater bandwidth for its operation.

However, fractal modulation could relatively easily be used as an additional pre-processing security measure before transmission. The fractal modulated signal would then be binarized and the new bit stream fed into a conventional frequency modulated digital communications system albeit with a considerably reduced information throughput for a given bit rate. The problem is as follows: given an arbitrary binary code, convert it into a non-stationary fractal signal by modulating the fractal dimension in such a way that the original binary code can be recovered in the presence of additive noise with minimal bit errors.

3.4.2 Fractal Modulation and Demodulation

The technique considered here focuses on the design of the algorithms which encode a bit stream in terms of two fractal dimensions that can be combined to produce a fractal signal that is characteristic of transmission noise. The transmission noise could include that associated with a range of frequency channels including the radio and microwave (mobile telecommunications for example) ranges.

Instead of working in terms of fractal dimension, D, we shall consider the Fourier dimension, q, which is related to D by

$$q = \frac{5-2D}{2}, \qquad 1 < D < 2, \qquad 0 < q < 1.$$

Fractal Modulation

Consider a stream of binary bits '...10110011...' , of length L where we allocate q_0 to bit=0 and q_1 to bit=1, where $q_0 < q_1$.

Compute a fractal signal of length N for each bit in the stream using values for q_0 or q_1.

Concatenate the resulting signals to produce a contiguous stream of non-stationary fractal noise.

Fractal Demodulation

The problem of reconstructing the original bit stream, is achieved as follows:

Estimate the parameter q via the power spectrum method, as introduced in Chapter 2, using a covenential moving window of size N to provide a stream of estimated values of q, i.e. $\hat{q}_i$, $i = 1, 2, 3, .., L$.

Reconstruct the bit stream from the following algorithm:

if $\hat{q}_i \leq \Delta$ then bit=0

if $\hat{q}_i > \Delta$ then bit=1,

where $\Delta = \frac{1}{2}(q_1 + q_0)$, the thresholding point.

3.5 Experimental Results

In this section we introduce some results that have been obtained through application of the MATLAB code developed for investigating Fractal Modulation (see Appendix A) and in particular, the estimation of the Fourier dimension q.

The results given in Table 3.1 and Table 3.2 are non-inclusive of an applied noise-to-signal ratio (i.e. NSR = 0) whereas in Table 3.3 result are presented with different NSR (for additive noise) for the fractal signal, applied before computing an estimate for q.

3.5.1 Results of Estimating q Without Additive Noise

Here, we have taken the fractal signal to be of size $N = 1024$ with different seeds, say seed=5, 10, 15, 20, and different values of the parameter q.

Comments on Tables

We note from the tables given that the estimated values of q alter according to the changes in its exact value and also change in seed value. As the value of the seed increases, the estimation error of the parameter also increase; this means that it is convenient to use small values of the seed (such 5 or 10 for example) for the purpose of encoding binary bits. The question that may arise is how can we control the effect of the seed value on the estimation error? The other principal result is that a fixed increment in the estimated values of the parameter q is of the order of 0.04 which is double the increment in the exact values of q.

Exact(q)	Seed=5		Seed=10		Seed=15	
	Est.(q)	Error(q)	**Est.(q)**	Error(q)	**Est.(q)**	Error(q)
0.06	0.04	33%	0.07	21%	0.02	58%
0.08	0.08	0%	0.11	41%	0.04	18%
0.10	0.12	20%	0.15	52%	0.10	40%
0.12	0.16	33%	0.19	60%	0.14	20%
0.14	0.20	42%	0.23	66%	0.18	32%
0.16	0.24	50%	0.27	70%	0.22	40%
0.18	0.28	55%	0.31	73%	0.26	47%
0.20	0.32	60%	0.35	76%	0.30	52%

Table 3.1: Estimated values of q, with different seeds.

Exact(q)	Seed=20		Seed=24		Seed=25	
	Est.(q)	Error(q)	**Est.(q)**	Error(q)	**Est.(q)**	Error(q)
0.06	0.08	44%	0.02	58%	0.06	10%
0.08	0.12	58%	0.06	18%	0.10	33%
0.10	0.16	66%	0.10	5 %	0.14	46%
0.12	0.20	72%	0.14	21%	0.18	55%
0.14	0.24	76%	0.18	32%	0.22	61%
0.16	0.28	79%	0.22	41%	0.26	66%
0.18	0.32	55%	0.26	47%	0.30	70%
0.20	0.36	60%	0.30	53%	0.34	73%

Table 3.2: Estimated values of q, with different seeds.

For example, Table 3.1 shows that with the seed=5, where the exact value of the parameter q starts from 0.06 and increases incrementally by 0.02 to 0.20, the corresponding estimated value start from 0.04 and increases incrementally by 0.04 (double of increment in the exact value) until 0.32.

3.5.2 Results of Estimating q With Additive Noise

Here, we added different percentages of noise (white noise) to the generated fractal signal before estimating the parameter q, where $q = 0.20$.

NSR	Seed=5, Seed of NSR = 5		Seed=5, Seed of NSR = 20	
	Est.(q)	Error(q)	**Est.(q)**	Error(q)
0.00	0.120	20%	0.12	20%
0.05	0.121	21%	0.126	26%
0.10	0.118	18%	0.125	24%
0.15	0.106	6 %	0.116	16%
0.20	0.090	1 %	0.102	2 %
0.25	0.070	24%	0.074	25%
0.30	0.057	42%	0.075	24%
0.35	0.058	41%	0.067	32%
0.40	0.610	39%	0.058	41%
0.45	0.045	54%	0.042	57%
0.50	0.041	58%	0.03	69%

Table 3.3: Estimated values of q with different values of NSR (exact $q = 0.10$).

The results in Table 3.3 show that as the NSR increases, the error in estimation of the parameter q also increase as is expected. Also, when we use the same seed value for the fractal signal and for the added noise signal, the estimation error of q is small compared with the estimation error associated with the case when the seed value is changed.

3.5.3 Illustrative Example

In this section we introduce an example for coding and decoding a bit streams.

Encoding a Binary Bit Stream

Suppose that the binary bit array to be encode is bn= '10110010', and we use initial values $N = 1024$ and $seed = 5$ with parameters $q_0 = 0.10$ and $q_1 = 0.20$.

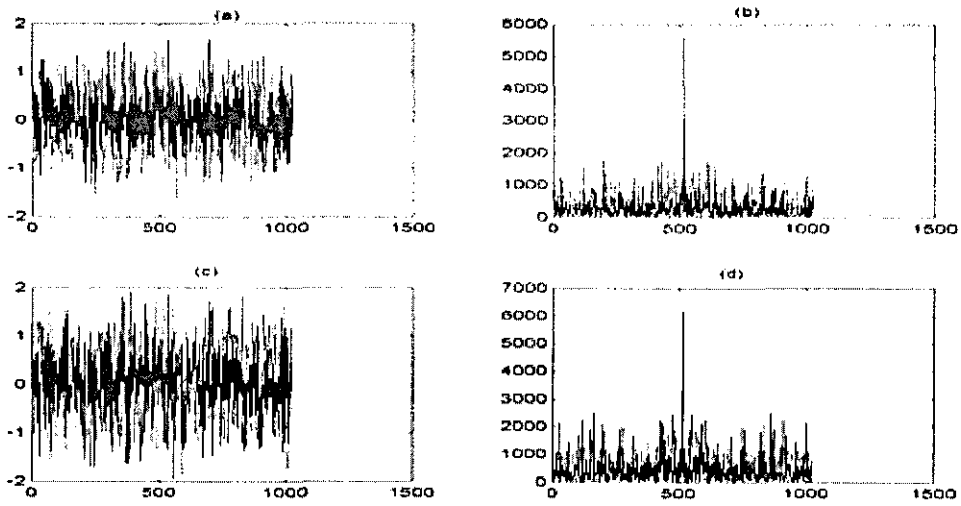

Figure 3.6: (a)&(b)Fractal signal with NSR=0 and its PS , (c)&(d) Fractal signal with NSR=0.25, and its PS.

Figure 3.7 shows a non-stationary contiguous stream of fractal modulated signals and we note that the texture of the whole signal varies according to changes in the bit stream: from 1 to 0 and 0 to 1.

Decoding the Binary Bit Stream

Assuming that we have used the previous modulated signal of length 8192, a window of length N=1024 is moved over the signal 1024 points at a time and we estimate the value of q from each segment. We therefore obtain an array of length L=8 giving the estimated values of the parameter q as follows: $\hat{q}_i = 0.32, 0.12, 0.32, 0.32, 0.12, 0.12, 0.32, 0.12$. The value of the threshold is $\Delta = \frac{q_1+q_0}{2} = 0.15$ and if we compare the values of $\hat{q}_i$ with the $\Delta = 0.15$ according to the algorithm given above, then we recover the original binary bit stream, with no error.

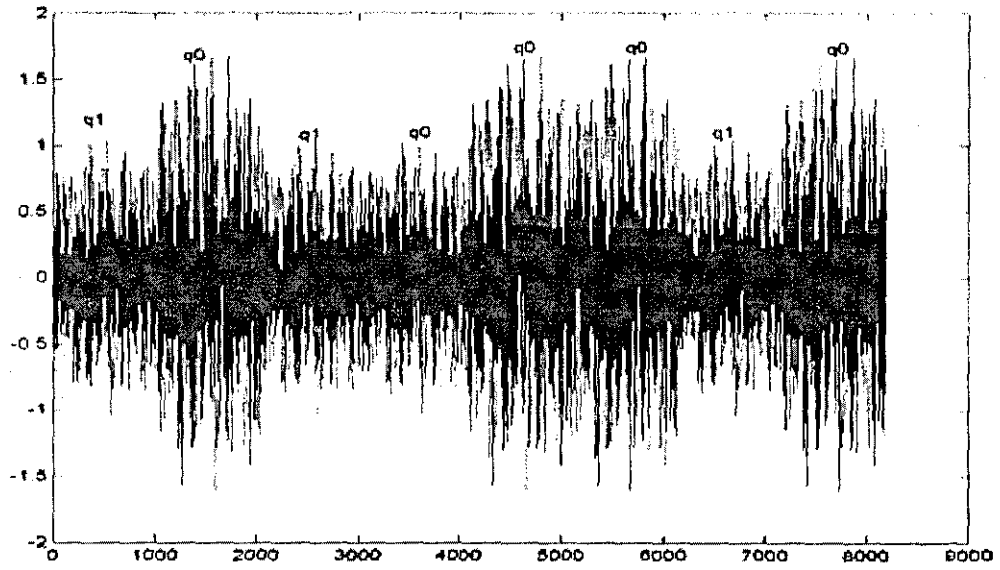

Figure 3.7: Non-Stationary contiguous stream of fractal modulated signals, with $q_0 = 0.10$ and $q_1 = 0.20$

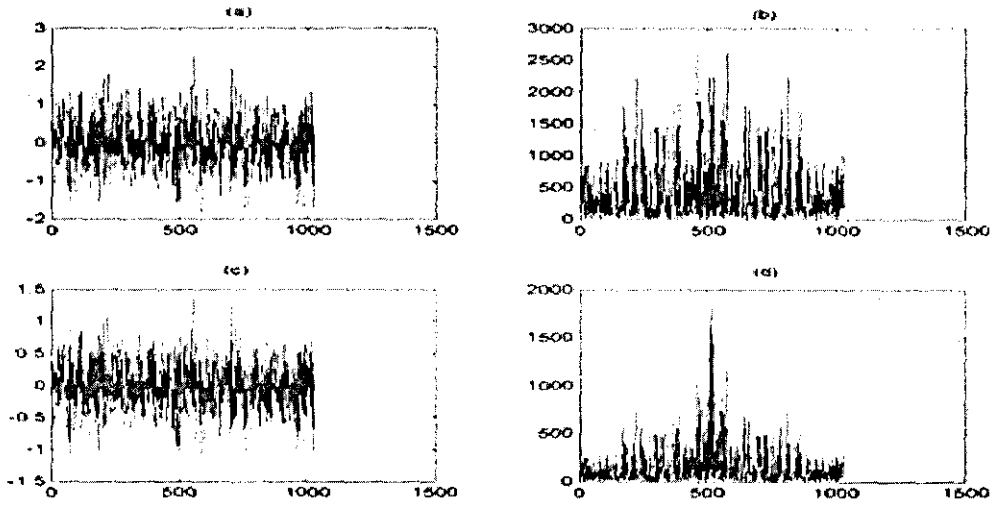

Figure 3.8: (a)&(b) Fractal signal with $q = 0.20$ and its empirical PS, (c)&(d) Fractal signal with $q = 0.10$ and its empirical PS

3.6 Multi-Fractal Modulation

Modulation techniques for digital communications systems are typically based on either Frequency Modulation or Phase Modulation. As discussed in Section 3.4.1, frequency modulation involves assigning a specific frequency to each 0 in the bit stream and another higher (or lower) frequency to each 1 in the stream, the difference between the two frequencies being minimized to provide space for other channels within the available bandwidth. Phase modulation involves assigning a phase value (0, $\pi/2$, π, $3\pi/2$) to one of four (and only four) possible combinations that occur in a bit stream (i.e. 00, 01, 10, 11).

In this thesis, we introduce a new technique to modulate binary bit streams, which can be considered as an extension to the fractal modulation technique discussed in the previous section, i.e. multi-fractal modulation. In this sense, fractal modulation, as discussed above is analogous to conventional frequency modulation and multi-fractal modulation, as discussed here, is analogous to phase modulation. The difference is that both fractal and multi-fractal modulation attempt to hide information in background noise, thus making the transmission entirely covert.

In multi-fractal modulation we try to embed a binary bit stream in background fractal noise, by modulating the values of one or two parameters using a generalized random scaling fractal model to be introduced in the following section. Ultimately, multi-fractal modulation can be considered to be an alternative approach to phase modulation although requiring a significantly greater bandwidth for its operation. Thus, for civilian purpose, which requires the use of minimal bandwidths, these techniques are not appropriate but for military

communications (where bandwidth is not so critically constrained) fractal and multi-fractal modulation techniques have a number of applications.

3.6.1 Generalized Random Scaling Fractal (GRSF) Model

Although statistical self-similarity and self-affinity are properties of many signals found in nature, the basic PSDF model $P(\omega) = \frac{1}{\omega^{2q}}$, which is associated with a random fractal signal, is not appropriate to all noise types and/or characteristic of their behavior as a whole. Most signals do have a high frequency decay for which the RSF model is appropriate but the complete power spectrum may have characteristics for which a simple power law (i.e $\frac{1}{\omega^{2q}}$) is not appropriate. This has led to the development of spectral partitioning algorithms which attempt to extract the most appropriate part of the spectrum for which the $\frac{1}{\omega^{2q}}$ power law applies.

Developing theoretically valid models for the spectral characteristics (PSDF)and/or the PDFs of stochastic fields is one of the principle aims of statistical mechanics. Ideally, what we require is a profile for the PSDF which characterizes a wider variety of PSDFs of which the the ω^{-2q} law is a special case. Following [1], a more general stochastic model can be proposed where the PSDF of a stochastic signal is assumed to be of the form

$$P(\omega) = c.\frac{\omega^{2g}}{(\omega_0^2 + \omega^2)^q} \tag{3.5}$$

where g (Numerator parameter) and q (Denominator parameter) are positive real numbers, c is a scaling parameter and ω_0 is a characteristic frequency parameter [2]. This model is a generalization of the RSF model discussed above

and thus is called the Generalized Random Scaling Fractal (GRSF) model. Here, the two parameters g and q can be consider to be Fourier dimensions which together with the parameter ω_0 provides a way of 'shaping' the PSDF of a stochastic field. Note that this model reduces to the PSDF for a fractal signal when $g = 0$, $\omega_0 = 0$ and $1 < q < 2$.

3.6.2 Basic Properties

The PSDF given in equation (3.5) has a maximum value at a certain frequency $\omega_m \in (0, \infty)$. To find this value, for simplicity, we write the PSDF in logarithmic form, i.e.

$$\begin{aligned}\ln P(\omega) &= \ln[c.\frac{\omega^{2g}}{(\omega_0^2 + \omega^2)^q}] \\ &= \ln c + 2g \ln \omega - q \ln(\omega_0^2 + \omega^2)\end{aligned}$$

and then find ω when

$$\frac{dP(\omega)}{d\omega} = 0$$

i.e.

$$\frac{d \ln P(\omega)}{d\omega} = \frac{1}{P(\omega)}.\frac{dP(\omega)}{d\omega} = \frac{2g}{\omega} - \frac{2\omega q}{\omega_0^2 + \omega^2} = 0$$

This implies that

$$\frac{dP(\omega)}{d\omega} = [\frac{2g}{\omega} - \frac{2\omega q}{\omega_0^2 + \omega^2}]P(\omega) = 0$$

and since

$$\begin{aligned}P(\omega) \neq 0, \qquad \forall \omega \in (0, \infty) \\ [\frac{2g}{\omega} - \frac{2\omega q}{\omega_0^2 + \omega^2}] = 0\end{aligned}$$

or

$$\frac{g}{\omega} = \frac{\omega q}{\omega_0^2 + \omega^2}$$

Thus

$$\omega^2 q = g\omega_0^2 + g\omega^2$$

and

$$\omega^2(q-g) = g\omega_0^2$$

Hence, the PSDF attain its maximum value at

$$\omega_m = \sqrt{\frac{g\omega_0^2}{q-g}} = \omega_0\sqrt{\frac{g}{q-g}}$$

such that $g < q$ but when $g \geq q$ then the maximum value of the PSDF does not exist and

$$\lim_{\omega\to\infty} P(\omega) = \infty.$$

If we apply the value ω_m in $P(\omega)$ we can obtain the maximum value

$$\begin{aligned} P(\omega_m) &= P(\omega_0\sqrt{\frac{g}{q-g}}) \\ &= c(\omega_0^2\frac{g}{q-g})^g(\omega_0^2 + \omega_0^2\frac{g}{q-g})^{-q} \\ &= c\omega_0^{2g}(\frac{g}{q-g})^g\omega_0^{-2q}(1+\frac{g}{q-g})^{-q} \\ &= c\omega_0^{2g-2q}(\frac{g}{q-g})^g(\frac{q}{q-g})^{-q} \\ &= c\omega_0^{2(g-q)}(\frac{g^g}{q^q})(q-g)^{q-g}. \end{aligned}$$

Beyond this point, the generalized PSDF decays and its asymptotic form is dominated by a ω^{-2q} power law which is consistent with RSF signals and many noise types at the high frequency end of their power spectra. At low frequencies, the spectrum is characterised by the term $(i\omega)^{2g}$.

3.7 Analysis

Given the generalized PSDF in equation (3.5), the complex spectrum equation of the noise signal $F(\omega)$ can be written as

$$F(\omega) = H(\omega).N(\omega)$$

where as before $N(\omega)$ is the complex spectrum of white noise. Here, the term 'white noise' is defined conventionally as noise whose PSDF is constant and $H(\omega)$ is the transfer function (ignoring scaling) given by

$$H(\omega) = \frac{(i\omega)^g}{(\omega_0 + i\omega)^q}$$

The noise function $f(x)$ is then given by

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega).N(\omega)e^{i\omega x} d\omega.$$

It is interesting to analyse this result with the aim of establishing the transform of $n(x)$ and so obtain $f(x)$. If we consider the definition of fractional derivative in terms of the inverse Fourier transform of $(i\omega)^g$, then using the convolution theorem we can write (for independent variable x)

$$f(x) = \int h(x-y)\frac{d^g}{dy^g}n(y)dy$$

where

$$h(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{1}{(\omega_0 + i\omega)^q} e^{i\omega x} d\omega.$$

Substitution of p for $i\omega$ allows us to write this result in terms of the inverse Laplace transform, i.e.

$$h(x) = L^{-1}[\frac{1}{(\omega_0 + p)^q}].$$

Since

$$L[x^q e^{-\omega_0 x}] = \frac{\Gamma(q+1)}{(\omega_0 + p)^{q+1}}$$

where, $q > -1$ and $Re(p + \omega_0) > 0$ it follows that

$$h(x) = \frac{1}{\Gamma(q)} \frac{e^{\omega_0 x}}{x^{1-q}}.$$

Hence $f(x)$ can be written in terms of the fractional integral transform:

$$f(x) = \frac{1}{\Gamma(q)} \int_{-\infty}^{x} \frac{e^{-\omega_0(x-y)}}{(x-y)^{1-q}} \frac{d^g}{dy^g} n(y) dy.$$

The scaling characteristics of this transform can be investigated by considering the function

$$\hat{f}(x; \omega_0) = \frac{1}{\Gamma(q)} \int_{-\infty}^{x} \frac{e^{-\omega_0(x-y)}}{(x-y)^{1-q}} . \frac{d^g}{dy^g} n(\lambda y) dy$$

$$= \frac{\lambda^g}{\lambda^q} . \frac{1}{\Gamma(q)} \int_{-\infty}^{\infty} \frac{e^{\frac{-\omega_0}{\lambda}(\lambda x - z)}}{(\lambda x - z)^{1-q}} \frac{d^g}{dz^g} . n(z) dz$$

$$= \frac{\lambda^g}{\lambda^q} f(\lambda x; \frac{\omega_0}{\lambda})$$

with substitution of z for λy.

Hence, the scaling relationship for this model is

$$Pr[f(\lambda x, \frac{\omega_0}{\lambda})] = \lambda^{q-g} . Pr[f(x, \omega_0)]$$

where $Pr[.]$ denotes the PDF. Here, as we scale x by λ, the characteristic frequency ω_0 is scaled by $\frac{1}{\lambda}$ a result that is in some sense consistent with the scaling property of the Fourier transform

$$f(\lambda x) \Leftrightarrow \frac{1}{\lambda} F(\frac{\omega}{\lambda}).$$

The interpretation of this result is that as we zoom in the signal $f(x)$ the distribution of amplitudes (i.e., the probability density function, PDF) remains the same subject to a scaling factor $\lambda^{(g-q)}$ and the characteristic frequency of the signal increase by a factor $\frac{1}{\lambda}$

Figure 3.9 shows a fractal signal of size 1024 points, generated using the GRSF model, with its PDF and another fractal signal of size 512 points with its PDF, where the second fractal signal is a half zoom-in of the first signal. Clearly, the PDF of the second signal is similar to the PDF of the first signal. This figure illustrates the characteristic of self-affinity in the two signals where the distribution of amplitudes for the two signals are the same over (two) different scales.

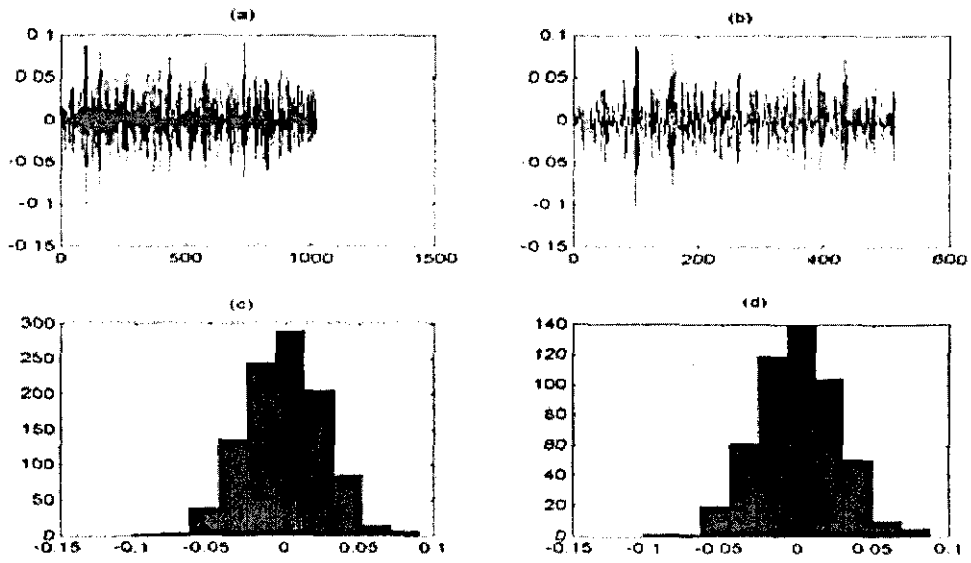

Figure 3.9: (a)&(c) Fractal signal of size N=1024 and its PDF, (b)&(d) Fractal signal of size N=512 and its PDF.

3.7.1 Parameter Estimation for the GRSF Model

The results discussed previously for the GRSF model reduce to the 'normal' theory of RSF signals when $g = 0$ and $\omega_0 = 0$. However, GRSF model gives a much greater degree of flexibility in terms of characterizing the PSDFs of many noise types, nearly all of which have some degree of statistical self-affinity, and PSDFs with power laws of irrational form. In terms of using this model to characterize signal texture, we consider the case where a suitable combination (some cluttering algorithm) of the parameters g, q,and c is taken to be a measure of texture, in particular, g and/or q and their product for example. In each case, we are required to obtain estimates for these parameters associated with the data f_i, $i = 1, 2, 3, ..., N$.

The general four-parameter problem is not easy to solve, primarily because of difficulties in linearizing $P(\omega)$ with respect to ω_0. However, suppose that a good estimate of ω_0 can be obtained, then we can compute estimates for g, q and c using a standard least squares method by constructing a logarithmic least squares estimate in the usual way. Suppose that we have a digital fractal signal f_i, $i = 1, 2, 3, ..., N$ whose expected power spectrum density function is described by a GRSF model of the form

$$\hat{P}(\omega) = c.\frac{\omega^{2g}}{(\omega_0^2 + \omega^2)^q}$$

The power spectrum method (PSM) requires the use of the least square principle in order to find the best estimation for the parameters g, q, ω_0 and c.

3.7.2 Power Spectrum Method for Estimating g, q and c

As discussed earlier, the general four-parameter problem is not easily to solve analytically, primarily because of difficulties in linearizing $P(\omega)$ with respect to ω_0. Thus, we shall concentrate only on estimating the three parameters g, q and c. Assuming that a good estimate for ω_0 can be obtained, suppose P_i is the empirical value of the power spectrum of the data f_i, $i = 1, 2, 3, ..., N$ where

$$P_i = a_i^2 + b_i^2$$

$$a_i = Real(F_i),$$

$$b_i = Imag(F_i),$$

$$F_i = FFT(f_i), \quad i = 1, 2, 3, ..., N$$

We shall consider two cases, i.e. when $c = 1$ and when $c \neq 1$.

Case 1: $c = 1$

Suppose that we wish to fit P_i to the expected form of a fractal power spectrum $\hat{P}$, where

$$\hat{P}_i = \frac{\omega_i^{2g}}{(\omega_0^2 + \omega_i^2)^q}, \qquad \omega_i > 0.$$

We construct the logarithmic least square estimate in the usual way, i.e.,

$$\ln \hat{P}_i = 2g \ln \omega_i - q \ln(\omega_0^2 + \omega_i^2)$$

and consider the error

$$E(g, q) = \sum_{j=0}^{N} (\ln P_i - \ln \hat{P}_i)^2$$

$$= \sum_{j=0}^{N} [\ln P_i - 2g \ln \omega_i + q \ln(\omega_0^2 + \omega_i^2)]^2$$

which is minimum when

$$\frac{\partial E}{\partial g} = 0, \qquad \frac{\partial E}{\partial q} = 0.$$

Then

$$\frac{\partial E}{\partial g} = -4 \sum_{j=0}^{N} [\ln P_i - 2g \ln \omega_i + q \ln(\omega_0^2 + \omega_i^2)] \ln \omega_i = 0$$

$$= \sum_{j=0}^{N} (\ln P_i)(\ln \omega_i) - 2g \sum_{j=0}^{N} (\ln \omega_i)^2 + q \sum_{j=0}^{N} \ln(\omega_0^2 + \omega_i^2)(\ln \omega_i) = 0$$

$$\Rightarrow 2g \sum_{j=0}^{N} (\ln \omega_i)^2 - q \sum_{j=0}^{N} \ln(\omega_0^2 + \omega_i^2)(\ln \omega_i) = \sum_{j=0}^{N} (\ln P_i)(\ln \omega_i) \qquad (3.6)$$

Similarly,

$$\frac{\partial E}{\partial q} = 2 \sum_{j=0}^{N} [\ln P_i - 2g \ln \omega_i + q \ln(\omega_0^2 + \omega_i^2)] \ln(\omega_0^2 + \omega_i^2) = 0$$

$$= \sum_{j=0}^{N} (\ln P_i)(\ln(\omega_0^2 + \omega_i^2) - 2g \sum_{j=0}^{N} (\ln \omega_i) \ln(\omega_0^2 + \omega_i^2) + q \sum_{j=0}^{N} (\ln(\omega_0^2 + \omega_i^2))^2 = 0$$

$$\Rightarrow 2g \sum_{j=0}^{N} (\ln \omega_i) \ln(\omega_0^2 + \omega_i^2) - q \sum_{j=0}^{N} (\ln(\omega_0^2 + \omega_i^2))^2 = \sum_{j=0}^{N} (\ln P_i) \ln(\omega_0^2 + \omega_i^2) \qquad (3.7)$$

Put,

$$a_{11} = 2 \sum_{j=0}^{N} (\ln \omega_i)^2, \qquad a_{12} = - \sum_{j=0}^{N} \ln(\omega_0^2 + \omega_i^2)(\ln \omega_i)$$

$$a_{21} = 2\sum_{j=0}^{N}(\ln\omega_i)\ln(\omega_0^2+\omega_i^2), \qquad a_{22} = -\sum_{j=0}^{N}(\ln(\omega_0^2+\omega_i^2))^2$$

$$b_1 = \sum_{j=0}^{N}(\ln P_i)(\ln\omega_i), \qquad b_2 = \sum_{j=0}^{N}(\ln P_i)(\ln(\omega_0^2+\omega_i^2)$$

We can write equation (3.6) and equation (3.7) in a more abbreviated form as

$$a_{11}g + a_{12}q = b_1$$

$$a_{21}g + a_{22}q = b_2$$

and these equations can be written in matrix form as

$$A.Q = B$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$Q = \begin{pmatrix} g \\ q \end{pmatrix} \quad ,$$

and

$$B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Thus, the vector of estimated parameters is

$$Q = A^{-1}.B \tag{3.8}$$

given that A is not a singular matrix.

