THE ANALYSIS OF CHAOTIC TIME SERIES

A Thesis
Presented to
The Academic Faculty

by

Joshua D. Reiss

In Partial Fulfillment
Of the Requirement for the Degree of
Doctor of Philosophy in Physics

Georgia Institute of Technology
May 25, 2001

THE ANALYSIS OF CHAOTIC TIME SERIES

Approved:

____
William L. Ditto, Advisor

____
Kurt Wiesenfeld

____
Michael Schatz

____
John Lindner

____
Steven Deweerth

____
Michael Farmer

Date Approved:_______________

Acknowledgments

The author wishes to express sincere appreciation to Professor William Ditto for his assistance in the preparation of this manuscript and his guidance throughout my research. In addition, special thanks to my collaborators and colleagues Brian Meadows, Visarath In, Joseph Neff, Paul Garcia, Barbara Breen, Jonathan Mason, Bruno Robert and Mark Sandler. Thanks also to the members of the thesis committee for their valuable input and assistance. More thanks to: my parents and my brothers, my professors, generous and helpful graduate peers, and all my friends in Atlanta and London. Lastly, thanks and appreciation to everyone who has helped me over the past few years who I neglected to mention above.

Table of Contents

List of Figures

List of Tables

ABSTRACT

The Analysis of Chaotic Time Series

Joshua D. Reiss

232 Pages

Directed by Dr. William L. Ditto

Chaotic time series analysis methods were applied to several experimental systems. Analysis of a Poincare section of magnetoelastic ribbon time series was used to construct symbolic dynamics and extract empirical quantities such as fractal dimension and Lyapunov exponents. In the pulse thermal combustion engine, analysis of several data sets was used to establish high dimensionality and complex dynamics. Data sets were also analyzed from an electric step motor. Low dimensional chaotic dynamics were observed and quantified. Each of these systems exhibited nonstationarity and other behaviors that made the analysis difficult and demonstrated flaws in established time series analysis techniques. Thus methods were devised to improve these techniques and synthesize them into a coherent package. Finally, a new design was proposed for a chaotic sigma delta modulator. Techniques from nonlinear dynamics and chaos theory were used to show that this modulator was stable and had desirable properties not exhibited in previously proposed designs.

CHAPTER ONE

INTRODUCTION

1. Historical background

As opposed to the many fields of mathematical and scientific endeavour which have grown out of philosophy, time series analysis has its roots in the social sciences. The need for time series analysis came from emerging financial markets, the population explosion and plague epidemics. The opportunity for time series analysis came from more accurate recordkeeping.

Although time series have been used extensively in astronomy since data was first gathered, the first work of detailed time series analysis is often attributed to a London cloth merchant in 1662. John Graunt[1] devised a "Mathematiques of my Shop-Arithmetique" that involved the creation of his own analytical tools to search for, predict, and interpret cyclical patterns in time series data. Such tools included histogram tables, averaging and windowing techniques, and error analysis. Significantly, he was able to extract the plague-related deaths from the others, and thus could give a detailed analysis of the long-term effects of plague epidemics. His subsequent appointment to the Royal Society lead to a great many scientists and mathematicians adopting his time series analysis techniques.

Most of the related mathematical work of the next two hundred years dealt more with independent observations than with time series, per se. Pascal, Fermat, Bernoulli and others laid down the framework for descriptions of statistical processes.[2] In 1821, Gauss developed the concept of mean squared error, one of the most important methods of quantifying error in data.[3] Simple time series analysis continued primarily in the analysis of financial data. This was often done to obscure the data, such as when the Bank of England chose to scale and index their data when presenting time series data to parliament in 1797. [4] Some notable exceptions occured in astronomy, such as Euler's 1749 study of errors in the observed orbits of planets.[5]

The late 19th century saw the exporting of financial time series analysis techniques to other applications. A method of interpolation was suggested by Gauss and Weber in 1836.[3] It was used in one of the first examples of simultaneous observations, thus in the creation of multidimensional time series. Poynting, known primarily for his work on magnetism, first provided a graph of moving averages in a study of public drunkenness in 1877.[6] The following year, Jevons introduced the semi-log plot to highlight cycles in meteorological and commercial data.[7] These and other developments also lead to the gradual transition towards visualization of data as opposed to presentation in tables (Ref. 2, p. 17-19).

It wasn't until the 1920s that chaos was observed in an experimental time series. However, Van der Pol dismissed it as "a subsidiary phenomenon."[8] He neglected to connect this to the behavior that Poincare had hypothesized at the turn of the century.[9] Perhaps this was because he was unable to extract and analyze the time series.[10]

Statistical tools continued to be developed throughout the first half of the twentieth century. In the first decade, Norton[11] developed tools for dealing with nonstationary processes. In addition, Cave-Browne-Cave[12], March[13] and Hooker[14] developed tools for autocorrelations in time series. Anderson[15] and Student[16] developed further techniques for elimination of noise. The 1920s saw the invention of the correlogram, one of the most popular tools for classifying serial correlations and oscillatory processes.[17] This work was also important because it assumed that the time series could be modeled as having self-determined dynamics. Most previous work assumed that time series were functions of time (typically cyclical) and noise.

Several scientists of this era made the assumption that time series were strictly functions of time and superimposed random errors. This allowed them to make full use of Fourier's analysis tools In a comment published in 1879 regarding a previous analysis of sunspots, Stokes[18] suggested the use of harmonic analysis to plot the intensity of each harmonic (e.g. power spectra). Schuster[19] and Turner[20] followed up on this suggestion in two further works of sunspot analysis. Terms such as periodograms and harmonic analysis entered the lexicon, and hence traditional signal processing was born. Signal processing continued to gain in importance with the growth of the electronics industry, and exploded in terms of usefulness with Cooley and Tukey's invention of the Fast Fourier Transform computer program in 1965.[21] Spectral analysis saw another fantastic leap with the introduction of wavelets in the mid 1980s.[21] Now signal processing techniques could be effectively extended to nonstationary time series.

The early 20th century also saw time series given proper mathematical underpinnings in the work of Kolmogorov[22], Markov[23] and other Russian mathematicians. By the 1940s time series analysis had advanced beyond the application of statistical techniques to experimental data. With the invention of information theory by Shannon and Weaver,[24] time series could be understood in terms of symbolic dynamics.

It wasn't until 1963 that chaos was recognized in time series data. Lorenz, a meteorologist, serendipitously noticed that his simulation was extremely sensitive to initial conditions.[25] Chaos theory slowly gained acceptance through the work of Smale, Yorke, May and others.[10] In the 1980s, a motley crew of researchers from UC Santa Cruz developed the essential tools of chaotic time series analysis. In 1980, Farmer and Crutchfield applied traditional signal processing techniques, in the form of power spectral analysis, to chaotic time series.[26, 27] The following year, Packard introduced the technique of delay coordinate embedding.[28] Shaw then applied information theoretic techniques to chaotic time series analysis.[29] In these and other papers, these four researchers and their collaborators introduced techniques for noise reduction, prediction, extraction of symbolic dynamics, modeling and quantification of chaotic time series, experimental or simulated. Much of the work described in this thesis is built upon the techniques that they developed.

The introduction of these techniques provided a framework with which one can analyze a chaotic time series. However, many questions remain unanswered about the effectiveness and practicality of each technique. Even with the vast amount of research within the past two decades, it is still often unclear which analysis techniques are most appropriate to a given time series. From the perspective of a researcher whose knowledge of time series analysis is limited, this question becomes even more vexing.

2. Goals and motivations

"In calmer moments we are all willing to concede that there is a danger that the quantitative worker may concentrate too exclusively upon those things which are incapable of measurement, and may even shape his concepts to the limited nature of his available data, while the qualitative worker - the pure theorist - may become so pure as to be unhelpful in a world wheree masureable quantities play an important role. Moreover in the last resort neither school will insist that it can progress without entering the domain of the other."
Translated from Marey, 1878

When a researcher is presented with time series data, he has several goals in mind- classification, visualization, understanding, and manipulation. The data is analyzed with the goal of understanding more about the underlying system. What is known about the underlying system determines the direction of future research. The researcher seeks to answer several questions-

1. What can I determine about the underlying dynamical system which generated the data?
2. How best should I visualize this system?
3. What processing of the data should be done that will reveal more about he underlying dynamics?
4. Can I, from analyzing the dynamics, determine additional experiments that should be done to generate future data?
5. Can I measure how well the data fit a certain model?
6. Can I measure the complexity of the data?

7. Can I extract distinct processes that are contributing to the dynamics generating the data?
8. Can I measure the noise in the data, and if so, can I extract the signal from the noise?
9. Can I determine how the data changes over time?
10. If I have several time series, what analysis can be made to determine the relationship between these series?
11. Have I taken enough data to do analysis? Can I get away with less?

Traditional methods of time series analysis come from the well-established field of digital signal processing.[30] Digital signal processing seeks to answer all of the above questions. Most traditional methods are well-researched and their proper application is understood. One of the most familiar and widely used tools is the Fourier transform. Indeed, many traditional methods of analysis, visualization, or processing begin with the use of a Fourier transform on the data set. This is because a Fourier transform changes a set of linear differential equations into an algebraic problem where powerful methods of matrix manipulation may be used.

However, these methods are designed to deal with a restricted subclass of possible data. The data is often assumed to be stationary, that is the dynamics generating the data are independent of time. It is also assumed that the dynamics are fairly simple, both low dimensional and linear. Compound this with the additional assumptions of low noise and a nonbroadband power spectrum, then it can be seen that a very limited class of data is often assumed. With experimental nonlinear data, traditional signal processing methods may fail because the system dynamics are, at best, complicated, and at worst, extremely noisy.

In general, more advanced and varied methods are often required. These new methods bring with them a collection of new questions. The researcher is then concerned additionally with the implementation of these new methods.

1. Can I trust the results of a method of analysis or visualization?
2. Does the method provide a way to measure its accuracy?
3. How long will it take to implement this method?
4. How difficult is it to implement this method?
5. What are the limitations of this method?
6. Are the results of analysis repeatable?
7. How useful are the results of the analysis?
8. Can I make the implementation of this method simpler or more useful by combining it with other methods?
9. Can I eliminate duplication by replacing other methods with this method?

Often when a time series analysis method is first presented, it is tested against simulated systems where the expected results are known. To show how effective it is with real data, it is applied to a well-known system with the addition of white or gaussian noise. In those cases where the authors apply it to an experimental system, a relatively simple well-studied system with low noise levels and low dimensional dynamics is often used. The goal of such an analysis is usually not to deal with the peculiarities and issues presented by a particular data set, but instead to demonstrate that application is at least sometimes possible. However, real world data is rarely so simple. The analysis method may fail once applied to more realistic data, or require an augmentation suitable to the system that is analyzed.

Therefore it is our hope that thorough analysis from a nonlinear dynamics perspective may yield more fruitful results. However, this is not a straightforward task. Calculation of empirical global nonlinear quantities, such as Lyapunov exponents and fractal dimension, from time series data is known to often yield erroneous results.[31, 32]. The literature is replete with examples of poor or erroneous calculations and it has been shown that some popular methods may produce circumspect results.[33, 34] Limited data set size, noise, nonstationarity and complicated dynamics only serve to compound the problem.

So the researcher is now presented with additional complications. The concerns about the data are compounded by concerns about analysis. For most researchers, the analysis methods are simply tools for the researcher to gain a better understanding of the system which generated the data. What is required are analysis methods that are relatively simple to implement and simple to interpret. Ideally the results of preliminary analysis should indicate the directions of further analysis, and from there, further directions of research.

This is the goal of this thesis. It is hoped that a synthesis of new methods of analysis in a consistent, unified form will enable researchers to perform analysis more efficiently and more confidently.

The authors have taken a skeptical approach to the analysis. We demand that quantitative results be confirmed by other independent methods. Even then we accept the results only qualitatively. Wherever possible, we point out where these methods may fail, and suggest criteria which may be used in validating results obtained using these or similar methods on other data sets.

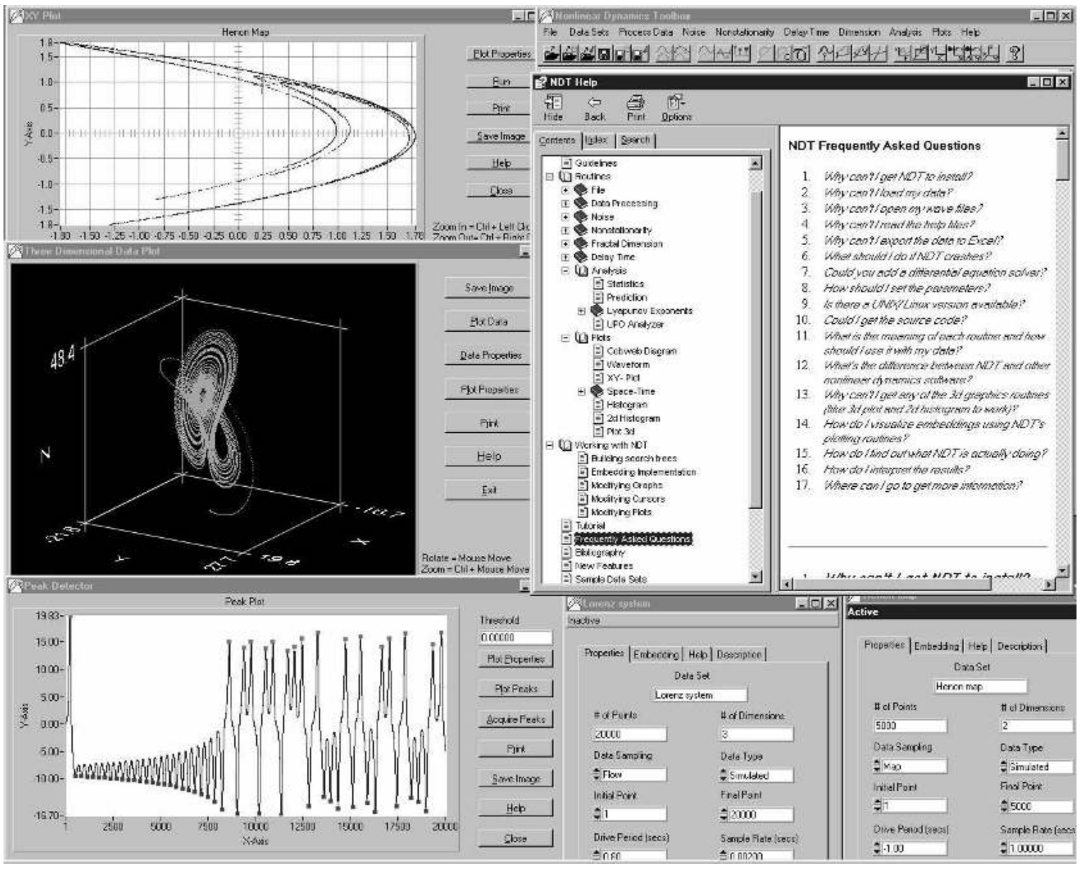
An additional problem is presented when new methods of data analysis are performed. In order to maintain generality, algorithms are usually presented with a vast number of parameters. This has the effect of making methods versatile and easily modifiable. However, suggested parameter values are often not available. A thorough knowledge of the subject matter is necessary in order to achieve reasonable results. Thus it is encouraged that the user first familiarize himself with the literature, the method and the programming techniques before attempting analysis. Many laymen and nonexperts are dissuaded from analyzing their data due to the amount of effort it would take to achieve reasonable results.

Thus we argue that reasonable parameter values should be suggested wherever possible. If possible, instructions should be given as to how to estimate or zero in on ideal parameters, and criteria should be developed that help deny or confirm whether the implementation is correct. Output from a numerical routine should also be extensive. Interpretations should be suggested, but the output should also be copious so that, if the user prefers, he can modify and interpret the results as he wishes. We feel this issue has not been thoroughly addressed in the literature or in other nonlinear dynamics software. Rather than complicating the question by requiring that a high level of understanding be present when using the analysis tools, we feel that it is preferable to embed extra functionality into the software. In essence, wherever possible, the software should incorporate the results of previous research on time series analysis. The onus of interpretation should be on the software, not on the researcher or user. We should stress that this is not simply a question of error bars. Error bars don't tell about systematic errors and neither do they tell if the underlying assumptions are justified.

Particular attention was paid to the graphical user interface. This is often neglected in scientific software. This is understandable, because the priority is placed on the algorithms used. However, lack of a graphical user interface also makes the software difficult to use and can obscure the results. Thus, we felt that an improved user interface would aid in the production and evaluation of quantitative results. In addition, we thought that a few nonscientific features would add greatly to the usefulness of the software. Therefore we wish to provide the ability to work with many data sets at the same time, to interface with popular software, to handle multiple data formats, and to customize the output extensively.

3. Overview

For these reasons the Nonlinear Dynamics Toolbox was created. The Nonlinear Dynamics Toolbox (NDT) is a set of routines for the creation, manipulation, analysis, and display of large multi-dimensional time series data sets, using both established and original techniques derived primarily from the field of nonlinear dynamics. In addition, some traditional signal processing methods are also available in NDT. A screen shot of the NDT user interface is provided in Figure 1.



NDT Frequently Asked Questions

1. *Why can't I get NDT to install?*
2. *Why can't I load my data?*
3. *Why can't I open my wave files?*
4. *Why can't I read the help files?*
5. *Why can't I export the data to Excel?*
6. *What should I do if NDT crashes?*
7. *Could you add a differential equation solver?*
8. *How should I set the parameters?*
9. *Is there a UNIX/Linux version available?*
10. *Could I get the source code?*
11. *What is the meaning of each routine and how should I use it with my data?*
12. *What's the difference between NDT and other nonlinear dynamics software?*
13. *Why can't I get any of the 3d graphics routines (like 3d plot and 2d histogram to work)?*
14. *How do I visualize embeddings using NDT's plotting routines?*
15. *How do I find out what NDT is actually doing?*
16. *How do I interpret the results?*
17. *Where can I go to get more information?*

Lorenz system — Inactive

Data Set	Lorenz system
# of Points	20000
# of Dimensions	3
Data Sampling	Flow
Data Type	Simulated
Initial Point	1
Final Point	20000
Drive Period (secs)	0.60
Sample Rate (secs)	0.00200

Active

Data Set	Henon map
# of Points	5000
# of Dimensions	2
Data Sampling	Map
Data Type	Simulated
Initial Point	1
Final Point	5000
Drive Period (secs)	-1.00
Sample Rate (secs)	1.00000

Figure 1. A screenshot of the user interface for the Nonlinear Dynamics Toolbox. This software was created by the author for the purpose of analyzing and visualizing chaotic time series data.

The Nonlinear Dynamics Toolbox is written entirely in portable C, and consists of some 50,000 lines of original code. The programs utilize subroutines from Numerical Recipes and LabWindows/CVI. In addition, some publicly available code for time series analysis has been ported with only minor modifications. The software has been designed for use under Windows 95, 98, NT, and all other similar variants. However, it is semiportable, and versions for other platforms may be easily created.

The modules are coded assuming a flat memory model, and virtual memory support, all of which is handled by the operating system. On a Pentium II based PC with 64 MB of memory, data sets of up to 1 million points can be manipulated in main memory without paying a virtual memory overhead penalty. In some cases, notably graphics, a user will notice significant differences between the modules on different machines. However, with these few exceptions, NDT descriptions (and functionality) are independent of the host computer.

The full power of NDT lies in the ability to manipulate multiple files in many different ways from one simple interface. Information can be shared between different routines so that multiple analyses can be performed quickly in one session. Almost all analysis and visualization presented in this thesis was performed using NDT.

Chapter 2 deals with ways that data can be generated or imported into the analysis program. This chapter explains the data structures used for storing and manipulating time series. It describes how data set parameters are stored, the format of the data, and how simulations are created. This chapter also contains a complete description of some of the most relevant routines used to analyze the data. Some background information on analysis methods is included, and the methods are put in the proper framework concerning how and when they should be applied.

Chapter 3 contains a description of the sorting and searching routines used. These are at the heart of most of the time series analysis methods that were implemented. The proper implementation of these methods allow complicated analysis to be performed in a highly efficient manner. We place primary importance on the Kd-tree, which is the preferred implementation for many routines. Benchmarking is performed comparing the Kd-tree with the KTree, a simple multidimensional sort, and a box-assisted sort. A description is also given for the multidimensional binary representation sort, which is preferred for calculation of many information theoretic quantities.

In Chapter 4, we give a more detailed description of some original analysis methods as well as modifications to analysis methods that we believe represent improvements over their original implementation. Some of the modifications are quite small, such as the simplification to Alan Wolf's method of determining the dominant exponent, while others, such as the method of determining generalized dimensions, are entirely original. This chapter also deals with how to extract the symbolic dynamics from a time series. A new method is devised and placed on a firm mathematical basis. Its accuracy and efficiency are described, and its usefulness is shown. Some examples of its use are presented, and it is compared with other methods.

The results are presented in Chapter 5. Explicit description of all the simulations and experimental systems analyzed in this thesis are described here. We perform analysis on various experimental time series and particularly complicated simulations. In these cases we do not have any direct knowledge of what the outcomes should be. Thus the only confirmation of results possible is to compare the results of routines that use completely different methods to estimate the same quantities. We also discuss the pitfalls of analysis of experimental data, especially nonstationary, high dimensional, or noisy data of possibly small data sets. We make some suggestions as to how best to treat such data.

In the conclusion, we summarize what was accomplished. We suggest some further directions of study, and point to where the greatest improvements can be made. We also speculate on the nature of this type of research, and on some of the most problematic issues. Credit is given to everyone who generously contributed to the work.

CHAPTER TWO

SOFTWARE IMPLEMENTATION

1. Data importing and simulation

1.1. Introduction

Throughout this work, we restrict our discussion to time series data. Typically, experimental time series data consists of a collection of data points sampled uniformly in time. However, we wish to perform analysis on a wide variety of data in a consistent manner. The time series may be from a simulation or from an experiment, and may also come in a wide variety of forms. If the data is from a simulation, it may be generated externally from another program or internally by the analysis software. In either case, the time series may have several data values at each time or be restricted to one data point per sample time. One would like to be able to deal with many types of time series data in a uniform manner.

1.1.1. Notation

The vectors are assumed to be sampled uniformly in time. That is, at each time interval Δt, a new vector is sampled, and Δt is a constant. Such a data set will be referred to in the text as a series of *M* vectors $\overline{X_0}, \overline{X_1}, ..., \overline{X_{M-1}}$ where each vector is *N* dimensional, $\overline{X_i} = (X_i^0, X_i^1, ...X_i^{N-1})$. If the time series is from a single channel, then these vectors become scalar quantities, $\overline{X_i} = X_i^0$. Alternately, the data set may be described as data from *N* channels, $\overline{X^0}, \overline{X^1}, ..., \overline{X^{N-1}}$, where each channel consists of *M* consecutive samples, $\overline{X^j} = (X_0^j, X_1^j, ...X_{M-1}^j)$.

Data sets, both experimental and simulated, come in two distinct forms that need to be distinguished when implementing time series analysis methods. The first is true multidimensional data. This is data where the time series consists of multiple channels, e.g., voltage and current values recorded simultaneously from a circuit. The other form is scalar data, consisting of single channel data where only one measurement is made at each time. In this case, other dimensions may be constructed by manipulations of the original data. Consider data from a delay coordinate embedding of the *x* dimension of Lorenz data. If only one variable from the system can be observed, *X*(*t*), then (*n*+1)-dimensional vectors are constructed in the form (*X*(*t*),*X*(*t*+*T*),...,*X*(*t*+*nT*)). This is a delay coordinate embedding. It is one method of phase space reconstruction, since it allows the phase space to be reconstructed from only one coordinate.

Mané[35] and Takens[36] independently showed that delay coordinate embedding was a valid technique for reconstructing vectors from scalar quantities. Packard, et. al.,[28] applied this technique to experimental time series, and Eckmann and Ruelle[37] showed that it was a valid method for examining dynamics of a system.

One dimension of the time series is extracted to use for embeddings. So we consider the series $X_1^j, X_2^j, ...X_{npts}^j$. The embedding is a reconstruction of vectors from this time series. Two parameters are used for the reconstruction, a delay *d* and an embedding dimension, *D*. *D* describes the dimensionality of the reconstructed vectors, and *d* describes the separation between data points from the original time series that are used as successive points in a vector. Suitable choices for each of these parameters may be determined through analysis of the data (see Chapter 2). Thus, from the time series, the following vectors are created

$$\begin{aligned}
&\overline{Y_0} = (X_0^j, X_d^j, X_{2d}^j, \ldots X_{(D-1)\cdot d}^j) \\
&\overline{Y_1} = (X_1^j, X_{1+d}^j, X_{1+2d}^j, \ldots X_{1+(D-1)\cdot d}^j) \\
&\ldots \\
&\overline{Y_{M-1-(D-1)\cdot d}} = (X_{M-1-(D-1)\cdot d}^j, X_{M-1-(D-2)\cdot d}^j, \ldots X_{M-1}^j)
\end{aligned} \qquad (1)$$

The *a*th component of the *b*th vector of the embedding is related to the original time series by

$$Y_b^a = X_{b+(a-1)\cdot d}^j \qquad (2)$$

Note that the reconstructed series contains $(D-1)\cdot d$ fewer vectors than the time series. This is because each reconstructed vector extends over $D\cdot d$ consecutive time series vectors. Any delay coordinate embedding vector past $\overline{Y_{M-1-(D-1)\cdot d}}$ would have components which extend beyond the original time series.

1.1.2. The data set structure

One may wish to switch back and forth between the two representations. For instance, if one wishes to compare the results of a Lyapunov exponent calculation from an embedding with the results of an exponent calculation from the original data, then it would make sense to store all dimensions of the data, but manipulate the data in its original form or as an embedding. Thus a general approach has been taken. All the original data is stored in memory. This data may be manipulated, analyzed and visualized either as embeddings of one dimension or in its original form. At no point does the data need to be copied to create or manipulate embeddings. If another form of phase space reconstruction is used, then a change needs to be made only to the function that states how vectors are constructed. All other aspects of an analysis routine may remain the same.

In addition, it is important that the parameters for each data set be saved. This means that the program must store all relevant data set settings in memory while working with the data, as well as saving these settings to disk for the next time the data set is opened. This is accomplished by defining a data set type, which is a C structure that stores all the data in a 2 dimensional array, and all data set settings. The data set structure is in the following form. Some of the structure's data have been removed from the description because they pertain primarily to user interface settings.

```
typedef struct
{
        char  name[300];
        int ndim;
        int npts;
        long double **matrix;
        double samplerate;
        int datatype;
        int datasampling;
        int initialpoint;
        int finalpoint;
        int step;
        int embed;
        int dimtoembed;
        int embeddingtype;
        int embeddim;
        int delaytime;
        int kd-tree;
        //additional optional parameters...
} DataSetType;
```

ndim denotes the number of dimensions for each data point, and npts is the total number of data points. The data is in the form of `matrix[npts][ndim]`. Thus all data sets are stored in memory as two dimensional arrays in the form `data[# of samples][# of points at each sample]`. *samplerate* is just one over the sample frequency. It is the time between consecutive samples, and is typically given in seconds. *datatype* describes the type of data that is being dealt with. This is an integer that assumes one of the following values: 0 if the data type is unknown, 1 if the data comes from a simulation and 2 if the data is from an experiment. *datasampling* describes how the data was sampled; 0 if this is unknown, 1 if the data is a Poincare section (or some similar form of sectioning flow data), 2 if the data is from a flow (many samples per period), and 3 if the data is from a map (it is not possible to take many samples per period).

initialpoint simply specifies which is the first point in the data set to use when performing an analysis, processing, or visualization routine. This is useful when one wants to remove the transients from the analysis, but does not want to remove the transients from the data set. Similarly *finalpoint* specifies the last point in the data set that will be used. *step* specifies how data points are skipped. If *step* =2, for instance, then only every other data point is used.

The next few parameters deal with delay coordinate embedding. In NDT, unless otherwise specified, delay coordinate embeddings are performed using the above procedure. Thus the original data is not modified, and modifications are performed simply by turning the embedding on or off and specifying parameters. The computational overhead is small. Equation (2) is used to determine the value of any dimension of any coordinate in the reconstruction. In the C/C++ coordinate language, arrays are indexed beginning at 0. The a^{th} component of the b^{th} vector of the delay coordinate embedding are thus determined from Equation (2) as

`Y[b][a]=X[b+a*delay][j]` (3).

Thus the added computation is only in the integer multiplication of `a*delay`. However, the memory savings may be quite large. No additional memory is required, since only the original series is used. In a common situation, the original data is one-dimensional, and a three dimensional embedding is required (for example, 3d visualization). Under these circumstances, almost three times the original data storage would have been required if a new array were created to store the embedded data.

embed is a boolean variable that is set to 1 if the data is to be embedded, and 0 if not. *embeddingtype* is also boolean. A value of 1 for the *embeddingtype* implies that a forward embedding is used (e. g., $\overline{X} = (X_1, X_2)$), and a value of 0 implies a backward embedding (e. g., $\overline{X} = (X_2, X_1)$). *dimtoembed* describes which dimension of the original data is to be embedded. *embeddim* is the embedding dimension of the embedded data, and *delaytime* is the delay between data points in the embedding.

Further parameters may be used to specify properties of the vectors. Initial or final vectors may be excluded, intermediate vectors may be skipped, or components of each vector may carry associated weights. However, the essential components of delay coordinate embedding have been fully described above.

The *kd-tree* parameter is used to specify if a search tree has been built. Other parameters, such as *noiselevel*, are specified but not used. These exist so that additional functionality may be built into the data structure at a later date.

1.2. Generating Data

1.2.1. Data acquisition

Analysis of any time series data set begins with the acquisition of the data. Whether experimental or simulated, this can be one of the most difficult tasks in the analysis process. The data analyzed comes in three distinct forms- maps, differential equations (or flows), and experimental data. For the experimental data, both the method of generating data and of importing it into the analysis software are discussed. In this section, the methods used to acquire data are discussed.

1.2.2. Data importing and exporting

Any data set in standard ASCII spreadsheet format may be read in as data. The file is assumed to have the .dat extension and no header information. However the .dat extension is optional and any other extension may be used. The spreadsheet format consists of columns of numbers (floating point or integer) separated by tabs, commas or spaces with no header. Thus a single time series would just be a series of numbers, such as

```
0.2345
-1.2429
3.4254472
0.233
1
3.32
```

Two dimensional data, such as position and velocity values would have two columns, one for position, one for velocity.

```
234.1 0.0235
230.1 0.3235
212.7 32.6
219.7 134.3
```

In conjunction with the data set, NDT looks for an *ini* file that stores information about the data. If an *ini* file is not found, these values are set at defaults. Similarly, data may be saved in ASCII spreadsheet format. Regardless of whether it is one generated from a sample system or from a previously opened file, it may be saved as a spreadsheet file with the *dat* extension. An associated *ini* file will also be saved. The data may be reloaded using the Open Data Set routine at a later date.

1.2.3. Simulation

Many systems were built directly into the software, as opposed to having the data collected elsewhere and then loaded into NDT. This was done for systems that are investigated frequently or were of special importance. It was also done so that many parameter settings could be compared quickly and easily.

In order to simulate these systems, an efficient and reliable scheme for numerical integration of ordinary differential equations was required. Numerical integration is often performed using the Euler method or Runge-Kutta integration. However, these methods have tremendous drawbacks in their reliability[38, 39] and do not offer an effective measure of their error. Even higher order variable Runge-Kutta integration may perform very poorly on stiff systems. For these reasons, a very advanced numerical integration routine, CVode[40, 41] was chosen.

CVODE is a solver for stiff and nonstiff initial value problems for systems of ordinary differential equation (ODEs). The capabilities of many older, Fortran based solvers have been combined in the C-language package CVODE. One key feature of the CVODE organization is that the linear system solvers comprise a layer of code modules that is separated from the integration algorithm, allowing for easy modification and expansion of the linear solver array. A second key feature is a separate module devoted to vector operations.

An ODE initial value problem can be written as

$$\dot{y} = f(t, y), \quad y(t_0) = y_0, \quad y \in R^n \qquad (4)$$

where $\dot{y}$ denotes the derivative dy/dt. This problem is stiff if the Jacobian matrix $J = \partial f / \partial y$ has an eigenvalue with a large negative real part. That is, one or more modes are strongly damped. Thus stiffness is a somewhat subjective call, but it is also an important one since it implies that numerical integration is prone to error. The underlying integration methods used in CVODE are variable-coefficient forms of the Adams (the Adams-Moulton formula) and BDF (Backward Differentiation Formula) methods. The numerical solution to an ODE initial value problem is generated as discrete values y_n at time points t_n.

A nonlinear system must be solved at each time step. In the nonstiff case, this is usually done with simple functional (or fixed point) iteration. In the stiff case, the solution is performed with some variant of Newton iteration. This requires the solution of linear systems. They are solved by either a direct or an iterative method. The direct solvers currently available in CVODE include dense (full) and banded methods, and a diagonal approximate Jacobian method. The iterative method is the Generalized Minimal Residual method.

The estimation and control of errors is an extremely important feature of any ODE solver. In CVODE, local trunctation errors in the computed values of y are estimated, and the solver will control the vector e of estimated local errors in accordance with a combination of input relative and absolute tolerances. The vector e is made to satisfy an inequality of the form

$$\|e\|_{W\,RMS,ewt} \leq 1 \tag{5}$$

where the weighted root-mean-square norm with weight vector w is defined as

$$\|v\|_{W\,RMS,w} = \sqrt{\frac{\sum_{i=1}^{N} (v_i w_i)^2}{N}} \tag{6}$$

The CVODE error weight vector ewt has components

$$ewt_i = \frac{1}{RTOL \cdot |y_i| + ATOL} \tag{7}$$

where the non-negative relative and absolute tolerances $RTOL$ and $ATOL$ are specified by the user. Here $RTOL$ is a scalar, but $ATOL$ can be either a scalar or a vector. The local error test (5) controls the estimated local error e_i in component y_i in such a way that, roughly, $|e_i|$ will be less than $(ewt_i)^{-1} = RTOL \cdot |y_i| + ATOL_i$. The local error test passes if in each component the absolute error $|e_i|$ is less than or equal to $ATOL_i$, or the relative error $|e_i|/|y_i|$ is less than or equal to $RTOL$. $RTOL = 0$ is used for pure absolute error control, and $ATOL_i = 0$ for pure relative error control.

Actual (global) errors, being an accumulation of local errors, may well exceed these local tolerances, so *RTOL* and *ATOL* should be chosen conservatively.

2. Analysis routines

2.1. Introduction

At the core of all the work presented in this thesis are the routines that NDT uses to process, analyze, and visualize the data. These routines are all original code, and incorporate a mix of well-established methods and original implementations. They are integrated together so that analysis and visualization can be performed in tandem. All the data sets are analyzed using the routines described herein.

The software that was created for this work includes a large number of routines that are not presented in this thesis. Specialized routines, such as the unstable periodic orbit analyzer, the information dimension routine, empirical mode decomposition, the Rosenstein method of determining dominant Lyapunov exponent, noise reduction and prediction routines have been successfully implemented but are not used in the analysis of the featured data sets from Chapter Five. Visualization routines, such as the two-dimensional histogram and the three-dimensional plotter are not described because their use should be apparent. Finally, the NDT software includes a large number of built in simulations. Because the emphasis here is on experimental systems, these simulations are not discussed.

2.2. Nonstationarity

Identification of nonstationarity or parameter drift in experimental data is a very important task before attempting to do quantitative analysis. Most analysis routines assume constancy of parameters, and quantitative results are very suspect if that is not the case.

No single method can definitively state that parameter drift has not occured, since some statistics may remain constant over the data set even while many of the properties are changing rapidly. However, a few simple statistics can identify many types of parameter drift easily and quickly. For the statistical drift routine, we look at how common linear statistics such as mean, maximum, minimum, standard deviation, skewness, and kurtosis may change when viewed over moving windows throughout the data.

The main parameters to be concerned with here are the window length and the discriminating statistic. A good choice of statistic is mean, since a fluctuating mean indicates that the data may be gradually rising or falling over time. With the appropriate window, measuring the mean may also be useful in identifying very low frequency fluctuations.

Suppose one looks at the mean with a data set of length n and window size w. Then starting at the first point X_1, the mean of points X_1 to X_w is calculated. This is plotted at X_1 on the X-Axis. Then the mean of points X_2 to X_{w+1} is calculated. This is plotted at X_2 on the X-Axis. One continues through the data set until the mean of points X_{n-w} through X_n has been plotted at point X_{n-w} on the X-axis. Thus one observes how the mean value changes over time.

Picking the appropriate window size is very important. Too small of a window size means that the original dynamics are recaptured, and too large of a window size will "average out" the fluctuations. Thus it is recommended that one should try a few windows in order to decide which is best.

2.3. Choosing delay time

2.3.1. Autocorrelation

The autocorrelation is defined from the standard deviation of a time series x, $\sigma_x = \sqrt{\overline{(x-\bar{x})^2}}$. The correlation *r* between two time series *x* and *y* is then defined as:

$$r = \overline{(x-\bar{x})(y-\bar{y})} / \sigma_x \sigma_y \qquad (8)$$

The autocorrelation r_τ is given by the correlation of the series with itself; use $x(t)$ and $x(t+\tau)$ as the two time series in Equation (8). It measures how well correlated x and y values are under different delays.

A good choice for the delay time to use when embedding the data should be *the first time the autocorrelation crosses zero*. This represents the lowest value of delay for which the x and y values are uncorrelated. However, since the autocorrelation function is a linear measure it does not provide accurate results in all situations. For Rossler data it provides a good choice of delay, but its results for Lorenz data are not entirely clear. For the Mackey Glass equation at default settings with timestep 0.25 autocorrelation should give a delay of about 50. For maps, it should pick a delay of one.

2.3.2. Mutual Information

The mutual information of two continuous random variables, X and Y with joint density function $f(x, y)$, is given by

$$I = I(X;Y) = \int f(x,y)\log\frac{f(x,y)}{f(x)f(y)}dxdy \qquad (9)$$

and an efficient method for estimating it from time series data is given in chapter four, section 6.

Inspection of the mutual information is a way to determine a useful delay time for plotting attractors. A good choice for the time delay T is one that, given $X(t)$, provides new information with measurement $X(t+T)$. Given a measurement of $X(t)$, this provides an estimate of how many bits on the average can be predicted about $X(t+T)$. A graph of $I(T)$ starts off very high (given a measurement $X(t)$, we know as many bits as possible about $X(t+0)=X(t)$). As T is increased, $I(T)$ decreases, then usually rises again. Fraser and Swinney suggest using the first minimum in $I(T)$ to select T, thereby revealing the first instance when the original data shares little information with its time delayed form .

The data is scaled and processed in the calculation, so the mutual information that is found may not be the same as that found using other software. However, the relative values of mutual information for different delays should be correct. One should find a reasonable value of the delay to use when embedding at *the first minimum of the mutual information*.

This is preferred over the autocorrelation function, because the mutual information can measure nonlinear correlations whereas the autocorrelation function only takes linear correlations into account. However, neither is definitive so it is recommended that both functions be used.

2.4. Dimension

2.4.1. Correlation

The Correlation Dimension routine provides an estimate of the correlation dimension using the Grassberger-Procaccia algorithm.[43-45] The idea is to construct a function $C(\varepsilon)$ that is the probability that two arbitrary points on the orbit are closer together than ε. This is usually done by calculating the separation between every pair of N data points and sorting them into bins of width dε proportional to ε. The correlation dimension is given by $D = d\log(C)/d\log(\varepsilon)$ in the limit $\varepsilon \to 0$, and $N \to \infty$. The Correlation Dimension is defined as the slope of the curve $C(\varepsilon)$ versus ε. C is the correlation of the data set, or the probability that any two points in the set are separated by a distance ε. A noninteger result for the correlation dimension indicates that the data is probably fractal. For the Henon map, the correlation dimension should be about 1.2. Using NDT, the correlation dimension can be measured two ways.

First, it can be measured from the slope of C(ε) versus ε. To obtain an estimate, move the two markers to various points on the graph. For too low or too high ε values, the results are inaccurate, so the slope must be measured in the midrange of the curve. Second, it can be measured from the points on the y-axis of the bottom plot. To obtain an estimate, move the markers to various points on this graph. Again, for too low or too high ε values, the results are inaccurate, so the estimate must also be measured in the midrange of the curve. A good value should be in the region where measurements of the dimension are most stable.

In order to speed up computation, we use the fact that computers store floating point numbers with a mantissa and integer exponent of 2. Thus it is possible to find the integer part of $\log_2(\varepsilon^2)$ by extracting the appropriate bits of the variable ε^2. Standard C libraries are available to do this.

The algorithm is order n^2 so it should be tried on a small data set first. By selecting quick, rough estimate, a fast estimate of the correlation dimension can be found. If the step size is increased to a value n, only every nth point is used in estimating correlation dimension. Thus a step size of 4 will make the program run roughly 4 times as fast. Nearby points in time may have a high correlation that can distort the estimate of correlation dimension. This problem may be overcome either by reducing the sample rate (using the reduce data routine) or by having a large number of orbits in relation to the data set size.

2.4.2. False Nearest Neighbors

The False Near Neighbors routine is a method of choosing the minimum embedding dimension of a one-dimensional time series. This method finds the nearest neighbor of every point in a given dimension, then checks to see if these are still close neighbors in one higher dimension.[46] The percentage of False Near Neighbors should drop to zero when the appropriate embedding dimension has been reached. The appropriate embedding dimension is a bit of a judgement call. If the number of fase near neighbors is not zero but is very small, the embedding dimension may still be suitable for most forms of analysis. For Lorenz data at default settings, an embedding of 4 should clearly demonstrate that a 3-dimensional embedding is appropriate. If the data is noisy, then it will not reach zero, but it should reach a low plateau at the appropriate embedding dimension. For the Henon map, the False Nearest Neighbors method should suggest an embedding dimension of 2.

An n-dimensional neighbor is considered false if the distance between the point and its neighbor in the $n+1^{st}$ dimension is greater than a criterion times the average distance between the point and its neighbor in each of the previous n dimensions.

For each vector $\mathbf{X}= (X_1,X_2,...X_n)$ in the time series look for its nearest neighbor $\mathbf{Y}=(Y_1,Y_2,...Y_n)$ in an n-dimensional space. Calculate the distance $\|\mathbf{X}-\mathbf{Y}\|$. Iterate both points and compute $R_n = \| X_{n+1} - Y_{n+1} \| / \|\mathbf{X}-\mathbf{Y}\|$. The first criterion is if Rn exceeds a user-specified threshold then this point is considered a false nearest neighbor. False nearest neighbors may also be identified by a second criterion, if $\|\mathbf{X}- \mathbf{Y}\| > Tolerance$, where *Tolerance* is a user specified tolerance times the size of the attractor. If either criterion holds, then a false neighbor has been identified. An appropriate embedding dimension is one where the percentage of false nearest neighbors is close to zero.

2.4.3. Generalized dimensions

First we define the generalized entropies. For q=0,1,2…,

$$
\begin{aligned}
H_q(\varepsilon) &= \frac{1}{1-q} \ln \sum_{i=1}^{N(\varepsilon)} P_i^q(\varepsilon) \ldots q \neq 1 \\
H_1(\varepsilon) &= -\sum_{i=1}^{N(\varepsilon)} P_i(\varepsilon) \ln P_i(\varepsilon)
\end{aligned}
\qquad (10)
$$

$P_i(\varepsilon)$ is the probability, out of all occupied boxes, of a point being in the ith occupied box. ε is the length of the side of a box. The generalized dimensions are then defined as

$$D(q) = -\lim_{\varepsilon \to \infty} \frac{H_q(\varepsilon)}{\ln \varepsilon} \qquad (11)$$

There are two ways to approximate the generalized dimensions for time series data. The first is if ε is sufficiently small such that the limit is approximately correct, but that we have enough data to get an accurate measurement for $H_q(\varepsilon)$. In which case we may use $D_q(\varepsilon) \approx -H_q(\varepsilon)/\ln \varepsilon$. $D(0)$ is the box counting dimension, $D(1)$ is the information dimension, and $D(2)$ is the correlation dimension.

However, the preferred method is to simply look at the slope of a plot of $H_q(\varepsilon)$ vs ln(ε).

2.5. Lyapunov Exponents

The determination of Lyapunov exponents from noisy, experimental data is a difficult task. Although many methods have been presented, there are also numerous examples of these methods breaking down when tested against real data, as well as questions concerning the validity of the methods. Thus the results of exponent determination were held to scrutiny. Criteria were established for identification of Lyapunov exponents.

There should be agreement between exponents as measured from different algorithms, and some measurement of error should be provided. Embedding parameters used in estimating exponents must be confirmed independently by other methods. The results should remain consistent under various parameter settings for exponent estimation. For flow data, a zero exponent should be clearly found. If it is possible to obtain both flow and sectioned data, estimation of exponents from each data set should be in agreement. The sum of all the exponents must be negative, and the sum of the positive exponents should be less than or equal to the metric entropy. In fact, for many cases they should be equal.[47] Under the proper conditions, the Lyapunov exponents should all, approximately, switch sign when measured from a time reversal of the data.[32]

The Lyapunov dimension may be defined as

$$D_\lambda = L + \frac{1}{|\lambda_{L+1}|}\sum_{j=1}^{L}\lambda_j \qquad (12)$$

where L is the maximum integer such that the sum of the L largest exponents is still non-negative. That is, $\sum_{j=1}^{L}\lambda_j \geq 0$, and $\sum_{j=1}^{L+1}\lambda_j < 0$. The Kaplan-Yorke conjecture[48] proposes that this is equal to the information dimension. Within error bounds, this seems to be true. Therefore, a final criterion is that the Lyapunov dimension estimates should agree with information dimension estimates.[49]

It is doubtful that all criteria can be satisfied unless one is dealing with a long, noise-free time series of low dimensional simulated data. Noise, high dimensionality and short time series length (few orbits or small number of points or both) negatively affect all methods of analysis. Some criteria, such as confirmation of embedding parameter choices are a virtual necessity before any calculation is made. Others, such as agreement between the Lyapunov dimension and information dimension, are very strong indicators that Lyapunov exponents have been reasonably determined. Still other criteria require calculations of quantities that are particularly difficult to compute from time series, such as metric entropy. If the dynamics of the system are unknown, results cannot be accepted or rejected based on definitive verification of the criteria. Rather, the worth of each criterion should be weighed and a subjective call as to the validity of Lyapunov exponent estimates should then be made. A more rigorous, statistical approach would be the use of Lyapunov exponents as a discriminating statistic between the time series and surrogate data. This has been attempted in the past, but the complexities of exponent estimation and the flaws in exponent estimation algorithms have made this method extremely difficult. Previous authors chose to reject the use of estimated Lyapunov exponents as discriminating statistics.[50]

CHAPTER THREE

SEARCHING AND SORTING METHODS

1. Introduction

Analysis of data often requires that all points in a data set that fulfill some criteria be selected. For analysis of multidimensional data, efficient searching becomes essential. If no sorting is performed, then searching may require that each data vector be examined. This is the brute force method, and it is a very time-consuming task. To find the nearest neighbor of each point in a data set of n vectors requires the comparison of $n(n-1)/2$ distances when using a brute force method. Considerable work has been done in devising searching and sorting routines that can be run far more efficiently. An extensive review of the multidimensional data structures that might be required is described by Samet, et al[51, 52]. Non-hierarchical methods, such as the use of grid files[53] and extendable hashing,[54] have been applied to multidimensional searching and analyzed extensively. In many areas of research the kd-tree has become accepted as one of the most efficient and versatile methods of searching. This and other search mechanisms have been applied extensively throughout the field of computational geometry.[55, 56] However, little work has been made within the nonlinear time series analysis community in this regard. It is our hope that we can shed some light on the beneficial uses of kd-trees in this field, and how they can be adapted and modified to the peculiarities of time series analysis from a nonlinear dynamics perspective. In this chapter, a variety of searching and sorting routines are investigated and compared. It is shown when different routines may be preferred, how the routines may be optimized, and how they may be tailored for a variety of different situations.

There are a variety of searches that are often performed on multidimensional data.[57] Ideally, a sorting and searching routine should be able to perform all the common types of searches, and be adaptable enough so that it may be made to perform any search. Of course, even for an unusual search, the searching and sorting routine should perform at least as efficiently as the brute force method.

Perhaps the most common type of search, and one of the simplest, is the nearest neighbor search. This search involves the identification of the nearest vector in the data set to some specified vector, known as the search vector. The search vector may or may not also be in the data set. Expansions on this type of search include the radial search, where one wishes to find all vectors within a given distance of the search vector, and the weighted search, where one wishes to find the nearest *N* vectors to the search vector.

Each of these searches (weighted, radial and nearest neighbor) may come with further restrictions. For instance, points or collections of points may be excluded from the search. Additional functionality may also be required. The returned data may be ordered from closest to furthest from the search vector, and the sorting and searching may be required to handle the insertion and deletion of points. That is, if points are deleted from or added to the data, these additional points should be added or deleted to the sort so that they can be removed or included in the search. Such a feature is essential if searching is to performed with real-time analysis.

One clear example of how search methods may need to be modified for nonlinear dynamics comes out of delay coordinate embedding theory. Here one is presented with data consisting of a single series of numbers (integer or floating point) that is assumed to represent one dimension of an *n*-dimensional system. The dynamics of that systems may be reconstructed by constructing multidimensional vectors from the original data via the use of a time delay τ. The original time series $X_1, X_2, ...X_N$ is thus transformed into *n*-dimensional vectors

$$\begin{aligned}\overline{Y_1} &= (X_1, X_{1+\tau}, X_{1+2\tau}, ...X_{1+(n-1)\tau})\\ \overline{Y_2} &= (X_2, X_{2+\tau}, X_{2+2\tau}, ...X_{2+(n-1)\tau}) \qquad (1)\\ &...\end{aligned}$$

To do this efficiently, especially in regards to memory use, any sorting and searching should work only by accessing the original, one-dimensional data, instead of acting on *n*-dimensional copies of the data. This requires modifications to the implementation of any multidimensional search method. Ideally these modifications should be implemented such that they are small, encapsulated, and allow for further modifications at a later date. Each of the search methods discussed will be compared in their ability to adapt to delay coordinate embedded data.

2. The Kd-tree

2.1. Method

The K-dimensional binary search tree (or kd-tree) is a highly adaptable, well researched method for searching multidimensional data. This tree was first introduced in Bentley, et al.,[58] studied extensively in Ref. 59 and a highly efficient and versatile implementation was described in Ref. 60. It is this second implementation, and variations upon it, that we will be dealing with here.

There are two types of nodes in a kd-tree, the buckets (or terminal nodes) and the internal nodes.The internal nodes have two children, a left and a right son. These children represent a k-dimensional partition of the hyperplane. Records on one side of the partition are stored in the left subtree, and on the other side are records stored in the right subtree. The terminal nodes are buckets which contain up to a set amount of points. A one dimensional kd-tree would in effect be a simple quicksort.

The building of the kd-tree works by first determining which dimension of the data has the largest spread, i.e., difference between the maximum and the minimum. The sorting at the first node is then performed along that dimension. A quickselect algorithm, which runs in order *n* time, finds the midpoint of this data. The data is then sorted along a branch depending on whether it is larger or smaller than the midpoint. This succeeds in dividing the data set into two smaller data sets of equal size. The same procedure is used at each node to determine the branching of the smaller data sets residing at each node. When the number of data points contained at a node is smaller than or equal to a specified size, then that node becomes a bucket and the data contained within is no longer sorted.

Consider the following data.

A.	(7,-3)
B.	(4,2)
C.	(-6,7)
D.	(2,-1)
E.	(8,0)
F.	(1,-8)
G.	(5,-6)
H.	(-8,9)
I.	(9,8)
J.	(-3,-4)

Table 1. Sample data used to depict the sorting and searching methods.

Figure 2 depicts an example partition in 2 dimensions for this data set. At each node the cut dimension and the cut value (the median of the corresponding data) are stored. The bucket size has been chosen to be one. The corresponding partitioning of the plane is given in Figure 3. We note that this example comes from a larger data set and thus does not appear properly balanced. This data set will be used as an example in the discussion of other methods.

