2001

Multifractal analysis of memory usage patterns

Jonathan Browning Crowell
West Virginia University

Multifractal Analysis of Memory Usage Patterns

Jonathan B. Crowell

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Bojan Cukic, Ph.D., Chair
Mark Shereshevsky, Ph.D.
Katerina Goseva-Popstojanova, Ph.D.

Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2001

Keywords: Fractal, Multifractal, Hölder exponent, Fractal Dimension,
Operating System, Memory

Abstract

Multifractal Analysis of Memory Usage Patterns

Jonathan Crowell

The discovery of fractal phenomenon in computer-related areas such as network traffic flow leads to the hypothesis that many computer resources display fractal characteristics. The goal of this study is to apply fractal analysis to computer memory usage patterns. We devise methods for calculating the Hölder exponent of a time series and calculating the fractal dimension of a plot of a time series. These methods are then applied to memory-related data collected from a Unix server. We find that our methods for calculating the Hölder exponent of a time series yield results that are independently confirmed through calculation of the fractal dimension of the time series, and that computer memory use does indeed display multifractal behavior. In addition, it is hypothesized that this multifractal behavior may be useful in making certain predictions about the future behavior of an operating system.

The author wishes to dedicate this work to world peace.

Acknowledgements

I would like to acknowledge my parents for their love and support, my committee members for their assistance, and Dr. Cukic and Dr. Shereshevsky for being excellent co-advisors. I could not have accomplished this without the patience and understanding of Dr. Shereshevsky who brought me up to speed on a great deal of mathematics.

Contents

List of Figures

Chapter 1

Introduction

The goal of this study is to suggest a possible method of determining whether an operating system has begun to age. The focus is on the patterns of resource use which, while initially appearing chaotic, are expected to grant insight into the state of the operating system after being subjected to fractal analysis.

Fractal geometry is a relatively new tool within mathematics, having been developed in the 1960s and 70s by Benoit Mandelbrot [12], which can be used to model many of the complex, chaotic phenomena that appear throughout nature. It also appears that many objects of human derivation such as traffic patterns and the stock market display fractal characteristics. If it can be established that the patterns of resource use in operating systems display fractal characteristics, then perhaps these subtle patterns will betray information about the age of the operating system and can be exploited to predict the best times to engage in operating system rejuvenation.

1.1 Fractals Defined

What, exactly, is a fractal? Minor controversies have swirled around the various precise definitions that have been offered, but a few key properties can nevertheless be identified. The five traits that follow have been identified by Kenneth Falconer [4].

1. Fine structure
2. Irregularity
3. Self-similarity
4. Fractional Dimension

5. Recursive Definition

To illustrate these qualities, it will be helpful to consider one of the most famous fractals, the Koch curve. The Koch curve is a mathematical object built by taking a line and replacing its middle third with two other lines each as long as the original removed segment. The two new lines are propped up, forming an equilateral triangle with the removed segment, and resulting in a structure consisting of four lines. The entire is process is then repeated on each of the four resulting lines, and then again on each of the 16 resulting lines, and so on infinitely many times. Figure 1.1 depicts the process. It should be stressed that the true fractal is the end result of performing this operation an infinite number of times – not any one of the intermediate stages.

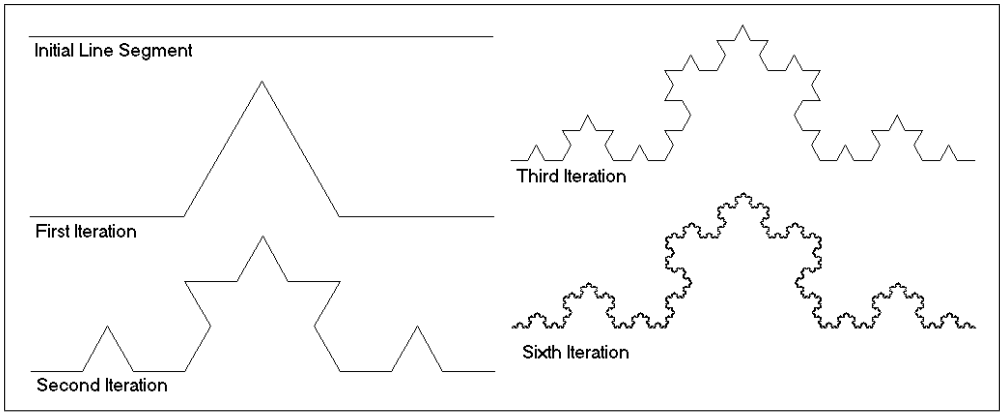

Figure 1.1: The first few iterations toward forming the Koch curve

1.1.1 Fine Structure

The Koch curve obviously has a fine structure in that it will display incredible detail at any scale of magnification. Unlike most shapes that appear complex at first but smooth out as we zoom in closer and closer, fractals such as the Koch curve never lose their detail. The elaborate structure of the Koch curve becomes more evident once we consider its length. The formula for the length of the Koch curve is:

$$1 + \sum_{i=0}^{\infty} \frac{4^i}{3^{i+1}} = \infty$$

Obviously, a line of infinite length twisted an infinite number of times, yet fitting into a finite space will have an infinitely fine structure that reveals detail at any scale.

1.1.2 Irregularity

The term "irregularity" is meant to capture the fact that fractals defy the explanatory tools of traditional geometry. Mandelbrot begins his classic work The Fractal Geometry of Nature with the following words:

> Why is geometry often described as "cold" and "dry?" One reason lies in its inability to describe the shape of a cloud, a mountain, a coastline, or a tree. Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightening travel in a straight line. More generally, I claim that many patterns of Nature are so irregular and fragmented, that, compared with *Euclid* - a term used in this work to denote all standard geometry - Nature exhibits not simply a higher degree but an altogether different level of complexity [13].

The Koch curve is obviously such an irregular shape. While the shapes of the first few iterations may be approximated with lines and triangles, the true fractal that results at the end of the process resists such efforts. Falconer says of the generic fractal: "it is not the locus of the points that satisfy some simple geometric condition, nor is it the set of solutions of any simple equation." [4]

1.1.3 Self-Similarity

If we concentrate only on the left third of the Koch curve at the bottom of figure 1 and blow it up to the size of the whole, we will find that the two are indistinguishable. This self-similarity at finer and finer levels of detail is a salient feature of fractals. Not all self-similarity need be geometric, however. Many fractal structures reveal a statistical self-similarity even when a surface self-similarity is not present. So, while a structure may not behave in exactly the same way at every scale, if it always behaves in the same sort of way, it meets this criterion for self-similarity. The fractal analysis in this study will employ this statistical notion of self-similarity.

1.1.4 Fractional Dimension

Mandelbrot coined the term "fractal":

> Fractal comes from the Latin adjective *fractus*, which has the same root as *fraction* and *fragment* and means "irregular or fragmented" [12]

A crucial sense in which a fractal is *fractus* is in its dimension. While traditional geometry deals exclusively with sets of whole dimensions such as dimension one, two, or three (or even higher),

fractal geometry deals with sets whose dimensions lie somewhere in between two whole dimensions. At first this may seem a nonsensical notion. After all, what could it mean for a mathematical object to have a dimension of 1.262?

A line or curve is, of course, one-dimensional, and a plane is two-dimensional. It turns out that there are some objects that are more than a line, yet less than a plane (and they have their counterparts for any two whole dimensions). The Koch curve provides one example of a geometrical object possessing a fractal dimension between one and two. Although it is built on the foundation of a one-dimensional line, it gradually begins to "fill out" as more iterations are performed. The end result is of infinite length, breaking it out of the bounds of one-dimensionality, yet, although it twists and turns to squeeze into a finite space, has an area of zero, barring it from entering two-dimensionality.

For a more mathematically precise thought experiment, imagine a fractal curve obtained by iterations (such as the Koch curve) where the n^{th} iteration consists of K_n segments of length l_n each. Then its fractal dimension is obtained as the limit of the ratio

$$\frac{\log K_n}{\log \frac{1}{l_n}} = \frac{\log K_n}{-\log l_n}$$

as $n \to \infty$. The ratio which defines the fractal dimension remains the same for the second iteration:

$$\frac{-\log 4^2}{\log \frac{1}{3}^2} = \frac{2(-\log 4)}{2(\log \frac{1}{3})} = \frac{-\log 4}{\log \frac{1}{3}}.$$

In general, for the n^{th} iteration the ratio is still the same:

$$\frac{-\log 4^n}{\log 1/3^n} = \frac{n(-\log 4)}{n(\log \frac{1}{3})} = \frac{-\log 4}{\log \frac{1}{3}}.$$

and therefore this value stays the same as $n \to \infty$, which yields the fractal dimension of 1.262 for the Koch curve.

1.1.5 Recursive Definition

A common trait of fractals that is related to their self-similarity and detail at infinitely small scales is the likelihood of a recursive definition. Giving a base case, providing some sort of rule for a transformation, and then saying "repeat" describes most common fractals. The Koch curve, of course, has an easy recursive definition: 1) take a straight line segment, 2) remove the middle third and replace it with the two sides of the equilateral triangle it would have formed, and 3) repeat this process for every resulting straight line segment.

Not every fractal can be described with so simple a recursive definition, however, and for the kind of fractals studied in this thesis, which are quite chaotic and display only statistical self-similarity,

a recursive definition may be quite complex. In the 1980s Michael Barnsley [2] discovered a unique and powerful variety of recursive definition for fractals known as an Iterated Function System or IFS. The IFS is the method of recursion which holds the most hope for describing the fractal behavior of computer memory use.

1.1.6 A Real-World Example: Coastlines

Some of the most interesting natural objects that display fractal characteristics are coastlines. They do not, of course, display the smooth, neat behavior of traditional geometrical objects. It is surprising, however, just how fractal they are. One naturally thinks of a coastline, say the coast of Britain, as having a length. Mandelbrot [13] has noted, however, that a typical coastlines length is "very large and so ill determined that it is best considered infinite." This is because as the granularity of the measuring tool becomes finer, the length measured increases exponentially. When a coastline is viewed from a great distance, only the largest bays and peninsulas are visible. When viewed from nearby, however, the coastline of each large bay and peninsula is seen to consist of scores of smaller bays and peninsulas — thereby increasing the measured length of the coastline. At an even greater magnification, even smaller bays and peninsulas appear, and so on ad infinitum. This is a classic characteristic of fractal objects.

In fact, coastlines display all five of the characteristics noted above: they have an intricate structure that reveals detail at every level; they are irregular in shape; they are self-similar in that every bay and peninsula is composed of smaller bays and peninsulas and so on; they have a fractal dimension (in fact, Mandelbrot [13] gives a method for calculating the fractal dimension of a coastline based on the slope of the logarithmic plot of the values for the length of a coastline as the length of the measuring tool decreases); and they have a recursive definition based on larger structures being composed of similar smaller structures which are in turn composed of similar yet smaller structures, and so on.

1.2 Overview of the Thesis

This thesis is divided into five chapters. Chapter two is an overview of work that is related to this study, including a discussion of some of the fields in which fractal geometry has been successfully applied in the past. Chapter three discusses the mathematical theory underpinning our research and also gives the specifics of the fractal analysis techniques employed in this study. Chapter four presents our data and the results we obtained. Chapter five concludes the study and offers suggestions for further work along the lines pursued in this study.

Chapter 2

Related Work

2.1 Introduction

The tools of fractal geometry have been applied in many surprisingly diverse fields such as image compression, cardiology, analysis of network traffic flow, and analysis of stock markets. Recently work in the area of software aging has been gaining momentum, and we hope to apply the tools of fractal geometry to this field as well.

In this chapter we will first outline the current work in software aging and rejuvenation, then briefly describe the applications of fractal geometry to image compression, stock market analysis, physiology, and network traffic analysis.

2.2 Software Aging

A phenomenon known as "software aging" has been identified and described in the recent work of Garg et al. [1], Huang et al. [25], and Castelli et al. [22]. It is important to distinguish between two varieties of software aging that are discussed in the literature. The first use of the term was introduced by David Parnas [19] and refers to the degradation of a software system over many years as maintenance of the system takes it in new directions and as the requirements of the organization change. The second use of the term, which is the one we are concerned with here, refers to the degradation of a software system over a period of hours to months as errors accumulate while the system is running. This use of the term "software aging" is applicable mainly to software systems that are designed to run for an extremely long period of time, such as a server in a client-server application.

Computer users everywhere are familiar with the phenomena of operating system crashes and software freezes. As opposed to a controlled halt and restart of the application or system, these events occur unexpectedly and usually result in a loss of data. The first and most important way to counteract system crashes is to design and build better software. The largest software systems in existence, however, are the most complex entities ever designed by man, and achieving perfection in their design and implementation is unlikely if not impossible. The next line of defense is to take proactive measures to anticipate crashes so that data can be saved and the system can be shut down, cleaned, and restarted in a rejuvenated state. This process has been described as "software rejuvenation."

The causes of software aging are the accumulation of numerical rounding errors, the corruption of data, the exhaustion of operating system resources, unreleased file locks, memory leaks, and other similar minor malfunctions. These errors are due to bugs in the system described by Gray [5] and Huang et al. [25] as *Heisenbugs* because they are non-deterministic and could remain inactive for a long time. (*Bohr bugs*, on the other hand, are deterministic and predictably lead to a software failure). *Heisenbugs* are difficult if not impossible to detect during system testing due to their non-deterministic character. Indeed, the term was first used to describe those bugs in a system that cease to occur whenever debugging or tracing flags are used during software compilation.

Since *Heisenbugs* are by definition extremely difficult to predict directly, we must look for other ways of predicting when a system is in a vulnerable state. Vaidyanathan and Trivedi [21] propose a semi-Markov reward model based on system workload and resource usage to estimate the time of failure of a system. The data they collected from Unix machines is not convincingly modeled through semi-Markov reward processes. The time series tend to fluctuate a great deal and appear chaotic rather than linear. We hope to have better success using fractal tools to analyze the data.

