NONLINEAR ANALYSIS OF COMPLICATED PHYSICAL SYSTEMS

By

Andrew Jason Penner

A Thesis submitted to
the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Physics
University of Manitoba
Winnipeg, Manitoba
2004

UNIVERSITY OF MANITOBA
DEPARTMENT OF
PHYSICS AND ASTRONOMY

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Nonlinear Analysis of Complicated Physical Systems**" by **Andrew Jason Penner** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 2004

Supervisor: _____
Randal Kobes

Readers: _____
Tom Osborn

_____
Richard Gordon

UNIVERSITY OF MANITOBA

Date: **2004**

Author: **Andrew Jason Penner**

Title: **Nonlinear Analysis of Complicated Physical Systems**

Department: **Physics and Astronomy**

Degree: **M.Sc.** Convocation: **October** Year: **2004**

Table of Contents

Table of Contents iv

List of Tables viii

List of Figures ix

Abstract i

Acknowledgements iii

Introduction 1

1 Nonlinear Dynamics 5
- 1.1 Introduction . . . 5
  - 1.1.1 Attractor . . . 6
- 1.2 The System At Hand . . . 10
  - 1.2.1 Manifolds . . . 13
  - 1.2.2 Kneading . . . 14
  - 1.2.3 Lyapunov Exponent . . . 15
  - 1.2.4 Winding Number . . . 22
  - 1.2.5 The Circle Map . . . 23
- 1.3 Poincaré Section . . . 26
- 1.4 Fixed Points . . . 28
- 1.5 Bifurcation . . . 30
- 1.6 Fractals . . . 39

2 Symbolic Dynamics 43
- 2.1 Introduction . . . 43
- 2.2 Partitions . . . 46

2.2.1 Pruning . . . 48
2.3 1D Approximation . . . 53
2.4 Admissibility . . . 55
2.5 Prime Cycles . . . 56
2.6 Numerical Orbit Hunting . . . 57
2.7 Conclusion . . . 62

3 Periodic Orbit Theory **66**
3.1 Introduction . . . 66
3.1.1 Evolution . . . 66
3.1.2 Trajectory Displacements . . . 67
3.2 Nonlinear Flows . . . 68
3.3 Periodic Orbit Calculations . . . 71
3.4 Statistics . . . 74
3.4.1 Hyperbolicity & Trace . . . 77
3.4.2 Lyapunov exponents . . . 80
3.4.3 Escape Rate . . . 81
3.4.4 Measure . . . 83
3.5 Cycles . . . 83
3.5.1 Zeta . . . 83
3.5.2 Cycle Expansion . . . 85
3.5.3 Pseudocycles . . . 87
3.6 Stability Ordering . . . 89
3.6.1 Ordering . . . 89
3.6.2 Smoothing . . . 90
3.6.3 Cutoff . . . 91
3.7 Conclusion . . . 93

4 Computation and Calculation **94**
4.1 Introduction . . . 94
4.2 Finding The Periodic Points . . . 94
4.2.1 Automation . . . 97
4.3 Eigenvalues . . . 99
4.4 Winding Number . . . 100
4.5 Pseudocycles . . . 101
4.6 Results . . . 102
4.7 Conclusion . . . 109

5 Fractals **111**
5.1 Introduction . . . 111
5.1.1 Dimension . . . 113
5.2 Fractal Classes . . . 116
5.2.1 Fractal Dimension & Stability Eigenvalues . . . 119
5.2.2 Fat Fractals . . . 120
5.2.3 MultiFractals . . . 120
5.3 Iterated Function Systems . . . 122
5.3.1 Contraction Mapping Principle . . . 124
5.3.2 The Hutchinson Operator . . . 124
5.3.3 Fundamental Theorem of Iterated Function Systems . . . 125
5.4 Fractal Interpolation . . . 136
5.5 Conclusion . . . 142

6 Iterated Interpolation Function Systems **144**
6.1 Introduction . . . 144
6.2 Linear Two Dimensional Interpolation Functions . . . 145
6.2.1 Simplify . . . 147
6.3 Quadratic Two Dimensional Interpolation Functions . . . 149
6.4 Partial Quadratic Two Dimensional Interpolation Functions . . . 149
6.5 Higher Degree Interpolation Techniques . . . 153
6.6 Potential Applications . . . 154
6.7 Conclusion . . . 161

7 Conclusions **162**
7.1 Symbolic Dynamics . . . 162
7.2 Numerical Periodic Orbit Methods . . . 162
7.3 Numerical Periodic Orbits . . . 163
7.4 Fractals . . . 164
7.5 Image Manipulation . . . 164
7.6 Future Projects . . . 164
7.7 Final Remarks . . . 166

A Periodic Orbit Hunting Code **168**

B Iterated Function System Code **181**
B.1 Draw.c . . . 181
B.2 fractalResized.h . . . 182
B.3 gdIFSCopyResized.c . . . 183

B.4 gdIFSCopyResized.h 191

Bibliography **193**

List of Tables

2.1 Allowed Orbits for Q = 0.76 63
2.2 Allowed Orbits for Q = 1.2577 64
2.3 Additional Allowed Orbits for Q = 1.2577 65

5.1 Fractal Dimensions of Various Fractals 114
5.2 IFS code for Barnsley's Fern [43] 128

List of Figures

1.1 Point Attractor 8
1.2 Limit Cycle 8
1.3 Our System 10
1.4 Logistic Kneading Sequence 15
1.5 Logistic Lyapunov Spectrum 19
1.6 Winding Number 23
1.7 Full Attractor 27
1.8 Simple Cross Section 28
1.9 Poincaré Section 29
1.10 The Logistic Diagram 29
1.11 Higher Return Maps 30
1.12 First return map for Q=0.76 31
1.13 Second return map for Q=0.76 31
1.14 Logistic Bifurcation Diagram 32
1.15 Logistic Bifurcation Diagram zoomed within regions $\mu = 3.8 \rightarrow 3.9$. 34
1.16 Logistic Bifurcation Diagram zoom within regions $\mu = 3.7 \rightarrow 3.75$ . . 35
1.17 The First Feigenbaum Constant 36
1.18 The Second Feigenbaum Constant 36
1.19 Our Bifurcation 37
1.20 Q=0.76 Bifurcation 38
1.21 Q=1.2577 Bifurcation 39
1.22 Logistic Box Counting - 4 41