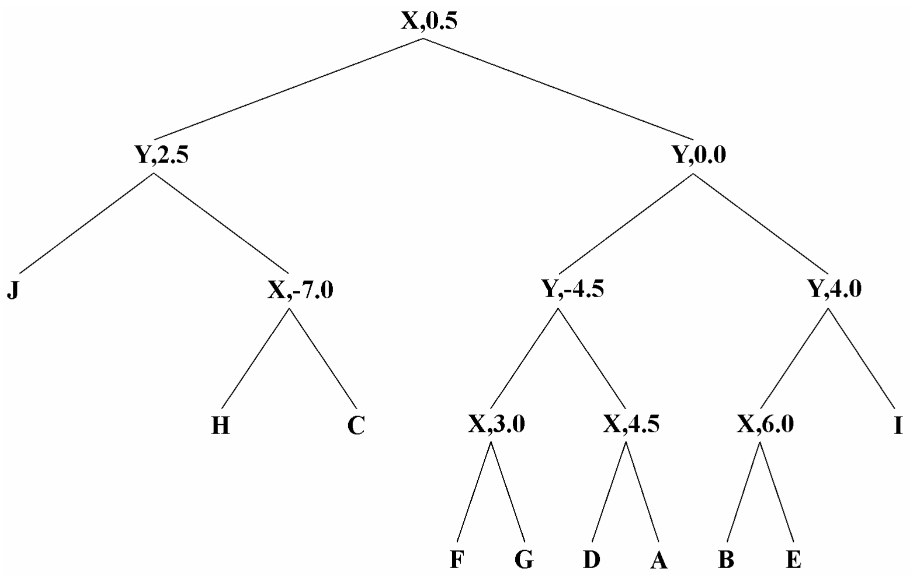


Figure 2. The kd-tree created using the sample data.

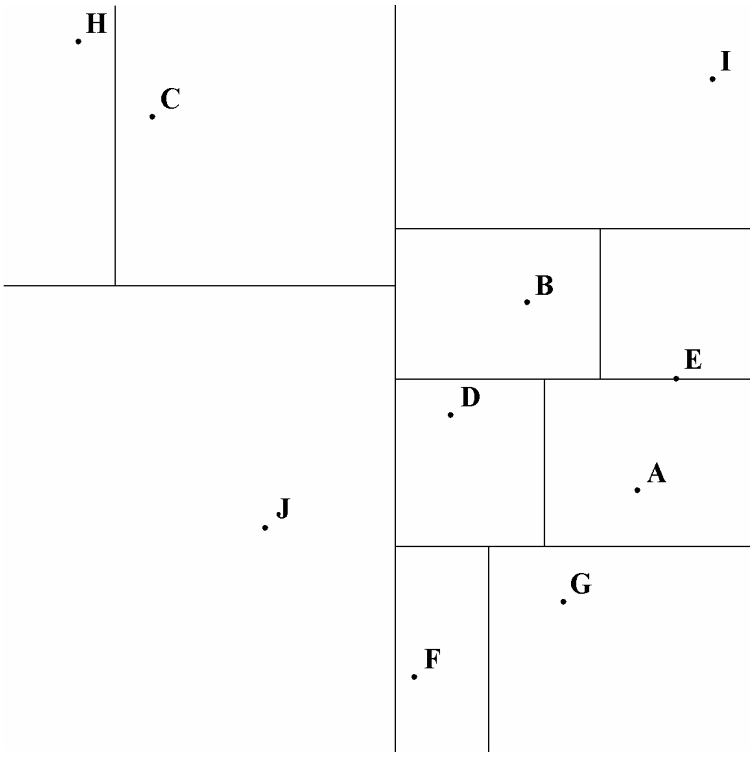


Figure 3. The sample data as partitioned using the kd-tree method.

A nearest neighbor search may then be performed as a top-down recursuve traversal of a portion of the tree. At each node, the query point is compared with the cut value along the specified cut dimension. If along the cut dimension the query point is less than the cut value, then the left branch is descended. Otherwise, the right branch is descended. When a bucket is reached, all points in the bucket are compared to see if any of them is closer than the distance to the nearest neighbor found so far. After the descent is completed, at any node encountered, if the distance to the closest neighbor found is greater than the distance to the cut value, then the other branch at that node needs to be descended as well. Searching stops when no more branches need to be descended.

The drawbacks of the kd tree, while few, are transparent. First, if searching is to be done in many different dimensions, either a highly inefficient search is used, or additional search trees must be built. Also the method is somewhat memory intensive. In even the simplest kd tree, a number indicating the cutting value is required at each node, as well as an ordered array of data (similar to the quicksort). If pointers to the parent node or principal cuts are used then the tree must contain even more information at each node. Although this increase may at first seem unimportant, one should note that experimental data typically consists of 100000 floating points or more. In fact, some data analysis routines require a minimum of this many points. If the database is also on that order, then memory may prove unmanageable for many workstation computers.

Bentley recommends the use of parent nodes for each node in a tree structure. A search may then be performed using a bottom-up approach, starting with the bucket containing the search point and searching through a small number of buckets until the appropriate neighbors have been found. For nearest neighbor searches this reduces computational time from O(log m) to O(1). This however, does not immediately improve on search time for finding near neighbors of points *not* in the database. Timed trials indicated that the increased speed due to bottom-up (as opposed to top-down) searches was negligible. This is because almost all computational time is spent in distance calculations, and the reduced number of comparisons is negligible.

3. The KTree

K-trees are a generalization of the single-dimensional M-ary search tree. As a data comparative search tree, a K-tree stores data objects in both internal and leaf nodes. A hierarchical recursive subdivision of the k-dimensional search space is induced with the space partitions following the locality of the data. Each node in a K-tree contains K=2^k child pointers. The root node of the tree represents the entire search space and each child of the root represents a K-*ant* of the parent space.

One of the disadvantages of the K-tree is its storage space requirements. In a standard implementation, as described here, a tree of N k-dimensional vectors requires a minimum of $(2^k + k)\cdot N$ fields. Only N-1 of the 2^k branches actually point to a node. The rest point to NULL data. For large k, this waste become prohibitive.

Consider the case of two-dimensional data (k=2, K=4). This K-tree is known as a quadtree, and is a 4-ary tree with each node possessing 4 child pointers. The search space is a plane and the partitioning induced by the structure is a hierarchical subdivision of the plane into disjoint quadrants. If the data consists of the 10 vectors described in the above section, then the corresponding tree is depicted in Figure 4 and the partitioning of the plane in Figure 5. Note that much of the tree is consumed by null pointers.

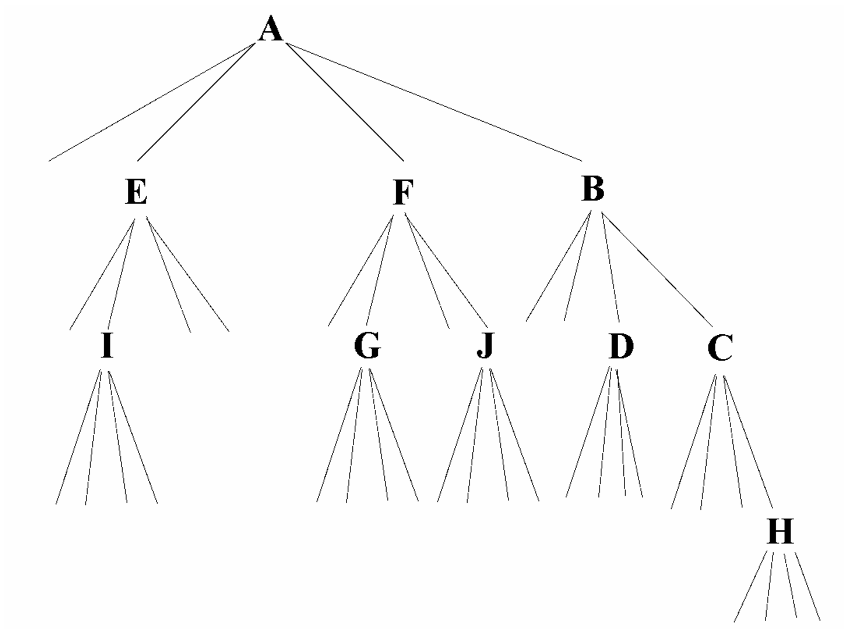


Figure 4. The ktree created using the sample data.

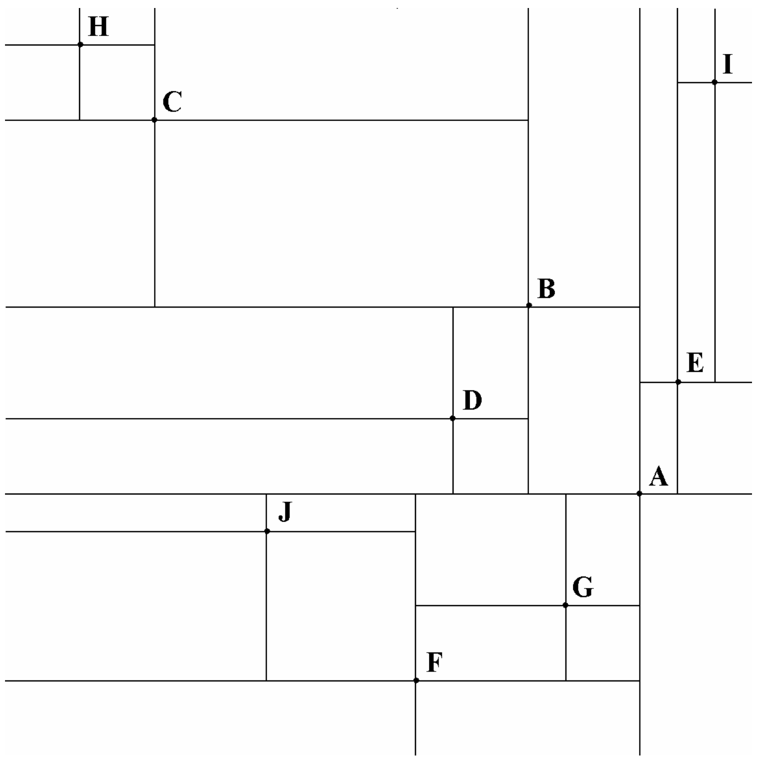


Figure 5 The sample data as partitioned using the k-tree method.

Searching the tree is a recursive two-step process. A cube that corresponds to the bounding extent of the search sphere is intersected with the tree at each node encountered. The bounds of this cube are maintained in a *k*-dimensional range array. This array is initialized based on the search vector. At each node, the direction of search is determined based on this intersection. A search on a child is discontinued if the region represented by the child does not intersect the search cube. This same general method may be applied to weighted, radial, and nearest neighbor searches. For radial searches, the radius of the search sphere is fixed. For nearest neighbor searches it is doubled if the nearest neighbor has not been found, and for weighted searches it is doubled if enough neighbors have not been found.

4. Multidimensional quicksort

4.1. Description

For many analyses, one wishes to search only select dimensions of the data. A problem frequently encountered is that most sort methods are optimized for only one type of search and a different sort would need to be performed for each search based on a different dimension or subset of all the dimensions.

An example of this comes out of delay coordinate embedding theory, as described in the Introduction. The choice of an embedding dimension is never a clear one. There are methods of finding a suitable embedding dimension that consist of comparing the results of various embedding dimensions and then the lowest dimension for which some criterion holds is typically chosen ([61] and references therein). Also one may wish to compare the results of analyses under different embeddings.

Using conventional methods, this would require sorting and searching the data for each embedding. The overhead in building a search tree or sorting the data is considerable. It is typically assumed that the data will only be sorted once, and so speed in the sort is willingly sacrificed in favor of devising an optimal search. But here, conventional sorts may perform poorly. If changing the embedding dimension requires a new sort, then an optimal search may still be slow due to the many long sort times. For analyses that require many different embedding dimensions, the goals are modified. One would like to be able to sort the data quickly, and in such a way that a variety of searches can be performed.

4.2. Method

The data is described as a series of *N* vectors where each vector is *n* dimensional, $\overline{X_k} = (X_k^1, X_k^2, \ldots X_k^n)$. A quicksort is performed on each of the *n* dimensions. The original array is not modified. Instead, two new arrays are created for each quicksort. The first is the quicksort array, an integer array where the value at position *k* in this array is the position in the data array of the *k*th smallest value in this dimension. The second array is the inverted quicksort. It is an integer array where the value at position *k* in the array is the position in the quicksort array of the value *k*. Keeping both arrays allows one to identify both the location of a sorted value in the original array, and the location of a value in the sorted array. Thus, if $\overline{X_1}$ has the second smallest value in the third dimension, then it may be represented as $\overline{X_{2,3}}$. The value stored at the second index in the quicksort array for the third dimension will be 1, and the value stored at the first index in the inverted quicksort array for the third dimension will be 2. Note that the additional memory overhead need not be large. For each floating point value in the original data, two additional integer values are stored, one from the quicksort array and one from the inverted quicksort array.

We begin by looking at a simple case and showing how the method can easily be generalized. We consider the case of two dimensional data, with coordinates x and y, where we make no assumptions about delay coordinate embeddings or uniformity of data.

Suppose we wish to find the nearest neighbor of the vector $\overline{A} = (x, y)$ .If this vector's position in the x-axis quicksort is *i* and its position in the y-axis quicksort is *j* (*i* and *j* are found using the inverted quicksorts), then it may also be represented as

$$\overline{A} = \overline{A_{i,x}} = (x_{i,x}, y_{i,x}) = \overline{A_{j,y}} = (x_{j,y}, y_{j,y}).$$

Using the quicksorts, we search outward from the search vector, eliminating search directions as we go. Reasonable candidates for nearest neighbor are the nearest neighbors on either side in the x-axis quicksort, and the nearest neighbors on either side in the y-axis quicksort. The vector $\overline{A_{i-1,x}} = (x_{i-1,x}, y_{i-1,x})$ corresponding to position *i*-1 in the x-axis quicksort is the vector with the closest x-coordinate such that $x_{i-1,x} < x$. Similarly, the vector $\overline{A_{i+1,x}} = (x_{i+1,x}, y_{i+1,x})$ corresponding to *i*+1 in the x-axis quicksort is the vector with the closest x-coordinate such that $x_1 > x$. And from the y-axis quicksort, we have the vectors $\overline{A_{j-1,y}} = (x_{j-1,y}, y_{j-1,y})$ and $\overline{A_{j+1,y}} = (x_{j+1,y}, y_{j+1,y})$. These are the four vectors adjacent to the search vector in the two quicksorts. Each vector's distance to the search vector is calculated and we store the minimal distance and the corresponding minimal vector. If $|x_{i-1,x} - x|$ is greater than the minimal distance, then we know that all vectors $\overline{A_{i-1,x}}$, $\overline{A_{i-2,x}}$,... $\overline{A_{1,x}}$ must also be further away than the minimal vector. In that case, we will no longer search in decreasing values on the x-axis quicksort. We would also no longer search in decreasing values on the x-axis quicksort if $\overline{A_{1,x}}$ has been reached. Likewise, if $|x_{i+1,x} - x|$ is greater than the minimal distance, then we know that all vectors $\overline{A_{i+1,x}}, \overline{A_{i+2,x}}, \ldots \overline{A_{N,x}}$ must also be further away than the minimal vector. If either that is the case or $\overline{A_{N,x}}$ has been reached then we would no longer search in increasing values on the x-axis quicksort. The same rule applies to $|y_{j-1,y} - y|$ and $|y_{j+1,y} - y|$.

We then look at the four vectors, $\overline{A_{i-2,x}}, \overline{A_{i+2,x}}, \overline{A_{j-2,y}}$ and $\overline{A_{j+2,y}}$. If any of these is closer than the minimal vector, then we replace the minimal vector with this one, and the minimal distance with this distance. If $|x_{i-2,x} - x|$ is greater than the minimal distance, then we no longer need to continue searching in this direction. A similar comparison is made for $|x_{i+2,x} - x|, |y_{j-2,y} - y|$ and $|y_{j+2,y} - y|$.

This procedure is repeated for $\overline{A_{i-3,x}}, \overline{A_{i+3,x}}, \overline{A_{j-3,y}}$ and $\overline{A_{j+3,y}}$, and so on, until all search directions have been eliminated. We find which of these four vectors is closest to the search vector. We then search the next four closest points. If any of these points is further away in its direction of search than the minimal distance, then we can eliminate that direction from further search. Also, if we reach the end of the data set in any direction, then we can eliminate that direction. We find the distance of the four points from our point of interest and, if possible, replace the minimal distance. We then proceed to the next four points and proceed this way until all directions of search have been eliminated.

The minimal vector must be the nearest neighbor, since all other neighbor distances have either been calculated and found to be greater than the minimal distance, or have been shown that they must be greater than the minimal distance.

Extension of this algorithm to higher dimensions is straightforward. In n dimensions there are $2n$ possible directions. Thus $2n$ immediate neighbors are checked. A minimal distance is found, and then the next $2n$ neighbors are checked. This is continued until it can be shown that none of the $2n$ directions can contain a nearer neighbor.

It is easy to construct data sets for which this is a very inefficient search. For instance, if one is looking for the closest point to (0,0) and one were to find a large quantity of points residing outside the circle of radius 1 but inside the square of side length 1 then all these points would need to be measured before the closer point at (1,0) is considered. However, similar situations can be constructed for most multidimensional sort and search methods, and preventative measures can be taken.

The power of this method comes into play when dealing with a delay coordinate embedding. In this case, one can sort just the original data set and perform all the searches from that sort. Neighbors to a point in any dimension are found from the sorted array of the original data. The same method of search is used. For example, consider 2 dimensional data constructed with a delay of one. The reconstructed data then ranges from (X_0, X_1) to (X_{N-2}, X_{N-1}) To find the near neighbors of (X_n, X_{n+1}) we find the near neighbors of X_n, from the sorted array and the near neighbors of X_{n+1} from the same array. A near neighbor using X_0 in the second coordinate or X_{N-1} in the first coordinate can immediately be eliminated from consideration because it is not used in the reconstruction. Such a point would then be skipped over in our search. So searches can be performed using any number of delays and any number of embedding dimensions from only one quicksorted array.

5. The box-assisted method

The box-assisted search method was described by Schreiber, et al.[62] as a simple multidimensional search method for nonlinear time series analysis. A grid is created and all the vectors are sorted into boxes in the grid. Figure 6 demonstrates a two-dimensional grid that would be created for the sample data. Searching then involves finding the box that a point is in, then searching that box and all adjacent boxes. If the nearest neighbor has not been found, then the search is expanded to the next adjacent boxes. The search is continued until all required neighbors have been found.

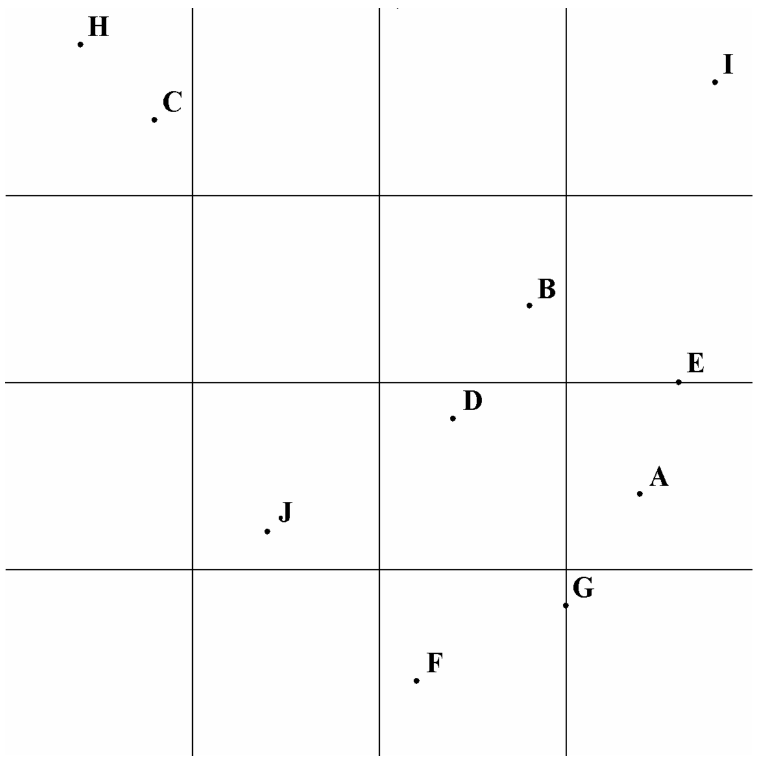


Figure 6 The sample data set as gridded into 16 boxes in two dimensions, using the box-assisted method.

A benefit of this method is that it fits into the class of bin based sorts. Hence the sort time is reduced from order O(*n*log *n*), for tree based methods, to O(*n*). One of the difficulties with this method is the determination of the appropriate box size. The sort is frequently tailored to the type of search that is required, since a box size is required and the preferred box size is dependent on the type of search to be done. However, one usually has only limited a priori knowledge of the searches that may be performed. Thus the appropriate box size for one search may not be appropriate for another. If the box size is too small, then many boxes are left unfilled and many boxes will need to be searched. This results in both excessive use of memory and excessive computation.

The choice of box dimensionality may also be problematic. Schreiber, et al.[62] suggest 2 or 3 dimensional boxes. However, this may lead to inefficient searches for high dimensional data. Higher dimensional data may still be searched although many more boxes are often needed in order to find a nearest neighbor. On the other hand, using higher dimensional boxes will exacerbate the memory inefficiency. In the benchmarking results shown later in the chapter, we will consider both two and three dimensional boxes.

6. The multidimensional binary representation

This method was designed with specific types of analysis (calculation of entropy, fractal dimension or generation of symbol dynamics) in mind. Consider a time series X_1, X_2, X_3…, X_N embedded in n dimensions with delay τ, as described in Eqn. (1). This creates a series of vectors, $\overline{Y_1}, \overline{Y_2}, ...$ It is these embedded vectors that one would like to sort and search, not simply the original time series.

The generalized procedure is as follows:

First scale all points in the time series to lie between 0 and $2^{32}-1$. This is the range of a long unsigned integer on most computers.

Sort the data using a quicksort based on the following comparison scheme: for vectors $\overline{Y_i} = (X_i, X_{i+\tau}, X_{i+2\tau}, ...X_{i+(n-1)\tau})$ and $\overline{Y_j} = (X_j, X_{j+\tau}, X_{j+2\tau}, ...X_{j+(n-1)\tau})$, created from a delay coordinate embedding, one determines whether $\overline{Y_i} < \overline{Y_j}$, $\overline{Y_i} > \overline{Y_j}$, or $\overline{Y_i} = \overline{Y_j}$, by making the following comparisons.

If the first bit in the binary representation of X_i is less than the first bit in the binary representation of Xj, then $\overline{Y_i} < \overline{Y_j}$. Else if the first bit in the binary representation of X_i is greater than the first bit in the binary representation of Xj, then $\overline{Y_i} > \overline{Y_j}$.

If these first bits are equal, then we make the same comparison on the first bits of $X_{i+\tau}$ and $X_{j+\tau}$.

If these are equal then we make the comparison for $X_{i+2\tau}$ and $X_{j+2\tau}$, and so on up to $X_{i+(n-1)\tau}$ and $X_{j+(n-1)\tau}$.

If all of these are equal, then we compare the second bits of X_i and X_j, and then, if necessary the second bits of $X_{i+\tau}$ and $X_{j+\tau}$, and so on up to $X_{i+(n-1)\tau}$ and $X_{j+(n-1)\tau}$. Again, if these are all equal we move on to the third bits of the binary representation. This procedure is completed until all bits of the binary representations of $\overline{Y_i}$ and $\overline{Y_j}$ have been compared. If they are equal, then we know that these two vectors are equal up to the resolution of the computer, and so they do not need to be ordered.

Notice that this means that the comparison of two vectors requires anywhere from 1 to $n \cdot 32$ binary comparisons. A uniform distribution implies that most vectors could be sorted using just a few binary comparisons, but highly clustered data would require far more, since for many vectors the binary representations would be quite similar.

For a two-dimensional embedding, sorted scaled data may be ordered in the following manner.

```
(0,0) -> 00 00 00 …00 00 00
(0,1) -> 00 00 00 …00 00 01
(1,0) -> 00 00 00 …00 00 10
(0,2) -> 00 00 00 …00 01 00
(0,3) -> 00 00 00 …00 01 01
(2,1) -> 00 00 00 …00 10 01
(4,2) -> 00 00 00 …10 01 00
(4,2) -> 00 00 00 …10 01 00
(4,3) -> 00 00 00 …00 11 11
(7,4) -> 00 00 00 …11 10 10
(7,7) -> 00 00 00 …11 11 11
```

where the first bit states if the first coordinate ranges from 0-3 or 4-7, the second bit if the second coordinate ranges from 0-3 or 4-7. The first and third bit together describe whether the first coordinate ranges from 0-1, 2-3, 4-5 or 6-7, and so on.

It should be noted that the vectors do not need to be stored in memory, since they can always be reconstructed from the original time series. Likewise, the scaling of the data may be performed within this step, so that the scaled data need not be stored either. The procedure also works if the data is not from a delay coordinate embedding. It can be generalized to any multidimensional time series.

The data is gridded into *n*-dimensional boxes of equal length ε, such that all vectors lie within these boxes. Figure 7 depicts such a grid for data from the Henon map. If a box is labeled *i*, then it has an associated probability, $P_i(\varepsilon)$, that a vector on the attractor will reside within this box. ε represents which of the 32 levels of the binary representation we are considering. The data is now ordered so that all the vectors within a box of any of the εs (1, ½, … 1/ (2^32 –1)) are ordered consecutively.

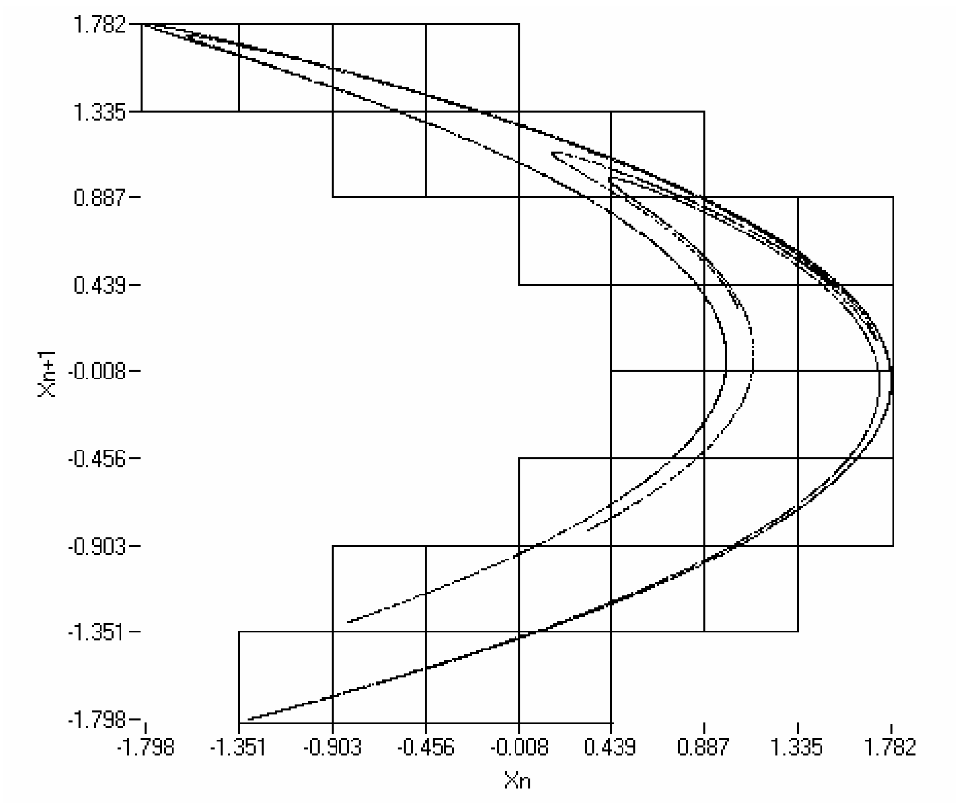


Figure 7. A gridding of the Henon attractor using the multidimensional binary representation. An 8 by 8 grid has been created where each square is represented as a 2d, 3 bit vector (00 00 00 to 11 11 11). Only 31 of the 64 possible squares are occupied.

This suggests an accurate method to keep track of how many vectors lie inside of each box of any size. One can scroll through the data and count how many vectors are in each box. That is, a vector is compared with the previous vector. There are 32 levels. If on any of the dimensions of the two vectors, the binary representations of the two vectors on that level are not equal, then we know that these vectors reside in different boxes.

An adaptation of this method that allows for insertion into the sorted data could be highly effective for the case of when an arbitrary number of data points can be generated. However, this is typically not possible for the case of experimental data.

No use is made in the above method of the sequencing of the original data. Consider a vector X_n residing in the box *i*. If in addition to the quicksort a reverse sort is maintained that maps a vector in the sorted data to the corresponding vector in the time series, then one can determine the vector X_{n+1}, and hence find the box *j* that contains this vector. Thus allowable sequences of symbol dynamics can be quickly determined. Benefits of this can be seen in the case of noisy data. Here, one sets an ε based on the estimated noise level of the data. The presence of noise implies that the estimate $f\left(X_n\right) = X_{n+1}$, may be incorrect. However, one can state that it might be mapped to a location to which any of the vectors within box *i* are mapped. Thus we have the information necessary to estimate a probability transition matrix $[P_{ij}]$, where $P_{ij} = \Pr\{X_{n+1} = j \mid X_n = i\}$. To estimate any P_{ij}, we find the number of vectors in box *i* that get mapped to a vector in box *j*, and then divide by the total number of vectors in box *j*.

7. The uniform KTree

This sort is a variation on the K-Tree that is used in estimating the generalized mutual information of a multidimensional time series. It is an extension of and variation upon a sort described by Fraser and Swinney.[42] Its application is described in detail in Chapter 4. The data is described as a series of vectors $\overline{X_1}, \overline{X_2}, ..., \overline{X_N}$ where each vector is *n* dimensional, $\overline{X_i} = (X_i^1, X_i^2, ...X_i^n)$, and the number of vectors, *N*, is a power of two. The vectors are assumed to be sampled uniformly in time. That is, at each time interval Δt, a new vector is sampled, and Δt is a constant. Alternately, the data may be described as *n* time series data sets, $\overline{\overline{X_1}}, \overline{\overline{X_2}}, ..., \overline{\overline{X_n}}$, where each data set consists of *N*, data points, $\overline{\overline{X_i}} = (X_1^i, X_2^i, ...X_N^i)$. We will use both of these notations depending on the situation.

One important issue is determining the range of possible values for a point $X_i^j \in \overline{\overline{X_j}}$. It is somewhat of an arbitrary choice, since the data set is confined to at most *N* distinct values, yet the full range of values may be infinite. The algorithm involves gridding the *n* dimensional space into *n* dimensional hypercubes and treating each occupied hypercube as an allowable symbol. The data is gridded into equiprobable partitions. This is seen in Figure 8 in the case of 16 2-dimensional data points. This has the added benefit that $p(x_1) = p(x_2)... = p(x_n)$.

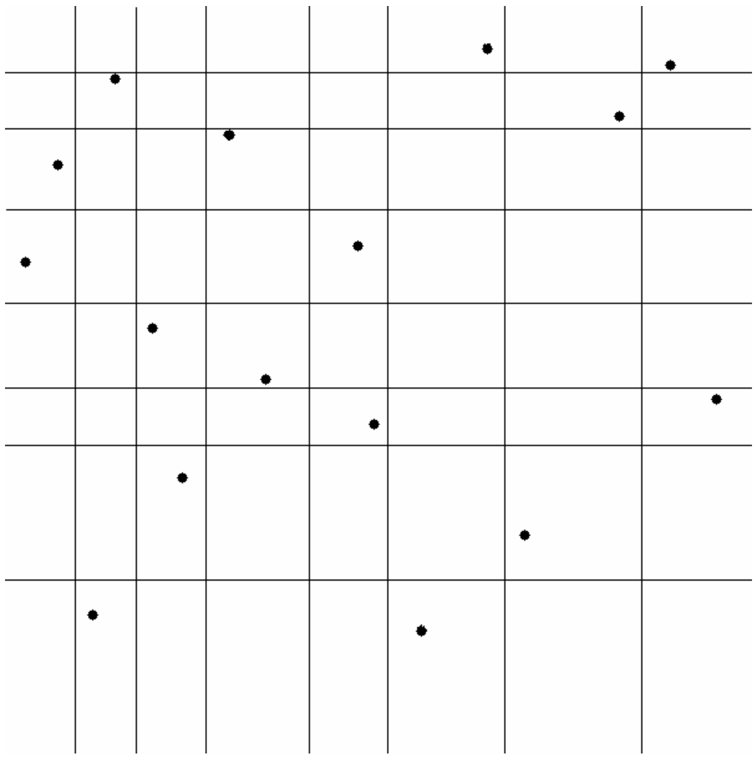


Figure 8. An example grid used to compute the mutual information of two time series. The cuts are made such that a point may occupy each row or column with equal probability.

A change of variables is first performed on the data, so that each time series $\overline{\overline{X_i}} = (X_1^i, X_2^i, ...X_N^i)$ is transformed into integers, $\overline{\overline{Y_i}} = (Y_1^i, Y_2^i, ...Y_N^i)$ such that the integer values fill the range 0 to *N*-1 and $X_j^i < X_k^i$ implies $Y_j^i < Y_k^i$. Thus the change of variables gives vectors $\overline{Y_1}, \overline{Y_2}, ..., \overline{Y_N}$. This data is used to create a $2^n -$ary tree (the *n*=2 tree is depicted in Figure 9). At each level of the tree a vector is sorted into one of the 2^n branches, depending on the value of each vector coordinate. At each node in the tree, the number of elements in or below that node is stored.

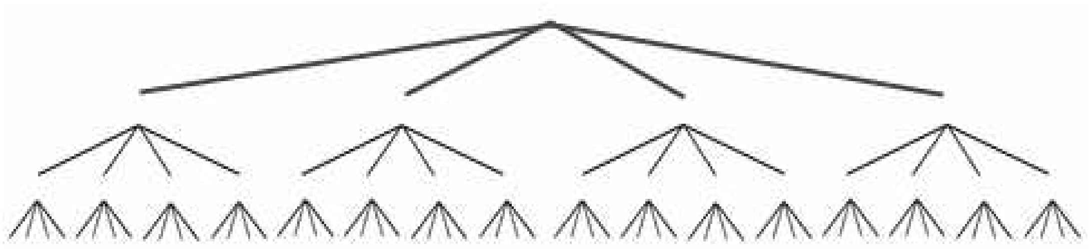


Figure 9. A portion of a 2-ary tree (quadtree) used to compute the mutual information of two time series.

Recall that N is a power of two (if not, then data points are removed), $N = 2^K$. The levels of the tree define successive partitions of the n-dimensional space, $G_0, G_1, \ldots G_{K-1}$. For a given level of the tree G_m, the space is partitioned into 2^{mj} hypercubes, $R_m(0), R_m(1), \ldots R_m(2^n m - 1)$ such that $R_m(j)$ may be partitioned into the hypercubes $R_{m+1}(2^n j), R_{m+1}(2^n j + 1), \ldots R_m(2^n j + 2^n - 1)$. Each hypercube has an associated probability $P(R_m(j))$, which is estimated by its frequency, the number of vectors in that hypercube divided by the total number of vectors, $N_m(j) /N$. This allows one to compute probabilities for each level of the tree. $P_i(R_m(j))$ is the probability of finding the ith coordinate of a vector to reside in the same partition along the ith direction as $R_m(j)$. Due to the equiprobable nature of the partitions, $P_i(R_m(j)) = 1/2^m$ for all i and j.

8. Benchmarking and comparison of methods

In this section we compare the suggested sorting and searching methods, namely the box assisted method, the KD-Tree, the K-tree, and the multidimensional quicksort. All of these methods are, in general, preferable to a brute force search.

However, computational speed is not the only relevant factor. Complexity, memory use, and versatility of each method will also be discussed. The versatility of the method comes in two flavors- how well the method works on unusual data and how adaptable the method is to unusual searches. The multidimensional binary representation and the uniform K-Tree, described in the previous two sections, are not compared with the others because they are specialized sorts used only for exceptional circumstances.

8.1. Benchmarking of the Kd-tree

One benefit of the kd-tree is its rough independence of search time on data set size. Figure 10 compares the average search time to find a nearest neighbor with the data set size. For large data set size, the search time has a roughly logarithmic dependence on the number of data points. This is due to the time it takes to determine the search point’s location in the tree. If the search point were already in the tree, then the nearest neighbor search time is reduced from O(log *n*) to O(1). This can be accomplished with the implementation of Bentley's suggested use of parent pointers for each node in the tree structure.[60]. This is true even for higher dimensional data, although the convergence is much slower.

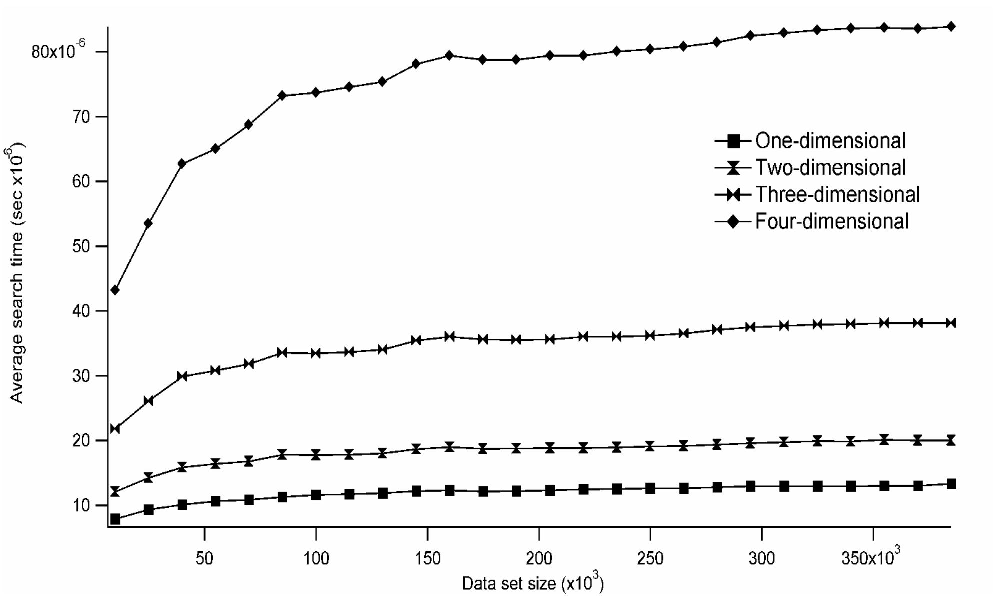


Figure 10. The dependence of average search time on data set size.

In Figure 11, the kd-tree is shown to have an exponential dependence on the dimensionality of the data. This is an important result, not mentioned in other work providing diagnostic tests of the kd-tree.[60, 63] It implies that kd-trees become highly inefficient for high dimensional data. It is not yet known what search method is most preferable for neighbor searching in a high dimension (greater than 4).

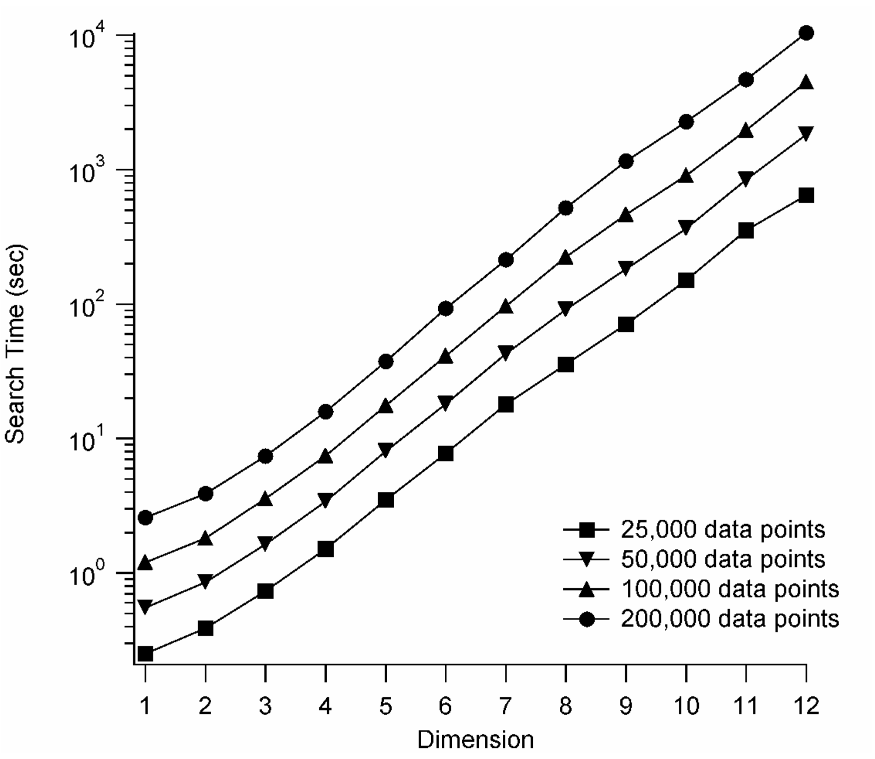


Figure 11. A log plot of search time vs dimension.

Figure 12 shows the relationship between the average search time to find *n* neighbors of a data point and the value *n*. In this plot, 10 data sets were generated with different seed values and search times were computed for each data set. The figure shows that the average search time is almost nearly linearly dependent on the number of neighbors *n*. Thus a variety of searches (weighted, radial, with or without exclusion) may be performed with only a linear loss in speed.

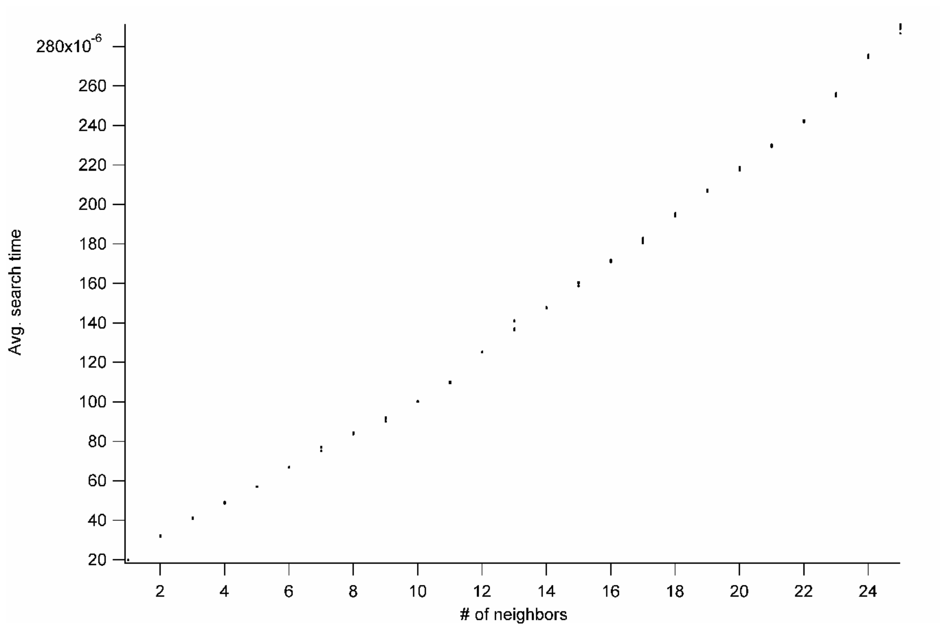


Figure 12. A plot of the average search time to find *n* neighbors of a data point, as a function of *n*. This shows that the relationship remains nearly linear over small *n*. 10 random data sets were generated with different seed values and the average time it takes to find a given number of neighbors was computed for each data set.

The drawbacks of the kd tree, while few, are transparent. First, if searching is to be done in many different dimensions, either a highly inefficient search is used, or additional search trees must be built. Also the method is somewhat memory intensive. In even the simplest kd tree, a number indicating the cutting value is required at each node, as well as an ordered array of data (similar to the quicksort). If pointers to the parent node or principal cuts are used then the tree must contain even more information at each node. Although this increase may at first seem unimportant, one should note that experimental data typically consists of 100000 floating points or more. In fact, some data analysis routines require a minimum of this many points. If the database is also on that order, then memory may prove unmanageable for many workstation computers.

We have implemented the kd-tree as a dynamic linked library consisting of a set of fully functional object oriented routines. In short, they consist of the functions

```
Create(*phTree, nCoords, nDims, nBucketSize,*aPoints);
FindNearestNeighbor(Tree,*pSearchPoint, *pFoundPoint);
FindMultipleNeighbors(Tree, *pSearchPoint, *pnNeighbors, *aPoints);
FindRadialNeighbors(Tree, *pSearchPoint, radius, **paPoints,
*pnNeighbors);
ReleaseRadialNeighborList(*aPoints);
Release(Tree);
```

8.2. Comparison of methods

The kd-tree implementation was tested in timed trials against the multidimensional quicksort and the box-assisted method. In Figure 13 through Figure 16 we depict the dependence of search time on data set size for one through four dimensional data, respectively. In Figure 13, the multidimensional quicksort reduces to a one dimensional sort and the box assisted method as described by [62] is not feasible since it requires that the data be at least two dimensional. We note from the slopes of these plots that the box-assisted method, the KDTree and the KTree all have an O(n log n) dependence on data set size, whereas the quicksort based methods have approximately O($n^{1.5}$) dependence on data set size for 2 dimensional data and O($n^{1.8}$) dependence on data set size for 3 or 4 dimensional data. As expected, the brute force method has O(n^2) dependence.

Despite its theoretical O(n log n) performance, the ktree still performs far worse than the box-assisted and kdtree methods. This is because of a large constant factor worse performance that is still significant for large data sets (64,000 points). This constant worse performance relates to the poor balancing of the ktree. Whereas for the kdtree, the data may be permuted so that cut values are always chosen at medians in the data, the ktree does not offer this option because there is no clear multidimensional median. In addition, many more branches in the tree may need to be searched in the ktree because at each cut, there are 2^k instead of 2 branches.

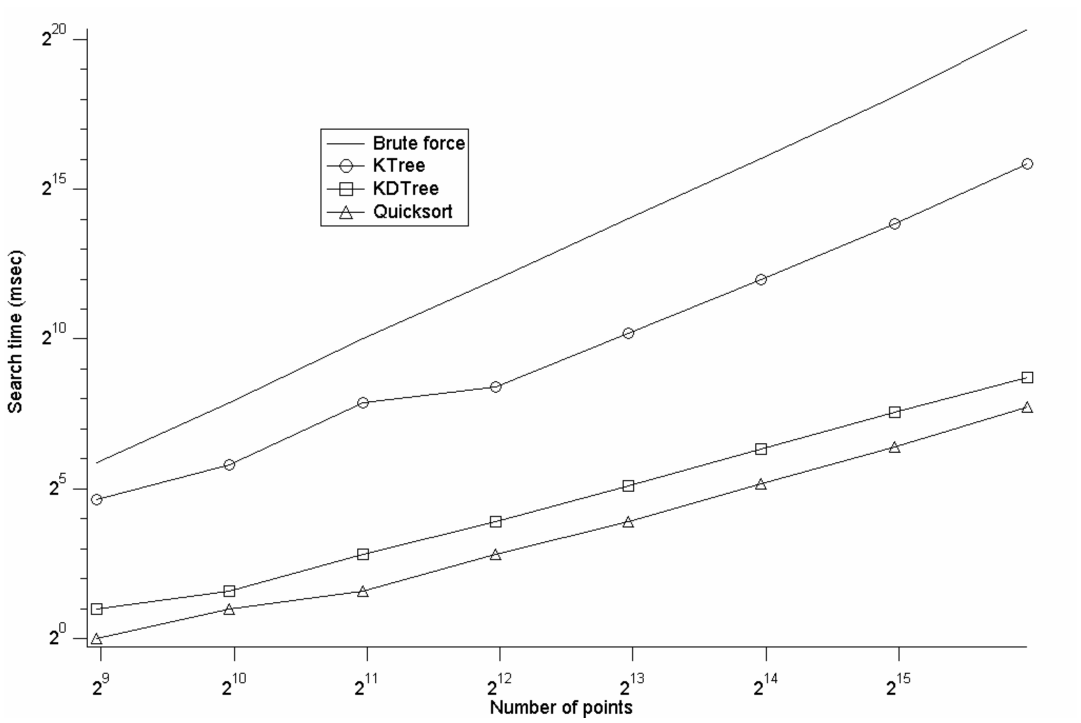


Figure 13. Comparison of search times for different methods using 1 dimensional random data.

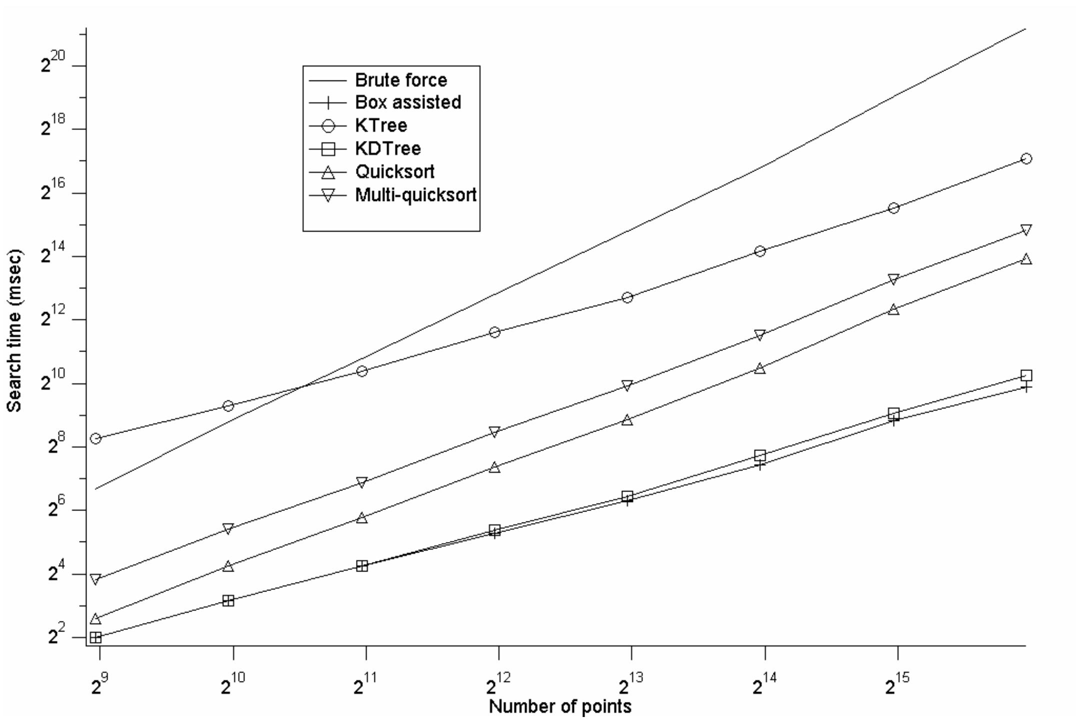


Figure 14. Comparison of search times for different methods using 2 dimensional random data.