2.3 Other Applications of Fractal Analysis

2.3.1 Fractal Image Compression

Over the last decade the possibility of applying the tools of fractal geometry to the problem of image compression has caused considerable excitement. The basic idea is that any image can be thought of as a fractal and can be generated through the same sort of simple methods used to generate classic fractals such as the Koch curve. The specific tool employed in the field of fractal image compression is known as an iterated function system.

Iterated Function Systems

An iterated function system (IFS) is a mathematical tool for generating fractal objects that relies on simple geometric transformations that are recursively repeated.

A remarkable fact about iterated function systems is that the image produced as the end result of the process may look nothing at all like the original image and, moreover, is entirely independent of the original image. The shape of the object that is the end result of the copying process is dependent only on the instructions given to the machine, and not on the original image that is the seed for the process.

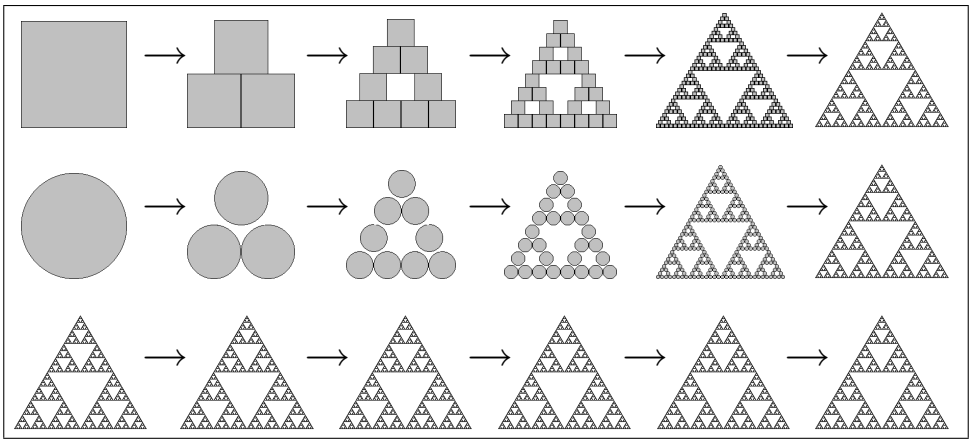

Figure 2.1: The progression toward the attractor of the system, the Sierpinski gasket, regardless of the shape of the initial image.

Whether image used in figure 2.1, for instance, begins with a circle or a square rapidly becomes irrelevant as the image progresses toward the attractor of the system, which is the Sierpinski gasket, a well known fractal object. The final image is called an "attractor" because the process leads directly to it regardless of the initial state.

The instruction set used to form the Sierpinski gasket from an initial image is

	a	b	c	d	e	f
1	0.5	0.0	0.0	0.5	0.0	0.0
2	0.5	0.0	0.0	0.5	0.5	0.0
3	0.5	0.0	0.0	0.5	0.25	0.5

in which the number of rows represents the number of different transformations to make, and each transformation $(x, y) \rightarrow (u, v)$ is of the form $u = ax + by + e$ and $v = cx + dy + f$. In this case the three transformation each reduce the original image by exactly half along both the x-axis

($u = 0.5x + 0y + e$) and exactly half along the y-axis ($v = 0x + 0.5y + f$). The first transformation does not move the image, the second transformation moves the image over 0.5 units along the x-axis ($e = 0.5, f = 0.0$), and the third transformation moves the image 0.25 units along the x-axis and 0.5 units up the y-axis ($e = 0.25, f = 0.5$).

Applications of IFS to Image Compression

A far more interesting example from the perspective of image compression is the fern depicted in figure 2.2. The fractal image that is the end result of the IFS is a breathtakingly realistic approximation of a Blackwort fern, yet the total amount of information needed to encode the image is only 24 numbers:

	a	b	c	d	e	f
1	0.849	0.037	-0.037	0.849	0.075	0.1830
2	0.197	-0.226	0.226	0.197	0.400	0.0490
3	-0.150	0.283	0.260	0.237	0.575	-0.0840
4	0.00	0.000	0.000	0.160	0.500	0.0000

While the IFS instruction matrix for the fern is more complex than the instruction matrix for the Sierpinski gasket, it can nevertheless be stored in only 96 bytes of memory using a 32-bit float for each number.

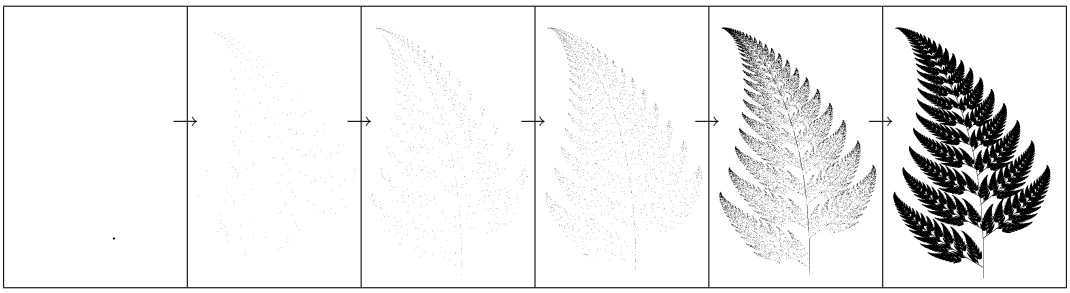

Figure 2.2: Stages along the way to the Blackwort fern.

By increasing the number and complexity of the transformations and adding an additional measure of sophistication, such as partitioning the image into sub-images which may embark on different transformation courses (thereby allowing images that are not strictly self-similar), any possible image can be formed. Compex images may require much initial partitioning, several dozen transformations, and information regarding coloring and 3-dimensional perspective, but it is nevertheless possible to store all the information necessary to form any image in a fraction of the space any other compression system requires.

There is a catch, however. Working backwards from an image to the set of mappings that would form the image under an IFS or a PIFS (a PIFS is an IFS that uses initial partitioning of the image) is a difficult problem which can be very time consuming and has yet to be perfected.

2.3.2 Stock Market

Due to the money at stake, the behavior of stock markets attracts a considerable amount of attention. The ability to understand and predict the trends in stock markets would be, of course, a tremendous advantage to a trader.

For many decades the reigning theory regarding the behavior of stock markets was the Efficient Market Hypothesis (EMH) [20]. According to this hypothesis, investors are rational and intelligent agents who have all possible information at their fingertips and always buy and sell securities according to their true value. The value of a stock only changes when news events, such as a storm or a coup, occur. By definition news events are random and unpredictable, so the behavior of the stock market is accordingly random and unpredictable.

Incorporated within the EMH is the random walk theory of Osborne, developed in 1964, which stated that "because price changes are independent (i.e., they are a random walk), we would expect the distribution of changes to be normal, with a stable mean and finite variance." [17]

The EMH leads to very clean linear models, which is part of its attraction, but unfortunately fails to accurately describe the behavior of stock markets. For one thing, the assumption of the rational investor is a fiction. Rarely do investors have all the relevant information at their fingertips, and even when they do they often do not behave rationally. Instead, investors commonly follow the crowd and trade with trends. This fact may offer an insight into the possible fractal behavior of stock markets: Fractal mathematics, and chaos theory in general, is founded upon the concept of feedback wherein a system's past states influences the direction it will take in the future. According to the EMH fluctuation in the price of a security are independent events, but actual data shows the existence of "bubbles" in the market. A bubble is caused when investors buy a stock at a high price in the expectation that other investors will also follow the trend, thereby driving the price even higher. Until the bubble eventually bursts, buying an overpriced stock is advantageous and selling an overpriced stock is disadvantageous. This feedback loop also occurs in the opposite direction, of course, resulting in stock market crashes such as "Black Monday" in October of 1987.

The Fractal Market Hypothesis (FMH) [20] attempts to explain the behavior of the market through nonlinear models that take into account fractal characteristics such as self-similarity and scale invariance. According to FMH, investors differ greatly amongst themselves. Some, such as day-traders, have short-term agendas, while at the other extreme, others have goals decades in the

future. Accordingly, these investors react differently to the same information. Rather than having a stock market at equilibrium, as the EMH suggests, FMH sees many equilibriums corresponding to many different investment horizons. In addition, rather than focusing on efficiency as the impetus driving investment, the FMH focuses on liquidity. It views the need to easily convert assets into cash as a primary motivation for investors.

While there is little controversy as to whether stock markets actually display fractal characteristics, exploiting these characteristics to enable prediction of the direction the markets will take remains a difficult task. Unexpected news events, such as an assassination, can never be predicted, yet will always have a short-term effect on the markets. The goal of fractal analysis is not to predict the minute to minute behavior of the markets, which would be impossible, but to improve the estimation of trends. A news event cannot be predicted, but perhaps investor behavior in the wake of a news event can be predicted to some extent.

2.3.3 Physiology

One area of nature in which fractal structures are found is living organisms. Human physiology, especially, has been noted for its many fractal characteristics. The branching structure of the circulatory and nervous systems, for instance, display fractal characteristics, and the lungs and intestines and liver make use of fractal like repetetive branching and folding to vastly increase surface area while keeping volume in check.

An interesting discovery by Goldberger et al. [18, 16] is that the human heartbeat displays a fractal structure in a healthy person while people with congestive heart failure often have a highly regular heartbeat. This is not to suggest that gross arrythmias are healthy, but rather that slight variations may aid in some way in warding off disease. Indeed, Goldberger points out that in addition to congestive heart failure, many other pathological conditions such as obsessive compulsive disorder also result in very predictable and rhythmical patterns.

Goldberger [18, 16] found that the time series plot of interbeat intervals was remarkably chaotic even in a healthy individual at rest, and that the plot was invariant over scales ranging from three minutes to several hours. Rather than behaving according to simple fractal patterns, heartbeats tended to display more complex behavior elucidated only through multifractal analysis.

2.3.4 Speech Recognition

Speech signal analysis has long been a difficult task for automated systems. For years, linear approaches have been used to model speech signals, in spite of the fact that speech signals are nonlinear [15]. The main tool for speech signal analysis has been Fourier transforms, which assume that speech

signals are stationary. In fact, speech signals are highly nonstationary, often change dramatically in their frequency and amplitude components [15], and contain many singularities. The existence of many singularities, and the importance of these singularities to the words being expressed, suggests fractal analysis through use of the Hölder exponent [3].

The inherently turbulent nature of speech signals [9, 15] leads to the need for mathematical models that incorporate this phenomenon. Véhel and Dauodi [3, 9] use multifractal analysis and concentrate on "tag points," which is the subset of points in the signal that contain the majority of the acoustic information, and singularities. They then use fractal interpolation based on IFS to model the speech signal. Maragos [15] calculates the short term fractal dimension [14] in order to approximate the amount of turbulence in the speech signal. Using the short term fractal dimension, he calculates a "fractogram" which is a function containing information "about the degree of turbulence inherent in short-time speech sounds at multiple scales" [15]. He finds that incorporating fractograms as features of speech recognitions systems leads to a decrease in the number of recognition errors.

Véhel and Dauodi [9] also use multifractal analysis of speech signals to work toward deriving new "voices" from existing dictionaries of sounds. Their approach is to begin with dictionaries of logatomes of two different voices and then perform interpolations between corresponding logatomes from different voices, all the while ensuring that the signal remains a logatome. Multifractal analysis of the turbulence inherent in speech signals is a tool they use to derive a sturdy functional representation of a logatome.

2.3.5 Network Traffic

In the early to mid 1990s it was discovered that network traffic on the Internet follows fractal behavior. The rate of traffic flow appears similar whether viewed at very large or very small time scales. This result was very counter-intuitive at the time. Previous analysis of voice traffic over telephone systems had indicated that the Poisson process model fit the data very well, and it was expected that network traffic would also fit a Poisson model.

Poisson processes are desirable from a statistical and engineering point of view for a couple of reasons. First of all, they are parsimonious, meaning that that the model uses very few parameters, which makes it elegant and easy to use. Poisson processes also enjoy a rare property among renewal processes in that the superposition of a number of Poisson processes yields another Poisson processes — again enabling ease of analysis. Third, Poisson processes are memoryless, which simplifies queuing models based on them. And finally, behavior that follows a Poisson process is easy to engineer for since one can expect the behavior to smooth out on the average. This would allow network traffic engineers to plan for a constant rate of traffic at large enough time scales.

Unfortunately, the Poisson process is not entirely adequate for modeling network traffic. It has worked well in modeling voice traffic over telephone systems, in addition to modeling many other occurrences, but analysis of network traffic has shown that rather than smoothing out over the long term, network traffic remains bursty even as the time scale increased. This continued burstyness of network traffic led to the mathematics of self-similarity, or fractal analysis, to be applied to network traffic.

A process is said to demonstrate fractal behavior if it is self-similar across many different time scales. Ideally a process would be perfectly self-similar and would be indistinguishable across many different time scales. Practically, however, such perfection is only achieved in mathematical constructions - not in the physical world. Without perfect self-similarity it becomes necessary to measure the degree of self-similarity that a process possesses. This is done through the use of statistics that capture similar properties of two different time series in spite of the fact that they are not identical.

Suppose $X_n, n \geqslant 0$, is a time series where each X_n is a measure of the traffic during successive time units of length $t > 0$. For modeling purposes, it would be convenient if X_n were a stationary process, meaning that its structure is in some sense invariant with respect to shifts in time. Stationary processes come in two forms: strict, and wide sense. Strict stationarity turns out to be too restrictive, so wide sense, which is weaker, bears the brunt of the workload in fractal network traffic analysis.