1.23 Logistic Box Counting - 16 41
1.24 Logistic Box Counting - 64 41

2.1 Symbolic Coding of the Logistic Diagram 47
2.2 Symbol Plane, Q = 0.76 51
2.3 Symbol Plane, Q = 1.2577 51

3.1 The Smoothing Functions 92

4.1 Escape Rate for Table 2.1 1 104
4.2 Escape Rate for Table 2.1 2 104
4.3 Escape Rate for Table 2.1 3 104
4.4 Escape Rate for Table 2.2 1 105
4.5 Escape Rate for Table 2.2 2 105
4.6 Escape Rate for Table 2.2 3 105
4.7 Observables for Table 2.1 Stability Ordering 106
4.8 Observables for Table 2.2 Stability Ordering 107
4.9 Observables for Table 2.1 Period Ordering 107
4.10 Observables for Table 2.2 Period Ordering 107

5.1 A Sample Julia Set 111
5.2 Self Affine Fractal 116
5.3 Self Similar Fractal 117
5.4 The Cantor Set 118
5.5 The Koch Curve 118
5.6 The Henon Map 121
5.7 The Zoomed Henon Map 121
5.8 The Fractal Tree 122
5.9 The Cantor Square 128
5.10 Barnsley's Fractal Fern 129
5.11 The first five iterates of the Sierpinski probability square 131

5.12 A Canadian Maple Leaf 143
5.13 A Good Reconstruction of the Maple Leaf 143
5.14 A Poor Reconstruction of the Maple Leaf 143

6.1 Fractal Reconstruction of Dilbert 158
6.2 Linear Reconstruction of Dilbert 158
6.3 Fractal Reconstruction of Text 158
6.4 Linear Reconstruction of Text 158
6.5 4x zoom Dilbert reconstruction using linear fractal interpolation . . . 159
6.6 4x zoom Dilbert reconstruction using linear interpolation 159
6.7 4x zoom text reconstruction using linear fractal interpolation 160
6.8 4x zoom text reconstruction using linear interpolation 160
6.9 The quadratic fractal reconstruction of entire image 160
6.10 The linear reconstruction of entire image 160
6.11 The linear fractal reconstruction of entire image 160
6.12 The quadratic fractal magnification around the face 160
6.13 The linear magnification around the face 160
6.14 The linear fractal magnification around the face 160
6.15 The quadratic fractal magnification around the eye 161
6.16 The linear magnification around the eye 161
6.17 The linear fractal magnification around the eye 161

Abstract

Chaos is fundamental to nature. Perhaps the most illustrative examples are atmospheric processes, but even further than that, some systems that had been commonly thought to be periodic, such as planetary motion, were recently proven to be chaotic. Chaos refers to a deterministic behavior characterized by a high sensitivity to a change of initial conditions. Due to these qualities any long term predictions are impossible, and consequently any solution of a given initial condition problem is not a physical observable. Calculating expectation values of observables for chaotic systems is usually done numerically and often marred by numerical artifacts. In this presentation we investigate a more subtle approach for evaluating expectation values for a chaotic system. We demonstrate our results in the example of a driven pendulum. Chaotic behavior may be thought of as motion over an infinite number of periodic orbits. A feasible mathematical solution to determine the expectation values is through cycle expansion, where an expectation value is calculated as a statistical average over periodic orbits in phase space. Statistical weight of each orbit is determined by its stability. Ultimately our goal was two fold. First we were to find the periodic orbits

that contributed the most to the averages. Second, through use of these relatively few periodic orbits, we were to estimate expectation values for the rotation number and Lyapunov exponent of the driven pendulum to a high level of accuracy, and compare them to the brute force numerical calculations. The second instance of nonlinear theory is the study is iterated function systems (IFS). These are methods of image compression, and manipulation. We show the usefulness of this method as an image manipulator specifically with respect to magnification properties, by comparing it to a standard method of digital zoom. We also discuss the possible applications of the fractal interpolation with respect to image compression.

Acknowledgements

To be fair there are far too many people that should be acknowledged here, as I had a lot of support from people in several faculties and departments in the University of Manitoba. Since space is limited, I will forced to cut the list short. My apologies to those who have been omitted, your support will never be forgotten.

I would like to thank my supervisor, Dr. Randy Kobes, for all of his support and advice through out my time as a Masters Student. His advice was time saving, and very straight forward. Without which I would not have been able to determine my future path in research.

I would also like to thank Slaven Peles, for his guidance and friendship through this project. His wisdom and patience along the way was a tremendous asset towards completing the project.

I am thankful to my girlfriend, Heather Matheson for her tolerance of my efforts in this project. Without her consistently useful programming ideas, and strong moral support, this project would have taken significantly longer than necessary.

I am also thankful to my family for their unending support for my goals. They were there in the beginning, they are still here now, and I know they will be there at the end.

Tom Osborn, who introduced me to the many interesting aspects of mathematical physics, especially towards relativity and quantum mechanics.

Richard Gordon, who has shed light on the topic of biological physics, and the necessity for people to investigate this area of science for a complete understanding of the natural sciences.

Winnipeg, Manitoba Andrew Jason Penner
June 28th , 2004

Introduction

Chaos occurs quite frequently in nature. In fact, by studying the chaos in systems we are studying a more accurate representation of the reality behind nature. Traditional physical analysis takes on the form of investigating real systems by comparing them to perfect devices, such as the simple harmonic oscillator, with the use of first order Taylor series expansions. It is not uncommon for these systems to be close to correct for some very specific applications. However, for a true understanding of nature we must be willing to leave the perfect system comparisons and look towards real systems. This forces us to also leave behind the approximations that are only true for special cases. A common example is the simple pendulum. As it oscillates with small angles the behaviour of the system is well understood. However as we push the system to act a little more extreme, the larger angles of rotation lead to some rather interesting dynamics. If we were to go further we could add a damping term and a driving force, making the system more and more general, such that the model may be applied to real world situations more readily. However, as we increase generality of a system integrability is usually lost.

In the cases where the linear approximations do not produce experimentally viable results, we allow the linear expansion but add a term that "kicks" the system by introducing an external energy so that it matches experimental results. This method of analysis is referred to as perturbation theory. It brings us even closer to the truth, without the difficulty of re-defining the dynamical equations governing the system, or without the need for numerical analysis. Again, this method does not always work, some systems cannot be treated as perfect and also may not be approximated through a perturbation. At this point the researcher must analyze the system as it is. Often, systems that we are interested in cannot be solved analytically, and thus must be solved numerically. [35]

The evolution of science over the past few decades has lead to some rather interesting nonlinear phenomenon, including deterministic chaos. A deterministic system is one where, given the initial conditions, we know the final state for all time. [52] With deterministic chaos, a dynamical system becomes highly dependent on the initial conditions. That is, the slightest change in the initial conditions produces an exponentially divergent set of solutions. This leaves a reasonably precise prediction of the future evolution of the system to a very short period of time. With linear deterministic systems one generally expects a single solution, or a single orbit solution. However, with the nonlinear systems one must study a distribution of orbits, thus leading us to a statistical analysis of nonlinear systems. All of this is the main focus

of the first few chapters of this paper.