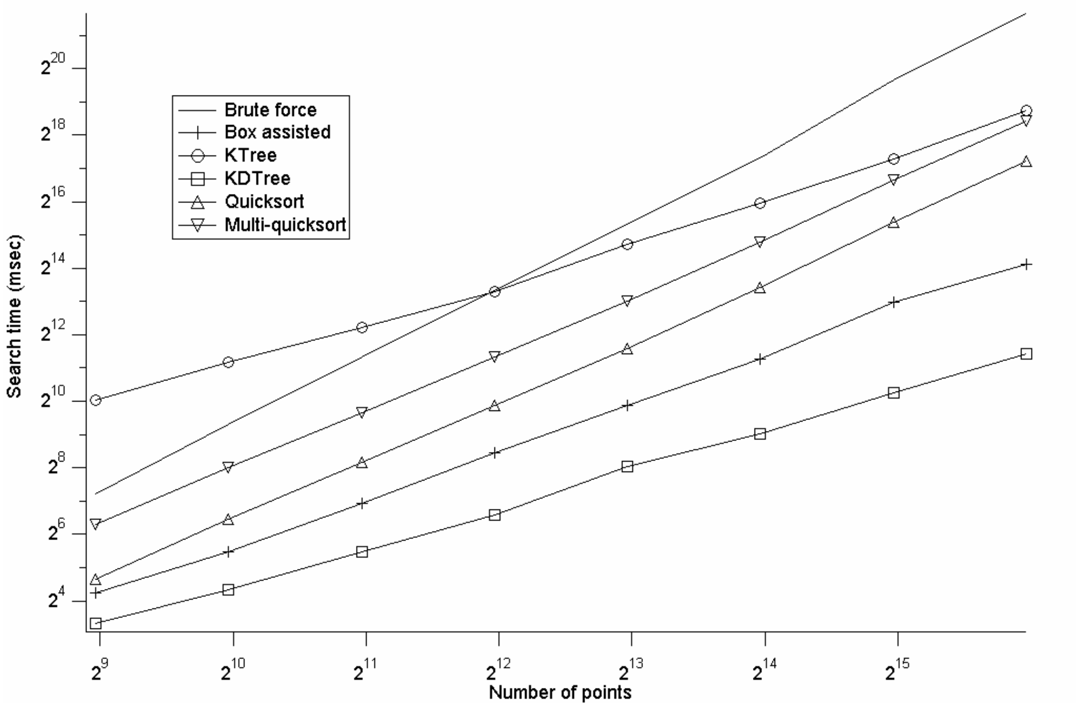


Figure 15. Comparison of search times for different methods using 3 dimensional random data.

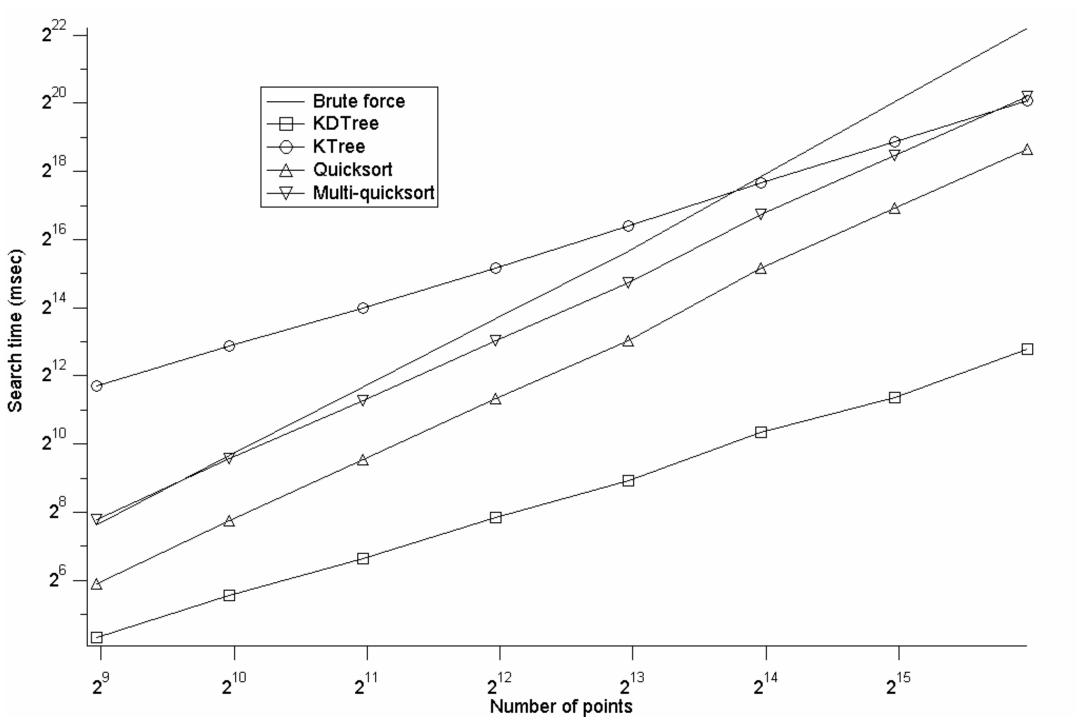


Figure 16. Comparison of search times for different methods using 4 dimensional random data.

However, all of the above trials were performed using uniform random noise. They say nothing of how these methods perform with other types of data. In order to compare the sorting and searching methods performance on other types of data, we compared their times for nearest neighbor searches on a variety of data sets. Table 2 depicts the estimated time in milliseconds to find all nearest neighbors in different 10,000 point data sets for each of the benchmarked search methods. The uniform noise data was similar to that discussed in the previous section. Each gaussian noise data set had a mean of 0 and standard deviation of 1 in each dimension. The Henon,[64] Lorenz,[25] and electric step motor[65] data sets are each from chaotic simulations. The identical dimensions and one valid dimension data sets were designed to test performance under unusual circumstances. For the identical dimensions data, uniform random data was used and each coordinate of a vector was set equal, e. g., $\overline{X_i} = (X_i^1, X_i^2, X_i^3) = (X_i^1, X_i^1, X_i^1)$. For the data with only one valid dimension, uniform random data was used in only the first dimension, e. g., $\overline{X_i} = (X_i^1, X_i^2, X_i^3) = (X_i^1, 0, 0)$.

Data set	**Dimension**	**Brute**	**Quicksort**	**Box (2) method**	**Box (3) method**	**KDTree**	**KTree**
Uniform noise	3	32567	2128	344	*210*	**129**	845
Gaussian	2	16795	*280*	623	X	**56**	581
Gaussian	4	44388	8114	54626	195401	**408**	*3047*
Henon	2	36707	*44*	227	X	**35**	345
Lorenz	3	59073	*373*	19587	26578	**49**	2246
Step motor	5	102300	*2019*	4870	5015	**174**	13125
Identical dimensions	3	33010	**19**	1080	5405	*42*	405
One valid dimension	3	30261	**31**	1201	7033	*37*	453

Table 2. Nearest neighbor search times for data sets consisting of 10000 points. The brute force method, multidimensional quicksort, the box assisted method in 2 and 3 dimensions, the KDTree and the KTree were compared. An X indicates that it wasn't possible to use this search method with this data. The fastest method is in bold and the second fastest method is in italics.

In all cases the Kd-tree proved an effective method of sorting and searching the data. Only for the last two data sets did the multidimensional quicksort method prove faster, and these data sets were constructed so that they were, in effect, one dimensional. In addition, the box method proved particularly ineffective for high dimensional gaussian data where the dimensionality guaranteed that an excessive number of boxes needed to be searched, and for the Lorenz data, where the highly nonuniform distribution ensured that many boxes went unfilled. The K-tree also performed poorly for high dimensional data (four and five dimensional), due to the exponential increase in the number of searched boxes with respect to dimension.

9. Conclusion

The Kd-search tree is a highly adaptable, well researched method for searching multidimensional data. As such it is very fast, but can be memory intensive, and requires care in building the binary search tree. The K-tree is a similar method, less versatile, more memory intensive, but easier to implement. The box-assisted method on the other hand, is used in a form designed for nonlinear time series analysis. It falls into many of the same traps that the other methods do. Finally the multidimensional quicksort is an original method designed so that only one search tree is used regardless of how many dimensions are used.

Tests indicated that, in general, the Kd-tree was superior to the other methods. Unlike, the box-assisted methods, its adaptability ensures that it is optimized for most unusual data sets. Its construction as a binary tree offers a far quicker search mechanism than the 2^k-ary tree approach of the K-tree. Although there are situations where other search methods may be preferred (such as when the time to build the tree is important), it is clear from the above results and analysis that the Kd-tree is the superior method for most searches on multidimensional data.

CHAPTER FOUR

ORIGINAL ANALYSIS METHODS

1. Introduction

In the process of implementing existing analysis methods, many holes in the theory of nonlinear data analysis were discovered. Existing methods were discovered to have flaws, or found that simple corrections could improve their power or efficiency. Likewise it was found that they could be extended to allow the determination of more empirical quantities. In this section, we discuss some of the original improvements on existing analysis methods, as well as some original analysis methods that were implemented.

2. Improvement to the false nearest neighbors method of determining embedding dimension.

2.1. Introduction

The identification of False Nearest Neighbors is a popular method of choosing the minimum embedding dimension of a time series for the purpose of phase space reconstruction. For a given embedding dimension, this method finds the nearest neighbor of every point in a given dimension, then checks to see if these are still close neighbors in one higher dimension. The percentage of False Nearest Neighbors should drop to zero when the appropriate embedding dimension has been reached.

We argue that the criteria for choosing an embedding dimension, as proposed in previous work, is flawed. We suggest an improved criterion that is still simple, yet properly identifies random data and high-dimensional data. We also show the limits of the method and of criteria that are used, and make suggestions as to how further improvements can be made.

One of the goals of time series analysis is to gain an understanding of the underlying dynamical system. However, this is a daunting task, since the time series gives incomplete information about the dynamics. Only one variable is being measured in a time series, yet a dynamical system is often represented mathematically by several differential or difference equations. The task then becomes how to visualize the time series in such a way as to reveal the dynamics in the full phase space of the system.

A phase space reconstruction is devised using a delay coordinate embedding.[66] From manipulations of the time series, delay coordinate embeddings allow one to reconstruct vectors representing the dynamics in the phase space. Rather than simply stating that "the time series evolves from X_1 to X_2" the researcher can now state that "the system evolves from state $\overline{Y_1}$ to $\overline{Y_2}$." Two parameters are used for the reconstruction, a delay τ and an embedding dimension *D*. From the time series, the vectors $\overline{Y_0} = (X_0, X_\tau, \ldots X_{(D-1)\tau})$, $\overline{Y_1} = (X_1, X_{1+\tau}, \ldots X_{1+(D-1)\tau})$,... are created. τ describes the separation between data to be used as successive points in a vector and *D* describes the dimensionality of the reconstructed vectors.

The two parameters of importance here are the choice of τ and the choice of D. Various methods are suggested for the determination of an appropriate delay, all with some measure of success.[42, 67, 68] However, even after a suitable delay time has been found, an appropriate embedding dimension is a subjective decision. One is generally guaranteed that a very large embedding dimension will completely unfold the attractor. The Takens embedding rule[36] states that, for an attractor of dimension d_A, any embedding greater than $2d_A$ will be sufficient. With experimental data of limited size, this rule may be used as a guideline for choosing an embedding dimension. Yet one may often use a much smaller embedding dimension. For instance the Henon map,[64] ($d_A \sim 1.8$)

$$X_{n+1} = 1 - 1.4X_n^{\ 2} + 0.3X_{n-1} \qquad (1)$$

may be completely unfolded in a two dimensional embedding and the Lorenz attractor[25] ($d_A \sim 2.1$)

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= -xz + rx - y \\ \dot{z} &= xy - bz \end{aligned} \qquad (2)$$

requires only a three dimensional embedding to unfold its attractor. Using a small embedding dimension may be preferable since it simplifies analysis and large embeddings compound the effects of noise and often require many more points for analysis. The question then becomes, what is the minimal embedding that is sufficient?

To this end, Kennel, et. al.,[46, 69] propose the method of false nearest neighbors. This is a popular analysis tool- it has been applied in a wide variety of fields,[70-74] and its meaning and implementation is an ongoing area of research.[61, 75, 76] The main objective in this section is to correct an important problem associated with current implementations of the method of false nearest neighbors. We will demonstrate that an alternative definition of false nearest neighbors and their interpretation will more accurately yield a minimal embedding dimension. This new definition is more robust in the presence of noise or high dimensionality, provides a statistical measure of confidence, and is easier to interpret. An added feature lies in its adherence to the spirit and intent of the original method, without losing any of the simplicity that has made identification of false nearest neighbors such a popular tool. The authors strongly recommend that future data analysis use this alternative definition or a similar implementation that corrects some of the problems associated with the original method.

2.2. Method

Consider a phase space reconstruction that was created using a delay coordinate embedding of a one-dimensional time series. If the embedding dimension is too small then two points that appear to be close in the reconstructed phase space may actually be quite far apart in the true phase space. These points are described as false nearest neighbors. As an example, consider Figure 17. Each of the points *A*, *B*, *C* and *D* in Figure 17 resides on a one-dimensional curve. Embedded in 2 dimensions, point *C* is the closest neighbor to point *A*. In a one-dimensional embedding, point *D* is incorrectly identified as the nearest neighbor. Yet neither *C* nor *D* is the closest neighbor to *A* on this curve. Only in three dimensions are there no self-crossings of this curve, and point *B* is shown to be the nearest neighbor of *A*.

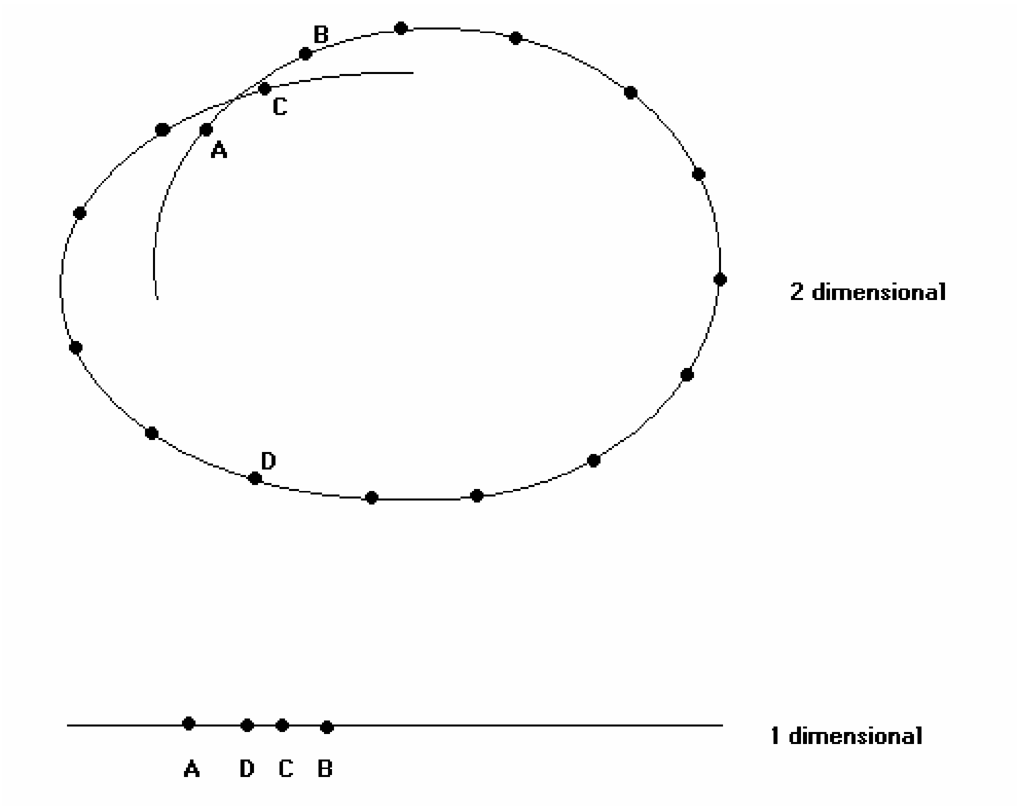


Figure 17 An example of false nearest neighbors. Points *A*, *B*, *C*, and *D* lie on a one-dimensional curve. In two dimensions, points *A* and *C* appear to be nearest neighbors. In one dimension, point *D* appears to be the nearest neighbor of *A*. In three dimensions the curve would not intersect itself, and it could be seen that *B* is the nearest neighbor to *A*.

Explicitly, the original criteria for identification of false nearest neighbors are as follows: consider each vector $\overline{Y_n} = (X_n, X_{n+\tau}, X_{n+2\,\tau}, ...X_{n+(D-1)\,\tau})$ in a delay coordinate embedding of the time series with delay τ and embedding dimension *D*. Look for its nearest neighbor $\overline{Y_m} = (X_m, X_{m+\tau}, X_{m+2\,\tau}, ...X_{m+(D-1)\,\tau})$. The nearest neighbor is determined by finding the vector $\overline{Y_m}$ in the embedding which minimizes the Euclidean distance $R_n = \| \overline{Y_n} - \overline{Y_m} \|$. Now consider each of these vectors under a D+1 dimensional embedding,

$$\overline{Y_n'} = (X_n, X_{n+\tau}, X_{n+2\ \tau}, ...X_{n+(D-1)\ \tau}, X_{n+D\ \tau})$$

$$\overline{Y_m'} = (X_m, X_{m+\tau}, X_{m+2\ \tau}, ...X_{m+(D-1)\ \tau}, X_{m+D\ \tau}) .$$

In a D+1 dimensional embedding, these vectors are separated by the Euclidean distance $R_n' = \| \overline{Y_n'} - \overline{Y_m'} \|$. The first criterion by which Kennel, et. al., identify a false nearest neighbor is if

$$\text{Criterion 1:} \sqrt{(R_n'^2 - R_n^{\ 2}) / R_n^{\ 2}} = \frac{| X_{n+D\tau} - X_{m+D\tau} |}{R_n} > R_{TOL} \qquad (3)$$

R_{TOL} is a unitless tolerance level for which Kennel, et. al., suggest a value of approximately 15 . This criterion is meant to measure if the relative increase in the distance between two points when going from *n* to *n+1* dimensions is large. 15 was suggested based upon empirical studies of several systems, although values between 10 and 40 were consistently acceptable. The other criterion Kennel, et. al., suggest is

$$\text{Criterion 2:} {R_n'}\big/{R_A} > A_{TOL} . \qquad (4)$$

This was introduced to compensate for the fact that portions of the attractor may be quite sparse. In those regions, near neighbors are not actually close to each other. Here, R_A is a measure of the size of the attractor, for which Kennel, et. al., use the standard deviation of the data, $R_A = \sqrt{<(x - <x>)^2>}$. If either (3) or (4) hold, then $\overline{Y_m}$ is considered a false nearest neighbor of $\overline{Y_n}$. The total number of false nearest neighbors is found, and the percentage of nearest neighbors, out of all nearest neighbors, is measured. An appropriate embedding dimension is one where the percentage of false nearest neighbors identified by either method falls to 0.

Although, these combined criteria correctly identify a suitable embedding dimension in many cases, they have several shortcomings. We will deal with each criterion in turn.

For criterion 1: Regardless of the data being considered, $|X_{n+D\tau} - X_{m+D\tau}| / R_n$ in equation (3) approaches zero as the embedding dimension increases. This is because, as one increases dimension, R_n increases although $|X_{n+D\tau} - X_{m+D\tau}|$, the increased distance due to adding a new dimension, may not. This may lead to falsely underestimating the correct number of embedding dimensions.

A neighbor may be considered false yet still be very close in the $(D+1)^{\text{th}}$ direction. This is because this criterion only measures relative increase. So in the case of two points are extremely close to each other in a D dimensional embedding, only a small separation in the $(D+1)^{\text{th}}$ dimension is required to identify a false nearest neighbor. A small amount of noise could easily generate such a situation. Whether or not these points are actually false neighbors is uncertain.

As pointed out in Kennel, et. al.,[46] and Brown,[69] a point and its nearest neighbor may not in fact be close to each other. Yet the original method still includes this as a point to be considered. This creates serious problems because it implies that small data sets (where near neighbors aren't close) R_n may be quite large. Thus the percentage of false nearest neighbors becomes dependent on data set size. For this reason, Kennel, et. al., suggested the second criterion. But as we shall see, that criterion does not properly address this issue.

For criterion 2: For the same reason that criterion 1 approaches zero as embedding dimension increases, this should gradually approach 100%. Criterion 2 may be written as $R'_n > R_A A_{TOL}$, where the right hand side is constant. Yet R'_n increases with each additional embedding dimension, since for each D, an extra $\left(X_{n+D\tau} - X_{m+D\tau}\right)^2$ is contributed to R'^2_n. So this criterion increases the percentage of false nearest neighbors for high embedding dimension. This holds even when the correct embedding dimension should be small.

This criterion labels nearest neighbors as false. However, it would be more correct to say that these neighbors were never identified, correctly or incorrectly, as close. In fact, if this criterion is ever nonzero, one would expect it to increase with embedding dimension, regardless of the correct embedding dimension or of the structure or quantity of data.

Standard deviation, which is Kennel, Brown, and Abarbanel's suggested measure of attractor size, may not be a good measure to use here. It measures the clustering about the mean. However, the standard deviation may be much larger than the distance between two neighbors, yet the neighbors may still not be close to each other. For instance, the Lorenz attractor has two lobes, so it should have a large standard deviation (points are clustered far from the mean). This implies that points on the same lobe may be far from each other yet still considerably less than the standard deviation. A more appropriate measure should be less dependent on the structure of the attractor.

For the above reasons, we choose a new method of identifying false nearest neighbors. For this, we return to the essential goals of the method:

(1) A false nearest neighbor is found if it may be correctly identified as a close neighbor in n dimensions, yet identified as far in the $(n+1)$th dimension.

(2) The percentage of false nearest neighbors is the percent of those neighbors that are identified as close which are then shown to be false nearest neighbors by (1).

The correct minimal embedding dimension is that for which the percentage of false nearest neighbors falls to its minimal level. It may not be possible to identify the correct embedding dimension if enough close nearest neighbors cannot be found. This is why it is important to properly define 'close.' Thus we provide a single explicit criterion definition as follows:

The average distance between points, δ, is estimated by taking a subsample of the original time series, and finding the average distance between any two points from this subsample. We note that this average distance is only used when multiplied by a threshold factor (ε_1 or ε_2). Hence other methods of estimating the average distance should be acceptable as long as the thresholds are scaled accordingly.

The distance between a point and its nearest neighbor is computed in each of the n dimensions. If, in each dimension, the distance is less than $\varepsilon_1 \delta$, then this neighbor is close. ε_1 is some small, user-defined threshold, large enough so that many close neighbors can be found, but small enough so that these two points are still close.

Trials indicated that ε_1=0.01 was a reasonable value for simulated data with no additive noise. A larger value is often necessary in the presence of white noise, since noise serves to increase the chances of points being far away. However, there are wide ranges of acceptable values for ε_1, since we are concerned with which embedding dimension gives a minimum in the percentage of false nearest neighbors, not the actual number for this percentage.

If the nearest neighbor is close to the original point, and the distance between the point and its nearest neighbor in the $(n+1)^{\text{th}}$ dimension is greater than $\varepsilon_2\delta$, then this is a false nearest neighbor. ε_2 is another user defined constant that adequately describes when two points are considered distant from each other.

2.3. Results

In Figure 18, we present the results of the use of this method with 1,000,000 points of uniform noise. Because the noise made it extremely difficult to find near neighbors, the thresholds for the new criterion were loosened. ε_1 was set to 0.1 (as opposed to 0.01) and ε_2 was set to $\sqrt{0.1}$. The value obtained from Criterion 1 dropped to 0, due to the fact that the average distance between nearest neighbors in n dimensions increases as a function of n, even though the average $(n+1)^{\text{th}}$ distance is constant. Criterion 2 increased with embedding dimension. This is because R_n' increased with embedding dimension D. But because R_n also increases with D, these nearest neighbors were never identified as close. However, pure noise represents an infinite dimensional system. So the percentage of false nearest neighbors should be independent of embedding dimension. The new criterion is correct in this regard since the percentage of false nearest neighbors it identifies remains constant with respect to changing embedding dimension.

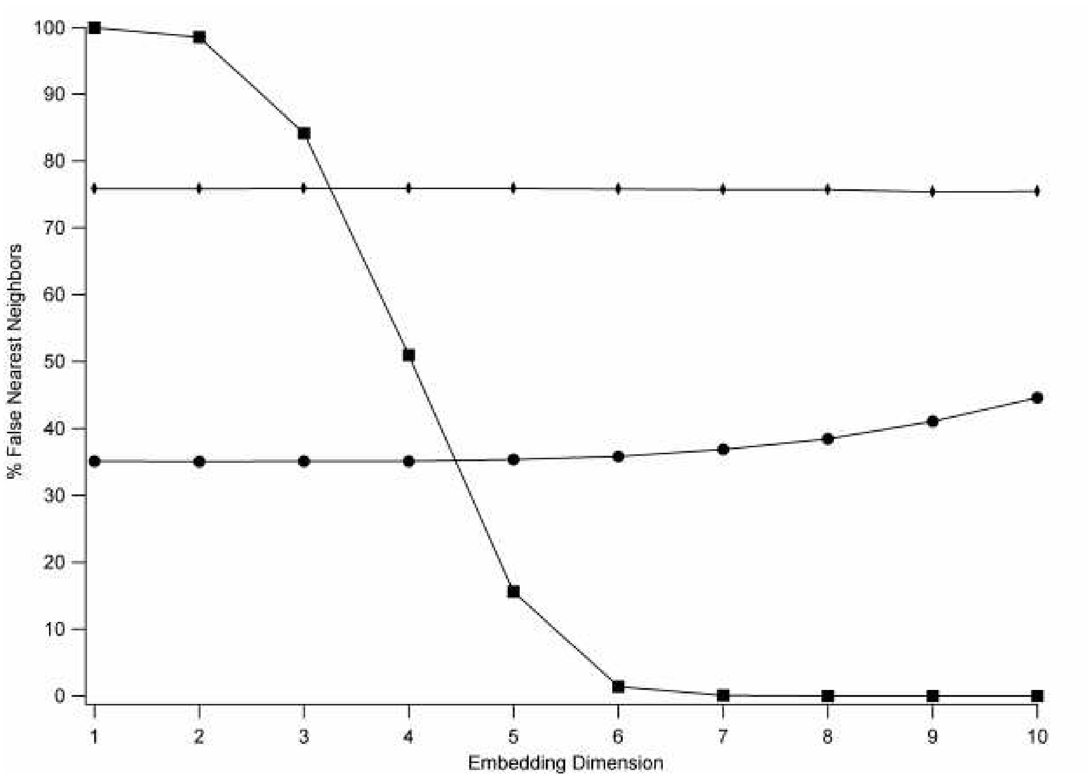


Figure 18. Data from 1,000,000 points of uniform white noise. The squares represent results from the first criterion, the circles from the second criterion and the diamonds from the new criterion. Close neighbors in n dimensions are defined as being separated by a distance no greater than 0.1δ in each of the n dimensions, where δ is the average distance between points. Far neighbors are defined as being separated by at least $\sqrt{0.1}$ δ in the *(n+1)*th dimension

In Figure 19, we show how the original criteria can overestimate the embedding dimension, and how the new criterion can be used both as a measure of noise and as a measure of whether enough data has been given. The data is from 10,000 points of the Henon map (equation 1) with the addition of Gaussian noise with 0 mean and standard deviation at 5% the standard deviation of the original data. Since the Henon map has a fractal dimension of 1.2, it is known that a 2 or 3 dimensional embedding is appropriate. In fact, without the presence of noise, a 2 dimensional embedding will suffice. The original criterion detects 5.7% false nearest neighbors in 3 dimensions. Thus the original method suggests an embedding dimension of at least four. This is due to the fact that, in a low dimensional embedding, this method incorrectly interprets the fluctuations in nearest neighbor distances due to noise as identification of false nearest neighbors.

In contrast, the new method reaches a constant value of 24% false nearest neighbors at an embedding dimension of 3. Because the Henon map has a fractal dimension of approximately 1.2, the Takens embedding theorem suggests an embedding dimension of 3 or less. Therefore this result is within reason for the Henon map. For a five dimensional embedding, only 26 close nearest neighbors were found from the original 10,000 points. Thus the inclusion of error bars was necessary. Error bars were computed by computing a 95% confidence interval on the proportion of false nearest neighbors. Higher dimensional calculations were not calculated because for the parameters chosen, it was not possible to find enough close neighbors to get an accurate percentage of the false ones.

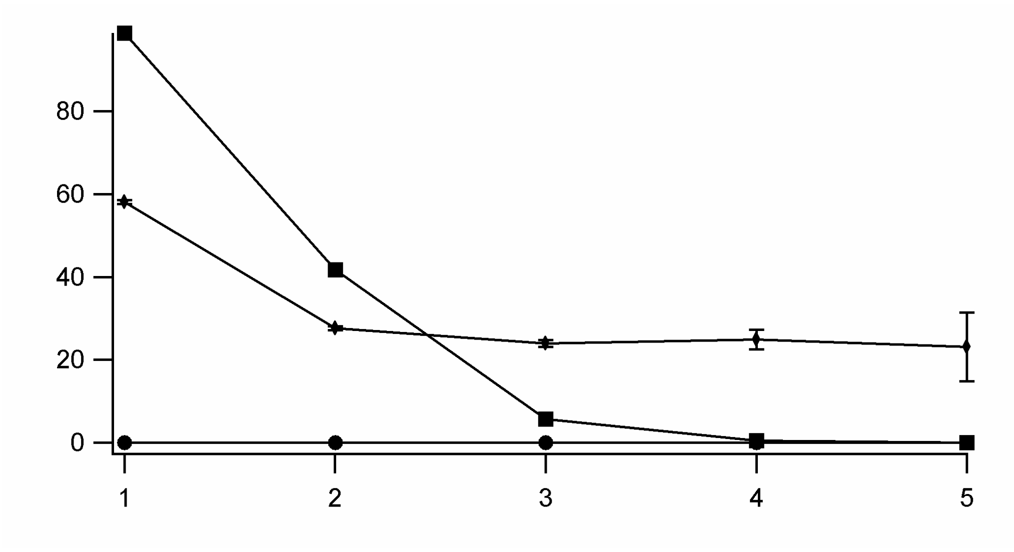


Figure 19. Data from 10000 points of the Henon map with Gaussian noise having a standard deviation equal to 5% the standard deviation of the original data. Squares represent results from the first criterion, circles from the second criterion and diamonds from the new criterion. Error bars correspond to a 95% confidence level. Close neighbors in *n* dimensions are defined as being separated by less than 0.01δin each of the *n* dimensions. Far neighbors are defined as being separated by at least 0.1δin the (*n*+1)th dimension.

2.4. Conclusion

One possible problem with this method is the choice of parameters. If the maximal distance for neighbors to be considered close is set too high, then complicated dynamics can occur within a region less than this distance. That is, even with a suitable embedding dimension, neighbors can appear false simply because they were incorrectly identified as close. A reasonable setting for ε_l is small enough that the dynamics on this length scale may be considered approximately linear, but large enough that a sizable number of nearest neighbors can be found. If an ε_l can not be found fitting these qualifications, then the size of the data set does not permit use of the false nearest neighbors method.

As we move to higher and higher embedding dimension, the distance between nearest neighbors in the embedding must get further apart. This is because each additional dimension contributes additional distance. If we set the condition for closeness too high, then these neighbors may not be identified as close. Thus we may incorrectly find more false nearest neighbors with increasing dimension. This effect is most noticeable in noise free data, such as the Lorenz system. Also regions of the attractor may have both variable density and variable local dimension. For data sets of small size, one may then be concerned with the question of when points are close, and how this definition should vary for sparse regions or regions with complicated dynamics. A very strict definition of closeness should, for a very large data set, be sufficient to deal with this problem. However, this is not always possible, so we strongly suggest the use of error bars and the use of as small of a closeness threshold as possible (see Figure 19).

There is a tradeoff here between simplicity and accuracy. A more accurate method would use all neighbors that can be considered close, rather than just the nearest neighbor. It would take into account the variable density on the attractor, and perhaps give a more quantitative interpretation of exactly what information is lost when using too small of an embedding dimension. However, the benefit of the method of false nearest neighbors is in its simplicity. It provides a simple method of determining embedding dimension that can be easily interpreted. Since it is designed for experimental systems, and it is designed for making reasonable estimates of a sufficient embedding dimension, rigor is not of high importance. Simplicity is favored, as long as the method does not underestimate or grossly overestimate embedding dimension.

Lastly we want to point out the relationship between dimensionality, data set size, and closeness threshold. For a high dimensional system, a very large number of points is needed to estimate fractal dimension. This is also true for estimating embedding dimension. Close points must be close in all dimensions, and there are few candidates for close neighbors until a large portion of the attractor is filled. Following the reasoning in Ding, et. al.,[31] we suggest a minimum of 10^D points to reliably determine if D is a reasonable embedding dimension. Noise only compounds the problem, since it causes nearby points to seem further away from each other in each dimension. Of course, more neighbors can be found by loosening the threshold, but this carries the risk of considering neighbors that are not close to each other. For this reason error bars are suggested, indicating whether enough neighbors are being considered. However, error bars only provide a measure of statistical confidence assuming that the underlying method of finding false nearest neighbors is correct. This is a problem that may be alleviated with a more rigorous definition of closeness.

3. Modification to the Wolf method of determining the dominant Lyapunov exponent.

3.1. Introduction

This was probably the first method implemented to estimate the dominant Lyapunov exponent of experimental data. Wolf[77] defined the exponents in terms of the rates of divergence of volumes as they evolve around the attractor. For an *n*-dimensional system, consider the infinitesimally small *n*-sphere centered around a point on the attractor. As this evolves, the sphere will become an *n*-ellipsoid due to the deforming nature of the attractor flow. Then the *i*th Lyapunov exponent is defined in terms of the exponential growth rate of the *i*th principal axis of the ellipsoid.

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln \frac{p_i(t)}{p_i(0)} \qquad (5)$$

where the exponents are ordered from largest to smallest. The largest exponent can be approximated from Equation (6)

$$\lambda_1 \approx \frac{1}{M\Delta t} \sum_{k=0}^{M-1} \ln \frac{L_{ks}(s)}{L_{ks}(0)} \qquad (6)$$

Here, Δt is the time (in seconds) of successive points in the time series. *M* is the number of steps of size *s* used to traverse the entire data set. That is, if there are *N* vectors, then *M* is the largest value such that $Ms \le N$. $L_{ks}(0)$ is the distance between the *ks*th point and an appropriately chosen near neighbor such that the vector between these points is aligned along the direction of the first principal axis of the ellipsoid. $L_{ks}(s)$ is the distance between these two points after they have been evolved forward *s* positions in the time series. Thus, in the limit of an infinitely long time series where $s \rightarrow \infty$, these two equations become equal..

One difficulty with this method is the presence of two opposing requirements. $L_{ks}(s)$ must be small enough such that the *n*-ellipsoid is reasonably small. This would be guaranteed in the theoretical limit of an infinitesimally small initial *n*-sphere. At the same time, $L_{ks}(s)$ must be aligned along the direction of the principal axis. Wolf suggests two parameter settings in the algorithm in order to meet both requirements. Choice of the nearest neighbor must result in a length $L_{ks}(0)$ less than a specified value. In addition, the angle between these two vectors must be less than another specified value.

In order to minimize parameter dependence, we used a variation on the Wolf method. In previous work,[77-79] both maximum allowable displacement and maximum angular displacement were left as free parameters. This makes it difficult to determine a reasonable parameter setting. We suggest using one parameter- the number of near neighbors to consider after a given number of iteration steps. Whether or not our new vector is replaced is determined by finding the vector among those from the near neighbors that has minimal angular displacement α, as defined in Equation (7). This may be the original vector used. This would imply that the vector has not grown enough to be out of the linear regime. If this point is not among the near neighbors, then the near neighbor giving the least angular displacement is used.

$$\alpha = \operatorname{acos}\left(\frac{\bar{X}\cdot\bar{Y}}{|X||Y|}\right) \qquad (7)$$

This alleviates one of the problems that Wolf, et al., mentioned. An appropriate maximum displacement is not the same over the entire attractor. In dense regions, a small separation may be considered large and nonlinear. In sparse regions of the attractor, a larger distance is acceptable. This is true of the Lorenz attractor. If we consider a fixed number of neighbors, then these points will be closer in dense regions of the attractor and further apart in sparse regions. This has the effect of varying the maximum displacement as appropriate. In its current implementation, the downside is that this increases the computation time significantly. However, computation was not so slow as to be infeasible.

4. An efficient method of computing an arbitrary number of generalized fractal dimensions.

4.1. Method

Consider a multidimensional time series, possibly generated from a delay coordinate embedding. Using the multidimensional binary representation sort described in Chapter 3, this data can be gridded into n-dimensional boxes of equal length ε, such that all vectors lie within these boxes. If a box is labeled *i*, then it has an associated probability, $P_i(\varepsilon)$, that a vector on the attractor will reside within this box. The generalized entropies, H_0, H_1, … are defined in terms of the probabilities of vectors occupying boxes. For q=0,1,2…,

$$H_q(\varepsilon) = \frac{1}{1-q} \ln \sum_{i=1}^{N(\varepsilon)} P_i^q(\varepsilon) \ldots q \neq 1$$
$$H_1(\varepsilon) = -\sum_{i=1}^{N(\varepsilon)} P_i(\varepsilon) \ln P_i(\varepsilon) \tag{8}$$

In the following we keep the conventions common in dimension definitions and use the natural logarithm as opposed to the log base 2. The generalized dimension of order q is then defined as

$$D(q) = -\lim_{\varepsilon \to \infty} \frac{H_q(\varepsilon)}{\ln \varepsilon} \tag{9}$$

Under this definition, $D(0)$, $D(1)$, and $D(2)$ are the box counting dimension, the information dimension and the correlation dimension, respectively. We also have the property that if $p>q$, then $D(p) <= D(q)$.

For the purposes of calculation we rewrite the generalized entropies as

$$H_q(\varepsilon) = \frac{1}{q-1}\left[\ln N^q - \ln \sum_{i=1}^{N(\varepsilon)} N_i^q(\varepsilon) \right], \ldots q \neq 0,1$$
$$H_1(\varepsilon) = \ln N - \frac{1}{N} \sum_{i=1}^{N(\varepsilon)} N_i(\varepsilon) \ln N_i(\varepsilon) \tag{10}$$
$$H_0(\varepsilon) = \ln N(\varepsilon)$$

where N is the total number of vectors, $N_i(\varepsilon)$ is the number of vectors in the ith box of length ε, and $N(\varepsilon)$ is the number of occupied boxes of length ε. In this form it is clear what is needed- an accurate method to keep track of how many vectors lie inside of each box. Preferably, so as to measure convergence, one would like to be able to compute entropies for many box sizes, and do so in reasonable time with limited memory resources.

ε in the generalized entropy represents which of the 32 levels of the binary representation we are considering. The data is now ordered so that all the vectors within a box of any size ε (1, ½, … 1/ (2^32 –1)) are ordered consecutively. The generalized entropies can be determined by scrolling through the data and counting how many vectors are in each box. That is, a vector is compared with the previous vector. Information is stored concerning each level of each generalized entropy to be determined. There are 32 levels. If on any of the dimensions of the two vectors, the binary representations of the two vectors on that level are not equal, then we know that these vectors reside in different boxes. That means we can process the box we were just looking at. Find where they disagree, and for each level below that add the number of vectors in the box. Each time we have determined how many vectors are in a box, we process that value and add the result to a running tally for the given entropy. When this is finished we have exactly what is needed for the generalized entropies.

Once the generalized entropies have been determined, there are two ways to approximate the generalized dimensions for time series data. The first is if ε is sufficiently small such that the limit is approximately correct, and we have enough data to get an accurate measurement for $H_q(\varepsilon)$. In which case we may use

$$D_q(\varepsilon) \approx -H_q(\varepsilon)/\ln\varepsilon \qquad (11)$$

However, the preferred method is to simply look at the slope of a plot of $H_q(\varepsilon)$ vs ln(ε), since this has a quicker rate of converge to the limit of ε–>0. We should mention that further information theoretic properties can be determined from the analysis of this sorting, such as the generalized mutual information of high dimensional data, $I_n(X_1,X_2,\ldots X_n)$, or estimation of the metric entropy.[80]

4.2. Accuracy

Unfortunately, although this is an efficient way of determining generalized dimensions and entropies, it is not necessarily accurate. Figure 20 and Figure 21 show estimates of the generalized entropies and generalized dimensions for a data set consisting of 5 million data points from the Ikeda map. Although convergence can be seen, it is a poor estimate because such a large number of points is required for convergence.

It has been shown that similar "box-assisted" methods often require huge amounts of data, due to the very slow convergence to the limit of infinite data, arbitrarily small ε.[33] This means that an adaptation of this method that allows for insertion into the sorted data could be highly effective for the case of when an arbitrary number of data points can be generated. However, this is typically not possible for the case of experimental data. Thus spatial correlation algorithms, although generally more computationally intensive, are often preferred for calculations of information dimension or correlation dimension. It is hoped that a spatial correlation algorithm could be adapted to the determination of generalized dimensions and generalized entropies.

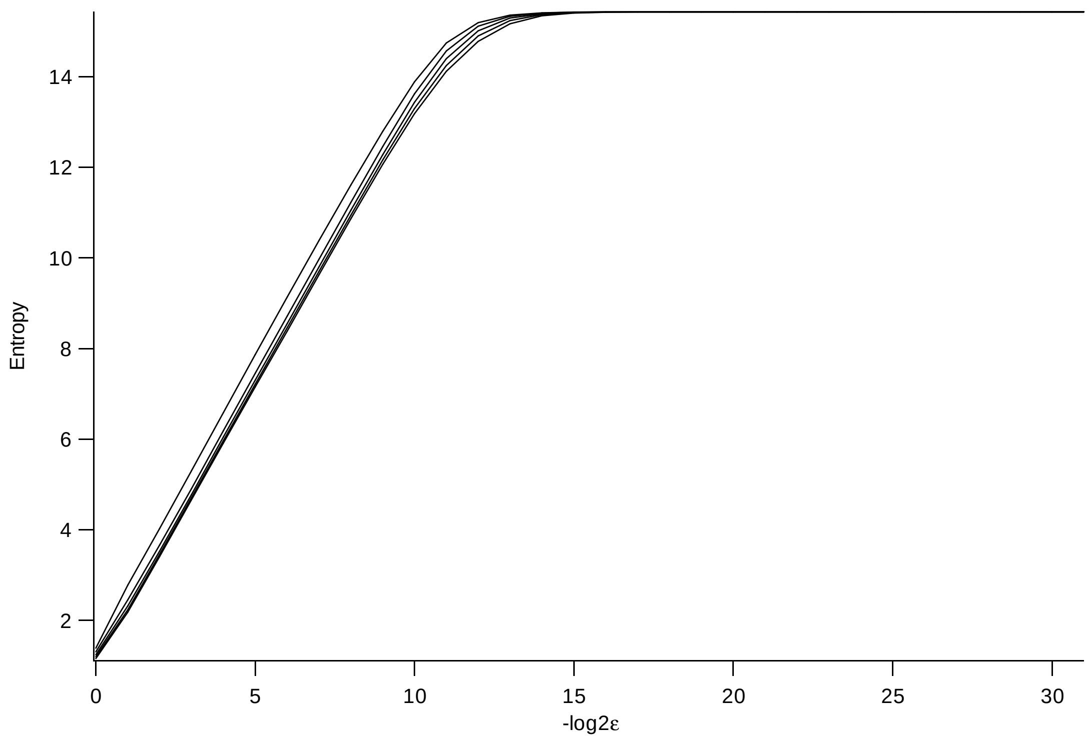


Figure 20 The first five generalized entropies of the Ikeda map. 5 million data points were used in order to get strong convergence. It can be seen that there is a large range of ε over which the slopes remain fairly constant.

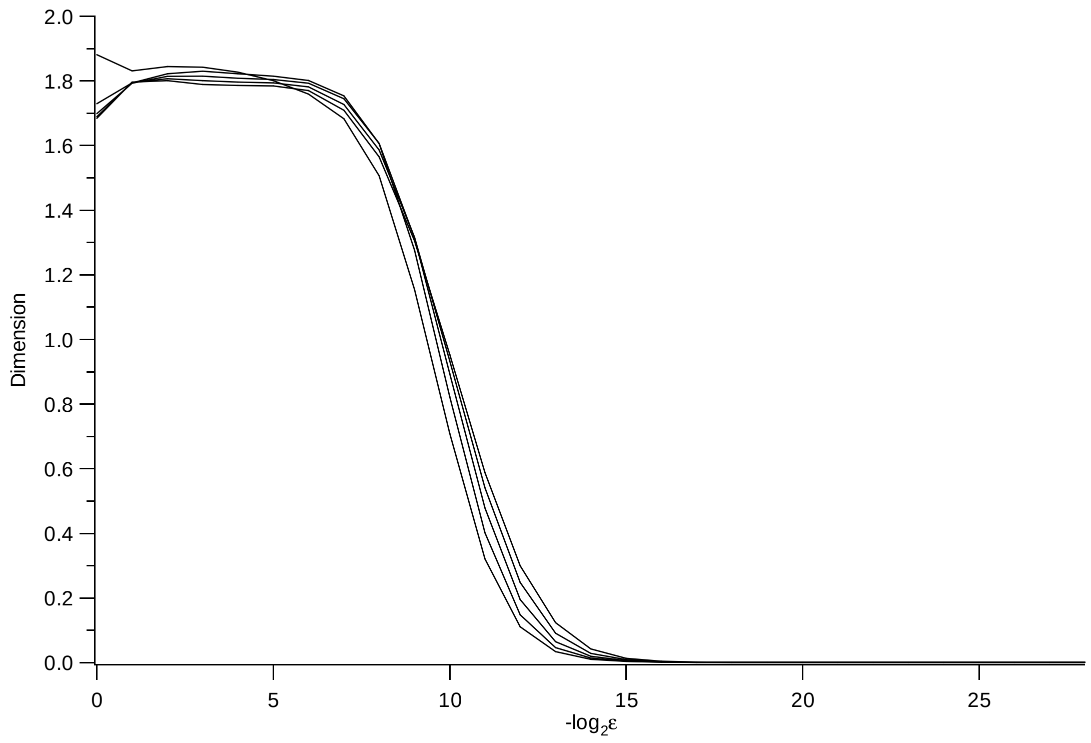


Figure 21 The first five generalized dimensions of the Ikeda map. 5 million data points were used in order to get strong convergence. Still there is large fluctuations in the results, particularly in D_0, the box-counting dimension.

5. Computation of symbol dynamics and information theoretic quantities

5.1. Transition matrix

No use is made in the above method of the sequencing of the original data. Consider a vector X_n residing in the box *i*. If in addition to the quicksort, a reverse sort is maintained that maps a vector in the sorted data to the corresponding vector in the time series, then one can determine the vector X_{n+1}, and hence find the box *j* that contains this vector. Thus allowable sequences of symbol dynamics can be quickly determined. Benefits of this can be seen in the case of noisy data. Here, one sets an ε based on the estimated noise level of the data. The presence of noise implies that the estimate $f\left(X_n\right) = X_{n+1}$, may be incorrect. However, one can state that X_n might be mapped into any box for which any point in box *i* are mapped. Thus we have the information necessary to estimate a probability transition matrix as well as entropy rates.

The probability transition matrix is [P_{ij}], where $P_{ij} = \Pr\{X_{n+1} = j \mid X_n = i\}$. To calculate any P_{ij}, we find the number of vectors in box *i* that get mapped to a vector in box *j*, and then divide by the total number of vectors in box *j*. As shown in Ref. 81, a similar method can be used to identify periodic orbits from symbolic dynamics.

5.2. Entropy rate

Identification of the transition matrix allows one to measure the entropy rate of the system, per symbol entropy of the *n* random variables.

$$\mathrm{H}(\chi) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, ..X_n) \qquad (12)$$

This is equivalent to the conditional entropy of the last random variable given the past, where $\{X_1\}$ is a stochastic process.

$$\mathrm{H}(\chi) = \lim_{n\to\infty} \frac{1}{n} H(X_n | X_{n-1}, X_{n-2}, ..X_1) \qquad (13)$$

The entropy rate describes how the entropy of the sequence grows with *n*, and thus quantifies the complexity of the system. If a suitable embedding dimension and delay have been found, or if we have multidimensional data from a simulation, then the time series may be described as a Markov chain. Furthermore, if the process is stationary, then the entropy rate becomes

$$\mathrm{H}(\chi) = -\sum_{ij} \mu_i P_{ij} \ln P_{ij} \qquad (14)$$

where μ_i represents the i^{th} eigenvalue of the transition matrix, that is, the probability of a vector being in the i^{th} box. Unfortunately, this is a fairly difficult quantity to compute, because it requires on the order of the square of the number of boxes to calculate. This is a more accurate measure of information loss than the traditional entropy since it takes into account the ordering of the data. However the entropy rate is a fairly stable quantity when measured over a great range of data set size and of gridding accuracy.

5.3. Topological entropy and periodic orbits

Suppose the time series data has been gridded into boxes of size ε. According to Baker and Gollub,[82] the metric entropy is found from

$$h_M = \lim_{\varepsilon \to 0} \lim_{T \to \infty} \left[-\frac{1}{T} \sum_{i=1}^{N(\varepsilon)} P_i(\varepsilon,T) \ln P_i(\varepsilon,T) \right] \qquad (15)$$

where $P_i(\varepsilon,T)$ represents the probability of a point being found in the i^{th} box of size ε after a time T. Computation of this quantity is quite difficult because of its dependence on limits, but it can be estimated from the transition matrix, which gives

$P_i(\varepsilon,T)$ for a given ε and T. The topological entropy can also be directly estimated from the transition matrix **M**,[83] using .

$$h_t = \lim_{p\to\infty} \frac{1}{p} \ln tr\mathbf{M}^p \qquad (16)$$

This gives an upper bound on the metric entropy, Equation (15), and also provides an estimate of the number of periodic orbits of period p, N_p, since

$$h_t = \lim_{p\to\infty} \frac{1}{p} \ln N_p \qquad (17)$$

Thus, with an appropriate embedding and the use of the multidimensional binary representation mentioned in Chapter 3, topological entropy, metric entropy and the entropy rate may all be estimated on the symbolic dynamics of a time series. We should note here that there is still some disagreement over the definitions and meanings of these quantities for multidimensional embedded time series data. Nevertheless, they do provide a measure of information loss in the system.

5.4. Symbolic dynamics and Conley index theory

5.4.1. Introduction

Symbolic dynamics play a central role in the description of the evolution of nonlinear systems. Yet there are few methods for determining symbolic dynamics of chaotic data. One difficulty is that the data contains random fluctuations associated with the experimental process. Using data obtained from a magnetoelastic ribbon experiment we show how a topological approach can be used to obtain a description of the dynamics in terms of subshift dynamics on a finite set of symbols. These topological methods have been developed in a way that allows for experimental error and bounded noise.

There has been considerable effort within the scientific community to develop methods to determine whether a given dynamical system is chaotic (see Ref. 30 and references therein). On the mathematical side three issues make rigorous analysis of chaotic dynamics difficult. The first is that one is dealing with global nonlinearities making it difficult to obtain the necessary analytic estimates. The second is that the individual solutions are inherently sensitive to the initial conditions. Finally, the objects of interest, namely invariant sets, can undergo dramatic changes in their structure due to local and global bifurcations. The problem is dramatically more complicated in the setting of experimental systems because of the introduction of noise, parameter drift, and experimental error.

Topological techniques for the analysis of time series[84, 85] have been introduced which, at least on the theoretical level, can be used to overcome these difficulties. As was shown in Ref. 81, these techniques can be successfully applied in the context of an actual physical system, a magnetoelastic ribbon subject to a periodically oscillating magnetic field. The actual implementation is described in more detail below. For the moment we remark that our ideas are extensions of the numerical methods developed by Mischaikow, et. al.,[84, 85] along with a reinterpretation of what is an appropriate embedding theorem[36, 37, 66] to justify the use of time delay reconstructions.

At the basis of our approach are three assumptions. The first is that the dynamics of the physical system can be represented by a continuous map $f : X \times \Lambda \to X$ where *X* represents the phase space and Λ the experimentally relevant range of parameter space. The second is that we are trying to describe the dynamics of an attractor for *f* in *X*. The third involves the relationship between the physical phase space and the values observed by the experimentalist. In particular, we allow the experimental observation to be represented as a multivalued map $\theta : X \to [\alpha_x, \beta_x] \subset \mathbf{R}$ where $|\alpha_x - \beta_x|$ may be thought of as an upper bound for the experimental error. However, we also assume that there is a continuous map $\gamma : X \to \mathbf{R}$ such that $\gamma(x) \in \theta(x)$ for all $x \in X$. One may view γ as representing the ``true'' measurement of the system. We hasten to point out that we are only assuming the existence of such a γ, but never assume that it is known. We should also point out that if it is impossible to choose θ and γ as above, then for some points in the physical phase space arbitrarily small changes in the physical system must lead to arbitrarily large changes in measurements.