Case 2: $c \neq 1$

As with the previous case, the PSDF will have the form

$$P(\omega) = c.\frac{\omega^{2g}}{(\omega_0^2 + \omega^2)^q}$$

and the logarithmic form of PSDF is

$$\ln P_i = \ln c + 2g \ln \omega_i - q \ln(\omega_0^2 + \omega_i^2)$$

with $C = \ln c$ and so

$$\ln P_i = C + 2g \ln \omega_i - q \ln(\omega_0^2 + \omega_i^2).$$

Here, we need to find the estimated values of the three parameters g, q and C by minimizing the value of

$$E(g, q, C) = \sum_{j=0}^{N} [\ln P_i - \ln \hat{P}_i]^2$$

or

$$E(g, q, C) = \sum_{j=0}^{N} [\ln P_i - C - 2g \ln \omega_i + q \ln(\omega_0^2 + \omega_i^2)]^2.$$

This achieved by finding the solutions to the three equations

$$\frac{\partial E}{\partial g} = 0, \quad \frac{\partial E}{\partial q} = 0 \quad \textit{and} \quad \frac{\partial E}{\partial C} = 0.$$

Then

$$\frac{\partial E}{\partial g} = -4 \sum_{j=0}^{N} [\ln P_i - C - 2g \ln \omega_i + q \ln(\omega_0^2 + \omega_i^2)] \ln \omega_i = 0$$

$$= \sum_{j=0}^{N} (\ln P_i)(\ln \omega_i) - C \sum_{j=0}^{N} \ln \omega_i - 2g \sum_{j=0}^{N} (\ln \omega_i)^2 + q \sum_{j=0}^{N} \ln(\omega_0^2 + \omega_i^2)(\ln \omega_i) = 0$$

$$\Rightarrow 2g\sum_{j=0}^{N}(\ln\omega_i)^2 - q\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2)(\ln\omega_i) + C\sum_{j=0}^{N}\ln\omega_i = \sum_{j=0}^{N}(\ln P_i)(\ln\omega_i) \tag{3.9}$$

$$\frac{\partial E}{\partial q} = 2\sum_{j=0}^{N}[\ln Pi - C - 2g\ln\omega_i + q\ln(\omega_0^2+\omega_i^2)]\ln(\omega_0^2+\omega_i^2) = 0$$

$$= \sum_{j=0}^{N}(\ln P_i)\ln(\omega_0^2+\omega_i^2) - C\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2)$$

$$-2g\sum_{j=0}^{N}(\ln\omega_i)(\ln(\omega_0^2+\omega_i^2) + q\sum_{j=0}^{N}(\ln(\omega_0^2+\omega_i^2))^2 = 0$$

$$\Rightarrow 2g\sum_{j=0}^{N}(\ln\omega_i)\ln(\omega_0^2+\omega_i^2) - q\sum_{j=0}^{N}(\ln(\omega_0^2+\omega_i^2))^2 + C\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2) = \sum_{j=0}^{N}(\ln P_i)\ln(\omega_0^2+\omega_i^2) \tag{3.10}$$

$$\frac{\partial E}{\partial C} = -2\sum_{j=0}^{N}[\ln P_i - C - 2g\ln\omega_i + q\ln(\omega_0^2+\omega_i^2)] = 0$$

$$= \sum_{j=0}^{N}\ln P_i - C\sum_{j=0}^{N}1 - 2g\sum_{j=0}^{N}\ln\omega_i + q\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2) = 0$$

$$\Rightarrow 2g\sum_{j=0}^{N}\ln\omega_i - q\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2) + CN = \sum_{j=0}^{N}\ln P_i \tag{3.11}$$

put

$$a_{11} = 2\sum_{j=0}^{N}(\ln\omega_i)^2, \qquad a_{12} = -\sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2)(\ln\omega_i), \qquad a_{13} = \sum_{j=0}^{N}\ln\omega_i,$$

$$a_{21} = 2\sum_{j=0}^{N}(\ln\omega_i)\ln(\omega_0^2+\omega_i^2), \qquad a_{22} = -\sum_{j=0}^{N}(\ln(\omega_0^2+\omega_i^2))^2, \quad a_{23} = \sum_{j=0}^{N}\ln(\omega_0^2+\omega_i^2),$$

$$a_{31} = 2\sum_{j=0}^{N} \ln \omega_i, \qquad a_{32} = -\sum_{j=0}^{N} \ln(\omega_0^2 + \omega_i^2), \qquad a_{33} = N$$

$$b_1 = \sum_{j=0}^{N} (\ln P_i)(\ln \omega_i), \qquad b_2 = \sum_{j=0}^{N} (\ln P_i) \ln(\omega_0^2 + \omega_i^2), \qquad b_3 = \sum_{j=0}^{N} \ln P_i.$$

Equations (3.9), (3.10) and (3.11) can be written in abbreviated form as a system of linear equations, i.e.

$$a_{11}g + a_{12}q + a_{13}C = b_1$$

$$a_{21}g + a_{22}q + a_{23}C = b_2$$

$$a_{31}g + a_{32}q + a_{33}C = b_3.$$

The matrix form of this system is

$$A.Q = B$$

where,

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

$$Q = \begin{pmatrix} g \\ q \\ C \end{pmatrix}$$

and

$$B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

The vector of estimated parameters is then

$$Q = A^{-1}.B \tag{3.12}$$

given that A is a non-singular matrix.

3.8 Experimental Results

Herein, we introduce some numerical results of estimating the parameters g and q of the GRSF model. These results depend on using the formulas given in equations (3.8) and (3.12) and the MATLAB code (see Appendix A) designed and executed for this exercise. We consider the two cases when the the scaling parameter c is not applied, i.e the results depend on the case when $c = 1$, and when the scaling applied.

3.8.1 Estimating the Numerator parameter g

Here, we present different tables that give us different exact values of the parameter g, ranging from 1.9 to 2.9 in increments of 0.1 with estimated values and the percentages of estimation errors. In each table the initial conditions are fixed with values $N = 1024$ and $seed = 4$. The denominator parameter q is changed per five tables with the values $q = 3, 4, 5$ whereas the characteristic frequency ω_0 is changed per table for values of $\omega_0 = 5, 10, 15, 20, 25$.

3.8.2 Estimating the Denominator parameter q

We present different tables of the estimation results for the parameter q where its exact values ranges from 3.1 to 4.1 with increments of 0.1 and with the same options that are used in the estimation of the parameter g.

Comments on Tables

Table 3.4 to Table 3.9 show results of estimation along with the error of esti-

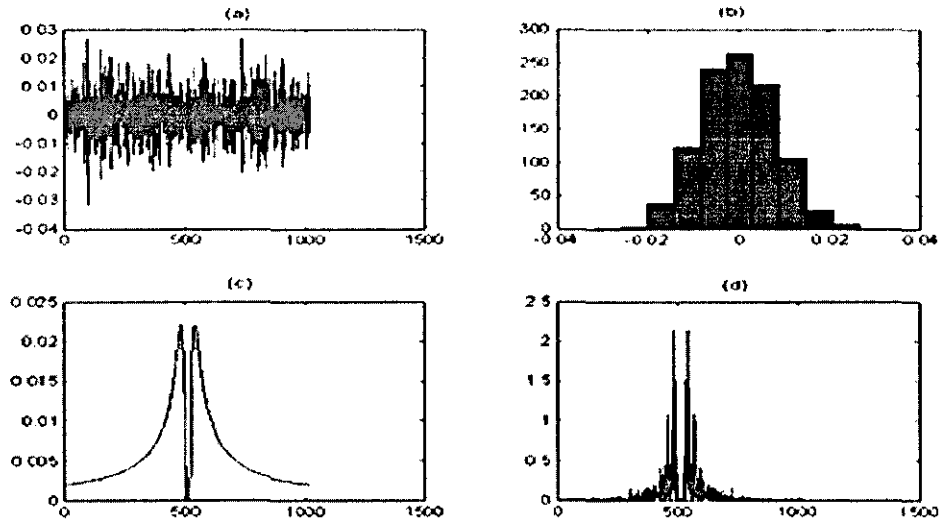

Figure 3.10: (a)&(b) Fractal signal with parameters g=3.5, q=4, $\omega_0 = 10$ and its PDF, (c)&(d), its theoretical and empirical power spectrum.

mation of the numerator parameter, g, when the denominator parameter q is fixed and the scaling parameter is not applied, whereas results of Table 3.10 to Table 3.12 when there is scaling. In these results different values of the characteristic frequency ω_0 are considered. The exact values of g are incremented by 0.1 with a different start value according to the fixed value of the parameter q. Table 3.13 to Table 3.15 show the estimated values along the estimation error of the parameter q with and without scaling.

As the exact value of the parameter (g or q) increases incrementally by 0.1, the corresponding estimated value increases by 0.2 (double the increment in the exact value). The error of estimation increases when the difference between the exact values of g and q increases.

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
1.9	0.21	89%	1.83	3%	2.35	19%
2.0	0.417	80%	2.03	1%	2.55	23%
2.1	0.617	71%	2.23	6%	2.55	27%
2.2	0.81	63%	2.43	10%	2.95	30%
2.3	1.01	56%	2.63	14%	3.15	33%
2.4	1.21	50%	2.83	18%	3.35	36%
2.5	1.41	44%	3.03	21%	3.55	39%
2.6	1.61	38%	3.23	24%	3.75	42%
2.7	1.81	32%	3.43	27%	3.95	44%
2.8	2.01	28%	3.63	30%	4.15	46%
2.9	2.21	24%	3.83	32%	4.35	50%

Table 3.4: Estimated values of g for $q = 3$

Exact(g)	$\omega_0 = 20$		$\omega_0 = 25$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)
1.9	2.62	38%	2.79	47%
2.0	2.82	41%	2.99	49%
2.1	3.02	44%	3.19	52%
2.2	3.22	46%	3.39	54%
2.3	3.42	49%	3.59	56%
2.4	3.62	51%	3.79	58%
2.5	3.82	53%	3.99	60%
2.6	4.02	55%	4.19	61%
2.7	4.22	56%	4.39	63%
2.8	4.42	58%	4.59	64%
2.9	4.62	59%	4.79	65%

Table 3.5: Estimated values of g for $q = 3$

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
2.9	2.20	24%	3.83	32%	4.35	50%
3.0	2.41	20%	4.03	34%	4.55	52%
3.1	2.61	16%	4.23	37%	4.75	53%
3.2	2.81	12%	4.43	38%	4.95	55%
3.3	3	9%	4.63	40%	5.15	56%
3.4	3.21	6%	4.83	42%	5.35	57%
3.5	3.41	3%	5.03	44%	5.55	58%
3.6	3.61	0.3%	5.23	45%	5.75	60%
3.7	3.81	3%	5.43	47%	5.95	61%
3.8	4.01	6%	5.63	48%	6.15	62%
3.9	4.21	8%	5.83	50%	6.35	63%

Table 3.6: Estimated values of g for $q = 4$

Exact(g)	$\omega_0 = 20$		$\omega_0 = 25$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)
2.9	4.62	59%	4.79	65%
3.0	4.82	61%	4.99	66%
3.1	5.02	62%	5.19	67%
3.2	5.22	63%	5.39	68%
3.3	5.42	64%	5.59	69%
3.4	5.62	65%	5.79	70%
3.5	5.82	66%	5.99	71%
3.6	6.02	67%	6.19	72%
3.7	6.22	68%	6.39	73%
3.8	6.42	69%	6.59	74%
3.9	6.62	70%	6.79	74%

Table 3.7: Estimated values of g for $q = 4$

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
3.9	4.21	24%	5.83	50%	6.35	63%
4.0	4.41	20%	6.03	51%	6.55	64%
4.1	4.61	16%	6.23	52%	6.75	65%
4.2	4.81	12%	6.43	53%	6.95	65%
4.3	5.01	9%	6.63	54%	7.15	66%
4.4	5.21	6%	6.83	55%	7.35	67%
4.5	5.41	3%	7.03	56%	7.55	68%
4.6	5.61	0.3%	7.23	57%	7.75	68%
4.7	5.81	3%	7.43	58%	7.95	69%
4.8	6.01	6%	7.63	59%	8.15	70%
4.9	6.21	27%	7.83	60%	8.35	70%

Table 3.8: Estimated values of g for $q = 5$

Exact(g)	$\omega_0 = 20$		$\omega_0 = 25$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)
3.9	6.62	70%	6.79	74%
4.0	6.82	70%	6.99	75%
4.1	7.02	71%	7.19	75%
4.2	7.22	72%	7.39	76%
4.3	7.42	72%	7.59	76%
4.4	7.62	73%	7.79	77%
4.5	7.82	74%	7.99	78%
4.6	8.02	74%	8.19	78%
4.7	8.22	75%	8.39	78%
4.8	8.42	75%	8.59	79%
4.9	8.62	76%	8.79	79%

Table 3.9: Estimated values of g for $q = 5$

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
3.9	7.36	88.7%	7.57	94.1%	7.63	95.6%
4.0	7.56	89 %	7.73	94.3%	7.83	95.7%
4.1	7.76	89.3%	7.93	94.4%	8.03	95.8%
4.2	7.96	89.5%	8.17	94.5%	8.23	95.9%
4.3	8.16	89.8%	8.37	94.7%	8.43	96 %
4.4	8.36	90 %	8.57	94.8%	8.63	96.1%
4.5	8.56	90.2%	8.77	94.9%	8.83	96.2%
4.6	8.76	90.4%	8.97	95 %	9.03	96.3%
4.7	8.96	90.6%	9.17	95.1%	9.23	96.4%
4.8	9.16	90.8%	9.37	95.2%	9.43	96.5%
4.9	9.36	91%	9.57	95.3%	9.63	96.5%

Table 3.10: Estimated values of g for $q = 5$, $c = 1.5$

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
1.9	3.36	76.9%	3.57	88 %	3.63	91.1%
2	3.56	78.1%	3.73	88.6 %	3.83	91.5%
2.1	3.76	79.1%	3.93	89.1%	4.03	91.9%
2.2	3.96	80.1%	4.17	89.6%	4.23	92.3%
2.3	4.16	80.9%	4.37	90.1%	4.43	92.6%
2.4	4.36	81.7%	4.57	90.5%	4.63	92.9%
2.5	4.56	82.4%	4.77	90.9%	4.83	93.2%
2.6	4.76	83.1%	4.97	91.2%	5.03	93.4%
2.7	4.96	83.7%	5.17	91.5%	5.23	93.7%
2.8	5.16	84.3%	5.37	91.8%	5.43	93.9%
2.9	5.36	84.9%	5.57	92.1%	5.63	94.1%

Table 3.11: Estimated values of g when $q = 3$, $c = 2$

Exact(g)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)	**Est.(g)**	Error(g)
2.9	5.36	84.9%	5.57	92.1%	5.63	94.1%
3	5.56	85.4%	5.77	92.4%	5.83	94.3%
3.1	5.76	85.8%	5.97	92.6%	6.03	94.5%
3.2	5.96	86.3%	6.17	92.8%	6.23	94.7%
3.3	6.16	86.7%	6.37	93.1%	6.43	94.8%
3.4	6.36	87.1%	6.57	93.3%	6.63	95 %
3.5	6.56	87.4%	6.77	93.5%	6.83	95.1%
3.6	6.76	87.8%	6.97	93.6%	7.03	95.3%
3.7	6.96	88.1%	6.17	93.8%	7.23	95.4%
3.8	7.16	88.4%	6.37	94%	7.43	95.5 %
3.9	7.36	88.7%	6.57	94.1%	7.63	95.6%

Table 3.12: Estimated values of g when $q = 4$, $c = 2$

Exact(q)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(q)	Error(q)	**Est.(q)**	Error(q)	**Est.(q)**	Error(q)
3.1	2.02	35%	3.65	18%	4.16	34%
3.2	2.22	31%	3.85	20%	4.36	36%
3.3	2.42	27%	4.05	23%	4.56	38%
3.4	2.62	23%	4.25	25%	4.76	40%
3.5	2.82	19%	4.45	27%	4.96	42%
3.6	3.02	16%	4.65	29%	5.16	43%
3.7	3.22	13%	4.85	31%	5.36	45%
3.8	3.42	10%	5.05	33%	5.56	46%
3.9	3.62	7 %	5.25	35%	5.76	48%
4.0	3.82	4 %	5.45	36%	5.96	49%
4.1	4.02	2%	5.65	38%	6.16	50%

Table 3.13: Estimated values of q for $g = 3$

Exact(q)	$\omega_0 = 20$		$\omega_0 = 25$	
	Est.(g)	Error(g)	**Est.(g)**	Error(g)
3.1	4.43	43%	4.61	49%
3.2	4.63	45%	4.81	50%
3.3	4.83	46%	5.01	52%
3.4	5.03	48%	5.21	53%
3.5	5.23	50%	5.41	55%
3.6	5.43	51%	5.61	56%
3.7	5.63	52%	5.81	57%
3.8	5.83	54%	6.01	58%
3.9	6.03	55%	6.21	59%
4	6.23	56%	6.41	60%
4.1	6.43	57%	6.61	61%

Table 3.14: Estimated values of g for $g = 3$

Exact(q)	$\omega_0 = 5$		$\omega_0 = 10$		$\omega_0 = 15$	
	Est.(q)	Error(q)	**Est.(q)**	Error(q)	**Est.(q)**	Error(q)
3.1	5.72	84.6%	5.92	91.2%	5.98	92.9%
3.2	5.92	85 %	6.12	91.5%	6.18	93.1%
3.3	6.12	85.5%	6.32	91.7%	6.38	93.3%
3.4	6.32	85.9%	6.52	92 %	6.58	93.5%
3.5	6.52	86.3%	6.72	92.2%	6.78	93.7%
3.6	6.72	86.7%	6.92	92.4%	6.98	93.9%
3.7	6.92	87.1%	7.12	92.6%	7.18	94.1%
3.8	7.12	87.4%	7.32	92.8%	7.38	94.2%
3.9	7.32	87.7%	7.52	93 %	7.58	94.4%
4	7.52	88 %	7.72	93.2%	7.78	94.5%
4.1	7.72	88.3%	7.92	93.3%	7.98	94.6%

Table 3.15: Estimated values of q when $g = 3$, $c = 2$

3.9 Multi-fractal Modulation and Demodulation

The purpose of multi-fractal modulation is to try and make the information content of the transmission phase 'look like' transmission noise so that any unauthorized recipient is incapable of distinguishing between the transmission of sensitive information and background noise. In this section, we present the forward algorithm to accomplish this purpose and later, we present the backward algorithm in which we demodulate the data to reconstruct the bit stream.

3.9.1 Multi-Fractal Modulation

Here, the method of modulation involves generating fractal signals in which four values of either the numerator parameter g or the denominator parameter q are used to differentiate between the bit-pairs 00, 01, 10 and 11 in a bit stream. Note that the optimal values of g and q are as given in the previous tables, where $g < q$. In what follows, we outline the technique when the modulation is based on values of the numerator parameter g for a fixed q. The basic approach is as follows:

(i) For a given bit stream allocate g_1, g_2, g_3 and g_4 for 00, 01, 10, and 11, respectively.

(ii) Compute a fractal signal of size N for each pair of bits in the stream.

(iii) Concatenate the results to produce a contiguous stream of fractal noise.

3.9.2 Multi-Fractal Demodulation

Here, we recover the encoded binary bits using fractal demodulation which is achieved by computing the estimated value of each of the parameters g_1, g_2, g_3 and g_4 via the power spectrum method using a moving window of size 1024. To accomplish this we define three Cut-Off Points (COPs), say, CP1, CP2 and CP3. These points divide the estimation region of the parameter into four sub-regions, say, R1, R2, R3 and R4 such that each estimated value of the used parameter in the encoding process belongs only to one of these regions (see, Figure 3.11). The bit stream is then obtained from the following algorithm:

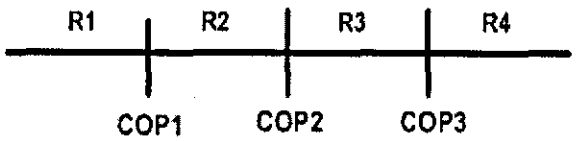

Figure 3.11: Estimation regions and COPs

if $\hat{g}_1 \in$ R1 then the bit-pair is 00

if $\hat{g}_2 \in$ R2 then the bit-pair is 01

if $\hat{g}_3 \in$ R3 then the bit-pair is 10

if $\hat{g}_4 \in$ R4 then the bit-pair is 11.

In practical, the COPs are pre-configured and are chosen depending on the results compounded in Table 3.4 to Table 3.15. In reality, when the contiguous stream of modulated fractal signals has been transmitted, it is important for the receiver to have the cut-off points in order be able to demodulate the signal and recover the encoded bit stream.

In this case, the cut-off points can be considered to be *private keys* which are known only by the sender (and receiver). There are, in principle, many ways to exchange the cut-off points between sender and receiver. One of these methods is to use the chirp coding technique in which we watermark the contiguous stream of fractal modulated signals by watermarking them with the cut-off points. In chapter 4 we present the background theory to this method and explain how the embedding of the cut-off points in the transmitted signal can additionally be used to authenticate the signal.

3.9.3 Illustrative Example

Herein, we present an illustrative example for encoding a stream of binary bits to produce a stream of modulated fractal signals, and then decode the contiguous stream of such signals to recover the encoded bit stream.

Modulating

Suppose we look forward to encode the stream of bits bn='00110110111100100110' via modulating the value of the parameter g by $g_1 = 3.5$, $g_2 = 3.6$, $g_3 = 3.7$, and $g_4 = 3.8$ with fixed value of the parameter $q = 4$ and the initial conditions $N = 1024$, $w_0 = 5$, $seed = 4$ without the scaling factor, i.e. $c = 1$. Figure 3.15 and Figure 3.16 show the plots of the contiguous streams of the multi-fractal modulated signals for the binary code bn when the scaling parameter is not applied and when it is applied with different values.

Figure 3.13 and Figure 3.14 show the theoretical PSDF and the empirical power spectrum of the fractal signal with four different values of the parameter g, a fixed value of the parameter q, and c=1.

Demodulating

To recover the encoded stream of bits we apply a moving window of size $N = 1024$ on the contiguous stream of modulated fractal signals that is shown in Figure 3.16. From each window segment we use the Power Spectrum Method (PSM) to estimate the parameter g and q, but are interested in the resulting values of g which are given as follow: 3.41, 4.01, 3.61, 3.81, 4.01, 4.01, 3.41, 3.81, 3.61, 3.81

To recover the encoded binary original bit stream, assume that the three pre-configured cut-off points are: CP1= 3.5, CP2= 3.7, CP3= 3.9; applying the decoding algorithm discussed in the previous subsection we get the original binary bit with no error.

Similarly, we can apply multi-fractal modulation by modulating value of denominator parameter q by four values, say q_1, q_3, q_3 and q_4, where the value of the parameter g is fixed and predefined together with the initial conditions.

Also we can encode the bit-pair by modulating the value of parameter g and the value of parameter q at time, with two values for each one, namely, g_1, g_2 and q_1, q_2, respectively. This means that we may encode 00 and 01 by modulating the value of g to g_1 and g_2, respectively, and modulate the value of q by q_1, and q_2, to encode 10 and 11, respectively. Figure 3.12 shows the block diagram of the coding and decoding processes.

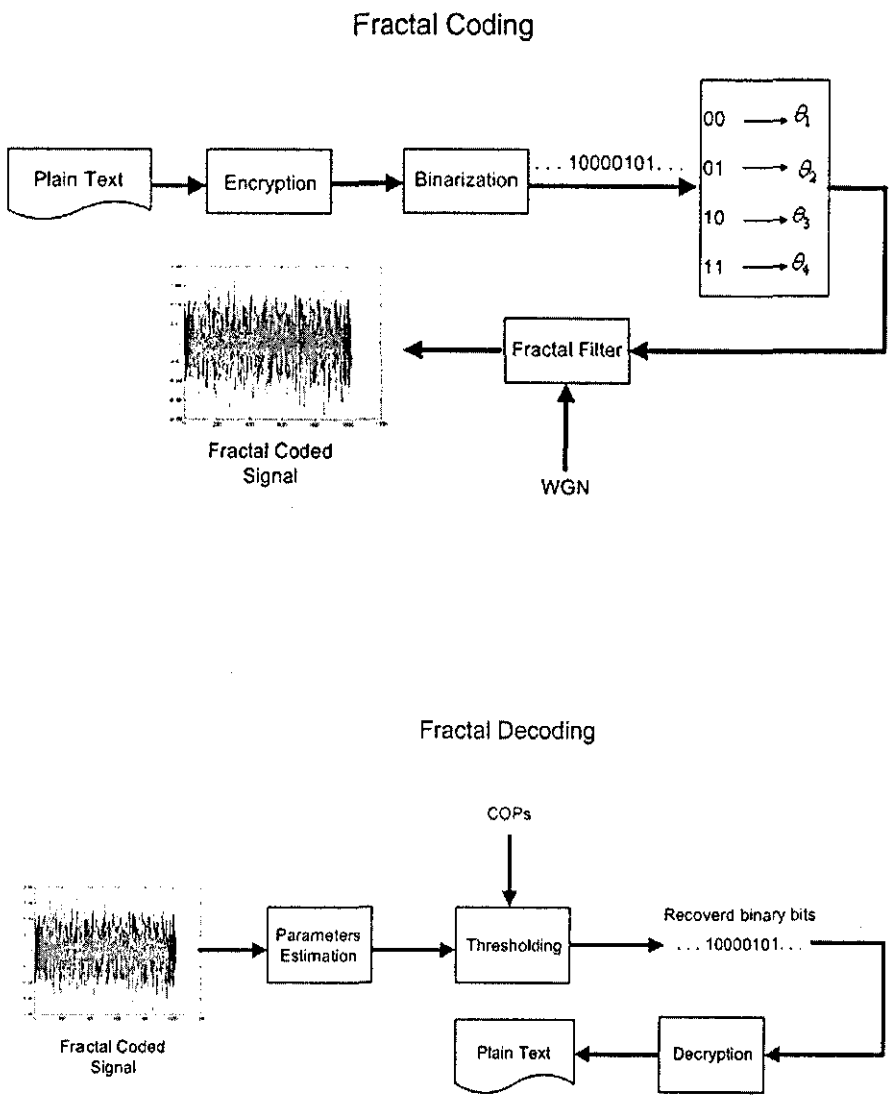

Figure 3.12: Fractal coding and decoding processes.

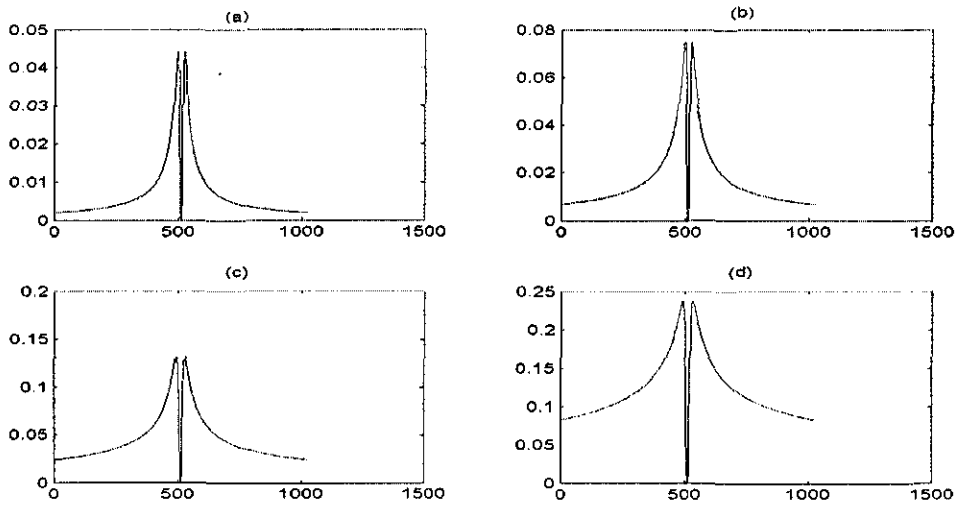

Figure 3.13: The PSDF with $q = 4$ and (a) $g_1 = 3.5$, (b)$g_2 = 3.6$, (c)$g_3 = 3.7$ (d) and $g_4 = 3.8$.

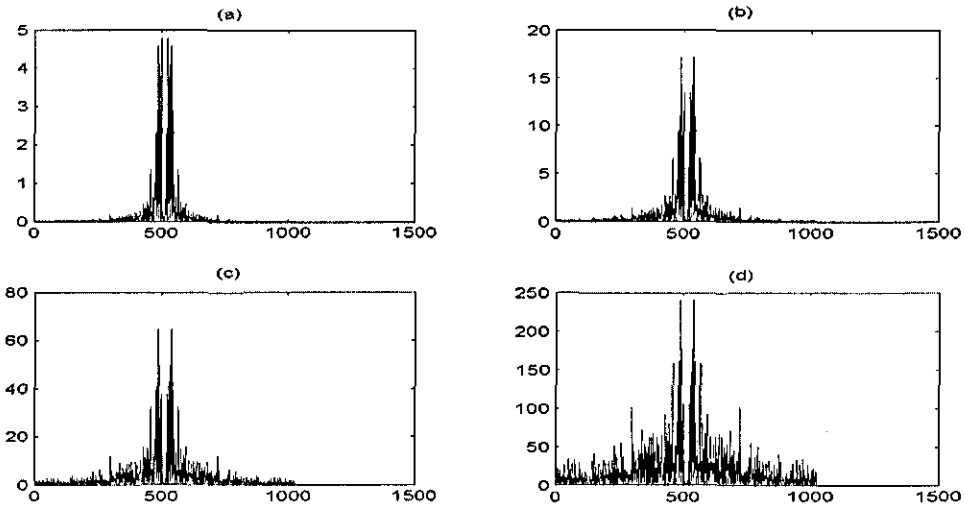

Figure 3.14: Power spectrum with $q = 4$ and (a) $g_1 = 3.5$, (b)$g_2 = 3.6$, (c)$g_3 = 3.7$ (d) and $g_4 = 3.8$.

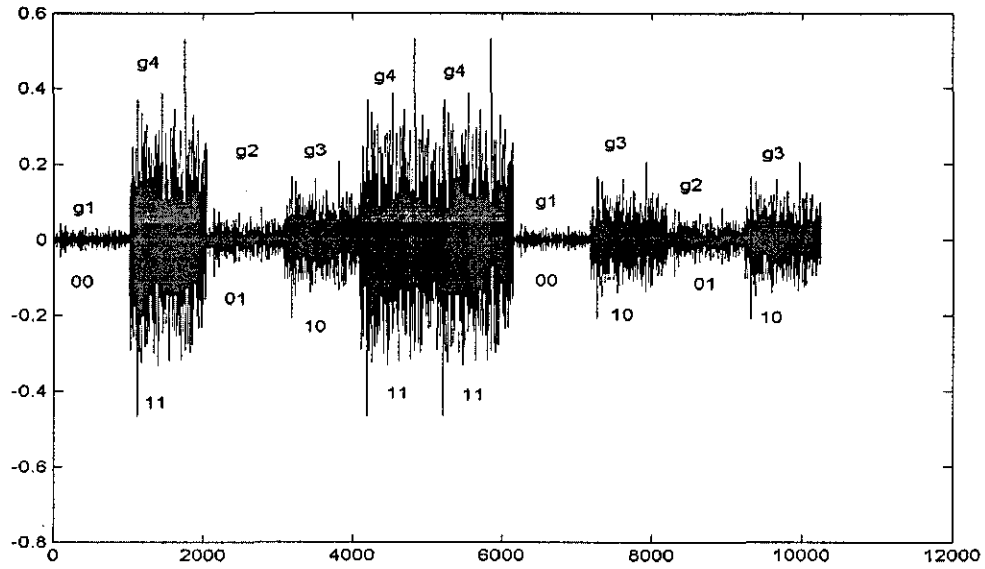

Figure 3.15: Multi-fractal modulated signals, without scaling factor (c=1).