Park and Willinger [10] define wide sense self similarity as follows: $X(t)$ is exactly wide sense self similar with Hurst parameter $H(\frac{1}{2} < H < 1)$ when the autocovariance, $\gamma(k)$, is given by the equation

$$\gamma(k) = \frac{\sigma^2}{2}((k+1)^{2H} - 2k^{2H} + (k-1)^{2H})$$

for all $k \geqslant 1$. The autocovariance is defined as $\gamma(k) = \mathbf{E}[(X_{n+k} - \mu)(X_n - \mu)]$, where μ is the mean value of the time series X_n. The above expression does not depend on n (is translation invariant), which is guaranteed by the second order stationarity assumption. $X(t)$ is asymptotically wide sense self similar if

$$\lim_{m \to \infty} \gamma^{(m)}(k) = \frac{\sigma^2}{2}((k+1)^{2H} - 2k^{2H} + (k-1)^{2H}).$$

Both exact and asymptotic wide sense self similarity have had wide success in analyzing network traffic.

Self similar processes may or may not be long range dependent, but asymptotic self similarity implies long range dependence due to the restriction of $H(\frac{1}{2} < H < 1)$ — leading some people to interchange the terms.

Park and Willinger [10] define a heavy-tailed distribution as follows: A random variable Z has a heavy tailed distribution if

$$\mathbf{Pr}\{Z > x\} \sim cx^{-\alpha}, \qquad x \to \infty$$

where $0 < \alpha < 2$ is called the tail index or shape parameter and c is a positive constant. This is in contrast to light-tailed distributions such as the exponential distribution and the Gaussian distribution. A random variable following a heavy-tailed distribution is fractal in that it displays extreme variability. It is suspected that heavy-tailed distributions in the size of the files that are transmitted over networks plays a major role in the heavy-tailed behavior of network traffic, which in turn leads to long range dependence and other fractal characteristics.

Chapter 3

Multifractal Signal Analysis: Theory and Application

A fractal function is a function whose graph displays many of the characteristics of a fractal. A function that displays the characteristics of more than one fractal is known as multifractal. The data dealt with in this study is a time series of values for a number of different parameters within an operating system. The extent to which the values of a time series display fractal and multifractal characteristics will be elucidated through calculation of the Hölder exponents at each point and calculation of the degree to which the plot of the function has a fractal dimension greater than one.

3.1 The Hölder Exponent: Theory

Information about the fractality of a function at a local point is obtained through the Hölder exponent of a function at that point, which is a measure of the fluctuation of the function at that point. A highly irregular and chaotic point in a function will have a lower Hölder exponent, and a smoother portion of a function will have higher Hölder exponent. A function may have different Hölder exponents at different points due to a variation in the amount of local fractality that the function displays.

A continuous function $f : \mathcal{R} \rightarrow \mathcal{R}$ is said to be a Hölder function with exponent α if

$$|f(x) - f(y)| \leq c\,|x - y|^{\alpha}$$

for some constant c [4]. Clearly, a differentiable function is always Hölder, with exponent $\alpha = 1$. In general, if this inequality holds for α_0, then it will hold for all $\alpha \leqslant \alpha_0$. The Hölder exponent of a

function, then, is the upper bound of α.

Specifically, a function has a Hölder exponent α at point t_0 if and only if:

i) for every real $\gamma < \alpha$:

$$\lim_{h \to 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = 0$$

and

ii) if $\alpha < +\infty$, for every real $\gamma > \alpha$:

$$\limsup_{h \to 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = +\infty$$

where P is a polynomial whose degree is less than or equal to α [11].

When the Hölder exponent of a function $f(t)$ at a certain point t_0 is greater than or equal to 1, the function $f(t)$ is differentiable, or locally smooth, at t_0. Geometrically, this means that the magnitude of the oscillations of the function near t decreases faster than the the distance to t. The behavior of the function $f(t) = t^{3/2} \sin(1/t)$ at $t_0 = 0$ where we have $\mathcal{H}_f(0) = 1.5$ illustrates the case when $\mathcal{H}_f(t_0) > 1$.

The extreme case is that of a locally constant function: the definition of the Hölder exponent clearly implies that for such a function $\mathcal{H}_f(t_0) = \infty$ (since $\log(0) = -\infty$). Because the constant function behaves similarly to the linear function $f(t) = at + b$, which (for $a \neq 0$) has Hölder exponent 1, we have adopted the convention that $\mathcal{H}_f(t_0) = 1$ when f is constant in a neighborhood of t_0.

In most of our data, however, the Hölder exponent stays within the range $(0, 1)$ which characterizes a non-differentiable, or fractal, function.

3.1.1 Weierstraß Functions

A classic example of a Hölder function is the Weierstraß function, which is a continuous, non-differentiable function of the form

$$W_{a,b}(x) = \sum_{k=0}^{\infty} a^k \cos(2\pi b^k x) \tag{3.1}$$

where $0 < a < 1 < b$ and $ab \geq 1$ [6]. Figure 3.1 contains two examples of a Weierstraß function.

The Weierstraß function is nowhere differentiable, which means that it contains a singularity at every point. In general, a Hölder function may contain singularities of varying strength. The more closely the function approximates smoothness and differentiability, the closer the Hölder exponent at that point will be to 1. The most chaotic sections of the function, however, will be characterized by Hölder exponents approaching 0. The Hölder exponent at any point t, $\mathcal{H}_f(t)$, in the Weierstraß function can be calculated as [11, 6]

$$\mathcal{H}_f(t) = \frac{|\log a|}{\log b} < 1 \tag{3.2}$$

The Weierstraß function has a steady Hölder exponent at all points, meaning that the strength of the singularities is constant at all points.

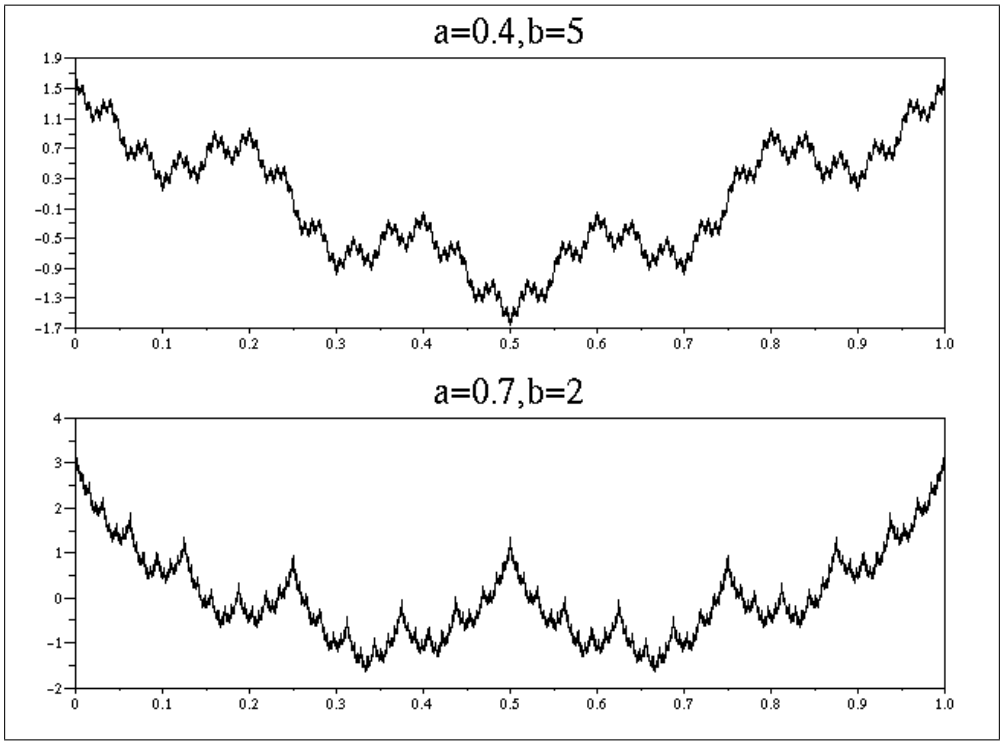

Figure 3.1: The graphs of two different Weierstraß functions. Note the singularities at every point and the self-similar structure of the plots.

A generalized Weierstraß function is a function whose Hölder exponent at each point is given by a seed function whose range is in the interval $[0, 1]$. The formula for this class of functions is

$$f(t) = \sum_{k=0}^{\infty} 3^{-ks(t)} \sin(3^k t),$$

where $s(t)$ is the seed function ranging from 0 to 1 [11]. Figure 3.2 shows graphs of two seed functions and their corresponding generalized Weierstraß functions. An interesting property of genralized Weierstraß functions is that $\mathcal{H}_{W_s}(t) = s(t)$ for all t. That is, the Hölder exponent of a generalized Weierstraß function is specified at every point by the input function $s(t)$.

3.2 The Hölder Exponent: Application

As the previous section has shown, once a Hölder exponent has been given, perhaps in the form of a function ranging over $[0, 1]$, it is a fairly simple and exact matter to generate a function with a

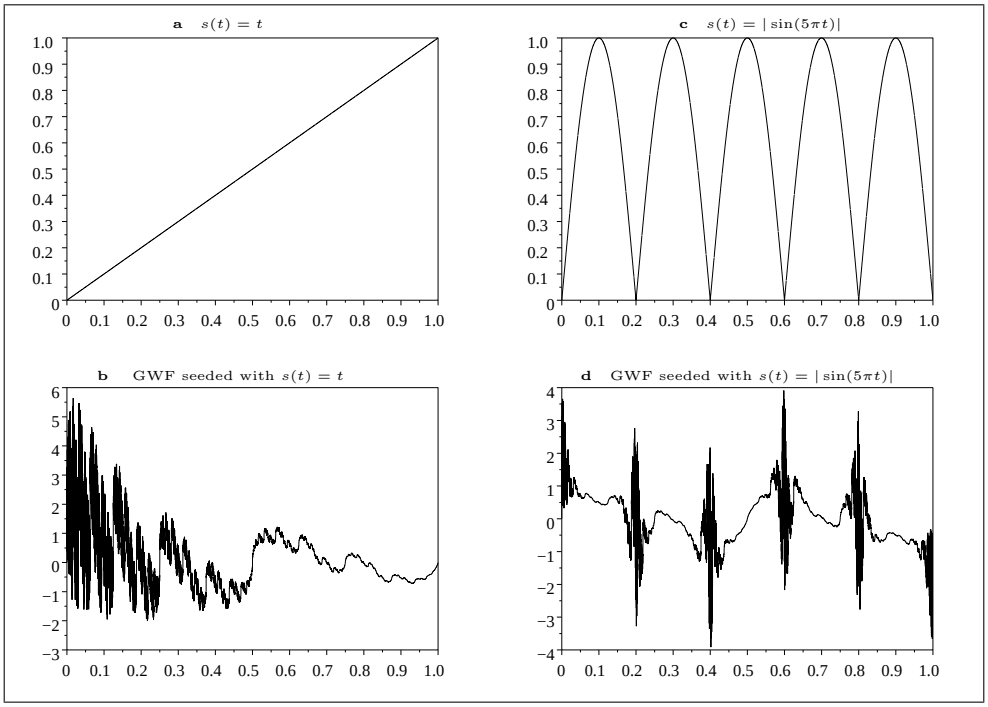

Figure 3.2: **(a)** is the function $s(t) = t$, **(b)** is the generalized Weierstraß function generated using $s(t) = t$ as the seed, **(c)** is function $s(t) = |\sin(5\pi t)|$, and **(d)** is the generalized Weierstraß function generated using $s(t) = |\sin(5\pi t)|$ as the seed.

matching Hölder exponent at each point. Working in the opposite direction is a less exact science, however.

Véhel and Daoudi have shown [23] that the Hölder exponent at a point t of a function $f(t)$ can be calculated with the formula

$$\mathcal{H}_f(t) = \liminf_{h \to 0} \frac{\log(|f(t+h) - f(t)|)}{\log(|h|)}. \tag{3.3}$$

Implementing an algorithm to calculate the Hölder exponent for a time-series of discrete data points (rather than to a continuous function) is part of the goal of this study. In the documentation of the Fraclab package of functions that accompanies Scilab, a Matlab clone, Daoudi has implemented an algorithm for calculating the Hölder exponent of a function. He describes his method as follows [8]:

> The algorithm uses the GIFS method to estimate the Hölder exponent at each point of a given signal. The first step of this method consists in computing the coefficients of the GIFS whose attractor is the given signal. In the second step, we replace each coefficient

> whose absolute value is greater than 1 (resp. smaller than 1/2) by 1 (resp. by 1/2). We then perform the computation of the limit that yields the estimated Hölder function using the chosen type of limit.

The "chosen type of limit" that Daoudi refers to is either a slope limit or a cesaro limit, which can be specified by the user when running his algorithm in the Fraclab package.

Dr. Mark Shereshevsky has developed a competing algorithm, based on formula (3.3), which I have implemented: Suppose $\{y_0, y_1, y_2, ..., y_n\}$ is a time series of n data points measured at uniform intervals. The idea is that the degree to which the function is smooth or chaotic (and therefore the strength of its Hölder exponent) at a certain data point y_i is a function of a number of data points preceding and following it. This number, s, is called the window width and is not fixed but is rather a parameter set by the user when running the algorithm. The weight that each of the $2s$ values (s values on each side of the data point) has on the ultimate calculation of the Hölder exponent is controlled by another parameter, λ. λ is called the weighted regression coefficient and must satisfy $0 < \lambda < 1$. Adjusting the strength of λ will allow us to adjust the degree to which the data points further away from the point we are interested in influence the calculation of the Hölder exponent at that point. Obviously nearby points should have a greater weight than distant points.

To calculate the Hölder exponent at the ith point in a data set, we begin by choosing a window width, s, and a value for λ. We have used $\lambda = 0.5$ and $s = 10$ for all the Hölder exponent calculations appearing in this study.

Next, we calculate a value, $R_{i,k}$, for each integer $k \neq 0$ from $-s$ to s in the following manner:

$$R_{i,k} = \frac{\log |y_{i+k} - y_i|}{\log(\frac{|k|}{n})}$$

In the above case, k must satisfy $0 \leq i + k \leq n$. For data points at the very beginning or end of the time series for which s values to the left or right are not available, s is shrunk to the appropriate size.