Another facet of chaos is the formation of fractals, which are geometric shapes. These have been investigated over several decades and have come into industrial use, through image manipulation. This use was also investigated and new techniques have been established.

The breakdown of the thesis goes as follows; Chapter 1 will discuss the major features of nonlinear systems, such as Lyapunov exponents, and fractals. Such properties are inherent to chaotic systems and often lead the researcher to the existence of chaos. It will also introduce the system we are investigating and the mechanics involved. Chapter 2 will discuss symbolic dynamics and the methods used for orbit hunting. Chapter 3 will discuss the methods used to analyze chaos such as periodic orbit theory. Chapter 4 will discuss how the methods in chapter 3 were implemented and the results obtained in our study. Chapter 5 will discuss Fractals in deeper detail, as well as introducing iterated function systems. Chapter 6 will look at the iterated function systems, both existing methods and future avenues of approach, as well as go into detail on fractal interpolation, from linear to a generalized higher order interpolation, as well as extending the theory to higher dimensions. Chapter 7 will give concluding remarks, and mention possible future projects that revolve around the major ideas expressed in the thesis.

Appendix I contains C code used in the periodic orbit theory. Appendix II contains

C code used for the interpolation IFS theory.

Chapter 1

Nonlinear Dynamics

1.1 Introduction

As mentioned earlier, nature is inherently nonlinear. The perfection regarded through physical analysis are not entirely accurate, at least not for all cases. So when we do analyze a dynamical system there is often information that is left out or ignored. That is not to say that the approximations that are made do not work, they are just not always applicable. (Such is the case in quantum scattering, where difficulties in analysis are reduced to simpler recursive calculations.) When studying dynamical systems one is left with a system of differential equations which may not reduce to an analytical solution. Systems with this type of solution were often neglected due to the need for intensive numerical solutions. Even when these solution techniques were used, the results were often mistaken for noise. Now, with the invention of faster computers these systems are more reasonable to solve, and may be done readily with already existing software.

Before investigating the nonlinear features of any system one must become accustomed to the different maps and features of a chaotic system. Originally there were three topological criteria that had to be satisfied by a chaotic system. These criteria were developed by Robert Devaney, and are:

- (a) The dynamical system has sensitive dependence on initial conditions. The slightest change in initial conditions will produce dramatically different results after a finite amount of time.
- (b) Periodic points for the system are topologically dense. As the trajectory evolves it comes arbitrarily close to a previous point in the trajectory.
- (c) The dynamical system is topologically transitive (space filling).

[18, 36] Although the above criterion are general it has been found that there are cross relations within the list such that one item leads to another. However, tests for each of these criteria may be done individually. The tests will be summarized in this chapter and many of them will be used throughout the thesis.

1.1.1 Attractor

When we are dealing with a dynamical system, we are dealing with a flow, $\mathbf{F}$, which is a continuous set of equations governing the motion of the system through time. There are two possible classifications for these systems, conservative and dissipative. To determine the classification we simply take the divergence of the flow, $\nabla \cdot \mathbf{F}$. If

this value is less than zero we have a dissipative system. If this value is greater than zero we have a divergent system. A bounded divergent system is known as chaotic. Through the analysis of a bounded divergent system, we find a set that the system naturally tends to, after a finite transient period.

An attractor is a set of points that any finite set of given initial conditions will converge to. These are points of stability, and are characteristic of a divergent trajectory. Technically, these are bounded subsets to which regions of initial conditions of nonzero phase space volume asymptote as time increases. But in plain language, this says that after a finite amount of time any nonstable initial condition will converge to these points. The most common example of an attractor comes from the driven damped harmonic oscillator, where different parameters lead to different attractors. The equation to be modelled is as follows;

$$\ddot{\theta} + \gamma\dot{\theta} + \sin\theta + F\sin\omega_D t = 0 \tag{1.1.1}$$

The attractors in this system can be in two forms. They can be a point attractor or they can be a limit cycle where the orbit eventually converges to an orbit or cycle around some point in phase space. The point attractor is generally a trait of a dissipative system, since these systems are bound to lose energy until a ground state, the point of attraction, is reached. The limit cycle is a trait of driven systems, as the dissipation due to frictional forces may be matched with the driving force. The natural attractor of chaotic systems are strange attractors, which are attractors

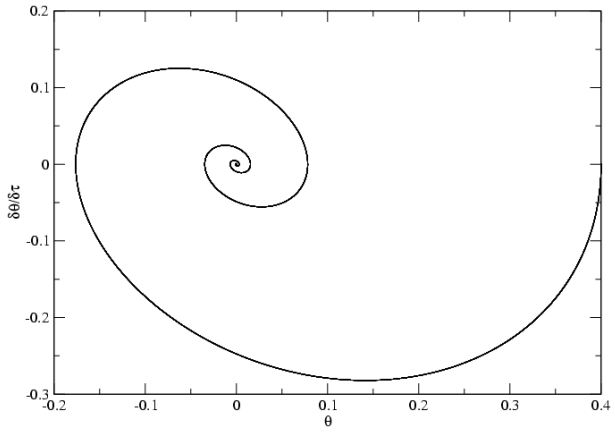

Figure 1.1: The damped pendulum point attractor

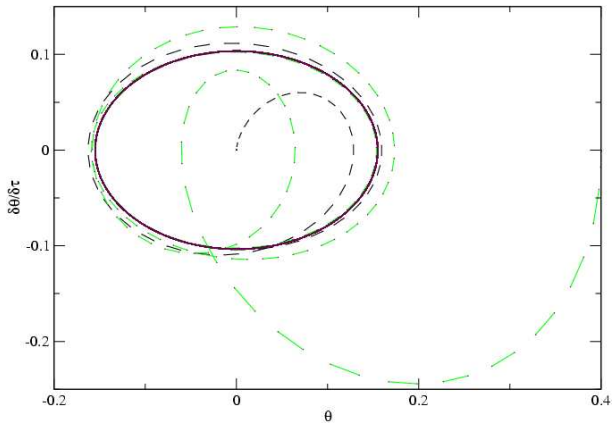

Figure 1.2: Driven damped pendulum limit cycle attractor

that have fractal dimension. These will be described later in this chapter. This type of attractor is separated from others by a few characteristics. The attractor is nowhere repeating, it is topologically dense, and transient. To claim the attractor is topologically dense is to say that the points that comprise the attractor come infinitesimally close to each other but are never exactly the same point. This is because, if the points were permitted to be the same, the entire attractor would end up bound in an exotic limit cycle, and the chaos would be lost. A common example of a strange attractor is the Hénon map as seen in references [48, 52, 24, 57, 68]

Possibly the most common attractor is the Logistic Map. This was originally a function produced to model population growth in a biological system. The intent was to model the growth and decline of a species over time with the constraints of limited food resources. They follow the model

$$x_{n+1} = \mu x_n(1 - x_n) \tag{1.1.2}$$

with μ being the rate of growth of the species. This is introduced here like the chaotic pendulum was introduced. The investigations into these systems are rather extensive in just about any book on chaos, so these will be used as examples throughout this thesis.