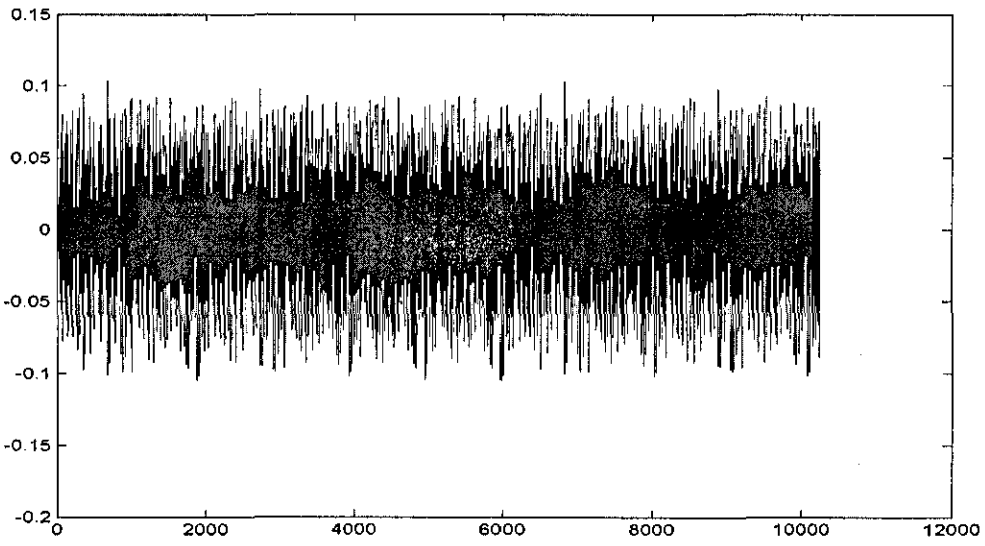

Figure 3.16: Multi-fractal modulated signals, with different scales.

3.10 References

[1] Tatom F.B., *The Application of fractional Calculus to the Simulation of Stochastic Processes*, Engineering Analysis Inc., Huntsville, Alabama, AIAA-89/0792, 1989.

[2] Blackledge J.M., Tuner M.J. and Andrews P.R., *Fractal Geometry in Digital Imaging*, Academic Press, ISBN 0-12-703970-8, London., 1998.

[3] Blackledge J.M., *Digital Signal Processing:Mathematical and Computation Methods, Software Development and Applications*, Horwood Publishing Limited, London, 2nd Edition, 2006.

[4] Samko S., Kilbas A., Marichev O., *Fractional Integrals and Derivatives*, Gordon and Breach Science Publishers, 1993.

[5] Blackledge J.M., *Applications of the Fractal Geometry to Pattern Recognition in Digital Images*, Science and Engineering Research Center, De Montfort University, UK, 1993.

[6] Blackledge J.M., Foxon B. and Mikhailov S., "Fractal modulation for digital communications systems," *Proc. of the IEEE Military Communications Conference MILCOM'98*, Boston, USA, Oct. 1998.

[7] Mandelbrot B.B., *The Fractal Geometry of Nature*, Freeman, 1983.

[8] Oldham K.B. and Spanier J., *The Fractional Calculus*, Academic Press, 1974.

[9] Blackledge J.M. Foxon B. Mikhailov S., "Fractal coding techniques," *Proc. of the IEEE Military Communications Conf. MILCOM'96*, Nice, France, 1996.

[10] Oldham K.B. and Spanier J., *The Fractional Calculus*, Academic Press, 1974.

[11] Miller K.S. and Ross B., *An Introduction to the Fractional Calculus and Fractional Differential Equations*, John Wiley& Sons Ltd., 1993.

[12] Oberhettinger F. and Badii L., *Table of Laplace Transforms*, Springer, 1973.

[13] Blackledge J.M., Foxon B. and Mikhailov S., "Fractal Dimension Segmentation," *Proc. of the 1st IMA Conf. on Image Processing*, pp. 249-289, 1997.

[14] Bracewell, *The Fourier Transform and its Applications*, McGraw-Hill, New york, 2nd edition, 1996.

[15] Evans A.K., "Fourier dimension, fractal dimension and the fractional derivative," *Technical Report*, 23, SERCenter, De Montfort University, Uk, 1997.

[16] Blackledge J.M., *On the Synthesis and Processing of Fractal Signals and Images in Applications of Fractals and Chaos*, Springer-Verlag, New York, 1993.

[17] Mikhailov S., *Fractal Modulation and Encryption*, PhD Thesis, De Montfort University, UK, 1999.

[18] Evans A.K.,"The Fourier dimension and the fractal dimensions", *Chaos*, Solitons and Fractals, vol. 9, no. 12, pp. R848-R851, 1995.

Chapter 4

Digital Watermarking and Self-Authentication

4.1 Information Embedding and Digital Watermarking

In this Chapter, we introduce a new watermarking technology as a solution to verifying (or invalidating) the authenticity of a signal. Digital watermarking and information embedding, which are also referred as data hiding, refer to the process of embedding one signal called the 'embedded signal' or 'digital watermark' within another signal called the 'host signal'. Here, i.e. within the context of this thesis, we introduce the idea of watermarking in order to satisfy the two purposes:

(i) to exchange the thresholding or cut-off points which are vital for reconstructing the encoded binary bits (as discussed in Chapter 3).

(ii) to authenticate the stream of fractal modulated signals in order to confirm authenticity of the signal before applying the reconstruction processes.

In this application, the host signal is a fractal modulated signal and the digital watermark is a binary coded bit stream of the cut-off points.

The watermarking method is based on *linear frequency modulated* 'chirp coding'. The principle underlying this approach is based on the use of a matched filter to provide a reconstruction of a chirp code that is uniquely robust in the case of very low signal-to-noise ratios. This is the principal reason as to why the method can be used so effectively to embed data in a host signal.

In what follows, we present a brief revision of the theoretical and computational aspects of the matched filter and the properties of a chirp to provide the essential background to the method. Signal code generating schemes are then addressed and details of the coding and decoding techniques considered.

4.1.1 The Matched Filter

In this thesis, the method of watermarking is based on application of a specific function - the chirp - coupled with a well defined processes - the matched filter. The matched filter is a result of finding a solution to the following problem: Given that a signal can be modelled in terms of the linear time invariant process

$$s_i = \sum_j p_{i-j} f_j + n_i,$$

where p_i is the Impulse Response Function (IRF), f_i is the information carried by the signal and n_i is noise, find an estimate for the IRF given by

$$\hat{f}_i = \sum_j q_j s_{i-j}$$

where

$$SNR = \frac{|\sum_i Q_i P_i|^2}{\sum_i |N_i|^2 |Q_i|^2}$$

is a maximum. Note that the ratio defining SNR is a measure of the signal-to-noise ratio (SNR). In this sense, the matched filter maximizes the signal-to-noise ratio of the output. Assuming that the noise n_i has a 'white' or uniform power spectrum, the filter Q_i which maximizes the signal-to-noise defined by SNR is given by

$$Q_i = P_i^*$$

and the required solution is therefore

$$\hat{f}_i = \text{IDFT}(P_i^* S_i).$$

Using the correlation theorem, we then have

$$\hat{f}_i = \sum_j p_{j-i} s_j.$$

The matched filter is therefore based on correlating the signal s_i with the IRF p_i. This filter is frequently used in systems that employ linear frequency modulated (FM) pulses - 'chirped pulses' - which will be discussed later.

4.1.2 Derivation of the Matched Filter

With the problem specified as above, the matched filter is essentially a 'by-product' of the 'Cauchy-Schwartz Inequality', i.e.

$$\left|\sum_i Q_i P_i\right|^2 \leq \sum_i \mid Q_i \mid^2 \sum_i \mid P_i \mid^2$$

The principal 'trick' is to write

$$Q_i P_i = \mid N_i \mid Q_i \times \frac{P_i}{\mid N_i \mid}$$

so that the above inequality becomes

$$\left|\sum_i Q_i P_i\right|^2 = \left|\sum_i \mid N_i \mid Q_i \frac{P_i}{\mid N_i \mid}\right|^2 \leq \sum_i \mid N_i \mid^2 \mid Q_i \mid^2 \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

From this result, using the definition of r given above, we see that

$$r \leq \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

Now, if r is to be a maximum, then we want

$$SNR = \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}$$

or

$$\left|\sum_i \mid N_i \mid Q_i \frac{P_i}{\mid N_i \mid}\right|^2 = \sum_i \mid N_i \mid^2 \mid Q_i \mid^2 \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

But this is only true if

$$\mid N_i \mid Q_i = \frac{P_i^*}{\mid N_i \mid}$$

and hence, r is a maximum when

$$Q_i = \frac{P_i^*}{\mid N_i \mid^2}.$$

Note that if the noise n_i is white noise, then its power spectrum $\mid N_i \mid^2$ is uniformly distributed. In particular, under the condition

$$\mid N_i \mid^2 = 1 \quad \forall i = 0, 1, ..., N-1$$

then

$$Q_i = P_i^*.$$

4.1.3 Pseudo Code for the Matched Filter

Using pseudo code the matched filter process is;

```
for i=1, 2, ..., n; do:
   sr(i)=signal(i)
   si(i)=0.
   pr(i)=IRF(i)
   pi(i)=0.
enddo
       forward_fft(sr,si)
       forward_fft(pr,pi)
 for i=1, 2, ..., n; do:
    fr(i)=pr(i)*sr(i)+pi(i)*si(i)
    fi(i)=pr(i)*si(i)-pi(i)*sr(i)
 enddo
 inverse_fft(fr,fi)
 for i=1, 2, ..., n; do:
 hatf(i)=fr(i)
 enddo
```

4.1.4 Deconvolution of Frequency Modulated Signals

The matched filter is frequently used in systems that utilize linear frequency modulated (FM) pulses. IRF's of this type are known as chirped pulses. Examples of where this particular type of pulse is used include real and synthetic aperture radar, active sonar and some forms of seismic prospecting.

Interestingly, some mammals (dolphins, whales and bats for example) use frequency modulation for communication and detection. The reason for this is the unique properties that FM IRFs provide in terms of the quality of extracting information from signals with very low signal-to-noise ratios and the simplicity of the process that is required to do this (i.e. correlation).

The invention and use of FM IRFs for man-made communications and imaging systems dates back to the early 1960s (the application of FM to radar for example); mother nature appears to have 'discovered' the idea some time ago.

Linear FM Pulses

The linear FM pulse is given (in complex form) by

$$p(t) = \exp(-i\alpha t^2), \quad \mid t \mid \leq T/2$$

where α is a constant and T is the length of the pulse. The phase of this pulse is αt^2 and the instantaneous frequency is given by

$$\frac{d}{dt}(\alpha t^2) = 2\alpha t$$

which varies linearly with t. Hence, the frequency modulations are linear which is why the pulse is referred to as a linear FM pulse. In this case, the signal that is recorded is given by (neglecting additive noise)

$$s(t) = \exp(-i\alpha t^2) \otimes f(t).$$

With matched filtering, we have

$$\hat{f}(t) = \exp(i\alpha t^2) \odot \exp(-i\alpha t^2) \otimes f(t).$$

Evaluating the correlation integral,

$$\exp(i\alpha t^2) \odot \exp(-i\alpha t^2) = \int_{-T/2}^{T/2} \exp[i\alpha(t+\tau)^2] \exp(-i\alpha\tau^2) d\tau$$

$$= \exp(i\alpha t^2) \int_{-T/2}^{T/2} \exp(2i\alpha\tau t) d\tau$$

and computing the integral over τ, we have

$$\exp(i\alpha t^2) \odot \exp(-i\alpha t^2) = T \exp(i\alpha t^2) sinc(\alpha T t)$$

and hence

$$\hat{f}(t) = T \exp(i\alpha t^2) sinc(\alpha T t) \otimes f(t).$$

In some systems, the length of the linear FM pulse is relatively long. In such cases,

$$\cos(\alpha t^2) sinc(\alpha T t) \simeq sinc(\alpha T t)$$

and

$$\sin(\alpha t^2) sinc(\alpha T t) \simeq 0$$

and so

$$\hat{f}(t) \simeq T sinc(\alpha T t) \otimes f(t).$$

Now, in Fourier space, this last equation can be written as

$$\hat{F}(\omega) = \left\{ \begin{array}{ll} \frac{\pi}{\alpha} F(\omega), & \mid \omega \mid \leq \alpha T \\ 0, & \text{otherwise} \end{array} \right\}$$

The estimate $\hat{f}$ is therefore a band limited estimate of f whose bandwidth is determined by the product of the chirping parameter α with the length of the pulse T. An example of the matched filter is given in Figure 4.1 obtained using the MATLAB code given below. Here, two spikes have been convolved with a linear FM chirp whose width or pulse length T is significantly greater than that of the input signal.

The output signal has been generated using an SNR of 1 and it is remarkable that such an excellent restoration of the input is recovered using a relatively simple operation for processing data that has been so badly distorted by additive noise. The remarkable ability for the matched filter to accurately recover information from linear FM type signals with very low SNRs leads naturally to consider its use for covert information embedding.

This is the subject of the section that follows which investigates the use of chirp coding for covertly watermarking digital signals for the purpose of signal authentication.

```
function MATCH(T,snr)
%Input:
%       T - width of chirp IRF
%       snr - signal-to-noise ratio of signal
%
n=512;     %Set size of array (arbitrary)
nn=1+n/2; %Set mid point of array
%Compute input function (two spikes of width m centered
%at the mid point of the array.
m=10; %Set width of the spikes (arbitrary)
```

```
for i=1:n
   f(i)=0.0; %Initialize input
   p(i)=0.0; %Initialize IRF
end
   f(nn-m)=1.0;
   f(nn+m)=1.0;
%Plot result
figure(1);
subplot(2,2,1), plot(f);
%Compute the (real) IRF, i.e. the linear FM chirp using a
%sine function. (N.B. Could also use a cosine function.)
m=T/2;
k=1;
for i=1:m
   p(nn-m+i)=sin(2*pi*(k-1)*(k-1)/n);
   k=k+1;
end
%Plot result
subplot(2,2,2), plot(p);
%Convolve f with p using the convolution theorem and normalize to unity.
f=fft(f); p=fft(p);
          f=p.*f;
          f=ifft(f); f=fftshift(f); f=real(f);
f=f./max(f); %N.B. No check on case when f=0.
%Compute random Gaussian noise field and normalize to unity.
noise=randn(1,n);
```

```
noise=noise./max(noise);
%Compute signal with signal-to-noise ratio defined by snr.
s=f+noise./snr;
%Plot result
subplot(2,2,3), plot(s);
%Restore signal using Matched filter.
%Transform to Fourier space.
s=fft(s);
%Compute Matched filter.
rest=conj(p).*s;
rest=ifft(rest); rest=fftshift(rest); rest=real(rest);
%Plot result
subplot(2,2,4), plot(rest);
```

4.1.5 Watermarking using Chirp Coding

In this section, we discusses a new approach to 'watermarking' digital signals using linear frequency modulated 'chirp coding'. The principle underlying to this approach is based on the use of a matched filter to provide a reconstruction of a chirped code that is uniquely robust, i.e. in the case of very low signal-to-noise ratios.

Chirp coding for authenticating data is generic in the sense that it can be used for a range of data types and applications(the authentication of speech and audio signals for example). Signal code generating schemes are then addressed and details of the coding and decoding techniques considered.

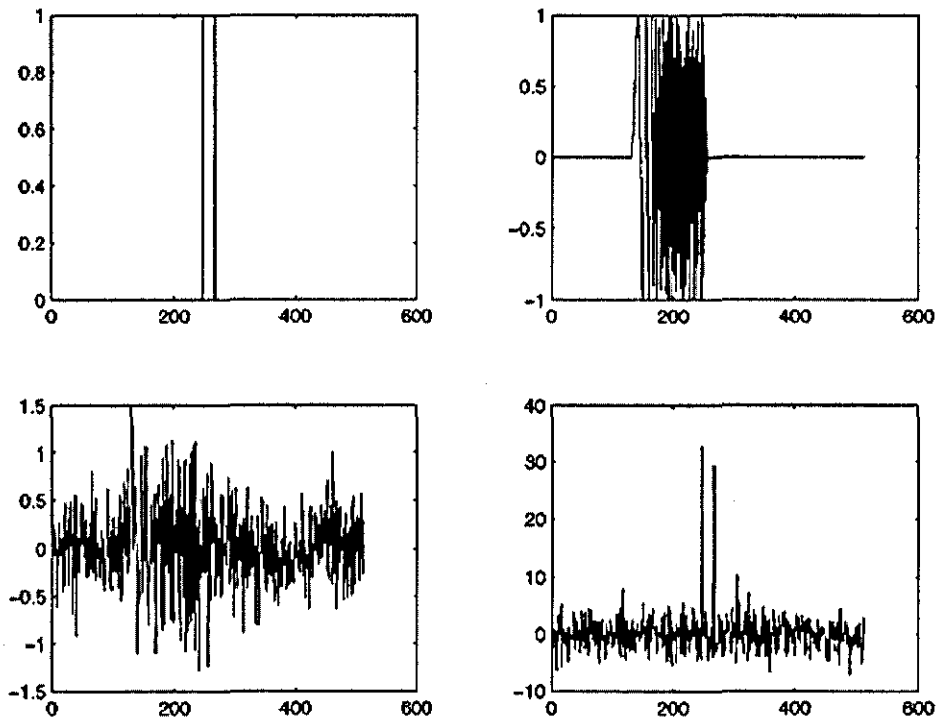

Figure 4.1: Example of a matched filter in action (bottom right) by recovering information from a noisy signal (bottom left) generated by the convolution of an input consisting of two spikes (top left) with a linear FM chirp IRF (top right). The simulation and restoration of the signal given in this example is accomplished using the MATLAB function MATCH(256,1).

4.1.6 Basic concepts

Methods of watermarking digital data have applications in a wide range of areas. Digital watermarking of images has been researched for many years in order to achieve methods which provide both anti-counterfeiting and authentication facilities. One of the principal equations that underpins this technology is based on the 'fundamental model' for a signal which is given by

$$s = \hat{P}f + n$$

where f is the information content for the signal (the watermark), $\hat{P}$ is some linear operator, n is the noise and s is the output signal. This equation is usually taken to describe a stationary process in which the noise n is characterized by stationary statistics (i.e. the probability density or distribution function of n is invariant of time).

In the field of cryptology, the operation $\hat{P}f$ is referred to as the processes of 'diffusion' and the process of adding noise (i.e. $\hat{P}f + n$) is referred to as the process of 'confusion'. In cryptography and steganography (the process of hiding secret information in images) the principal 'art' is to develop methods in which the processes of diffusion and confusion are maximized, an important criterion being that the output s should be dominated by the noise n which in turn should be characterized by a maximum[1] (i.e. a uniform statistical distribution).

Digital watermarking and steganography can be considered to form part of the same field of study, namely, cryptology. Being able to recover f from s provides a way of authenticating the signal. If, in addition, it is possible

[1] A measure of the lack of information on the exact state of a system

to determine that a copy of s has been made leading to some form of data degradation and/or corruption that can be conveyed through an appropriate analysis of f, then a scheme can be developed that provides a check on: (i) the authenticity of the data s; (ii) its fidelity.

Formally, the recovery of f from s is based on the inverse process

$$f = \hat{P}^{-1}(s - n)$$

where $\hat{P}^{-1}$ is the inverse operator. Clearly, this requires the field n to be known *a priori*. If this field has been generated by a pseudo random number generator for example, then the seed used to generate this field must be known *a priori* in order to recover the data f. In this case, the seed represents the private key required to recover f. However, in principle, n can be any field that is considered appropriate for confusing the information $\hat{P}f$ including a pre-selected signal. Further, if the process of confusion is undertaken in which the signal-to-noise ratio is set to be very low (i.e. $\|n\| >> \|\hat{P}f\|$), then the watermark f can be hidden covertly in the data n provided the inverse process $\hat{P}^{-1}$ is well defined and computationally stable. In this case, it is clear that the host signal n must be known in order to recover the watermark f leading to a private watermarking scheme in which the field n represents a key. This field can of course be (lossless) compressed and encrypted as required. In addition, the operator $\hat{P}$ (and its inverse $\hat{P}^{-1}$) can be key dependent. The value of this operator key dependency relies on the nature and properties of the operator that is used and whether it is compounded in an algorithm that is required to be in the public domain for example.

Another approach is to consider the case in which the field n is unknown and to consider the problem of extracting the watermark f in the absence of this

field. In this case, the reconstruction is based on the result

$$f = \hat{P}^{-1}s + m$$

where

$$m = -\hat{P}^{-1}n.$$

Now, if a process $\hat{P}$ is available in which $\|\hat{P}^{-1}s\| >> \|m\|$, then an approximate (noisy) reconstruction of f can be obtained in which the noise m is determined by the original signal-to-noise ratio of the data s and hence, the level of covertness of the diffused watermark $\hat{P}f$. In this case, it may be possible to post-process the reconstruction (de-noising for example) and recover a relatively high-fidelity version of the watermark, i.e.

$$f \sim \hat{P}^{-1}s.$$

This approach (if available) does not rely on a private key (assuming $\hat{P}$ is not key dependent). The ability to recover the watermark only requires knowledge of the operator $\hat{P}$ (and its inverse) and post-processing options as required. The problem here is to find an operator that is able to recover the watermark effectively in the presence of the field n. Ideally, we require an operator $\hat{P}$ with properties such that $\hat{P}^{-1}n \to 0$.

In this application, the operator is based on a chirp function, specifically, a linear Frequency Modulated (FM) chirp of the (complex) type $\exp(-i\alpha t^2)$ where α is the chirp parameter and t is the independent variable. This function is then convolved with f. The inverse process is undertaken by correlating with the (complex) conjugate of the chirp $\exp(i\alpha t^2)$. This provides a reconstruction for f in the presence of the field n that is accurate and robust with very

low signal-to-noise ratios. Further, we consider a watermark based on a coding scheme in which the field n is the input. The watermark f is therefore n-dependent. This allows an authentication scheme to be developed in which the watermark is generated from the field in which it is to be hidden. Authentication of the watermarked data is then based on comparing the code generated from $s = \hat{P}f + n$ and that reconstructed by processing s when $\|\hat{P}f\| >> \|n\|$. This is an example of a self-generated coding scheme which avoids the use, distribution and application of reference codes. Here, the coding scheme is based on the application of Daubechies wavelets. There are numerous applications of this technique in areas such as telecommunications and speech recognition where authentication is mandatory. For example, the method can readily be applied to audio data with no detectable differences in the audio quality of the data. The watermark code is able to be recovered accurately and changes relatively significantly if the data is distorted through cropping, filtering, noise or a compression system for example. Thus, it provides a way making a signal tamper proof.

4.1.7 Matched Filter Reconstruction

Given that

$$s(t) = \exp(-i\alpha t^2) \otimes f(t) + n(t),$$

after matched filtering, we obtain the estimate

$$\hat{f}(t) \simeq T \sin(\alpha T t) \otimes f(t) + \exp(i\alpha t^2) \odot n(t).$$

The correlation function produced by the correlation of $\exp(i\alpha t)$ with $n(t)$ will in general be relatively low in amplitude since $n(t)$ will not normally have

features that match those of a chirp. Thus, it is reasonable to assume that

$$\|T\sin(\alpha Tt) \otimes f(t)\| >> \|\exp(i\alpha t^2) \odot n(t)\|$$

and that in practice, $\hat{f}$ is a band-limited reconstruction of f with high SNR. Thus, the process of using chirp signals with matched filtering for the purpose of reconstructing the input in the presence of additive noise provides a relatively simple and computationally reliable method of 'diffusing' and reconstructing information encoded in the input function f. This is the underlying principle behind the method of watermarking described here.

4.1.8 Chirp Coding, Decoding and Watermarking

We now return to the issue of watermarking using chirp functions. The basic model for the watermarked signal (which is real) is

$$s(t) = \text{chirp}(t) \otimes f(t) + n(t)$$

where

$$\text{chirp}(t) = \sin(\alpha t^2).$$

We consider the field $n(t)$ to be some pre-defined signal to which a watermark is to be 'added' to generate $s(t)$. In principle, any watermark described by the function $f(t)$ can be used.

On the other hand, for the purpose of authentication we require two criteria:

(i) $f(t)$ should represent a code which can be reconstructed accurately and robustly;

(ii) the watermark code should be sensitive (and ideally ultra-sensitive) to any degradation in the field $n(t)$ due to lossy compression, cropping or highpass and lowpass filtering for example.

To satisfy condition (i), it is reasonable to consider $f(t)$ to represent a bit stream, i.e. to consider the discretized version of $f(t)$ - the vector f_i - to be composed of a set of elements with values 0 or 1 and only 0 or 1.

This binary code can of course be based on a key or set of keys which, when reconstructed, is compared to the key(s) for the purpose of authenticating the data.

However, this requires the distribution of such keys (public and/or private). Instead, we consider the case where a binary sequence is generated from the field $n(t)$.

There are a number of approaches that can be considered based on the spectral characteristics of $n(t)$ for example. These are discussed in later on, in which binary sequences are produced from the application of wavelet decomposition.

Chirp Coding

Given that a binary sequence has been generated from $n(t)$, we now consider the method of chirp coding. The purpose of chirp coding is to 'diffuse' each bit over a range of compact support T. However, it is necessary to differentiate between 0 and 1 in the sequences. The simplest way to achieve this is to change the polarity of the chirp. Thus, for 1 we apply the chirp $\sin(\alpha t^2)$, $t \in T$ and for 0 we apply the chirp $-\sin(\alpha t^2)$, $t \in T$ where T is the chirp length. The chirps are then concatenated to produce a contiguous stream of data, i.e. a signal composed of $\pm$chirps.

Thus, the binary sequence 010 for example is transformed to the signal

$$s(t) = \begin{cases} -\text{chirp}(t), & t \in [0, T); \\ +\text{chirp}(t), & t \in [T, 2T); \\ -\text{chirp}(t), & t \in [2T, 3T). \end{cases}$$

The period over which the chirp is applied depends on the length of the signal to which the watermark is to be applied and the length of the binary sequence. In the example given above, the length of the signal is taken to be $3T$.

In practice, care must be taken over the chirping parameter α that is applied for a period T in order to avoid aliasing and in some cases it is of value to apply a logarithmic sweep instead of a linear sweep. The instantaneous frequency of a logarithmic chirp is given by

$$\psi(t) = \psi_0 + 10^{at}$$

where

$$a = \frac{1}{T} \log_{10}(\psi_1 - \psi_0)$$

ψ_0 is the initial frequency and ψ_1 is the final frequency at time T. In this case, the final frequency should be greater than the initial frequency.

Decoding

Decoding or reconstruction of the binary sequence requires the application of a correlator using the function $\text{chirp}(t), \quad t \in [0, T)$. This produces a correlation function that is either -1 or +1 depending upon whether $-\text{chirp}(t)$ or $+\text{chirp}(t)$ has been applied respectively.

For example, after correlating the chirp coded sequence 010 given above, the correlation function $c(t)$becomes

$$c(t) = \begin{cases} -1, & t \in [0, T) \\ +1, & t \in [T, 2T) \\ -1, & t \in [2T, 3T) \end{cases}$$

from which the original sequence 010 is easily inferred, the change in sign of the correlation function identifying a bit change(from 0 to 1 or from 1 to 0). Note that in practice the correlation function may not be exactly 1 or -1 when reconstruction is undertaken and the binary sequence is effectively recovered by searching the correlation function for changes in sign. The chirp used to recover the watermark must of course have the same parameters (inclusive of its length) as those used to generate the chirp coded sequence. These parameters can be used to define part of a private key.

Watermarking

The watermarking process is based on adding the chirp coded data to the signal $n(t)$. Let the chirp coded signal be given by the function $h(t)$, then the watermarking process is described by the equation

$$s(t) = a\left[\frac{bh(t)}{\|h(t)\|_\infty} + \frac{n(t)}{\|n(t)\|_\infty}\right]$$

and the coefficients $a > 0$ and $0 < b < 1$ determine the amplitude and the SNR of s where

$$a = \|n(t)\|_\infty.$$

The coefficient a is required to provide a watermarked signal whose amplitude is compatible with the original signal n. The value of b is adjusted to provide

an output that is acceptable in the application to be considered and to provide a robust reconstruction of the binary sequence by correlating $s(t)$ with $\text{chirp}(t), t \in [0, T)$. To improve the robustness of the reconstruction, the value of b can be increased, but this has to be offset with regard to the perceptual quality of the output, i.e. the perturbation of n by h should be as small as possible.

4.1.9 Code Generation

In the previous section, the method of chirp coding a binary sequence and watermarking the signal $n(t)$ has been discussed where it is assumed that the sequence is generated from this same signal. In this section, the details of this method are presented. The problem is to convert the salient characteristics of the signal $n(t)$ into a sequence of bits that is relatively short and conveys information on the signal that is unique to its overall properties.

In principle, there are a number of ways of undertaking this. For example, in practice the digital signal n_i, which will normally be composed of an array of real numbers, could be expressed in binary form and each element concatenated to form a contiguous bit stream. However, the length of the code (i.e. the total number of bits in the stream) will tend to be large leading to high computational costs in terms of the application of chirp coding/decoding. What is required, is a process that yields a relatively short binary sequence (when compared with the original signal) that reflects the important properties of the signal in its entirety. Two approaches are considered here:

(i) power spectral density decomposition; (ii) wavelet decomposition.

Power Spectral Density Decomposition

Let $N(\omega)$ be the Fourier transform $n(t)$ and define the Power Spectrum $P(\omega)$ as

$$P(\omega) = \mid N(\omega) \mid^2 .$$

An important property of the binary sequence is that it should describe the spectral characteristics of the signal in its entirety. Thus, if for example, the binary sequence is based on just the low frequency components of the signal, then any distortion of the high frequencies components will not affect the watermark and the signal will be authenticated.

Hence, we consider the case where the power spectrum is decomposed into N components, i.e.

$$P_1(\omega) = P(\omega), \quad \omega \in [0, \Omega_1);$$

$$P_2(\omega) = P(\omega), \quad \omega \in [\Omega_1, \Omega_2);$$

$$\vdots$$

$$P_N(\omega) = P(\omega), \quad \omega \in [\Omega_{N-1}, \Omega_N).$$

Note, that it is assumed that the signal $n(t)$ is band-limited with a bandwidth of Ω_N.

The set of the functions $P_1, P_2, ..., P_N$ now reflect the complete spectral characteristics of the signal $n(t)$. Since each of these functions represents a unique part of the spectrum, we can consider a single measure as an identifier or tag.

A natural measure to consider is the energy which is given by the integral of the functions over their frequency range.

In particular, we consider the energy values in terms of their contribution to the spectrum as a percentage, i.e.

$$E_1 = \frac{100}{E} \int_0^{\Omega_1} P_1(\omega) d\omega,$$

$$E_2 = \frac{100}{E} \int_{\Omega_1}^{\Omega_2} P_2(\omega) d\omega,$$

$$\vdots$$

$$E_N = \frac{100}{E} \int_{\Omega_{N-1}}^{\Omega_N} P_N(\omega) d\omega,$$

where

$$E = \int_0^{\Omega_N} P(\omega) d\omega.$$

Hence, code generation is then based on the following steps:

(i) Rounding to the nearest integer the (real) values of E_i to decimal integer form:

$$e_i = \text{round}(E_i), \quad \forall i.$$

(ii) Decimal integer to binary string conversion:

$$b_i = \text{binary}(e_i).$$

(iii) Concatenation of the binary string array b_i to a binary sequence:

$$f_j = \text{cat}(b_i).$$

The watermark f_j is then chirp coded as discussed previously.

Wavelet Decomposition

The wavelet transform is defined by

$$\hat{W}[f(t)] = F_L(t) = \int f(\tau) w_L(t,\tau) d\tau$$

where

$$w_L(t,\tau) = \frac{1}{\sqrt{\mid L \mid}} w\left(\frac{t-\tau}{L}\right).$$

The wavelet transformation is essentially a convolution transform in which $w(t)$ is the convolution kernel but with a factor L introduced.

The introduction of this factor provides dilation and translation properties into the convolution integral (which is now a function of L) that gives it the ability to analyse signals in a multi-resolution role. The code generating method is based on computing the energies of the wavelet transformation over N levels.