To describe the output of our method it is necessary to recall that a transition matrix *A* on *k* symbols is a $k \times k$ matrix whose entries a_{ij} take values of 0 or 1. Given *A* one can define a subset of the set of bi-infinite sequences on *k* symbols,

$$\Sigma_A = \{ s_n \mid s_n \in \{1, 2, ..., k\} \text{ and } a_{s_n, s_n+1} = 1 \} \qquad (18)$$

Let $\sigma : \Sigma_A \to \Sigma_A$ denote the shift dynamics $\sigma(s)_n = s_{n+1}$. If we choose the dimension of the reconstruction space to be *d*, then applying our method results in a finite collection of regions N_i, $i = 1,2,...,k$ in $\mathbf{R}^d$ and a $k \times k$ transition matrix A for which the following conclusion can be justified. Given any sequence $\{\lambda_n\} \subset \Lambda$ and any element $s \in \Sigma_A$ there exists an initial condition $x \in X$ such that

$$\theta\left(f^n\left(x,\lambda_n\right)\right)\times\theta\left(f^{n+1}\left(x,\lambda_{n+1}\right)\right)\times...\times\theta\left(f^{n+d}\left(x,\lambda_{n+d}\right)\right)\cap N_{s_n} \neq \varnothing \quad (19)$$

In other words, given any sequence of perturbations in the parameter settings, there exists an initial condition such that, up to experimental error, the reconstructed dynamics describes the observed physical dynamics.

5.4.2. Experimental setup

Data was gathered from the magnetoelastic ribbon experiment for the specific purpose of providing data which could prove the validity of this technique. A full description of this experiment is provided in Chapter Five, Section Three. The ribbon was operated in a regime that appeared to give fairly low-dimensional, possibly chaotic dynamics. Although environmental effects were minimized, the system was extremely sensitive, and thus a significant amount of error needed to be taken into account when doing the calculations. The data set consisted of 100,000 consecutive data points $\{v_n \mid n = 1,...100,000\}$ taken from voltage readings on a photonic sensor, sampled at the drive frequency of 1.2 Hz. The voltage readings were measured up to 10^{-3} volts, and corresponded to the position of the ribbon once per drive period.

5.4.3. Implementation of our method

In an attempt to provide a concise and coherent description of our method we will present it in the context of our analysis of the above mentioned ribbon data. For an abstract mathematical description of the method the reader is referred to [81]. We selected 30,000 data $\{v_n \mid n = 30,001,...60,000\}$ from our data set. We chose a reconstruction dimension of 2 producing the reconstruction plot of $U := \{u_n = (v_n, v_{n+1})\} \subset \mathbf{R}^2$, as indicated in Figure 22.

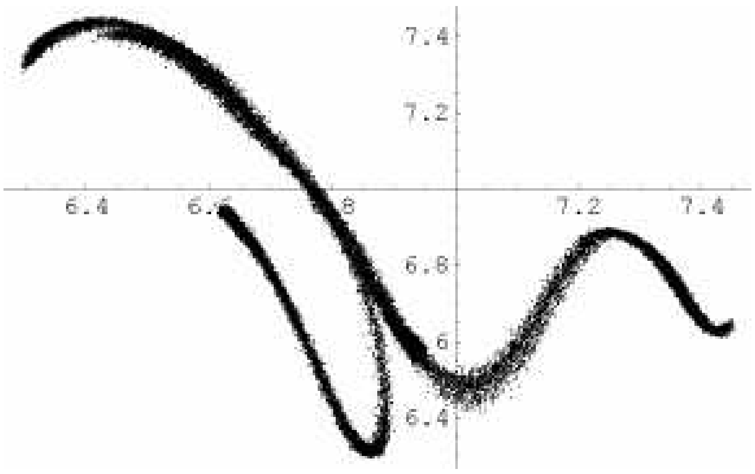


Figure 22 Time delay plot of u_n=(v_n, v_{n+1}), n=30,001,..., 60,000, where v_n is the nth voltage.

In order to produce nontrivial topology for our reconstructed system we divided $\mathbf{R}^2$ into a grid of squares with each side of length 0.0106 volts. Let **G** denote the set of squares which contain a point $u_n \in U$. Let $Y \subset \mathbf{R}^2$ denote the region determined by this collection of squares Figure 23.

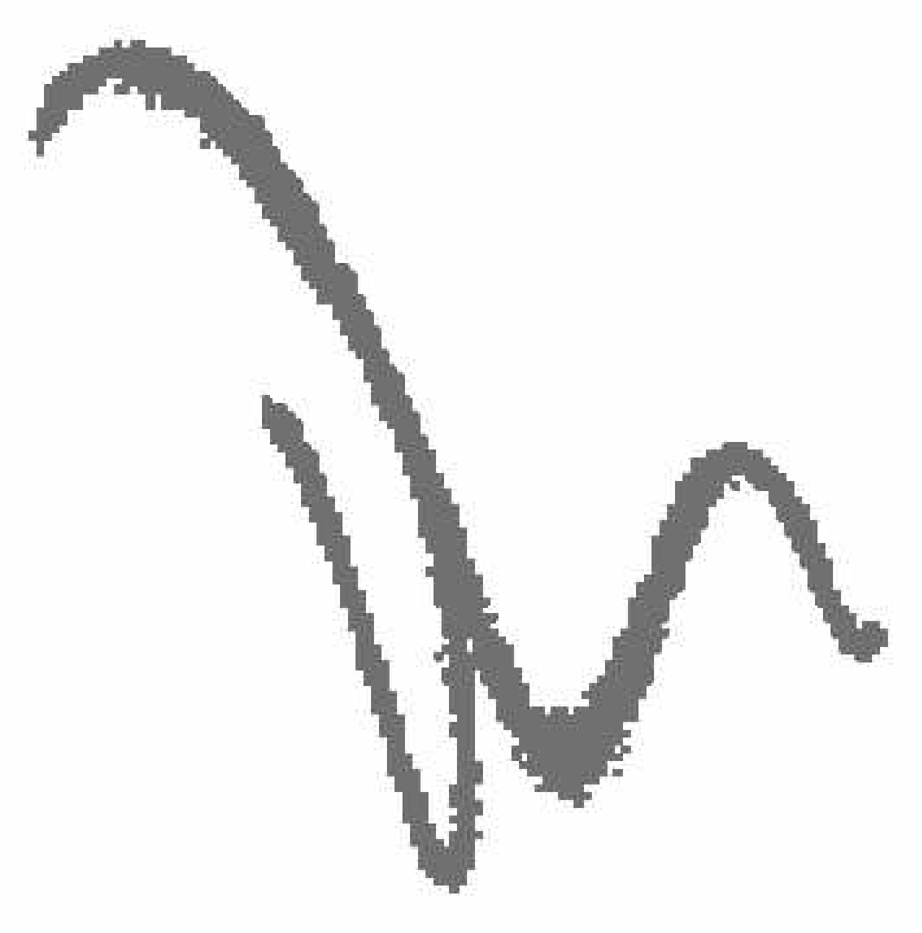


Figure 23 Squares with edges of length 0.0106 volts that contain points $u_n \in U$ of the time delay reconstruction.

The next step is to define a dynamical system on *Y*. This dynamical system is supposed to capture the observable dynamics of the experimental system. Since we know that the physical system is subject to noise and experimental error the derived dynamical system must be very coarse. With this in mind we shall not try to describe the dynamics on any scale smaller than that of the squares in **G** and our dynamical system will take the form of a multivalued map **F** which takes squares to sets of squares.

To be more precise, let $G \in \mathbf{G}$ and let $\left\{ u_{a_i} \middle| i = 1, 2, ... I \right\} \subset U$ be the set of points which lie in *G*. From the time series we know that $u_{a_i} = \left(v_{a_i}, v_{a_i+1} \right) \mapsto \left(v_{a_i+1}, v_{a_i+2} \right) = u_{a_i+1}$. Let $G_i^{'} \in \mathbf{G}$ such that $u_{a_i+1} \in G_i^{'}$. Then up to first approximation we will require that $G_i^{'} \in \mathbf{F}(G)$. Unfortunately, this definition is not sufficient. One reason is that there may be very few samples in the grid square *G*, so that $G_1^{'}, G_2^{'}, ..., G_I^{'}$, are isolated squares far apart from each other. We would like $\mathbf{F}(G)$ to be a connected set and to capture all possible images of points of *G*, not just ones for which we have samples from the time series. Therefore, we include all grid squares contained in the smallest rectangle enclosing $G_1^{'}, G_2^{'}, ..., G_I^{'}$ in $\mathbf{F}(G)$. However, sometimes this is still not enough. This is because the images of four squares meeting at one point may not intersect which prevents the map **F** from having a continuous selector. We deal with this problem in the following way. For each grid point we look at images of four grid squares which meet at that point. If they do not intersect, we increase each image by the set of all squares which intersect the set representing it. We iterate this process until there are no empty intersections. Finally, since we want **F** act on **G**, we intersect each image of a grid square with **G**.

By means of this procedure we have constructed a multivalued map **F** which we believe provides outer limits on the observable dynamics of the experimental system. Trajectories in this dynamical system consist of sequences of squares of the form $\left\{G_i \in \mathbf{G} \middle| G_{i+1} \in \mathbf{F}(G_i)\right\}$.

We must now face the issue of what dynamics is represented by **F**. To do this we must make several slight theoretical digressions. Entirely ignoring the question of the magnetoelastic ribbon for the moment, let $g : \mathbf{R}^n \to \mathbf{R}^n$ be an arbitrary continuous map. Recall that $S \subset \mathbf{R}^n$ is an invariant set of g if for every $x \in S$ there exists a bi-infinite sequence $\left\{x_i\right\}_{n=-\infty}^{\infty} \subset S$ such that x=x_0 and x_{i+1}=$g(x_i)$. While invariant sets are the object of interest in dynamical systems, they can be extremely difficult to study directly since they may change their properties dramatically through various local or global bifurcations. For this reason we shall make use of the following notion. A closed bounded set $N \subset \mathbf{R}^n$ is an isolating neighborhood if the maximal invariant set in N does not intersect the boundary of N. It is easily shown that if N is an isolating neighborhood for the dynamical system generated by g, then it is an isolating neighborhood for sufficiently small perturbations of g.

Returning to the magnetoelastic ribbon we look for isolating neighborhoods in Y under the multivalued dynamical system **F**. We begin by choosing $C_0 \subset \mathbf{G}$. There are a variety of criteria that can be used in choosing C_0, the important point is that it must be a strict subset of **G**. We now begin the following procedure. Define $C_1 = C_0 \cap \mathbf{F}(C_0) \cap \mathbf{F}^{-1}(C_0)$ where $\mathbf{F}^{-1}(C_0) = \left\{G \in \mathbf{G} \middle| \mathbf{F}(G) \cap C_0 \neq \varnothing\right\}$. Now delete a component of C_1 which touches the boundary of C_0 relative to Y. This two step procedure is repeated until $C_{n+1}=C_n$. The resulting set C_n is an isolating neighborhood for **F** (see Figure 24). Observe that C_n consists of 4 disjoint sets labelled N_i, i=1,2,…

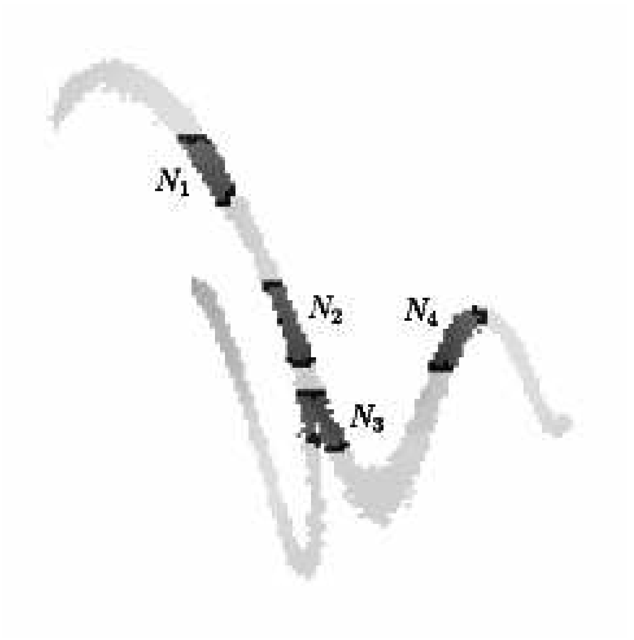


Figure 24 The four shaded regions labelled N_1,...N_4 make up the set C_n. The darkly shaded regions on the boundary of the N_i are L.

Returning to the theoretical level we need to resolve the following issue. We can compute isolating neighborhoods, but it is the structure of the corresponding isolating invariant set that we are interested in. To pass from isolating neighborhoods to the structure of the invariant set we will use the Conley index.[86] We only need the following small portion of the theory for our purposes. Given an isolating neighborhood N of g, a pair of closed bounded sets (K,L) with $L \subset K \subset N$ is an index pair if the following conditions are satisfied:

- $x \in K$ and $g(x) \in N$, then $g(x) \in K$

- $x \in L$ and $g(x) \in N$, then $g(x) \in L$
- $x \in K$ and $g(x) \notin N$, then $x \in L$
- the maximal invariant set in *N* is a subset of the interior of *K**L*.

The importance of an index pair is that the homology groups $H_*(K,L)$ and a homology map $\tilde{g}_* : H_*(K,L) \to H_*(K,L)$ induced by *g* are invariants of the maximal invariant set contained in *N*. If under a change in the dynamics *N* remains an isolating neighborhood then the homology group and map does not change.

Returning to the multivalued dynamics of **F**, we produce an index pair as follows. Let *L* consist of the elements of $\mathbf{F}(C_n)$ which touch the boundary of C_n relative to *Y*. Let $K = C_n \cap L$. Then, *(K,L)* is an index pair for **F**. Figure 24 indicates the resulting index pair.

We now compute $H_*(K,L)$ using $\mathbf{Z}_2$ coefficients and determine that it is a four dimensional vector space and the corresponding map on homology is the matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad (20)$$

At this point we can describe the essential difference between this development and that based on the embedding theorems mentioned earlier. The embedding theorems assume that the points $U \subset \mathbf{R}^2$ actually represent elements of trajectories of a fixed smooth map on a subset of $\mathbf{R}^2$ and that the dynamics of this map can now be embedded into the dynamics of the physical system. We make no such assumption. Instead we use the existence of the continuous map $\gamma : X \to \mathbf{R}$ to lift the algebraic topological quantities associated with *A* to *X*. This allows one to conclude that *A* represents a transition matrix for a symbolic coding of the dynamics in the physical phase space.

At the risk of being redundant this implies that given any small random environmental effects on the experiment and any sequence $b \in \Sigma_A$, there exists an initial condition for the physical system such that using our observational method the trajectory will, up to experimental error, pass through the regions labeled N_i, i=1,..4 in the manner indicated by *b*. For example the sequence ...4132224134132413... lies in Σ_A and hence there exists an initial condition for the ribbon such that our observations would indicate that up to experimental error its reconstructed trajectory lies in the following sequence of regions

$$N_4 \to N_1 \to N_3 \to N_2 \to N_2 \to N_2 \to N_4 \to N_1 \to N_3 \to N_4 ... \quad (21)$$

5.4.4. Conclusion

Our method provides an explicit description in terms of symbolic dynamics of the chaotic behavior of the magnetoelastic ribbon. This is a much finer description of the dynamics than is usually presented from experimental data. For example, given *A* it is easy to conclude that the topological entropy for the ribbon must be greater than ln 1.4656.

Two other methods that are commonly used to analyze chaotic data involve the approximation of Lyapunov exponents or the determination of a fractal dimension. Both the determination of the Conley index and of Lyapunov spectra require some similar assumptions:

- The data provided is a reasonable approximation to what one would obtain if measurements could be performed for an infinite amount of time;
- Phase space reconstruction provides a good approximation to the underlying dynamics;
- The underlying system is governed by a deterministic set of equations.

For the computation of Lyapunov spectra one must assume that the reconstruction has been done in a sufficiently high dimensional space. In our setting this is no longer an assumption. In particular, whenever one computes nontrivial algebraic topological quantities the resulting conclusions about the dynamics are correct. Of course, if one chooses a dimension that is too low, then the resulting multivalued map will either not contain any nontrivial isolating neighborhoods or the corresponding algebra will be trivial. It is easy to construct examples where the embedding fails but the algebra is still nontrivial.

An even more fundamental difference is that the Lyapunov exponent is a global quantity that is highly dependent on noise and very sensitive to perturbations. What's more a lot of questions about Lyapunov exponents are unanswered. How much noise and parameter drift can there be before any estimate of the exponents becomes meaningless? The Lyapunov exponent is limited in what it can tell us about the dynamics. In its traditional form, it is a global quantity that tells nothing about what sort of changes occur in the local dynamics. Also it does not provide us with significant information for use in a symbolic dynamics. This sort of description which we provide may be useful in applications such as control theory.

On a more concrete level we performed Lyapunov exponent calculations and embedding dimension calculations on this data set. The dominant Lyapunov exponent seemed to be roughly 0.48. However, under reasonable parameter regimes, the computed Lyapunov exponent ranged between 0.45 and 0.99. This indicates that computation of the Lyapunov exponent is extremely parameter dependent.

The same problems exist with a determination of fractal dimension. The dimension tells us little about the underlying dynamics. In addition, a reasonable estimate of dimension is very difficult to compute from experimental data. In theory, because it is a limit that is reached only with infinite data, the fractal dimension is not a computable object.

Measurement of the embedding dimension also yielded somewhat questionable results. This algorithm identifies the percentage of false nearest neighbors for a given embedding dimension. Although we found that an embedding dimension of three should be sufficient, the minimum embedding dimension differed greatly depending on our criteria for identification of a false nearest neighbor. In some analyses, it seemed an embedding dimension of 5 was necessary to decrease the percentage of false nearest neighbors to less than 5%. As noted before, the symbolic dynamics approach implemented here does not have these problems.

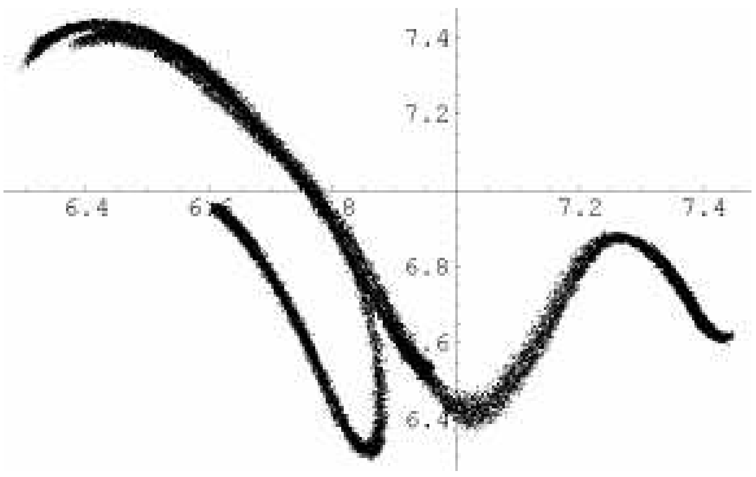


Figure 25 Time delay plot of u_n=(v_n, v_{n+1}), n=70,001,..., 100,000, where v_n is the nth voltage.

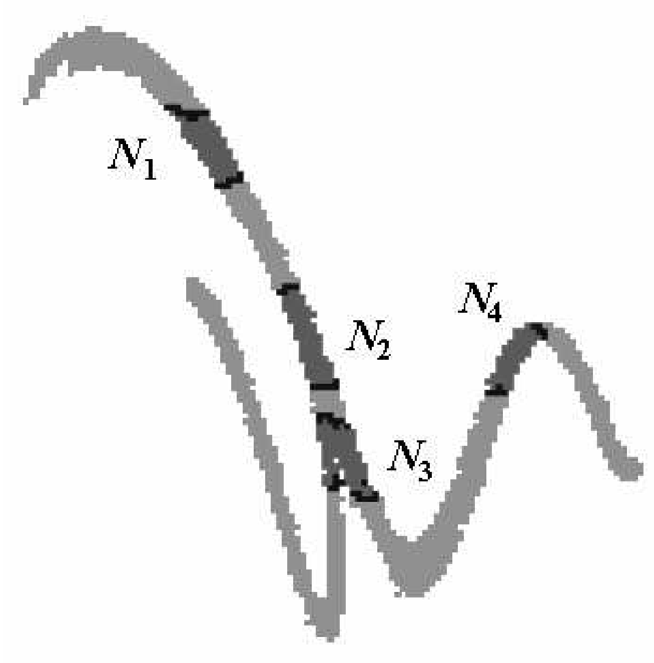


Figure 26 The four shaded regions labelled N_1,...N_4 make up the set C_n. The darkly shaded regions on the boundary of the N_i are L.

This approach appears to provide robust repeatable conclusions. We repeated the above mentioned proceedure using the points $\{v_n \mid n = 70001,...100000\}$. As one can see from Figure 25 there are observable differences in this collection of data points. After applying our procedure we obtain the index pairs indicated in Figure 26 which are also slightly different. However, on the level of the algebra and hence the symbolic dynamics we obtained the same transition matrix given in Equation (20).

In conclusion, we have proposed a theoretically justified and experimentally validated method which takes time series data as an input and produces the output of a transition matrix and its associated regions in reconstruction space which may be used to rigorously verify chaos, analyze and identify invariant sets, and determine properties of the global dynamics above the noise level. The power in this method is that the noise has been taken into account before any analysis is done. All analysis is on a scale where the results of the analysis are robust with respect to noise. Where commonly used, quantitative measures of chaos (such as Lyapunov exponent estimates) may fail because of sensitivity in the analysis. In contrast verification of chaos from analysis of the transition matrix is robust.

6. Computation of the mutual information function from multidimensional time series.

6.1. Introduction

This work builds on the method of computing mutual information that was suggested by Fraser and Swinney.[42] In that work, they suggested a method of estimating the mutual information between two one-dimensional time series. The mutual information of two random variables, $X \in \mathrm{X}$ and $Y \in \Psi$, is given by

$$I(X;Y) = \sum_{x \in \mathrm{X}} \sum_{y \in \Psi} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \qquad (22)$$

where the convention $0 \log 0 = 0$ is used. The logarithm is typically taken to base 2 so that the mutual information is given in units of bits. This provides a measure of the amount of information that a measurement of *X* contains regarding the measurement of *Y*, and vice-versa. For instance, if *X* and Y are completely independent, then I(*X*;*Y*)=0. On the other hand, if *X* and Y have equal probability mass functions, $p(x) = p(y)$, then the mutual information function assumes its maximum value,

$$I(X;X) = -\sum_{x \in \mathrm{X}} p(x) \log p(x) \qquad (23)$$

which is the Shannon entropy of *X*. This definition has a straightforward extension to the case of continuous random variables, with probability densities f(x) and f(y),

$$I(X;Y) = \int f(x,y) \log \frac{f(x,y)}{f(x)f(y)} dx\, dy \qquad (24)$$

Computation of the mutual information function from a time series of finite length where the full range of dynamics are not known becomes a challenging problem. In Ref. 42, 67 and 80, Fraser and Swinney gave a powerful method of estimating the mutual information of two time series, and examined some of its uses in phase space reconstruction. In particular, they conjectured on how it might be extended to a computation of the mutual information, I_n, common to *n* time series.

"I_n is the number of bits that are redundant in a vector measurement (I_1 is the old mutual information). The redundancy occurs because knowledge of some components of a vector measurement can be used to predict something about the other components. In an algorithm that calculated I_n, elements would be subdivided into 2^{n+1} parts at each level instead of just 4 and the statistical test would be altered, but otherwise the algorithm could be the same as the present version. If the vector were a time-delay reconstruction, plots of $I_n(T)$… could be used to choose delays for higher-dimensional reconstructions. If n were large enough (larger than the number of degrees of freedom), I_n should be a monotonically decreasing function of T, and the slope of I_n/n should give an estimate of the metric entropy."[42]

This work picks up where they left off. In the following section, we describe an efficient method of computing I_n.

6.2. Method

The data is described as a series of vectors $\overline{X_1}, \overline{X_2}, ..., \overline{X_N}$ where each vector is n dimensional, $\overline{X_i} = (X_i^1, X_i^2, ... X_i^n)$, and the number of vectors, N, is a power of two. The vectors are assumed to be sampled uniformly in time. That is, at each time interval Δt, a new vector is sampled, and Δt is a constant. Alternately, the data may be described as n time series data sets, $\overline{\overline{X_1}}, \overline{\overline{X_2}}, ..., \overline{\overline{X_n}}$, where each data set consists of N, data points, $\overline{\overline{X_i}} = (X_1^i, X_2^i, ... X_N^i)$. Both notations are used depending on the situation. For the sake of simplicity, we use a slightly different notation for the generalized mutual information than was suggested by Fraser and Swinney. The generalized mutual information is defined as

$$I_n = I(X_1, X_2, ... X_n) = \sum p(x_1, x_2, ... x_n) \log \frac{p(x_1, x_2, ... x_n)}{p(x_1) p(x_2) ... p(x_n)} \quad (25)$$

For the case of time series data this may be written as

$$I_n = I(\overline{\overline{X_1}}, \overline{\overline{X_2}}, ..., \overline{\overline{X_n}}) = \sum_{x_1 \in \overline{\overline{X_1}}} \sum_{x_2 \in \overline{\overline{X_2}}} ... \sum_{x_n \in \overline{\overline{X_n}}} p(x_1, x_2, ...x_n) \log \frac{p(x_1, x_2, ...x_n)}{p(x_1)p(x_2)...p(x_n)} \quad (26)$$

so that I_2 is the mutual information as typically defined. One important issue is determining the range of possible values for a point $X_i^j \in \overline{\overline{X_j}}$, and the associated probability mass function. That is, if the mutual information is defined for alphabets X and Ψ, with associated probability mass functions $p(x) = \Pr\{X = x\}, x \in \mathrm{X}$ and $p(y) = \Pr\{Y = y\}, y \in \Psi$, then X and Ψ need to be chosen. It is somewhat of an arbitrary choice, since the data set is confined to at most *N* distinct values, yet the full range of values may be infinite. The algorithm involves gridding the *n* dimensional space into *n* dimensional hypercubes and treating each occupied hypercube as an allowable symbol. Thus the resulting value of mutual information depends on our gridding choice. However, mutual information is most useful in relative comparisons. The criteria used for our gridding choice are the consistency and reliability of results, as well as the efficiency of computation. Therefore we follow Fraser and Swinney's suggestion of gridding each dimension into equiprobable partitions. This allows us to use the uniform KTree described in Chapter Three, Section 7. It has the added benefit that $p(x_1) = p(x_2)... = p(x_n)$.

The generalized mutual information is computed through a traversal of the tree. The levels of the tree define successive partitions of the *n*-dimensional space, $G_0, G_1, ...G_{K-1}$. For a given level of the tree G_m, the space is partitioned into 2^{mj}

hypercubes, $R_m(0), R_m(1), \ldots R_m(2^n m-1)$ such that the hypercube $R_m(j)$ may be partitioned into $R_{m+1}(2^n j), R_{m+1}(2^n j+1), \ldots R_m(2^n j+2^n-1)$. Each hypercube has an associated probability $P(R_m(j))$, which is simply the number of vectors in that hypercube divided by the total number of vectors, $N_m(j)\ /N$. Thus the n-dimensional mutual information may be estimated for any level m of the tree.

$$i_m = \sum_{j=0}^{2^{nm}-1} P(R_m(j)) \log \frac{P(R_m(j))}{P_1(R_m(j))P_2(R_m(j))\ldots P_n(R_m(j))} \quad (27)$$

where $P_i(R_m(j))$ is the probability of finding the ith coordinate of a vector to reside in the same partition along the ith direction as $R_m(j)$. Due to the equiprobable nature of the partitions, $P_i(R_m(j)) = \frac{1}{2^m}$ for all i and j. Hence

$$i_m = mn + \sum_{j=0}^{2^{nm}-1} P(R_m(j)) \log P(R_m(j)) \quad (28)$$

Note that the contribution to i_m of $R_m(j)$ is $mnP(R_m(j)) + P(R_m(j)) \log P(R_m(j))$ and the contribution to i_{m+1} of $R_m(j)$ is $(m+1)nP(R_m(j)) + \sum_{k=2^n j}^{2^n(j+1)-1} P(R_{m+1}(k)) \log P(R_{m+1}(k))$. So in going from i_m to i_{m+1} a $nP(R_m(j))$ is added and the $P(R_m(j)) \log P(R_m(j))$ term is replaced by $\sum_{k=2^n j}^{2^n(j+1)-1} P(R_{m+1}(k)) \log P(R_{m+1}(k))$. If the mth level has no substructure, then $P(R_{m+1}(2^n j)) = \ P(R_{m+1}(2^n j+1)) = \ldots\ P(R_{m+1}(2^n(j+1)-1)$. So the contribution to i_{m+1} of $R_m(j)$ is the same as the contribution to i_m of $R_m(j)$. So, $I_n = F_1(R_0(0))$ where,

$$F_1(R_m(j)) = P(R_m(j))\log P(R_m(j)) \text{, if no substructure} \qquad (29)$$

$$F_1(R_m(j)) = nP(R_m(j)) + \sum_{k=2^n j}^{2^n(j+1)-1} F(R_{m+1}(k)) \text{, if there is substructure.}(30)$$

Fraser and Swinney suggest a χ-squared test for substructure. However, this is a source of error and is unnecessary. Instead, we choose to stop the calculation when no further subdivision of $R_m(j)$ is possible, and hence we are guaranteed no substructure.. This is the case if $N(R_m(j)) \le 2$, since 2 points will always be partitioned into 2 different cubes and one point can not be partitioned further. So, $I_n = \frac{F(R_0(0))}{N} - \log N$ where,

$$F(R_m(j)) = 0 \text{, if } N(R_m(j)) < 2 \qquad (31)$$

$$F(R_m(j)) = 2 \text{, if } N(R_m(j)) = 2 \qquad (32)$$

$$F(R_m(j)) = nN(R_m(j)) + \sum_{k=2^n j}^{2^n(j+1)-1} F(R_{m+1}(k)) \text{ if } N(R_m(j)) > 2 \,(33)$$

Each node $R_m(j)$ in the tree contains $N(R_m(j))$ and pointers to $R_{m+1}(2^n j), R_{m+1}(2^n j+1), \ldots R_m(2^n j + 2^n - 1)$. The tree is traversed from left to right using equations (31)-(33) to keep a running sum of the mutual information. The minimum value of the generalized mutual information occurs when $\overline{\overline{X_1}}, \overline{\overline{X_2}}, \ldots, \overline{\overline{X_n}}$ are completely independent. In which case, $p(x_1, x_2, \ldots x_n) = p(x_1)p(x_2)\ldots p(x_n)$ and $I_n = 0$. The maximum value of the mutual information occurs when $\overline{\overline{X_1}} = \overline{\overline{X_2}} = \ldots \overline{\overline{X_n}}$. In which case $p(x_1, x_2, \ldots x_n) = p(x_1)p(x_2)\ldots p(x_n)$ and

$$
\begin{aligned}
I_n(X_1, X_1, \ldots X_1) &= -\sum (n-1)\, p(x_1) \log p(x_1) \\
&= -\sum_{j=0}^{2^{n(K-1)}-1} (n-1) P(R_K(j)) \log P(R_K(j)) \qquad (34) \\
&= \sum_{j=0}^{2^{n(K-1)}-1} (n-1) P(R_K(j))(K-1) = (n-1)(K-1)
\end{aligned}
$$

6.3. Implementation

The following functions are used by the Mutual Information routine.

```
struct MutualNode *BuildMutualTree(struct MutualNode *p,int
**i_array,int NumberPts,int NumberDim,int
NumberCubes,int MaxLevel,int Value,int Level,int *Cut);
void TraverseMutualTree(struct MutualNode *p,int NumberDim,int
NumberCubes,double *F);
void FreeMutualTree(struct MutualNode *p,int NumberCubes);
double MutualInformation (int **i_array,int NumberPts,int
NumberDim);
```

and the following functions are used to sort a matrix for use by the Mutual Information routine.

```
int InverseSortVector(double *array,int NumberPoints,int
*arraysort,int *arrayinversesort);
int InverseSortMatrix(double **array,int NumberPoints,int
NumberDim,int **arraysort,int **arrayinversesort);
int InverseCompare(const void *vp,const void *vq);
```

6.4. Multichannel mutual information of simulated systems

6.4.1. Lorenz attractor

The Lorenz system[25] is a simple model of Rayleigh-Benard instability. The Lorenz equations are given by:

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= r \cdot x - y - x \cdot z\\ \dot{z} &= x \cdot y - b \cdot z\end{aligned}$$

where σ=10, b=8/3, and r=28. Figure 27 demonstrates the effect of scaling on the data. Dense regions in the data are spread out, while sparse regions are compacted. Although absolute distances are not preserved, relative distances are, thus making computation of the mutual information possible.

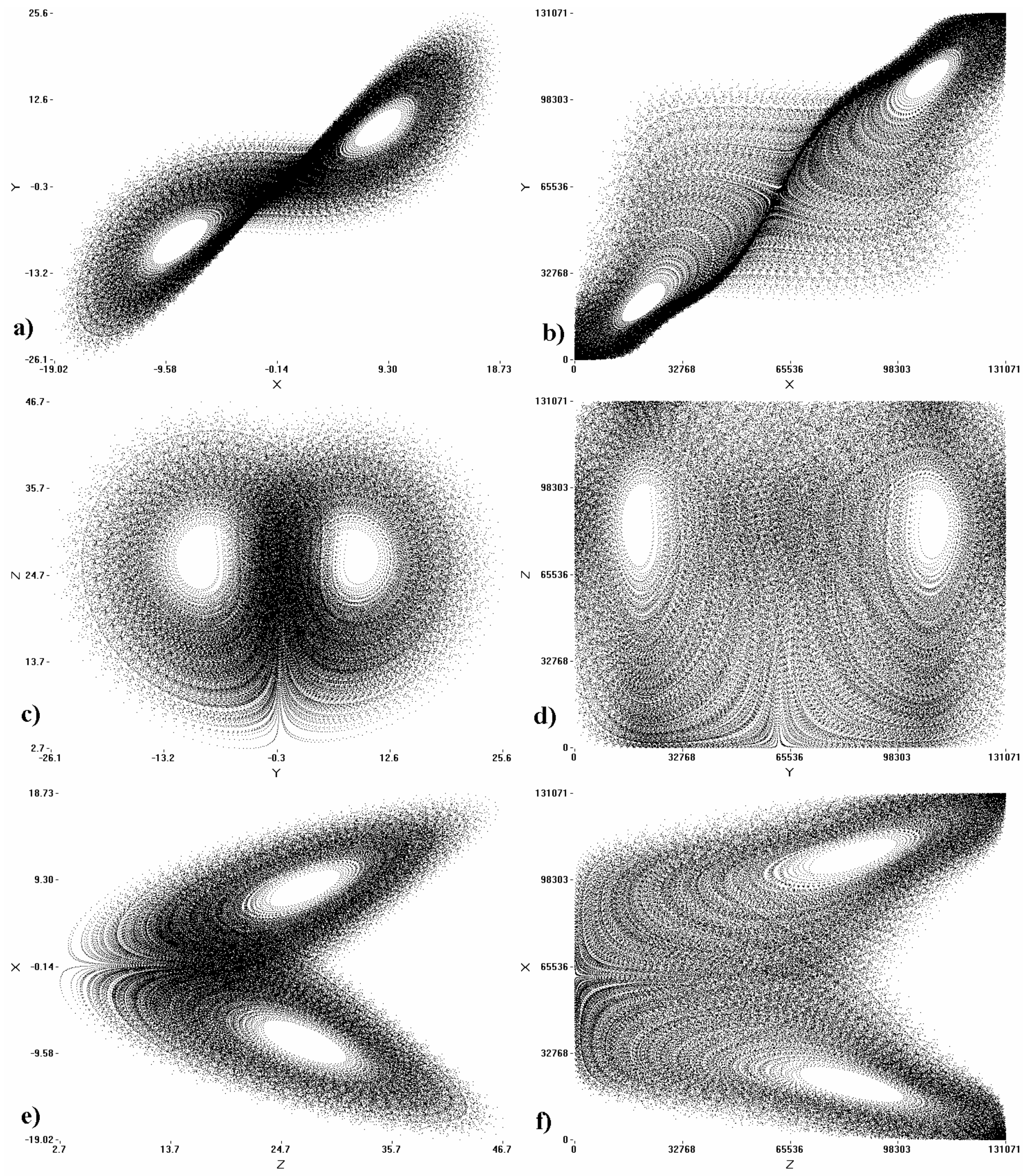


Figure 27. The Lorenz attractor before and after sorting. a) **is y vs x from the original data,** b) **y vs x from the transformed data,** c) **z vs y from the original data,** d) **z vs y from the transformed data,** e) **x vs z from the original data, and** f) **is x vs z from the transformed data. Each plot contains 2^{17} points with a time step of 0.002.**

Figure 28 depicts mutual information as a function of time step for several data sets from the Lorenz system. This illustrates how mutual information can be used to determine whether enough data has been collected in order to map out the structure of the system. A sufficiently large time step implies that the proximity in space of nearby points in time won't distort the measurements of information. Similarly, a sufficiently large data set succeeds in minimizing the error, as can be seen by the considerably larger error in measurements from the smaller data set in Figure 28.

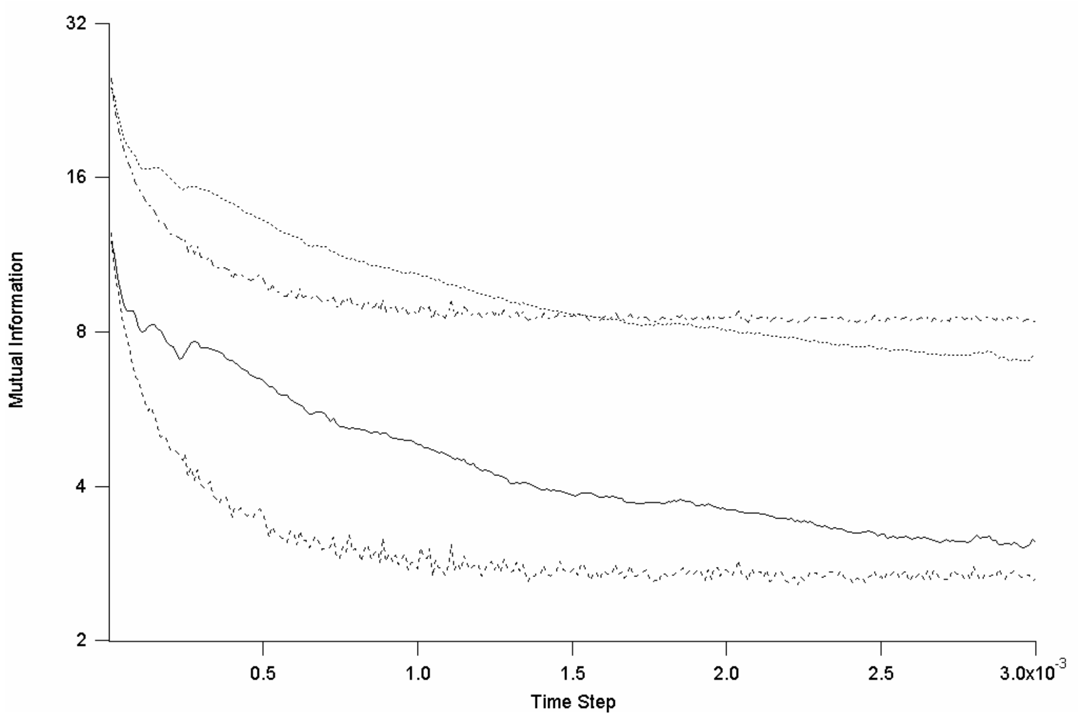


Figure 28. Mutual information as a function of time step for several data sets from the Lorenz system, the dashed-dotted line represents 2^{18} data points of the 3 dimensional data (x, y, and z coordinates), the dotted line represents 2^{13} data points of the 3 dimensional data, the dashed line represents 2^{18} data points of the x and y coordinates and the filled line represents 2^{13} data points of the x and y coordinates. The y-axis of the graph is on a log scale in order to accentuate the structure of each plot.

Figure 29 depicts mutual information as a function of delay for the Lorenz system. The first minimum of the mutual information function is easily identified from two dimensional embeddings, but less easily identified using a three dimensional embedding. This is because the information provided by the third coordinate is sufficient to guarantee that a large amount of information is always shared between the coordinates.

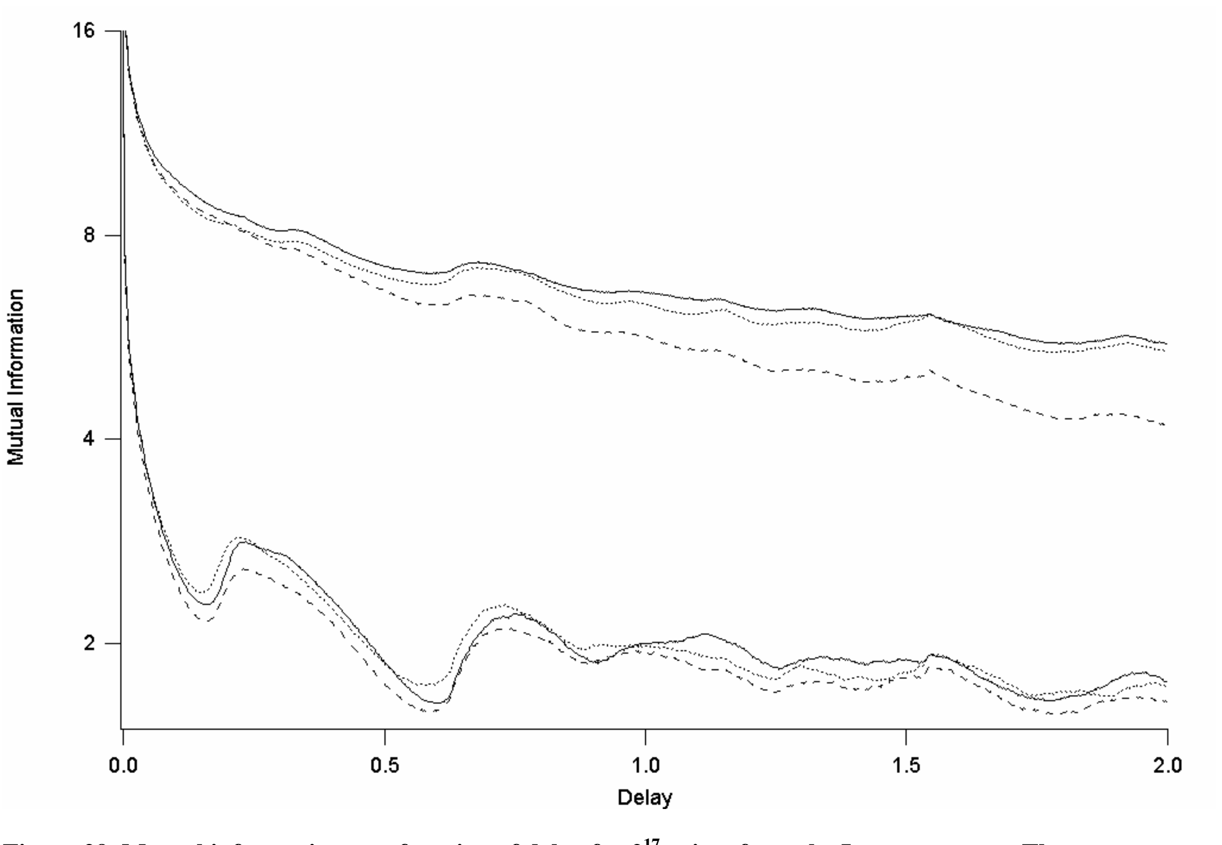


Figure 29. Mutual information as a function of delay for 2^{17} points from the Lorenz system. The sampling rate is 0.002 and the delay is given in terms of the time step. The y-axis is on a log scale in order to help distinguish the plots. The top straight line is from a 3D embedding of the x data, the top dotted line from a 3D embedding of the y data, the top dashed line from a 3D embedding of the z data, the bottom straight line from a 2D embedding of the x data, the bottom dotted line from a 2D embedding of the y data, and the bottom dashed line is from a 2D embedding of the z data.

6.4.2. Rossler chaos

The second system is the Rossler equations.[87]

$$
\begin{aligned}
\dot{x} &= -z - y \\
\dot{y} &= x + 0.15 \cdot y \\
\dot{z} &= 0.2 + z(x - 10)
\end{aligned}
$$

The effects of scaling are depicted in Figure 30. Here the distortion is dramatic when compared with the results from the Lorenz system (Figure 27). This is because there is a very strong correlation between the y and z coordinates, as well as between the x and z coordinates. Scaling the data removes much of this correlation so that mutual information can be measured effectively.

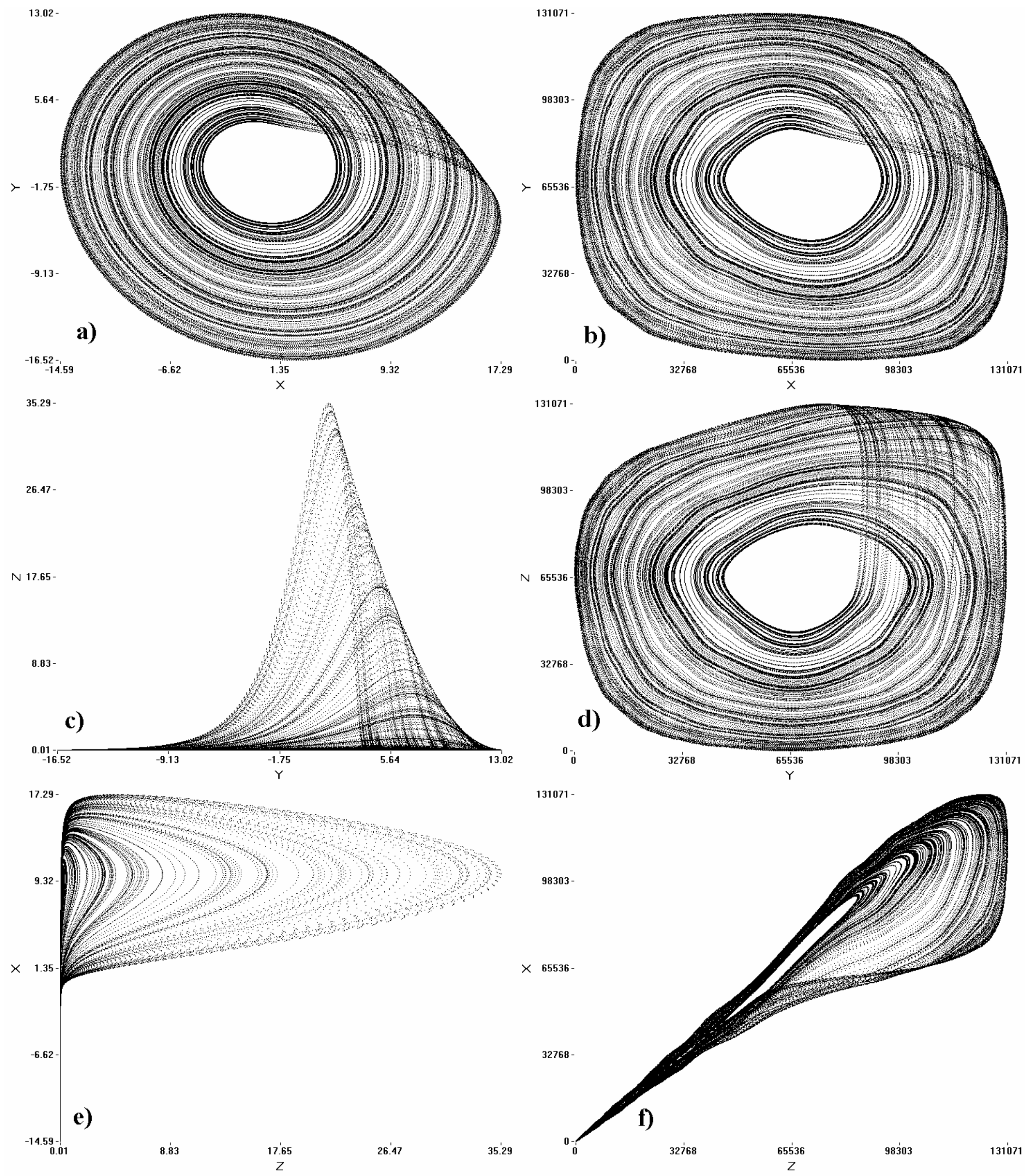


Figure 30. The Rossler attractor before and after sorting. a) **is y vs x from the original data,** b) **y vs x from the transformed data,** c) **z vs y from the original data,** d)**z vs y from the transformed data,** e) **x vs z from the original data, and** f) **is x vs z from the transformed data. Each plot contains 2^{17} points with a time step of 0.002.**

Figure 31 depicts mutual information as a function of time step for several data sets from the Rossler system. Compared with the Lorenz system, time step and data set size have a far greater effect on error for the Rossler system. This is because the Rossler system has regions that are very rarely visited, so that small differences in data sets may greatly effect how that region's density is measured. This is less of a problem for the Lorenz system where a relatively small portion of the attractor can give a good estimate of the overall distribution

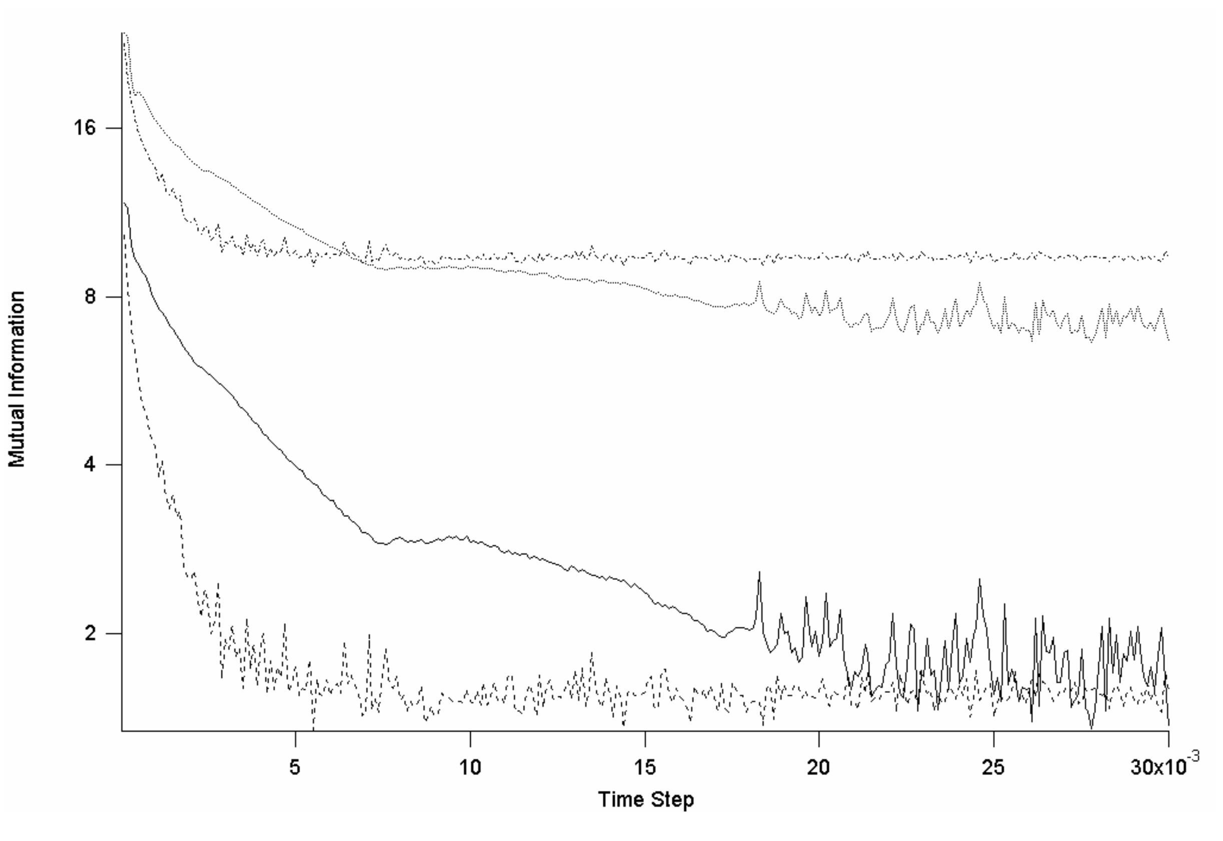


Figure 31. Mutual information as a function of time step for several data sets from the Rossler system, the dashed-dotted line represents 2^{18} data points of the 3 dimensional data (x, y, and z coordinates), the dotted line represents 2^{13} data points of the 3 dimensional data, the dashed line represents 2^{18} data points of the x and y coordinates and the filled line represents 2^{13} data points of the x and y coordinates. The y-axis of the graph is on a log scale in order to accentuate the structure of each plot.