Then, in the second step, for each integer j, $1 \leq j \leq k$, calculate

$$h_{i,j} = \min\{R_{i,k} : |k| \leq j\}$$

(This corresponds to taking the $\liminf$ in the equation for the Hölder exponent).

In the third step, calculate $\mathcal{H}_i$, the estimate of the Hölder exponent at the ith data point by computing the weighted average of the approximations $h_{i,j}$:

$$\mathcal{H}_i = \frac{1-\lambda}{1-\lambda^k}(h_{i,1} + \lambda h_{i,2} + \lambda^2 h_{i,3} + ... + \lambda^{k-1} h_{i,k})$$

where more weight is given to terms with a smaller k (i.e., h closer to 0 in equation (3.3)).

I have tested our group's algorithm against Daoudi's algorithm on generalized Weierstraß functions, for which the Hölder exponent is known in advance. As the graphs in figure (3.3) show, the output of our algorithm approaches the graph of the real Hölder exponent much more closely than does the output of Daoudi's algorithm.

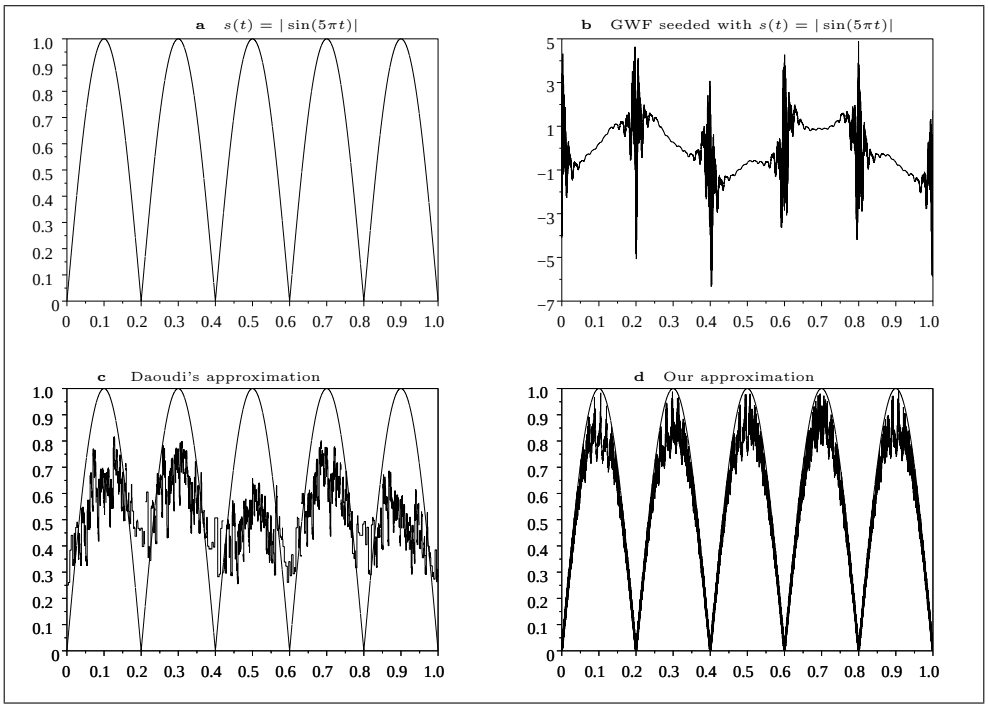

Figure 3.3: **(a)** is the function $s(t) = |\sin(5\pi t)|$, **(b)** is the generalized Weierstraß function generated using $s(t) = |\sin(5\pi t)|$ as the seed, **(c)** is the approximation of the Hölder exponent using Daoudi's fraclab implementation, and **(d)** is the approximation of the Hölder exponent using our implementation.

3.3 Multi-Dimensional Hölder Exponent Analysis

Corresponding changes in the fractality of several different parameters may lead us to consider the behavior of more than one parameter in unison. This can be accomplished through computing a multidimensional Hölder exponent. Data points taken from each of p different parameters observed at the same point in time can be represented by a point (vector) in a p-dimensional space, $\mathcal{R}^p$. The distance, D, between two points, $\mathbf{x} = (x_1, x_2, \ldots x_p)$ and $\mathbf{y} = (y_1, y_2, \ldots y_p)$, in a p-dimensional space

is computed using the formula

$$D = ||\mathbf{x} - \mathbf{y}|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_p - y_p)^2}$$

The definition of the Hölder exponent for a function $f : \mathcal{R} \rightarrow \mathcal{R}^p$ whose values are p-dimensional vectors can be obtained by modifying equation (3.3):

$$\mathcal{H}_f(t) = \liminf_{h \rightarrow 0} \frac{\log(||f(t+h) - f(t)||)}{\log(|h|)}. \tag{3.4}$$

The Hölder exponents computed in this manner can be plotted and studied in the same manner as one-dimensional Hölder exponents.

3.4 Signals Versus Measures

The memory-related data collected for this study are signals. Each data point represents a reading of an internal state in the system at an instant in time, and the values are not additive. The non-additive nature of the data simply means that summing the values of the data at two successive points is not a sensible operation. This is as opposed to measures, which are a function of an interval of time and which are additive. For example, the number of system calls made during a certain period of time, which is one of the workload parameters collected for this study, is a measure. If we take the number of system calls made from time 0 to time 1 and add it to the number of system calls made from time 1 to time 2, we will have the number of system calls made from time 0 to time 2. Adding all the values for the system call data series will give us the total number of system calls made during that interval of time. Adding all the values in the memory-related time series, which are signals, however, will give a nonsensical result.

Measures can be thought of as counting processes in which the value of the measure for a particular time interval is the total number of events occuring during that time interval. No events at all can occur at a single instant in time, however, since events require a duration of time to occur in. Converting a measure to a signal may, however, be desireable for certain types of analysis. This is accomplished through computing the density of the measure, $f_\mu(t)$, using the formula

$$f_\mu(t) = \lim_{\varepsilon \rightarrow 0} \frac{\mu[t - \varepsilon, t + \varepsilon]}{2\varepsilon} \tag{3.5}$$

in which ε represents an interval of time.

Unfortunately, the formula above cannot be applied to measures which demonstrate fractal characteristics. This is because fractal measures by definition consist of chaotic fluctuations between periods of very high activity and periods of very sparse activity. There may be bursts of incredible

density followed closely by periods of almost total nonactivity. This nonuniformity renders the above formula inapplicable for computing the density of fractal measures, since the limit in equation (3.5) often does not exist.

The formula for computing Hölder exponents given in equation (3.3) assumes that the data being analyzed is a signal, so this formula cannot be applied to measures. There is, however, a parallel formula for computing the Hölder exponent of a measure:

$$\mathcal{H}_{\mu(t)} = \liminf_{\varepsilon \to 0} \frac{\log(\mu([t, t+\varepsilon]))}{\log(\varepsilon)}. \tag{3.6}$$

Notice that if the limit in equation (3.5) exists, i.e., μ has a density at t, then its Hölder exponent is 1.

Fractal measure analysis is beyond the purview of this study, however. While some of the data we have collected is in the form of measures, the granularity of our data collection process is too coarse to allow the application of fractal measure analysis. Fractal network traffic analysis, discussed in the previous chapter, uses measures almost exclusively, but the data collection process employed in those studies was fine-grained enough to take data readings mere millionths of a second apart. Our data points are a million times coarser in granularity and would not yield accurate enough results.

3.5 Fractal Dimension: Theory

As was noted in chapter 1, fractal objects have a dimension that usually falls between two whole numbers. One way to think about the fractality of a function is in terms of the amount of space its plot takes up in a plane. A smooth one-dimensional function can be thought of as taking up an infinitely small fraction of the space in a two-dimensional plane. Fractal functions are not so simple, however. They may fluctuate wildly and twist in on themselves in such a way as to "fill out" space within a two-dimensional plane.

If each side of the square on the plane were divided into n units, we could think of the plane as consisting of n^2 sub-squares — corresponding to the grid formed by the n divisions on each side. The fractal dimension of a geometric figure, E, in the square can be thought of as the limit as n goes to ∞ of the logarithmic ratio of sub-squares that the function passes through to the total number of sub-squares. If we denote by K_n the number of these sub-squares that intersect E, we can calculate the fractal dimension, FD, of E as

$$FD(E) = \lim_{n \to \infty} \frac{\log K_n}{\log n}$$

In this study, when we talk about the fractal dimension of a function f, what is actually meant is the fractal dimension of its plot.

In the example of a perfectly smooth function such as $f(x) = x$, the graph will pass through exactly n of the n^2 sub-planes, yielding a dimension of

$$FD(f) = \frac{\log n}{\log n} = 1$$

which is just what we would expect. On the other hand, (multi-)fractal functions fluctuate so much from point to point that their fractal dimension is greater than 1. It is obvious, however, that for any function $f : \mathcal{R} \to \mathcal{R}$, $1 \leqslant FD(f) \leqslant 2$.

It has been shown by Hunt [7] that for fractal functions that display a single degree of fractality (as opposed to multifractality), the difference between 2 and the fractal dimension of the function is equal to the Hölder exponent of the function:

$$FD(f) = 2 - \mathcal{H}_f \tag{3.7}$$

3.6 Fractal Dimension: Application

In this section we explain a method for calculating the fractal dimension of a time series and show that our method yields accurate results when employed on a Weierstraß function whose fractal dimension is known (via its Hölder exponent which is given).

First, we choose the number of points in the time series to be a power of two. Then, in order to fit the plot of our time series into the $[0, 1] \times [0, 1]$ square, we normalize the time series using the formula

$$f(x) = \frac{f(x) - \min}{\max - \min}$$

where max and min are the maximum and minimum values of the points in the time series for which we are computing the dimension. This ensures that the smallest point in the series falls at 0 and the largest point falls at 1. This constricts the calculation of the fractal dimension to the area of the plane that that time series actually occupies. If this method of normalization were not used, the calculation of the fractal dimension of a time series in which the values fell in the range ten million to eleven million would take into account all the white space from 0 to ten million - which would clearly skew the result.

Once we have chosen the number of points in the time series to be a power of two, we divide the $[0, 1] \times [0, 1]$ square up into n^2 sub-squares, where n is a smaller power of two. We then count the number of sub-squares, K_n, that the lines connecting the time series values pass through. This is accomplished through iterating through every range $n_i \to n_{i+1}, i = 0, 1, ..., n$ in the time series and counting every subplane from the one containing the minimum value of the time series in that range through the one containing the maximum value of the time series in the same range. We then

continually repeat this process, each time doubling the value of n until n is equal to the number of points in the time series. Using linear regression we calculate the slope of the best-fit line on the points $\log_2(K_n)$ versus $\log_2(n)$ for all the values of n that we have used.

3.6.1 Fractal Dimensions of Weierstraß Functions

A good test of our method of computing the fractal dimension of a function is to compute the fractal dimension of a fractal function for which the result in known ahead of time.

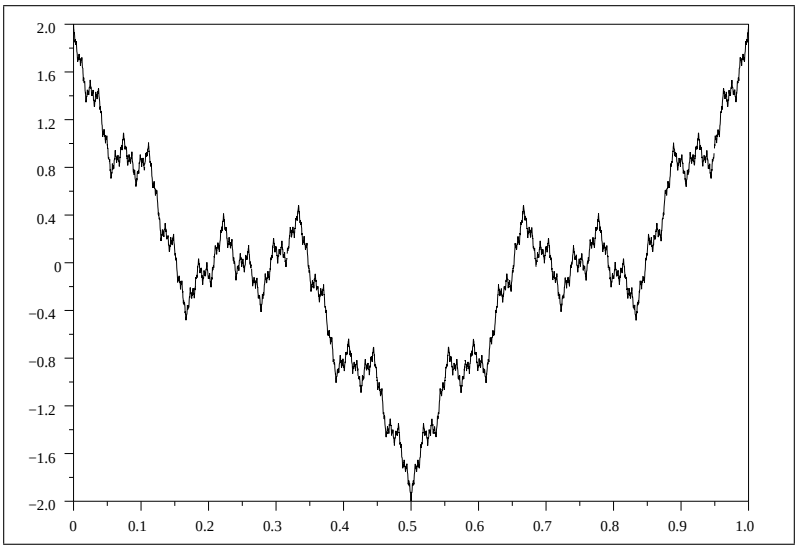

Figure 3.4: Weierstraß function with $a = 0.5$ and $b = 3$.

A Weierstraß function is such a function. Figure (3.4) is a Weierstraß function generated using the formula in equation (3.1) with $a = 0.5$ and $b = 3$. By equation 3.2 we can calculate the Hölder exponent of this function as

$$\mathcal{H} = \frac{|\log(0.5)|}{\log(3)} = 0.631$$

Using equation (3.7), we can now infer that the fractal dimension of this Weierstraß function is

$$2 - 0.631 = 1.369$$

Our box-counting method for calculating th fractal dimension of this funtion gave a result of 1.32, which, while not perfect, is off by only five percent.

Chapter 4

Data and Results

4.1 The Data Collection

The data used in this study was collected from a machine called Naur, a server in the computer science and electrical engineering department at West Virginia University. Naur has 256 MB of RAM, dual 333 MHz processors, runs SunOS 5.5.1, and is one of the most heavily used servers in the department. The data was collected once per second for eight days (which is 691,200 seconds) from September 15 through September 22, 2001, using the sar utility. "sar" stands for "system activity reporter."

The following six memory parameters were used in this study:

1. **sml_mem** - the amount of memory the kernal memory allocator has reserved for small requests.
2. **sml_alloc** - the amount of memory allocated to satisfy small requests.
3. **lg_mem** - the amount of memory the kernal memory allocator has reserved for large requests.
4. **lg_alloc** - the amount of memory allocated to satisfy large requests.
5. **freemem** - average pages available to user processes.
6. **freeswap** - disk blocks available for page swapping.