Thus, the signal $f(t)$ is decomposed into wavelet space to yield the following set of functions:

$$F_{L_1}(\tau), \quad F_{L_2}(\tau), \quad \ldots \quad F_{L_N}(\tau).$$

The (percentage) energies of these functions are then computed, i.e.

$$E_1 = \frac{100}{E} \int \mid F_{L_1}(\tau) \mid^2 d\tau,$$

$$E_2 = \frac{100}{E} \int \mid F_{L_1}(\tau) \mid^2 d\tau,$$

$$\vdots$$

$$E_N = \frac{100}{E} \int \mid F_{L_N}(\tau) \mid^2 d\tau,$$

where

$$E = \sum_{i=1}^{N} E_i.$$

The method of computing the binary sequence for chirp coding from these energy values follows that described in the method of power spectral decomposition.

Clearly, whether applying the power spectral decomposition method or wavelet decomposition, the computations are undertaken in digital form using a DFT and a DWT (Discrete Wavelet Transform) respectively.

4.1.10 MATLAB Application Programs

Two prototype MATLAB programs have been developed to implement the watermarking method discussed. The *coding process* reads in a named file, applies the watermark to the data using wavelet decomposition and writes out a new file using the same file format. The *Decoding process* reads a named file (assumed to contain the watermark or otherwise), recovers the code from the watermarked data and then recovers the (same or otherwise) code from the watermark.

The coding program displays the decimal integer and binary codes for analysis. The decoding program displays the decimal integer streams generated by the wavelet analysis of the input signal and the stream obtained by processing the signal to extract the watermark code or otherwise. This process also provides an error measure based on the result

$$e = \frac{\sum_i | x_i - y_i |}{\sum_i | x_i + y_i |}$$

where x_i and y_i are the decimal integer arrays obtained from the input signal and the watermark (or otherwise).

In the application considered here, the watermarking method has been applied to audio (.wav) files in order to test the method on data which requires that the watermark does not affect the fidelity of the output (i.e. audio quality). Only a specified segment of the data is extracted for watermarking which is equivalent to applying and off-set to the data. The segment can be user defined and if required, form the basis for a (private) key system. In this application, the watermarked segment has been 'hard-wired' and represents a public key. The wavelets used are Daubechies wavelets computed using the MATLAB wavelet toolbox. However, in principle, any wavelets can be used for this process and the actual wavelet used yields another feature that can form part of the private key required to extract the watermark.

Coding Process

The coding process is compounded in the following basic steps:

Step 1: Read a .wav file.

Step 2: Extract a section of a single vector of the data (note that a .wav contains stereo data, i.e. two vectors).

Step 3: Apply wavelet decomposition using Daubechies wavelets with 7 levels. Note, that in addition to wavelet decomposition, the approximation coefficients for the input signal are computed to provide a measure on the global effect of introducing the watermark into the signal. Thus, 8 decomposition vectors in total are generated.

Step 4: Compute the (percentage) 'energy values'.

Step 5: Round to the nearest integer and convert to binary form.

Step 6: Concatenate both the decimal and binary integer arrays.

Step 7: Chirp code the binary sequence.

Step 8: Scale the output and add to the original input signal.

Step 9: Re-scale the watermarked signal.

Step 10: Write to a file.

In the MATLAB code that follows, the above procedure has been implemented where the parameters for segmenting and processing data of a specific size have been 'hard wired'.

```
%read .wav audio file
[au2,fs,nbit]=wavread('wavefile');

%clear screen
clc

%Set data size (arbitrary) to be watermarked (assumed to be less than
or equal to data in file).
data_size=1500150;

%Extract single set of data composed of 1500150 (arbitrary) elements.
au1=au2(1:data_size,1);

%Set scaling factor.
div_fac=270;

%Set data segment origin.
data_seg=300031;

%Extract segment of data from data_seg to data_size and
%compute the maximum value.
au=au1(data_seg:data_size,1);
au_max1=max(au1(data_seg:data_size,1));
```

```
%Apply wavelet decomposition using Daubechies (4) wavelets with 7 levels.
[ca cl]=wavedec(au(:,1),7,'db4');

%Compute the approximation coefficients at level 7.
appco=appcoef(ca,cl,'db4',7);

%Determine coefficients at each level.
detco7=detcoef(ca,cl,7);
detco6=detcoef(ca,cl,6);
detco5=detcoef(ca,cl,5);
detco4=detcoef(ca,cl,4);
detco3=detcoef(ca,cl,3);
detco2=detcoef(ca,cl,2);
detco1=detcoef(ca,cl,1);

%Compute the energy for each set of coefficients.
ene_appco=sum(appco.^2);
ene_detco7=sum(detco7.^2);
ene_detco6=sum(detco6.^2);
ene_detco5=sum(detco5.^2);
ene_detco4=sum(detco4.^2);
ene_detco3=sum(detco3.^2);
ene_detco2=sum(detco2.^2);
ene_detco1=sum(detco1.^2);

%Compute the total enegy of all the coefficients.
tot_ene=round(ene_detco7+ene_detco6+ene_detco5+ene_detco4...
       +ene_detco3+ene_detco2+ene_detco1);

%Round towards nearest integer the percentage energy of each set.
pene_hp7=round(ene_detco7*100/tot_ene);
pene_hp6=round(ene_detco6*100/tot_ene);
pene_hp5=round(ene_detco5*100/tot_ene);
pene_hp4=round(ene_detco4*100/tot_ene);
pene_hp3=round(ene_detco3*100/tot_ene);
pene_hp2=round(ene_detco2*100/tot_ene);
pene_hp1=round(ene_detco1*100/tot_ene);

%Do decimal integer to binary conversion with at least 17 bits.
```

```
tot_ene_bin=dec2bin(tot_ene,31);
f7=dec2bin(pene_hp7,17);
f6=dec2bin(pene_hp6,17);
f5=dec2bin(pene_hp5,17);
f4=dec2bin(pene_hp4,17);
f3=dec2bin(pene_hp3,17);
f2=dec2bin(pene_hp2,17);
f1=dec2bin(pene_hp1,17);

%Concatenate the arrays f1, f2, ... along dimension 2 to
%produce binary sequence - the watermark wmark.
wmark=cat(2,tot_ene_bin,f7,f6,f5,f4,f3,f2,f1);

%Concatenate decimal integer array.
per_ce=cat(2,tot_ene,pene_hp7,pene_hp6,pene_hp5,...
           pene_hp4,pene_hp3,pene_hp2,pene_hp1);


%Write out decimal integer and binary codes.
d_string=per_ce
b_string=wmark

%Assign -1 to a 0 bit and 1 for 1 bit.
for j=1:150
   if str2num(wmark(j))==0
      x(j)=-1;
   else
      x(j)=1;
   end
end

%Initialize and compute chirp function using a log sweep.
t=0:1/44100:10000/44100;
y=chirp(t,00,10000/44100,100,'log');

%Compute +chirp for 1 and -chirp for 0, scale by div_fac and concatenate.
znew=0;
for j=1:150
   z=x(j)*y/div_fac;
```

```
   znew=cat(2,znew,z);
end

%Compute length of znew and watermark signal.
znew=znew(2:length(znew));
wmark_sig=znew'+au1;

%Compute power of watermark and power of signal.
w_mark_pow=(sum(znew.^2));
sig_pow=(sum(au1.^2));

%Rescale watermarked signal.
wmark_sig1=wmark_sig*au_max1/max(wmark_sig);

%Concatenate and write to file.
wmark_sig=cat(2,wmark_sig1,au2(1:data_size,2));
wavwrite(wmark_sig,fs,nbit,'wm_wavefile');
```

Decoding process

The decoding process is as follows:

Step 1: Steps 1-6 in the coding processes are repeated.

Step 2: *Correlate the data with a chirp identical to that used for chirp coding.*

Step 3: Extract the binary sequence.

Step 4: Convert from binary to decimal.

Step 5: Display the original and reconstructed decimal sequence.

Step 6: Display the error.

```
%Read watermarked file and clear screen.
[au,fs, nbit]=wavread('wm_wavefile');

%Set parameters for data processing.
```

```
data_size=1500150;
data_seg=300031;

%Extract data.
au1=au(data_seg:data_size,1);

%Do wavelet decomposition.
[ca cl]=wavedec(au1,7,'db4');

%Extract wavelet coefficients.
appco=appcoef(ca,cl,'db4',7);
detco7=detcoef(ca,cl,7);
detco6=detcoef(ca,cl,6);
detco5=detcoef(ca,cl,5);
detco4=detcoef(ca,cl,4);
detco3=detcoef(ca,cl,3);
detco2=detcoef(ca,cl,2);
detco1=detcoef(ca,cl,1);

%Compute energy of wavelet coefficients.
ene_appco=sum(appco.^2);
ene_detco7=sum(detco7.^2);
ene_detco6=sum(detco6.^2);
ene_detco5=sum(detco5.^2);
ene_detco4=sum(detco4.^2);
ene_detco3=sum(detco3.^2);
ene_detco2=sum(detco2.^2);
ene_detco1=sum(detco1.^2);

%Compute total energy factor.
tot_ene=round(ene_detco7+ene_detco6+ene_detco5+ene_detco4...
   +ene_detco3+ene_detco2+ene_detco1);

%Express energy values as percentage of total.
%energy and round to nearest integer.
pene_hp7=round(ene_detco7*100/tot_ene);
pene_hp6=round(ene_detco6*100/tot_ene);
pene_hp5=round(ene_detco5*100/tot_ene);
pene_hp4=round(ene_detco4*100/tot_ene);
```

```
pene_hp3=round(ene_detco3*100/tot_ene);
pene_hp2=round(ene_detco2*100/tot_ene);
pene_hp1=round(ene_detco1*100/tot_ene);
per_ene=cat(2,tot_ene,pene_hp7,pene_hp6,pene_hp5,...
            pene_hp4,pene_hp3,pene_hp2,pene_hp1);

%Output original decimal integer code obtained from
%signal via wavelet decomposition.
original_d_string=per_ene;
original_d_string
orig=original_d_string;

%Compute chirp function.
t=0:1/44100:10000/44100;
y=chirp(t,00,10000/44100,100,'log');
%Correlate input signal with chirp and recover sign.
for i=1:150
   yzcorr=xcorr(au(10000*(i-1)+1:10000*i),y,0);

   r(i)=sign(yzcorr);
end
%Recover bit stream.
 for i=1:150
   if r(i)==-1
      recov(i)=0;
   else
      recov(i)=1;
   end
end
%Convert from number to sring.
recov=(num2str(recov,-8));

%Covert from binary to decimal and concatenate.
rec_ene_dist=cat(2,bin2dec(recov(1:31)),bin2dec(recov(32:48)),...
bin2dec(recov(49:65)),bin2dec(recov(66:82)),bin2dec(recov(83:99)),...
bin2dec(recov(100:116)),bin2dec(recov(117:133)),bin2dec(recov(134:150)));

%Write out reconstructed decimal integer stream recoverd from watermark.
reconstructed_d_string=rec_ene_dist;
```

```
reconstructed_d_string
rec=reconstructed_d_string;

%Write out error between reconsructed and original watermark code.
error=sum(abs(rec-orig))/sum(abs(rec+orig))
```

4.1.11 Discussion

In a practical application of this method for authenticating audio files for example, a threshold can be applied to the error value. If and only if the error lies below this threshold is the data taken to be authentic.

The prototype MATLAB programs provided have been developed to explore the applications of the method for different signals and systems of interest to the user. Note that in the decoding program, the correlation process is carried out using a spatial cross-correlation scheme (using the MATLAB function *xcorr*), i.e. the watermark is recovered using the process $\text{chirp}(t) \odot s(t)$ instead of the Fourier equivalent $\text{CHIRP}^*(\omega)S(\omega)$ where CHIRP and S are the Fourier transforms of chirp and s respectively (in digital form of course). This is due to the fact that the 'length' of the chirp function is significantly less than that of the signal. Application of a spatial correlator therefore provides greater computational efficiency. The method of digital watermarking discussed here makes specific use of the chirp function. This function is unique in terms of its properties for reconstructing information (via application of the Matched Filter) that has been 'diffused' through the convolution process, i.e. the watermark extracted is, in theory, an exact band-limited version of the original watermark as defined in the presence of significant additive noise, in this case, the signal into which the watermark is 'embedded'.

The approach considered here allows a code to be generated directly from the input signal and that same code used to watermark the signal. The code used to watermark the signal is therefore self-generating. Reconstruction of the code only requires a correlation process with the watermarked signal to be undertaken. This means that the signal can be authenticated without access to an external reference code. In other words, the method can be seen as a way of authenticating data by extracting a code (the watermark) within a code (the signal). For example, audio data watermarking schemes rely on the imperfections of the human audio system. They exploit the fact that the human auditory system is insensitive to small amplitude changes, either in the time or frequency domains, as well as insertion of low amplitude time domain echos.

Spread spectrum techniques augment a low amplitude spreading sequence which can be detected via correlation techniques. Usually, embedding is performed in high amplitude portions of the signal, either in the time or frequency domains. A common pitfall for both types of watermarking systems is their intolerance to detector de-synchronization and deficiency of adequate methods to address this problem during the decoding process.

Although other applications are possible, chirp coding provides a new and novel technique for fragile audio watermarking. In this case, the watermarked signal does not change the perceptual quality of the signal. In order to make the watermark inaudible for example, the chirp generated is of very low frequency and amplitude. Using audio files with sampling frequencies of over 1000Hz, a logarithmic chirp can be generated in the frequency band of 1-100Hz. Since the human ear has low sensitivity in this band, the embedded watermark will

not be perceptible. Depending upon the band and amplitude of the chirp, the signal-to-watermark ratio can be in excess of 40dB.

Various forms of attack can be applied which change the distribution of the percentage sub-band energies originally present in the signal including filtering (both low pass and high pass), cropping and lossy compression (e.g. MP3 compression) with both constant and variable bit rates.

In each case, the signal and/or the watermark is distorted enough to register the fact that the data has been tampered with. Further, chirp based watermarks are difficult to remove from the signal since the initial and the final frequency is at the discretion of the user and its position in the data stream can be varied through application of an offset, all such parameters being combined to form a private key.

4.2 Exchanging the Cut-Off Points

In this section we explain how to exchange cut-off points between sender and receiver as detailed in Chapter 3. These points are vital for the reconstruction of the encoded binary bits. In the case of multi-fractal modulation, in order to recover the encoded binary bits, we compare the estimated values of the parameters with the cut-off points. These cut-off points can be considered as private keys and the exact values of the parameters as public keys.

The cut-off points a critical set of decimal numbers which need to be converted to binary form to construct the watermarking signal. After chirp coding, the signal is embedding in the fractal modulated signal.

In what follows, we describe the basic algorithm for coding the cut-off points:

Coding the cut-off points

Step 1: Consider the cut-off decimal numbers to be: CP1, CP2 and CP3.

Step 2: Convert these number into a stream of binary bits.

Step 3: Encode the binary stream using a chirp signal.

Step 4: Scale the output signal and add to the contiguous stream of fractal modulated signals.

Step 5: Re-scale the watermarked signal.

Decoding the cut-off points

The basic steps for extracting the cut-off points from the received fractal modulated signal are as follow:

Step 1: Correlate the received signal with a chirp identical to that used for chirp coding.

Step 2: Extract the binary sequence.

Step 3: Convert the binary sequence to a decimal sequence which gives the set of recovered cut-off points.

Authentication of the received signal is achieved if the receiver is able to read the recovered plaintext. Figure 4.2 shows the block diagram of encoding a plain text file and watermarking the binary coded cut-off points in the resulting fractal modulated signal. Figure 4.3 shows the block digram for recovering the cut-off points and the demodulation of the received signal.

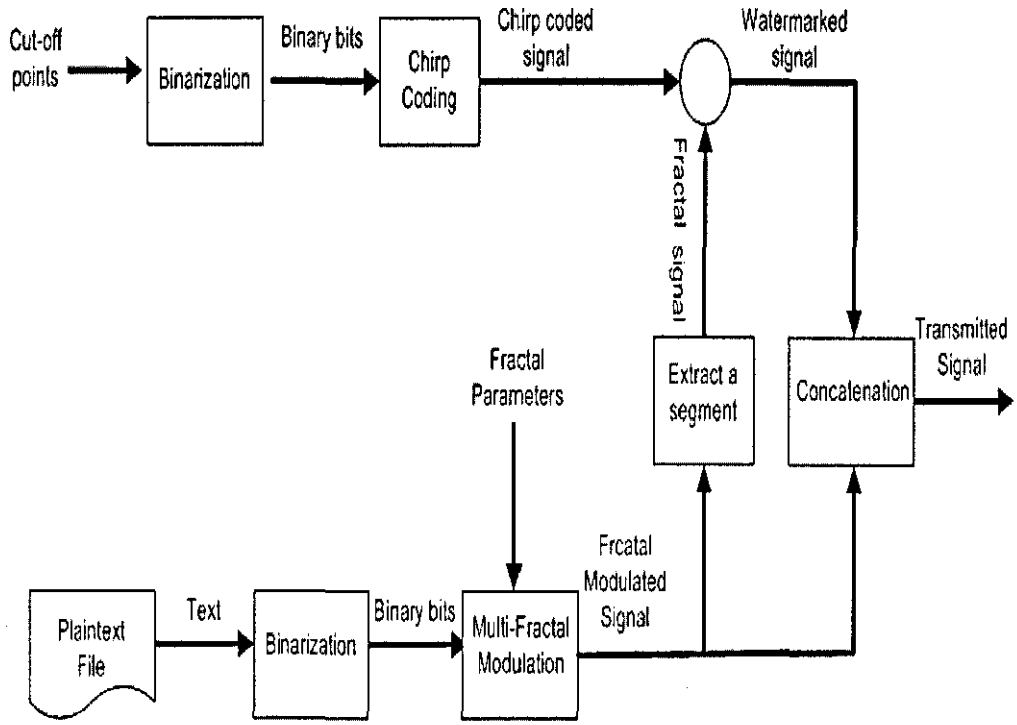

Figure 4.2: Modulation and Watermarking.

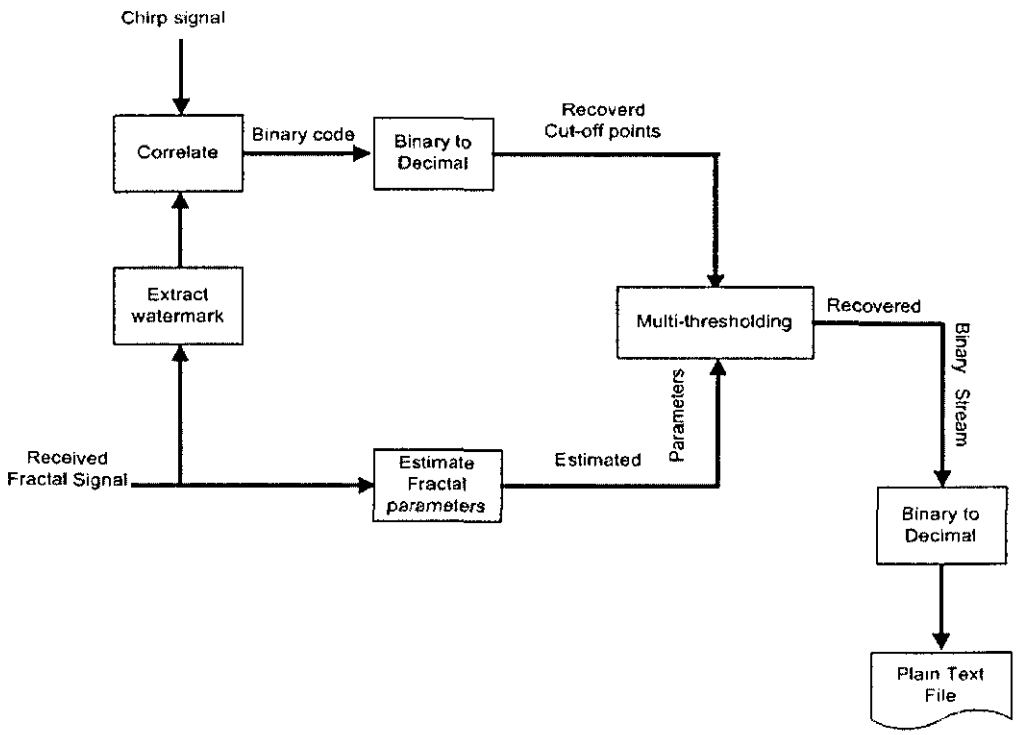

Figure 4.3: Demodulation and authentication.

4.3 References

[1] Cox I.J., Bloom J.A. Miller M.L., *Digital Watermarking*, Morgan-Kaufman, 2002.

[2] Katzenbaiser S. and Petitcolas F.A.P., *Information Hiding Techniques for Stenography and Digital Watermarking*, Artech, 2000.

[3] Petitcolas F.A.P., Anderson R.J. and Kuhn M.G., "Information Hiding: A survey," *Proc. IEEE*, vol. 87, no. 7, pp. 1062-1078, 1999.

[4] Blackledge J.M, *Digital Signal Processing:Mathematical and Computation Methods, Software Development and Applications*, Horwood Publishing Limited, London, 2nd Edition, 2006.

Chapter 5

Internet Traffic Data Analysis

5.1 Introduction

The Internet is, by definition, a collection of networks throughout the world that agree to communicate using specific telecommunications protocols, the most basic being the Internet Protocol(IP) and Transmission Control Protocol (TCP), and the services supplied by these networks. The Internet has become a kind of common infrastructure and plays an important role in various human activities. It has made it possible for people all over the world to effectively and inexpensively communicate with each other. Traffic in the Internet results from the uncoordinated actions and operations of a very large population both of users and network devices. Internet traffic has been increasing exponentially in recent years. The major reason for this increase in traffic is the introduction of new applications on the Internet. These applications range from text-based utilities such as file transfer, electronic mail and network news from the early

days of the Internet, to the advent of desktop video conferencing, multimedia streaming, the world-Wide-Web, Grid computing and electronic commerce on today's Internet. Another reason is that many users have become connected to the Internet and are now actively exchanging data for business or personal use on a routine basis. However, an important feature of Internet traffic is that it is far too complex to be understood and controlled in details. The complexity of its traffic has a large number of causes, which may be divided into two classes. The first class stems from the network and its enormous current size spanning the whole planet with, in many places, a very fine structure. The second class is directly linked with the flow of the data. This results in several kinds of complexity such as the different multi-layered protocols and their hierarchies (e.g. Transmission Control Protocol (TCP), Internet Protocol (IP), HyperText Transport Protocol (HTTP)) which all have their own dynamics, mechanisms, and timescales.

Traffic sources are themselves distributed in a very inhomogeneous way, with high concentrations around universities and research centers, etc. Furthermore, protocols split and combine data so that streams of packets often correspond to a complex mixture of sources.

5.2 The Concept of Packets

Most computer networks do not transfer a message of data as an arbitrary string of continuous bits. Instead, the network system divides data into small blocks called *packets*, which it sends individually. Typically, one Ethernet packet contains 64 to about 1500 bytes, but larger packet sizes may be used

with Jumbo Frames. All information exchanges - whether a short e-mail message, a large file transfer, or a complicated Web transaction - are broken down into these basic blocks. Computer networks that use packet technology are called *packet networks* or *packet switching networks.*

Two facts motivate the use of packets data as organised and required by network system design. First, a sender and receiver need to coordinate transmission to ensure that data arrives correctly, where the data can be lost when the transmission errors occur. Dividing data into small blocks helps a sender and receiver to determine which blocks arrive intact and which do not. Second, because communication circuits and associated modern hardware are relatively expensive, multiple computers often share underlying connections and hardware.

To ensure that all computers receive fair and prompt access to a shared communication facility, a network system cannot allow one computer to deny access to others. Using small packets ensures fairness, contrary to the alternative used in early computer networks which did not guarantee fair access; and allowed an application program to hold a shared communication resource for an arbitrarily long time - an application was permitted to finish before another application could begin using the resource. Thus, to avoid having one computer holding a network for an arbitrary time, modern computer networks enforce the use of packets.

A network permits one computer to send a packet, then blocks that computer from sending another. Meanwhile the network permits another computer to send a packet, and so on.

A single computer can hold a shared resource only long enough to send a single packet, and must wait until other computers have a turn before sending a second packet [1].

The Internet's infrastructure consists of a series of *routers* interconnected by *links*. Each packet of each network connection is self-contained in the sense that its header contains complete 'addressing' information. Thus, all packets in a single message do not have to travel along the same path. As traffic conditions change, they can be dynamically routed via different paths in the network, and they can even arrive out of order.

The routers along the packet's path inspect the header of each packet they receive in order to determine the next *hop* (either a another router or the destination computer) to which they should forward the packet so that it will ultimately reach its destination. The destination computer reassembles the packets into their proper sequence. Each packet is transmitted independently from the other packets that have already been transmitted or still await transmission; the routers do not keep track of which packets belong to which active connections. Thus, a router can 'forget' about a packet as soon as it has been forwarded.

Sometimes routers receive more packets than they can immediately forward, this means that the number of injected packets exceeds the capacity of the network, so packets are temporarily stored in a *buffer* of a router increasing the delay of the packets through the network. Sometimes, in the worst case, they are dropped because the router's buffer has a finite size; such a state is called *congestion*. Format specifications of individual packets for transmission through the network form one of the Internet's underlying *protocols* (this is

fundamental and is called the Internet Protocol, or IP). Other protocols regulate other aspects of Internet communications, such as how to divide streams of data into individual packets such that the original data can be delivered to the receiving computer, even if some of the individual packets are lost due to drops or damage (a form of transport protocol) which is built on top of IP.

5.3 Internet Traffic Noise

The explosive growth in the Internet data traffic has made the study of the nature and statistical characteristics of this type of traffic increasingly important. Modern Internet traffic measurements system such as *Tcpdump, Windump* and *Eathrel*, record the time in 1 microsecond at which a packet of information arrives along with other information such as the source address, destination address, etc. and packet size of the data transferred. From this record, a time series of the data transmitted (the number of bytes or packets arriving at a router) per time unit interval (flow density) is then obtained.

5.3.1 Self-affinity of Internet Traffic Noise

Although Internet traffic is purely man-made, it can be studied as a natural phenomenon. Measurement and analysis of Internet traffic has been the subject of interest as long as there has been Internet traffic. In the last decade, there have been several empirical and analytical studies of high resolution traffic measurements from a variety of working packet networks, including Ethernet Local-Area Networks (LANs) [2], Wide Area Network (WAN) [3], World

Wide Web (WWW) data transfer [4] and other communication systems; all have convincingly demonstrated the presence of *self-affine* characteristics.

Self-affinity and *self-similarity* are associated with *fractals* - objects that appear the same regardless of the scale at which they viewed (See Chapter 2). Self-similar objects look the same or behave the same when viewed at different degrees of 'magnification' or different scales over a dimension. This dimension may be space (length, width) or time. With Internet data, this it manifested in the absence of a natural length of a 'burst'; at every time scale, ranging from a few milliseconds to minutes and hours, bursts occur that consist of sub-periods separated by less bursty sub-periods, so that there is no definite duration of 'busy' or 'silent' periods. In other words, a segment of a traffic signal at some time scale looks or behaves like an appropriately scaled version of the traffic measured over a different time scale.

A variety of studies have demonstrated that measured traffic data (i.e. the number of bits, bytes or packets that traverse a given link or node per time unit) in Internet networks exhibits surprising scaling properties over a wide range of time scales; that is, actual network traffic looks statistically the same over small (i.e. over time scales of the order of milliseconds or seconds) and large (i.e. time scales of the order of minutes and beyond) scales and no natural length of a 'burst' is discernible: at every time scale ranging from milliseconds to seconds to minutes and beyond, bursts have the same qualitative appearance and cause the resulting traffic to exhibit *fractal*-like characteristics. These studies have challenged the commonly assumed models for network traffic data (e.g Poisson model). Traditional analysis of network traffic generally neglects the existence of self-similarity. These models do not 'fit' the statistical charac-

teristics of Internet traffic data, primarily because they are not able to capture the fractal behavior of Internet traffic. Were Internet traffic to follow Poisson statistics, it would have a characteristic burst length which would tend to be smoothed over a long enough time scale. However, measurements of real traffic indicate that significant traffic burstiness is present on a wide range of time scales, see Figure 5.1 . Since a self-similar process has observable bursts on all time scales, it exhibits *Long-Range Dependence (LRD)*; values at any instant are typically correlated with values at all future instances.

Many queuing models based on Poisson or Markovian processes often underestimate the burstiness of traffic. Such models have a characteristic burst length which tends to be smoothed out by averaging over a long enough time scale. These traffic models do not posses long range characteristics. However, real traffic measurements show that significant traffic burstiness is observed on a wide range of time scales. Thus Markovian models which are classically used for network design cannot capture the observed long range characteristics of the actual network data traffic. Fluctuations occur on the data traffic traces that may also appear in very short intervals resulting in bursts. Bursty traffic, which may be observed quite easily in actual network traffic, cannot be modeled properly using traditional traffic models [5]. Understanding the nature of network traffic is critical in order to properly design and implement computer networks and network services. Recent studies have revealed that self-similarity has a profound impact on network performance. The original study on the self-similar behavior in measured network traffic was reported in [2] and was based on an extensive statistical analysis of traffic measurements of Bellcore's Ethernet traffic Local Area Network (LAN) over four periods (ranging from 21 to 48 hours) from 1989-1993.

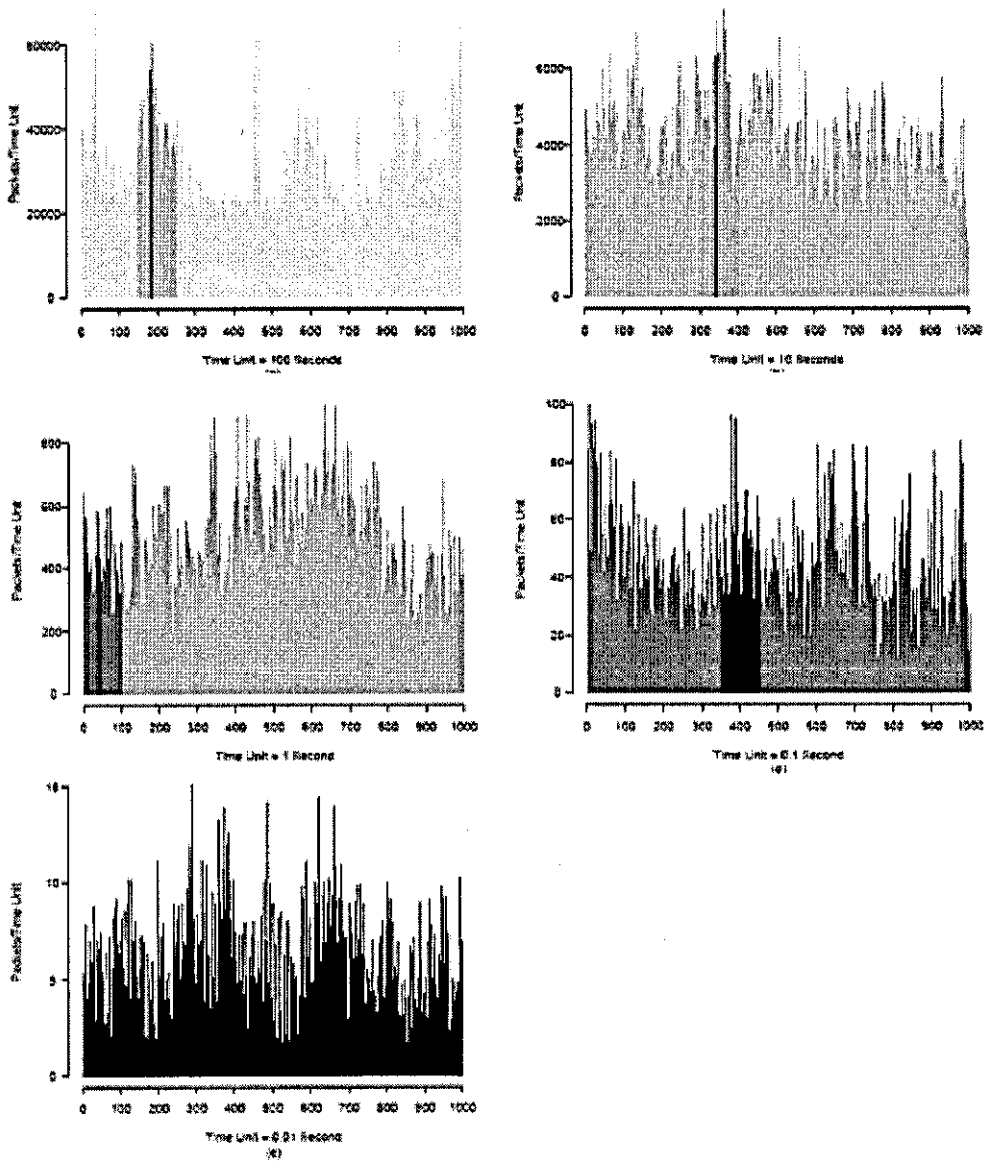

Figure 5.1: Pictorial 'proof' of self-similarity: LAN traffic over five different time units (from [2]).