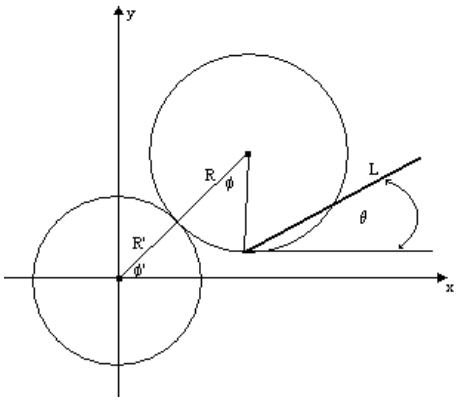

Figure 1.3: The Two Gears and a Rod Pendulum, R' denotes the planetary gear

1.2 The System At Hand

The particular system we have been studying consists of two gears and a rod, as pictured in figure 1.3. The system is held horizontal thus we may neglect the effects of gravity. The first gear is held in a fixed location and is not allowed to rotate. The center of this gear is the origin of our coordinate system. The second gear is free to rotate around the first gear at a constant frequency. The final component is the homogeneous rod, attached to the boundary of the second gear at a fixed point. The rod has full rotational freedom. The dynamics of this system may be easily studied using Lagrangian mechanics. With;

$$\begin{aligned} x &= (R' + R)\cos\phi' - R\cos(\phi' + \phi) + l\cos\vartheta \\ y &= (R' + R)\sin\phi' - R\sin(\phi' + \phi) + l\sin\vartheta \end{aligned} \tag{1.2.1}$$

where we can take the relationship between the prime and unprimed coordinates is $R'\phi' = R\phi$. We consider the kinetic energy of the system to integrate over the mass

of the system, so we get

$$T = \int \dot{x}^2 + \dot{y}^2 dm \tag{1.2.2}$$

The vertical system does not experience potential energy so the Lagrangian consists solely of kinetic energy, thus one can write the Lagrangian equation as:

$$L = \frac{1}{2}I\dot{\vartheta}^2 + \frac{1}{2}(1+r)mlR\dot{\phi}\dot{\vartheta}\{\cos(\vartheta - r\phi) - \cos(\vartheta - (1+r)\phi)\} + 2mR^2(1+r)^2\dot{\phi}^2(1-\cos(\phi)) \tag{1.2.3}$$

where I is the inertia of the rod, and the gear ratio $r = R/R'$. This leads to the 2 equations of motion:

$$I\ddot{\vartheta} + \frac{1+r}{2}mlR\omega_d^2\{r\sin(\vartheta - r\phi) - (1+r)\sin(\vartheta - (1+r)\phi)\} = 0 \tag{1.2.4}$$

$$\ddot{\phi}(1 - \cos\phi) = 0 \tag{1.2.5}$$

However, the assumption used for the solution of our system is that the angular velocity of the moving gear is constant, that is to say $\dot{\phi} = \omega_d$ or $\phi = \omega_d t$. This simplifies our Lagrangian to

$$L = \frac{1}{2}I\dot{\vartheta}^2 + \frac{1}{2}(1+r)mlR\omega_d\dot{\vartheta}\{\cos(\vartheta - r\phi) - \cos(\vartheta - (1+r)\phi)\} \tag{1.2.6}$$

where we removed the last term of the Lagrangian because it was equal to a total time derivative, after our substitution. Now our equation of motion becomes

$$I\ddot{\vartheta} + \frac{1+r}{2}mlR\omega_d^2\{r\sin(\vartheta - r\phi) - (1+r)\sin(\vartheta - (1+r)\phi)\} = 0 \tag{1.2.7}$$

the second equation of motion no longer exists since we lost a degree of freedom by forcing the angular velocity to be constant.

Now, we must consider frictional forces that are experienced by real systems. The effect this has on the equations of motion is to add a term to the right hand side, $Q_{\dot{\vartheta}}$ where the subscript indicates a frictional force that depends on the velocity term. So we add $Q_{\dot{\vartheta}} = -\eta\dot{\vartheta}$ to the RHS. This leads us to:

$$\ddot{\vartheta} + \frac{\eta}{I}\dot{\vartheta} + \omega_o^2 \left\{ \sin(\vartheta - r\phi) - \frac{(1+r)}{r}\sin(\vartheta - (1+r)\phi) \right\} = 0 \tag{1.2.8}$$

where we define the natural frequency of the system to be:

$$\omega_o^2 = r(1+r)\frac{mlR}{2I}\omega_d^2 \tag{1.2.9}$$

Now, after some more simplifying substitutions and a change of variable, we are left with a dimensionless equation:

$$\ddot{\theta} + \frac{1}{Q}\dot{\theta} + \sin(\theta) = -\frac{ar}{Q} + \frac{1+r}{r}\sin(\theta - \phi) \tag{1.2.10}$$

where

$$a = \frac{\omega_d}{\omega_o} = \sqrt{\frac{2L}{3r(1+r)R}}, \quad Q = \frac{\omega_d}{\eta}\sqrt{r\frac{1+r}{2}mlRI}, \quad \phi = \omega_d t = a\tau \tag{1.2.11}$$

The equation is no longer explicitly time dependent. This becomes very important for ease of analysis later, since we will want to break the differential equation into a 3-Flow system:

$$\begin{gathered} \tfrac{dx}{dt} = y(t) \\ \tfrac{dy}{dt} = -\tfrac{y(t)}{Q} - \sin(2\pi x(t)) - \tfrac{ar}{Q} + \tfrac{1+r}{r}\sin(2\pi x(t) - z(t)) \\ \tfrac{dz}{dt} = a \end{gathered} \tag{1.2.12}$$

These sets of equations are the focus of much of this thesis. We now turn to a global feature of a dynamical system.