Figure 32 depicts mutual information as a function of delay for the Rossler system. This is similar to a plot in Fraser, et. al.,[42] Like the Lorenz system, minima are less easily identified for three dimensional embeddings. One implication of this is that the delay time is less important for high dimensional embeddings.

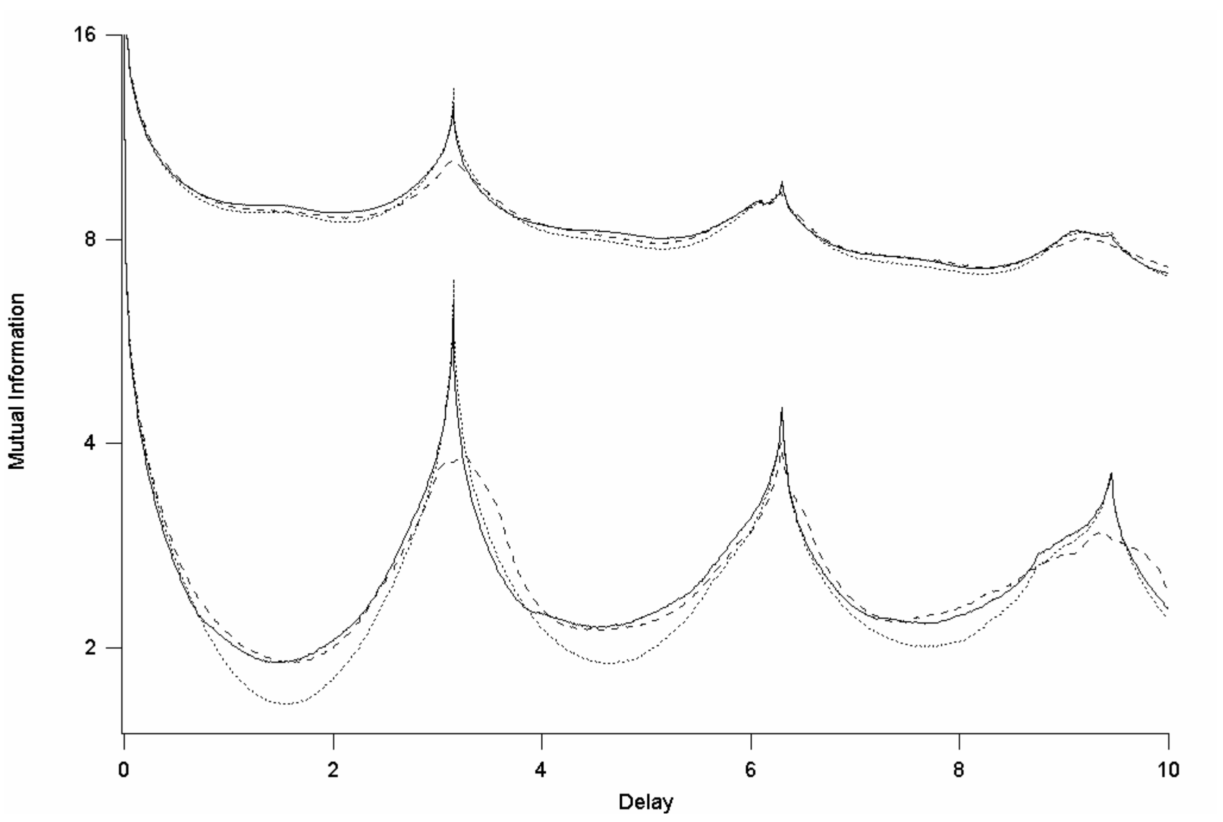


Figure 32. Mutual information as a function of delay for 2^{17} data points from the Rossler system the sampling rate is 0.01 and the delay is given in terms of the time step. The y-axis of the graph is on a log scale in order to help distinguish the plots. The top straight line is from a 3D embedding of the x data, the top dotted line is from a 3D embedding of the y data, the top dashed line is from a 3D embedding of the z data, the bottom straight line is from a 2D embedding of the x data, the bottom dotted line is from a 2D embedding of the y data, and the bottom dashed line is from a 2D embedding of the z data.

CHAPTER FIVE

RESULTS OF ANALYSIS ON VARIOUS DATA

1. Introduction

Time series analysis of high dimensional experimental data is a difficult task. Many methods of analysis fail when performed on data for which there is little a priori knowledge of the underlying dynamics. The true test of the viability of an analysis method is how well it performs on experimental data. In this chapter, we look at the success of analysis methods used on both simulated data (from well-known systems and from unexplored complex models) and from experimental data. Specifically, the data was examined for evidence of chaotic and fractal dynamics. Extracting fractal dimension and Lyapunov spectra from noisy data is known to be problematic, so the results were interpreted with skepticism and validity criteria was established for the results.

2. The Pulse Thermal Combustor

2.1. Introduction

From the theoretical work by G. A. Richards *et al.*,[89] it was shown that the thermal pulse combustor can have a number of advanced fossil energy applications. The pulse combustor engine had its start in the 1940's when Germany used them to propel the V-1 rockets. Some of its descendants are still used today in rockets and weapon propulsion systems. Certain heating systems use this engine as the main burner since it produces more efficient burning than the standard burner. Some power generating stations also use similar engines for the boilers.

Recent work on this combustor has shown that the engine has complex dynamics ranging from steady combustion to regular pulsating combustion to irregular pulsating combustion to flame-out or extinction. Daw *et al.* have shown extensively (in both the model and the experimental system) that the thermal pulse combustor can produce chaotic dynamics.[88, 90] Previously this was neglected due to the lack of recognition of chaos as a separate state and the lack of tools to help identify it. With the existence of chaos also comes the possibilities of other behaviors that are catastrophic and undesirable such as the flaming out or quitting of the engine.

In the chaotic system it is often found that the chaotic attractor suddenly ceases to exist when a parameter has changed beyond a critical point. The system catastrophically falls into a new attractor with less desirable properties. Once the system is in this new attractor, even changing the parameter back to a region where chaos previously occurred can not reverse the sequence of events. This sudden destruction of the chaotic attractor in favor of a steady-state or periodic motion is called a boundary crisis. As a system parameter p is changed monotonically toward a critical parameter p_c, the basin of attraction of the undesirable orbit approaches the basin of attraction of the chaotic attractor. For $p = p_c$, the two basins touch, making pathways for the orbit to exit the chaotic attractor. For $p > p_c$, the chaotic attractor is replaced by a chaotic transient. For initial conditions that place the system in a region of phase space within the basin of the chaotic attractor, the system will pull in and evolve into what seems like the chaotic attractor. After some time the system exits this chaotic transient and moves to the other attractor.

In the thermal pulse combustor, lowering the fuel/air ratio parameter will cause the engine to pulse chaotically. Reducing the parameter further will cause the engine to suddenly flame-out. If one were to increase the parameter to where the engine was pulsing before, the engine would not reignite. But the parameter close to where the system makes the transition to flame-out has desirable characteristics if one can keep it running at that point. First, the engine is running very lean because the fuel to air ratio is lower than when it pulsed regularly. By running the engine lean one can save fuel, lower atmospheric emissions (due to less unburned fuel), and increase efficiency because of the large amplitude variation in the pressure inside the combusting chamber. This large pressure variation also increases the heat transfer rate because the wall temperature of the combustion chamber increases dramatically.

2.2. Theory

The thermal pulse combustor considered here has been described previously by Richards *et al.*,[89] In *et al.*[91] and Daw *et al.*[90] The thermal pulse combustor consists of no moving parts. The two main components of the combustor are the combustion chamber and the acoustic resonator (tailpipe). The experimental combustion chamber has an interior diameter of 5.1 cm and internal volume of 295 cm^3. The tailpipe, which acts as an acoustic resonator, has an internal diameter of 2.0

cm and a variable length of 30.5 to 91.5 cm. By varying the length of the tailpipe, the dominant acoustic frequency can be adjusted, thus tuning how the combustion process and the acoustic resonator interact.

Gas propane fuel and air are injected via two opposing jets into the combustion chamber from its sides at the opposite end from the tailpipe as shown in Figure 33 and Figure 34. Inside the combustion chamber the jets are deflected to swirl the gases and to enhance mixing. The mixture is injected at a high pressure to ensure a choked flow, resulting in a constant mass flow into the combustion chamber even during the large pressure changes present in the pulsating mode. Choked flow is used to maintain constant air and fuel flow rates. The mass flow rate and equivalence ratio are controlled by varying either the supply pressures or the diameters of the choke orifices. A spark plug is used initially to ignite the mixture; however, once the combustor has warmed-up, the heat from the combustor wall and the recompression effect of the reversed flow from the tailpipe after the exhaust phase causes the mixture to auto-ignite, thus making the spark plug unnecessary.

The combustion chamber wall is maintained at a constant temperature with a convection constant *h*. After combustion is completed, the hot gas accelerates through the tailpipe and is exhausted into the surroundings. What makes the thermal pulse combustor unique is that unlike conventional pulse combustors, it uses neither mechanical nor aerodynamical valves to regulate the inlet fuel and air. The mixture flows at constant rate into the chamber. A conventional pulse combustor regulates its inlet fuel and air flows based upon the pressure inside the combustion chamber.

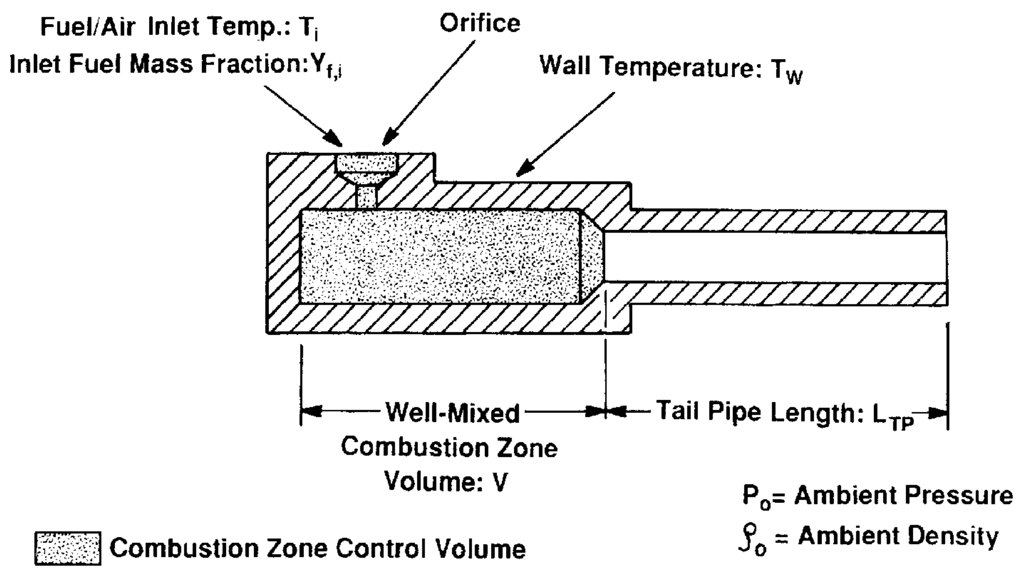

Fuel/Air Inlet Temp.: T_i
Inlet Fuel Mass Fraction: $Y_{f,i}$
Orifice
Wall Temperature: T_W
Well-Mixed Combustion Zone Volume: V
Tail Pipe Length: L_{TP}
P_o= Ambient Pressure
ρ_o = Ambient Density
Combustion Zone Control Volume

Figure 33 Geometry of the model thermal pulse combustor.

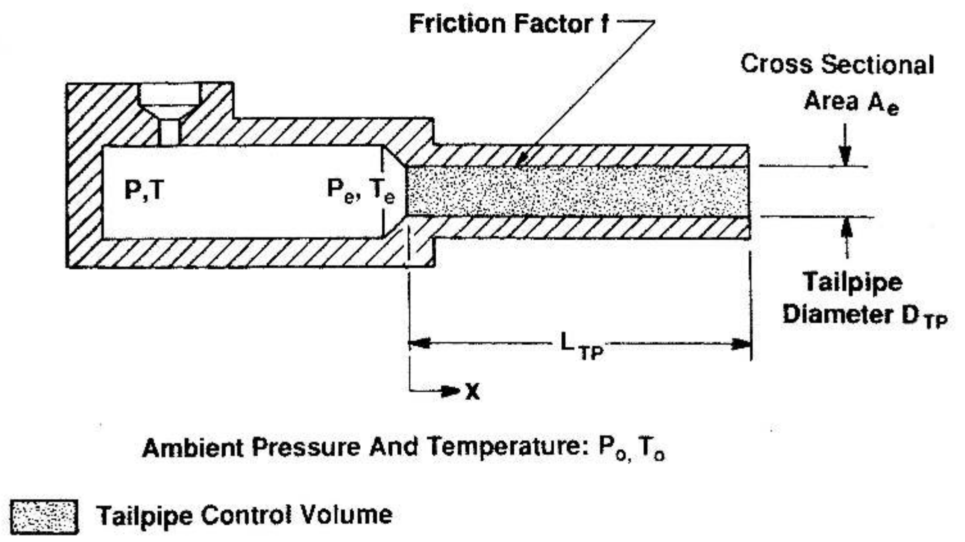

Friction Factor f
Cross Sectional Area A_e
P,T
P_e, T_e
Tailpipe Diameter D_{TP}
L_{TP}
X
Ambient Pressure And Temperature: P_o, T_o
Tailpipe Control Volume

Figure 34 Geometry of the model thermal pulse combustor. The combustion product is moved into the tailpipe.

2.3. Model

The thermal pulse combustor model was based on the assumptions that the gas in the combustion zone is perfectly stirred, the combustion process is controlled by a single-step, bimolecular reaction for air and propane, combustion gases behave as ideal gases with constant specific heats (C_v and C_p), kinetic energy of gas is negligible compared to the internal energy in the combustion zone; and flow in the tailpipe has properties equivalent to those prevailing at its entrance. These assumptions and conservation equations for momentum, mass, and energy lead to four normalized ordinary differential equations:

$$\frac{d\tilde{T}}{dt} = \gamma\left\{\frac{1}{\tau_f} + \frac{1}{\tau_{HT}} + \frac{1}{\tau_c}\right\}\frac{\tilde{T}}{\tilde{P}} - \left\{(\gamma-1)\frac{Z_e}{\rho_0} + \frac{1}{\tau_f} + \frac{\gamma T_0}{\tau_{HT} T_w}\right\}\frac{\tilde{T}^2}{\tilde{P}} \quad (1)$$

$$\frac{d\tilde{P}}{dt} = \gamma\left\{\frac{1}{\tau_f} + \frac{1}{\tau_{HT}} + \frac{1}{\tau_c}\right\} - \gamma\left\{\frac{Z_e}{\rho_0} + \frac{T_0}{\tau_{HT} T_w}\right\}\tilde{T} \quad (2)$$

$$\frac{dY_f}{dt} = \frac{\left(Y_{f,i} - Y_f\right)}{\tau_f}\frac{\tilde{T}}{\tilde{P}} - \left(\frac{C_p T_0}{\tau_c \Delta H}\right)\frac{\tilde{T}}{\tilde{P}} \quad (3)$$

$$\frac{d\tilde{U}_e}{dt} = \left(\tilde{P}_e - 1\right)\left(\frac{R T_0 \tau_f}{L_{TP} L_{C2}}\right)\frac{\tilde{T}_e}{\tilde{P}_e} - \frac{L_{C2} f}{2 D_{TP} \tau_f}\frac{\tilde{U}_e^{\,3}}{\left\|\tilde{U}_e\right\|} \quad (4)$$

$\tilde{T}$ is the combustion zone temperature normalized to the ambient temperature, $\tilde{P}$ is the combustion zone pressure normalized to the ambient pressure, Y_f is the mass of fuel per total mass inside the combustion zone (mass fraction), and $\tilde{U}_e$ is the tailpipe velocity normalized with the cold flow tailpipe velocity. The variable τ_f is the characteristic flow time or the combustion residence time for the combustion chamber and τ_{HT} is the characteristic heat transfer time. The τ_c is the characteristic combustion time, and it is the only time constant that is dependent on the state space variables. It is defined as

$$\tau_c = \left[A' \frac{\Delta H_f}{C_p T_0} \frac{\tilde{P}^2}{\tilde{T}^{3/2}} Y_f^2 \exp\left(\frac{T_{act}}{\tilde{T}} \right) \right]^{-1} \qquad (5)$$

The other variables are standard, as defined in Table 3.

Nomenclature	**Description**	**Initial Value**
γ	Ratio of spcific heats	1.27
τ_f	Flow time	0.026670 s
τ_{HT}	Heat transfer time	0.040110 s
τ_c	Combustion time	calculate in process
ρ_o	Ambient density	1.12 kg/m^3
T_o	Ambient temperature	300.0 K
T_w	Combustor wall temperature	1200.0 K
$Y_{f,\,i}$	Inlet fuel mass fraction	0.06
C_p	Specific heat for constant pressure	1200.0 J/kg-K
ΔH	Heat of reaction	4.6x10^7 J/kg
L_{c1}	First characteristic length	0.0119 m
L_{c2}	Second characteristic length	0.8486 m
f	Friction factor along tailpipe	0.03
h	Heat transfer coefficient	120.0 W/m^2-K
D_{TP}	Diameter of the tailpipe	0.0178 m
L_{TP}	Length of tailpipe	0.6100 m
V	Combustion volume	0.0001985 m^3
C_k	Specific heat at constant temperature	3.385x10^8
T_{ACT}	Dimensionless activation temperature	50.0
P_o	Ambient pressure	1.01325x10^5 Pa
A'	Kinetic constant for fuel reaction rate	3.85x10^8 s^{-1}
π	Usual π value	3.141592
τ_m	Maximum resident time for mixture in combustor chamber	0.0000 s
$\tilde{P}_e$	Pressure at tailpipe entrance	dimensionless
$\tilde{T}_e$	Temperature at tailpipe entrance	dimensionless

Table 3 Listing of the variables used in the model and initial values used in the numerical simulation. The subscript zero indicates the ambient condition and the subscript *e* indicates the condition at the tailpipe entrance.

The initial conditions corresponded to filling the combustion chamber with the fuel and air mixture to one atmospheric pressure. The initial high temperature of 5.0 times the ambient temperature ignites the fuel/air mixture. The combustion produces an immediate pressure rise and accelerates the hot gases toward the tailpipe. When the amount of gas quickly leaving the combustion zone exceeds the amount remaining in the combustion zone, pressure begins to drop. As a result of the hot gases moving rapidly down the tailpipe, it drops below the ambient pressure. The combined effect of cooling and pressure drop quenches the combustion reaction. The incoming fuel and air continues to mix in a relatively cool environment, along with the remaining combustion product in the chamber. The gas flow in the tailpipe slows, stops and then reverses direction. Recompression from the backflow and heat transfer from the combustion chamber wall combine to cause the temperature to rise as the fuel and air mixture builds up until reaching the critical point for combustion. At this point the combustion cycle starts once more.

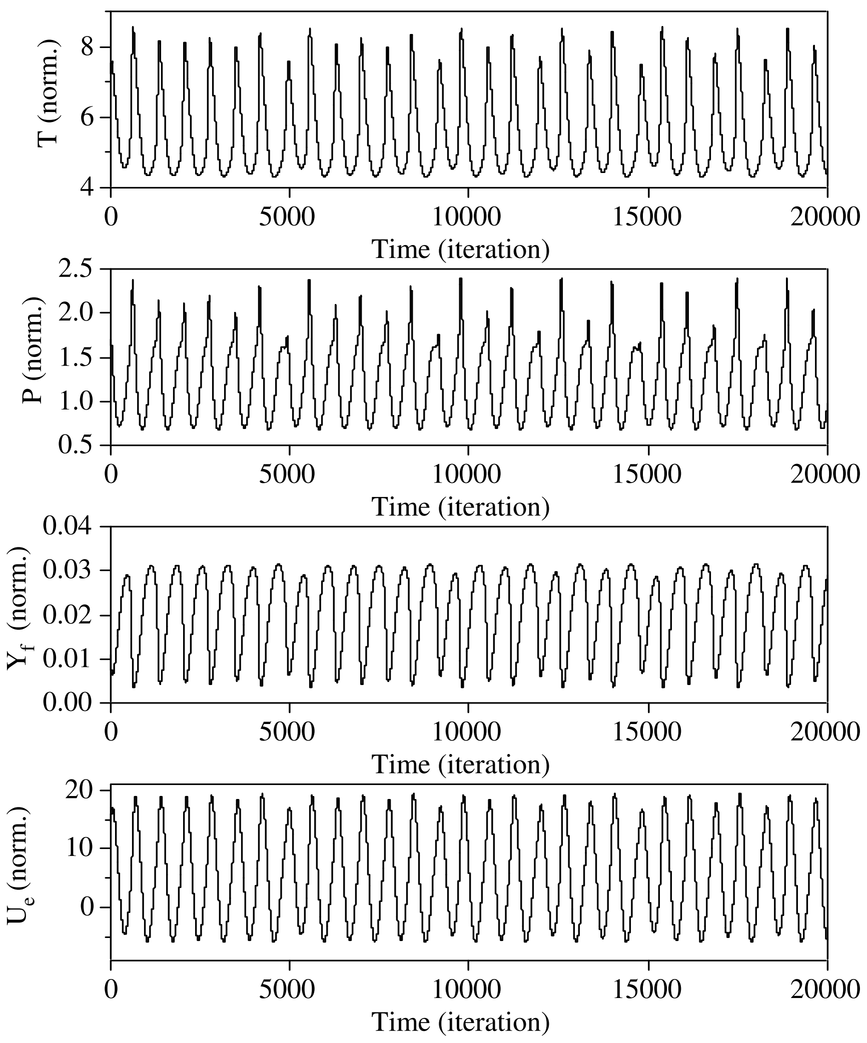


Figure 35. Time series of the four state variables. The parameter τ_f was set at 0.02670.

The differential equations, (1) - (4) were integrated using a 4th-order Runge-Kutta routine with a time step of $1.00\text{x}10^{-5}$ second. The initial conditions for the simulation are listed in Table 2. Figure 35 shows an example of the four state space variables during the pulsing mode, and Figure 36 shows the projection of the full phase space onto the temperature-pressure plane. Only the pressure variable was used since only the pressure can be measured during actual experimental runs. From the pressure time series a peak detection algorithm was used to pick off the maximum of each peak. The peak values were used to construct a return map by plotting the current peak versus the previous peak.

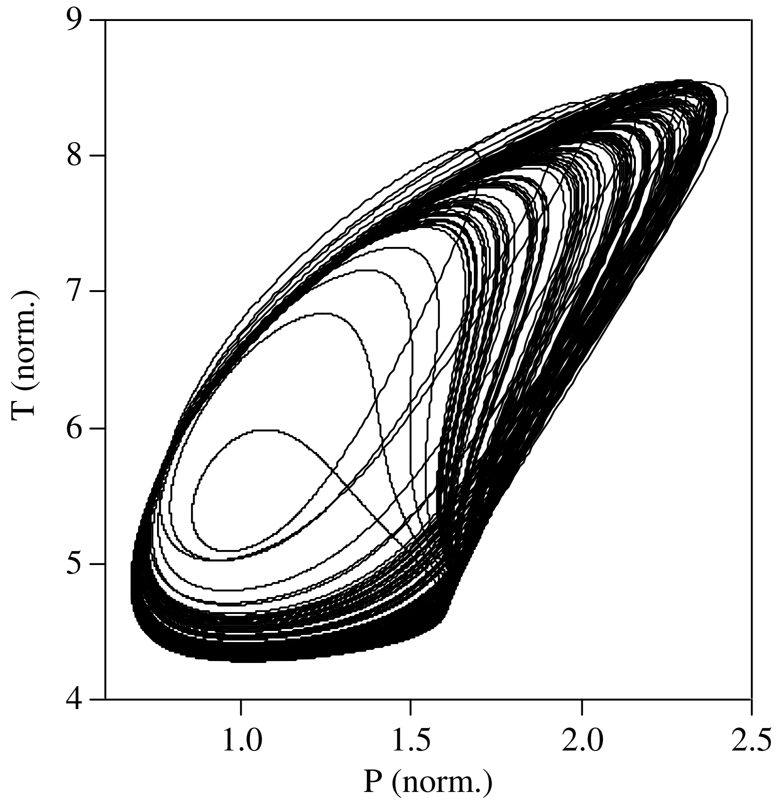


Figure 36. Projection of the time series in Figure 35 onto the temperature-pressure plane.

Through the course of this numerical experiment, only the characteristic resident time for the fuel/air mixture in the combustion chamber, τ_f, was varied. Since it is inversely proportional to the flow rate of the fuel/air mixture, lowering τ_f means increasing the flow rate. This parameter was used as the bifurcation parameter and as a perturbing parameter. In addition, this choice also has a relevance to the actual experiment because the flow rate is the parameter that will be used to manipulate system dynamics.

As τ_f was decreased (the fuel/air flow increased), the thermal pulse combustor model showed a period doubling route to chaos. Similar results were also obtained from previous work done on the model.[88, 92] However their observable variable was temperature instead of pressure. Decreasing τ_f led to a crisis with $\tau_{f,critical}$ at about 0.026696, and the system eventually crossed the boundary of the basin of attraction toward a stable fixed-point (flame-out state) at a much lower temperature (Figure 37). Once flame-out had occurred, reversing τ_f to a value much greater than $\tau_{f,critical}$ would not make the system pulse again.

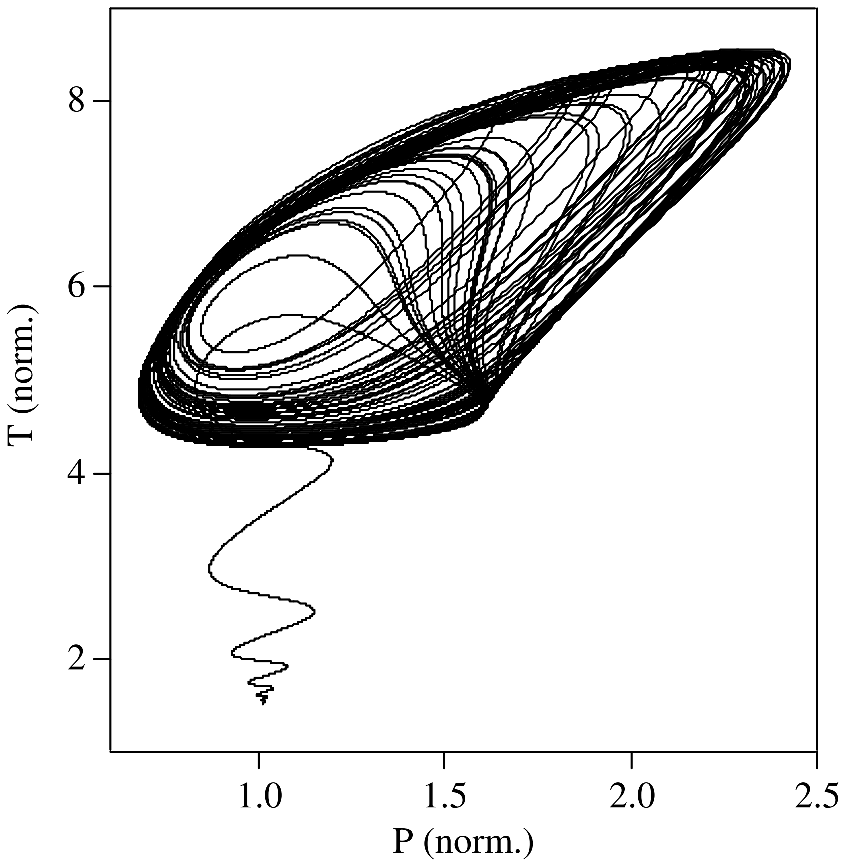


Figure 37. Projection of the flow onto the temperature-pressure plane. τ_f was slightly below the critical value for flame-out. The system evolved on a chaotic transient, then spiraled down toward the flame-out state.

Note that the fuel mass fraction of the pulse combustor was fluctuating around some mean value for regular pulsing. The fluctuations correspond to the combustion cycle and the rebuilding of the fuel/air concentration in the combustion chamber. For fluctuation in the chaotic parameter, the fuel mass fraction was oscillating around a lower mean value, which indicates a higher percentage of the fuel burns during combustion. Hence the combustion in the thermal pulse combustor is more efficient when it is operating in the chaotic mode. Similar behavior has been observed in the actual thermal pulse combustor. Thus, the hope of the study is to extend the operating range of the combustor to lower τ_f values and yet to prevent it from achieving flame-out. The method utilized here will be applicable to the actual thermal pulse combustor as well as other systems exhibiting similar behaviors.

2.4. Data acquisition

Dynamic pressure measurements were made using a piezoelectric pressure transducer (Kistler 206L). After analog band-pass filtering, the transducer output is collected using a PC with a National Instruments A/D card (AT-MIO-16X). The pressure tap is water-cooled to prevent overheating of the transducer. Since the tap acts as a second acoustic resonator, an 8.5 m coil of 0.635 cm diameter tubing was attached to reduce the tap's acoustic frequency sufficiently to prevent any significant perturbation of the combustion process.

The data consisted of a time series of voltage measurements at a very high sampling rate (20 kHz) relative to the dominant frequency (approximately 200 Hz). Data sampling began after the combustor had presumably reached thermal equilibrium, in order to avoid the transients. (However, see comments later in this section.) All of the following analysis is on a data set of 2^{20} data points, or portions thereof. A small portion of this data set is depicted in Figure 38.

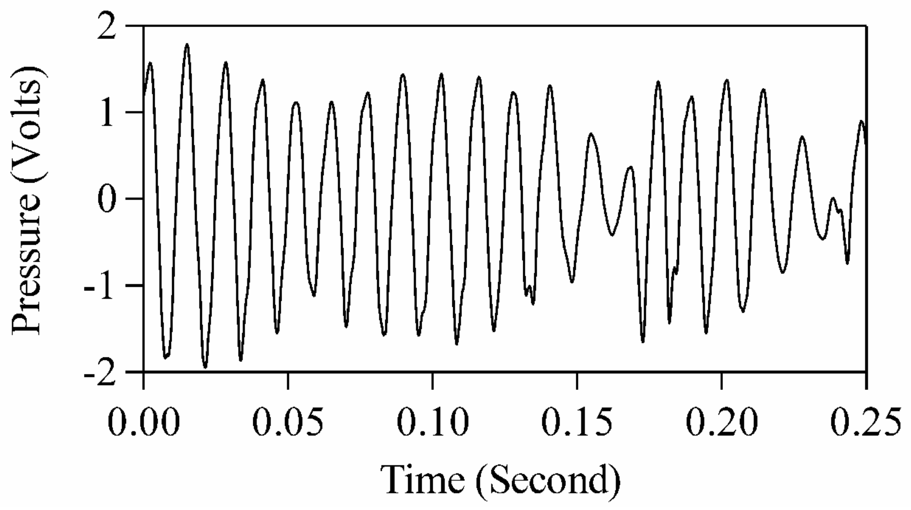


Figure 38. A plot of 2500 points from the data set.

2.5. Analysis of data

2.5.1. Introduction

Although the fluid dynamics of this engine and its operation is relatively simple, the dynamics of its motion are poorly understood. Traditional signal processing methods may fail here because the system dynamics are, at best, complicated, and, at worst, extremely noisy. Therefore it is our hope that analysis from a nonlinear dynamics perspective may yield more fruitful results.

2.5.2. Power spectral density

The starting point for most analysis of a given data set is to see what the frequency distribution might look like. If the dynamics is simple, then we might see only a few sharp spectral lines with some low noise floor that tell most of the story about the system's behavior. Unfortunately it is not the case here. The spectral density for this data set is very broad (see Figure 39). The power spectral density is taken on the full length of the data set of 1048576 (2^{20}) points using the Welch windowing with segment length of 8192 points. The graph shows a peak at about 80.56 Hertz, which corresponds to the main resonant frequency of the thermal pulse combustor. The subsequent peaks are the harmonics. In order to say something more quantitative about the dynamics, we will rely on subsequent analyses.

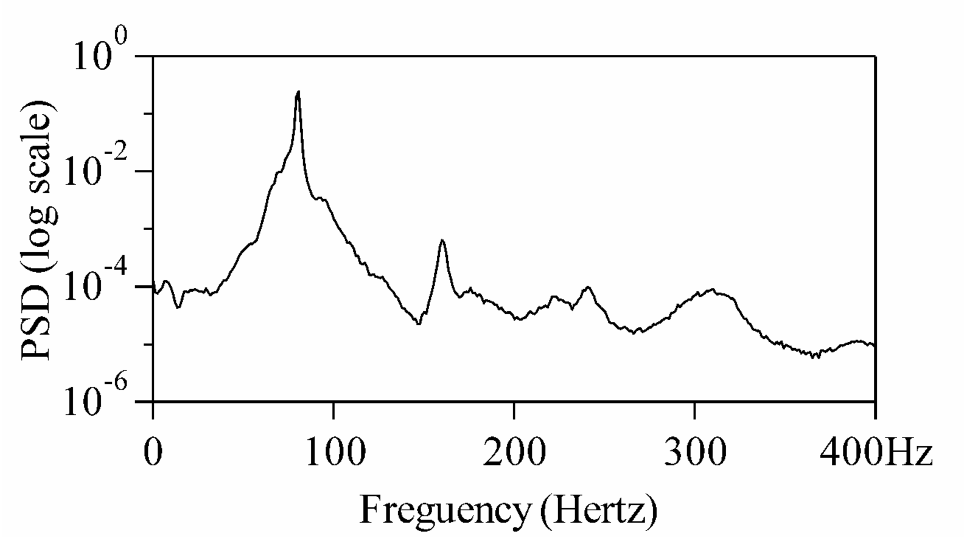


Figure 39. The power spectrum of the entire data set.

2.5.3. Nonstationarity

An important question to ask concerning complicated experimental data is whether or not the parameters controlling the dynamics remain stable and stationary throughout the data. Stability is guaranteed in a simulation, but in an experiment it is far from certain. Hence the question becomes whether or not parameter drift is significant.

A few simple tests were performed that would identify strong drifts in the data. The time series was divided into 128 pieces of equal length (8192 points each, covering about 80 cycles). From this it was possible to compare statistics (mean, standard deviation, skewness, kurtosis) on each piece of the data set. All of these results except the mean did not change significantly over time. However, this is not definitive. A nonlinear system could undergo a change in the dynamics that does not show up on any of these measurements. However, as can be seen from Figure 40, the mean value undergoes a fluctuating trend that does not appear to be random. This fluctuation is quite small in relation to the full extent of the data. The system may be in a long-term transient stage, since the temperature of the chamber is steadily decreasing because the combustor is running near a flameout regime where the fuel-to-air ratio is very low. In this regime, there is less heat generated from the combustion process. This causes a general decrease in the wall temperature that may not damp out in experimentally accessible times. Accordingly the data used in this analysis is limited to a portion where there is very little change in temperature over time. The fluctuation in the mean is small (~ 1%) and on the order of the noise in the system, so it was ignored.

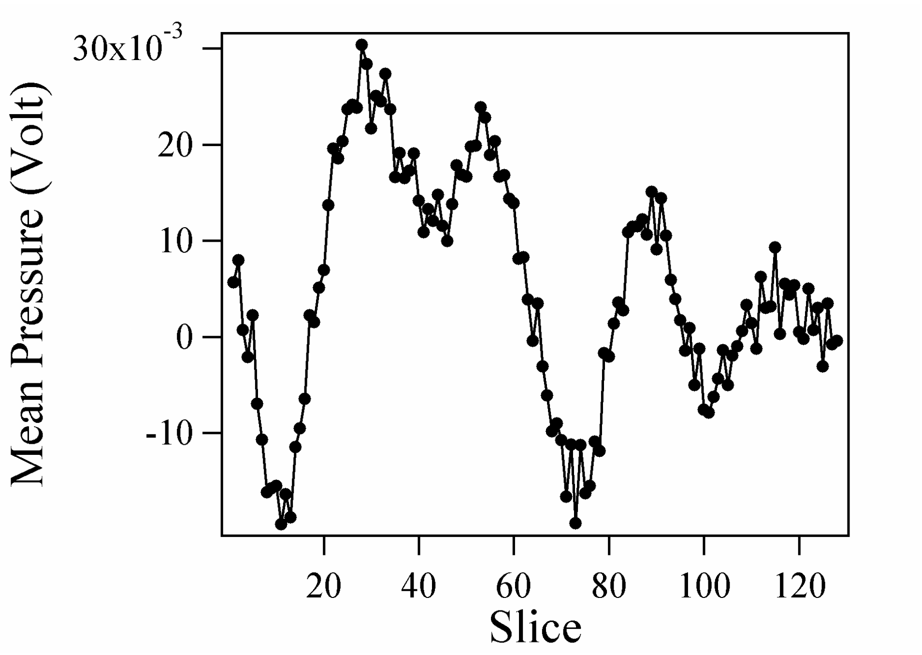


Figure 40. Nonstationary behavior of the data mean. Successive values of the average of windows of length 8192 are computed throughout the entire data set of size 1,048,576.

2.5.4. Delay coordinate embedding

The phase space was reconstructed by taking the time series $X_1, X_2, ...X_N$ and creating D-dimensional vectors using a time delay τ. With a time series of unknown nature, suitable values of the delay and embedding dimension may be determined both through other analysis methods or by visual inspection. A reasonable value for the delay may be suggested either by the first zero crossing of the autocorrelation function or by the first minimum of the mutual information function, as either value is plotted as a function of delay.

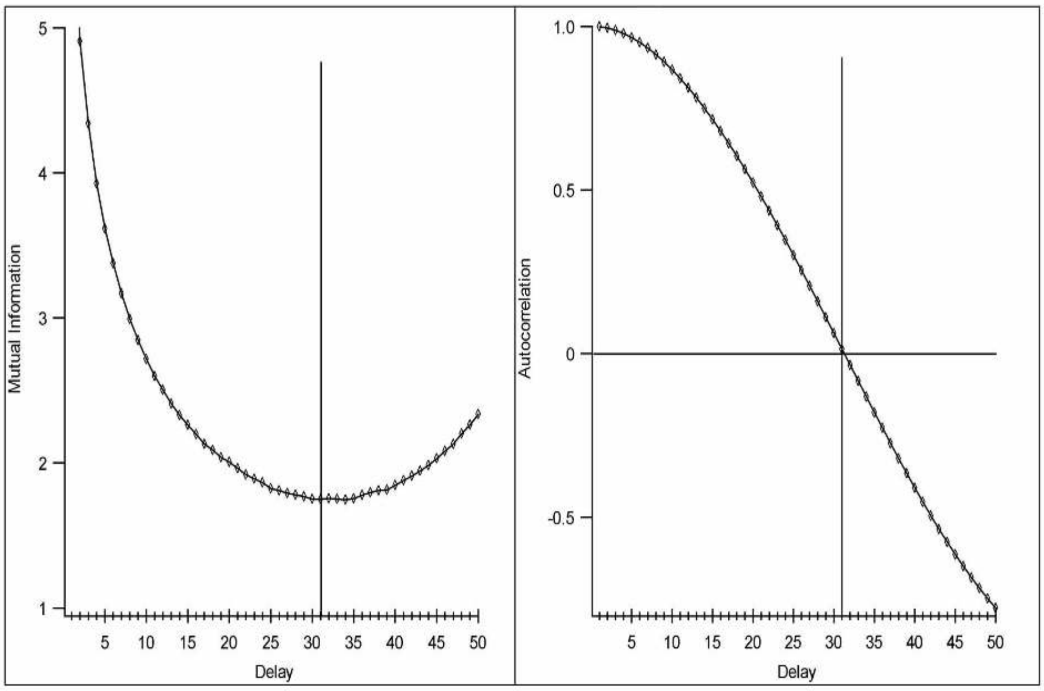


Figure 41. Two techniques for estimating the delay to use in a delay coordinate embedding of the combustion data. The first method is to choose the first minimum of the mutual information function. The second, less sophisticated method uses the first zero crossing of the autocorrelation function. Both methods suggest a delay of approximately 31.

However, for the combustion engine data, the mutual information function and the autocorrelation function were in perfect agreement. As shown in Figure 41 both values suggested a delay of 31. This was in agreement with visual inspection since 2 and 3 dimensional plots revealed the most structure near this value of delay (see Figure 42 and Figure 43). Structure is clearly evident in these plots. Unfortunately, they also reveal a complexity or noise dependence that makes the fine scale structure very difficult to detect.

Choice of embedding dimension is not as precise. Theoretically, any number greater than $2D$ is sufficient, where D is the box counting dimension of the object. .[66]

In practice however, one can often get away with a lot less than that. If too high an embedding dimension is used, then analysis becomes much more difficult and time consuming.

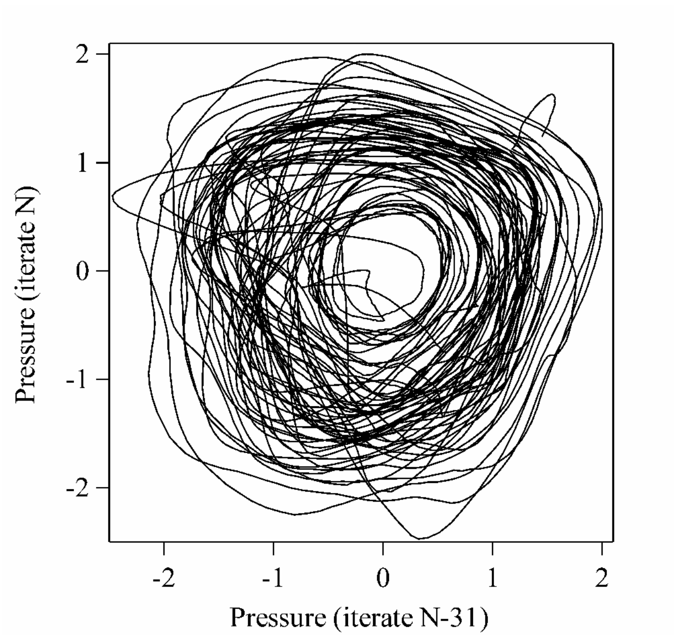


Figure 42. A two dimensional plot of 8000 points from the data set, with a delay of 31.

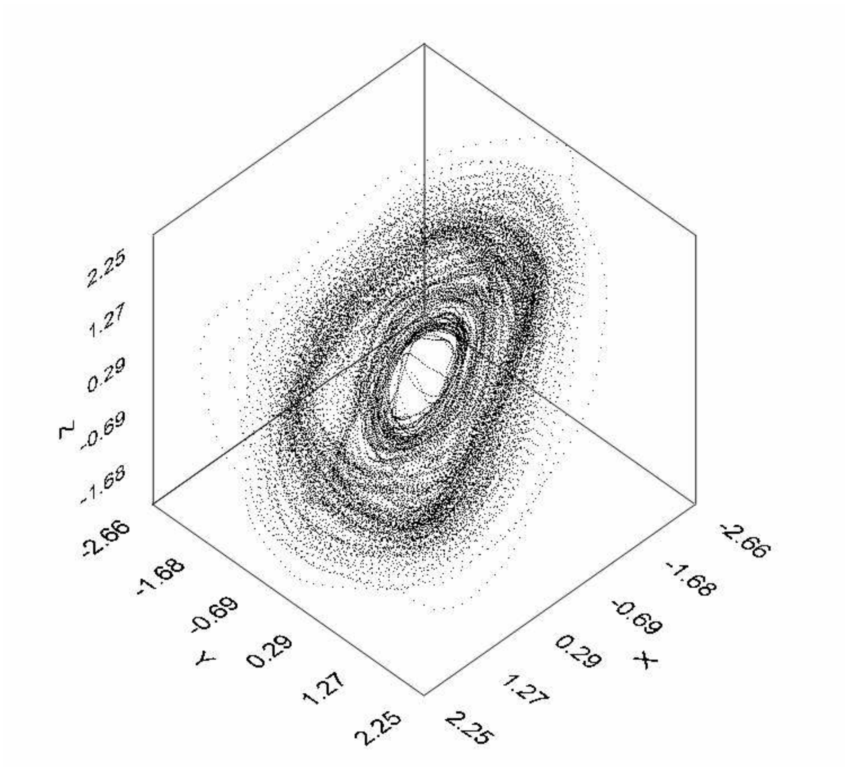


Figure 43. A three dimensional plot of 30000 points from the data set, with a delay of 31.

The method of false nearest neighbors[46] was chosen as the primary technique for determining the embedding dimension. The results agreed with what was suggested for application with real world data. As shown in Figure 44, the percentage of false neighbors dropped dramatically as the embedding dimension increases from 5 to 6. As will be shown later, this is in at least rough agreement with what was found to be a suitable embedding dimension for determination of Lyapunov exponents or fractal dimensions.

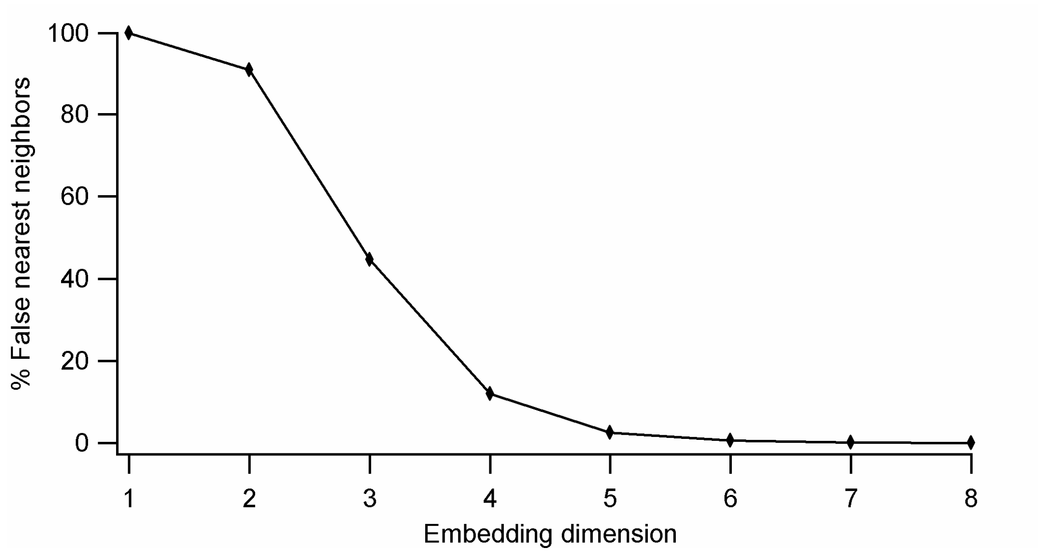


Figure 44 A plot of the results of the false nearest neighbors routine. An appropriate embedding dimension is found when the percentage of false near neighbors drops to a value near zero. This plot indicates that the embedding dimension should be at least 6.

2.5.5. Fractal dimension

The most common way to estimate the fractal dimension of a time series is using the Grassberger-Proccacia algorithm[43,93-95] to approximate the correlation dimension. One determines the correlation function, *C(ε)*, the probability that two arbitrary points on the orbit are closer together than a distance ε. In the limit $\varepsilon \to 0, N \to \infty$, the value $d\log C / d\log \varepsilon$ converges on the correlation dimension.

The correlation dimension is determined where a plateau is found in the slope of a plot of log($C(\varepsilon)$) versus log(ε). This method however, has several flaws. One is often concerned with other definitions of dimension, such as the information dimension. Although the correlation dimension and information dimension are similar and usually very close, this is the first of many approximations. Estimates of correlation dimension using the Grassberger-Proccacia algorithm are highly susceptible to noise and data set size. Each of these diminishes the region of the plateau. For higher dimensional data these problems are aggravated since minimal noise and exponentially more data are required to identify the plateau region. In addition, data with a high sample rate may exhibit strong correlations that skew the estimates. Having a large number of orbits compared to the sample rate may alleviate this problem. The approximations due to noise, data set size, nonstationarity and so on are inherent in the data set. But the Grassberger-Proccacia algorithm also uses an approximation to the definition of correlation dimension. Additional approximations, such as the use of the slope, are inherent in almost every dimension estimation routine. Finally, the Grassberger-Proccacia algorithm is essentially $O(N^2)$, where N is the number of points. Computation on a large data set can take days, even with the use of powerful computers. Comparing points against only a small sample of the data, k points, can speed it up $O(kN)$. But this is still technically an N-squared algorithm, and this additional approximation further compounds the problems.

Analysis was attempted on data sets of varying size and varying embedding dimension. Due to the high dimensionality and nonstationarity of the data, they rarely showed a clearly defined plateau, nor did they show clear convergence with large embeddings. This is not surprising, given the demands and limitations of the Grassberger-Proccacia algorithm. At best they indicated a fractal dimension of 4.28 ± 0.2 (Figure 45). Although this value agrees roughly with the choice of embedding dimension, more analysis was necessary to confirm the results. It was important to eliminate several sources of error in the computation of dimension. It was also necessary to compute many dimensions, such as the box counting, information, and correlation dimensions directly. Second, because of the strong dependence of dimension calculations on data set size, it was preferable that the entire data set be used in computation. This last criterion required that the method used must be fast- an $O(N^2)$ algorithm was out of the question.

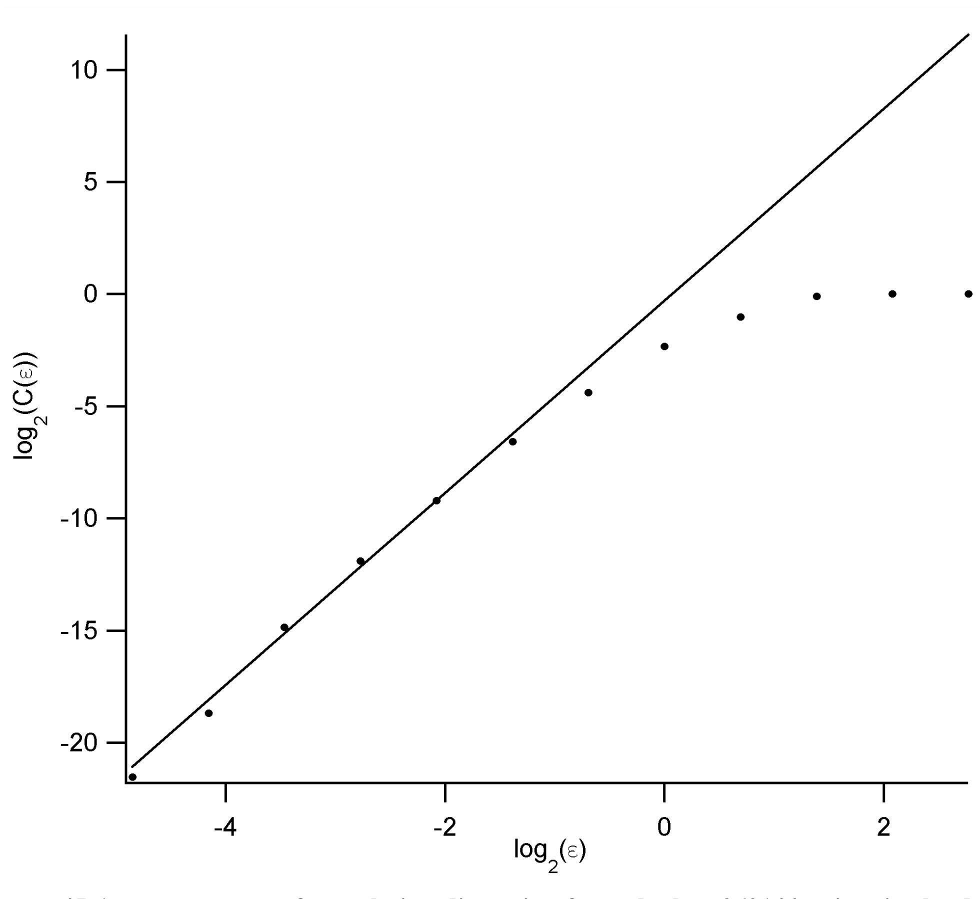


Figure 45 A measurement of correlation dimension from the last 262144 points in the data set. The correlation dimension is estimated from the slope of $\log_2(C(\varepsilon))$ vs. $\log_2(\varepsilon)$. The straight line plotted has slope 4.28.