The study given in [2] illustrated some of the most striking differences between self-similar models and the standard models for packet traffic considered in the literature. The statistical patterns were found to be essentially the same for all periods, despite tremendous changes in the network. Moreover, the measured traffic satisfied the criteria for self-similarity. The fluctuations in the number of bytes or packets transmitted per time unit were statistically the same, whether the time units were minutes or milliseconds (see, Figure 5.1). In simple terms, the measured internet traffic behaved in a self-similar way which intuitively means that the traffic is 'similar to itself' on all time scales. The self-similarity of the traffic relates to, and is define in terms of, aggregates of the time series.

A number of important follow-up studies have provided further evidence of the prevalence of self-similar traffic patterns in measured traffic from Wide Area Networks (WANs) (e.g. [3] and [4]). On the other hand, in [4] it is observed that traffic due to WWW transfers show characteristics that are consistent with self-similarity. They trace the genesis of Web traffic self-similarity to the heavy-tailed distribution of available file sizes in the Web. It has been argued in [5] that transferring files whose sizes are drawn from a heavy-tailed distribution will generate self-similarity in network traffic [4], [13]. However, the fundamental stochastic explanation of these phenomena still represents an open research issue.

All studies given in the open literature provide inspirations for investigating new types of traffic models. In particular, authors have encouraged the use of stochastic processes which have distributions with infinite moments. Self-similar structures and hence fractal processes have been employed for this reason.

The issue of self-similarity has also been addressed in various studies from many aspects including its impact on network performance [15], modelling techniques [16], [15] and causes for the appearance of self-similarity (e.g. [4], [17], [18]).

In the following section, we describe, from a mathematical viewpoint, the concept of self-similar processes, discuss their most important mathematical and statistical properties, modelling approaches and outline methods for analyzing self-similar data.

5.4 The Mathematical Description of Self-Similarity and Fractality

We present a brief description of the mathematics of self-similarity in term of both continuous-time and discrete-time stochastic processes but concentrating on this phenomenon in terms of a discrete-time series.

5.4.1 Continuous-Time Process

A continuous-time stochastic process $X = \{X(t), \quad t \geq 0\}$ is considered to be *statistical self-similar* (more precisely, *self-affine*) if

$X(t) \overset{D}{\approx} a^{-H}X(at)$ for any $a > 0$ and $(0.5 \leq H \leq 1)$

The symbol $\overset{D}{\approx}$ means 'asymptotically equal to' in the sense of finite-dimensional distributions (i.e. same statistical properties). Here, the parameter H is called the *Hurst parameter* or the *index of self-similarity* which indicates the 'degree'

of self-similarity, i.e. the degree of persistence of the statistical phenomenon. A value of $H = 0.5$ indicates the lack of self-similarity, whereas large values for H (close to 1) indicates a large degree of self-similarity in the process.

Self-affinity indicates that the graph $(t, X(t))$ remains statistically unchanged when the time and the amplitude axes are simultaneously scaled by a factor a and a^H, respectively. We say that $X(t)$ has the same *scaling behavior* on all time scales. Practically, statistical self-similarity means that the following conditions are valid [8]:

- mean: $E[X(t)] = a^{-H}E[X(at)]$.
- variance: $var(X(t)) = a^{-2H}var[X(at)]$.
- autocorrelation function: $R_X(t,\tau) = a^{-2H}R_X(at, a\tau)$.

5.4.2 Discrete-Time Process

Let $X = \{X_t : t = 0, 1, 2, 3, ..., \}$, be a discrete-time stochastic process (stationary time-series). X_t represents; for example, the number of bytes or packets per time interval observed in a given link of a network. Define another time series (m-aggregated) $X^{(m)} = \left\{X_k^{(m)}, k = 1, 2, 3, ...\right\}$ where $X^{(m)}$ is obtained by averaging the original series X over non-overlapping adjacent blocks of size m, i.e. replacing each block by its sample mean. That is $X^{(m)}$, for each $m = 1, 2, 3, ...$, is given by

$$X_k^{(m)} = \frac{1}{m} \sum_{i=km-m+1}^{km} X_i, \qquad k = 1, 2, 3, ...$$

In this case $X^{(1)}$ represents the highest resolution that is possible for the process. Lower resolution evolutions of the process $X^{(m)}$ can be obtained by m-aggregating the X process, for instance

$$X_k^{(4)} = (X_{4k-3} + X_{4k-2} + X_{4k-1} + X_{4k})/4.$$

The process $X_k^{(4)}$ therefore represents a less-detailed copy of the process $X^{(1)}$. In the case that statistical properties (e.g. mean, variance) are preserved with aggregation, then the process is of a self-similar nature.

There are two classes of self-similar processes, namely, *exactly self-similar* and *asymptotical self-similar* processes. Process X is said to be *exactly self-similar* with parameter β $(0 < \beta < 1)$ if for all $m = 1, 2, 3, ...$ the following conditions are fulfilled:

- variance $var[X^{(m)}] = m^{-\beta} var[X]$.
- autocorrelation function , $R(X^{(m)}, k) = R(X, k)$.

Note that this means that the series is *distributionally* self-similar; the distribution of the aggregated series is the same (except for changes in scale) as that of the original. Another class of self-similar process is the so called *asymptotically self-similar* process if we meet conditions with large values of k and $m = 1, 2, 3, ...$:

- variance $var[X^{(m)}] = m^{-\beta} var[X]$.
- autocorrelation function , $R(X^{(m)}, k) \longrightarrow R(X, k)$.

This feature is in contrast to stochastic processes where the autocorrelation function degenerates as $k \longrightarrow \infty$. As a result of this, self-similar processes can show *long-range dependence (LRD)*. Long-range dependence in time series is the presence of a significant correlation between observations of signals separated by large time spans. It is closely linked with self-similar stochastic processes and random fractals which have been considered extensively, though only recently, for signal processing applications.

A process with long-range dependence has an autocorrelation function $R(k) \approx k^{-\beta}$ as $k \rightarrow \infty$, where $0 < \beta < 1$. Thus, the autocorrelation function of such a process follows a power law, as compared to the exponential decay exhibited by traditional models. One of the attractive features of using self-similar models for time series, when appropriate, is that the degree of self-similarity of a series is expressed using only a single parameter. This parameter expresses the speed of decay of the series autocorrelation function. Thus, for self-similar series with long-range dependence, $1/2 < H < 1$. As $H \rightarrow 1$, the degree of both self-similarity and long-range dependence increases.

The parameter β is related to the Hurst parameter by $\beta = 2(1 - H)$. Fractional Gaussian noise with $1/2 < H < 1$ is the standard example of an exactly self-similar (Gaussian) process with self-similar parameter H. Generally self-similarity means that when a discrete time or continuous time stochastic process is scaled in time, similar patterns can be seen. The process in large scale is a copy of itself in smaller timescales.

5.4.3 Principal Properties

Mathematically, self-similarity manifests itself in a number of equivalent ways:

(i)The variance, $Var(X^{(m)})$, of the sample mean decreases more slowly than the reciprocal of the sample size (*slowly decaying variances*), i.e.

$$Var(X^{(m)}) \approx c_1 m^{-\beta}, \qquad 0 < \beta < 1, \quad c_1 > 0, \quad m \to \infty.$$

(ii) The autocorrelation function decays hyperbolically rather than exponentially fast, implying a non-summable autocorrelation function $\sum_k R(k) = \infty$ (*long range dependence*).

(iii) The power spectral density function $P(\omega)$ obeys a power-law ($1/\omega - noise$) that is $P(\omega) \approx c_2 \omega^{-\lambda}$ with $0 < \lambda < 1$ and $\lambda = 1 - \beta, \qquad c_2 > 0$.

Thus, we can conclude that, for both classes of self-similar processes, the variance $Var[X^{(m)}]$ decreases more slowly than $1/m$ as $m \longrightarrow \infty$. This can be compared to stochastic processes where the variances decreases in proportion to $1/m$ and approaches zero as $m \longrightarrow \infty$. The most striking feature of a self-similar processes is, however, the fact that the autocorrelation function does not degenerate when $m \longrightarrow \infty$.

The effect of self-similarity in network traffic is shown in [14], which compares a self-similar series with a compound Poisson series having the same distributional characteristics. It is shown that Poisson models for network aggregated traffic become essentially uniform when aggregated by a factor of 1000; while actual network traffic shows no such decrease in variability over the same range of aggregation.

5.5 Fractal Characteristics of Internet Traffic

As discussed in Chapter 2, the concept of fractals (*mono-fractals and multi-fractals*) was first introduced by Mandelbrot in the early 1970's. Since then, fractal processes have been widely used in a variety of research fields such as image processing, signal processing, stock market modelling, and recently network traffic characterization. It is indicated in [11] that we define a process to posses fractal (mono-fractal) characteristics, if there exist a relationship of the form

$$Q(\tau) \propto \tau^D, \tag{5.1}$$

where Q is a certain quantity of the underlying process that depend on τ and D. Herein, τ denotes a resolution in *time* or *space* of observation variables at which Q is evaluated, and D is a *fractal dimension*.

To declare *fractality*, the above relationship is supposed to hold for a range of different τ-values, with a value of D that is less than the embedded dimension (i.e. $D < 2$). Due to the extreme variability the Internet traffic data exhibiting such fractal-like structures over almost all scales of resolution, the fractal characteristics can exist both in *temporal* and *spatial* scales. Indeed, one of measures of self-similarity is based on such an equation. When Q is the variance of data then D is $-\beta$, so that the fractal dimension D is identified to be β via

$$Var(X^{(m)}) \propto m^{-\beta} = m^{2H-2}$$

which describes fractal behavior of data in time.

In the case of *multi-fractals* we refer to the situation where the exponent β varies from one range of scales (m) to another.

It is also possible to observe that a similar description of a time series data X or its m-aggregation $X^{(m)}$ can be found with the resolution in the magnitude of data [12]. Assume that a range of data is to be divided into equal segments (windows) of size ϵ, and count the number of segments, say $N(\epsilon)$, that contain the data. Equation(5.1) can be written as

$$N(\epsilon) \propto \epsilon^{-D}$$

and the fractal dimension D at resolution ϵ can be obtained from the slope of plotting $\ln N(\epsilon)$ versus $\ln \epsilon$, i.e.

$$D = \frac{-\ln N(\epsilon)}{\ln \epsilon}. \tag{5.2}$$

Note that equation (5.2) is expected to give the dimension for a range of scales depending on the size of the data. The time series X or $X^{(m)}$ is fractal if equation (5.2) is valid over appreciable range of scales. For random data (not fractal), and since all values are equally likely to occur, D=1. Thus, equation (5.2) gives an indication that all values in the range of data are not equally probable if D is fractional. As in all natural causes, it is to be noted that equation (5.2) is expected to give the dimension for the average of scales depending on the size and accuracy of data. Multi-fractals can exist with exponent D's, depending on the scales (ϵ) of observation.

5.5.1 Why is Internet Traffic Fractal?

The reasons behind self-similarity in network traffic have not been clearly identified. Since the pioneering work on self-similarity of network traffic by Leland et. al. [2], many studies have attempted to determine the cause of

this phenomenon. Initial efforts focused on application factors. For example, in [4], the authors investigated, in some cases, the cause of self-similarity by focusing on the variability in the size of documents transfered and the inter-request time. They proposed that the heavy-tailed distribution of a file size and 'user think time' might potentially be the cause of self-similarity (fractality) found in WWW traffic. In other words, the behavior of self-similarity in the traffic of World Wide Web transfers can be explained in terms of file system characteristics and user behavior.

Other studies have show that the irregular ON/OFF nature of the Internet communications with heavy - tailed file sizes can result in self-similar processes [13], [14]. As Mandelbrot found as early as the 1960s, a large number of independent ON/OFF sources, when taken together, produce self-similarity, provided the ON and OFF cycles of each source are themselves (heavy-tailed); this is precisely what the studies reported in [13] and [14] found for the cases considered.

The purpose of the protocols, as used for Internet traffic, is to establish on orderly transfer of information. They do this in part by adjusting the rate at which packets are sent to correspond to the rate at which they arrive. A few studies have considered the possibility that underlying network protocols such as TCP can cause or exacerbate the phenomenon, whereas, others explain that such protocols may cause departure from self-similarity, at least on small time scales. Anna Gilbert [19], describes wavelet-based scaling analysis of simulated Internet traffic in which multi-fractal behavior was found to results from TCP.

```
TIMESTAMP SOURCE PORT DESTINATION PORT FLAG SEQNUM ACKNUM
19:52.731470 406.17.8.12.64826 > 723.65.19.6.www: S 4256930:4256930(0)
19:52.731889 723.65.19.6.www > 406.17.8.12.64826: S 768500:768500(0) ack 4256931
19:52.732200 406.17.8.12.64826 > 723.65.19.6.www: . ack 768501 win 17520
19:52.738205 406.17.8.12.64826 > 723.65.19.6.www: P 4256931:4257101(170) ack 768501
19:52.743248 723.65.19.6.www > 406.17.8.12.64826: P 768501:5769840(1339) ack 4257101
19:52.758535 406.17.8.12.64826 > 723.65.19.6.www: F 4257101:4257101(0) ack 5769840
19:52.758862 723.65.19.6.www > 406.17.8.12.64826: . ack 4257102
19:52.759700 723.65.19.6.www > 406.17.8.12.64826: F 5769840:5769840(0) ack 4257102
19:52.759935 406.17.8.12.64826 > 723.65.19.6.www: . ack 5769841
.
.
.
.
.
```

Figure 5.2: Tcpdump Trace format.

The key finding of the study that was conducted in [19] is that user-related variability results in self-similar scaling over large time scales, wheres the presence of TCP and other flow control algorithms underlies a transition to more complex multi-fractal scaling over small time scales.

5.6 Network Traffic Measurements

A stream of packets between a particular host and server is often described as a *flow* or a *trace* of network traffic; if we wish to know how the Internet is being used, we have to measure these traffic *flows*. The data set considered in this work was a collection of measurements made of a network in place at the Loughborough University (LU) campus in March 2006.

As part of the University network, a number of LANs belong to the Halls of Residence (HR). We considered all workstations (approximately 4500) that belong to HR. All packet headers have been captured simply using the *tcpdump* utility, which is widely available in Unix environments. The monitoring station for running *tcpdump* was at the intrusion detection server in the Computing Services Department.

Among all the possible traffic characteristics, the work focused on two properties: *packet size* and *packet timestamp*. Measurements were made over the course of one hour, and records made of all packets arriving at or originating from the host site.

By way of an example, we consider three traces at three different times on March 30, 2006, namely, at 10:30 AM (LU-HR-1 trace), 14:30 PM (LU-HR-2 trace), and 2:30 AM (LU-HR-3 trace)[20]. These time periods were selected because they correspond with periods of Low (L), Medium (M) and High (H) network utilization, respectively. Table 5.1 gives a summary description of these measurements. Information is provided on the trace name, the duration time over which data were collected, the total number of packets and bytes collected, maximum and minimum size of the trace packets, and the mean and the standard deviation of the packet sizes.

The data was obtained as an ASCII file which included a number of extraneous data elements, such as IP addresses of sources and destination hosts and TCP ports. These were removed using Perl script (an editor used to removing redundant data).

An example of a *tcpdump* trace is given in Figure 5.2. The trace format shows a timestamp for each packet at a microsecond resolution, source and destination

IP addresses, source and destination port numbers and packet size. The used measurements represent the number of bytes that arrived over the server at the Computing Services and the corresponding timestamps. for their arrivals. The data in its original format was at the scale of $1\mu s$.

This was aggregated to resolve the data at different time scales of 1 s, 0.1 s, 0.01 s and 0.001 s for computational convenience. In other words, since each packet captured with *tcpdump* has a timestamp, the traffic trace is divided up into time intervals (bins) of size 100 ms for example.

For each time interval, the number of packets or bytes that arrived is counted. The resulting time series array with the number packets (or bytes) that arrived in each time interval is considered as the set of observations in this work.

Trace Name	Time	Duration Sec.	Num. of Packets	Num.of Bytes	Max Packet	Min Packet	Mean of Bytes	Std. of Bytes
LU-HR-1 (L)	@10:30	2156	8m	5000m	8408	0	635	10240
LU-HR-2 (M)	@ 14:30	4974	8m	4000m	8406	0	504	997
LU-HR-3 (H)	@ 2:30	4549	8m	5500m	8404	0	701	1092

Table 5.1: Qualitative description of the used data ($m \equiv 10^6$).

5.7 Estimating the Hurst Parameter from Real Network Traffic Measurements

Several methods are commonly used for measuring the self-similarity or the long range dependence of a real network traffic trace. Here we mention two

important examples, the *Rescaled Adjusted Range* plot (R/S plot) and the *Variance Time* plot.

5.7.1 The R/S Method

The R/S method is one of the oldest and best known methods for estimating H. Let X_k, $k = 1, 2, 3, ..., N$ be a set of N observations for the number of bytes or packets in each interval (bin) and let $\overline{X}(N)$ and $S(N)$ be the mean and the standard deviation of these observations.

The R/S-statistic or *rescaled adjusted range* , is defined by the ratio:

$$\frac{R(N)}{S(N)} = \frac{max(0, w_1, w_2, w_3, ..., w_N) - min(0, w_1, w_2, w_3, ..., w_N)}{S(N)}$$

where ,

$$w_k = (X_1 + X_2 + X_3 + ... + X_k) - k.\overline{X}(N), \qquad k = 1, 2, 3, ..., N.$$

Husrt found empirically, that for many time series observed in nature, they are well represented by the relation

$$\frac{R(N)}{S(N)} \approx C.N^H, \qquad N \to \infty$$

where C is a finite positive constant. By taking logs we obtain

$$log(\frac{R(N)}{S(N)}) \approx log(C) + H.log(N).$$

Therefore, the slope of a plot of $log(R/S)$ against $log(N)$ provides the Hurst parameter [6].

5.7.2 Variance-Time Method

This method relies on the slowly decaying variance of a self-similar series. Let X_k be a series of observations for the number of bytes or packets in each interval (bin) $k = 1, 2, 3, ..., N$. If we take a sample of m points then the variance-time plot is obtained by plotting $log[Var(X^{(m)}]$ against $log(m)$ and by fitting simple lines through the resulting points in the plane. An estimate of the Hurst parameter is given by $H = 1 - \beta/2$ where β is slope of the plot [6].

5.8 Experimental Proof of the Fractal Nature of Internet Traffic

A simple way of understanding fractality (self-similarity) is in terms of scale-invariance. Basically, this means that whatever be the time-scale over which the traffic is plotted, the plots will appear (intuitively) very 'similar' to one another.

In each of the figures, i.e. Figure 5.3 to Figure 5.5, there are four time series plots of size 1024 for the Internet bytes traffic induced by the three reference traces: LU-HR-1, LU-HR-2 and LU-HR-3. The horizontal axis represents the time scale; the vertical axis represents traffic load in bytes per unit time. The plots are produced by aggregating the (bytes) traffic into discrete time units (bins) of 1 ms, 10 ms, 100 ms and 1000 ms. The plots of the Internet packets traffic for the same traces are given in Figures 5.6 to 5.8, with the same considerations as those in the plots of bytes traffic whereas the vertical axis represents the number of packets per unit time.

We observe, from the plots given in each figure, that all the plots are 'similar' to each other, i.e. the bytes (or packets) traffic appears to look the same over the whole spectrum of time scales.

Starting from a time scale 1 ms, all subsequent plots are obtained from the previous one by increasing the time resolution by a factor of 10. It is apparent that all the plots appear almost similar and exhibit self-similarity, indicating that different values of magnification give similar plots. In other words, all these plots show clearly the burstiness of the Internet traffic across many time scales (1 ms, 10 ms, 100 ms or 1000 ms). Increasing the time scale (time bin) of observation, say from 1 ms to 1000 ms, does not cause the traffic to 'smooth out' as would normally be expected. Instead, the traffic of all traces continues to exhibit burstiness.

Figure 5.9 shows the plots of sample of size 1024 points of inter-arrival packet time sequences corresponding to the given traces. The horizontal axis represents the sample point, the vertical axis represents the inter-arrival time in seconds. If we look at the plots of the time series of the Internet bytes or packets traffic or the plots of the packets inter-arrival time sequence, we see a similar appearance to the plots in each figure, regardless of time scale; this is the characteristic signature of the fractal behavior of such time series. Notice that, in each plot, the absence of a fixed length of a 'burst'; we observe different burst lengths on different time scales. This experimental observation is consistent with the prior studies.

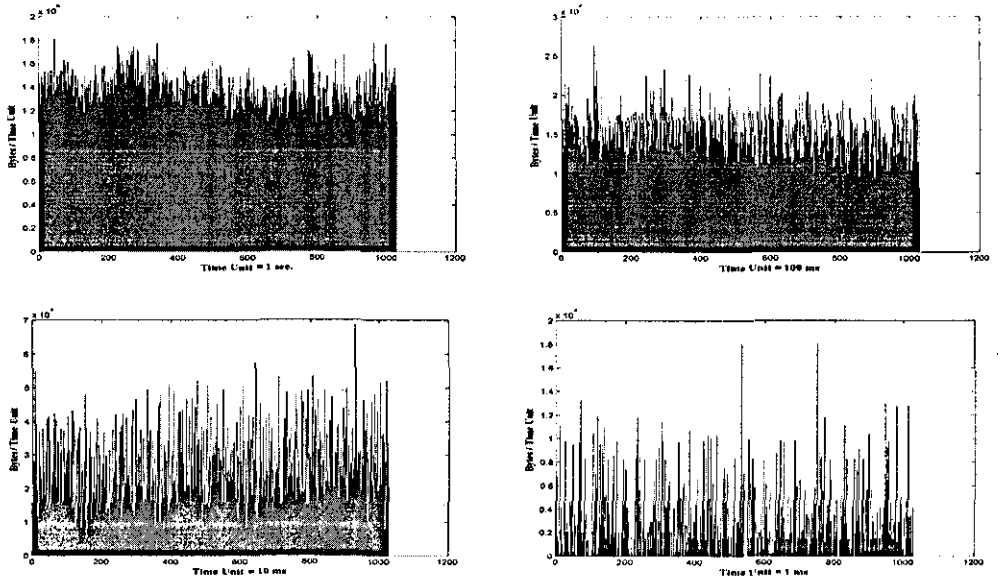

Figure 5.3: Internet Bytes Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000ms (1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace: LU-HR-1.

5.9 Characterization of Internet Traffic Noise using a Fractal Model

There are many models in the literature that consider the main characteristics of Internet traffic noise either by processing real measurements of packets (or bytes) traffic in time or the frequency domain. Fractals are applicable when the underlying processes being modelled have a similar appearance regardless of the time or observation scale and much of the traffic 'riding' the Internet can be modelled using fractals. Further, it is arguable that as the Internet has become larger and larger, the fractal nature of the traffic has become more and more pronounced. In this section we introduce a novel method that depends on the frequency analysis through which we attempt to capture the fractal

behavior of Internet traffic by adopting a Random Scaling Fractal model to characterize the self-affine characteristics of the traffic.

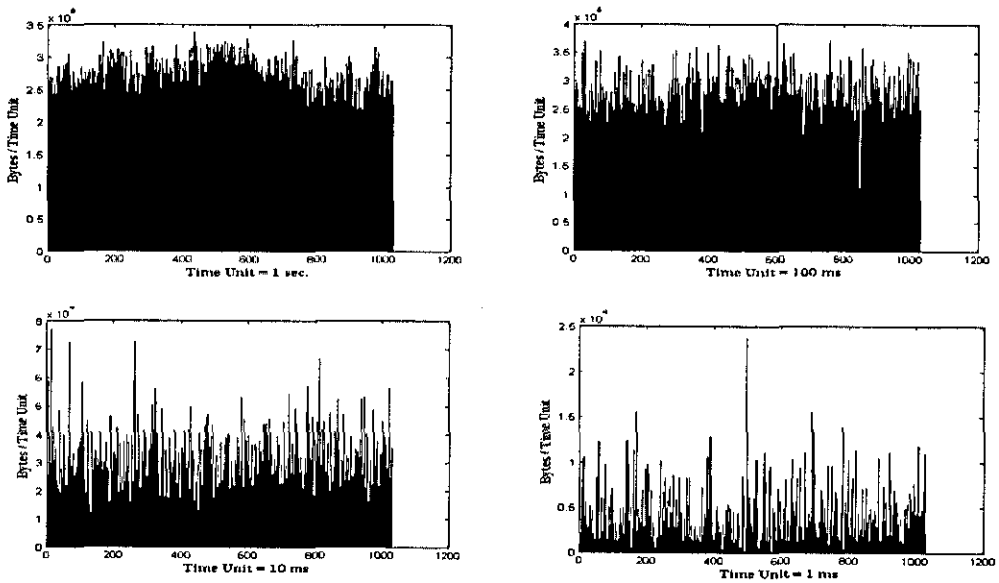

Figure 5.4: Internet Bytes Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000 ms(1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace:LU-HR-2.

5.9.1 Random Scaling Fractal Noise

Many noise signals observed in nature are random fractals. Random Scaling Fractals (RSFs) are signals whose probability distribution function or PDF has the same 'shape' irrespective of the scale over which they are observed. Random fractal noise fields are *statistically self-similar* or *self-affine*; 'they look the same' in a stochastic sense at different scale. As discussed at the beginning of this chapter, Internet traffic time series exhibit the features of self-affinity fields so we can consider such a series to be an example of RSF noise.

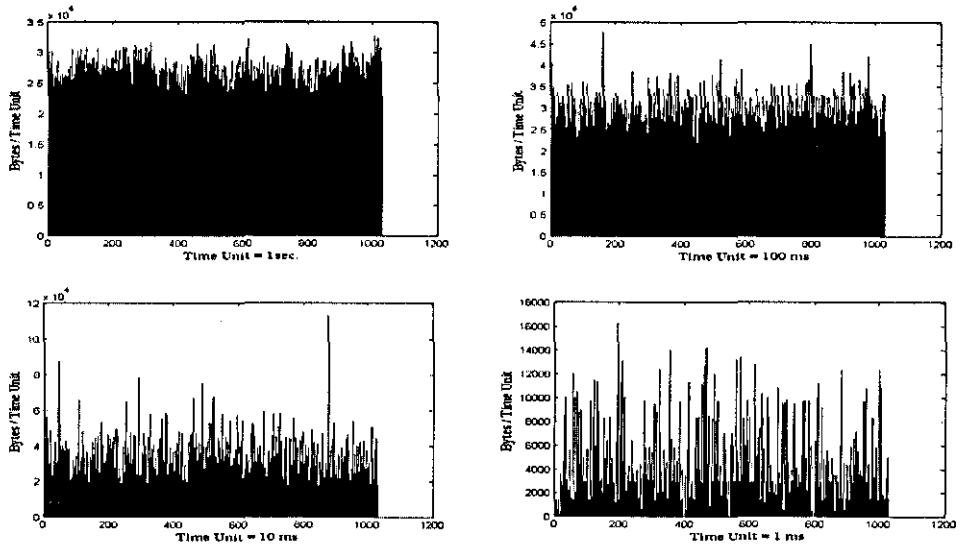

Figure 5.5: Internet Bytes Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000 ms(1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace:LU-HR-3.

RSF noise is characterized by power spectra whose frequency distribution is proportional to $1/\omega^q$ where ω is the frequency and $q > 0$ is the 'Fourier dimension' (see Chapter 2), a value that is simply related to the Fractal Dimension, D and Hurst(Dimension) parameter H. The relationship between the q, H and D is given by (see Chapter 2)

$$q = H + 1/2 = (5 - 2D)/2.$$

This power law describes the conventional RSF model which is based on stationary processes in which the 'statistics' of the RSF signals are invariant of time and the value of q is constant. As discussed earlier in this Chapter, the Hurst parameter (Dimension) H measures the features of self-affinity of a time series in the time domain.

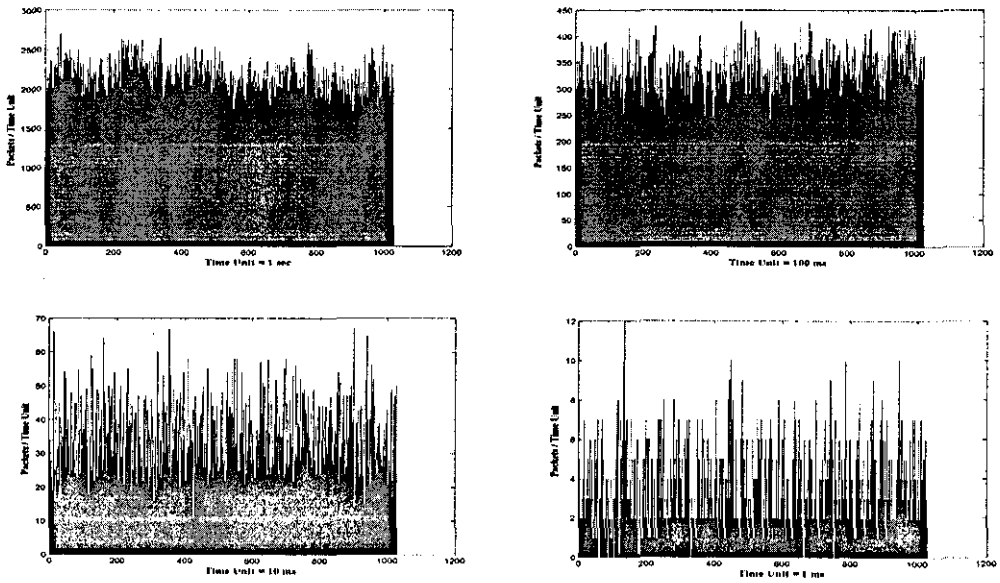

Figure 5.6: Internet Packets Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000 ms (1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace:LU-HR-1.