1.2.1 Manifolds

Manifolds are the solution space for any dynamical system. This is an especially significant concept for chaotic systems. Analysis of the topology of manifolds is an important concept for symbolic dynamics, since symbolic dynamics utilizes topological structures and symmetries. Defining the topological structure that the chaotic system exists in is often a fairly easy task. Based on the equation of motion we have a manifold $\mathbb{S}^1 \times \mathbb{S}^1$ which comes from two degrees of rotational freedom, the planetary gear and the rod, as well as the equation forcing the system to be autonomous. We therefore have a smooth manifold for the third degree of freedom, and with that comes symmetry. This knowledge allows us to better analyze the system before applying any numerical techniques. An example of such a form of analysis is symbolic sequencing, to be described in chapter 2. Concepts such as the winding number, seen in section 1.2.4, also depend on the type of manifold of a system, since by its very definition, it requires a closed set. Furthermore, the Poincare section actually restricts the manifold investigated to $\mathbb{S}^1$. With this system defined we now need to concentrate on the defining features of chaos.

1.2.2 Kneading

Folding is a phenomenon experienced by bounded systems. Basically, it is a return within phase space, and it allows a trajectory to be periodic. Folding prevents a system from having a range which is greater than the domain. Many textbooks on the matter describe a folding system in terms of a Smale map, or a horseshoe map. This definition is suitable for the topologists that study in this field, however on a more practical level, folding is best described in Kinsner [36] where an example of the logistic map folding is demonstrated. Basically, what we may consider the map as given in equation 1.1.2. The diagonal line $y = x$ on the map is the substitution line, since it is the points along this line that are swapped for the new position x_n. This undergoes folding since the map is bounded on the unit line, and any point within the unit line when compared to the substitution line will result in another point within the unit line. When folding is also accompanied with a method known as stretching, where one takes an interval of the unit line, and determines how the resulting line will appear after the first iteration on the map. Kneading is the formal name for the process of folding along with stretching. Kneading may be seen in Figure 1.4 where $r = 4$. What is seen in Figure 1.4 is that a sample interval [0.2,0.4] after one iteration becomes the interval [0.64,0.96] which has a larger range. So this map has experienced folding as well as stretching. If we were to continue this process we would find that the interval [0.2,0.4] eventually covers the entire unit line. This is true regardless of

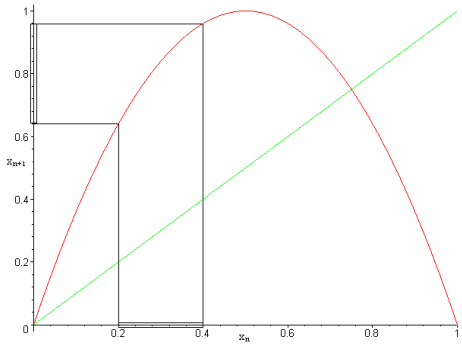

Figure 1.4: A sample of the kneading of the logistic map (r=4) after one iteration

the size of the original range. The fact is, the smaller the range the larger the number of iterations are needed for the same end result. Thus we see that in a chaotic regime the trajectory is topologically dense, thus validating Devaney's last two criterion for chaos.

1.2.3 Lyapunov Exponent

One of the best understood qualities of a chaotic system is the divergence from trajectories based on initial conditions. Since this is a common characteristic of chaotic systems there must be a way of analyzing it, and thus identifying the chaos associated with it. The exponential separation between two trajectories by a perturbation is called the Lyapunov exponent [33]. Generally speaking, there is one Lyapunov exponent for each degree of freedom in the system. A Lyapunov exponent in a flow equation is simply the Lyapunov exponent of the Poincare section 1.3 of the attractor.[35] The Lyapunov exponent is easiest understood for maps, and we will be

going into this in more detail in chapter 3.

The Lyapunov characteristic exponent is a method of quantifying the sensitive dependence of a system on the initial conditions.[33] This is a good method of determining the level of chaos in a system, since it measures the exponential divergence of a system with even the smallest difference of initial conditions. Calculation of the Lyapunov exponent is a complicated task, however the algorithm for calculating it is fairly straight forward. The Lyapunov exponent is the coefficient of the exponential separation of two initial guesses λ. These values can be of three states, they can be 0, thus indicating to the observer that there is no divergence or convergence of the system over time. This simply means the system does not change if different initial conditions are used. The Lyapunov exponent may be less than 0, indicating that the two initial conditions will eventually converge to the same trajectory. If the Lyapunov exponent is greater than 0 we have a situation where the two trajectories will diverge no matter how close the guesses are. The final state will be completely different over a finite amount of time, thus the system is chaotic. In the limiting case $\lambda \to \infty$ the system becomes random. What we start with is the idea that there is a relation between the steps along a trajectory, $\delta x(t) = e^{\lambda t}\delta x(0)$ where $\delta x(t) = x_1(t) - x_2(t)$ and $\delta x(0) = x_1(0) - x_2(0)$. The Lyapunov exponent is then found to be $\lambda = \frac{1}{t}\ln\frac{\delta x(t)}{\delta x(0)}$. The logarithm argument is of the form of a well defined operation, the derivative of

the mapping function, $f(x)$. So we have:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln \left| \frac{df(x)}{dx} \right|_{x=x_i} \tag{1.2.13}$$

If we were to drop the logarithm we would be left with another dynamical average, the Lyapunov number, or more conventionally e^{λ}, which has the form:

$$L = \lim_{n\to\infty} \left(\prod_{i=1}^{n} \left| \frac{df(x)}{dx} \right|_{x=x_i} \right)^{\frac{1}{n}} \tag{1.2.14}$$

To simplify matters more, a periodic orbit has a finite number of rotations, and thus the number n is finite, N, so we get:

$$\lambda = \frac{1}{N} \sum_{i=1}^{N} \ln \left| \frac{df(x)}{dx} \right|_{x=x_i} \tag{1.2.15}$$

.

So, if we were to take two trajectories with initial points separated by ϵ_0 that result in a separation of ϵ_n after n iterations we have that $x_0, x_0 + \epsilon_0 \to x_n, x_n + \epsilon_n$ and so we may define λ as:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \ln \frac{\epsilon_n}{\epsilon_0} \tag{1.2.16}$$

which characterizes the sensitivity to initial conditions of a 1 dimensional map. Basically, no matter how small ϵ_0 is, the separation of the resulting trajectories will be large after sufficient iterations.

For an analytic approach, we can take

$$\epsilon_n = f^n(x + \epsilon_0) - f^n(x) \tag{1.2.17}$$

where what we are really interested in is the limit as $\epsilon \to 0$. The Lyapunov relation in equation 1.2.16 becomes

$$\lambda = \lim_{n\to\infty} \lim_{\epsilon_0 \to 0} \frac{1}{n} \ln \frac{f^n(x+\epsilon_0) - f^n(x)}{\epsilon_0} = \lim_{n\to\infty} \frac{1}{n} \ln \left| \frac{df^n}{dx} \right| \tag{1.2.18}$$

and so for one dimensional systems we have that the Lyapunov exponent strictly depends on the first derivative evaluated along points on the trajectory.