For these reasons, a newly devised method to estimate an arbitrary number of generalized dimensions in $O(N \log N)$ time was used. A method of finding generalized entropies was described in Chapter 4. Once the generalized entropies have been determined, the generalized dimensions may be found by calculating the slope of a plot of $-H_q(\varepsilon)$ vs ln(ε).

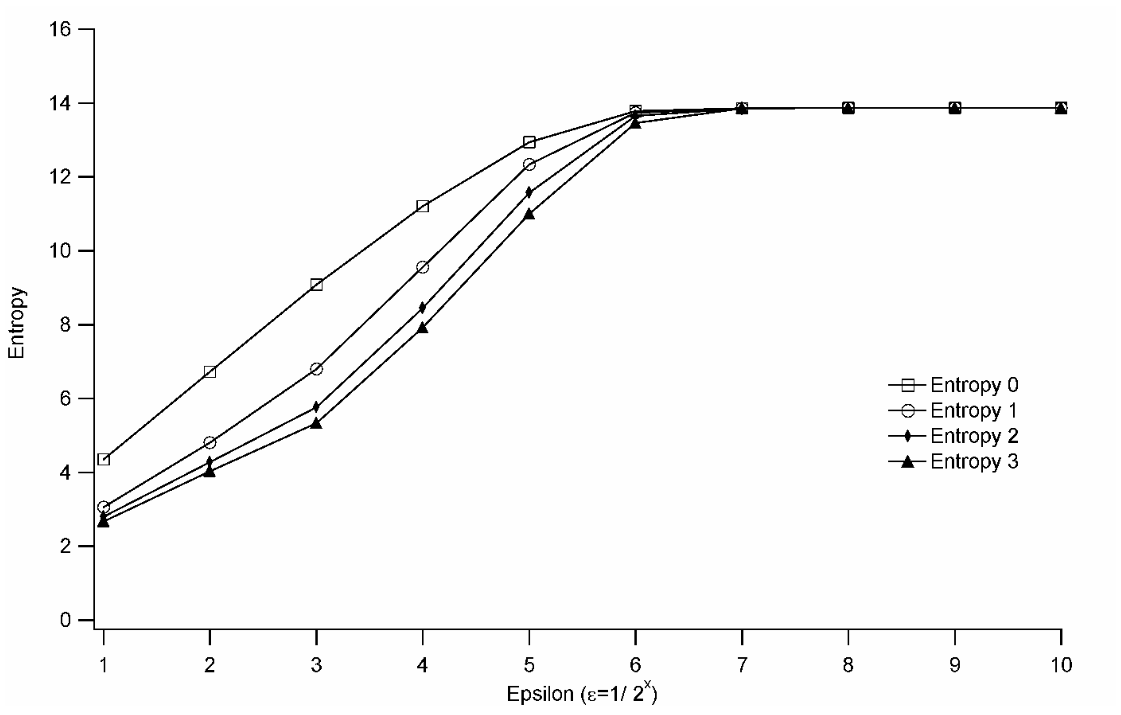


Figure 46. The first four generalized entropies. These correspond to generalized dimensions of $D(0) = 3.41 \pm 0.4$, $D(1) = 4.02 \pm 0.25$, $D(2) = 4.49 \pm 0.25$, $D(3) = 4.44 \pm 0.3$. Each of these was calculated using the entire data set embedded in 7 dimensions.

Figure 46 presents the results of our calculations performed on combustion data. Displayed are estimates of the first four generalized entropies for varying box size with an embedding dimension of 7. Additional tests were also performed for the embedding dimensions 3-6, and for the next four generalized entropies. The results indicated that, for $p > q$, $D(p) \le D(q)$, which agrees with theory. For large box size, the box counting dimension varies widely from the others. This is not surprising, since the box counting dimension is more susceptible to errors. It is also a poor quantity to use since it says nothing about the density of the attractor, only about its shape. However, the box counting dimension and all the others converge in the mid-region, before diverging slightly and then dropping to zero (due to data set size). It is this mid region that parallels the plateau region of the Grassberger-Proccacia algorithm. The estimates for fractal dimension ranged from 4.0 to 4.5, for all fractal dimensions calculated when embedding dimension was greater than 4. With the exception of the box counting dimension, this was true for all of the first eight generalized dimensions. The correlation dimension, for instance, was estimated at $D(2) = 4.49 \pm 0.25$, where the error was estimated based on the fluctuation of the slope of the entropy in the mid region. This agrees with our choice of 6 for the embedding dimension, and is in rough agreement with the result from the Grassberger-Proccacia algorithm. A fractal dimension of 4 to 4.5 indicates that an embedding dimension as high as 9 may be necessary, but as is often the case, a lower embedding dimension may be used.

2.5.6. Lyapunov exponents

Before determination of Lyapunov exponents was attempted, the data was embedded in 6 dimensions with a delay of 31, as suggested by the false nearest neighbors routine, the autocorrelation function and the mutual information function. Fractal dimension was estimated as between 4 and 4.5, so a local embedding dimension of 5 was chosen, yielding 5 exponents in the calculation of the full spectra.

As shall be seen, this is in agreement with the observation that the sum of the exponents must be negative.

Two methods of determining Lyapunov exponents were implemented; the method of Eckmann and Ruelle[96] for determining the Lyapunov spectra, and the method of Wolf, et al.,[77] for determining the largest exponent. Since these two methods are fundamentally different, one would not expect agreement between the estimates to be simply due to them both incorporating the same mistakes. The Wolf method involves following a trajectory in its path around the attractor. The rate of growth between points on this trajectory and a nearby trajectory is used to estimate the largest Lyapunov exponent. When the distance between these trajectories becomes too large, then a new nearby trajectory is found that is within a reasonable angular distance from the previous nearby trajectory. The Eckmann and Ruelle method involves using a small neighborhood of points and iterating them forward to estimate the local Jacobian, and then determining Lyapunov exponents from the eigenvalues of the Jacobians around the attractor.

The folding of the attractor brings diverging orbits back together. So any effects of nonlinearities will most likely serve to move all exponents closer to zero. Hence a slight underestimate of the positive exponents was expected.

One check on the Wolf algorithm was calculating the average angular displacement. This was typically less than 20%, well within reasonable bounds. Wolf showed that the angular displacement errors are not likely to accumulate, but each error may skew the largest positive exponent downwards.

Eckmann-Ruelle Method

Points in Fit	Iteration	λ_1	λ_2	λ_3	λ_4	λ_5
10	17	0.036963	0.018985	0.00268	-0.020285	-0.061305
10	18	0.034956	0.017727	0.001948	-0.018823	-0.062807
10	**19**	**0.032672**	**0.018569**	**0.000123**	**-0.01854**	**-0.057501**
10	20	0.031778	0.016857	0.000442	-0.018088	-0.054543
10	21	0.029999	0.015513	0.000411	-0.017223	-0.052437

Modified Wolf Algorithm

Number of Neighbors	Average angular displacement	λ_1
16	**0.770722**	**0.031167**

Table 4. Estimation of Lyapunov exponents. All calculations were performed with a 5 dimensional embedding and a delay of 31. An iteration step size of 19 represented the best estimate of the Lyapunov spectra (due to the high accuracy of the zero exponent). An estimate of the dominant exponent from the Wolf method is included to show that there is agreement between the two methods.

In Table 4, results of exponent calculations are provided. All calculations were performed with a time delay of 31, global embedding dimension (used to unfold the attractor when identifying nearby points) of 6 and a local embedding dimension (used in exponent calculations) of 5. The exponents are given in units of 1/time, where the time scale is defined so that the time between samples is 1. Many more calculations were performed until a reasonable and stable parameter regime was found for both methods. Note that the zero exponent snaps into place for an appropriate choice of the iteration step size.

Several of our criteria are determined immediately upon inspection. The zero exponent was identified with a high degree of accuracy. The sum of the exponents is negative, while the sum of the first four is positive. This indicates that a fractal dimension between 4 and 5 was a reasonable estimate. These exponents also give a Lyapunov dimension of 4.571. Although there is large error in both Lyapunov dimension and information dimension estimates, this value is within acceptable limits. Results from the Wolf algorithm are included in order to show that the two methods provide rough agreement. The Wolf method by itself is not sufficient since there are few checks on its validity.

However, it was not possible to confirm all criteria. Measurement of the metric entropy is still ongoing work. Sectioning the data introduced additional noise and measurement of exponents from the section was even more uncertain. Thus it was not possible to get agreement between exponent estimates from the section and from the flow, nor was it expected. Time reversal results were also inconclusive at best. However, simulated data with the addition of noise would not usually switch the signs of the exponents under time reversal either. So the sign change of exponents when the data is reversed may not be a suitable criterion for noisy data.

2.5.7. Comparison with other work

To our knowledge, there has been one previous publication attempting to analyze the nonlinear dynamics of a thermal pulse combustion engine.[90] Some differences in approach are clear. Much of their work centered around comparisons of results between theory (a simulated 4th order ODE model of the combustor) and experiment. They also attempted analysis of many small data sets, whereas the analysis contained herein concentrated on one large data set. Some technical differences are also quite clear. Although they calculated the mutual information function, they chose to embed the system using the singular value decomposition method (SVD) of Broomhead and King.[68] In this section, the authors chose not to use SVD because of the interpretation difficulties described by Fraser, et al. (Ref. 67 and references therein). However, the primary difference is their claim of low dimensionality in the system. This was not seen in any of the methods used here- false nearest neighbors, calculation of the generalized dimensions, or calculation of the Lyapunov spectrum. Some of these differences may be explained by the use of parameters in the data acquisition. The authors of this work also believe that more stringent criteria was necessary in Daw, et al,[90] to verify chaos in the combustion engine. In light of this and the previous work, it is clear that more analysis should be done so that a firm conclusion may be reached and the differences between the two approaches may be rectified.

2.5.8. Conclusion

The data from the combustion engine appears to represent a five dimensional system with two positive Lyapunov exponents. This system may therefore be considered high dimensional and hyperchaotic. As such, computation of fractal dimension and of Lyapunov exponents are difficult tasks. Some consistency was achieved between results using various methods of analysis. However, precise quantitative results were not possible. This was not a surprise since each of these methods of analysis was susceptible to noise, drift and data set size, and the errors are more problematic with high dimensional time series. The analysis herein demonstrated the limitations of many methods for time series analysis. This work also showed how multiple analyses can be used to confirm each others results.

3. The Magnetoelastic Ribbon

3.1. Overview

The magnetoelastic ribbon[97-99] has rich nonlinear dynamical behavior ranging from periodic to chaotic and everything in between, but it is simple enough to be described by a low dimensional model.[100] It was the first experimental system to demonstrate chaos control.[99] Also observed in this system were quasiperiodic and strange non-chaotic behavior[101], crises[102], scaling characteristics[103] and stochastic resonance[104]. The experimental apparatus is presented in Figure 47.

$$\frac{dY_f}{dt} = \frac{\left(Y_{f,i} - Y_f\right)}{\tau_f}\frac{\tilde{T}}{\tilde{P}} - \left(\frac{C_p T_0}{\tau_c \Delta H}\right)\frac{\tilde{T}}{\tilde{P}} \qquad (6)$$

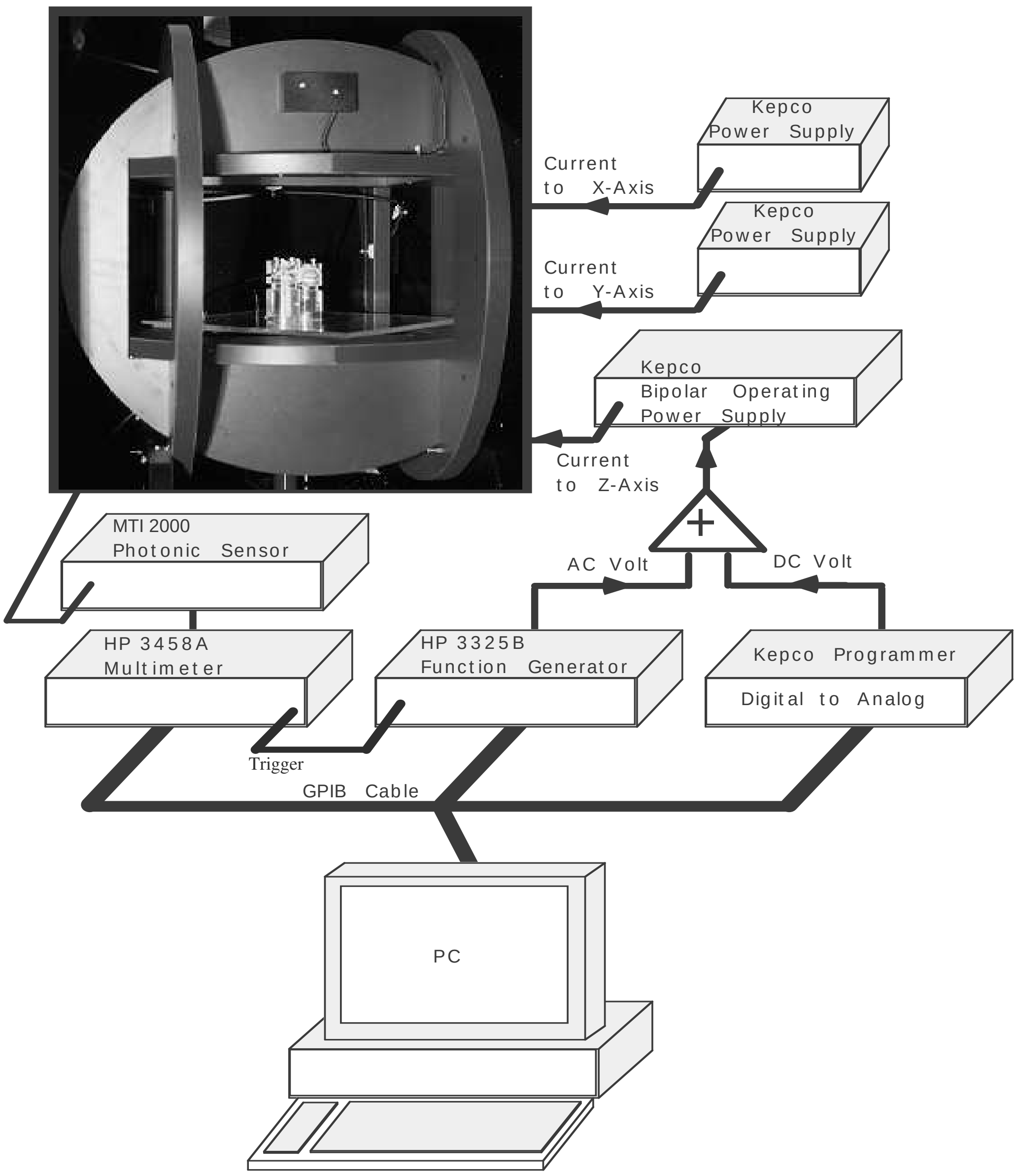


Figure 47 Instrumental layout for the magnetoelastic ribbon setup.

The magnetoelastic ribbon was clamped at the base in a Plexiglas holder such that the free vertical length was greater than the Euler buckling length. This gave the ribbon an initial buckled configuration (Figure 48). The holder and the ribbon were placed within three mutually orthogonal pairs of Helmholtz coils. The coils were needed to compensate for the ambient field and to apply a nearly uniform vertical magnetic field along the ribbon's length, as illustrated in Ditto *et al.* [105]. In this system, a magnetic field component H_{dc} produced from the dc current was applied to the ribbon in order to center the operating region at a point where the ribbon is soft. A magnetic field component H_{ac} produced from the ac current was then applied to modulate the system around H_{dc}. The alternating magnetic field H_{ac} made the ribbon go from soft to stiff and back to soft again stiff again and finally soft again at a driving frequency *f.* At a large enough H_{ac} magnitude, the ribbon oscillated chaotically. Any one of the parameters H_{dc}, H_{ac} or frequency *f* could be varied.

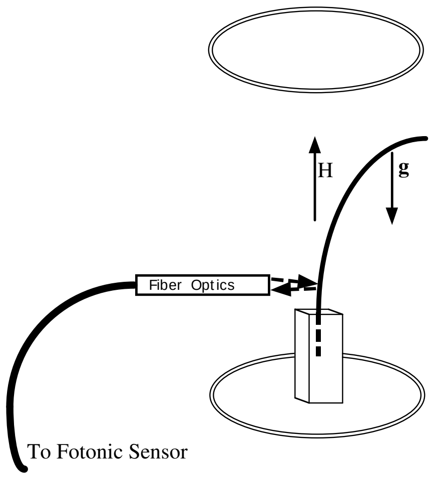


Figure 48 The ribbon setup inside the Helmholtz coils. H is the applied magnetic field and g is the gravitational pull on the ribbon. This measurement provides sufficient information to completely describe the state of motion of the ribbon unless the frequency is high enough to cause sections of the ribbon to move out of phase with each other. Such traveling waves traversing the ribbon is termed "shimmering."

3.2. Experimental Setup

3.2.1. Ribbon

At the core of the setup was the magnetoelastic ribbon (Metglas 2605SC™), an amorphous magnetic material ($Fe_{81}B_{13.5}Si_{3.5}C_2$) approximately 100 mm long by 3 mm wide by 0.025 mm thick. The material resembled stiff Christmas tinsel. When a weak field was applied, the Young's modulus and the stiffness of the ribbon decreased by an order of magnitude, causing it to buckle under its own weight. The characteristic response of the ribbon's Young's modulus to the applied magnetic field is shown in Figure 49. The Young's modulus of the ribbon decreased dramatically with a small change of the magnetic field, which changed the ribbon from stiff to soft very quickly. As the field was increased the ribbon's Young's modulus became level between 0.7 and 1.0 Oe. The typical operating point for the experiments was in this region. Any further increase in the field stiffened the ribbon quickly, as indicated by a sharp rise in the curve. Overall the Young's modulus of the ribbon could change by over a factor of 10 in response to a change of magnetic field.

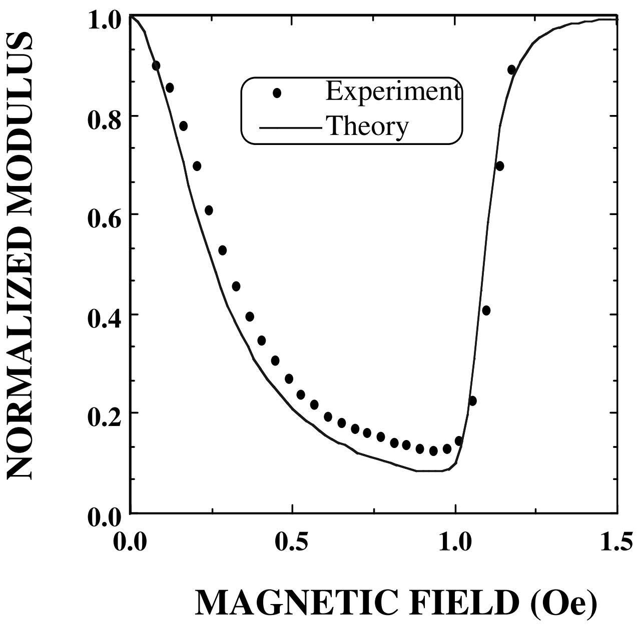


Figure 49 Normalized Young's modulus of the magnetoelastic ribbon as a function of applied magnetic field. The modulus was normalized to zero field modulus. The solid line was the theoretical prediction and the dots were the measured result. (Data courtesy of Mark L. Spano).

3.2.2. Coils

The setup consisted of three sets of Helmholtz coils that were used to generate two orthogonal horizontal fields and a vertical field. The design of the coils was based on the theory that a pair of coils, with optimal inter-coil spacing, produced the most uniform field possible for that size coils. The two most important parameters were the desired area of uniformity and the magnitude of magnetic field required. The size of the coils was determined by the required area of the uniform magnetic field, a cylinder 2 inches in diameter by 3 inches in length along the z-axis (vertical axis). Table 5 gives the complete specifications for the coils.

Specification	X-Axis	Y-Axis	Z-Axis

Area of Uniformity (cylinder)	2" x 3" (Diameter x Length) Cylinder along z-axis		
Uniformity	<0.05 %	<0.05 %	<0.05 %
Overall Outer Diam.	30.75"	27.0625"	23.25"
Nominal Intercoil Spacing (face to face)	13.5"	11.875"	9.875"
Reisistance (per coil pair at 70 F)	16.7 Ohms	13.1 Ohms	3.0 Ohms
Rated Operating Current	1.0 Amps	1.0 Amps	4.0 Amps
Central Field at Rated Operating Current	5.0 Oe	5.0 Oe	15.2 Oe
AC Operation (Yes/No)	No	No	Yes
Nominal Inductance			38 mH
Maximum Frequency			100 Hz
Rated Operating Current			2.83 A (rms)
Voltage at Rated Operating Current & Frequency			68.0 V (rms)

Table 5. A listing of the specifications of the Helmholtz coils

3.2.3. Power supplies

Kepco power supplies (model CC 21-1M) were connected to each pair of coils that produced the horizontal fields. The coils were used to counteract ambient fields in the room. The coils that produced the vertical field were powered by a Kepco Bipolar Operating Power Supply (BOP 100-2M). The bipolar operating power supply proportionally converted any input voltage into current. The voltage supplies came from a function generator (Hewlett-Packard 3325B) which supplied the ac component of the voltage, and from a dc voltage power source (Kepco voltage programmer or digital-to-analog (D-A) converter). These were combined in a voltage adder box before going into the bipolar operating power supply.

3.2.4. Holder

A Plexiglas holder was set at the center of the coils (Figure 47). The holder consisted of two vertical cylinder-like structures. One structure, which was placed at the center of the platform, was used to hold the magnetoelastic ribbon. The other, which was placed off-center, was used to hold the fiber optics of the fotonic sensor. The ribbon holder and the fotonic sensor holder were capable of rotating horizontal rotation, tilting and variable height. The base heights (the distance measured from the Plexiglas platform to the top of the holder) were set at the level of the bottom of the uniform vertical magnetic field cylinder.

3.2.5. Sensor

The fotonic sensor (MTI 2000 Fotonic Sensor) was used to measure the position of the ribbon. The instrument consisted of a tungsten light source with constant intensity. Light was sent through a set of randomly arranged fiber optics that had half of its bundles used for emitting light and the other half used to detect light. Light went through the fiber optics and exited at the other end, about half a meter from the tungsten light source. The light that reflected off the magnetoelastic ribbon entered the detecting fiber bundles, which were arranged randomly among the emitting fibers. At the other end of the bundle was a photodetector that would determine the intensity of the reflected light. From this intensity, the relative position of the ribbon was determined using the relationship between the outgoing beam and the reflected beam. This instrument measured the position with high accuracy and minimal noise. A calibration was made to account for the placement distance between the ribbon and the end of the fiber optics as well as for the reflectivity of the surface. The intensity of the source and the loss in the fiber bundles had been determined and calibrated at the manufacturing site. The output of the fotonic sensor was given in voltages and measured by the sensor with a high resolution multimeter (Hewlett-Packard 3458A). This instrument could read up to 28-bit resolution from the fotonic sensor output. Since the experiments were designed to take stroboscopic data, the multimeter was triggered by a synchronizing signal sent from the ac function generator. This allowed the multimeter to read the signal once per driving period of the magnetic field.

3.2.6. Computer

A desktop computer (Dell 466ME) was connected to the multimeter, the function generator, and the Kepco voltage programmer through a general purpose interface board (GPIB). A custom program was written to communicate with each instrument. Through this program the amplitude and frequency of the ac voltage could be set on the function generator, and the dc voltage could be set on the Kepco voltage programmer. The program could also communicate with the multimeter to acquire data.

3.2.7. Calibration of the Helmholtz coils

The Helmholtz coils were calibrated to counteract the ambient fields at the experimental site. Using a high precision Gauss meter, the ambient field along one of the horizontal axes of a pair of coils was determined. Current was applied to the corresponding set of coils until the produced field canceled the measured ambient field. Next the probe was positioned along the axis of another set of coils. The ambient field in this direction was determined. Current was applied to the appropriate set of coils until the fields canceled each other. Along one horizontal axis (x-axis in the experimental setup) a current of 41.0±1.0 mA was given to a pair of coils. The other pair of coils (y-axis) was given a current of 12.0±1.0 mA.

The vertical field was used to drive the ribbon. Any ambient field in this direction would simply change the zero of the dc field. To determine the relationship between the requested voltage from the computer and the vertical magnetic field produced in the Helmholtz coils, a series of dc voltage outputs were requested through the Kepco programmer. The corresponding magnitude of the field produced was then recorded for each voltage output. The field magnitude produced by the coils was linearly dependent on the voltage programmer outputs.

3.2.8. Vibration isolation and temperature control

The magnetoelastic ribbon was very sensitive to temperature changes and vibrations. To minimize this, the Helmholtz coils and the ribbon were placed on a vibration isolation table (model TMC Micro-g) that was 3.5x6 feet. The table was constructed from nonmagnetic stainless steel to prevent any interference with the magnetic field produced by the Helmholtz coils. In addition, the table was tuned to isolate low frequency vibrations ranging from 3 - 21 Hertz. Active vibration control was avoided for fear of small control perturbations affecting the experiment. The table was placed snugly inside a temperature controlled box. In order to minimize distortion due to magnetic fields, no metal was used in the construction. The box had the dimensions of 4x7x6.5 feet (W x L x H). One inch thick Styrofoam was used to insulate all six sides of the box and the box was constructed using only wood and glue. To control the temperature inside the box, air was pumped at a rate of 53 cubic feet per minute through a temperature control unit (Omega CN9000A) then through 4 big heating resistors before passing back into the box. Since this unit could only add heat to the box, it alone could not maintain the temperature.

The entire setup, except the computer and temperature control unit, was placed inside a self-contained air conditioned room. The temperature of the room was set at 70^{o} F and the temperature of the box at 87^{o} F. Thus, the low temperature of the room brought the temperature inside the box down and the temperature control unit added heat to counterbalance the room effect. The temperature inside the box was maintained to within $\pm 0.5^{o}$ F. In addition, a sealed Plexiglas rectangular cover was placed over the ribbon and fiber optic holder inside the Helmholtz coils. The temperature at the ribbon fluctuated to within $\pm 0.1^{o}$ F.

3.3. Data Analysis

3.3.1. Acquired Data

A wide variety of data sets were acquired from this system. This was primarily because before analysis was begun, there was little knowledge of the parameter settings to use in order to observe a particular dynamics. This was also done to gain familiarity with the experimental system and to gain an appreciation for the subtleties of temperature, vibration and humidity dependence in the system. Thus numerous bifurcation plots of the data, for ranges of applied dc and ac currents were collected. Figure 50 depicts a typical bifurcation diagram of this system. It is from this bifurcation plot that suitable parameters were chosen for a 100,000 point strobed data set to analyse. The applied dc current was set to 2212.454212 mV, with the ac current set at 3200mV, 0.95 Hz. The first fifty data points (transient stage) were ignored and the total data acquisition time lasted approximately 29 hours and 15 minutes. All of the following analysis is performed on this 100,000 point data set.

A delay coordinate embedding plot of the data set is depicted in Figure 51. A one dimensional wrapped string-like structure (similar to the Henon map) is evident. However, the folding may be an artifact of nonstationarity in the data. That is, parameter drift during the course of data acquisition may cause the data values to shift slightly.

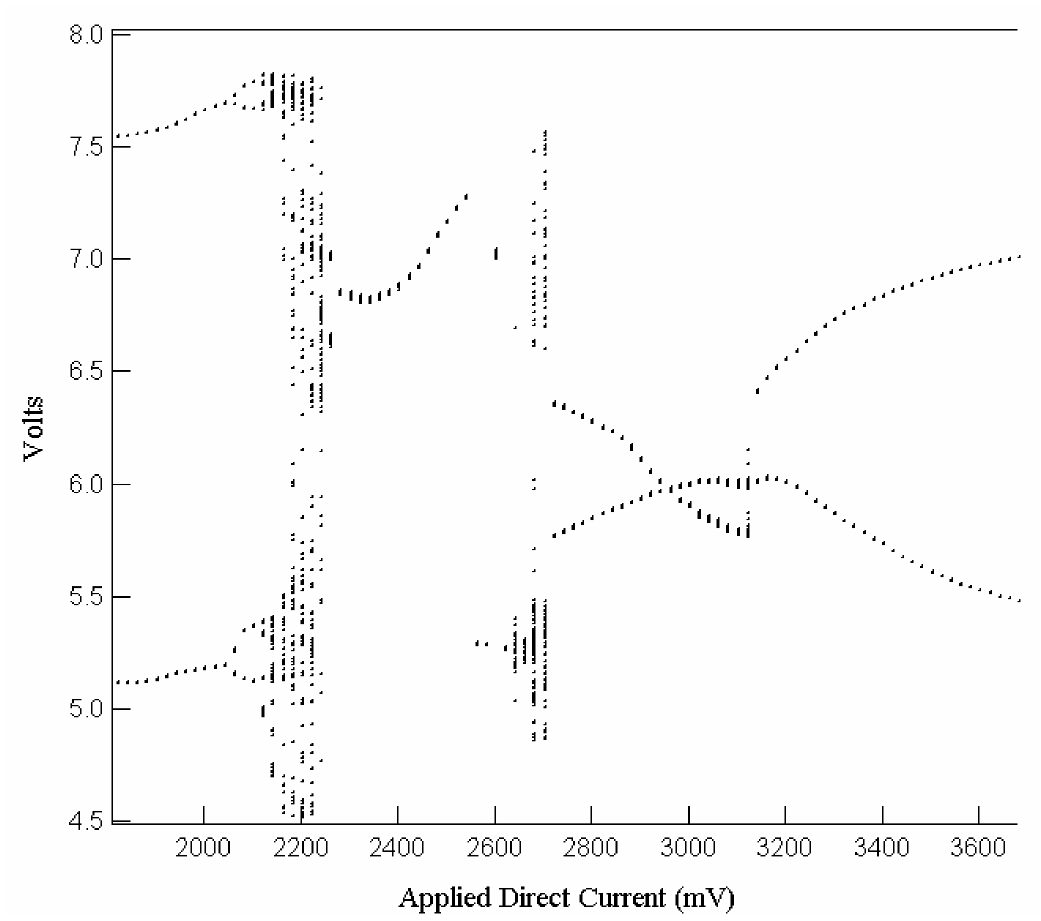


Figure 50. A bifurcation plot of strobed data from the magnetoelastic ribbon experiment. Data was taken with applied DC voltage ranging from 1800 to 3680 mV in intervals of 20 mV. Transients were ignored and the next 100 data points were acquired at each interval The ac current was set at 3200mV, 0.95 Hz.

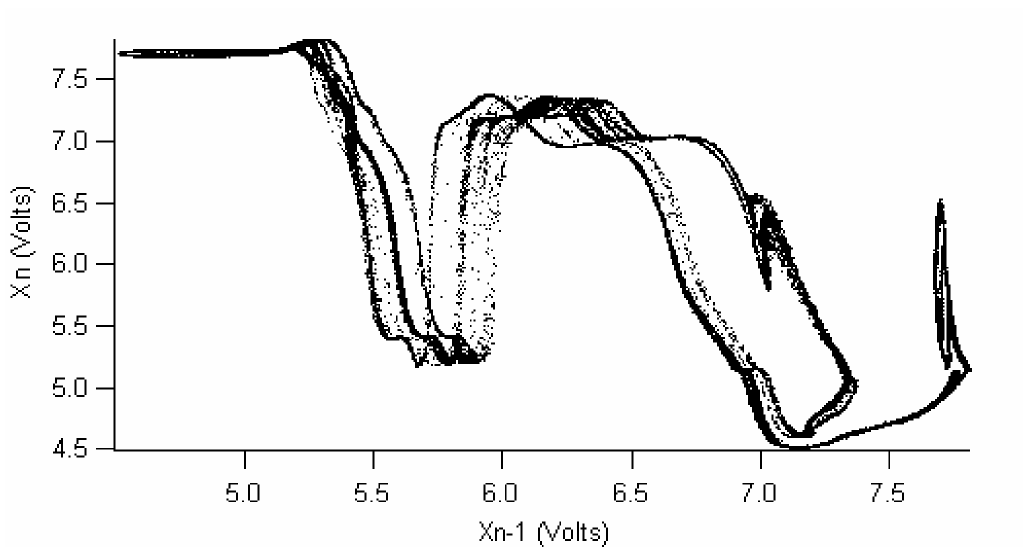


Figure 51. Time one return map of the magnetoelastic ribbon data.

3.3.2. Nonstationarity

A window of length 8192 was applied to the time series From this it was possible to compare how the windowed mean and the windowed standard deviation varied as functions of time. As can be seen from Figure 52, the mean value undergoes a fluctuating trend that does not appear to be random. This fluctuation is quite small in relation to the full extent of the data, 3.2Volts. There are several possible causes of the nonstationarity. First, the system may be in a long term transient stage. However, if this were so, then one would expect the mean value and the standard deviation to be converging to constant values. To a small degree, this is evident in Figure 53, which plots the standard deviation of these windows. However, the convergence is not sufficient to warrant removal of a portion of the data in order to analyze a stationary system. Thus we conclude that long term dynamics are evident in the data.

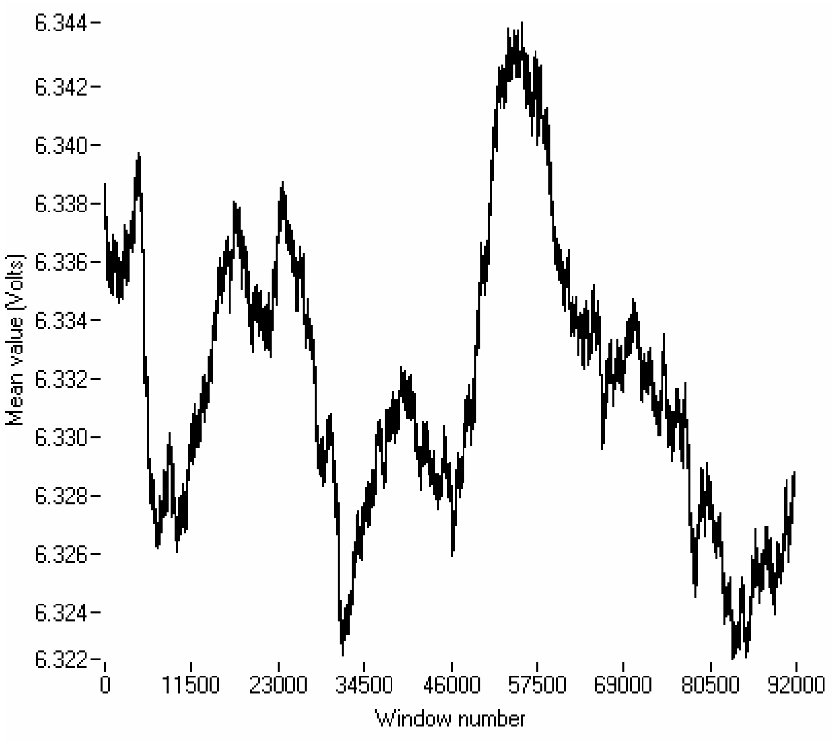


Figure 52. Nonstationarity of the mean in magnetoelastic ribbon data. Overlapping windows of length 8192 were applied to the data and the mean of each window is plotted.

Ideally, a much longer data set should be acquired so that the long term dynamics can be properly taken into account. But a longer data set would also introduce drift due to temperature changes, so this was not considered. Because the variation in the window mean (at most 22mV, or 0.6% of the range of the data) and the variation in the window standard deviation (at most 35mV or 3.0% the standard deviation of the entire data set) are relatively small, and because these appear to be part of the dynamics of the system, as opposed to part of a drift in the parameters, it was assumed that nonlinear analysis was still possible. However, this nonstationarity does imply that one should be circumspect concerning the exact values of the quantitative results obtained on this data.

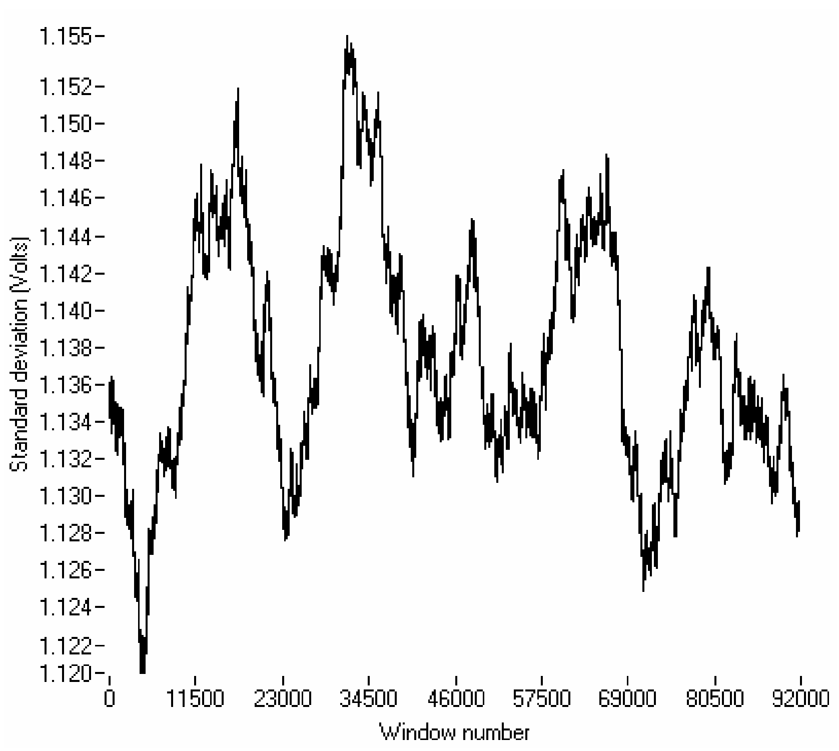


Figure 53. Nonstationarity of the standard deviation in magnetoelastic ribbon data. Overlapping windows of length 8192 were applied to the data and the standard deviation of each window was plotted.

3.3.3. Fractal dimension

Fractal dimension was estimated using both the Grassberger-Proccacia algorithm to approximate the correlation dimension and the method of finding generalized entropies method described in Chapter 4. The correlation dimension is determined where a plateau is found in the slope of a plot of $\log(C(\varepsilon))$ versus $\log(\varepsilon)$, where C(ε) is the correlation function and ε is a distance (normalized to one for the maximal distance between two points in the data). This plot is depicted in Figure 54. Clearly, estimation of the slope is difficult because the slope is nonconstant, regardless of the embedding dimension chosen. This is in part because 100,000 data points is not sufficient to get a reasonable estimate of dimensionality for an embedding dimension of 4 or 5, and also because nonstationarity may appear as a fractal structure. For small box size ε, the distribution appears to be discrete points, and the correlation dimension estimate thus approaches zero. But a relatively small box size is necessary to identify the fine scale structure that may exist.

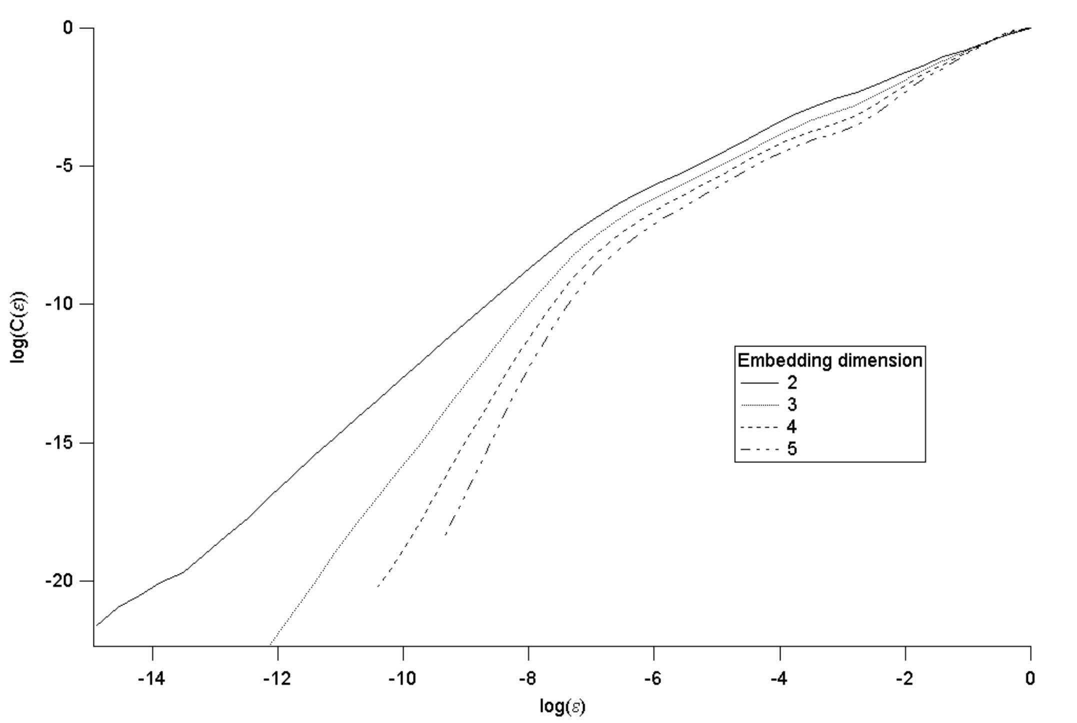


Figure 54. Correlation dimension from the magnetoelastic ribbon data. The correlation dimension is estimated from the slope of log(C(ε)) vs. log(ε). This function was estimated for the data embedded in 2, 3, 4, and 5 dimensions.

Thus we can not trust estimates of the slope for log(ε)<-8, since this slope is due to the finite number of data points. Luckily the slopes do assume a relatively constant value for –4<log(ε)<-7. Estimates of the slope in this region are approximately 1.055, 1.129, 1.310 and 1.351 for embedding dimensions of 2, 3, 4 and 5, respectively. This agrees with the visual inspection that suggests the structure is similar to the Henon map (correlation dimension 1.2). One implication of this low fractal dimension is that an embedding dimension of three ($> 2 \cdot 1.351$) should be sufficient to capture the dynamics of the system. Therefore, further analysis is performed using a three dimensional embedding.

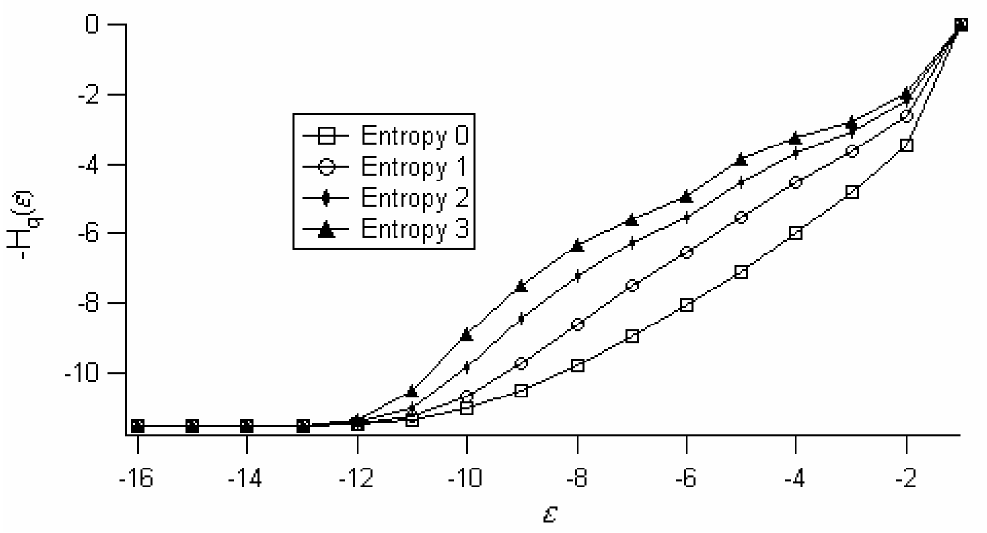


Figure 55. Plot of -$H_q(\varepsilon)$ vs ln(ε) for the magnetoelastic ribbon data embedded in 3 dimensions. These correspond to generalized dimensions of 1.33, 1.40, 1.24 and 1.16 for the zeroth, first, second and third order fractal dimensions, respectively.

Figure 55 depicts the first four generalized dimensions of the data. Displayed are estimates of the first four generalized entropies for varying box size with an embedding dimension of 3. With the exception of the box counting dimension $D(0)$, which is difficult to estimate, the results indicated that, for $p > q$, $D(p) \le D(q)$. This agrees with theoretical arguments. In addition, for a three dimensional embedding, the correlation dimension estimated using the Grassberger-Proccacia algorithm and estimated here ($D(2)$), differ by only 5%. Therefore, we may assume that this is a fairly reliable estimate.

3.3.4. Lyapunov exponents

The dominant Lyapunov exponent was estimated using the Wolf, et al.,[77] method. This is depicted in Table 6. Increasing the iteration step may sometimes underestimate the dominant Lyapunov exponent because the distance between points may approach the size of the attractor, and thus the rate of divergence is constrained by that size. Also increasing the number of near neighbors used may underestimate the value because this allows a larger distance between neighbors. For a significantly chaotic system ($\lambda_1 >> 0$) this outweighs the accuracy improvement through aligning the vectors along the direction of the dominant exponent. Furthermore, prior results in estimation of fractal dimension suggested that an embedding dimension of three should be used. Thus our best estimate of the dominant exponent is given by the bold faced entry in Table 6, $\lambda_1=0.898$.

The Lyapunov spectrum was then estimated using the Jacobian based method of Eckmann and Ruelle.[96] The data was embedded in two dimensions (as opposed to three), because higher dimensional embeddings force near neighbors to be close in time due to the finite number of data points. Ten neighbors were used in order to gain an accurate fit around each point, although the large number of neighbors and hence large distances, guarantees some underestimation of all exponents. Finally, an iteration step of two was used as a compromise between the effects of noise for low step size and the underestimation of divergence for large step size.

Under these conditions, the Eckmann-Ruelle algorithm provides an estimate of the Lyapunov spectrum as λ_1=0.2137 and λ_2=-0.7223. Because of all the inherent difficulties with this method, we expect these values to be underestimates and hence not in complete agreement with the estimate of λ_1 from the Wolf method. However, they do agree within an order of magnitude, the sum of the exponents is negative and the Lyapunov dimension may be estimated as $D_\lambda = 1 + \frac{1}{|-0.7223|}0.2137 = 1.30$, which agrees with the estimate of the information dimension from Figure 55, $D_1 = 1.40$. Thus we have a rough confirmation of the Kaplan-Yorke conjecture and further validation of the numerical analysis.

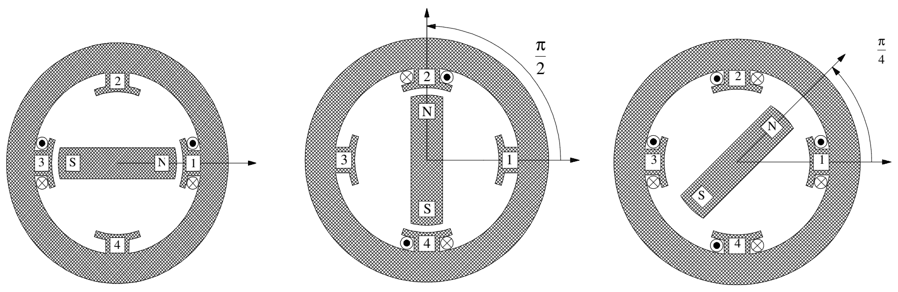
Embedding dimension	Number of near neighbors used	Iteration step	λ_1
2	10	1	1.55398
2	20	1	1.344258
2	10	2	1.239887
2	20	2	1.098532
3	**10**	**1**	**0.898159**
3	20	1	0.827021
3	10	2	0.820322
3	20	2	0.763024
4	10	1	0.680146
4	20	1	0.664335
4	10	2	0.673306
4	20	2	0.650527

Table 6. Estimates of the largest Lyapunov exponent using the divergence based Wolf method. The bold faced row indicates the best estimate given knowledge of the correct embedding dimension and knowledge of the imperfections of this algorithm.

4. The Electric Step Motor

4.1. Overview

The electric step motor is a type of motor that provides incremental motion, or steps, in response to pulses of current that alternately change the polarity of the stator poles. The main advantage of an electric step motor is its open-loop operation. That is the position control can be achieved without shaft position feedback. The shaft can be stopped in any position with a high degree of accuracy, thus producing incremental displacements. It is used in numerous applications such as printers, hard disks, toys and robots.



a) $I_\alpha = I_n$, $I_\beta = 0$ b) $I_\alpha = 0$, $I_\beta = I_n$ c) $I_\alpha = I_n$, $I_\beta = I_n$

Figure 56 Principle of the step motor

As depicted in Figure 56, the stator has windings, 1 and 3 in series fed by the voltage U_α, and 2 and 4 in series fed by the voltage U_β. I_α is the current in the windings 1 and 3 and I_β is the current in the windings 2 and 4. The rotor has permanent magnets. Torque is developed by the tendency of the rotor and stator magnetic fields to pull into alignment according to the sequential feeding of the phases. If phase α (windings 1-3) is fed, stator induction is horizontal and the rotor is also horizontal (Figure 56, part a). If phase β (windings 2-4) is fed, stator induction is vertical and the rotor turns one step (Figure 56, part b). If the two phases are fed simultaneously, induction produced by the stator has an intermediate position, the rotor turns a half step. Phases are switched alternately. Lets consider the following cycle:

1 ($I_\alpha = In, I_\beta = -In$), 2 ($I_\alpha = In, I_\beta = In$),

3 ($I_\alpha = -In, I_\beta = In$), 4 ($I_\alpha = -In, I_\beta = -In$)

The rotor has four stable positions during the switch cycle, which are -π/4, π/4, 3π/4, and 5π/4. This is the supply mode that is most frequently used and is called mode 2.

The torque has two origins. First, teeth on the stator and on the rotor create a variable resistance. Second, the magnetization of the rotor creates an interaction between the rotor magnets and the stator currents. According to the physical phenomenon responsible for the torque, motors can be classified as variable reluctance motors, permanent magnet motors, or hybrid motors. Variable resistance motors have teeth on the stator and on the rotor, but no permanent magnet on the rotor. In this case, the torque is due to the variable resistance. Permanent magnet motors have rotors radially magnetized as described in Figure 56.

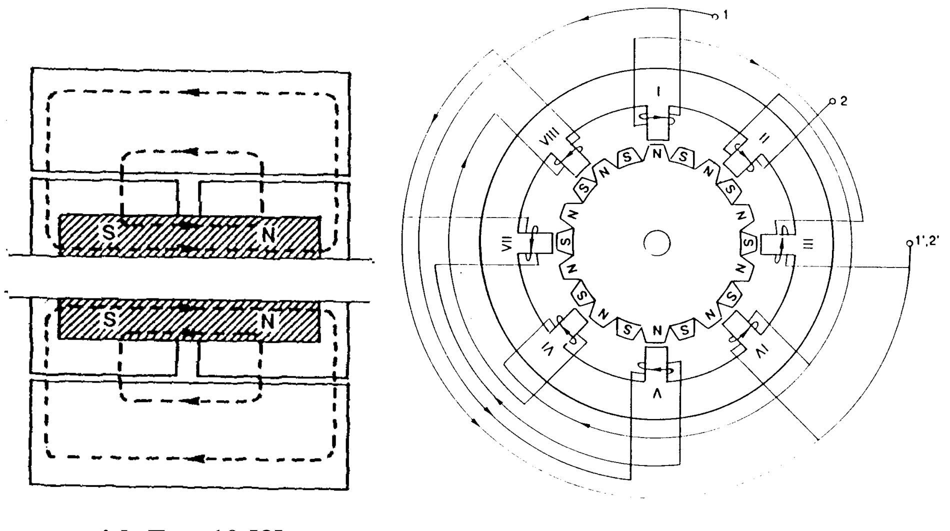


Figure 57. Motor with Z_r = 10 [3]

The motor that we consider is a commercial motor, the Crouzet 82940 002. It belongs to the third type of stepper motor, the hybrid motors. The hybrid motors are the most common type. The stator has salient poles, with two phases. The rotor includes a cylindrical axial permanent magnet, mounted on the shaft and placed between two iron disks with teeth (Figure 57). Due to the axial magnet, Z_r teeth of the same disk have identical polarity, that is, one disk bears Z_r south poles and the other bears Z_r north poles. Teeth of the disks are shifted an electric angle π, so a stator tooth faces alternately north and south poles.