We now present a description of these features through processing the time series in the frequency domain in which we assume that the power spectrum of this signal is dominated by a RSF model $P(\omega) = c/\omega^q$ where $c > 0$. Let $X(t)$, in the time domain, be a time series of Internet (bytes or packets) traffic which is assumed to be a self-affine signal. The power spectrum of such a signal can be written as $P(\omega) = |X(\omega)|^2$, where $X(\omega)$ is Fast Fourier Transform (FFT) of the time series in the frequency domain (i.e. $X(\omega) = fft(X(t))$). For such a time series the power spectrum obeys the RSF model

$$P(\omega) = \frac{c}{\omega^q}$$

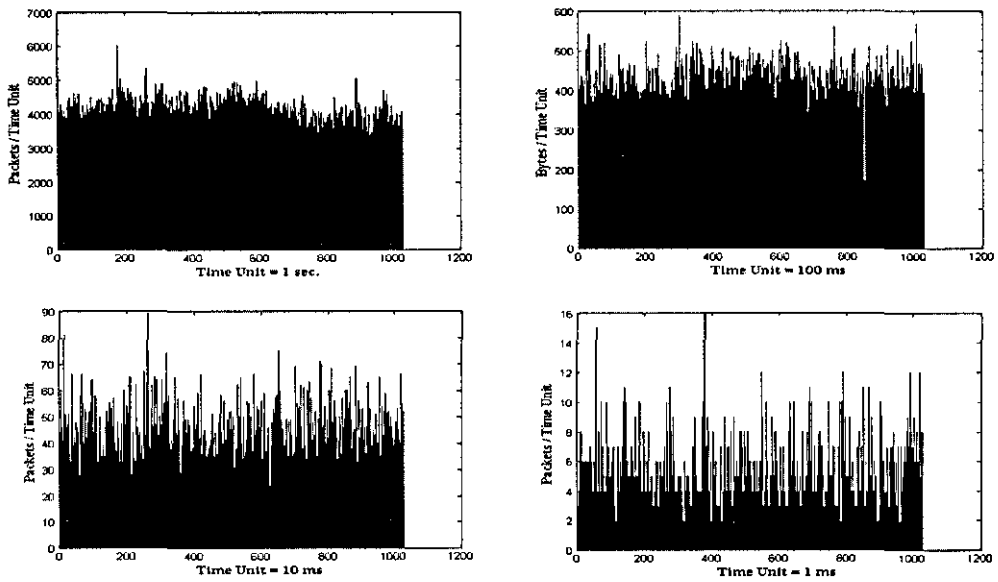

Figure 5.7: Internet Packets Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000 ms (1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace:LU-HR-2.

Figure 5.10 and Figure 5.11 show example of different plots of the measured power spectrum of Internet (bytes and packets) traffic over four different time units. Figure 5.12 shows the plot of measured power spectrum of packets inter-arrival times. These plots give the evidence that the power spectrum of the time series signal of internet traffic obeys the RSF model.

We can characterize the behavior of Internet traffic through estimating the parameter q in the proposed model where the estimated values of this parameter reflect the degree of self-similarity (fractality) in the internet traffic.

To do this we apply the Least Square Method on the measurements of Internet traffic which is described in the following Section. Note that the Wiener-Kintchine theorem expresses the relation between the Fourier transform of the

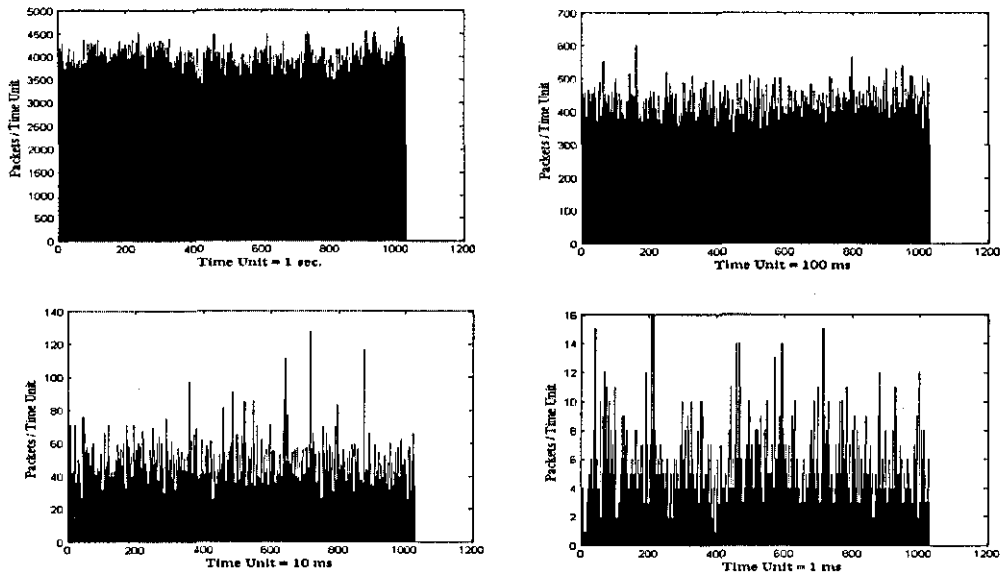

Figure 5.8: Internet Packets Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000 ms (1 sec.), Upper Right: 100 ms, Lower Left: 10 ms, and Lower Right: 1 ms aggregations, Trace:LU-HR-3.

auto-correlation function and the power spectrum density function (PSDF) of a time series as

$$R(k) \leftrightarrow P(\omega)$$

where $R(k)$ is the auto-correlation function of the time series and

$$P(\omega) = \int_{-\infty}^{\infty} R(k) e^{-ik\omega} dk$$

5.9.2 Power Spectrum Method

Let $X_i, \quad i = 1, 2, 3, ..., N$ (N being a power of 2) be an aggregated time series that represent the number of bits, bytes or packets that occur in a predefined

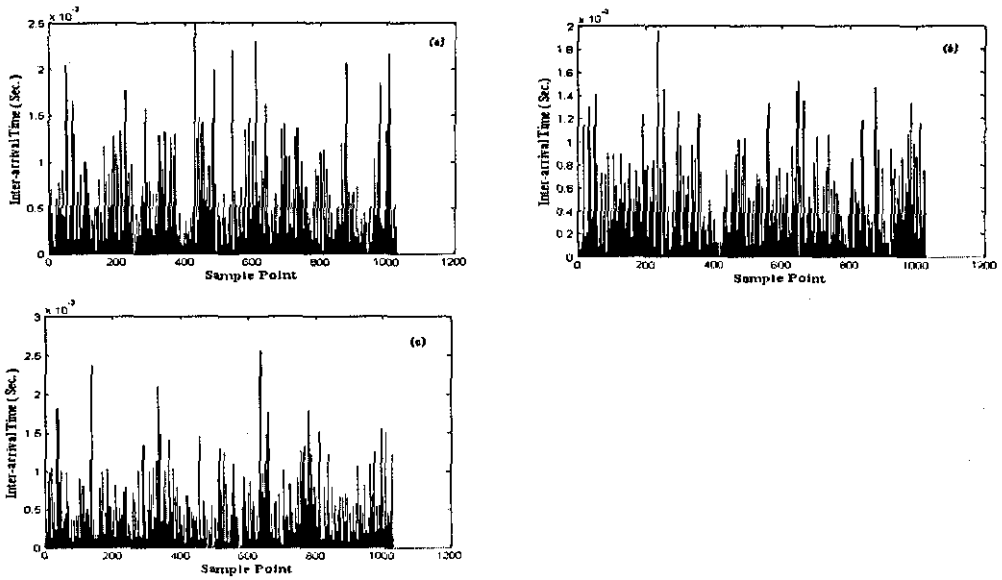

Figure 5.9: Inter-arrival Packets Times Bursts of Traces: (a)LU-HR-1, (b)LU-HR-2 and (c)LU-HR-3.

unit of time, say, 1 sec, 100 msec, 10 msec, or 1 msec. Consider the case in which the digital power spectrum $P_i \equiv P(\omega_i)$ is given by applying a FFT to this time series. This data can be approximated by

$$F(\omega_i) = \frac{c}{\mid \omega_i \mid^{\frac{q}{2}}}$$

or

$$\mid F(\omega_i) \mid^2 = \hat{P}(\omega_i) = \frac{c}{\mid \omega_i \mid^q}$$

and by using the least squares method we can estimate q and c as follows:

$$E(q,c) = \sum_{i=1}^{N} [\ln P(\omega_i) - \ln \hat{P}(\omega_i)]^2$$

$$= \sum_{i=1}^{N} [\ln P(\omega_i) - C + q \ln \omega_i]^2$$

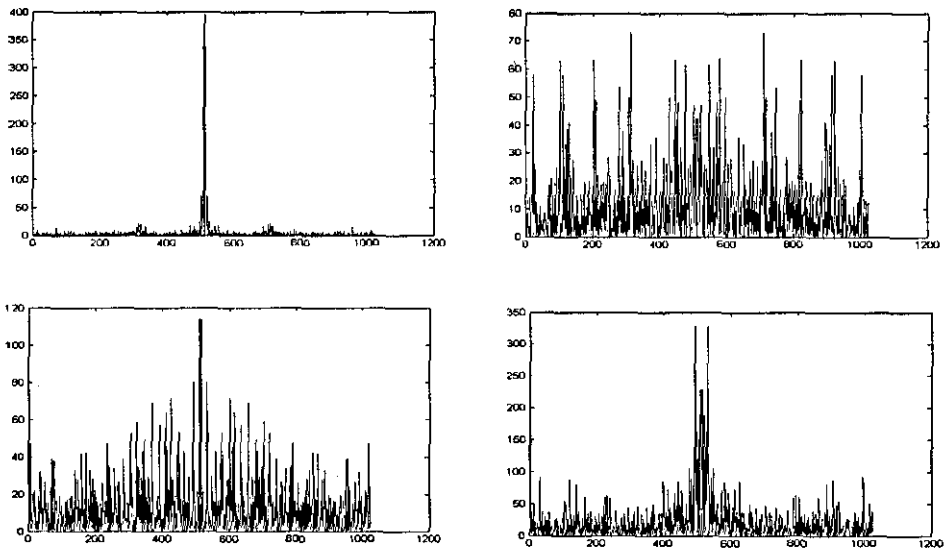

Figure 5.10: Measured Power Spectrum of Bytes Traffic: from top to bottom and from left to right: 1sec (q=0.45), 100ms (q=0.28), 10ms (q=0.10) and 1ms (q=0.15).

where $C = \ln c$, and it is assumed that $\omega_i > 0$ and $P(\omega_i) > 0$, $\forall i$

Differentiating E with respect to q and C gives

$$\frac{\partial E}{\partial q} = 2\sum_{i=1}^{N}[\ln P(\omega_i) - C + q \ln \omega_i] \ln \omega_i$$

$$\frac{\partial E}{\partial C} = -2\sum_{i=1}^{N}[\ln P(k_i) - C + q \ln \omega_i].$$

Solving

$$\frac{\partial E}{\partial q} = 0$$

and

$$\frac{\partial E}{\partial C} = 0$$

we obtain the following formulas for q and C respectively:

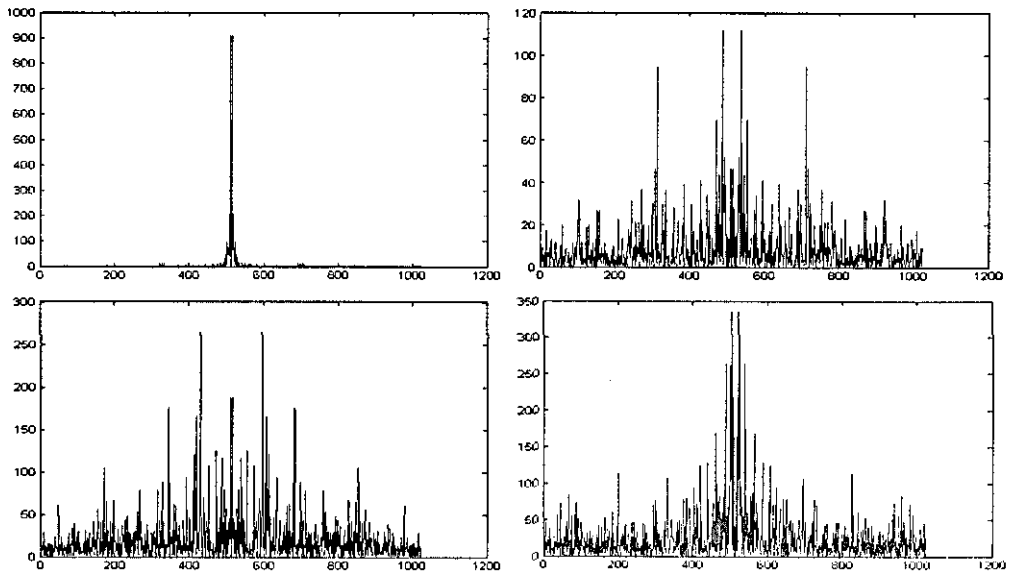

Figure 5.11: Measured Power Spectrum of Packets Traffic: from top to bottom and from left to right: 1sec (q=0.76), 100ms (q=0.28), 10ms (q=0.23) and 1ms (q=0.44).

$$q = \frac{N\sum_{i=1}^{N}(\ln\omega_i)\ln P(\omega_i) - (\sum_{i=1}^{N}\ln\omega_i)(\sum_{i=1}^{N}\ln P(\omega_i)}{(\sum_{i=1}^{N}\ln\omega_i)^2 - N\sum_{i=1}^{N}(\ln\omega_i)^2}$$

$$C = \frac{1}{N}\sum_{i=1}^{N}\ln P(\omega_i) + \frac{q}{N}\sum_{i=1}^{N}\ln\omega_i$$

where $c = \exp C$. Here, $P(\omega_i)$ is the measured power spectrum of the signal and ω_i its spatial frequency. Since the power spectrum of real signals of size N is symmetric about the DC level, where the DC level is taken to the mid point $\frac{N}{2}+1$ of the array, so in practice we consider only the $\frac{N}{2}$ data that lie to the right of DC [10].

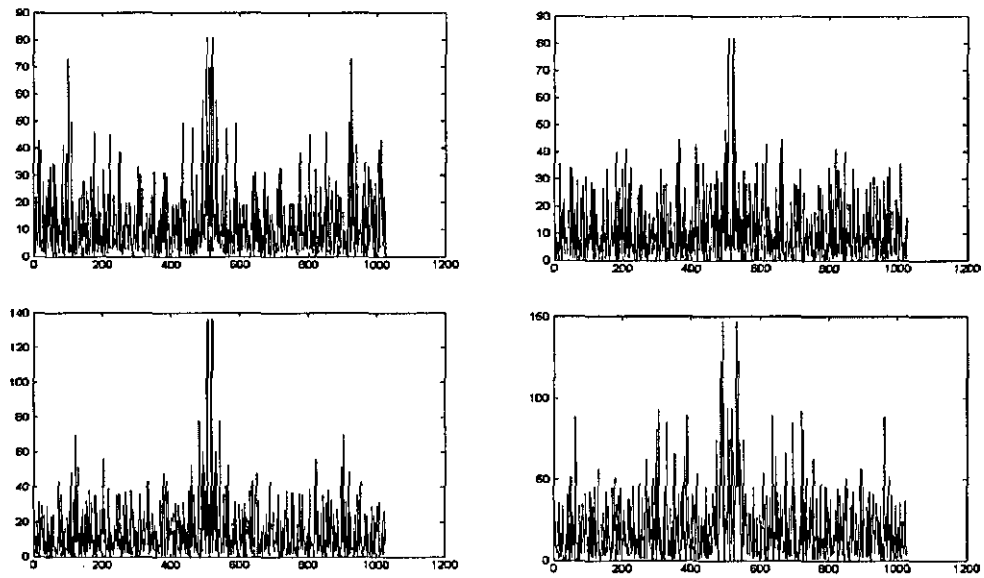

Figure 5.12: Measured Power Spectrum of Packets Inter-arrival times, from top to bottom and from left to right:(q=0.08),(q=0.15),(q=0.19) and (q=0.11).

5.10 Fractal Parameters Estimation for Internet Traffic Data

The two main properties of network packets are their *sizes* in bytes and their $inter-arrival$ times; both are taken to obey a Random Scaling Fractal model. To estimate the fractal parameter in these series we convert them to the frequency domain in which we assume that the empirical power spectrum of each series has an envelope of Power Spectrum Density Function (PSDF) given by $P(\omega) = |\omega|^{-q}$. In the following sections the results of estimation of the parameter for the proposed fractal model are given.

5.10.1 Parameter Estimation for Packet Size Time Series

Suppose that $\{X\}$ be a time series of the number of bytes gathered, corresponding to four different values of a time unit $1ms, 10ms, 100ms$ and $1s$. These time series are taken from the traces of the Internet traffic described in Section 5.6. By using a moving window technique, we choose a window of size $N = 1024$ to move over the points of the time series according to the given time unit. From each window segment we apply the power spectrum estimation method to estimate the Fourier dimension q after transforming to the spectral domain of a given segment.

The following algorithm summarizes the steps of the estimation process (using m-code syntax):

Step 1: Use a window of size $N = 1024$ over the points of a given time series to extract a segment array, say $X_i, \qquad i = 1, 2, 3, ..., N$

Step 2: Normalize the segment: $Y_i = X_i/max(X_i)$.

Step 3: Compute the Discrete Fourier Transform (DFT) of Y_i using a Fast Fourier Transform (FFT) and (with spectral shifting) to yield $Z_i = fftshift(fft(Y_i))$.

Step 4: Compute the empirical power spectrum of Z_i, i.e. $P = |Z_i|^2$.

Step 5: Extract the right halve of the power spectrum (excluding the DC term at N/2+1).

Step 6: Compute the parameter q using the computational formula of the PSM given above.

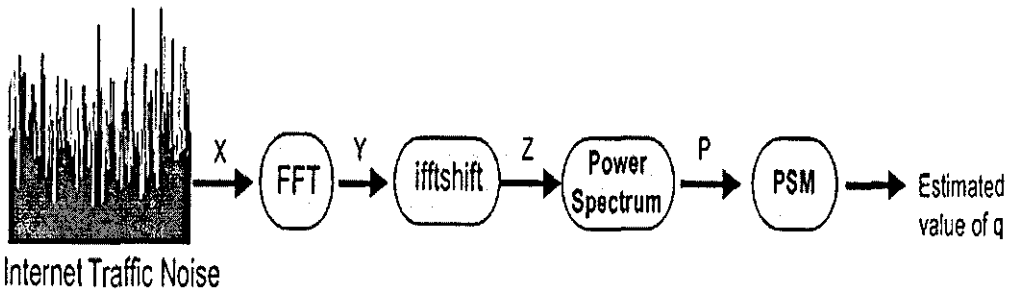

Figure 5.13: Block diagram for estimating the Fourier dimension, q, of Internet Traffic.

Step 7: Iterate Step 1 through to Step 6 until the end of the time series.

Figure 5.13 shows a block diagram of the estimation method. Table 5.2 through to Table 5.5 gives the result of estimating the Fourier dimension q according to the different time scales from the traffic of bytes and packets through for the traces described in Table 5.1.

Wind. No.	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
1	0.07	0.29	0.20	0.25	0.25	0.31
2	0.10	0.24	0.19	0.15	0.21	0.31
3	0.10	0.31	0.36	0.35	0.32	0.26
4	0.11	0.31	0.19	0.15	0.31	0.40
5	0.12	0.27	0.13	0.23	0.36	0.34
6	0.10	0.35	0.16	0.24	0.17	0.42
7	0.04	0.20	0.18	0.29	0.15	0.52
8	0.07	0.29	0.42	0.40	0.30	0.44
9	0.17	0.30	0.14	0.27	0.40	0.43
10	0.16	0.36	0.29	0.24	0.14	0.24

Table 5.2: Estimated values of q for time scale *1 ms*.

Wind. No.	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
1	0.02	0.31	0.19	0.34	0.11	0.32
2	0.17	0.31	0.12	0.26	0.16	0.29
3	0.10	0.26	0.20	0.33	0.18	0.26
4	0.11	0.40	0.14	0.32	0.09	0.23
5	0.07	0.34	0.06	0.22	0.17	0.33
6	0.008	0.42	0.16	0.18	0.04	0.15
7	0.03	0.52	0.23	0.29	0.07	0.22
8	0.13	0.44	0.20	0.32	0.15	0.15
9	0.04	0.43	0.38	0.58	0.04	0.24
10	0.08	0.24	0.18	0.39	0.13	0.22

Table 5.3: Estimated values of q for time scale *10 ms*.

In each table, the first column is the window number whereas the other three columns are of the estimated values of the Fourier dimension q for the three traces. Under each of the trace column there are two sub-columns; one to estimate the parameter from the bytes traffic and one to estimate the parameter from the packets traffic. Note from these tables that as the time resolution increases then the estimated values of the parameter q decreases; whereas the variation among the estimated values increases as the time resolution increases, i.e. as the utilization of the network increases the estimated value of q, either from bytes or packets traffic, also increases. This is to be expected given the fractal model Internet traffic noise being considered. In other words, the theoretically expected results are consistent with those obtained experimentally.

Wind. No.	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
1	0.25	0.44	0.32	0.45	0.20	0.28
2	0.26	0.35	0.24	0.49	0.08	0.16
3	0.18	0.31	0.36	0.40	0.13	0.23
4	0.30	0.48	0.17	0.25	0.13	0.26
5	0.29	0.51	0.31	0.35	0.36	0.38
6	0.35	0.52	0.17	0.27	0.24	0.28
7	0.34	0.44	0.22	0.31	0.28	0.29
8	0.39	0.51	0.30	0.29	0.31	0.34
9	0.41	0.53	0.26	0.40	0.10	0.26
10	0.19	0.34	0.27	0.33	0.20	0.30

Table 5.4: Estimated values of q for time scale *100 ms*.

Wind. No.	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
1	0.77	0.70	0.54	0.64	0.54	0.49
2	0.58	0.56	0.58	0.80	0.43	0.51
3	0.74	0.86	0.56	0.63	0.61	0.76

Table 5.5: Estimated values of q for time scale *1000 ms*.

Statistics	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
Min.	0.04	0.20	0.13	0.15	0.14	0.24
Max.	0.17	0.36	0.42	0.40	0.40	0.52
Mean	0.10	0.29	0.23	0.26	0.26	0.37
Std.	0.04	0.05	0.10	0.08	0.09	0.09

Table 5.6: Some statistics on the estimated values of q for time scale *1 ms*.

Statistics	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
Min.	0.08	0.24	0.06	0.18	0.04	0.15
Max.	0.17	0.52	0.38	0.58	0.18	0.33
Mean	0.08	0.37	0.19	0.37	0.11	0.24
Std.	0.05	0.09	0.08	0.11	0.05	0.06

Table 5.7: Some statistics on the estimated values of q for time scale *10 ms.*

Statistics	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
Min.	0.18	0.17	0.17	0.25	0.08	0.16
Max.	0.41	0.53	0.36	0.49	0.36	0.38
Mean	0.30	0.44	0.26	0.35	0.20	0.28
Std.	0.08	0.08	0.06	0.08	0.09	0.06

Table 5.8: Some statistics on the estimated values of q for time scale *100 ms.*

Tables 5.6 to 5.9 give statistical information (the Max., Min, Mean and Standard Deviation (Std.)) of the estimated values of q that are given in Tables 5.2 to 5.5. These values suggest that the Internet traffic noise behaves in a non-stationary way and that the traffic is characterized by more than one parameter value leading to the hypothesis of that Internet traffic is a multi-fractal phenomenon. In other words, the traffic has more than one value of the Fourier dimension. Table 5.10 presents a summary of some statistics of the estimated values of q for all traces together at different time scales.

Statistics	LU-HR-1 (L)		LU-HR-2 (M)		LU-HR-3 (H)	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet
Min.	0.58	0.56	0.54	0.63	0.43	0.49
Max.	0.77	0.86	0.58	0.80	0.61	0.76
Mean	0.70	0.71	0.56	0.69	0.53	0.59
Std.	0.10	0.15	0.02	0.09	0.09	0.15

Table 5.9: Some statistics on the estimated values of q for time scale *1000 ms.*

Statistics	1 ms		10 ms		100 ms		1000 ms	
	Byte	Packet	**Byte**	Packet	**Byte**	Packet	**Byte**	Packet
Min.	0.04	0.15	0.04	0.006	0.08	0.16	0.43	0.49
Max.	0.42	0.52	0.38	0.58	0.41	0.53	0.77	0.86
Mean	0.20	0.31	0.12	0.31	0.20	0.36	0.59	0.59
Std.	0.10	0.09	0.08	0.10	0.09	0.10	0.10	0.15

Table 5.10: Summary statistics for all traces over different time scales.

5.10.2 Parameter Estimation for Packet Inter-arrival Times

The inter-arrival times are that sequence which consists of the timestamp differences between consecutive packets. To characterize the behavior of such sequences we estimate the Fourier dimension q in the RSF model by using the same steps as those described in the previous section.

Figure 5.9 shows the plot of the first 1024 sample points of the packets inter-arrival time sequence of the three traces LU-HR-1, LU-HR-2 and LU-HR-3. The horizontal axis is the sample points whereas the vertical axis represents the inter-arrival times in seconds. From the Figure 5.9 we note that the inter-arrival time of packets has a fractal like behavior, i.e. the series of these times can be considered as a Random Scaling Fractal series. To characterize the series of inter-arrival times of packets we apply the same approach for estimating the Fourier dimension.

Wind. No.	LU-HR-1	LU-HR-2	LU-HR-3
1	0.14	0.09	0.34
2	0.26	0.08	0.08
3	0.10	0.17	0.21
4	0.11	0.03	0.24
5	0.17	0.06	0.10
6	0.14	0.12	0.16
7	0.12	0.09	0.20
8	0.10	0.16	0.12
9	0.14	0.09	0.10
10	0.26	0.15	0.14

Table 5.11: Estimated values of q for packet Inter-arrival times.

Table 5.11 shows the estimated values of the parameter q from sequences of the packet inter-arrival times. The first column is the window number and the other columns are of the estimated values of q correspond to the LU-HR-1, LU-HR-2, and LU-HR-3 traces. Table 5.12 gives some statistics on the estimated values of q for the inter-arrival time sequences that are given in Table 5.11. The mean of the estimated values of q takes on large values when there is high utilization (LU-HR-3 trace) as compared with the mean when there is low to medium utilization (LU-HR-1 or LU-HR-2 traces).

Statistic.	LU-HR-1	LU-HR-2	LU-HR-3
Min.	0.10	0.03	0.08
Max.	0..26	0.17	0.34
Mean	0.15	0.10	0.17
Std.	0.06	0.05	0.08

Table 5.12: Some statistics on the the estimated values of q for packet Inter-arrival times.

5.11 References

[1]Douglas E. Comer, *Computer Networks and Internet*, Prentice Hall, 4th edition, 2004.

[2]Leland W., Taqqu M., Willinger W., and Wilson D., "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. on Networking*, vol. 2, no. 1, pp. 1-15, 1994.

[3]Paxon V. and Floyd S., "Wide-area traffic: The failure of Poisson modelling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226-244, June 1995.

[4]Crovella M. E. and Bestavros A. , "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Transactions on networking*, vol. 5, pp. 835-846, Dec. 1997.

[5]Marke E. Crovella M. and Bestavros A., "Explaining World Wide Web traffic self-similarity," *Tech. Rep.*, Boston University, CS Dept., Boston, Oct. 1995.

[6]Beran J., *Statistics for Long-Memory Processes*, Chapman and Hall, New York, 1994.

[7] Keresteci E., Sema F., Ersoy C. and Ufuk M., "Generation and evaluation of self-similar traffic in computer networks,"Dept. of Computer Engineering, Bogazii University, Istanbul, Turkey, 1996.

[8] Willinger W., Govindan R., Jamin V. and Shenker S., "Scaling phenomena in the Internet: Critically examining criticality," *PNAS*, vol. 99, pp. 2573-2580, Feb. 2002.

[9] Abrahansson H., "Traffic measurement and analysis," *SICS Tech. Rep.*, Kista, Sweden, Sept. 1999.

[10] Turner M., Blackledge J. and Andrew P., *Fractal Geometry in Digital Imaging*, Academic Press Ltd., UK, 1998.

[11] Willinger W. and Paxson V., "Where Mathematics meets the Internet," *Notices of the American Mathematical Society*, vol. 45, no. 8, pp. 961-970, Sept. 1998.

[12] Cipra B. A., "Oh what a tangled Web we have woven...," *SIAM News*, vol. 33, no. 2, 2001.

[13] Taqqu M.S., Willinger W., and Sherman R., "Proof of a fundamental result in self-similar traffic modelling," *Computer Communication Review*, vol. 27, pp. 5-23, 1997.

[14] Willinger W., Taqqu M. S., Leland W. E. and Wilson D. V., "Self-similarity in high speed packet traffic: Analysis and modelling of Ethernet traffic measurements," *Statistical Science*, vol. 10, no. 1, pp. 67-85, 1995.

[15] Park K., Kim G., and Crovella M., "On the effect of self-similarity on network performance," *Proc. of the SPIE International Conf. on Performance and Control of Network System*, pp. 296-310, Nov. 1997.

[16] Ost A. and Boudwijn R.H.,"Modelling and evaluation of pseudo self-similar traffic with Infinite-State Stochastic Petri Nets," *Proc. of the workshop on formal method telecom.*, pp. 120-136, Sept. 1999.

[17] Peha J. M., "Protocols can make traffic appear self-similar," *Proc. of the 1997 IEEE/ACM/SCS Comm. Networks and Distributed System. Modelling and Simulation Conf.*, pp. 47-52, Jan. 1997.

[18] Veres A. and Boda M., "The chaotic nature of TCP congestion control," *IEEE INFOCOM 2000*, pp. 1715-1723, Apr. 2000.

[19] Gilbert A. C., Fledman A. and Willinger W., "Data networks: Investigating the multi-fractal nature of Internet WAN traffic," *Proc. of the ACM/SIGCOMM'98*, Canada, Sept. 1998.

[20] The internet traffic archive, Personal communication with Network manager at Computing Services Dept./Loughborough Univ., April 2006.

[21] Williamson C., "Internet traffic measurement," *IEEE Internet Computing*, vol. 5, no. 6, pp. 70-74, Nov. 2001.

[22] Cleveland W. and Sun D., "Internet traffic data," *Journal of the American Statistical Association*, vol. 95, pp. 979-985, 2000.

[23] Mian S., "Traffic modelling: the advent of fractals," *Communications Engineer*, vol. 1, no. 1, pp. 32-35, Feb. 2003.

[25] Jacques Levy Vehel, "Fractal and multi-fractal Traffic," *Project Fractals, INRIA Rocquencourt*, Le Chesnay Cedex, France, 2002.

[26] Taqqu M., Teverovsky V., and Willinger W., "Is network traffic self-similar or multi-fractal?" *Fractals*, no. 5, pp. 6373, 1997.

[27] Molnar S. and Terdik G., "A General fractal model of Internet traffic," *In Proc. of IEEE LCN 2001*, Tampa, Florida, Nov. 2001.

[28] Lucas M., Wrege D. , Dempsey, Bert J. and Weaver A., "Statistical characterization of Wide-Area IP traffic," *Proceedings of Sixth International Conference on Computer Communications and Networks (IC3N'97)*, pp. 442-447, 1997.

[29] Chakraborty D., Ashir A., Suganuma G., Mansfield K., Roy T. and Shiratori N., "Self-similar and fractal nature of Internet traffic," *Int. J. Network Mgmt*, no. 14, pp. 119-129, 2004.