One must keep in mind that the Lyapunov exponent is defined for a chaotic system. Since chaotic systems are bounded, they must experience folding as described above. Equation 1.2.16 only works for orbits that do not experience this folding, and the value for ϵ_n must be less than the size of the attractor [55]. Basically, one must choose the size of ϵ_n well, and to circumvent the folding effects.

If we consider the logistic map, it follows from Equation 1.2.18 that the Lyapunov exponent takes on the value:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln |r(1-2x_i)| \tag{1.2.19}$$

which results in a Lyapunov spectrum as seen in Figure 1.5

This system lies in a bounded set [0,1], and since it is bounded we have that the orbits must experience folding, which occurs at the diagonal $y = x$, also known as the substitution line [36]. We see in Figure 1.4, the orbit folds whenever the current position x_n is greater than the critical point $x = 0.5$. The separation still occurs but in the opposite direction, thus the use of the absolute brackets in the

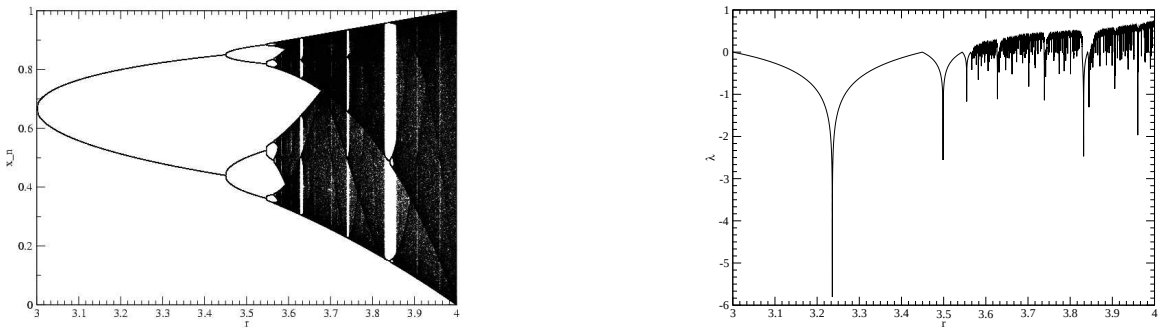

Figure 1.5: The Logistic Bifurcation and the corresponding Lyapunov Spectrum

Equation 1.2.18, which allows use to circumvent the folding process. In figure 1.5 we see a direct comparison between the bifurcation diagram and the Lyapunov spectrum. The basic traits are that in periodic regions the Lyapunov exponent is negative, and whenever a bifurcation occurs we have an instability, and thus a Lyapunov exponent of zero. When we enter the region where $r > r_\infty$ the Lyapunov exponent begins to take on positive values, and thus we have that the system becomes divergent. In the region beyond r_∞ there are points where the Lyapunov exponent goes negative, corresponding to regions of periodicity in the chaotic regime.

The above works well for 1 dimensional systems; however since the systems of interest are usually higher dimensional, we need to generalize the above equations to higher dimension. To start, there is an obvious connection between the Jacobian of a system, and the first derivative. So the first thing we do is write the Lyapunov calculation as:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln |\mathbf{J}|_{x=x_i} \tag{1.2.20}$$

where $|\mathbf{J}|$ is simply calculated as $|det(\mathbf{J})|$ or even simpler $|\prod_i \Lambda_i|$ with Λ_i being the eigenvalues of the Jacobian. The details for justification will be presented in chapter 3. Analogously, we find that the Lyapunov numbers of the flow are the products of the eigenvalues, over an asymptotic limit. Although this defined formally the Lyapunov exponent, it is difficult to determine this analytically for flows, so a more numerical approach is required. What one does to perform a general calculation of the largest Lyapunov exponent is fairly straightforward. First you pick a point on the attractor. Next you pick a point that is close to it, and that is also on the attractor, finding the distance between these two points, using a metric of your choice. Now evolve the two points through the Poincare section, and find the distance between the two resulting points. Take the log of the ratio of these distances. Now to repeat this process, the second point is already far from the original trajectory, and therefore a new point is necessary. If we follow the equation

$$\begin{aligned} x_{b0} &= x_{a1} + \tfrac{d_0}{d_1}(x_{b1} - x_{a1}) \\ y_{b0} &= y_{a1} + \tfrac{d_0}{d_1}(y_{b1} - y_{a1}) \end{aligned} \tag{1.2.21}$$

where d_0 is the previous point separation, and d_1 is the new point separation, we get a decent separation of the new trajectory point, and a guess that deviates from this guess. Finally one simply calculates the average of all these separations. This method works well for simple maps however a more useful approach comes from Wolf [2] where the Jacobian matrix is used, along with re-orthonormalization. One difficulty with the original calculation is that for high resolution of the Lyapunov exponent, one may

quickly exhaust computing power, and so only short time values may be easily found.

The study of the Lyapunov spectra is commonly performed using symbolic techniques. In this the negative λ are denoted as '-' since the important part of the Lyapunov exponent is in direction of convergence. Zero, and positive Lyapunov exponents are likewise referred to as '0', and '+' respectively. If the Lyapunov spectrum is completely negative,(-,-,...) we have a convergent system in all directions, and the resulting attractor is referred to as a fixed point. If the spectrum has one zero with the rest negative (-,-,0,-,...) we have a system that converges from all but one direction in phase space, which results in a 1 dimensional curve.[55] If the system is bounded, the attractor is cyclic, and thus a periodic orbital motion ensues. With two zeros we have a 2 dimensional system, which results in quasiperiodic motion.[55] All we need is one Lyapunov exponent greater than zero to have an unstable system, since trajectories in at least one direction will be exponentially separating. Again, if the system is bounded, such as the case for a driven pendulum, and has at least one positive Lyapunov exponent the system is defined to be chaotic.[36]

Haken's Theorem simply states that in a 2D system, a trajectory will either converge to a fixed point in space (corresponding to Lyapunov exponent less than 1), or the trajectory will be a periodic orbit (corresponding to a Lyapunov exponent equal to zero). No other possibilities can exist for a system with such low dimensionality.