The studied motor has Z_r=12 teeth on each rotor disk, so the rotor has 2*12=24 poles. It is fed in mode 2. That is, U_α and U_β are square voltages with a phase shift of π/2. Hence the motor has 48 stable positions. It is a 48 steps per tour motor. With a stepping angle of 360/48=7.5^o.

The motor is supposed to operate at synchronous speed. The rotation speed is usually proportional to the supply frequency. But when the frequency increases, erratic behavior occurs, leading to a loss of synchronism. Usual studies tend toward elaborate motor control that avoids this problem. It is hoped that a nonlinear dynamics approach will yield a better understanding and better controls.

We study a hybrid step motor, two-phased, 48 steps/tr. The motor is unloaded. The two phases, respectively noted α and β, are supplied by U_α and U_β in mode 2, i.e. two square voltages shifted of $\frac{\pi}{2}$. The motor is modeled as follows

$$L\dot{I}_\alpha = U_\alpha(t) - RI_\alpha + K_e\Omega_m \sin(Z_r\theta_m) \qquad (7)$$

$$L\dot{I}_\beta = U_\beta(t) - RI_\beta - K_e\Omega_m \cos(Z_r\theta_m) \qquad (8)$$

$$\dot{\theta}_m = \Omega_m \qquad (9)$$

$$J\dot{\Omega}_m = K_h I_\beta \cos\left(Z_r\theta_m\right) - K_h I_\alpha \sin\left(Z_r\theta_m\right) - K_d \sin\left(4Z_r\theta_m\right) - F\Omega_m - \Gamma_c \quad (10)$$

I_α and I_β: currents in phases α and β, θ_m : angular position, Ω_m : rotation speed, R=45Ω (phase resistance), L =275mH (phase inductance), Z_r =12 (teeth number), J= 18.10^{-6} kgm^2 (inertia), K_h=K_e=0,463 Nm/A (emf constant and torque constant), K_d=16 mNm (detent torque), F=10^{-4} Nms/rd (friction coefficient), Γ_c=0 (load torque).

In previous work it has been shown that this motor might exhibit chaotic behavior.[65, 106] In this section we will analyze time series from both the model and the experiment. We will use a variety of techniques from chaotic time series analysis to show that the system is indeed chaotic and that there is considerable agreement between the model and the experiment.

The analysis that follows concentrates on the simulation and four experimental data sets- two Poincare sections and two flow data sets. The section data sets both consist of 5,000 points of two dimensional data, and the flow data sets both consist of 1,000,000 points of 2-dimensional data. Each data set is labelled by the drive frequency times the number of steps per tour, 4, divided by the number of stable positions, 12. Hence the flow data set, Crouz203, corresponds to a drive frequency of 50.75Hz. A typical time series from the flow data is shown in Figure 58. Results from all the data sets will be analyzed and compared to show the dynamics of the experiment and how these differ from the dynamics of the model.

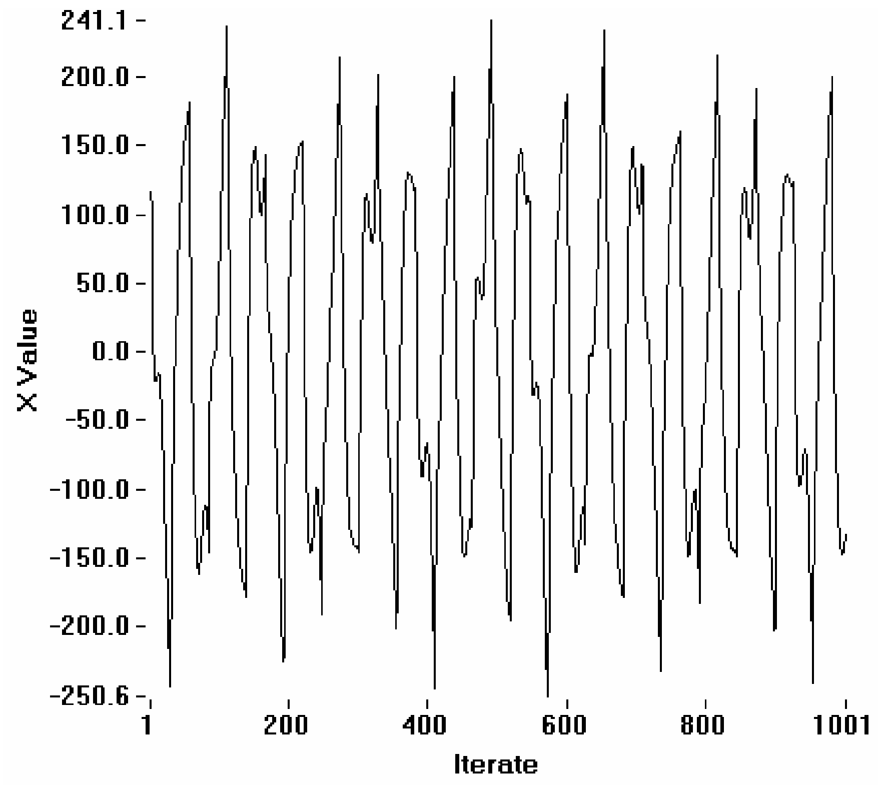


Figure 58. A plot of 10,000 points from the data set, Crouz203 (points 10000 to 10999). This portion is typical of the data.

4.2. Nonstationarity

A few simple tests were performed that would identify strong drifts in the data. A sliding window of length 20,000 points was applied to the two flow data sets. Results of the drift in the mean are depicted in Figure 59. In both data sets, it is clearly demonstrated that there was an abrupt change in the dynamics at the midpoint of each data set. The cause of this change is unknown, but it is almost certainly an artifact of the data acquisition system. However, this is not definitive. A nonlinear system could undergo a change in the dynamics that does not show up on any of these measurements. However, as can be seen from Figure 59, the mean value undergoes a fluctuating trend that does not appear to be random. This fluctuation is quite small in relation to the full extent of the data. The system may be in a long-term transient stage. The fluctuation in the mean is small (~ 1%) but it may affect the results of chaotic time series analysis methods. Due to the abrupt change in the dynamics at the midpoint of both data sets, it was decided that the analysis should be performed on each half of each data set separately.

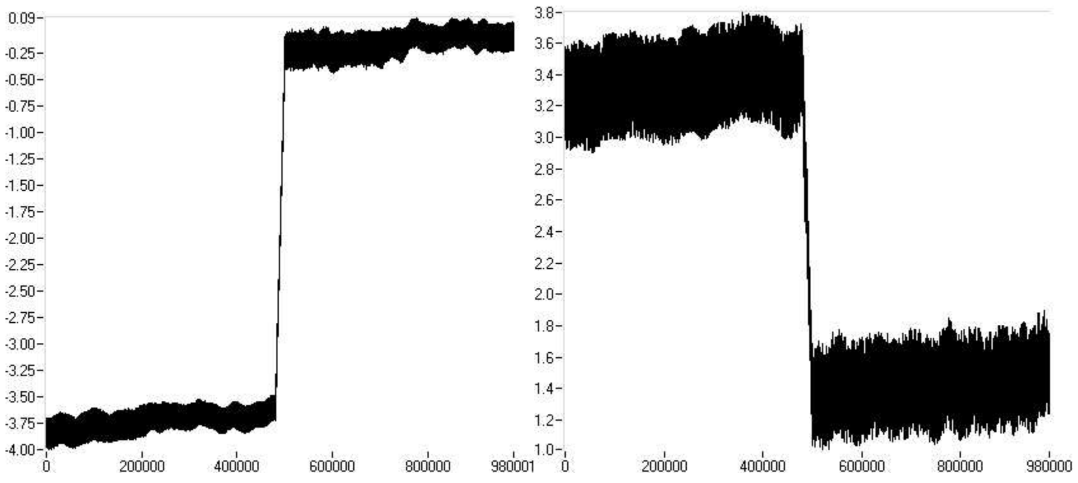


Figure 59. Nonstationary behavior of the data mean. On the left are estimates of the mean for windows of length 20000 from the data set Crouz203, on the right are estimates of the mean for windows of length 20000 from the data set Crouz239.

4.3. Delay coordinate embedding

A reasonable value for the delay may be suggested either by the first zero crossing of the autocorrelation function or by the first minimum of the mutual information function, as either value is plotted as a function of delay. The mutual information often gives a better value because it takes nonlinear correlations into account. However, for the step motor data, the mutual information function and the autocorrelation function were in perfect agreement. As shown in Figure 60 both values suggested a delay of approximately 13. Other estimates of the appropriate delay from any of the flow data sets gave values between the range of 9 to 16. This was in agreement with visual inspection since 2 and 3 dimensional plots revealed the most structure near this value of delay (see Figure 61). Structure is clearly evident in these plots. Unfortunately, they also reveal a complexity or noise dependence that makes the fine scale structure very difficult to detect.

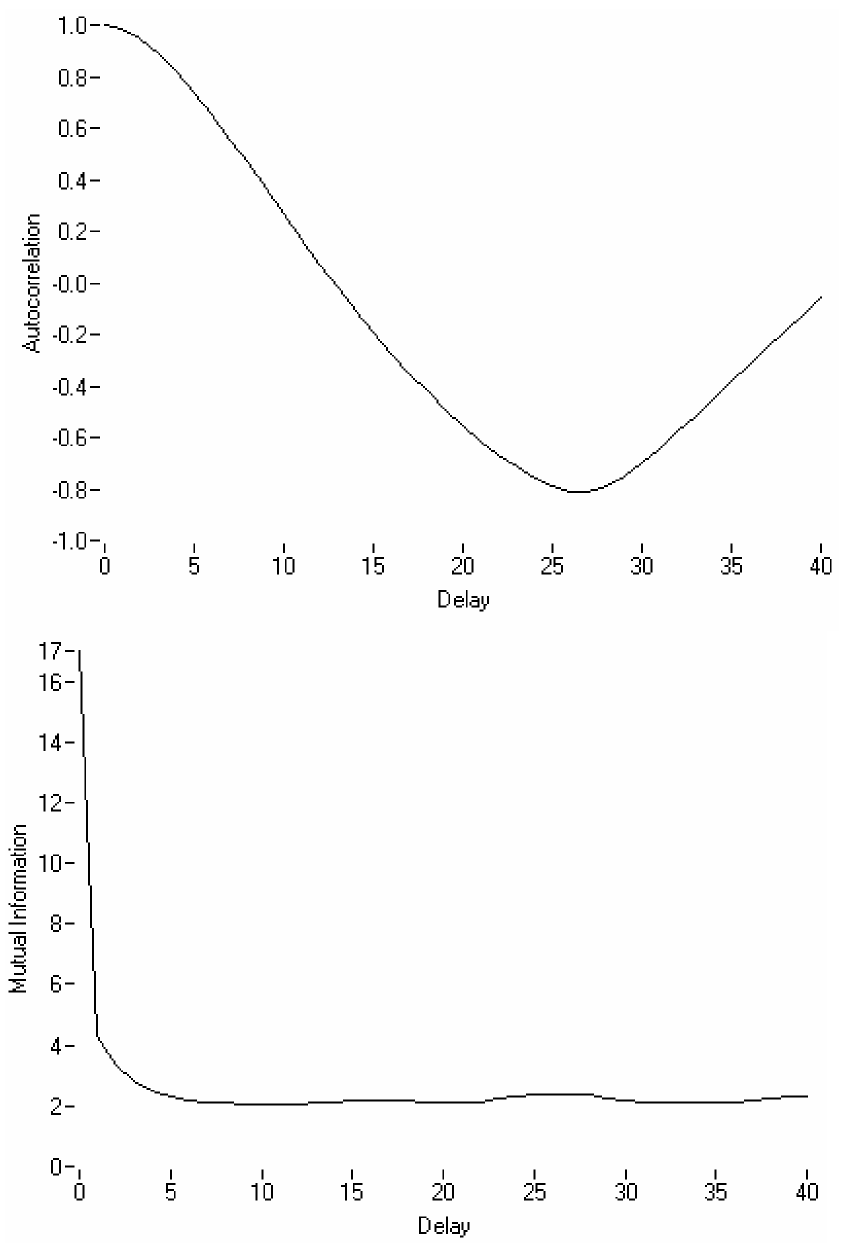


Figure 60. Two different techniques for estimating a delay to use in a delay coordinate embedding. The first method is to choose the first minimum of the mutual information function. The second, less sophisticated method uses the first zero crossing of the autocorrelation function. Both methods suggest a delay of approximately 13. Both plots use the first half of the data set Crouz203.

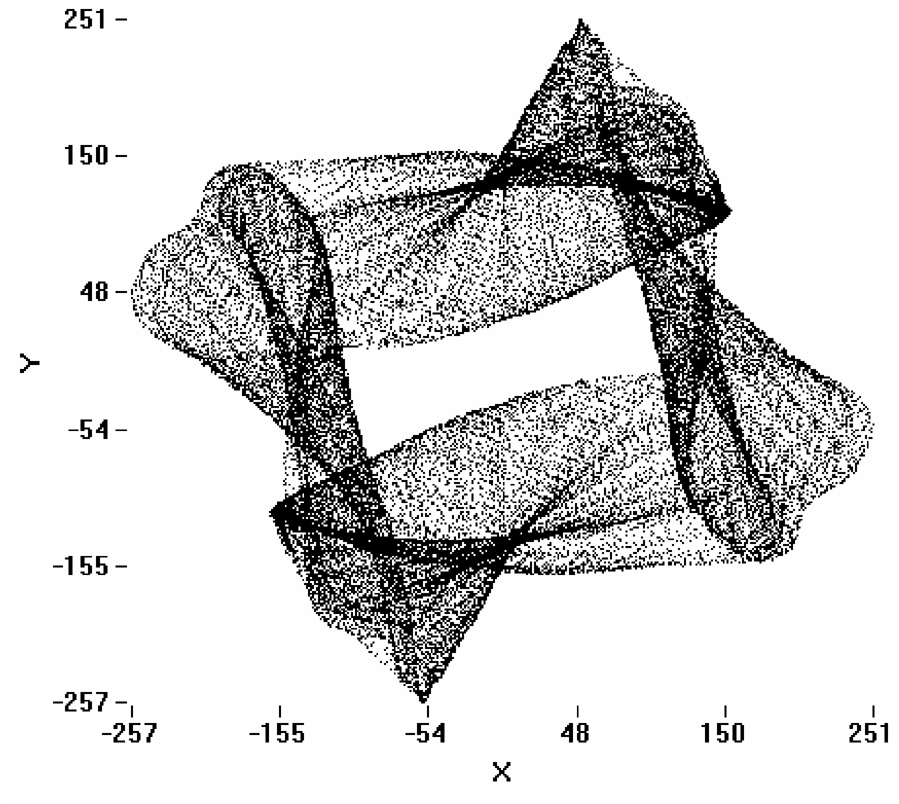


Figure 61. A two dimensional plot of the first 50,000 points from Crouz203, with a delay of 11.

The method of false nearest neighbors was chosen as the primary technique for determining the embedding dimension. The results agreed with what was suggested for application with real world data. As shown in Figure 7, the percentage of false neighbors dropped dramatically as the embedding dimension increases from 5 to 6. As will be shown later, this is in at least rough agreement with what was found to be a suitable embedding dimension for determination of Lyapunov exponents or fractal dimensions.

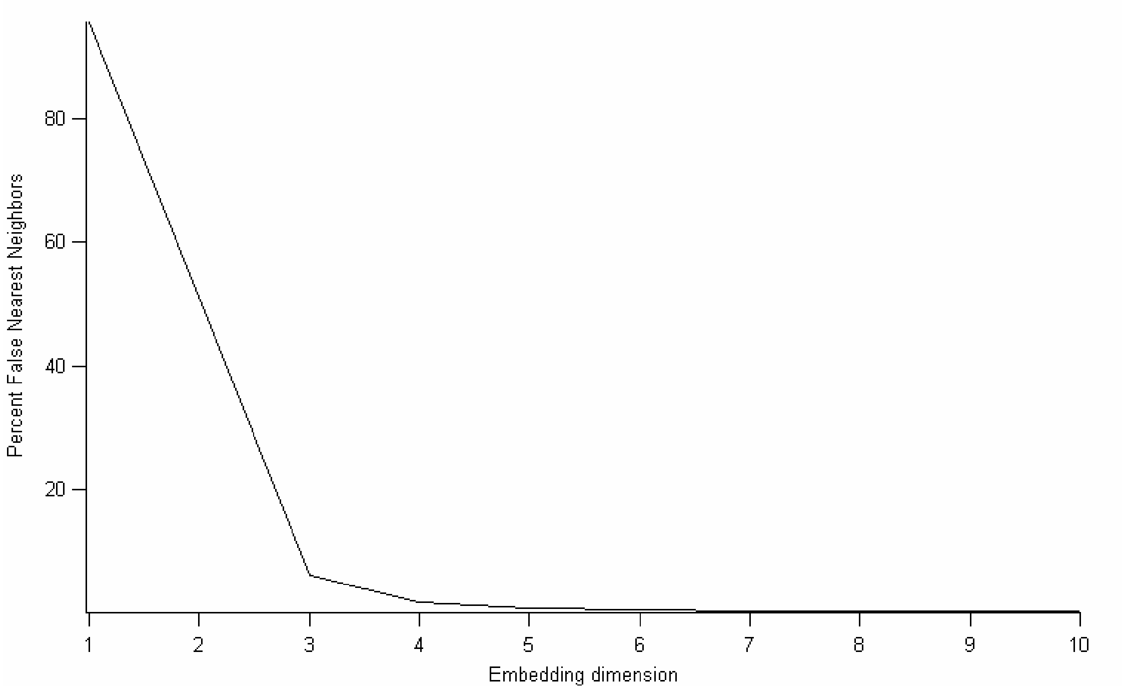


Figure 62. A plot of the results of the false nearest neighbors routine as applied to the first half of Crouz203. An appropriate embedding dimension is found when the percentage of false near neighbors drops to a value near zero. This plot indicates that the embedding dimension should be at least 4.

4.4. Fractal dimension

Analysis was attempted on data sets of varying size and varying embedding dimension. Results of estimations of the correlation dimension for the second half of Crouz239 are depicted in Figure 63. A plateau is evident for log(ε) in the range –1.7 to –3.0. Here, the correlation dimension can be estimated to be between 1.1 and 1.5. This is a fairly low dimensional system and thus it should be relatively easy to manipulate. Although this value agrees roughly with the choice of embedding dimension, more analysis was necessary to confirm the results.

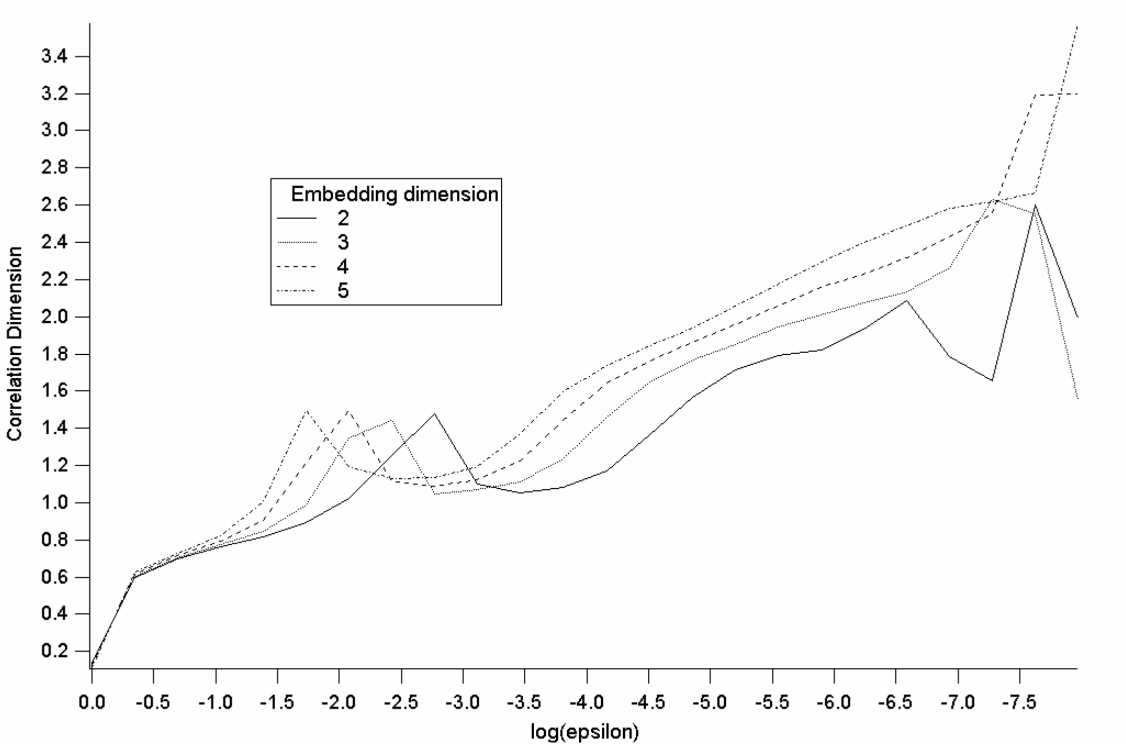


Figure 63. This measures the correlation dimension estimated from the second half of data set Crouz239. The correlation dimension is estimated from the slope of log(C(ε))vs. log(ε). It can be estimated from the plateau region of the plot, where it is between 1.1 and 1.5.

Figure 64 presents the results of our calculations performed on step motor data. Displayed are estimates of the first four generalized entropies for varying box size with an embedding dimension of 4. Additional tests were also performed for the embedding dimensions 3-6, and for the next four generalized entropies. The results indicated that, for $p > q$, $D(p) \leq D(q)$, which agrees with theory. For large box size, the box counting dimension varies widely from the others, since the box counting dimension $D(0)$ is more susceptible to errors. It is also a poor quantity to use since it says nothing about the density of the attractor, only about its shape. However, the box counting dimension and all the others converge in the mid-region, before diverging slightly and then dropping to zero (due to data set size). It is this mid region that parallels the plateau region of the Grassberger-Proccacia algorithm.

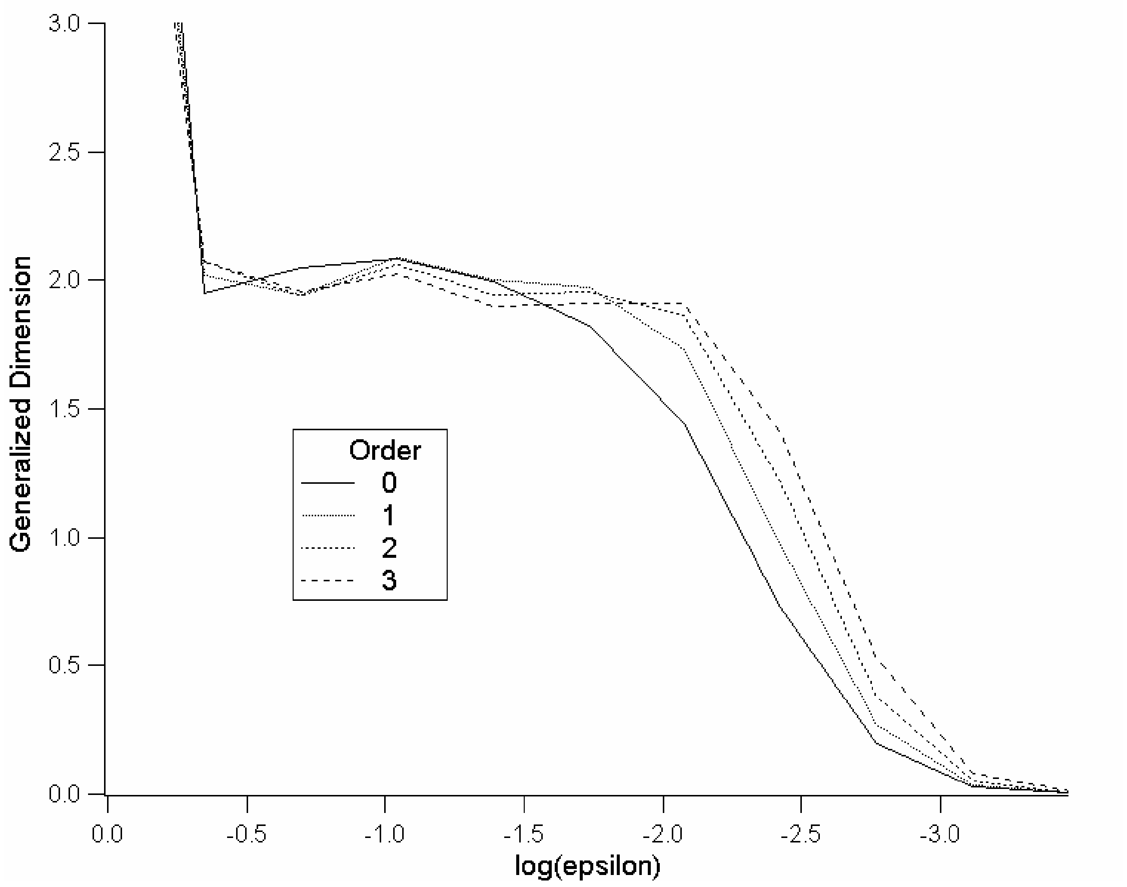


Figure 64. The first four generalized entropies. The values may be estimated as $D(0) = 2.08 \pm 0.1$**,** $D(1) = 2.09 \pm 0.1$**,** $D(2) = 2.05 \pm 0.2$ **and** $D(0) = 2.02 \pm 0.2$**. Each of these was calculated using the second half of Crouz203 embedded in 4 dimensions.**

The estimates for fractal dimension ranged from 1.8 to 2.2, for all fractal dimensions calculated when embedding dimension was greater than or equal to 4. With the exception of the box counting dimension, this was true for all of the first four generalized dimensions. The correlation dimension, for instance, was estimated at $D(2) = 2.05 \pm 0.2$, where the error was estimated based on the fluctuation of the slope of the entropy in the mid region. This agrees with our choice of 4 for the embedding dimension, and is in rough agreement with the result from the Grassberger-Proccacia algorithm. A fractal dimension of up to 2.5 indicates that an embedding dimension as high as 5 may be necessary, but as is often the case, a lower embedding dimension may be used.

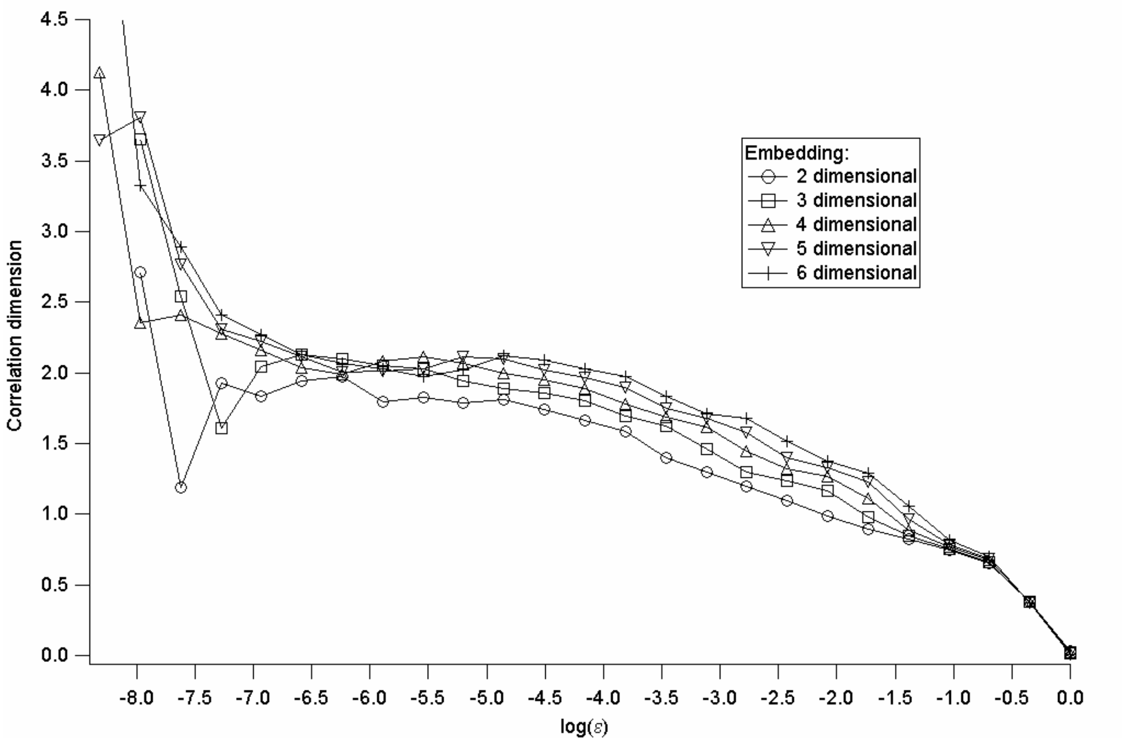


Figure 65. This measures the correlation dimension estimated from the first half of data set Crouz203. The correlation dimension is estimated from the slope of log(C(ε) vs. log(ε). It can be estimated from the plateau region of the plot, where it is approximately 2.2

4.5. Lyapunov exponents

Before determination of Lyapunov exponents was attempted, the data was embedded with a delay of 13, as suggested by the false nearest neighbors routine, the autocorrelation function and the mutual information function. Fractal dimension was estimated as between 2 and 4, so a local embedding dimension of 4,5 or 6 was chosen, yielding 3, 4, or 5 exponents in the calculation of the full spectra. As shall be seen, this is in agreement with the observation that the sum of the exponents must be negative.

The Eckmann-Ruelle[96] method was used to determine Lyapunov spectra. The folding of the attractor brings diverging orbits back together. So any effects of nonlinearities will most likely serve to move all exponents closer to zero. Hence a slight underestimate of the positive exponents was expected.

exponents	embedding dimension	points in fit	iteration size	λ_1	λ_2	λ_3	λ_4
3	3	9	1	0.381672	0.045924	-0.307301	
3	3	9	2	0.354878	0.064599	-0.189015	
3	3	9	3	0.310627	0.068485	-0.141331	
3	4	16	1	0.046712	-0.032937	-0.217158	
3	4	16	2	0.095562	-0.027207	-0.205269	
3	4	16	3	0.098218	-0.02492	-0.174718	
3	5	25	1	-0.007088	-0.04303	-0.183316	
3	5	25	2	0.048134	-0.036329	-0.202159	
3	5	25	3	0.056969	-0.037843	-0.177225	
4	4	16	1	0.099903	0.012053	-0.04605	-0.310671
4	4	16	2	0.153753	0.018143	-0.057847	-0.252268
4	4	16	3	0.14469	0.018831	-0.057908	-0.211251
4	**5**	**25**	**1**	**0.01649**	**-0.010436**	**-0.050132**	**-0.28072**
4	5	25	2	0.086138	-0.00244	-0.063969	-0.259551
4	5	25	3	0.084314	-0.002873	-0.066443	-0.21669

Table 7. Estimation of Lyapunov exponents for the first half of the flow data set, Crouz239. The bold faced selection represent the best estimates of the exponents, since it has an exponent closest to zero and the sum of the exponents is negative.

In Table 7 and Table 8 results of exponent calculations are provided. The exponents are given in units of 1/time, where the time scale is defined so that the time between samples is 1. Many more calculations were performed until a reasonable and stable parameter regime was found for both methods. Note that the zero exponent snaps into place for appropriate parameter settings.

exponents	embedding dimension	points in fit	iteratio n size	λ_1	λ_2	λ_3	λ_4
3	3	9	1	0.404882	0.04426	-0.32303	
3	3	9	2	0.37753	0.068513	-0.192187	
3	3	9	3	0.329768	0.072848	-0.137364	
3	4	16	1	0.051662	-0.033821	-0.228997	
3	4	16	2	0.106886	-0.02685	-0.208742	
3	4	16	3	0.112637	-0.024475	-0.172856	
3	5	25	1	-0.00583	-0.044957	-0.187635	
3	5	25	2	0.059612	-0.037073	-0.210866	
3	5	25	3	0.068587	-0.036298	-0.183896	
4	4	16	1	0.111423	0.014208	-0.04756	-0.306655
4	4	16	2	0.162717	0.019409	-0.057432	-0.247336
4	4	16	3	0.155307	0.020102	-0.056715	-0.21321
4	5	25	1	0.021594	-0.009072	-0.045046	-0.266796
4	**5**	**25**	**2**	**0.088914**	**-0.004143**	**-0.063278**	**-0.251144**
4	5	25	3	0.089064	-0.00702	-0.066332	-0.221641

Table 8. Estimation of Lyapunov exponents for the second half of the flow data set, Crouz239. The bold faced selection represent the best estimates of the exponents, since it has an exponent closest to zero and the sum of the exponents is negative.

Several of our criteria are determined immediately upon inspection. The zero exponent was identified with a high degree of accuracy. The sum of the exponents is negative, while the sum of the first two is positive. This indicates that a fractal dimension between 2 and 3 was a reasonable estimate.

exponents	embedding dimension	points in fit	iteration size	λ_1	λ_2	λ_3	λ_4
2	2	5	1	2.453502	-0.172849		
2	2	5	2	1.624457	0.83530		
2	2	5	3	1.149623	0.673577		
2	3	9	1	-0.467303	-0.915287		
2	3	9	2	0.601609	-0.486722		
2	3	9	3	0.671703	0.206204		
3	3	9	1	1.440059	0.210937	-0.593496	
3	3	9	2	1.084312	0.539116	-0.225776	
3	3	9	3	0.850945	0.576722	0.149585	
3	4	16	1	-0.258083	-0.46754	-0.890886	
3	**4**	**16**	**2**	**0.466546**	**-0.056553**	**-0.692082**	
3	4	16	3	0.456796	0.142458	-0.354413	
4	4	16	1	0.953211	0.240227	-0.132071	-0.671263
4	4	16	2	0.7604	0.425944	0.043779	-0.498245
4	4	16	3	0.616248	0.408489	0.155694	-0.240587

Table 9. Estimation of Lyapunov exponents for the sectioned data set, Crouz203section. The bold faced selection represent the best estimates of the exponents, since the sum of the exponents is negative and it is proportionally similar to the results of Table 7 and Table 8.

Sectioning the data introduced additional noise and measurement of exponents from the section was even more uncertain. Thus it was not possible to get agreement between exponent estimates from the section and from the flow, nor was it expected. However, Lyapunov spectrum estimates from the Poincare section data is provided in Table 9. We note that the estimates here are scaled from the estimates provided in Table 7 and Table 8, because the sampling rate is different. Although the sample rate for the sectioned data, Crouz203section is known to be 50.75 Hz, the sampling rate of the flow data sets is unknown.

4.6. Conclusion

The data from the step motor appears to represent a four dimensional system with one positive Lyapunov exponents. This system may therefore be considered low dimensional and chaotic. As such, computation of fractal dimension and of

Lyapunov exponents is possible and accurate. Although the data appeared nonstationary, it is still an excellent system for analysis. This is because the system is less dependent on temperature and other external factors than the magnetoelastic ribbon, and less noisy than the combustion engine.

One hindrance in the analysis was a lack of communication between the experimentalists and the author. Because of this, many issues such as the cause of the huge jump in Figure 59, and the unknown sampling rate of the flow data sets remain unresolved. However, the collaboration continues and it is expected that better data will be available for future analysis.

5. Multibit chaotic sigma delta modulation

5.1. Introduction

Conventional analog to digital and digital to analog converters are based on the linear, multibit Pulse Code Modulation (PCM) format. They require high-precision analog circuits and they are vulnerable to noise and interference. In recent years, the consumer audio industry has moved towards oversampled nonlinear converters for many applications. An important oversampling A-D or D-A conversion strategy now employed is the sigma delta modulator. In sigma delta converters the signal is sampled at a high sampling frequency and converted to a low bit binary output. They are cheaper to manufacture than PCM converters, consume less power, and operate well at the voltage range used in battery-powered audio equipment. Thus sigma delta modulators are used in the digital to analog converters of many compact disc players and in the audio processing of many wireless communications systems, such as cellular phone technology. In addition, the Sigma-Delta bitstream format is under consideration for the mastering and archiving of audio recordings.

Unfortunately, sigma delta modulators are susceptible to limit cycle oscillations that are not present in the input signal. These idle tones may be audible to the listener when sigma-delta modulation is used for audio signal processing. One method of eliminating idle tones is the operation of a sigma delta modulator in the chaotic regime. But chaotic modulation of a first order sigma delta modulator, as previously proposed, is a poor system for signal processing. The modulator may become unstable (input to the quantizer becomes unbounded) and can result in inaccurate output.

In this section, we investigate chaotic phenomena in single bit and multibit first order sigma-delta modulators. We look at a variety of different ways to operate a sigma delta modulator chaotically. Particular attention is placed on the occurrence of periodic orbits or limit cycles. By investigating the nonlinear dynamics of these systems, we are able to show that a new form of chaotic sigma delta modulation may be used. We show that this variation on a traditional first order sigma-delta modulator, together with a multibit implementation, may produce an effective, stable chaotic modulator that accurately encodes the input and also helps prevent the occurrence of idle tones.

5.2. Background

Sigma-delta modulators operate using a tradeoff between oversampling and low resolution quantization. That is, a signal is sampled at much higher than the Nyquist frequency, typically with one bit quantization, so that the signal may be effectively quantized with a resolution on the order of 14-20 bits.[108] Recent work has concentrated on tone suppression[109-111], multibit modulation[112] and chaotic modulation.[109, 113-115]

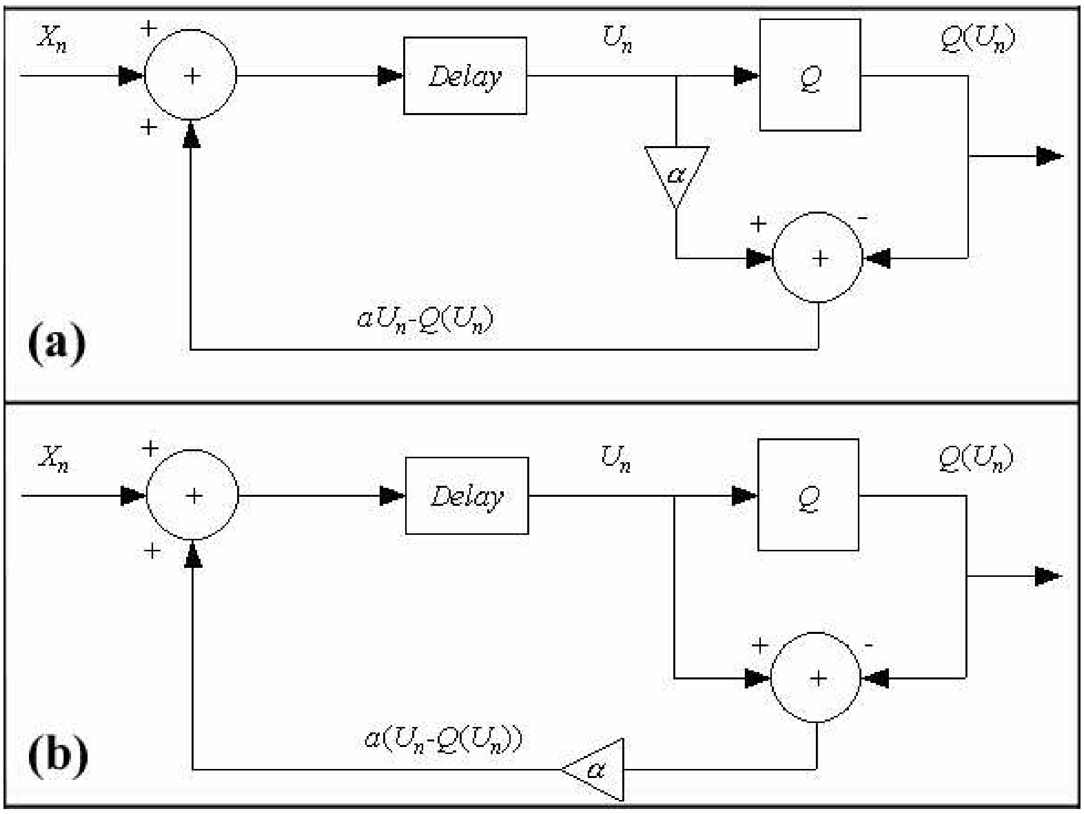


Figure 66. Block diagrams for the two systems. In (a), gain is applied just to the integrator output. In figure (b), gain is applied to the quantizer error, that is, the difference between the integrator output and the quantizer output.

The simplest, first order sigma-delta modulator consists of a 1-bit quantizer embedded in a negative feedback loop which also contains a discrete-time integrator, as depicted in Figure 66(a). The input to the modulator is sampled at a frequency higher than the Nyquist frequency and is converted into a binary output. The system may be represented by the map[116]

$$U_n = \alpha U_{n-1} + X_{n-1} - Q(U_{n-1}) \qquad (11)$$

where X represents the input signal, bounded by -1 and $+1$, and Q is the quantizer

$$Q(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{if } u < 0 \end{cases} \qquad (12)$$

In this representation, the output $Q(U_n)$ represents the quantization of input $X_{n\text{-}1}$. The initial conditions, X_0 and U_0, are typically set to 0. On average, the quantized output will be approximately equal to the input. If α=1, then this system works by quantizing the difference between the input and the accumulated error. When the error grows sufficiently large, the quantizer will flip in order to reduce the error. The operation of such a first order sigma delta modulator is depicted in Figure 67. The input is a 1.5kHz sine wave with amplitude 0.8 sampled at a frequency of 256kHz (these parameters were chosen to accentuate the behavior of the modulator). Typically, the integrator leaks due to finite operational amplifier gain, which is represented by α<1. If α>1, then the modulator may behave chaotically for constant input. This chaotic system has been studied extensively in [115] and [117] and the references therein.

If a gain is added to the quantization error (difference between integrator output and quantized output), as opposed to the integrator output, then the difference equation describing this modified sigma delta modulator takes the form

$$U_n = X_{n-1} + \alpha(U_{n-1} - Q(U_{n-1})) \qquad (13)$$

This system is depicted in Figure 66(b). It is relatively simple to implement in a circuit, and still accomplishes the goals of sigma delta modulation.

An alternative representation of (13) is found by defining $V_n = U_n / \alpha$ and $Y_n = X_n / \alpha$. Hence

$$V_n = \alpha V_{n-1} + Y_{n-1} - Q(\alpha V_{n-1}) \qquad (14)$$

This allows the gain to be applied only to the integrator output and to the input signal. Thus no gain needs to be applied directly to the quantization. In the case of a single-bit quantizer, (14) has the same functional form as (11).

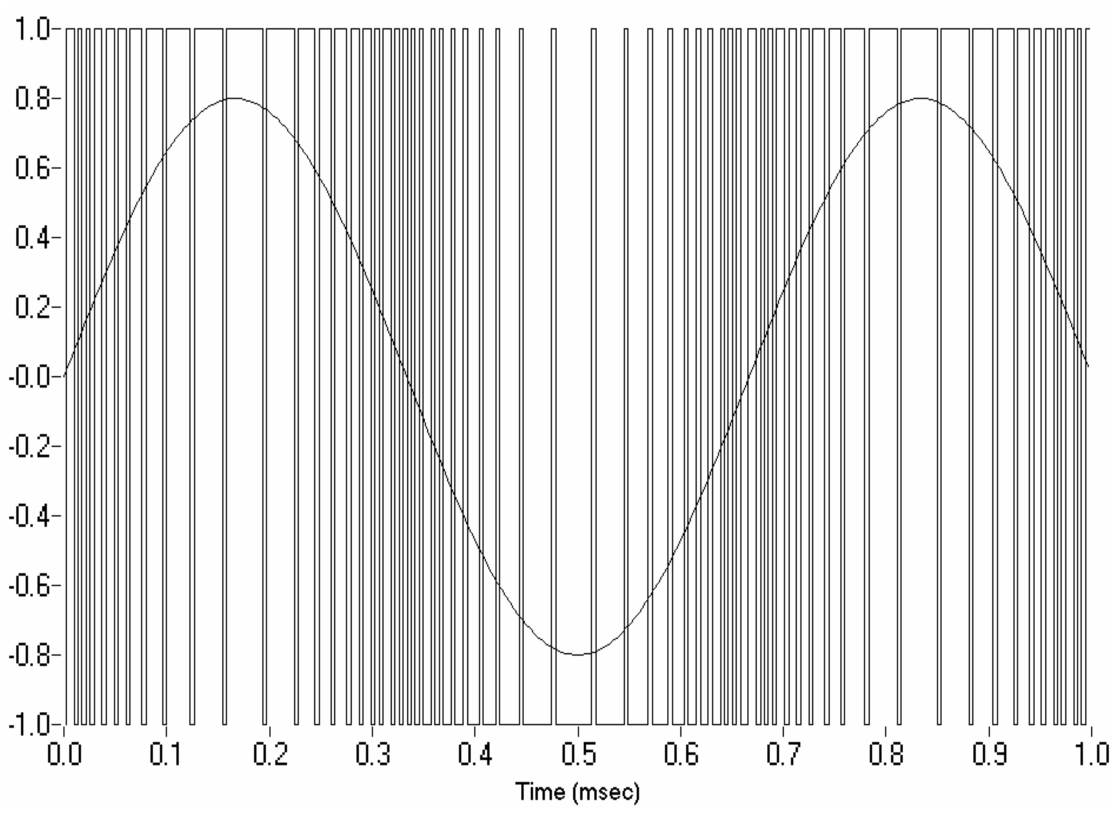


Figure 67. A 1.5 kHz sine wave with an amplitude of 0.8 is sampled at a frequency of 256kHz. The input sine wave and the quantized output of the sigma delta modulator are depicted.

In this work, we consider chaotic modulators where a gain term multiplies either the integrator output (11) or the error term (13). In particular, we consider whether either form of chaotic modulation is an effective means of idle tone prevention. We demonstrate that for the case of gain applied to integrator output, although an implementation of a chaotic multibit modulator may lead to idle tone suppression, it may not be practical. This is because in many cases, the output of a chaotic modulator does not effectively approximate the input.

A multibit implementation of either (11) or (13) may offer increased resolution in the quantization. Rather than quantizing the output into –1 and 1, the output can instead assume a range of discrete values. For an n bit first order sigma delta modulator, the quantized output can assume one of m=2^n states. The quantizer would take the form

$$Q(u) = \begin{cases} 2(m-1)/m & \text{if} \quad u \geq 2(m-2)/m \\ 2(m-3)/m & \text{if} \quad 2(m-2)/m > u \geq 2(m-4)/m \\ 2(m-5)/m & \text{if} \quad 2(m-4)/m > u \geq 2(m-6)/m \\ \vdots & \qquad \vdots \\ -2(m-1)/m & \text{if} \quad -2(m-2)/m > u \end{cases} \quad (15)$$

Here, for reasons explained in the section on bifurcations, we assume that quantizer input is in the range –2 to 2. Thus, the systems that will be studied are

1. The 1st order, single bit sigma delta modulator with gain applied to the integrator: equations (11) and (12).

2. The 1st order, single bit sigma delta modulator with gain applied to the error: equations (13) and (12).

3. The 1st order, multi bit sigma delta modulator with gain applied to the integrator: equations (11) and (15).

4. The 1st order, multi bit sigma delta modulator with gain applied to the error: equations (13) and (15).

5.3. Analysis

5.3.1. Bifurcations

System 1 (Equations (11) and (12)), a first order, single bit sigma delta modulator, is perhaps the most well known and simplest form of sigma delta modulation. It exhibits chaos if the gain is in the range $1 < \alpha \leq 2$. The bifurcation diagram of this system is depicted in Figure 68.

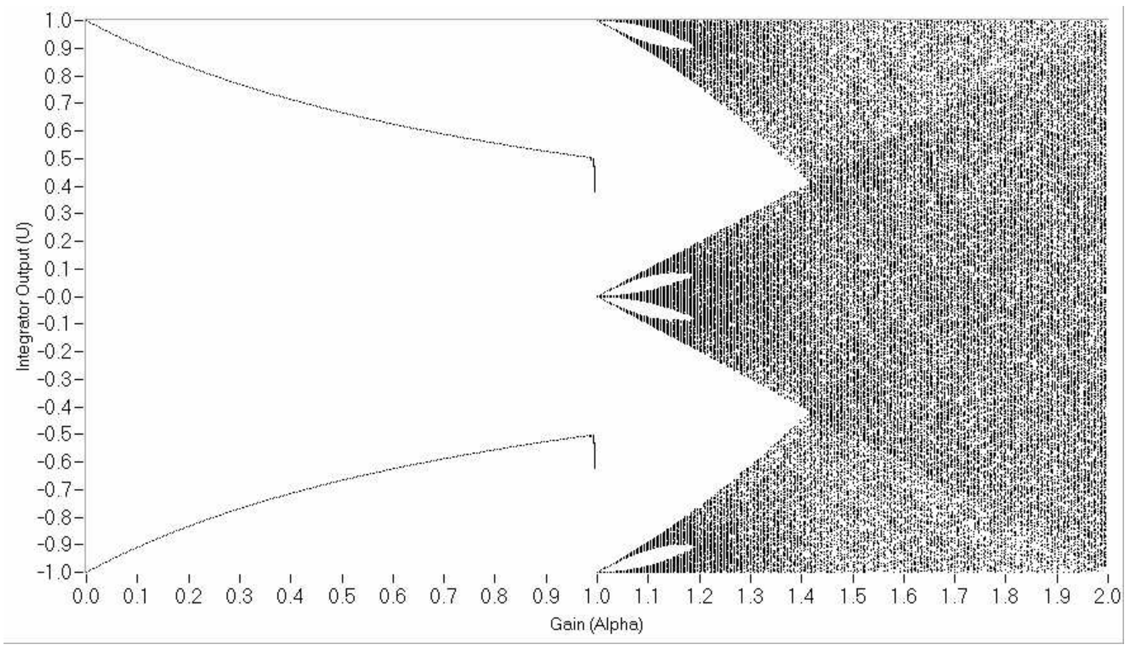


Figure 68. Bifurcation diagram of a first order, one bit sigma delta modulator with 0 input and gain applied to the integrator output (System 1).

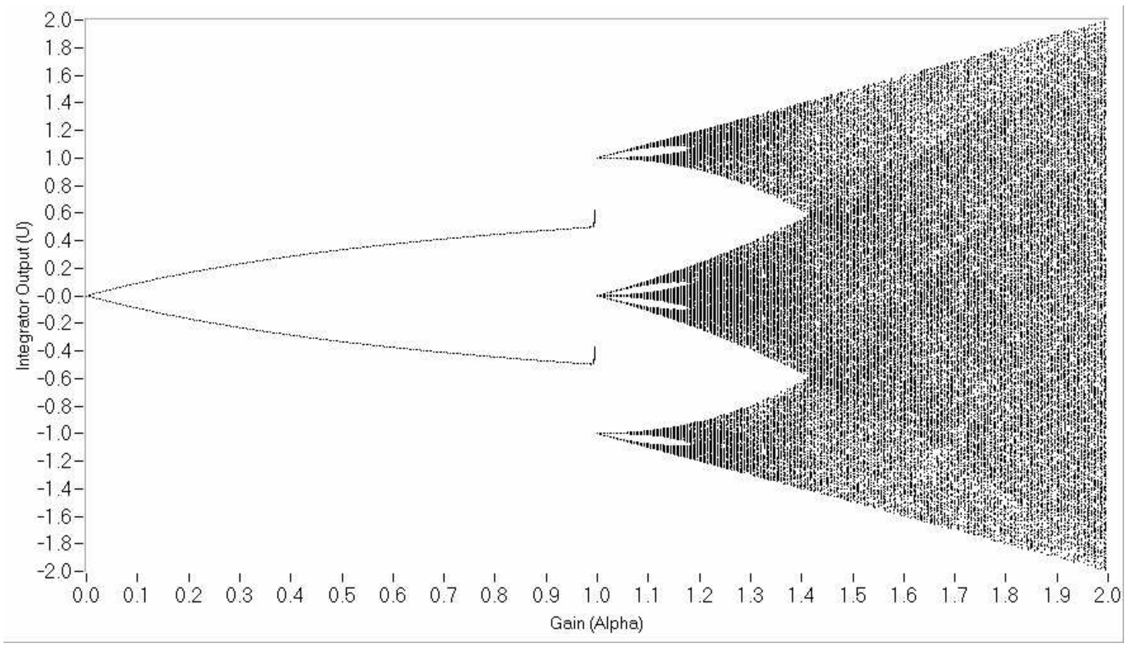


Figure 69. Bifurcation diagram of a first order, one bit sigma delta modulator with 0 input and gain applied to the error (System 2).