[30] Mian S., Ghassempoory M. and Bentall M., "Mathematical analysis of network traffic," *In Proc. of Student Conference on Research and Development*, Shah Alam, Malaysia, 2002.

[31] Cao J., Cleveland W., Lin D., and Sun D., "On the non-stationarity of Internet traffic," *Proc. of ACM SIGMETRICS '01*, pp. 102112, 2001.

[32] Chong K. and Choo Y., "Fractal analysis in Internet traffic time series," arXiv:physics/0206012, vol. 3, Jun 2003.

[33] Fischer M., and Fowler T., "Fractal, Heavy-Tails and the Internet," *Sigma Technology Summaries: Mitretek Systems*, no. 2, pp. 11-16., May 2003.

[34] Wisitpongphan N. and Peha J., "Effect of TCP on self-similarity of network traffic," *Proc. of the 12th International Conference on Computer Communications and Networks, ICCCN 2003*, pp. 370- 373, Oct. 2003.

[35] Fowler T., "A Short tutorial on fractals and Internet traffic," *The Telecommunication Review 10*, Mitretek Systems, McLean, VA, pp.114, 1999.

[36] Paxon V. and Floyd S., "Wide area traffic: the failure of Poisson modelling," *In Proc. ACM SIGCOMM'94*, London, UK, pp. 45-56, Sept. 1994.

Chapter 6

Covert Transfer of Data Through the Internet

6.1 Introduction

In this chapter we introduce a novel method of using the fractal characteristics of internet traffic to disguise or camouflage the transfer of a digital file through the Internet. For applications to securing data transmission through the Internet, instead of attaching a complete file (encrypted or otherwise) at a given time, we first encrypt the file and then split a binary representation of the file into a number of blocks. Each block is then saved as a sequence (a trace) of binary files in which the statistical characteristics of such traces fit with the characteristics of the actual trace of Internet packets. In principle, for the purpose of encrypting the plaintext any encryption product can be used;

for example, the Crypstic^{TM}system which is provided in the CD that accompanies this thesis and was used for this work.

The sizes of the split files are determined by the fractal characteristics of the Internet through which the data is transmitted using the Random Scaling Fractal model. At the same time, a sequence of inter-submission times are generated using the same model. This sequence is used to formulate the required timestamps. The data is then applied to the Internet as e-mail attachments and submitted according to the sequence of timestamps computed. The recipient of the data recovers the information by concatenating the file sequence and decrypting the result.

In terms of known published materials, this approach is the first of its kind to use the self-affine nature of Internet traffic in order to disguise the transmission of a digital file by splitting the file into a number of blocks (files) whose size and submission times are compatible with the bursty fractal lengths of Internet traffic.

6.2 Simulation of Internet Traffic Data

In this section we focus on simulating the two variables of Internet traffic data, namely, the sizes of packets and the corresponding timestamps.

6.2.1 Synthetic Generation of a Fractal Traffic Trace

The method of generating a synthetic traffic trace for the purpose of securely transferring a set of files as e-mail attachments is considered.

Generally, this equates to the problem of letting the behavior of e-mail arrivals match the behavior of packets observed in actual computer networks. In what follows, we introduce the steps associated with the method to generate a synthetic trace or sample path which displays the same statistical properties as the actual data traffic. In this way, we generate synthetic data sequences that exhibit similar features to the measured traffic. The points of the sample path represent the sizes in bits of the split files.

In general, this method of generating a synthetic trace with fractal characteristics depends on generating white Gaussian noise that is filtered using the Random Scaling Fractal model with a suitable value of the parameter q. To ensure that the synthetic trace is representative of a trace that is likely to be encountered in the 'real world', the synthetic trace is used the estimated values of q from the captured traces of actual Internet traffic.

The inputs to the method are q, the desired Fourier dimension, and N, the desired number of points in the synthesized sample, (where N is a power of 2). Here, the parameter q takes a value from the table of estimated values (see, Table 5.2 to Table 5.5) or it may take the average of these values (see, Table 5.6 to Table 5.9). In what follows, we give a description of the principal steps (quasi m-code based):

Step 1: Generate a stream of white Gaussian noise, sayX_i, where $i = 1, 2, 3, ..., N$

Step 2: Compute the Discrete Fourier Transform (DFT) of X_i using a Fast Fourier Transformation (FFT) to give a new series (in the complex domain), namely, $Y_i = fft(X_i)$.

Step 3: Choose a value of the fractal parameter q.

Step 4: Apply the fractal filter on Y_i using the RSF model to obtain $Z_i = Y_i * 1/k_i^q$.

Step 5: Compute the inverse DFT of Z_i using an FFT to give W_i.

Step 6: Consider the real part of W_i giving fractal noise stream, say s_i, where $s_i = real[W_i]$.

Step 7: Compute the Hilbert transform of the fractal signal, $H_i = hilbert(s_i)$.

Step 8: Compute the modulus of H_i and normalize the result, $h_i = \frac{|H_i|}{\max(H_i)}$.

Step 9: Rescaling the sequence h_i, by multiplying each element of h_i by a scaler C ($R_i = C.h_i$), where C is a power of 2, and then round down the elements to the nearest integer. The integer values of the sequence R_i, which lie in the interval $(0, C]$, constitute the *sizes* series of the of the actual files (packets) sizes as described above.

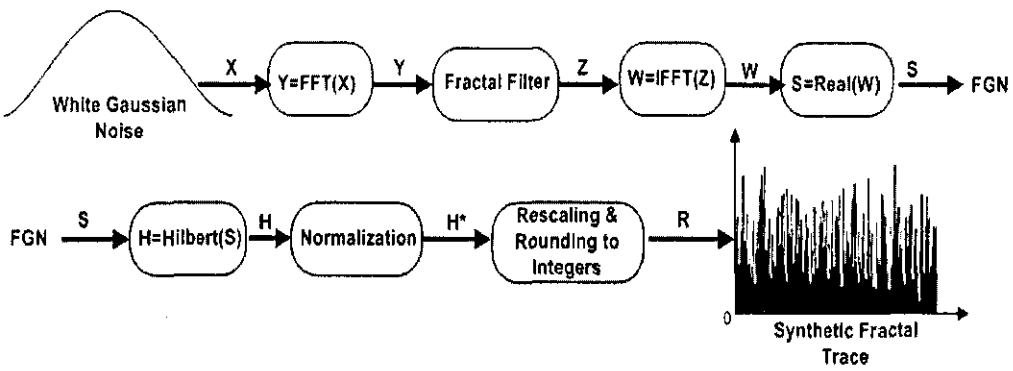

Figure 6.1: Block Diagram for the Synthesis of a Fractal Trace.

Figure 6.1 shows a block digram of the method of generating a synthetic fractal trace. Figure 6.2 shows four time series plots of synthetic sample paths.

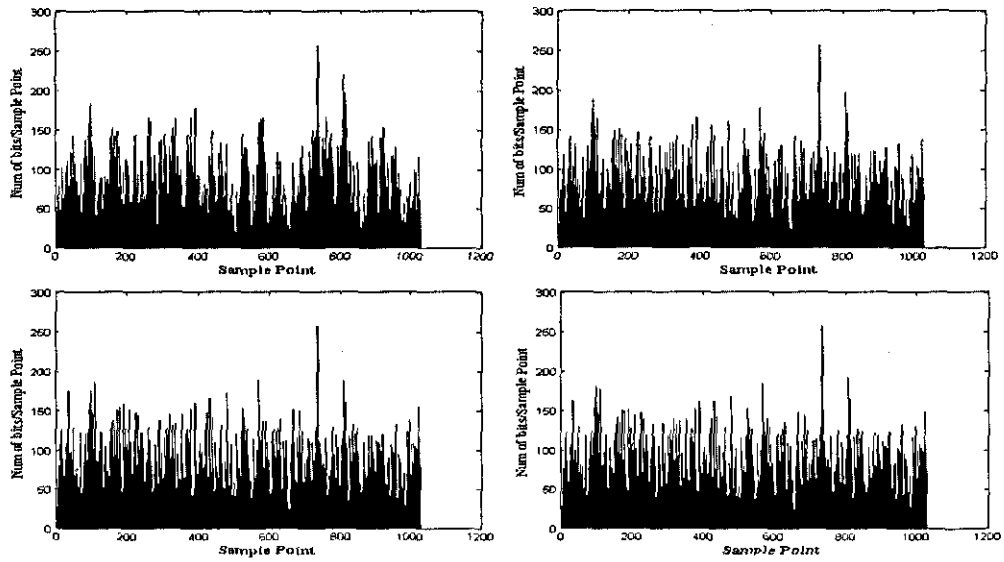

Figure 6.2: Synthetic Internet Bits Traffic Bursts over Four Orders of Magnitude: 1 sec, q=0.6 (top-left); 100 ms, q=0.3 (top-right); 10 ms, q=0.13 (bottom left) and 1 ms, q=0.2 (bottom-right).

Each path is of length 1024. The horizontal axis represents the sample number; the vertical axis represents traffic load in bytes per time unit. The plots are produced according to the algorithm given above and with four different estimated values of the Fourier dimension $q = 0.59, 0.2, 0.12$, and 0.2. The estimated values of q considered are the values of means over the three traces at different time scales given in Table 5.10.

From this figure we see clearly that the plots of the generated series using the algorithm described above appear to be somewhat like to the plots of the actual series of the Internet traffic (e.g. see Figure 5.3 to Figure 5.5).

6.2.2 Synthetic Generation of the Timestamps Sequence

A timestamps sequence, say $T_1, T_2, T_3, ..., T_N$ is required to send each file in the trace of files at time intervals compatible with the fractal characteristic of Internet noise, i.e. the fractal time signature. The principal steps to generate such sequence are given below:

Step 1: Generate a series of white Gaussian noise, sayX_i, where $i = 1, 2, 3, ..., N$.

Step 2: Compute the Discrete Fourier Transform (DFT) of X_i using a Fast Fourier Transformation (FFT) yields a new series in complex domain, namely, $Y_i = fft(X_i)$.

Step 3: Choose a value of the fractal parameter q from the estimated values of the real timestamp sequences that given in Table (5.11).

Step 4: Apply the fractal filter on Y_i using the RSF model to obtain $Z_i = Y_i * 1/k_i^q$.

Step 5: Consider the real part of the inverse DFT of Z_i using an FFT to give a series of fractional Gaussian noise, $U_i = real[ifft(Z_i)]$.

Step 6: Compute the Hilbert transform of the fractal series, $H_i = hilbert(U_i)$.

Step 7: Compute the modulus of H_i and normalize the result, $h_i = \frac{|H_i|}{\max(H_i)}$.

Step 8: Rescale the sequence h_i by multiplying each h_i by a scaler C to give $I_i = C.h_i$ where C is a power of 2 and then round the elements to the nearest integers toward zero.

Step 9: Write the sequence of timestamps in 'military form', i.e, hh:mm:ss (hours, minutes and seconds respectively).

To do this, we first consider an initial and then add the inter-submission values obtained in Step 8 above to the initial time in an accumulative way until the last timestamp is reached.

Inter-submission Time (Second)	Timestamp
53	22:11:38
8	22:11:46
7	22:11:53
7	22:12:00
4	22:12:04
43	22:12:47
28	22:13:15
21	22:13:36
12	22:13:48
5	22:13:53
.	.
.	.
.	.

Table 6.1: The first 10 points of synthetic Inter-submission times with the corresponding Timestamps.

Table 6.1 shows the first 10 points of a sequence of timestamps that is obtained by applying the above algorithm according to the RSF model with $q = 0.10$ and the initial time is '22:10:45'. Figure 6.3 shows the plot of 1024 sample points of the sequence of inter-submission with $q = 0.10$. Note that the plot of the inter-submission time sequence is similar to the plot of the actual packet inter-arrival times sequence (see Figure 5.9) where the estimated value of q from this sequence is 0.10.

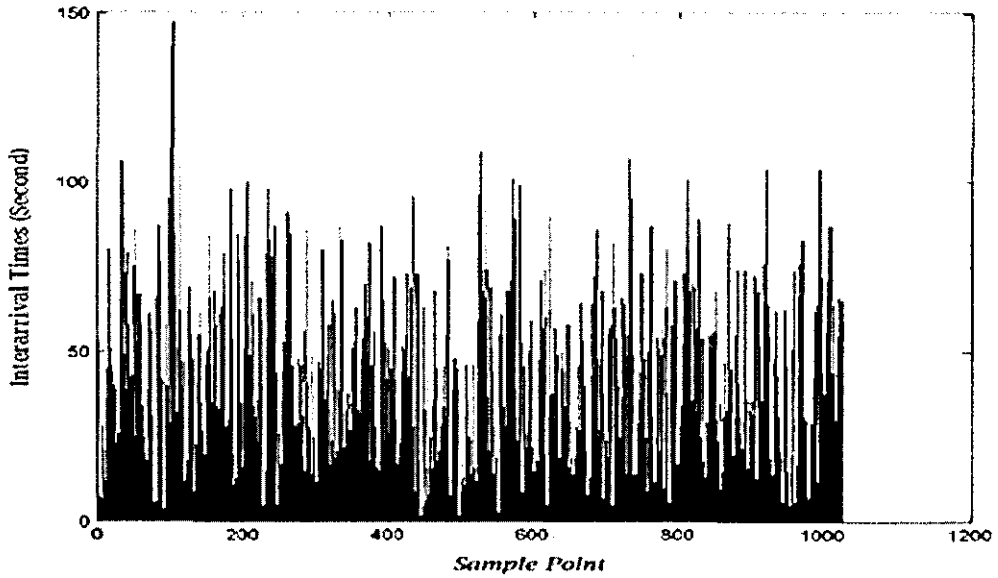

Figure 6.3: Plot of the synthetic sequence of inter-submission times for $q = 0.10$

6.3 Transmission of Files Between a Sender-Receiver Pair on a Network

In recent years the Internet has become the most popular media for information transfer in the world. Terms like 'e-mail' and 'e-commerce' are well known and accessible to everyone. Here, the resulting series R_i where $i = 1, 2, 3, ..., N$ represents the synthetic fractal trace, and N is the length of this trace.

Each element in the generated trace is considered to be the size of each file of bits, and the length of such trace is the number of the split files. We describe the mechanism of sending and receiving the files below.

In light of the above, if we consider the set of split files to be $file_1, file_2, file_3, ..., file_N$ then $size(file_1) = R_1$, $size(file_2) = R_2$, ..., $size(file_N) = R_N$. These files are then sent over the Internet according to the sequence of timestamps, $T_1, T_2, T_3, ..., T_N$.

Sending Files

For the application of securing data transmission through the Internet, instead of attaching a complete file (encrypted or otherwise) at a given time, we first encrypt the file and then split a binary representation of the file into a number of blocks.

Each block is then saved as a sequence of binary files whose size is determined by the fractal characteristics of the Internet through which the data is to transmitted using the Random Scaling Fractal model.

At the same time, a sequence of inter-submission times are generated using the same model. The data is then applied to the Internet as e-mail attachments and submitted according to the sequence of inter-submission times computed.

Here, it is assumed that the inter-submission times are compatible with the inter-arrival times in that they adhere to the same fractal model that is assumed to be stationary over the given period of interest.

To make the traffic associated with a transmitted file sequence (packets) compatible with the fractal nature of internet traffic, we send an e-mail with one attached file only, say file $file_i$, at a timestamp T_i, where $i = 1, 2, 3, ..., N$. Figure 6.4 shows the block diagram of the mechanism for sending a digital file over the Internet by splitting its binary representation into a trace of digital files.

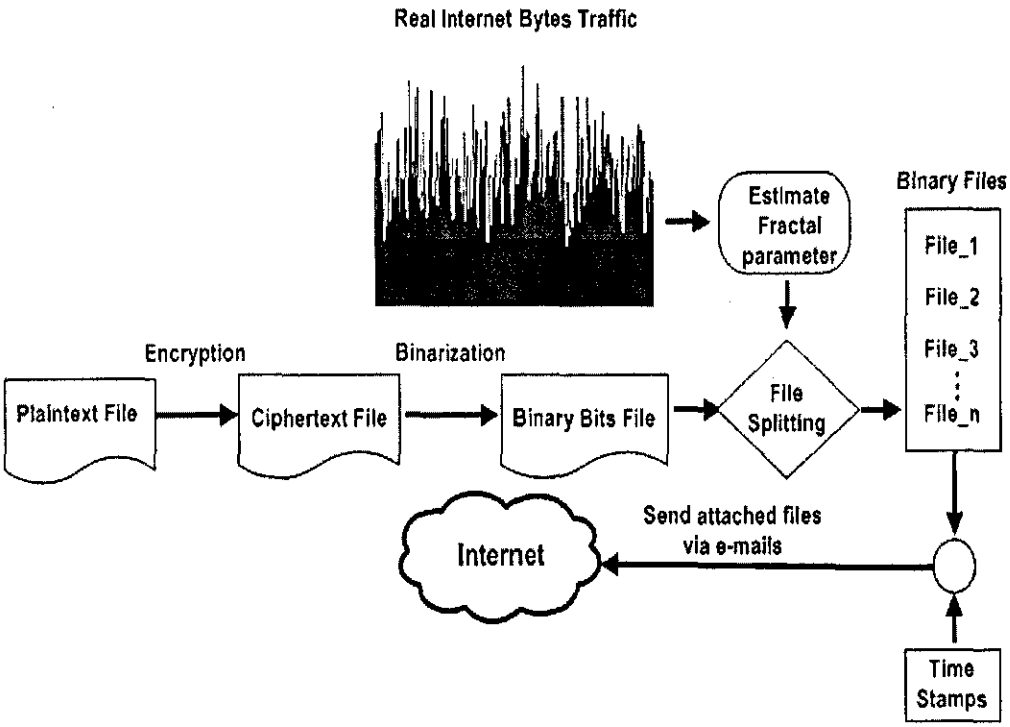

Figure 6.4: Block Diagram for the mechanism for sending a digital file over the Internet.

Receiving Files

Once all the attached files have arrived at the recipients location, they are recombined into their original file. For the purpose of increasing the security of the communication, we first encrypt the whole file before binarization. Upon reception and after concatenation of the binary contents of the received files, we decrypt the resultant to recover the plaintext file.

The operation of encryption and decryption can be based on any algorithm but here, we use the Crypstic^{TM} system.

6.4 References

[1]Shu-Gang Liu, Pei-Jin Wang and Lin-Jie Qu, "Modelling and simulation of self-similar data traffic," *Proc. of 2005 International Conference on Machine Learning and Cybernetics*, vol. 7, pp. 3921- 3925, Aug. 2005.

[2] Paxson V. and Floyd S., "Why we don't know how to simulate the Internet," *Proc. of the winter simulation conference*, 1997.

[3] Bo R. and Lowen S., "Fractal traffic models for Internet simulation," *Proc. of fifth IEEE Symposium (ISCC 2000)*, pp. 200 - 206, July 2000.

[4] Paxson V., "Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic, " *ACM SIGCOMM, Computer Communication Review*, vol. 27, no. 5, pp. 518, Oct. 1997.

[5] Jeong H., McNickle D., Pawlikowski K., "A Comparative study of generators of synthetic self-similar Teletraffic," *HSNMC 2003*, LNCS 2720, pp. 571-580, 2003.

[6] Guo L., Crovella M. and Matta I., "How does TCP generate pseudo-self-similarity?" *Proc. of the International Workshop on Modelling, Analysis and Simulation of Computer and Telecommunications Systems* , Cincinnati, OH, Aug. 2001.

[7] Paxson V., "Fast approximation of self-similar network traffic," *Lawrence Berkeley National Lab.*, Tech. Rep., LBL-36750, Berkeley, USA, Apr. 1995.

[8] Sergio L. and Derong L., "A Fast method for generating self-similar network traffic," *Comm. Technology Proc.*, vol. 1, pp. 54-61, Aug. 2000.

[9] Veitch D., Backar J., Wall J., Yates J., and Roughan M., "On-line generation of fractal and multi-fractal traffic," *In Proc. of the Workshop on Passive and Active Measurement (PAM 2000)*, 2000.

[10] Ledesma S. and Liu D., "Synthesis of fractional Gaussian noise using linear approximation for generating self-similar network traffic," *Computer Communication Review*, vol. 30, no. 3, pp. 4-17, 2000.

[11] David A. , "Improved fast, approximate synthesis of fractional Gaussian noise," *Proc. of the Hawaii International Conference on Statistics and Related Fields*, Honolulu, Hawaii, June. 2002.

Chapter 7

Software Development and Test Results

7.1 Introduction

In previous chapters we have considered the overall methodology developed for the work of this thesis. This chapter provides a 'sketch' of the basic modules and source code involved and has been designed to provide the reader with the essential features of the system designed for this thesis in terms of the theoretical and modeling issues discussed. The functions and modules have been prototyped in MATLAB and the m-code is discussed further in Appendix A which in turn relates to the complete m-code given in the CD at the back of this thesis.

The principal issue of this thesis is the design of a secure transmition system to transmit an encrypted or otherwise file covertly using Internet traffic noise

as a form of fractal camouflage. Such files could be text, image or audio files for example. To do this, two transmission systems have been considered.

The first system is based on transmission of digital data through wireless communications, whose background theory is given in Chapter 3 and Chapter 4. The second system considered is for transferring a digital file over the the Internet as an email attachment.

In this chapter, we highlight the structural operations of the two systems referring to the m-code that is sourced in its entirety on the CD.

7.2 Wireless Transmission System

In this system, to transmit a file of digital data, we use a way of encoding a binary form of the file by modulating parameter values of a proposed fractal model. To restore the data file the process of demodulating the received signals is applied. The modulation and demodulation process takes place in the following sequence:

Sender

Firstly, the sender encrypts the plaintext file using Crypstic^{TM}; then, the binary form of the ciphertext is encoded as a contiguous stream of fractal signals. The system developed for application of this process is based on the following user dependent operations:

- Select the fractal model to be used for encoding.
- Choose initial values for generating a white noise signal.

- Choose values of the model parameters.
- Choose values of the Cut Off Points (COPs).
- Encode the binary bit stream version of the ciphertext.
- Create a watermark of the COPs.
- Embed the watermark in the fractal modulated signal.
- Save the watermarked signal in a file to send later.

Recipient

- Run the Cut-Off points extraction module.
- Run the Demodulation module.
- Run the Decryption module (Crypstic™).

7.3 Encoding and Decoding of a bit stream using a fractal model

This module permits the user to choose either the Random Scaling Fractal (RSF) model or the Generalised Random Scaling Fractal (GRSF) model for encoding a bit stream and then to choose which parameter(s) is used for the modulation process. If the sender chose the RSF model then the encoding process will take place through modulating the parameter q. If a GRSF model is chosen, then either value of the parameter g or/and q can be modulated. The parameters that the user requires to consider are itemised as follows:

- Initial values
  - RSF model : N, seed
  - GRSF model: N, seed, K0
- Parameter sets
  - RSF model : c, q
    - Modulate q: (q1,c1), (q2,c2), (q3,c3) and (q4,c4).
  - GRSF model: c, g, q
    - Modulate g: (g1,c1), (g2,c2), (g3,c3) and (g4,c4).
    - Modulate q: (q1,c1), (q2,c2), (q3,c3) and (q4,c4).
    - Modulate g and q: (g1,c1), (g2,c2), (q1,c1) and (q2,c2)

The basic m-code for reading a file and its conversion to binary form is as follows:

- Open and read a ciphertext file

```
fid = fopen('ciphertext.bin','rb');
[INtext]=fread(fid);
status=fclose(fid);
```

- Convert the given text into binary form

```
for i=1:length(INtext)
    bin_bits=dec2bin(INtext(i),8);
    Stream_bits=[Stream_bits bin_bits];
end
```

7.3.1 Modulation Module

The theoretical background to the design of this module has been explained in detail in Chapter 3. Herein, we present the m-code for the modulation module. The demodulation module is given in the following section. In case of the RSF model, this module inputs four values for each of the parameters q and c together with the binary stream to be encode. For each bit-pair in the given stream, 00, 01, 10 and 11, it assigns a value for q and another value for c and then generates a fractal signal of size N=1024 by filtering a white noise field using the fractal filter $\frac{c}{\omega^q}$. Finally, it concatenates all the fractal signals generated to produce a contiguous stream of modulated signals.

```
for i=1:2:length(Stream_bits)
    bit=Stream_bits(i:i+1);
    switch bit
        case  '00'
             ry=GenFract1q(N,q1,c1); fs=[fs ry];

        case  '01'
             ry=GenFract1q(N,q2,c2); fs=[fs ry];

        case  '10'
             ry=GenFract1q(N,q3,c3); fs=[fs ry];
        case  '11'
             ry=GenFract1q(N,q4,c4); fs=[fs ry];
    end
end
```

7.3.2 Watermarking module

To exchange the Cut-Off Points (COPs), namely CP1, CP2, and CP3, between the sender/receiver, this program embeds a watermark into the modulated signal; the watermark is a chirp coded signal of these COPs.

Chirp Coding

- Inputs the COPs: CP1, CP2 and CP3.
- Converts COPs into binary form and concatenate them.

```
for i=1:3
    cp=dec2bin(CP(i),10);
    CP_bin=[CP_bin cp];
end
```

- Initialise chirp

```
t=0:1/441:99/441;
y=chirp(t,00,100/441,100);
```

Once the COPs have been input converted to binary form a chirp is initialized. Each binary bit is multiplied by the chirp. A bit 1 produces a positive chirp (starts with +1) and a bit 0 produces a negative chirp (starts with -1). These chirps are then concatenated together to produce a coded signal of a certain size depending on the number of bits and the length of the chirp signal.

```
for i=1:30
    if str2num(CP_bin(i))==0
        x(i)=-1;
    else
        x(i)=1;
    end
end

for j=1:30
    z=x(j)*y;
    znew=cat(2,znew,z); %Concatenate the chirp coded signals.
end
```

Watermarking Insertion

- The program scales the chirp coded signal and computes the watermark signal.

```
znew1=znew./div_fac;
fs1=fs(1:length(znew1));
wmark_sig=znew1+fs1;
```

- The watermarked signal is rescaled and concatenated to the original (fractal modulated) signal.

```
wmark_sig1=wmark_sig*max(fs1)/max(wmark_sig);
Send_Signal=cat(2,wmark_sig1,fs);
```

Flags and Writing to a File

For the receiver to be able to recognize which model has been used in the modulation process flags are introduced into the data. These flags consist of spike(s) of unit amplitude. The sender introduces a number of spikes depending on which fractal model is used and which parameter value(s) has been modulated in the encoding process. In the RSF model case, the modulation process is based on the parameter q alone and so one spike is added.

If the GRSF is used then there are three alternatives associated with modulation process, i.e. modulate g or/and q parameter. In this model one of the two parameters is fixed when the other is modulated. Two spikes are added when the modulation is based on g, while three spikes are added if the modulation is based on q. When both g and q are used for modulation, four spikes are added.

```
switch parameter  % adding spikes of amplitude 1.
                  case 1 %modulate g.
                       Send_Signal=Send2g;
                       Send_Signal=[1 1 Send_Signal];
                  case 2 %modulate q.
                       Send_Signal=Send2q;
                       Send_Signal=[1 1 1 Send_Signal];
                  case 3 %modulate g and q.
                       Send_Signal=Send2gq;
                       Send_Signal=[1 1 1 1 Send_Signal];
                  otherwise
                       error_message;
                  break
          end % end  switch of parametr.
```

The modulated signal is then saved after adding the flags.

```
fid=fopen('C:\MATLAB6p5\work\Multi-Fractal Modulation\...
Signal_File.dat','w');
c=fwrite(fid,Send_Signal,'float');
fclose(fid);
```

7.4 Selection of Parameter Values

The program comes with pre-configured (hard-wired) parameter values. These parameters are used for encoding binary bits and include the scaling parameter(s), seed value and Fourier dimension for the RSF model.

For the GRSF model the parameters include the scaling parameter(s), characteristic frequency k_0, seed value and the Fourier dimensions g and q. In both models there are a number of Cut Off Points (COPs) associated with such parameters of the given models which are also pre-configured.

The pre-configured parameter values have been chosen through executing the program 'Main_Fractal.m' to obtain the tables for these values as given in Chapter 3. The automated-optimise selection of parameters is the subject of future work which lies beyond the scope of this thesis.

The chirp function (associated with the method explained in Chapter 4) in MATLAB needs four parameters to run, i.e.

$$y = chirp(t, f_0, t_1, f_1)$$

The value of t determines the length (period) of the chirp itself. The values of f_0, t_1, and f_1 are explained later in this chapter.

7.4.1 Demodulation Module

This module is the reverse of the modulation module in which restoration of the encoded bits present in the received signal is considered. The principal aspects of the MATLAB code associated with this module are as follows:

- Open and read the file of the received signal

```
fid=fopen('C:\MATLAB6p5\work\Multi-Fractal Modulation\...
Signal_File.dat','w');
Recv_Signal=fread(fid,'float');
fclose(fid);
```

- Determine the used model and parameter. To decode the binary bits from the received signal it is necessary to determine which model and parameters have been used in the modulation module.

7.4.2 Cut-Off Points (COPs) Extraction Module

In this module, the embedded COPs are extracted and then passed on to the demodulation module. In order to extract the COPs, the user must have a copy of the exact parameters used by the sender to reconstruct the chirp. Once the chirp is reconstructed, it is then correlated with the modulated signal for COPs extraction. The principal source code associated with the module is itemised below.

1. Initialize chirp function.

```
t=0:1/441:99/441;
y=chirp(t,00,100/441,100);
```

2. Correlate to recover the COPs from the chirp coded signal. The COPs are recovered in terms of a set of positive and negative integers. The interest does not lie in the integers themselves, but in their sign; positive values are denoted by 1 and negative numbers by -1.

```
for i=1:30
  yzcorr=xcorr(Chirp_Signal(100*(i-1)+1:100*i),y,0);
  r(i)=sign(yzcorr);
end
```

3. Recover bit stream. Assign 1 to a positive number and 0 to a negative number.

```
for i=1:30
  if r(i)==-1
```

```
        rec(i)=0;
    else
        rec(i)=1;
    end
end
```

4. Restore the cut-off points

```
recov=(num2str(rec,-8));
CP1=bin2dec(recov(1:10))/100;
CP2=bin2dec(recov(11:20))/100;
CP3=bin2dec(recov(21:30))/100;
```

7.4.3 Bit Decoding

After separating the watermarked signal form the received signal and restoring the COPs it is necessary to demodulate the rest of the signal to recover the encoded bits. This can be done through estimating the values of the used parameters from the received signal. To do this a window of size N=1024 is moved over the signal and for each window position the Power Spectrum Method is used to estimate the parameter value that was used in the modulation process. The m-code required for this is as follows:

- **Estimate the Modulated Parameter**

```
for j=1:N:length(fs);
    C1=C1+1;
    FS=fft(fs(j:N*C1)); a=real(FS); b=imag(FS);
```

```
%Compute the emperical power spectrum.
    for i=1:N/2-1
    p(i)=a(i)^2+b(i)^2;
    end
  a11=0;  a12=0; a1=0; a2=0;
    for i=1:N/2-1
    k=abs(i-m);
        a1=a1+log(k);
        a2=a2+log(p(i));
        a12=a12+log(k)*log(p(i));
        a11=a11+log(k)*log(k);
    end
% estimate the value of q.
    estm_q=((N/2-1)*a12-a1*a2)/(a1^2-(N/2-1)*a11);
% Concatenate the estimated values of q.
    recv_param=[recv_param estm_q];
end
```

- **Apply Thresholding Technique.**

  Herein, the estimated values of the used parameters in the encoding process are compared with the restored COPs values and the encoded bit-pairs recovered.

```
for i=1:length(recv_parm)
     if recv_parm(i) < CP1
          recv_bits=[recv_bits '00'];
     elseif  recv_parm(i) > CP1 & recv_parm(i) < CP2
          recv_bits=[recv_bits '01'];
```

```
        elseif  recv_parm(i) > CP2 & recv_parm(i)< CP3
            recv_bits=[recv_bits '10'];
        else
            recv_bits=[recv_bits '11'];
        end
    end
```

– **Recover the cipher text**

```
    for i=1:8:length(recv_bits)
        c=c+1;
        Text=char(bin2dec(recv_bits(i:c*8)));
        OUtext=[OUtext Text];
    end
    fid=fopen('ciphertext.bin','wb');
    cpunt = fwrite(fid,OUtext);
    status=fclose(fid);
```

7.4.4 Chirp Function

The chirp function in MATLAB generates a swept-frequency cosine (chirp) signal. The chirp block outputs a swept-frequency cosine (chirp) signal with unity amplitude and continuous phase. To specify the desired output chirp signal, its instantaneous frequency function must be defined, also known as the output frequency sweep. The frequency sweep can be linear, quadratic, or logarithmic, and repeats once every sweep time by default.