Another important constraint on the Lyapunov exponents is that the sum of all

exponents of a trajectory must equal the expectation value of the dissipation of the flow. This effectively offers:

$$\sum_{i=1}^{n} \lambda_i = lim_{t\to\infty} \frac{1}{t} \int (\nabla \cdot f)(\tau) d\tau \tag{1.2.22}$$

So, for a dissipative system, the sum of the Lyapunov exponents must be negative, whereas for a conservative system the sum of the exponents must be zero. The Lyapunov spectrum allows for analysis of the phase space diagrams, which is especially useful for higher dimensional systems where 2 dimensional projections are not entirely useful.[55]

Using the divergence relation, (Equation 1.2.22), we find that the sum of the Lyapunov exponents for our system is $-\frac{1}{Q}$ and from Haken's theorem we see that in general there is always a zero Lyapunov number, and so considering the largest exponent we easily determine the remaining exponent. Through the use of the Lyapunov exponents we can determine if the dynamical system has sensitive dependence on initial conditions, and in so satisfying the first of the three listed criterion for a chaotic system.

1.2.4 Winding Number

Another common feature of a chaotic system is the winding or rotation number. This is defined as the average number of rotations executed by the orbit in the short direction for each rotation it makes in the long direction[48]. In graphical terms, this

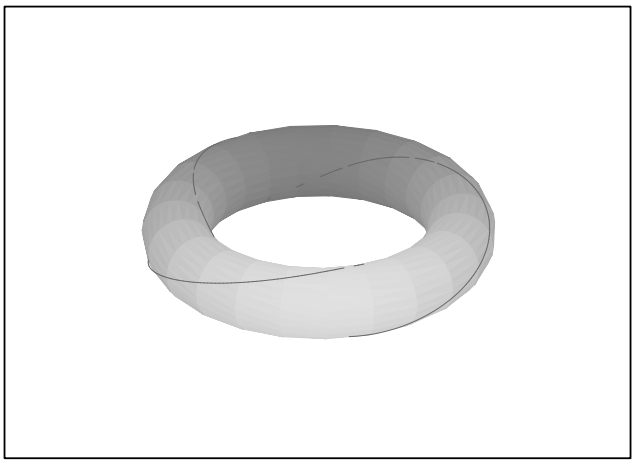

Figure 1.6: Torus with a winding value 1, twice around the smaller circumference, and twice around the larger circumference

is the number of times a sub-rotation completes a cycle, before the major rotation has completed the cycle. Mathematically this is $R = \frac{\Omega_S}{\Omega_L}$. If this ratio is irrational the entire manifold is covered because the loop never closes or crosses its own path. If the winding number is rational and in a reduced fractional form we claim that the numerator and the denominator are directly related to the period of the orbit.

In our system the winding number is the ratio of the number of rotations of the gear around the origin of our coordinate system as compared to the number of rotations made by the rod around its axis.

1.2.5 The Circle Map

The winding number has a close relationship with a map introduced by Arnol'd [48], the circle map. This type of map is relevant when the dynamical properties of the

manifold $\mathbb{S}^1$ map onto itself. In particular if the flow is parameterized with an angular variable, on the domain [0,2π] these maps may be written as $\theta_{n+1} = f(\theta)(\text{mod } 2\pi)$.[57] The study of circle maps was originally motivated by coupled oscillators, since each oscillator moves at its own frequency. If we proceed using a simple two pendulum coupling we are assuming a system that has two frequencies, ω and ω'. We then have a system that may be described by $\theta(t) = \omega t(\text{mod } 1)$. If we further use a Poincare section at frequency ω', the system may be measured by $\theta_n = \theta(t_0 + n/\omega')$ which results in the map $\theta_{n+1} = (\theta_n + w)$ with $w = \omega/\omega'$ This describes a rotation by a fraction w of a full turn per sampling period. If the value w is rational (ie of the form p/q for p,q$\in$ $\mathbb{Z}$), then we have $\theta_{n+q} = \theta_n + qw = \theta_n$ (mod 2π). If the value w is not rational the sequence $\{\theta_n\}$ densely fills the interval [0,2π]. [57] This is regarded as the quasiperiodic regime, which means that it is a superposition of two incommensurate frequencies. In coupled oscillators one observes something known as frequency locking; the frequency ratio of the two oscillators remains fixed at a rational value p/q in a finite range w$\in$ $[p/q - \Delta\rho_1, p/q + \Delta\rho_2]$

By the work of Arnol'd the circle map was extended to a general form, $\theta_{n+1} = (\theta_n + w + K\sin(\theta_n))\, mod(2\pi)$ The $k\sin(\theta_n)$ term models the nonlinear coupling in the system. This map reveals information about the frequency locking mechanism in a nonlinear system. If one wants to study the asymptotic regime of the Arnol'd map,

they need to define the rotation number as;

$$R = \frac{1}{2\pi} \lim_{m\to\infty} \frac{1}{m} \sum_{n=0}^{m-1} \Delta\theta_n \tag{1.2.23}$$

where $\Delta\theta_n = w + K\sin(\theta_n)$ [48]

If this maps the circle onto itself the following properties of the rotation number can be established;[57]

- The rotation number does not depend on the orbit used to compute it, that is it does not depend on the initial conditions.
- If R is irrational the circle map is equivalent to a pure rotation, the motion is quasiperiodic, or chaotic. An irrational winding number alone does not identify chaos in a system.
- If $R = p/q$ is rational, the asymptotic regime is a periodic orbit of period q. The periodic points of the orbit are ordered along the unit circle as with the pure rotation.

What one is doing here is testing the rotation of the system from an initial point. The most useful method of calculating the winding number in flows is the simple procedure that first we set up some initial condition (IC), then we use the Poincaré iterate to force the evolution of the IC's and the difference between the initial conditions and the resulting evolution, in the x direction. Then finally divide by the

number of times the Poincaré iterate has been called. This should be done several times over, say 10^6, to remove any transient behaviour. This information is only meaningful for a system constrained to a finite manifold.