In addition, the following three parameters were collected to monitor the workload of the system:

1. **scall/s** - system calls of all types.
2. **cpu** - the percentage of time that the CPU is in use.
3. **pgin/s** - page-in requests per second.

4.1.1 Normalization of the Data

All the memory-related data we collected was normalized to fit within the range from 0 to 1. This was done in order to better be able to compare the different memory pararameters with each other. The actual values of the different parameters differed by several orders of magnitude, which would have made meaningful comparison difficult without normalization. For instance, the time series for the freemem parameter contained values as low as 123, while the time series for freeswap contained values as high as 1,500,000, and the time series for lg_mem contained values as high as 14,000,000.

In addition, the fluctuations within each time series tended to be on different orders of magnitude. A fluctuation of 1,000 was a relatively large move for the freemem parameter, while a fluctuation of 10,000 was a relatively small move for the lg_alloc parameter, and jumps of over one million were not uncommon.

By taking the maximum and minimum values over all the data for each parameter and then adjusting the data according to the formula

$$f(x) = \frac{f(x) - \min}{\max - \min}$$

we were able to deal with each parameter within the convenient range of 0 to 1. This adjustment to the data preserves the shape of the graph and the fractality (or lack thereof) of the data. In addition, it is necessary for the computation of the fractal dimension of the data.

4.2 Results

4.2.1 Behavior of the Memory Parameters

Analysis of the data showed that the six memory parameters we studied displayed markedly different fractal behavior. The sml_mem and lg_mem parameters, representing the amount of memory set aside for small and large memory request respectively, were relatively smooth in their behavior. While these parameters displayed marked jumps from time to time, they tended to immediately stabilize after a jump and remain stable for at least a few seconds and at most several minutes. Basically, these two memory parameters did not fluctuate a great deal. Figure 4.1 shows the graphs of the sml_mem and lg_mem parameters for the time period from midnight to 2:20 AM on September 16.

The sml_alloc and lg_alloc parameters displayed behavior that was similar to the sml_mem and lg_mem parameters respectively in terms of the broad movements of the data, but which differed with respect to the degree of fluctuation.

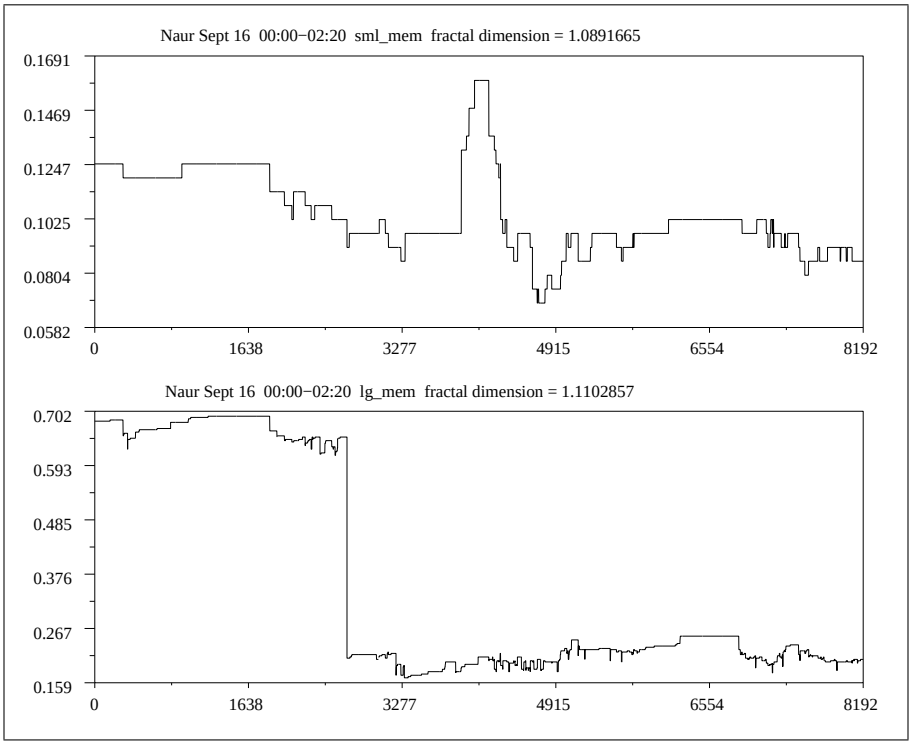

Figure 4.1: sml_mem and lg_mem from midnight to 2:20 AM on September 16.

Intuitively this makes sense. Whenever a large increase or decrease occurs in the sml_mem or lg_mem parameter, the bounds of the sml_alloc and lg_alloc parameters are immediately effected. Since the operating system cannot allocate more memory for small or large requests than it has available in the reserved pool for those requests, the amount of memory allocated cannot exceed the amount of memory reserved for allocation. If the operating system is forced to reduce the amount of memory reserved for small requests to a level below the amount allocated, it will also be forced to reduce the amount of memory allocated. Alternatively, if there are a large number of small requests and the operating system is not being taxed in other departments, it may increase the amount of memory reserved for small requests.

The broad trends of the sml_alloc and lg_alloc parameters, then, are constrained to follow the broad trends of the sml_mem and lg_mem parameters. On a detailed level from moment to moment, however, there is little necessary correspondence between the amount of memory reserved for requests of a certain size and the actual amount of memory allocated for requests of that size. Processes are continually starting and stopping and requesting or releasing handfulls of memory. This constant give and take will result in a high degree of minor fluctuations of the allocation parameters. Figure 4.2 shows the graphs of the sml_alloc and lg_alloc parameters for the time period from midnight to

2:20 AM on September 16. While these graphs disply much more chaotic behavior from moment to moment, there is an obvious correspondence to the graphs of figure 4.1 in the overall shape. Keep in mind that due to the normalization of the data, the values along the y-axis do not correspond directly to the actual values of the data. It is possible for the magnitude of a normalized allocation parameter to exceed the magnitude of a normalized reservation parameter without corresponding behavior occuring in the non-normalized values of the data.

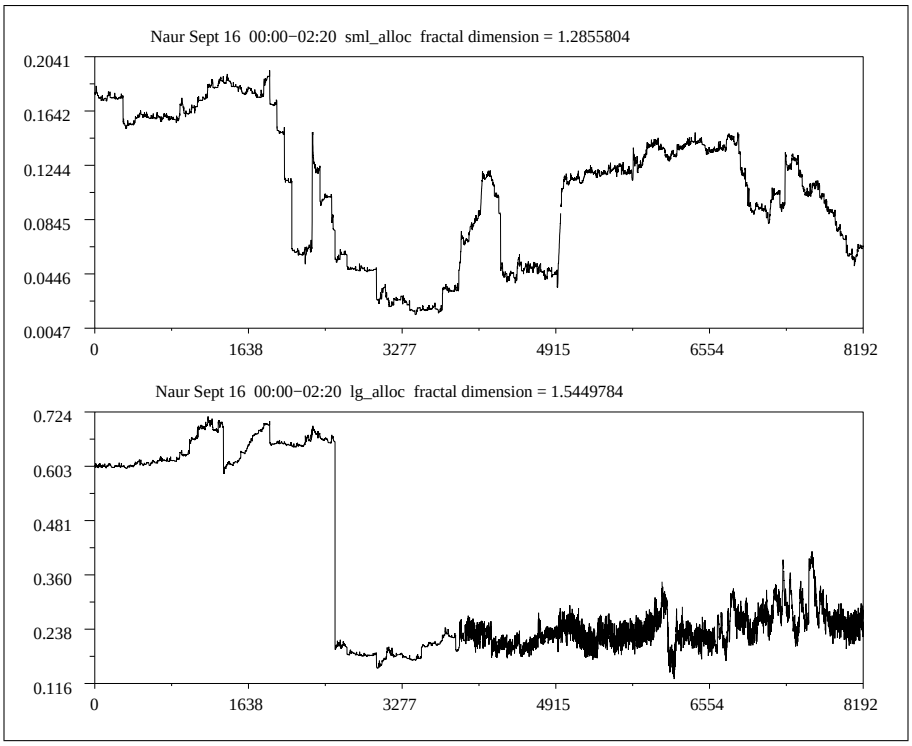

Figure 4.2: sml_mem and lg_mem from midnight to 2:20 AM on September 16.

The freemem and freeswap parameters both displayed a great deal of chaotic fluctuation. The freemem parameter tended to be highly unpredictable, while the freeswap parameter consistently alternated between quiet stages of little activity and chaotic stages distinguished by rapid fluctuations of approximately even magnitude. In addition, the freeswap parameter was occasionally punctuated by a very short-lived but large drop in its value. Figure 4.3 shows a representative graph of both freemem and freeswap also taken from the midnight to 2:20 AM time period on September 16.

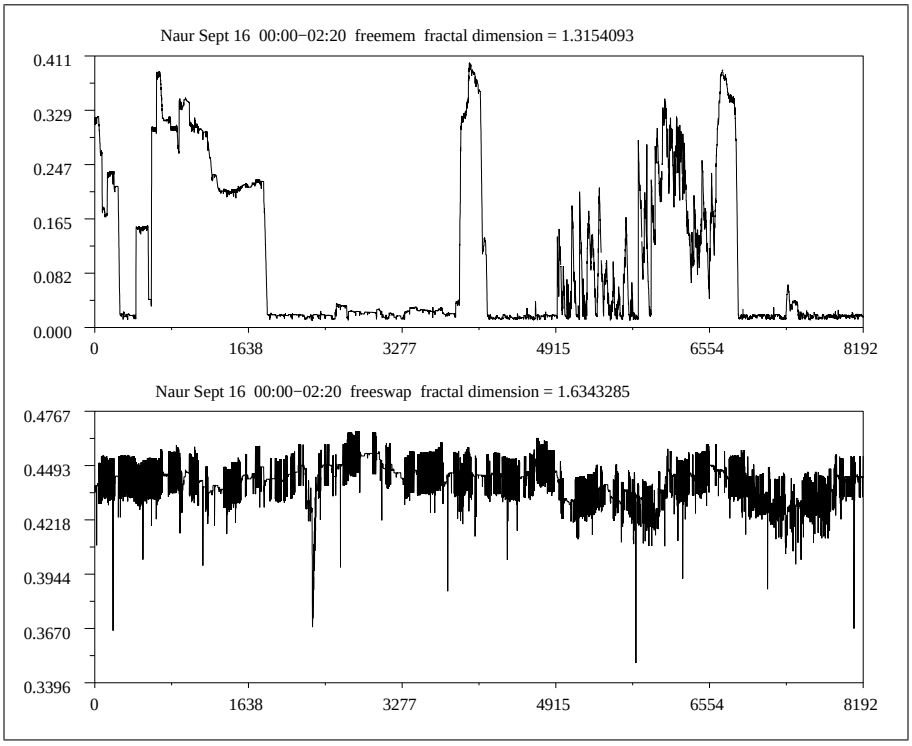

Figure 4.3: freemem and freeswap from midnight to 2:20 AM on September 16.

4.2.2 Behavior of the Hölder Exponent

We initially assumed that our calculations of the Hölder exponent would range between 0 and 1. Acting on this assumption, we assigned the value 1 to the Hölder exponent whenever there was no change in the value of a string of points in the time series. We did this because our formula for calculating the Hölder exponent requires us to take the log of the absolute value of the difference between two points (see equation (3.3)), and when this difference is 0, the logarithm is undefined. We found, however, that our Hölder exponents occasionally exceeded one. See section 3.1 for a theoretical explanation.

We found that for those parameters that displayed relatively smooth behavior, the Hölder exponent tended to accumulate at 1 most of the time, with sharp spikes downward whenever a move occured in the data. Figure 4.4 compares the graphs of the sml_mem and lg_mem parameters to the graphs of their Hölder exponents for the midnight to 2:20 AM time period on September 16. The histograms of the Hölder exponents clearly indicate where the bulk of the Hölder exponents occur. In these cases, the Hölder exponent is so heavily weighted at 1 that the parameters can be said to exhibit no fractal behavior at all.

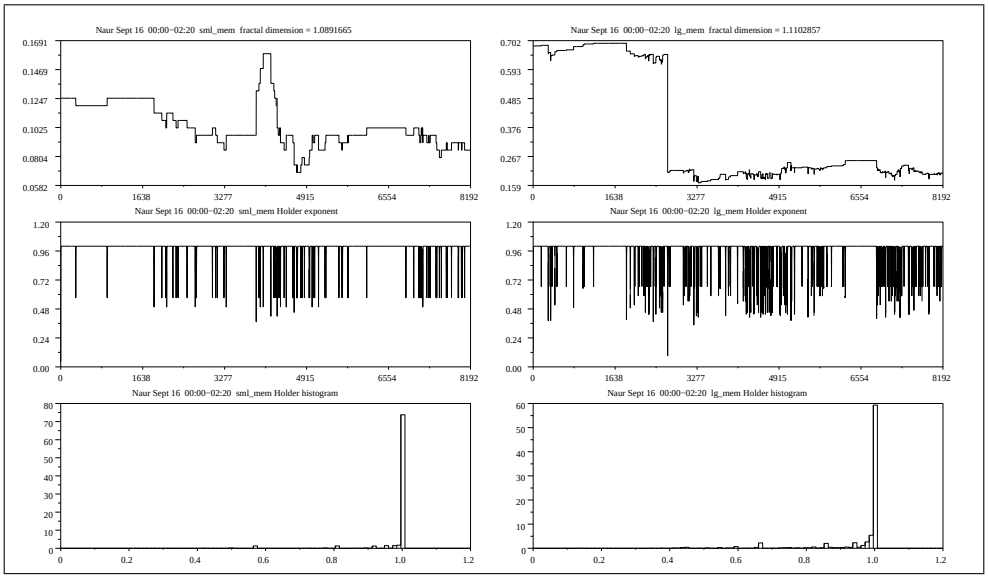

Figure 4.4: sml_mem and lg_mem with their Hölder exponents from midnight to 2:20 AM on September 16.