We can obtain a variety of chirps which can be implemented differently.

Here the main use of a chirp is to embed COPs. Thus, given a chirp function with certain parameters, the recipient needs to recover the same parameters in order for him/her to recover the COPs. The sender has a wide range of parameter values to use which the recipient must have *a priori*.

The MATLAB chirp function used here has the general form

$$y = chirp(t, f_0, t_1, f_1)$$

and generates samples of a linear swept-frequency cosine signal at the time instances defined in array t where t is time instance (sec), f_0 is the instantaneous frequency at time $t = 0$ (Hz) and f_1 is the instantaneous frequency at time t_1 (Hz). The chirp function can be set to run in three different modes - linear, quadratic and logarithmic as follows:

Linear: $f_i(t) = f_0 + \beta$ (instantaneous frequency sweep) where $\beta = (f - 1 - f_0)/t_1$ which ensures that the desired frequency breakpoint f_1 at time t_1 is maintained.

Quadratic: $f_i(t) = f_0 + \beta t^2$ where $\beta = (f_1 - f_0)/t_1^2$. If $f_0 \geq f_1$, the output waveform is a downsweep, with a default shape that is convex. If $f_0 < f_1$, the output waveform is an upsweep, with a default shape that is concave.

Logarithmic: $f_i(t) = f_0 + 10^{\beta t}$ where $\beta = (\frac{f_1}{f_0})^{\frac{1}{t_1}}$

7.5 FracNet: Transferring a File via the Internet

The developed software is called 'FracNet' by which we transfer a digital data file through splitting the binary form of the given file into a number of binary files of different sizes. At the same time a 'time table' of timestamps is constructed where each of resulting files from the splitting process is sent via the Internet as an e-mail attachment. The main steps in sending and receiving of the digital file are summarized as follows:

Sending Procedure

- Capture real Internet traffic.
- Estimate the fractal parameter from this traffic.
- Generate a synthetic fractal trace.
- Encrypt the given plaintext file.
- Covert the ciphertext to binary file.
- Split the file into a number of binary files of fractal size.
- Generate a list of timestamps of fractal times.
- Attach each file in an e-mail and send each attachment for specific timestamp.

Receiving Procedure

- Concatenate the received binary files in one file.
- Convert to ASCII and then to ciphertext file
- Decrypt the resultant file.

Note that most of the work is to be done by the sender.

Comma Separated Value (CSV) files.

We assume that a trace of Internet traffic has been captured using the *tcpdump* utility, which is widely available in Unix environments. The captured trace consists of the sizes of the arrived packets and the corresponding timestamps of their arrival. This data set is saved as a Comma Separated Value (CSV) file, say, 'Trace1.txt'

Read the Traffic Trace

```
RNG=[0,0,5000000,4];
Trace=dlmread('C:\Documents and Settings\...
Desktop\HR-Traces\Trace1.txt',',',RNG);
```

The code here describes the reading of the first $5x10^6$ lines of numeric data form the ASCII delimited file 'Trace1.txt'. The array Trace contains 5 columns, the first four columns are of the timestamps and the last column is for the sizes of packets that arrived at these timestamps.

For example, the following segment represent the data that have been read using the above code from the file 'Trace1.txt'. The first four columns are the timestamps and the last column is the packet size. For example, the first line means that a packet of size 1460 bytes has arrived at '10:30:01.560108'

```
Trace =

    10    30    1    560108    1460
    10    30    1    560118     588
    10    30    1    560126     588
    10    30    1    560171    1452
    10    30    1    560334    1460
    10    30    1    560392     588
    10    30    1    560409     596
    10    30    1    560424       0
    10    30    1    561199       0
    10    30    1    562322    1024
    10    30    1    562329       0
```

Aggregating of a Time Series of Packets or Bytes

The program permits the sender to enter a time unit which is used to aggregate the packets or bytes that arrived in the time unit. The time unit can be 1 sec (1000 ms), 100 ms, 10 ms or 1 ms. However, there is an option for the user to either aggregate the number of packets or bytes that have arrived in a given time unit.

Estimation of the Fourier Dimension

Once the packets or the bytes are aggregated in a time series, then the following MATLAB code is used to estimate fractal parameter q (the Fourier Dimension) from this series. The estimated value of q reveals the degree of self-affinity in the aggregated traffic. The results of the estimation for such parameters is presented in Chapter 5.

```
Result=[];
Number=Packet_Series;
format short
if length(Number) < 1024
    disp('This size of data should 1024 or more...');
    return;

else
D=fix(length(Number)./1024);
Trc=Number(1:D*1024);
N=1024; %size of the signal that we need.
m=N/2+1;
C2=0;jj=0;
for C1=1:N:length(Trc)
    jj=jj+1;
    C2=C2+1;
fs=Trc(C1:C2*N); % Extract only signal of size 1024.
fs=fs/max(fs);   % Normalize the signal.
FS=fft(fs);      % Calculate the FFT.
FS=fftshift(FS); % Apply shifting.
ps=abs(FS).^2;   % Calculate the power spectrum.
p=ps(1:N/2);     % Take the left halve of power spectrum.
%calculations to recover the estimated q (estmt_q).
x1 =0; %sum[log(k)]
x2 =0; %sum[log(p)]
x12=0; %sum[log(k)*log(p)]
```

```
x11=0; %sum[log(k)^2]

 for i= 1:N/2
     k=abs(i-m);        % k=frequency.
     if((p(i)~=0) & (k~=0))
         x1=x1+log(k);
         x2=x2+log(p(i));
         x12=x12+log(k)*log(p(i));
         x11=x11+log(k)*log(k);
     else
    N=N;
     end
 end
 estm_q=[(N/2)*x12-x1*x2]/[x1^2-(N/2)*x11];
 Result(1,jj)=C2;
 Result(2,jj)=estm_q;
end % end of windowing.
end
```

7.5.1 Generation of a Synthetic Fractal Trace

A synthetic fractal trace is a sequence of integer numbers which represents the sizes of the split files.

Generate a fractal noise sequence

Here, the user enters a value for the fractal parameter q; this value can be an estimated value of q from real internet traffic. The fractal noise sequence is generated by filtering a white noise sequence using the fractal model $\frac{1}{\omega^q}$

```
N=1024;          %size of the signal requiered.
m=N/2+1;         %Calculate the mid-point, say m, of the signal.
seed=4;          %Locate the seed.
randn('seed',seed);
s=randn(1,N);   %Generate GWN signal s of size N.
y=fft(s);        %Take FFT  of the signal s.
for i=1:N
    k=abs(i-m);
    if k==0             %Exclude the DC, where K=0.
        y(i)=y(i);
    else
        PS=1/k^q;       %Calulate the fractal filter.
        y(i)=y(i)*PS;  %Aplly the Fractal Filter on the signal.
    end
end
x=ifft(y);               %Take inverse FFT of y.
fs=real(x);              %The fractal signal.
```

The following steps are then implemented:

- Generate a fractal trace.
- Encrypt the given file.
- Split the encrypted file.
- Produce a tabulated list of timestamps.
- Attach each file to an e-mail and send each attachment at a specific timestamp.

Manipulating the Generated Fractal Sequence.

This module manipulates the generated sequence of decimal numbers to produce an equivalent sequence of an integer numbers which is composed of the sizes of the split files. In the following code there is a multiplication by scaler parameter which being a power of 2.

```
H=hilbert(fs);
%Take the Modulus of H, to obtain Md.
Md=abs(H);
%Normalize Mdh, to obtain Mdh1.
Md1=Md./max(Md);
%Multiply Mdh1 by 256, to be the scale from 0 to 256.
Md2=(256*Md1);
%Round to the nearest integer num.
Trace=round(Md2);
```

File Splitting

The file is split into a number of smaller files. All the split files are saved in an archive called 'Send-Files'. The mechanism of splitting starts by taking the binary stream of the given file and the generated fractal trace of integer numbers. The number of split files is equal to the length of the generated traffic trace. The program counts a number of bits in the stream that is equal to the value of elements in the trace. However, if the sum of numbers in the integer trace is greater than the length of bits in the stream then the program continues to split the stream until there are no more bits in the stream and neglects the rest of the elements in the trace sequence that are not used. If the length of the bit stream is more than the sum of numbers in the integer trace then the program saves the last sub-stream in the last file regardless of the value of the last element in integer trace. This process is the basis for the following m-code.

```
NUM =length(R);
for i=1:NUM
    C2=C2+R(i);
if length(Bn)-C2<0
    C2=length(Bn);
end

%This  when the length of Bn is  more than sum of numbers in R.
if length(Bn)>sum(R) & i==1024
   C2=length(Bn);
end
```

```
    substrem=Bn(C1:C2);
    C1=C1+R(i);
     file = strcat('Send-Files\file',num2str(i),'.txt');
    fid = fopen(file,'wt');
    fwrite(fid,substrem);
    fclose(fid);
end
```

7.5.2 Generation of a Table List of Timestamps

This module generates a list of timestamps where the split files are to be sent at times specified by this list. To generate this list we first estimate the fractal parameter from the real packets inter-arrival times sequence. This obtained by subtracting the consecutive timestamps of packets in the captured trace of internet traffic.

```
Hour=Trace(:,1);
Minute=Trace(:,2);
Second=Trace(:,3);
Msec=Trace(:,4);

Tim=360*Hour+60*Minute+Second+Msec./1000000;
for i=2:length(Trace)
    Intr(i)=Tim(i)-Tim(i-1);% Inter-arrival times sequence.
end
```

The timestamps generated are in military form , i.e. 'hh:mm:ss', where hh, mm, and ss denote hours, minutes, and seconds, respectively.

7.5.3 Defragmentation of Received Files

Once the attached binary files have arrived at the recipients site, then its contents of bits are concatenated to form a contiguous stream which is converted into text and saved in a file. This file is the ciphertext file which is then decrypted using Crypstic^{TM} to restore the plaintext file. Note that the security associated with this software is compounded in the following points:

- The generated fractal trace depending on the used fractal parameter and this is based on the estimated values from real Internet traffic trace around the time the data is sent.
- The scaling parameter which adjusts the maximum number of bits in the split files of generated trace ; which takes a value being of power 2.
- The generated list of timestamps which depends on both the estimated parameters from a real traffic trace and on the initial time that is chosen for executing the program; these are not fixed but change from time to time.

Chapter 8

Summary of Work and Conclusions

This thesis has discussed cryptography in the context of the transmission of encrypted information using a covert approach. This has been achieved by considering the noise characteristics of the environment in which information is transmitted and has involved a study of random scaling fractal fields for the purpose of modelling such environments (e.g. Internet Traffic). The object of this study has been to develop a system in which the transmission of encrypted data is difficult to track by a potential interceptor thereby reducing the risk of an attack initiated by observing a communication that includes encrypted data.

The method of encryption can be based on the use of any commercially available product, e.g., Crypstic^{TM} originally developed at Loughborough University and marketed by Crypstic Limited.

Details relating to this system are provided on the CD that accompanies this thesis. Crypstic^{TM} is a dynamic multi-algorithmic system that performs an *iterative nonlinear transformation of information* in an unpredictable but deterministic manner. In terms of chaos theory, the sensitivity to the initial conditions together with the mixing property ensures cryptographic confusion and diffusion.

The work in this thesis has been designed to extend the Crypstic^{TM} system so that the encrypted information it produces can be transmitted covertly using the fractal models for Radio and Internet Traffic considered for this thesis.

8.1 Cryptography using Chaotic Systems

Chaotic systems are algorithmically random and thus cannot be predicted by a deterministic Turing machine even with infinite power. However, chaotic systems are predictable by a probabilistic Turing machine. Finding probabilistically unpredictable chaotic systems is a central problem for chaos based cryptography. A rarefied sample $x_k, x_{2k}, \ldots x_{nk}, \ldots$ from a time series $x_1, x_2, x_3, \ldots$, produced by a chaotic and mixing system, is asymptotically independent: for any n, elements $x_{(n-1)k}$ and x_{nk} will be more and more independent as k increases.

Chaotic systems with analytical solutions of the form $x_n = \Psi(x_0, n)$ say and multi-valued maps $x_{n+1} = f(x_n)$ can, theoretically, deliver computationally unpredictable (pseudo-random) sequences. The advantage of such a generator is the random access, i.e. any element x_n can be computed directly from the initial condition (seed) x_0.

The cryptographic secrecy is kept in the seed and the solution $\Psi(x_0, n)$. All conventional cryptographic systems (encryption schemes, pseudo-random generators, hash functions) are *binary pseudo-chaotic systems*, defined on a finite space of strings. Such systems are periodical but have a limited sensitivity to the initial conditions, i.e. the Lyapunov exponents are positive only if measured at the beginning of the process (before one can see the cycles). This mixing property leads to pseudo-randomness.

Iterative block ciphers can be viewed as a combination of two linked pseudo-chaotic systems: data and round-key systems. The iterated function of the data system includes: nonlinear substitutions, row shifts, column mixing etc. The round-key system is a pseudo-random generator providing the sensitive dependence of the ciphertext on the key.

Technically, most pseudo-random generators are based on a *stretch-and-fold* transformation: first, the state is stretched over a large space (e.g, multiplying or raising in power), then folded into the original state space (using an appropriate periodical function). In mathematical chaos, the stretch-and-fold transformation forms the basis of the majority of iterated functions.

Like all modern encryption systems that depend on the exploitation of pseudo random sequence generators (use pseudo random or pseudo chaotic methods), there can be no total guarantee that the information is not 'open' to a successful attack. This fact is the principal issue upon which this thesis has indirectly focused.

The transmission of encrypted data using the covert techniques discussed in this thesis provides a greater level of security, not in terms of the 'strength' of the encrypted data, which has been attempted by Crypstic Limited through

application of a multi-algorithmic approach using deterministic chaos, but in terms of the 'weakness' of an interceptor to recognise that a transmission has taken place.

8.2 Multi-fractal Modulation

Current wireless communication systems rely on two principal modulation techniques, namely, frequency modulation and phase modulation coupled with the use of techniques for increasing the temporal complexity of a transmission such as frequency hopping or spread spectrum.

Fractal modulation has a synergy with frequency modulation in the sense that it is based on modulating the value of the fractal dimension using just two states. Multi-fractal modulation, as developed in this thesis, is based on using four values of the fractal dimension to encode four bit-pairs (i.e. 00, 11, 01 and 10).

However, unlike frequency or phase modulation, fractal and multi-fractal modulation both attempt to generate a signal from a given bit stream that has characteristics that are indistinguishable from the background noise of a wireless (or wirebased) environment. In this thesis, we have not only shown that multi-fractal modulation can be applied in a practical sense for a standard random scaling fractal model (involving a two parameter problem - the scaling constant c and Fourier dimension q) but have extended the method to include a generalised random scaling fractal model (involving a four parameter problem - the scaling constant c, the carrier frequency ω_0 and the Fourier dimensions q and g).

8.3 Internet Traffic Noise

The work reported in this thesis has included an extensive study of Internet traffic noise which has been shown, through the many publications now available, to have fractal properties. The principal objective of this thesis has been, by application of the methods developed for multi-fractal modulation, to design a practical approach to transmitting information (encrypted or otherwise) that is characterised by the fractal properties inherent in the Internet at or around the time a transmission is to occur. This has been achieved by generating Internet compatible fractal parameters for the segmentation of a file attachment into a sequence of files which are then sent at specified times through a series of multiple email transmissions (with associated attachments). A variety of models have been considered for this process.

8.4 Self-authentication

The material discussed in Chapter 4 introduces a novel method for the covert transmission of data by watermarking such data using chirp coding. In principle, this approach can be used to embedded any bit stream in any signal. This requires the conversion of any data type into a bit stream either directly or through some transformation appropriate to the application. For applications in which the bit stream is a direct or indirect representation of the signal to be watermarked, a process can be implemented that provides a method of self-authentication, i.e. where the signal authenticates itself. This process has a number of potential applications including digital rights management.

Further, most watermarking methods require access to the original data in order to verify the existence or otherwise of a watermark, especially when the watermark is a deterministic signature rather than a statistical signature. The self-authentication technique discussed in Chapter 4 and applied to audio .wav files does not require access to the original data. However, the principal application of this technique with regard to the remit of this thesis has been to transmit the cut-off points that are needed to recover the ciphertext from a fractal modulated signal. But since the ciphertext (and thus the plaintext) can not be recovered from a fractal modulated signal without access to the cut-off points, the approach used for their covert transmission from sender to receiver, indirectly represents a form of data authentication but not self-authentication, as discussed in Chapter 4.

8.5 Software Development

The development of the software used to investigate the methods developed for this thesis and the design of a prototype package has formed a major part of the work undertaken. The MATLAB source code developed has been packaged into a simple user interface that allows the user to implement covert transmissions through the Internet. This can be accomplished with or without application of Crypstic^{TM}, i.e. either the plaintext can be transmitted or the Crypstic^{TM} ciphertext. The next obvious stage in the software development of this system is to integrate the procedures into Crypstic^{TM}. This requires the object library developed to be re-written in C++ and an appropriate system integration to be implemented.

8.6 Further Development and Extensions

8.6.1 Fractal Modulation

The extension of fractal modulation to multi-fractal modulation for bit-pair encoding and the successful implementation of this process for a conventional random scaling fractal model, leads us to consider the case when multi-value Fourier dimensions are used to encode bit-triplets, bit-quartets, i.e. using slight changes in the value of the Fourier dimension to encode the sequences 000, 111, 011, 110, 010, 101,... and 0000, 1100, 1011, 0011, 1010, 0001,... If successfully applied, such an approach could yield smoother transitions from one bit sequence to another. In the method considered here, application of appropriate scaling values have been required in order to produce a contiguous stream of fractal noise with a constant amplitude envelope. These scaling parameters have then been required to recover the information required. Extending multi-fractal modulation to include multiple bit-sets may provide a method that does not necessitate re-scaling. However, this approach must be off-set with the accuracy available through application of the Power Spectrum Method to recover the Fourier dimension using a conventional moving window principle.

8.6.2 Watermarking

Watermarking appropriate data with hidden (encrypted) data has a number of benefits with regard to covert transmission. The most important feature concerns the transmission of covertly encrypted data which should be avoided

if possible. The reason for this is that it demonstrates to an interceptor that the information being transmitted may have some importance and that it is therefore worth attacking the transmission. This aspect of data analysis can of course be used to propagate disinformation, i.e. encrypting information that you want the enemy to know and to be confident of especially when they have had to put work into extracting it from a relative strong (but not too strong!) ciphertext.

On the other hand, it is often very useful to transmit sensitive information by embedding it in non-sensitive information. The chirp coding method discussed in Chapter 4 provides a way of hiding information in a data file composed of a signal that can be of any appropriate form and type. For example one can watermark an audio file with chirp coded data without distorting the audio output in any way. Transmission of a watermarked audio file therefore provides a way transferring sensitive data that looks 'normal'. In this sense the audio data can be used as a form of camouflage. The same idea can be applied to fractal signals.

In the case of fractal and/or multi-fractal modulation, the modulation method can be used convey one type of data and the watermarking method discussed in Chapter 4 used to convey another type of data; the data types being either independent or inter-dependent, encrypted or otherwise. Thus, the introduction of fractal modulation and chirp based watermarking introduced in this thesis provides a range of combination that can be used for the covert transmission of sensitive data that lie beyond the scope of this thesis and form the basis for interesting future work.

Appendix A

MATLAB Prototyping

A.1 Introduction

This appendix provides details of the MATLAB functions developed for this thesis. MathWorks Inc MATLAB is an ideal platform for numerical work and is routinely used for rapid prototyping, i.e. the rapid development of MATLAB code for testing new algorithms. This includes use of the large library of intrinsic functions offered by MATLAB and the increasingly wide range of specialist toolboxes offered by the system.

There are two main sections in this appendix, in the first one we describe the software programs that were developed for the purpose of transmitting data in wireless communications. The other section includes a description of the software programs that were developed for the purpose of transferring data via a non-secure channel, like the Internet.

The MATLAB functions (full MATLAB code) are given in the accompanying CD at the back of this thesis.

A.2 Wireless Communications: Multi-Fractal Modulation and Demodulation

This section describes both processes of encoding a plaintext file that is to be sent in the form of a modulated fractal signal and demodulating the received signal to recover the plaintext file. The plaintext could be a text, an image or an audio file.

A.2.1 Main_Sending

The Main_Sending folder contains all the developed MATLAB programs used for sending data.

MainSend.m

This is the main program that is execute by the sender, which calls all other functions that are necessary for encoding a stream of binary bits. This program contains two phases, the first is the encryption and the other is the modulation phase. In this program the complete contiguous stream of fractal modulated signals is saved as a data file 'Signal_File.dat'.

This stream constitutes the watermark formed by the chirp signal of the Cut-Off Points and a number of spikes positioned at the start.

The sender's program is launched by running 'MainSend.m'. The receiver mode is 'MainReceive.m' The following two phases and the programs in each of them are then executed sequentially depending on the following selection:

Phase One: Encryption

In this phase the function 'crypstic.exe' is executed to encrypt a plaintext file and produces a ciphertext file, say 'ciphertext.bin'.

Phase Two: Modulation

In this phase the program prompts the sender to choose either the RSF(q, c) or the GRSF(g, q, c) model to generate a fractal signal. If the sender chose the RSF , this means the encoding process of the binary bit will be modulated using the value of the parameter q and the program 'Send1q.m' will be executed.

If the sender has chosen the GRSF model, then the encoding process of the binary bits will be through modulating four different values of the parameter g or parameter q; it may also modulates two different values for each of g and q at the same time. However, if the modulation is for values of g, q or g and q then the programs 'Send2g.m', 'Send1q.m' and 'Send2gq.m' are executed, respectively.

Choosing the Model: RSF(q,c)

Here we shall describe the main programs that are executed when the sender chooses the RSF model for the purpose of modulation.

Send1q.m

This program opens and reads the 'ciphertext.bin' file and then converts its contents into a stream of binary bits. After that it calls the program 'Modulate1q.m' to encode this stream. The program 'Send1q.m' prompts the sender to input four different values of parameters q, and q_4 and also four different values of scaler parameter c, that corresponds to the values of the parameter.

Here the size of signal is fixed, $N = 1024$. The Cut-Off Points (COPs), CP_1, CP_2 and CP_3 are also entered in this program by the sender. These values are pre-defined depending on the values of q, so it changes as the values of parameter change.

For the purpose of exchanging the COPs between the sender and the receiver, this program embeds the watermark signal of these points in a modulated fractal signal. The watermark signal is the chirp coded signal of the binary representation of the COPs. The watermarked signal is then concatenated with the modulated fractal signal.

Modulate1q.m

This program receives the binary bits stream and all values of the parameters, and start to encode each bit-pair,(00, 01, 10 and 11) by generating a fractal signal of size $N = 1024$ corresponding to the parameter values q_1, q_2, q_3 and q_4. It concatenates all the signals into one contiguous stream of modulated signals. The generation of a fractal signal is executed by calling the program 'GenFract1q.m'.

GenFract1q.m

This program generates a white Gaussian noise (WGN) signal of size $N = 1024$ and applies the fractal filter, c/ω^q to the signal, where q is q_1, q_2, q_3 or q_4 and c equals to c_1, c_2, c_3 or c_4. The benefit of the scaling parameters is to adjust all the modulated signals to be similar.

Chirp_Cod3.m

This program takes the COPs, CP1, CP2, and CP3 , as decimal values and binarize them then encode the resultant binary stream using chirp function , chirp(t,00,100/441,100), to produce a chirp coded signal of that COPs.

Choosing the Model: GRSF(g, q, c)

In case the sender chose the GRSF(g,q,c) model to encode the stream of binary bits, then the modulation will be either for value of g or/and q.

Send2g.m and **Send2q.m**

These programs do the same thing as in 'Send1q' but the fractal filter that is used by both programs is $c\frac{\omega^{2g}}{(\omega_0^2+\omega^2)^q}$.

Modulate2g.m and **Modulate2q.m**

The two program modulate four different values for each of parameter g and q, respectively. Each of such programs open and read the 'Ciphertext' file and one of the two programs calls 'GenFract2g.m' and the other calls 'GenFract2q.m' to generate a fractal signal with the given values of the parameter q, g, c, $k0$, where in case of modulating value of g then value of q is fixed but we fix q when the modulation is in g value.

GenFract2g.m and **GenFract2q.m**

These programs generate a fractal signal corresponding to the given values of g and q, respectively. On the other hand, if the sender chose the two parameters g and q in the encoding of bits then the program 'Send2gq.m' is executed in which it the function 'Modulate2gq.m' is called to encode a two bit-pair in the binary bits stream by modulating two values of each g and q, namely, g_1, g_2 and q_1, q_2.

The generation of a fractal signal is through calling the function 'GenFract2gq.m'. In this case the sender need to enter four Cut-Off Points , say CP1, CP2, CP3 and CP4 and the function 'Chirp_Cod4.m' is called by the function 'Send2gq' to produce the chirp signal form of these COPs.

In all of the programs described above, and to be the receiver able recognize which model and parameter(s) have been used in encoded the binary bits , there are flags of spikes of length one have been positioned at the beginning of the modulated signal.

A.2.2 Main_Receiving

The Main_Receiving folder contains all the developed MATLAB programs that used for restoring original data form the modulated signal in which it was transmitted.

MainReceive.m

'MainReceive.m' program is used by the recipient and calls all functions needed for decoding and recovering the plaintext form the received signal. The re-

ceiver's program is launched by running 'MainReceive.m'. The following two phases and the programs in each of them executed sequentially.

Demodulation Phase:

In particular, in this phase of this program it is decided, depending on the number of spikes at the beginning of the received signal, which model and parameter have been used by the sender in the modulation process. Once it is decided so one of the functions (Receive1q.m, Receive2g.m, Receive2q.m, or Receive2gq.m) is called after removing the spike(s).

Receive1q.m, Receive2g.m, Receive2q.m, or Receive2gq.m

One of these functions is called by the 'MainReceive' depending on the number of spikes. In the called function the extraction of COPs takes place by calling the function 'Chirp_Decod3.m' or the function 'Chirp_Decod4' in case of there being four Cut-Off Points.

In each of these receiving functions recovers values of the parameters that are used in the encoding process are obtained from calling either Demodulate1q.m, Demodulate2g.m, Demodulate2q.m or Demodulate2gq.m. These values are compare with the COPs to restore the binary bit stream.

Chirp_Decod3.m or Chirp_Decod4

This program takes the watermarked segment of the received signal and correlate it with a chirp signal identical to that used for chirp coding at the side of sender, i.e., chirp(t,00,100/441,100). After that the COPs are restored.

Demodulate1q.m, Demodulate2g.m, Demodulate2q.m or Demodulate2gq.m

After removing the added flags of spikes and the embedded watermark signal of COPs, one of these programs receives the modulated signal and uses the move window principle with a window size of $N = 1024$ to move over each signal to estimate the required parameters from each segment, using the Power Spectrum Method, and then concatenates the estimated values to recover the vector.

Decryption Phase

In this phase and in the program 'MainReceive.m', the recovered ciphertext is taken and the 'Crypstic' software is used to decrypt 'ciphertext.bin' file to obtain the recovered 'plaintext.txt' file.

A.3 FracNet: Internet Communications

In this section we describe the software programs, that is called FracNet, developed for the secure transferring of a digital data file via Internet communications. Here, there are two folders, the first one is 'Real_FracNet' and the other is 'Synth_FracNet'.

A.3.1 Real_FracNet

This folder contains all the MATLAB programs that are required to analyse the real Internet traffic.

Main_Real.m

This is the main program in which the captured real internet traffic data is read as a text file. This data represents packets sizes (in bytes) with the corresponding timestamps for a resolution of 1 microsecond. This main program calls three functions, namely, 'NUM_PACKET.m' , 'NUM_BYTE.m' and 'Estm_q.m'

NUM_PACKETS.m

This function aggregates the number of packets that arrive in a given time interval. There is an option in the function to choose on of the time intervals, namely, 1 sec., 100 ms, 10 ms, and 1 ms. So the input of this function is the captured trace of real internet traffic and the required time interval, whereas, the output is a time series of the number of packets that arrived in a time interval.

NUM_BYTES.m

This function does the same job as 'NUM_PACKETS.m' but it aggregates the number of bytes.

Estm_q.m

This module measures the fractal behavior of the time series of aggregated bytes (or packets) and of the sequence of inter-arrival times. This is done by estimating the fractal parameter q using the Power Spectrum Method (PSM), in which we adopted the RSF model which is assumed as the theoretical power spectrum of aggregated time series.

A.3.2 Synth_FracNet

This folder contains the required programs to generate a synthetic fractal trace that is used to split an input plaintext file into a number of binary files; it also contains the programs to generate a table list of timestamps.

Main_Synth.m

The 'Main_Synth.m' program calls the required functions to accomplish these tasks. The functions are, RSF.m, Fract_Trace.m, Fragm_File.m, DeFragm_Files, and Gen_Timestamp.

RSF.m

In this function a fractional Gaussian sequence is generated with an option for the value of the fractal parameter q.

Fract_Trace.m

This function manipulates the generated fractional sequence of floating points to produce a fractal trace of integer numbers. These numbers represent the sizes of the split files.

Fragm_File.m

In this program a ciphertext file, in its binary form, is split into a number of binary files. This ciphertext results from the encryption of an input plaintext in the main program, and the encryption is done by the 'Crypstic' engine. Herein, the split files are saved as text files of binary bits in the 'Send_Files' folder; these files are sent later as attachments according to a list of timestamps which is generated by the function 'Gen_Timestamp.m'.

Gen_Timestamp.m

To send the fragmented files as attachments we require the times (*timestamps*). This function generates a fractal sequence of inter-arrival times, with an option on the fractal parameter q. This sequence, with an option on the initial timestamp, is used to produce a table of timestamps in military form.

DeFragm_Files.m

This function does the job of defragmentation (concatenation) for the received binary files at the recipients site. Once the binary files are concatenated into one contiguous binary stream, the stream is converted into text and saved in a file, 'Rec-Ciphertext.txt'. This file is the ciphertext file which is then decrypted using Crypstic^{TM} to restore the plaintext file 'Rec-Plaintext.txt'.