System 2 (Equations (13) and (12)), has a slightly different bifurcation diagram. It also exhibits chaos if the gain is in the range $1 < \alpha \leq 2$. The bifurcation diagram of this system is depicted in Figure 69. Notably, the dynamics here are somewhat different. For instance, the integrator output does not immediately reach the extremes as α is increased past 1. The full range of integrator output is between –2 and 2, and for $\alpha \geq 1$, the range of output extends from $-\alpha$ to α. This is a direct consequence of the fact that the bifurcation diagram measures possible values of $U_n = \alpha V_n$ from Equation (13). It may seem problematic at first, since the expected input, *X*, is between –1 and 1. However, as shall be seen later, as long as the average integrator output sufficiently approximates the input, then this is not a difficulty. We simply require that the input signal be bounded by ± 1, even though the quantizer can accept input bounded by ± 2.

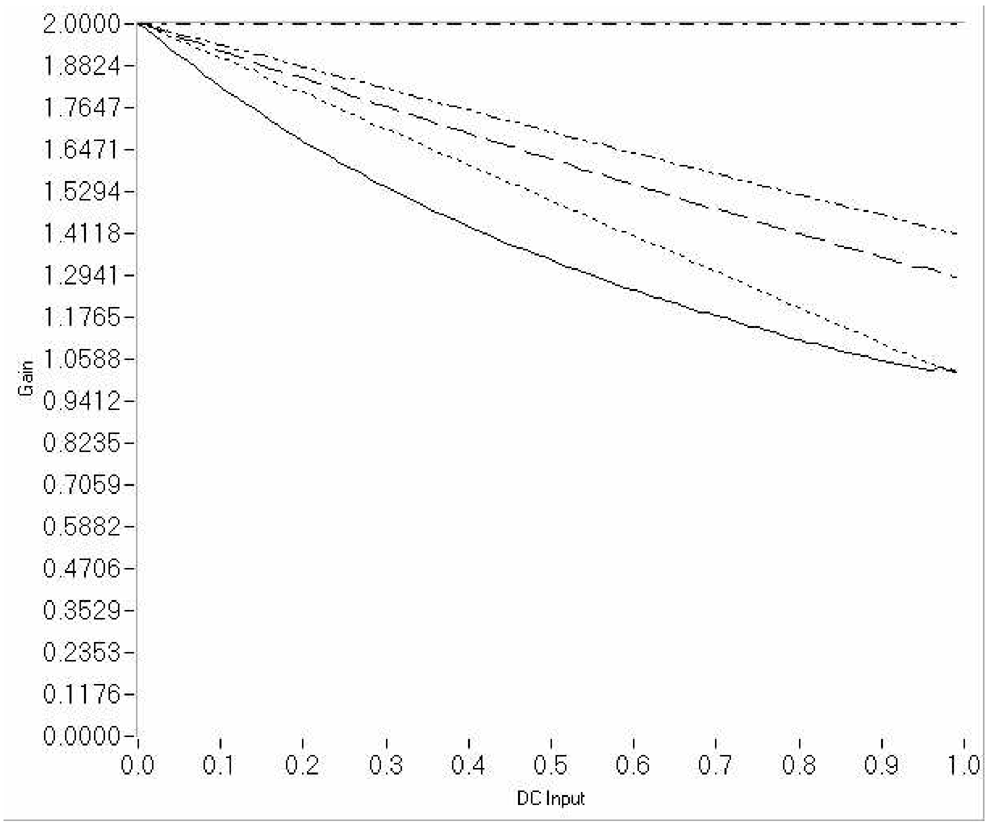


Figure 70. The stability regime of a sigma delta modulator for various values of gain and constant input in the range 0 to 1. The solid line and below represents the bounded stable regime for a 1 bit modulator with gain applied to the integrator output (System 1). The dashed line represents the bounded stable regime for a 2 bit modulator and the dot-dot-dashed line for a 3 bit modulator (System 3). For a modulator with gain applied to the error, the dotted line and below represents the stable regime for the 1 bit case (System 2), and the dot-dashed line and below represents the stable regime for the 2 bit case (System 4).

5.3.2. Stability

One difficulty with operating a sigma delta modulator with greater than unity gain is that, for nonzero input, the modulator may become unstable. That is, $U_n \to \pm\infty$ as $n \to \infty$. This is illustrated in Figure 70, which depicts the size of the stable regime for input $0 \leq X \leq 1$ (the plot is symmetric for $-1 \leq X \leq 0$) and gain $0 \leq \alpha \leq 2$. Operating a one bit sigma delta modulator, Equation (12), in the chaotic regime becomes unworkable for any large input, since the integrator output diverges. For this and other reasons (notably poor SNR ratio), System 1 is considered a poor means of tone suppression,[118, 119]. Although this can be improved through the use of a multibit quantizer, it is still problematic.

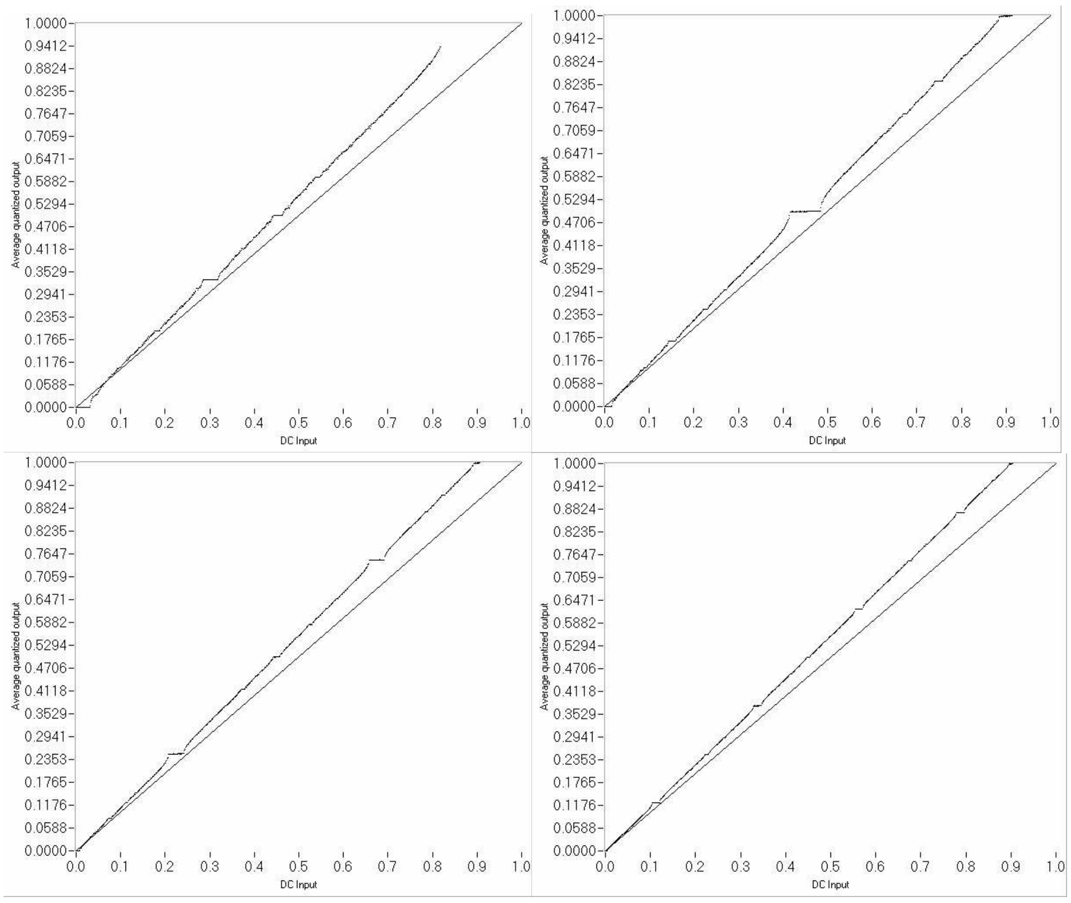


Figure 71. Clockwise from top-left. The average quantized output as a function of the input for a 1 bit, 2 bit, 3 bit, and 4 bit sigma delta modulator with gain applied to integrator output. The gain is set to 1.1. The 45 degree line represents the ideal average quantization.

The stable regime is significantly increased if the gain is applied to the difference between the quantizer output and the integrator output (Equation (13)). For a 1 bit quantizer, the stable regime is greater than a 2 bit traditional sigma delta modulator. If we move to a 2 bit quantizer implementation of Equation (13), then the entirety of the domain has bounded integrator output.

5.3.3. Quantization error

A minimum necessary requirement for sigma delta modulation is that the quantizer output approximate the input signal. That is,

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}Q(U_i)=X \qquad (16)$$

for constant input *X*. This must hold for any allowable input. All the modulators considered have input in the range $-1\leq X\leq 1$. With unity gain ($\alpha=1$), Equation (16) holds for the single and multibit, first order sigma delta modulators. However, this is typically not true for $\alpha\neq 1$. Feely and Chua[120] showed that integrator leak, $\alpha<1$, may cause the average output of the sigma delta modulator to assume discrete values that misrepresent the input. The resulting structure of average quantized output as a function of the input is known as a devil's staircase. As shown in Figure 71, this is also the case for a traditional sigma delta modulator with $\alpha>1$ (Equation (11)). In fact, for nonunity gain, the average output is approximately αX. Using a multibit modulator is not sufficient to alleviate this problem. Although it increases the bounded region of the modulator, and minimizes the discrete steps, it does not succeed in making the output more effectively track the input This is a fundamental problem that is often overlooked in the literature.[121]

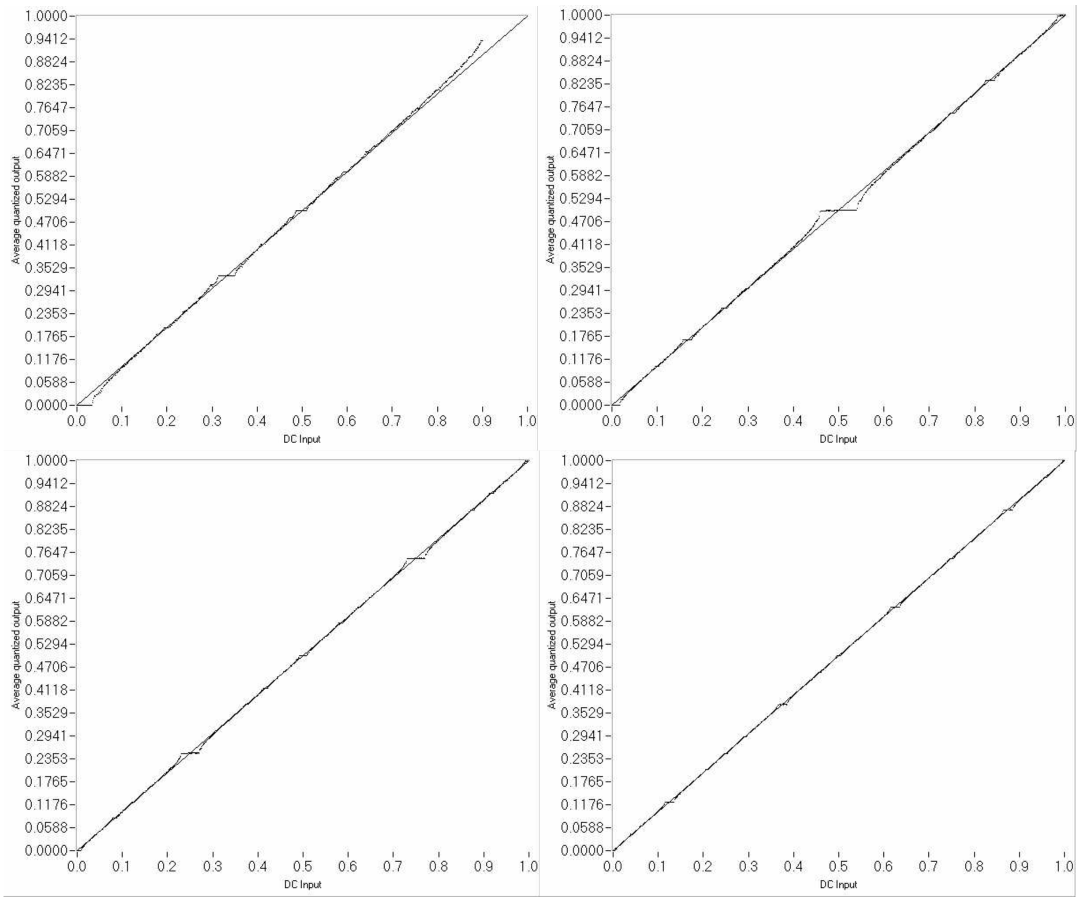


Figure 72. Clockwise from top-left. The average quantized output as a function of the input for a 1 bit, 2 bit, 3 bit, and 4 bit sigma delta modulator with gain applied to quantization error. The gain is set to 1.1. The 45 degree line represents the ideal average quantization.

However, the modified modulator of (13) behaves quite differently. Figure 72 shows that this modulator, although assuming discrete values, still approximates the input. That is, the average output as a function of input has a slope of 1. In addition, a multibit implementation helps to minimize the length of the stairs in the devil's staircase structure. In other words, for a modulator operating in the chaotic region, as the number of bits used by the quantizer is increased, the average quantized output approaches the average quantized output of an ideal sigma delta modulator.

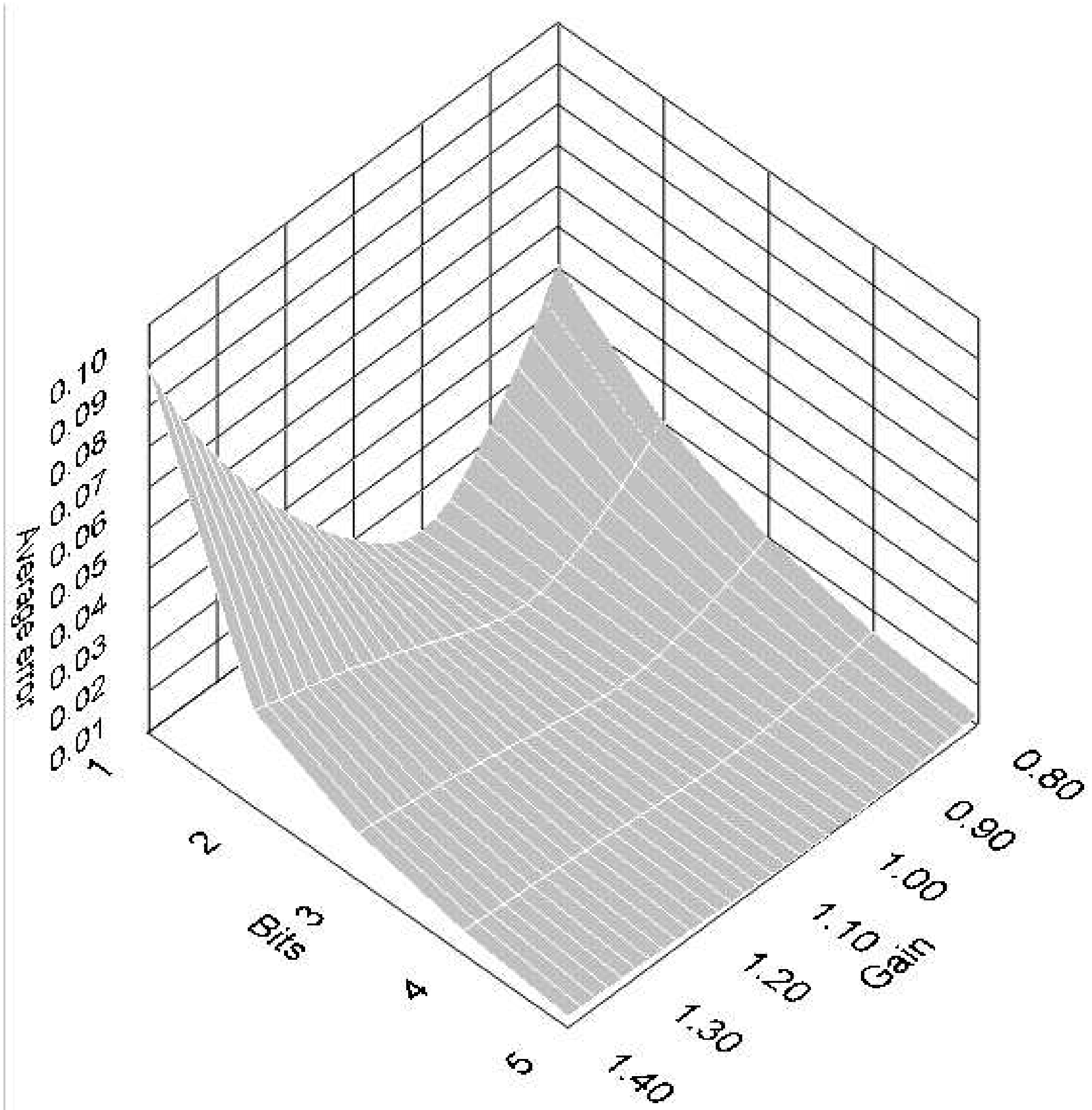


Figure 73. A three-dimensional plot of average output error vs number of bits vs gain. Input is varied over the range –1 to 1. For each input number of bits, and gain, the absolute value of the input minus the average quantized output is found. This is then averaged over the entire input range, to give an average error dependent on the number of bits in the quantizer and the gain in the modulator (Equation (13)). 1 to 5 bit modulators having gain ranging from 0.8 to 1.4 are compared.

To further demonstrate this, a 3 dimensional plot of average output error vs number of bits vs gain is depicted in Figure 73. Notice that quantizer error is greatly reduced as the number of bits in the quantizer is increased. Thus the loss in accuracy as the gain is increased (and chaos is introduced) may be compensated for by adding bits to the quantizer.

5.3.4. Symbol sequences

On a practical level the output prior to quantization is not of primary concern. More importantly, the quantized output must accurately encode the input signal without producing idle tones. That is, tones which are not present in the input signal may appear in the quantized output. For instance, a constant input of 0 with α=1 and initial condition U_0=0 will produce an output sequence of 1,-1,1,-1,1,-1… This gives the false impression of an oscillating input. Longer period cycles will produce tones at lower frequencies which may appear audible to the listener.

One proposed method of eliminating these tones is to operate the sigma delta modulator in the chaotic regime. Although the output will still approximate the input, limit cycles might be eliminated. As an example, a constant input of 0 with α=1.5 will produce an output 1,-1,-1,1,-1,1,-1,-1,1,-1,1,-1,1,1… This is an endless pattern that never settles into a limit cycle.

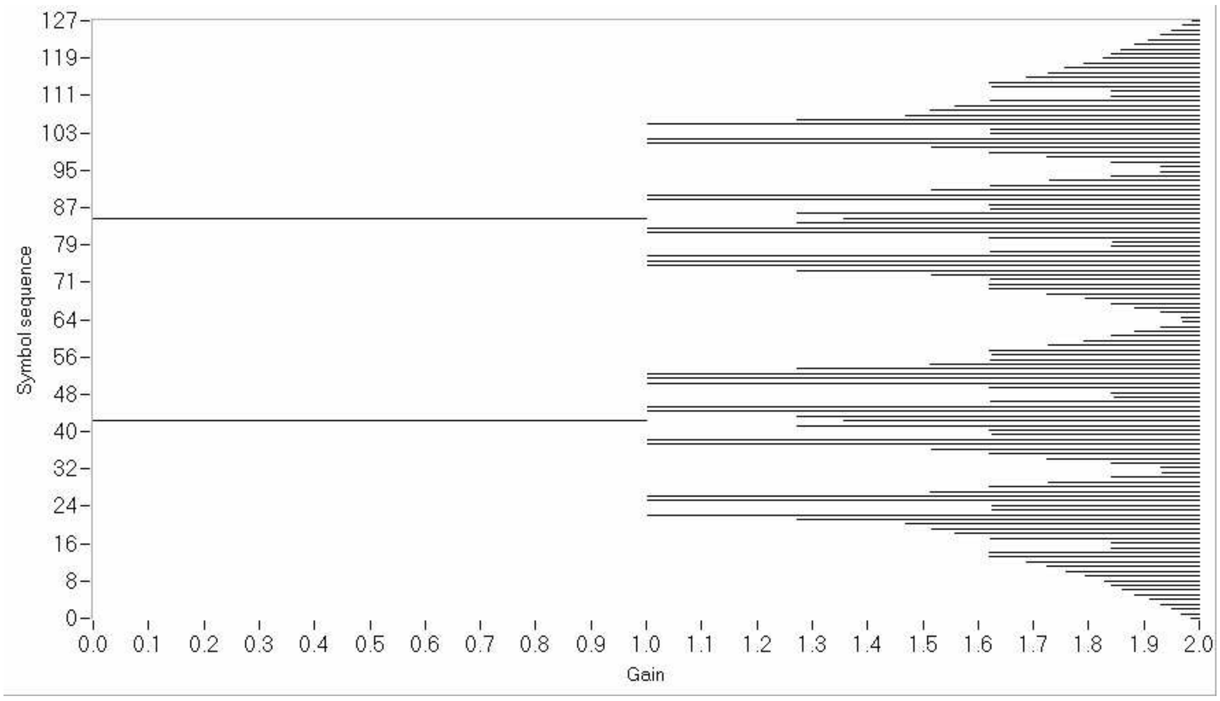


Figure 74. The permissible 7 bit sequences that can be generated using a first order, single bit sigma delta modulator.

Thus the bifurcation diagrams that were produced earlier are not illuminating because they say nothing about the range of quantized dynamics. For these reasons it is important to investigate the symbol sequences that can be generated for various values of α. Figure 74 depicts the seven bit symbol sequences that can be generated for a single bit sigma delta modulator with zero input (System 2). Seven bits were chosen simply for resolution- the qualitative structure of the resultant plot is the same for various choices of the number of bits. For gain ranging from 0 to 2 in increments of 0.001, 100,000 successive quantizer outputs were calculated. A sliding window of seven bits was applied to produce output sequences in the range 0000000 to 1111111 (0 to 127). A cyclic symbol sequence would be counted as multiple sequences, e.g., 0101010… and 1010101… are counted as separate allowable symbol sequences. This figure is exactly the same for both System 1 and System 2. This demonstrates that limit cycles are more dominant with α 2 since there is a smaller range of allowable dynamics.

5.3.5. Power Spectra

One of the key reasons to attempt sigma delta modulation in the chaotic regime is to see if it can effectively eliminate idle tones, while at the same time preserving the frequencies in the input signal. For this reason, the power spectrum is an appropriate tool.

In Figure 75, intensity plots are shown that reveal how the power spectrum is changed for gain from 0 to 2. Figure 75(a) depicts the power spectral intensity over the full range of gain and frequency for a 2 bit sigma delta modulator (System 4) with zero input. For $\alpha \le 1$, an idle tone exists with a frequency of 0.5. This is due to the quantizer flipping between ½ and -½. This tone is effectively removed in the chaotic regime. (b), (c) and (d) depict the power spectral intensity at 2x, 4x, and 8x magnification, respectively. They depict the self-similar, fractal nature of the power spectrum for $\alpha > 1$. This is another indication of chaos in sigma delta modulation.

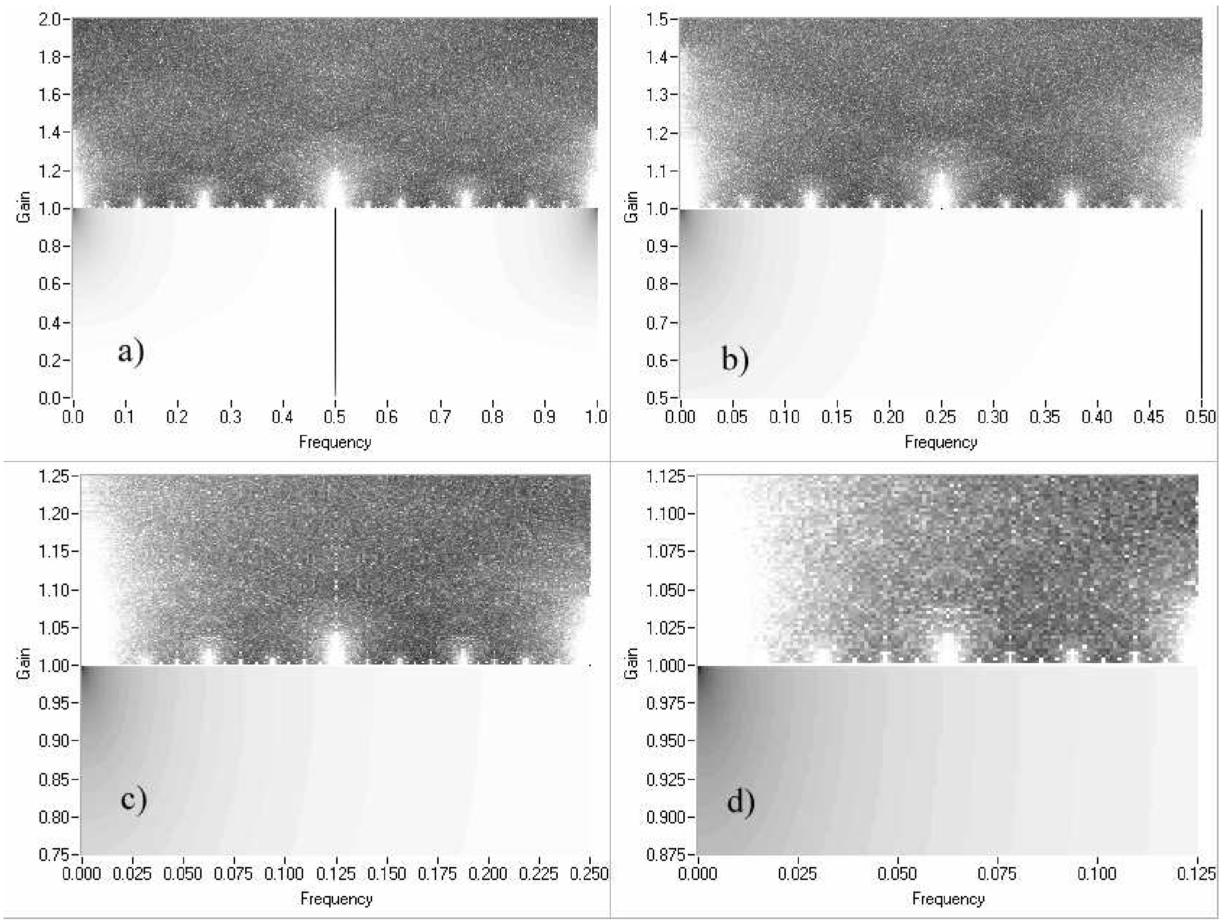


Figure 75. Intensity plots of power spectra a) is for a first order sigma delta modulator with constant zero input and gain ranging from 0 to 2.(b), (c) and (d) are successive magnifications (2x, 4x, and 8x) that indicate the fractal self-similar nature of the power spectrum.

In Figure 76, power spectra are depicted for an input signal with 32 times oversampling, $X_n = 0.5 \cdot \sin(2\pi \cdot n/64)$, applied to (System 4). Figure 76(a) depicts the power spectrum for the input. As expected, peaks are seen at frequencies of 1/64 and 63/64. However, for a 2 bit sigma delta modulator with unity gain, the output power spectrum exhibits additional peaks at all multiples of 1/64 (Figure 76(b)). In Figure 76(c), the modulator is operated at maximum gain (Equation (13)), $\alpha = 2$. The idle tones are completely removed, and replaced by chaotic fluctuations similar to broadband noise. This noise can be filtered, thus leaving only the frequencies that were apparent in the original signal.

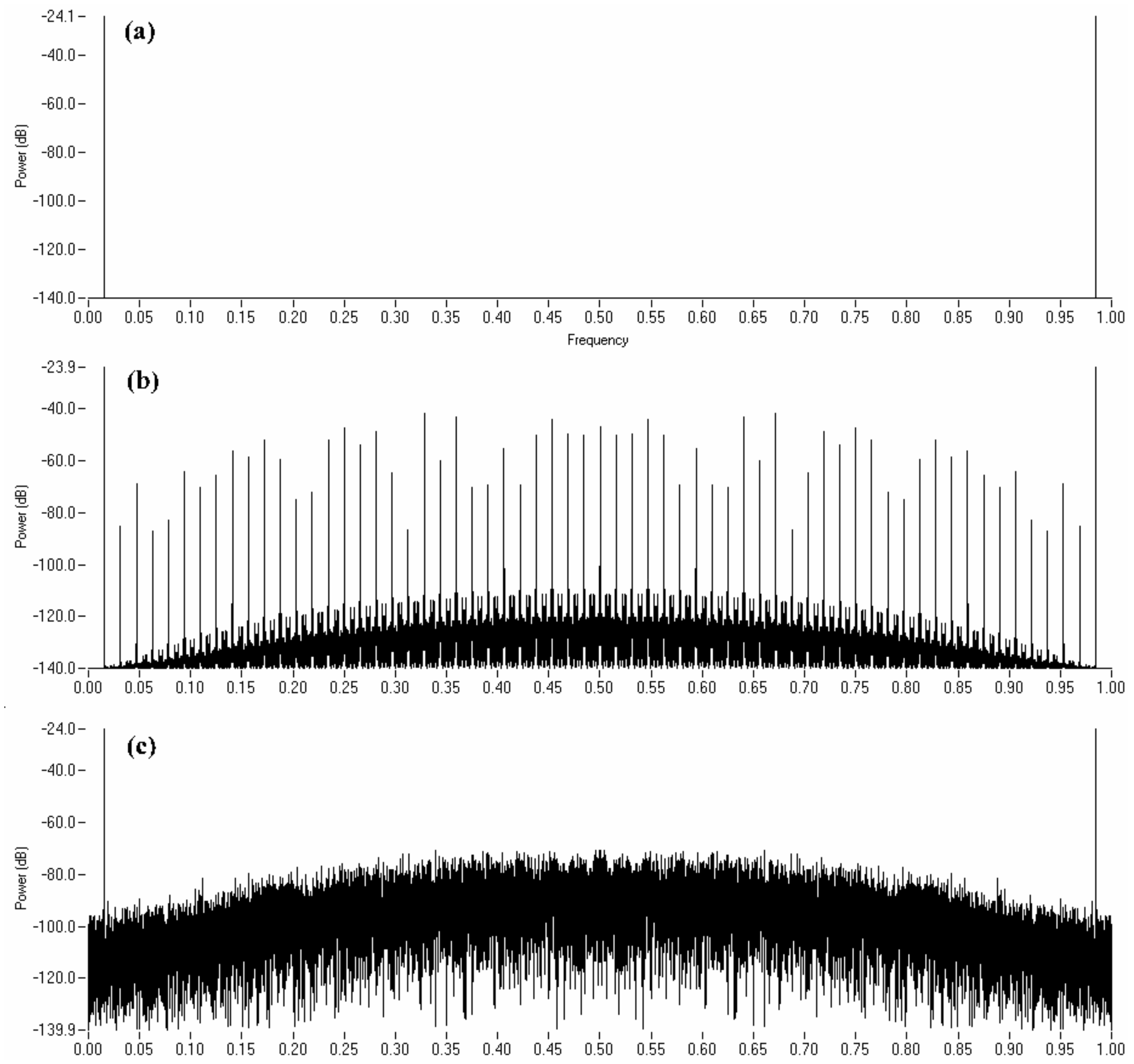


Figure 76. Power spectra for the input signal $X_n = 0.5 \cdot \sin(2\pi \cdot n/64)$. (a) is the power spectrum for the input signal, (b) is the power spectrum for the quantized output signal with gain set to 1, and (c) is the power spectrum for the quantized output signal with gain set to 2. The power is assumed to have a base value of 10^{-7} (-140dB).

5.4. Conclusions

A conventional first order sigma delta modulator, where gain is applied to the integrator output, does not approximate input for $\alpha \neq 1$. This is true even if multibit quantizers are used. The errors in quantization due to the Devil's staircase structure introduced by chaotic modulation can be compensated for by using a multibit quantizer. However, this does not correct the fact that the output is offset from the input for a traditional chaotic sigma delta modulator. If instead the gain is applied to the error in quantization, then the sigma delta modulator may achieve accurate quantization over a far greater range of input. If a multibit quantizer is also used, then the modulator can be made stable over the full range of input. This has the benefit that idle tones can be removed from the quantization process by operating in the chaotic regime.

CHAPTER SIX

CONCLUSION

Almost by definition, chaotic data is difficult to analyze. Its extreme sensitivity to initial conditions imply difficulty in reproducing data. Only the slightest change in initial conditions or the slightest noise may cause the system to enter a very different trajectory. The inherent nonlinearities imply that linear analysis techniques either fail or become meaningless. Also the likelihood of the system exhibiting broadband power spectra means that most digital signal processing techniques don’t produce meaningful results.

The techniques that are designed for the analysis of chaotic data also have inherent difficulties. Reliable fractal dimension estimation requires enormous amounts of relatively noise-free data. Lyapunov exponent calculations are highly susceptible to parameter settings of both the algorithm and of the embedding technique. These routines try to estimate properties that only converge to their actual values in the limit of infinite data sampled with infinite accuracy, something which no experiment can provide. Estimation of negative exponents becomes extremely difficult because these quantities give rise to exponentially small effects in the data.

In the combustion engine, the magnetoelastic ribbon and the electric step motor, there were additional difficulties. Primary amongst these was the nonstationarity. Figure 40, Figure 52, Figure 53, and Figure 59 all demonstrated that there were long term dynamics affecting these systems which could not be sufficiently captured in the data sets provided. These effects came from several sources: intrinsic dynamics, errors in the data acquisition system, and fluctuations due to environmental effects and other factors. This should not be modeled as noise. The drift had an effect on the dynamics that affected all analysis. Further analyses that were not presented in this thesis, such as noise reduction, prediction and periodic orbit identification, were similarly affected.

Yet some results were clear. Each of these systems exhibited complicated dynamics. They each appeared to have positive Lyapunov exponents, a strong indicator of chaos. The step motor and the magnetoelastic ribbon also seemed fairly low dimensional (fractal dimension less than 3), which implied that further analysis would be possible. The combustion engine was more noisy and the dynamics appeared complex. Thus the author would recommend a sophisticated approach to any control or prediction algorithm implemented on this system.

Certain analysis methods appeared quite robust. The mutual information routine produced reliable results on each system. The false nearest neighbors routine was also successful, once the criteria had been improved. The author also showed that Conley Index theory could be applied to experimental systems. This provided a means of extracting symbolic dynamics and a rigorous verification of chaos in the magnetoelastic ribbon experiment.

The use of efficient searching and sorting methods allowed the false nearest neighbors routine and the Lyapunov estimation routines to operate at a resonable speed. Thus detailed analysis became possible where otherwise time constraints would have made it infeasible. Improved searching speed has applications throughout the field of nonlinear time series analysis. Many prediction, noise reduction, Lyapunov spectra, dimension estimation, and information theoretic analysis routines rely on the investigation of near neighbors in a multidimensional space. An optimized search routine, such as the kd-tree, offers improved efficiency for all of these methods. In fact, wherever, multidimensional data needs to be searched, the benchmarking and analysis of the search methods may prove useful. In addition to the work presented in this thesis, the author has investigated the use of efficient multidimensional searching routines in the field of music information retrieval.[122]

Symbolic dynamics also played a strong role in the investigation of multibit chaotic sigma delta modulation. Figure 74 was the equivalent of a bifurcation diagram, except that it represented the allowable symbol dynamics in the output of the modulator as the gain was varied. This provided a clear indication of how to modify the gain in a chaotic modulator such that idle tones are either removed or drop below the noise level. Although chaotic sigma delta modulation has been investigated before, this research into multibit chaotic modulation is new. Previous investigations concluded that chaotic modulation was impractical because of its instabilities. However, Figure 70 showed that it was possible to operate a sigma delta modulator in the chaotic regime such that its output was stable throughout the entire range of operation. This encouraging result implies that there are potentially huge practical applications of chaos in the field of D-A and A-D converters. Further discussion of this work will be available in References 107, 123 and 124.

Continuation of the work on sigma delta modulation is clearly recommended. But much work should also be done in terms of investigating the dynamics of the combustion engine and the electric step motor. Both of these systems have practical uses, and both of them may benefit from a thorough understanding of their chaotic regimes. Controlled operation in the chaotic regime could yield improved efficiency in both systems.

The analysis methods also have room for improvement. A measure of reliability should be available for all dimension and exponent calculations. Although a method of determining errors in the false nearest neighbors routine was devised, this routine could benefit from a less arbitrary method of setting parameters. Furthermore, the interpretation of the multidimensional mutual information is uncertain. It is clear that it may be useful in the analysis of chaotic data, but it may also be useful in the analysis of a wide variety of multichannel systems. Its use as a measure of the shared information in multichannel sound systems has been suggested by the author.[125]

Finally, we note the benefits of providing all the analysis and visualization tools in one complete package. The Nonlinear Dynamics Toolbox is now a shareware software package that is used throughout the nonlinear dynamics community. Its efficiency and simplicity has allowed many researchers to use the techniques of chaotic time series analysis on their own data. Its continued development is a long term goal of the author.

BILBLIOGRAPHY

1 J. Graunt, *Natural and Political Observations Upon the Bills of Mortality* (Arno River Press, Salem, New Hampshire, 1662).

2 J. L. Klein, *Statistical Visions in Time* (Cambridge University Press, Cambridge, UK, 1997).

3 Gauss, in *Gauss's work (1803-1826) on the theory of least squares* (Mimeograph, Princeton, NJ, 1957), p. 1.

4 B. o. England, (Bank of England Archives, London, 1802).

5 S. M. Stigler, *The history of statistics: the measurement of uncertainty before 1900* (Belknap Press of Harvard University Press, Cambridge, MA, 1986).

6 J. H. Poynting, in *Collected Scientific Papers* (Cambridge University Press, Cambridge, 1920), p. 497.

7 W. S. Jevons, in *Investigations in Currency and Finance*, edited by H. S. Foxwell (Macmillan, London, 1884), p. 206.

8 B. van der Pol and J. van der Mark, Nature **120**, 363 (1927).

9 H. e. b. G. Poincare, D. L.), *New Methods of Celestial Mechanics* (American Institute of Physics, New York, 1993).

10 J. Gleick, *Chaos - Making a new science* (Viking, New York, 1987).

11 J. Norton, *Statistical studies in the New York money market* (Macmillan, New York, 1902).

12 F. E. Cave-Browne-Cave, Proceedings of the Royal Society of London **74**, 403 (1905).

13 L. March, Journal de la Societe de Statistique de Paris **46**, 255 (1905).

14 Hooker, Journal of the Royal Statistical Society **68**, 696 (1905).

15 O. Anderson, Journal of the Royal Statistical Society **90**, 548 (1927).

16 S. W. Gosset, Biometrica **10**, 179 (1914).

17 G. U. Yule, Journal of the Royal Statistical Society **89**, 1 (1926).

18 G. G. Stokes, Proceedings of the Royal Society **29**, 122 (1879).

19 A. Schuster, Philosophical Transactions , 69 (1906).

20 H. H. Turner, Monthly Notices of the Royal Astronomical Society , 714 (1913).

21 B. B. Hubbard, *The world according to wavelets* (A. K. Peters Ltd., Wellesley, MA, 1998).

22 A. N. Kolmogorov, *Foundations of the Theory of Probability* (Chelsea, New York, 1933).

23 A. A. Markov, Izvestija Fiziko-Mathematicheskoe Obshchestva pri Kazanskom Universitet **8**, 110 (1906).

24 C. E. Shannon and W. Weaver, *The mathematical theory of information* (University Press, Urbana Ill., 1949).

25 E. N. Lorenz, J. Atmos. Sci. **20**, 130 (1963).

26 J. P. Crutchfield, in *Univ. Calif. Sen. Thes., Sta. Cruz*, 1979).

27 J. D. Farmer, J. P. Crutchfield, H. Froehling N. H. Packard, *et al.*, **357**, 453 (1980).

28 N. H. Packard, J. P. Crutchfield, J. D. Farmer, *et al.*, Phys. Rev. Lett. **45**, 712 (1980).

29 R. S. Shaw, Z. Naturforsch. **36**, 80 (1981).

30 H. D. I. Abarbanel, *Analysis of observed chaotic data* (Springer, New York, 1996).

31 M. Ding, C. Grebogi, E. Ott, *et al.*, Phys. Rev. Lett. **70**, 3872 (1993).

32 U. Parlitz, Int. J. of Bifurcation and Chaos **2**, 155 (1992).

33 J.-P. Eckmann and D. Ruelle, Physica D **56**, 185 (1992).

34 J. A. Vastano and E. J. Kostelich, in *Dimensions and Entropies in Chaotic Systems*, edited by G. Mayer-Kress (Springer-Verlag, Berlin, 1986), p. 100.

35 R. Mané, in *Dynamical systems and turbulence*, edited by D. A. Rand and L. S. Young (Springer-Verlag, Berlin, 1981), Vol. 898, p. 230.

36 F. Takens, in *Dynamical Systems and Turbulence (Warwick 1980)*, edited by D. A. Rand and L.-S. Young (Springer-Verlag, Berlin, 1980), Vol. 898, p. 366.

37 J.-P. Eckmann and D. Ruelle, Rev. Mod. Phys. **57**, 617 (1985).

38 W. H. Press, S. T. Teukolsky, W. T. Vetterling, *et al.*, *Numerical Recipes in C: the art of scientific computing* (Cambridge University Press, Cambridge, England, 1992).

39 S. Chandra, Commun. Numer. Methods Eng. **13**, 739 (1997).

40 S. D. Cohen and A. C. Hindmarsh, (Lawrence Livermore National Laboratory, 1994).

41 S. Cohen and A. Hindmarsh, Computers in Physics **10**, 138 (1996).

42 A. M. Fraser and H. L. Swinney, Phys. Rev. A **33**, 1134 (1986).

43 P. Grassberger and I. Procaccia, Phys. Rev. Lett. **50**, 346 (1983).

44 P. Grassberger, Phys. Lett. A **128**, 369 (1988).

45 H. Kantz and T. Schreiber, , 1995), Vol. 5, p. 143.

46 M. B. Kennel, R. Brown, and H. D. I. Abarbanel, Phys. Rev. A **45**, 3403 (1992).

47 J. P. Crutchfield and N. H. Packard, Physica D **7**, 201 (1983).

48 P. Constantin and C. Foias, Commun. Pure Appl. Math. **38**, 1 (1985).

49 P. Frederickson, J. L. Kaplan, E. D. Yorke, *et al.*, J. Diff. Eqns. **49**, 185 (1983).

50 J. Theiler, S. Eubank, A. Longtin, *et al.*, Physica D **58**, 77 (1992).

51 H. Samet, *Applications of Spatial Data Structures* (Addison-Wesley, 1989).

52 H. Samet, *The design and analysis of spatial data structures* (Addison-Wesley, 1989).

53 H. Hinterberger, K. C. Sevcik, and J. Nievergelt, ACM Trans. On Database Systems **9**, 38 (1984).

54 N. Pippenger, R. Fagin, J. Nievergelt, *et al.*, ACM Trans. On Database Systems **4**, 315 (1979).

55 K. Mehlhorn, *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry* (Springer-Verlag, 1984).

56 F. P. Preparata and M. I. Shamos, *Computational geometry: An introduction* (Springer-Verlag, New York, 1985).

57 J. Orenstein, Information Processing Letters **14**, 150 (1982).

58 J. H. Bentley, Communications of the ACM **18**, 509 (1975).

59 J. H. Friedman, J. L. Bentley, and R. A. Finkel, ACM Trans. Math. Software **3**, 209 (1977).

60 J. L. Bentley, in *Sixth Annual ACM Symposium on Computational Geometry*, San Francisco, 1990), Vol. 91.

61 H. D. I. Abarbanel and M. B. Kennel, Phys. Rev. E **47**, 3057 (1993).

62 T. Schreiber, Int. J. of Bifurcation and Chaos **5**, 349 (1995).

63 R. F. Sproull, Algorithmica **6**, 579 (1991).

64 M. Hénon, Commun. Math. Phys. **50**, 69 (1976).

65 M.-C. Pera, B. Robert, and D. Guegan, in *5th Experimental Chaos Conference*, edited by W. Ditto and e. al. (World Scientific, Boca Raton, Florida, 1999).

66 T. Sauer, J. A. Yorke, and M. Casdagli, J. Stat. Phys. **65**, 579 (1991).

67 A. M. Fraser, Physica D **34**, 391 (1989).

68 D. S. Broomhead and G. P. King, Physica D **20**, 217 (1986).

69 R. Brown, in *Proceedings of the 1st Experimental Chaos Conference 1991*, edited by S. Vohra, M. Spano, M. Shlesinger, L. Pecora and W. Ditto (World Scientific, Singapore, 1992), p. 24.

70 R. Kappagantu and B. F. Feeny, J. Sound Vib. **224**, 863 (1999).

71 D. Vretenar, N. Paar, P. Ring, *et al.*, Phys Rev E` **60**, 308 (1999).

72 J. Jeong, D. J. Kim, J. H. Chae, *et al.*, Med. Engineering and Physics **20**, 669 (1998).

73 F. Sattin and E. Martines, Phys. Plasmas **6**, 100 (1999).

74 A. S. Soofi and L. Y. Cao, Econ. Lett. **62**, 175 (1999).

75 R. Hegger, H. Kantz, and T. Schreiber, Chaos **9**, 413 (1999).

76 T. Aittokallio, M. Gyllenberg, J. Hietarinta, *et al.*, Phys. Rev. E **60**, 416 (1999).

77 A. Wolf, J. B. Swift, H. L. Swinney, *et al.*, Physica D **16**, 285 (1985).

78 A. Wolf, in *Chaos*, edited by A. V. Holden (University Press, Manchester, 1986).

79 A. Wolf and J. A. Vastano, in *Dimensions and entropies in chaotic systems)*, edited by G. Mayer-Kress (Springer-Verlag, Berlin, 1986).

80 A. M. Fraser, IEEE Trans. Information Theory **35**, 245 (1989).

81 K. Mischaikow, M. Mrozek, J. Reiss, *et al.*, Phys. Rev. Lett. **82**, 1144 (1999).

82 J. Baker and J. Gollub, *Chaotic dynamics* (University Press, Cambridge, 1990).

83 D. P. Lathrop and E. J. Kostelich, Phys. Rev. A **40**, 4028 (1989).

84 K. Mischaikow and M. Mrozek, Bull. Amer. Math. Soc. (N.S.) 32 (1995).

85 K. Mischaikow and M. Mrozek, Math. of Computation **67**, 1023 (1998).

86 L. Arnold, C. Jones, K. Mischaikow, *et al.*, in *Dynamical Systems*, edited by R. Johnson (Springer, Montecatini Terme, 1994).

87 O. E. Roessler, Phys. Lett. A **57**, 397 (1976).

88 C. S. Daw, M. B. Kennel, C. E. A. Finney, *et al.*, Phys. Rev. E **57**, 2811 (1998).

89 G. A. Richards, G. J. Morris, D. W. Shaw, *et al.*, Combustion Science Technology **94**, 37 (1993).

90 C. S. Daw, J. F. Thomas, G. A. Richards, *et al.*, Chaos **5**, 662 (1995).

91 V. In, M. Spano, J. Neff, *et al.*, Chaos **7**, 605 (1997).

92 M. A. Rhode, R. W. Rollins, and A. J. et al. Markworth, J. Appl. Phys. **78**, 2224 (1995).

93 P. Grassberger and I. Procaccia, Physica D **9**, 189 (1983).

94 P. Grassberger and I. Procaccia, Phys. Rev. A **28**, 2591 (1983).

95 P. Grassberger and I. Procaccia, Physica D **13**, 34 (1984).

96 J.-P. Eckmann, S. O. Kamphorst, D. Ruelle, *et al.*, Phys. Rev. A **34**, 4971 (1986).

97 H. T. Savage, W. L. Ditto, P. A. Braza, *et al.*, J. Appl. Phys. **67**, 5919 (1990).

98 F. C. Moon and P. J. Holmes, J. Sound Vib. **65**, 285 (1979).

99 W. L. Ditto, S. N. Rauseo, and M. L. Spano, Phys. Rev. Lett. **65**, 3211 (1990).

100 J. F. Heagy and W. L. Ditto, J. Nonlinear. Sci. **1**, 423 (1992).

101 W. L. Ditto, M. L. Spano, H. T. Savage, *et al.*, Phys. Rev. Lett. **65**, 533 (1990).

102 J. C. Sommerer, in *Proc. of the 1st experiment. Chaos Conf. Arlington V., Oct. 1-3, 1991*, edited by S. Vohra, M. Spano, M. Schlesinger, L. Pecora and W. Ditto (World Scientific, Singapore, 1992), p. 269.

103 J. C. Sommerer, W. L. Ditto, C. Grebogi, *et al.*, Phys. Rev. Lett. **66**, 1947 (1991).

104 M. L. Spano, M. Wun-Folge, and W. L. Ditto, Phys. Rev. A **46**, 5253 (1992).

105 W. L. Ditto, S. Rauseo, R. Cawley, *et al.*, Phys. Rev. Lett. **63**, 923 (1989).

106 M. C. Pera, B. Robert, and C. Goeldel, in *8th European Conference on Power Electronics and Applications*, Lausanne, Switzerland, 1999).

107 J. Reiss and M. B. Sandler, CHAOS **11** (2001).

108 W. Chou and R. M. Gray, IEEE Transactions on Information Theory **37**, 500 (1991).

109 V. Friedman, IEEE Trans. Communication **36**, 972 (1988).

110 A. J. Magrath and M. B. Sandler, Electronics Letters **31** (1995).

111 A. J. Magrath and M. B. Sandler, in *IEEE International Symposium on Circuits and Systems*, 1996).

112 S. J. Park, R. M. Gray, and W. Chou, in *Proceedings of the ICASSP 91*, Toronto, Canada, 1991), p. 1957.

113 O. Feely, Int. J. Circuit Theory and Applications **25**, 347 (1997).

114 D. T. Hughes, S. McLaughlin, G. Ushaw, *et al.*, in *Chaos in Communications*, edited by L. M. Pecora (SPIE-The International Society for Optical Engineering, Bellingham, Washington, 98227-0010, USA, 1993), p. 194.

115 O. Feely, IEEE Circuits and Systems Society Newsletter **11**, 1 (2000).

116 R. M. Gray, IEEE Transactions on Communications **COM-35**, 481 (1987).

117 L. Risbo, (Technical University of Denmark, 1994).

118 C. Dunn and M. B. Sandler, Journal of the AES **44**, 227 (1995).

119 R. Schreier, IEEE Transactions on Circuits and Systems **147**, 539 (1994).

120 O. Feely and L. O. Chua, IEEE Trans. Circuits and Syst. **CAS-38**, 1293 (1991).

121 O. Feely and L. O. Chua, Int. J. of Circ. Theory and Appl. **21**, 61 (1993).

122 J. D. Reiss, J.-J. Aucoutier, and M. B. Sandler, in *2nd Annual International Symposium on Music Information Retrieval*, Bloomington, Indiana, USA (2001).

123 J. Reiss and M. B. Sandler, in *European Conference on Circuit Theory and Design*, Espoo, Finland, (2001).

124 J. Reiss and M. B. Sandler, in *Nonlinear Dynamics of Electronic Systems*, Delft, The Netherlands, (2001).

125 J. D. Reiss, N. Mitianoudis, and M. B. Sandler, in *110th Convention of the Audio Engineering Society*, Amsterdam, The Netherlands, (2001).