The Hölder exponents for the less stable parameters tended to fluctuate a great deal more than the Hölder exponents for the stable parameters and also displayed sensitivity toward the changes in the amount of fluctuation of the parameter in question. Figure 4.5 compares the graphs of the freemem and freeswap parameters to their Hölder exponents for the midnight to 2:20 AM time period on September 16.

The freemem parameter displays a period of burstiness between approximately the 4,900th second and the 6,500th second. The Hölder exponent for this period shows a corresponding decrease, indicating a rise in fractality. The histogram of the Hölder exponent shows four dominant levels of fractality — one at approximately 0.52, another at approximately 0.67, a third just above 0.8, and a fourth at 1. These four different levels of fractality correspond to the upper and lower ranges of each of the two distinct sections in the plot of the Hölder exponent. Although the distinction between these various levels of fractality is not very pronounced, and the accumulation of Hölder exponents at 1 indicates a level of non-fractality rather than fractality, it is nevertheless possible to describe this behavior as multifractal.

Far more pronounced multifractal behavior is evident in the freeswap parameter, however. From the plot of the data it is evident that a repetitive pattern of smoothness followed by burstiness is the manner in which this parameter operates (the plot in figure 4.5 is representative of all the plots for this parameter in this respect.) The corresponding Hölder exponent closely follows the plot of the data,

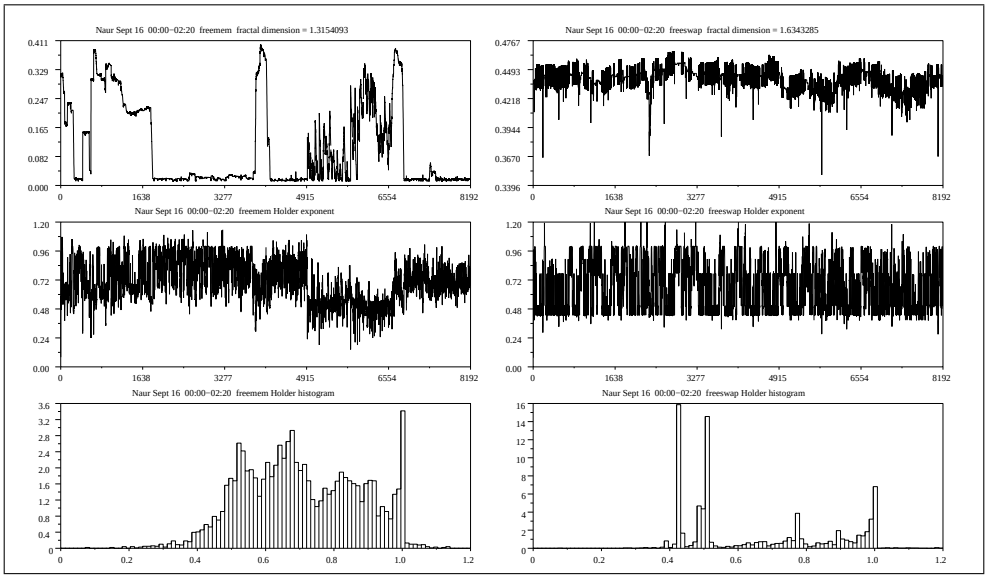

Figure 4.5: freemem and freeswap with their Hölder exponents from midnight to 2:20 AM on September 16.

rising when the data is smooth and falling when the data begins to fluctuate. Rather than exhibiting chaotic behavior or wild fluctuations, however, the freeswap parameter fluctuates in a controlled manner, causing the Hölder exponent to level off at certain plateaus. There are definite bands discernible in the plot of the Hölder exponent. The histogram of the Hölder exponent illustrates the band-like behavior of the Hölder exponent by showing demonstrating a strong proclivity of the data toward fractal behavior at Hölder exponents of approximately 0.42, 0.51, 0.78, and 1.

Figure 4.6 compares the graphs of the sml_alloc and lg_alloc parameters to their Hölder exponents for the midnight to 2:20 AM time period on September 16. The amount of fluctuation of the sml_alloc parameter is relatively constant throughout the time period, leading to a Hölder exponent whose behavior, while chaotic, is at least chaotic in a relatively consistent fashion throughout the time period. The discerning eye may notice, however, a very slight downward shift in the Hölder exponent at approximately the mid-point of the series. This minor shift in fractality is practically invisible in the plot of the data, is only revealed after careful search in the Hölder exponent, but is quite evident in the histogram of the Hölder exponent. The two humps in the histogram indicate two distinct accumulations of the Hölder exponent. One at approximately 0.9,and the other at approximately 0.7. Both these Hölder exponents are high, however, so, although there is multifractality, the degree of fractality in each case is only slight.

The plot for the lg_alloc parameter in figure 4.6 is more interesting, however. An abrupt change

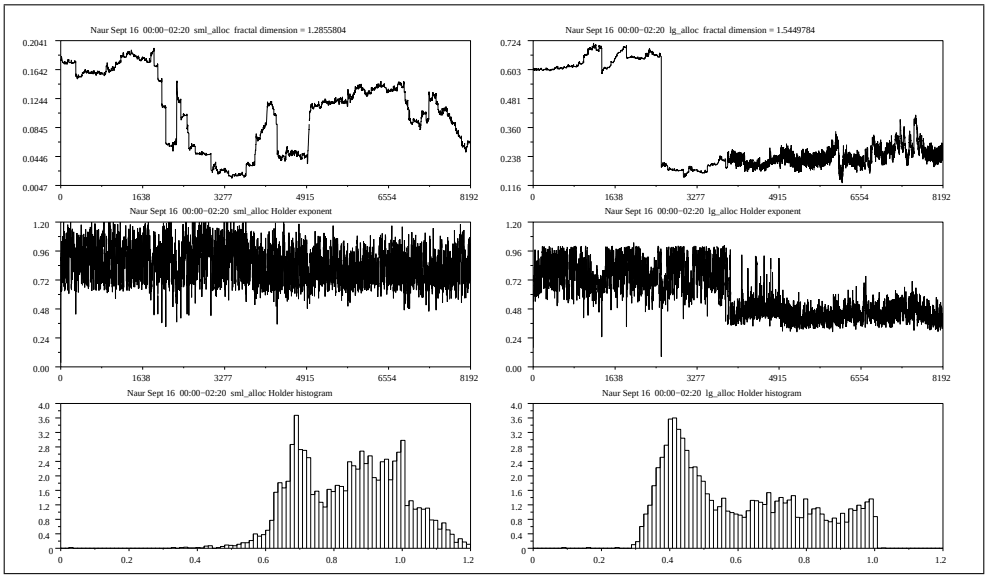

Figure 4.6: sml_alloc and lg_alloc with their Hölder exponents from midnight to 2:20 AM on September 16.

in fractality occurs at approximately the midpoint in the series, and the change is easily noticed in the plot of the data, the plot of the Hölder exponent, and the histogram of the Hölder exponent. As soon as the data begins fluctuating wildly, the Hölder exponent drops dramatically. The histogram captures this behavior through a pronounced hump at approximately 0.4, and a large accumulation at 0.6 and above, corresponding to the second half and the first half of the time series respectively.

The abrupt change in the fractal behavior of the lg_alloc parameter can be more easily appreciated if we split the time series up into two sections at the point of the change and analyze them each seperately. The left side of figure 4.7 depicts the lg_alloc parameter with its Hölder exponent from 23:50 PM on September 15 until 01:00 AM on September 16. This corresponds to the time just prior to the change in behavior. The right side of figure 4.7 depicts the lg_alloc parameter with its Hölder exponent from 01:10 AM until 02:20 AM on September 16, which is the time just after the change in behavior.

The data before and after the point of the change hardly seem to be measures of the same parameter, the Hölder exponent is appreciably different, and the histogram shows a pronounces lunge to the left. In addition, the measure of the fractal dimension shows that before the change, the fractal dimension of the lg_alloc parameter was only 1.25, while after the change the fractal dimension leapt up to 1.63. One may wonder what event occurs between 01:00 and 01:10 AM to cause this marked change in behavior. It turns out that this change is related to the nature of the

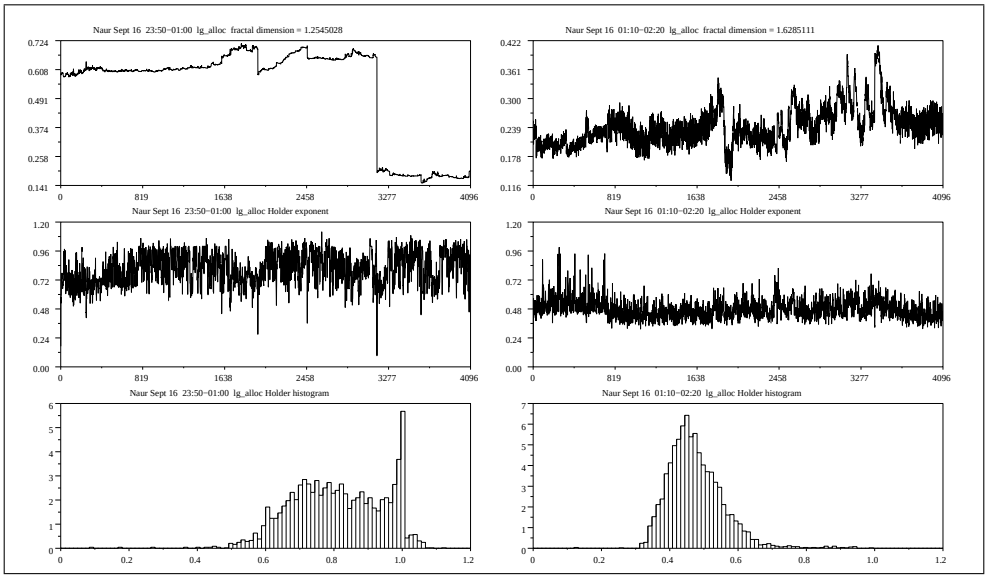

Figure 4.7: lg_alloc with its Hölder exponent before and after the change in fractality.

workload on Naur at that time.

4.2.3 Fractality in Relation to Workload

At precisely 01:05 every morning, a machine named Spawn initiates a backup procedure with the Naur. This procedure requires the transfer of large amounts of data, resulting in a large number of requests for large segments of memory. The memory is only kept for the short duration of the time taken to transfer the data before it is returned to the pool and another request is made to expedite the transfer of another chunk of data. This continual back and forth causes the chaotic behavior seen in the plot of the amount of memory allocated to fulfill large requests. There is a very strong correlation between the fractality of the lg_alloc memory resource and the workload event of system backup.

Interestingly, while there is a definite correlation between the type of work the machine is undertaking and the fractality of the lg_alloc parameter, there is not a strong correlation between fractality and traditional measures of system workload such as the percentage of CPU usage. In fact, plotting the lg_alloc parameter for the midnight to 2:20 AM time period beside plots of such measures of workload as CPU usage, the number of system calls, and the amount of paging activity going on, as has been done in figure 4.8 shows that only the amount of paging activity, which is directly related to memory use, shows a strong relation to the lg_alloc parameter.

In fact, the obvious change in workload from approximately the 1650th second until the 3000th

second, characterised by a rise in the amount of paging, a marked drop in the number of system calls, and a rise in the percentage of CPU use, is not reflected at all in the fractality of the lg_alloc parameter (or indeed in the fractality of any other parameter). The close relationship of paging to memory use may lead us to wonder why a change in the paging behavior is not reflected in the fractality of the lg_alloc parameter during the first instance of change visible in the plot, while it is well reflected in the second change. This is because the rise in paging behavior during the first segment is an even rise that is not characterized by much fluctuation. Although the average amount of paging rises during the first segment, there are actually fewer high spikes in the plot, indicating less fluctuation. This sustained, even behavior is not likely to be reflected in a rise in the fractality of the correspondin memory parameters.

The second instance of change, however, is not only a rise in the amount of paging, but a definite increase in the fluctuation of the paging. Unlike the first segment which sustained the paging activity at between approximately 30 and 50 for over five minutes, the second segment of change is characterized by fluctuations of over 100 from second to second, with occasional bursts rising to over 400.

The most we can say about the relation of fractal memory patterns to system workload, then, is that it is ambiguous. Some workload shifts are accompanied by corresponding shifts in the fractal behavior of the memory usage patterns, and some are not. The important consideration seems to be not the amount of work the system is doing, but the type of work. Very large processes such as system backup are likely to markedly affect the fractality of memory usage patterns, but more mundane shifts in workload may have little or no correspondence with the fractality of the memory usage patterns. At this stage, however, we are not sure which workloads play a decisive role in determining the multifractal behavior of the memory usage patterns. We hope for more insight after further research.

4.2.4 Multidimensional Fractality

Corresponding changes in the fractality of several different parameters, such as is evident in the lg_alloc and paging parameters may lead us to consider the behavior of more than one parameter in unison. Since the paging parameter is a measure rather than a signal, we cannot apply our methods of analysis to it. We can, however, apply multidimensional analysis to the six memory parameters that we are concerned with. Figure 4.9 shows a plot of the multidimensional Hölder exponent computed using the formula from equation 3.4 for the midnight to 2:20 AM period on September 16 and combining all six memory parameters that have previously only been considered separately.

In this case, the multidimensional Hölder exponent closely resembles the one-dimensional Hölder exponent for the the lg_alloc parameter. This suggests that the lg_alloc parameter dominates the multidimensional Hölder exponent. This is due, no doubt, to the pronounced fractality of that parameter as well as the pronounced change in behavior visible in that parameter. The other parameters do not contribute much illumination to the multidimensional Hölder exponent because they reflect either little fractality or else sustained periods of constant fractality. In this particular case, the fractal behavior of the operating system resource use is better captured through consideration of each parameter individually.

Under different circumstances and using different parameters, however, multidimensional Hölder exponent analysis may prove to be more enlightening. In a scenario in which a number of different parameters simultandously displayed small amounts of fractality, as opposed to this situation in which only one parameter launches into markedly fractal behavior, multidimensional Hölder exponent analysis would serve to magnify the fractal behavior rather than blur it.

4.2.5 Fractal Dimension

In the previous chapter we showed that our calculation of the fractal dimension of a time series corresponds well to the expected fractal dimension based upon the Hölder exponent. The correspondence between the fractal dimension of function and the Hölder exponent of a function is given in equation (**??**). This result is proven to hold for Weierstraß functions, which have a single Hölder exponent throughout the function [7]. Our analysis of memory usage patterns has shown that these patterns are, by and large, multifractal. The histograms of the Hölder exponents of the memory parameters tend to be spread out over a broad range rather than concentrated at a single point. Often, however, a data series will have a single dominant fractal level within its overall multifractal behavior. In these cases, we found that our calculation of the fractal dimension corresponded well with the dominant Hölder exponent. Figures 4.10 and 4.11 show eight different plots of time periods for which a memory parameter was found to have a dominant Hölder exponent. The results are summarized in the following table:

	a	b	c	d	e	f	g	h
Dominant Hölder exponent	0.63	0.69	0.72	0.45	0.62	0.68	0.67	0.68
Fractal Dimension (FD)	1.35	1.31	1.32	1.63	1.36	1.36	1.33	1.30
2− FD	0.65	0.69	0.68	0.37	0.64	0.64	0.67	0.70
$\lvert(2 - FD) - \mathcal{H}\rvert$	0.02	0.00	0.04	0.08	0.02	0.04	0.00	0.02

These results show that when a signal behaves in a multifractal manner but with a strong disposition toward one plateau of fractality, the two independent methods of determining the fractality of the signal that were employed in this study yield surprisingly consistent results. This is a strong indication that there is indeed fractal behavior occuring in at least the freemem, lg_alloc, and sml_alloc parameters, since these three parameters were most likely to yield accurate measures of their fractality through the two different methods employed.

The fractal dimensions calculated for the sml_mem and lg_mem parameters were also consistent with the Hölder exponents computed for those parameters, but since the Hölder exponent was strongly bent toward 1 and the fractal dimension was usually also around 1, these results are not as interesting for the purposes of demonstrating fractal behavior. Instead, these parameters could be said to have been independently shown to be non-fractal.

The freeswap parameter displayed the most pronounced multifractal behavior. In every histogram of the Hölder exponent, such as the one in figure 4.5, two very pronounced spikes are visible. This strong multifractal behavior thwarts are attempts to independently confirm our results through computation of the fractal dimension.

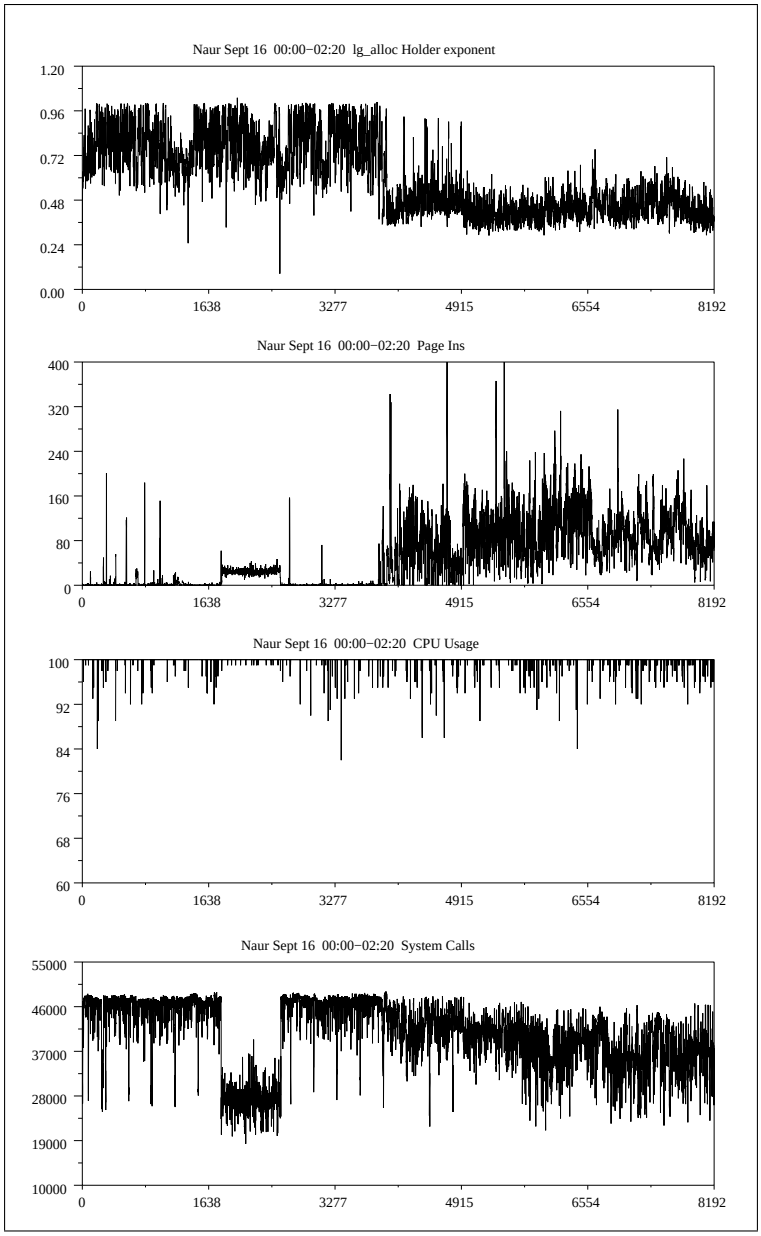

Figure 4.8: lg_alloc with its Hölder exponent compared to three different workload parameters.

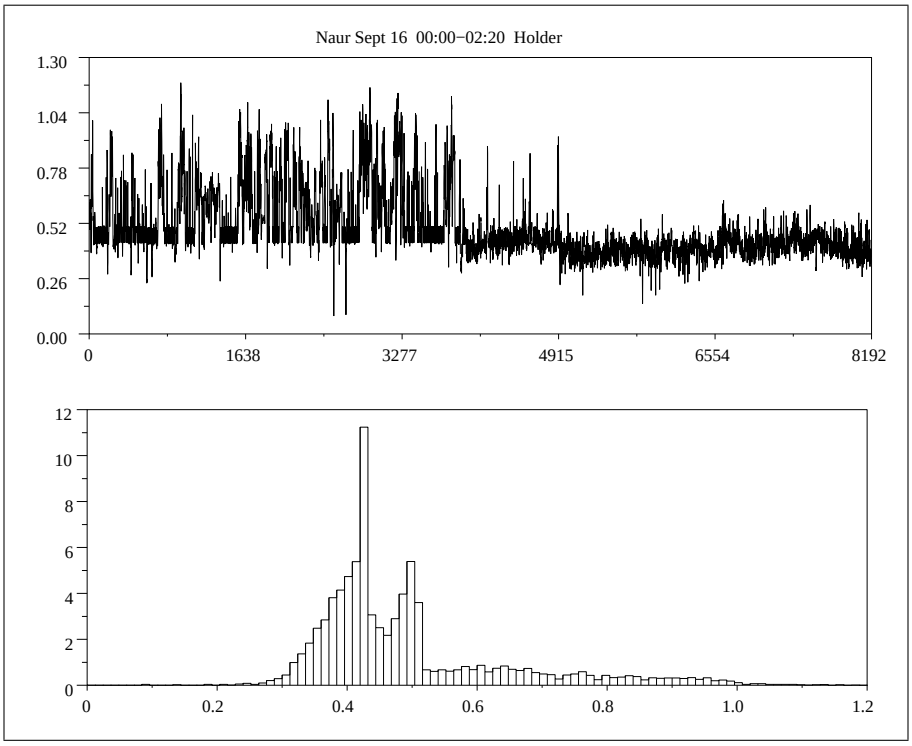

Figure 4.9: Multidimensional Hölder exponent from midnight to 2:20 AM on September 16.

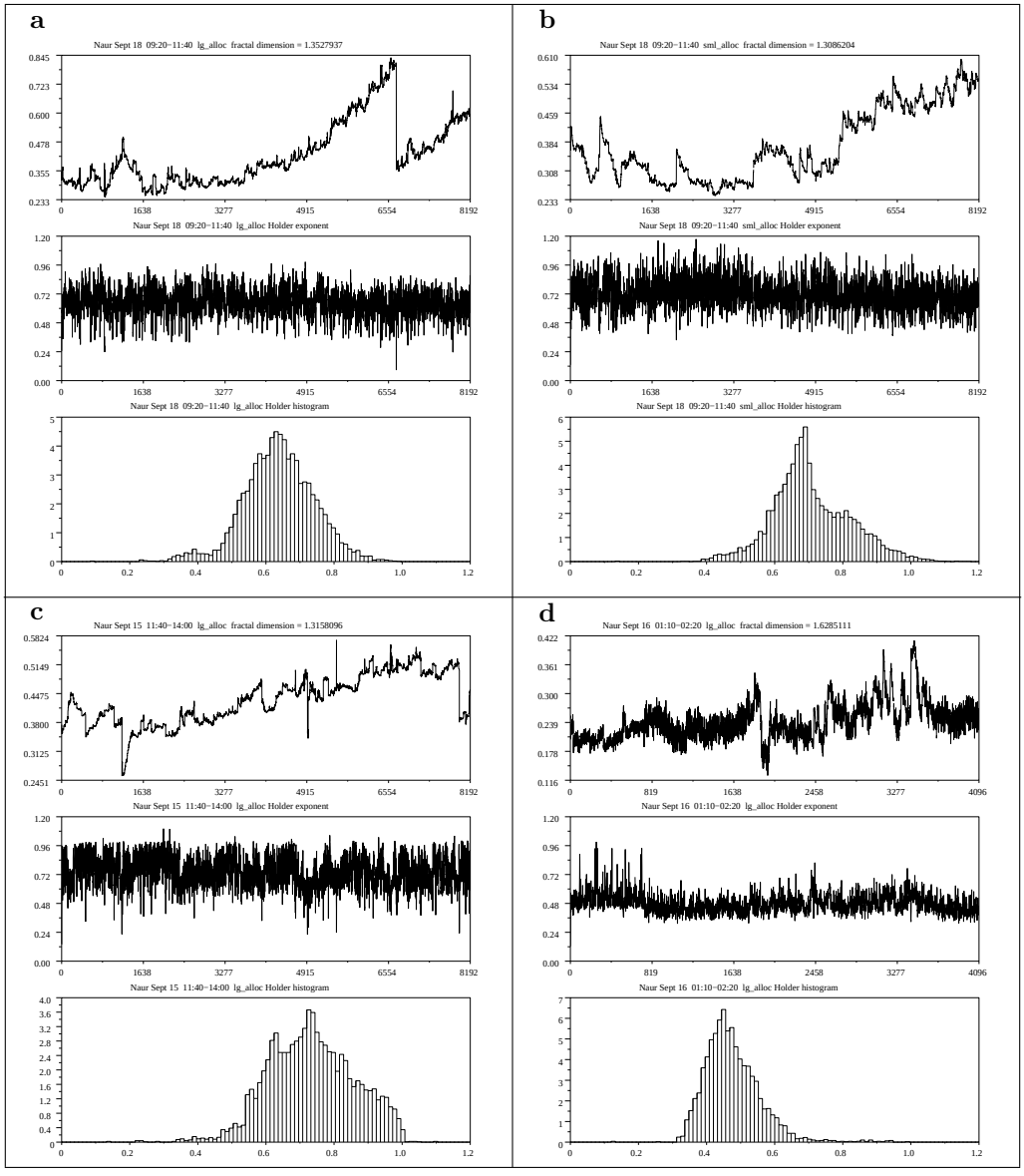

Figure 4.10: Comparison of Hölder exponent with fractal dimension for sml_alloc and lg_alloc at different times.

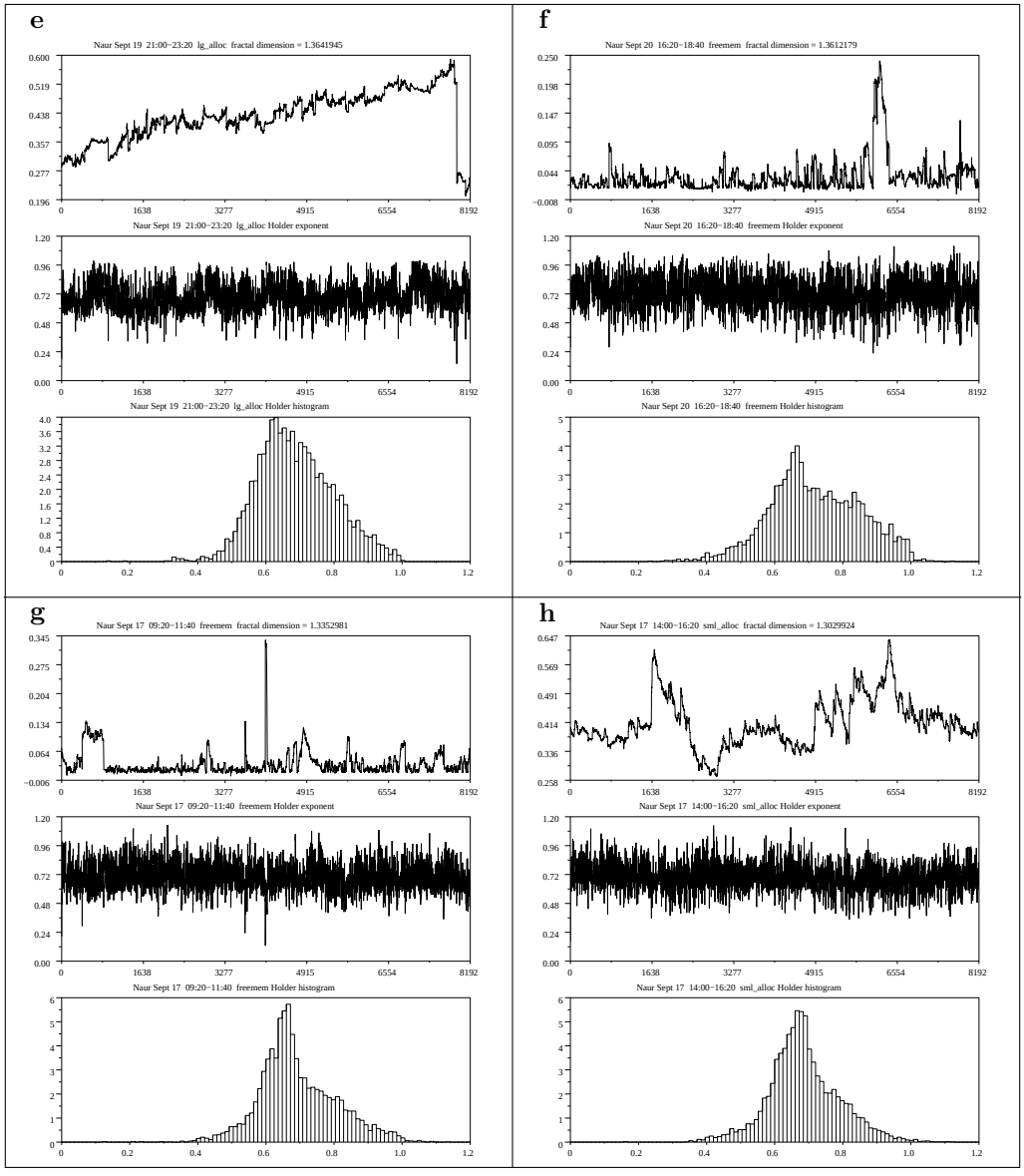

Figure 4.11: Comparison of Hölder exponent with fractal dimension for sml_alloc, lg_alloc, and freemem at different times.

Chapter 5

Conclusion and Further Work

5.1 Conclusion

The results of this study showed definite fractal behavior of several memory-usage parameters. The most interesting results were obtained through analysis of the lg_alloc parameter, but positive results were also obtained from the freemem, freeswap, and sml_alloc parameters. Analysis of the sml_mem and lg_mem parameters proved useful as examples of a lack of fractality to contrast with the more fractal parameters. The parameter displaying the strongest multifractality was the freeswap parameter.

The Hölder exponent behaved as expected — falling with rises in the fluctuation of the parameters and rising when the parameters stabilized. Analysis of the Hölder exponent using histograms showed humps and spikes corresponding to the different plateaus of fractal behavior within the overall multifractal behavior. For many time periods, there was a dominant hump in the histogram of the Hölder exponent, suggesting a strong disposition toward a single fractal level. For these time periods, analysis of the fractal dimension confirmed the calculation of the Hölder exponent surprisingly well for an experimental situation such as ours.

5.1.1 Automated Detection of Fractality

Often a resource will begin to display fractal behavior at a certain point in time without exhibiting a dramatic rise or fall in the scale of its behavior. Without the aid of a plot, the onset of fractality may be difficult to detect. The Hölder exponent provides a means of detecting fractality through the pure analysis of numbers. For instance, figure 5.1 shows the lg_alloc parameter along with Hölder exponent and histogram for the hour and eight minutes immediately preceding 4:13 AM on

September 22nd, and the hour and eight minutes immediately succeeding 4:13 AM on September 22nd.

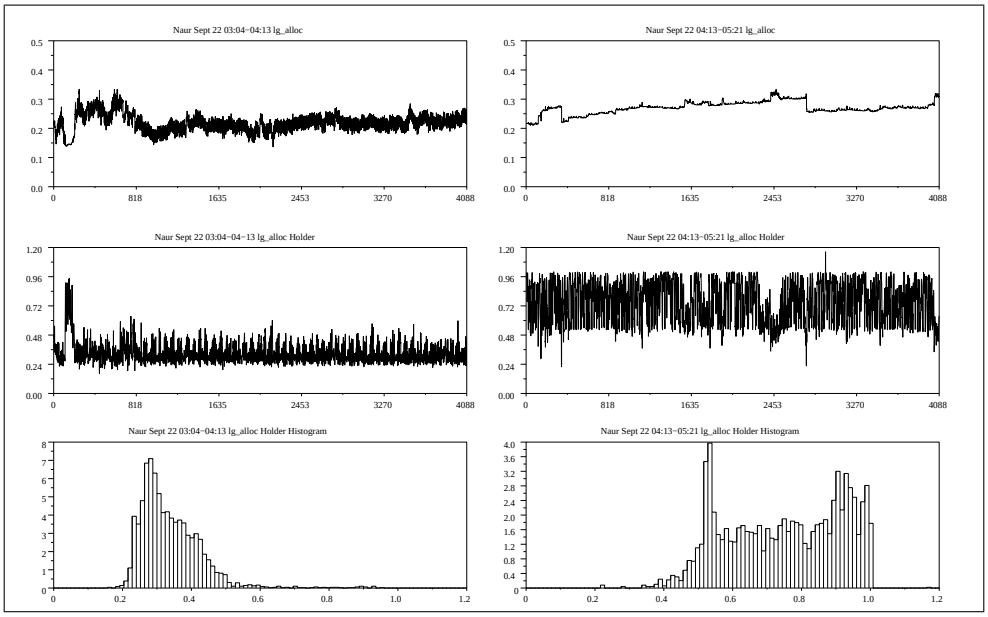

Figure 5.1: Lg_alloc resource remaining at relatively constant scale but dramatically changing in fractality

During these two time periods the magnitude of the lg_alloc resource remains within the same general area — never falling lower than 0.135 and never exceeding 0.345. Without the aid of a plot, a casual glance at the numbers would not immediately reveal a change in the behavior of the lg_alloc parameter. A casual glance at the numbers in the Hölder exponent, however, would immediately reveal that a change has occured. The weight of the values of the Hölder exponent for the time preceding the change is 0.3 with only a very few Hölder exponent exceeding 0.5, while after the change occurs almost all of the Hölder exponents exceed 0.5.

The Hölder exponent may, then, be a useful means by which an operating system could monitor the fractal behavior of its resources. By keeping a record of a small moving average of the Hölder exponent for a certain resource available at all times, the operating system would be able to quickly detect the onset of fractal behavior in that resource.

One may wonder why an operating system would be interested in knowing whether any of its resources were displaying fractal behavior. It is hypothesized that a computer whose resources are in the throes of fractality is not in a stable condition. Figure (5.2) shows that the onset of fractal behavior in the lg_alloc resource during the mornings of September 17th and September 22nd was

immediately followed by a large drop in the availability of that resource. This example indicates that such outburst of fractality may be signs of software ageing developing in the system.

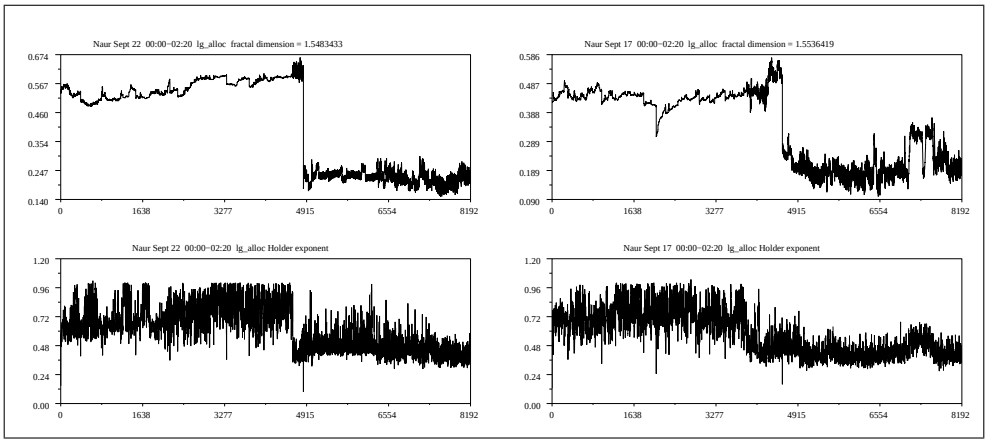

Figure 5.2: Availability of the lg_alloc resource falls soon after the onset of fractality.

Knowledge that an operating system is ageing would be invaluable in helping to prevent the loss of data through crashes. If the link between fractality and software ageing can be strongly established, predictoins about the future behavior of the operating system may become feasible — enabling us to take preventive action if a crash is imminent. Instead of suffering the loss of data, we could save the data and reboot the system cleanly.

5.2 Further Work

Much further work is suggested by this study. Developing methods for the fractal analysis of operating system resources that are measures rather than signals would allow us to monitor many more operating system resources and perhaps make more accurate predictions about the future behavior of the system. Fractal measure analysis would require a more strenuous data collection process, however, in which readings are made at least tens of thousands of times per second. This would require the use of specialized software and perhaps even specialized hardware.

Probabilistic techniques that have previously been used in the fractal analysis of network traffic, such as long-range dependence analysis and Hurst exponent analysis, could be applied to operating system resource data. In addition, modeling operating system resources through multifractal Brownian Motion, such as has already been done for network traffic (see [24]), could lead to a greater understanding of the fractal behavior of those resources. Wavelet-based fractal analysis is another area that holds promise for studying the fractal behavior of time series, but which we did not have

the breadth to pursue in this study.

In addition, simulating operating system crashes while measuring various system resource parameters leading up to the crash would grant better insight into whether system crashes are anticipated by fractal behavior. Fractal behavior in relation to other characteristics of the system could also be observed. For instance, unusual use of the system due to a security breach might be detected through the unusual fractal behavior of certain resources.

Multidimensional fractal analysis could also be employed to a greater extent than has been done in this study to study the behavior of many resources in conjunction.

Bibliography

[1] A. Puliafito M. Telek K. Trivedi A. Pfening, S. Garg. Optimal software rejuvenation for tolerating soft failures. *Performance Evaluation*, 27 and 28, October 1996.

[2] Michael Barnsley. *Fractals Everywhere*. Academic Press, 1988.

[3] Khalid Daoudi and Jacques Lévy Véhel. Speech signal modeling based on local regularity analysis. *IASTED IEEE International Conference on Signal and Image Processing*, 1995.

[4] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley and Sons, 1990.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[6] G. H. Hardy. Weierstrass's non-differentiable function. *Transactions of the American Mathematical Society*, 17(3):301–325, Jul 1916.

[7] Brian R. Hunt. The hausdorff dimension of graphs of weierstraß functions. *Proc. Amer. Math. Soc.*, 126:791–800, 1998.

[8] INRIA – Unite de recherche de Rocquencourt - Project Meta2, Domaine de Voluceau – Rocquencourt – B.P. 105-78153 Le Chesnay Cedex (France). *Fraclab*. http://www-rocq.inria.fr/scilab.

[9] Evelyne Lutton Jacques Véhel, Khalid Daoudi. Fractal modeling of speech signals. *Fractals*, 2(3):379–382, June 1994.

[10] W. Willinger K. Park. *Self-Similar Network Traffic and Performance Evaluation*, chapter Self-Similar Network Traffic: An Overview. Wiley-Interscience, 2000.

[11] Yves Meyer Khalid Daoudi, Jacques Lévy Véhel. Construction of continuous functions with prescribed local regularity. *Journal of Constructive Approximation*, 14(3):349–385, 1998.

[12] Benoit Mandelbrot. *Fractals: Form, Chance, and Dimension*. W.H. Freemen and Company, 1977.

[13] Benoit Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freemen and Company, 1982.

[14] Petros Maragos. Fractal aspects of speech signals: Dimension and interpolation. *Proc. IEEE ICASSP*, pages 417–420, May 1991.

[15] Petros Maragos. Modulation and fractal models for speech analysis and recognition. *Proceedings of COST-249 Meeting*, Feb 1998.

[16] R. Morin A.L. Goldberger L.A. Lipsit N. Iyengar, C.K. Peng. Age-related alterations in the fractal scaling of cardiac interbeat interval dynamics. *American Journal of Physiology*, 40(4):1078–1084, 1996.

[17] M. Osborne. Periodic structure in the brownian motion of stock prices. *Operations Research*, pages 345–379, May-June 1942.

[18] A. Goldberger S. Havlin M. Rosenblum Z. Struzik H. Stanley P. Ivanov, L. Amaral. Multifractality in human heartbeat dynamics. *Nature*, 399:461–465, Jun 1999.

[19] David Parnas. Software aging. *Proceedings of the 16th Intl. Conference on Software Engineering*, pages 279–287, 1994.

[20] Edgar E. Peters. *Fractal Market Analysis*. John Wiley and Sons, 1994.

[21] K. Trivedi and K. Vaidyanathan. A measurement-based model for est of resource exhaustion in operational software systems. *Proceedings of the 19th International Symposium on Software Reliabiltiy Engineering*, 1998.

[22] P. Heidelberger S. W. Hunter K. S. Trivedi K. Vaidyanathan W.P. Zeggert V. Castelli, R.E. Harper. Proactive management of software aging. *IBM J. Res. and Dev.*, 45(2):311–332, Mar 2001.

[23] Jacques Lévy Véhel and Khalid Daoudi. Generalized IFS for signal processing. *IEEE DSP Workshop*, Sep 1996.

[24] Jacques Lévy Véhel and Rudolf Riedi. Fractional brownian motion and data traffic modeling: The other end of the spectrum. In E. Lutton J. Lévy Véhel and C. Tricot, editors, *Fractals in Engineering*. Springer, 1997.

[25] Nick Kolettis N. Dudley Fulton Yennun Huang, Chandra Kintala. Software rejuvenation: Analysis, module and applications. *Proceedings of the 25th Intl. Symposium on Fault-Tolerant Computing*, 1995.