1.3 Poincaré Section

There has been a lot of work done on attractors and possibly the most significant was done by Henri Poincaré, who took a cross section of the attractors and analyzed the data based on the findings in the cross section. This cross section took on the name of its discoverer, Poincaré. Technically, a Poincaré section is a stroboscopic view of the phase space, where the period of the strobe is usually natural to the system. It would be a simple task to project the attractor onto a 2D plane, however this may not be the most useful option, as sometimes it makes a bigger mess, as seen in Figure 1.7. With the Poincaré section we take a slice through the attractor that is parallel to the projection plane. What we are really observing is the evolution of the system as it passes through the Poincaré section. Often this is beneficial for analysis since two dimensional maps are much easier to follow than a three dimensional manifolds. We do not lose any generality of the system during this process since we control the location of the cross section. Poincaré noticed that if a curve was simple it was likely to have analytical solutions, if the curves were irregular and complicated then they represent a chaotic system. [33] In most cases the Poincaré section is taken at the driving frequency. The chaotic attractor of the system is determined by solving

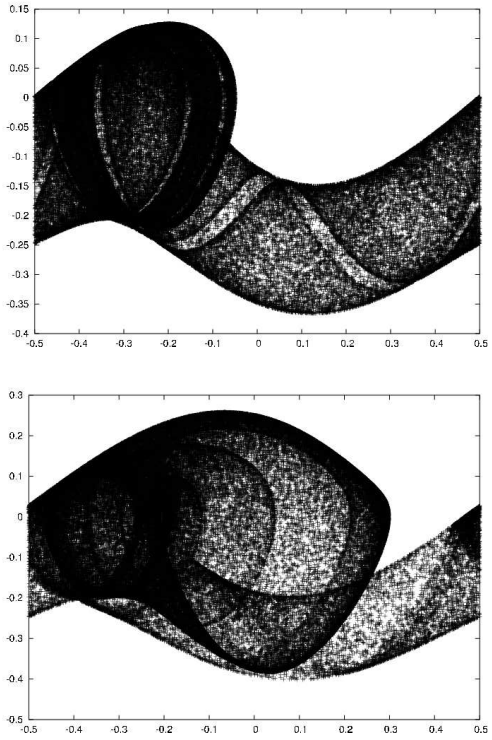

Figure 1.7: Projection of full attractors on plane

the ODE numerically. Clearly figure 1.7 is rather messy, but if one considers the Poincaré section see Figure 1.8 you get a dramatically less complicated form. Using the Poincaré section in the calculation, this simplifies rather nicely. See Figure 1.9 for an example.

In our case, this was done by the insertion of a Poincaré function within the original general solution of the ODE. One simply replaces the step intervals for the solver to be $2\pi/a$ [48] where 'a' is as described in equation 1.2.11. This forces the solver to display only the points occurring on the Poincaré section. A simple modification to the code, simplifies the analysis procedures dramatically.

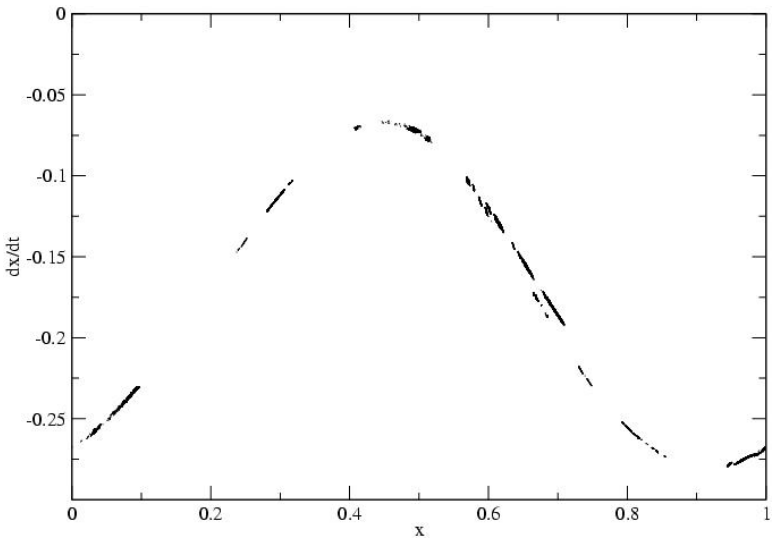

Figure 1.8: Q=0.76 section at $\frac{27\pi}{64}$ within a tolerance of 0.01

Poincaré conjectured that all of the information of the full attractor is captured in the Poincaré section. [48, 18] The placement of the Poincaré surface is of high relevance for the usefulness of the result. An optimal surface first maximizes the number of intersections if at the same time the attractor remains connected. Basically, the number of points collected in the section must be maximal, as well as the attractor comprising of a macroscopically connected curve. The second criterion is that the time delay between the return of the orbit through the section. The reference [57] contains graphical demonstrations of this effect.

1.4 Fixed Points

Another common feature of chaotic systems is the existence of fixed points. They follow the form $f(\vec{\mathbf{x}}) = \vec{\mathbf{x}}$ These may be found using return maps and the diagonal

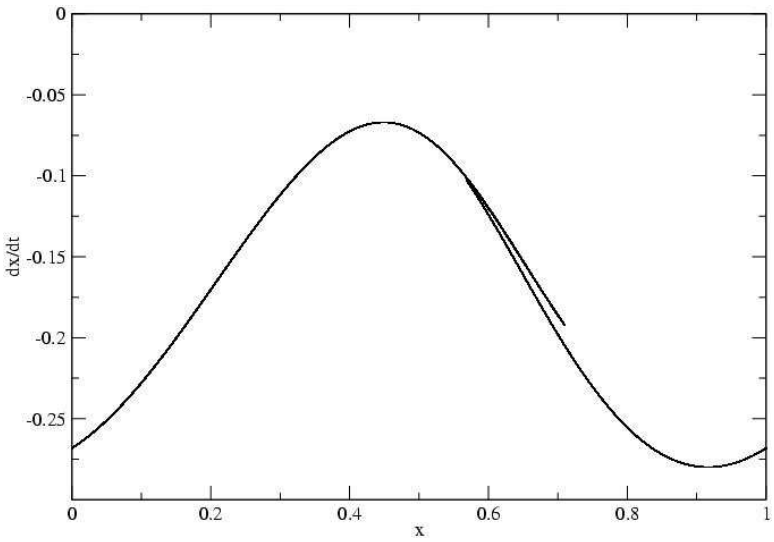

Figure 1.9: Q=0.76 True Poincaré section of above attractor

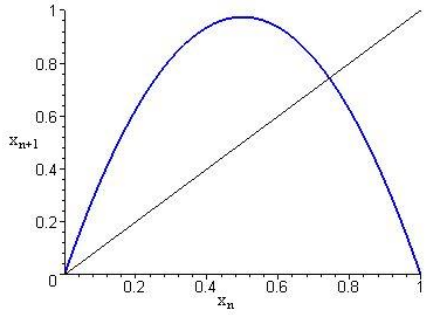

Figure 1.10: The Logistic Diagram

line y = x. Any point on the return map that intersects the diagonal is a fixed point. These points are important for finding periodic points in specific situations. [26] They also lead the researcher to find fixed points of higher order periodic points, when they use higher order return maps. See Figure 1.10 for a common example of a return map, the logistic return map. In the Logistic return map, the next point is plotted against the current, in the free coordinates. These maps are critical for finding periodic orbits

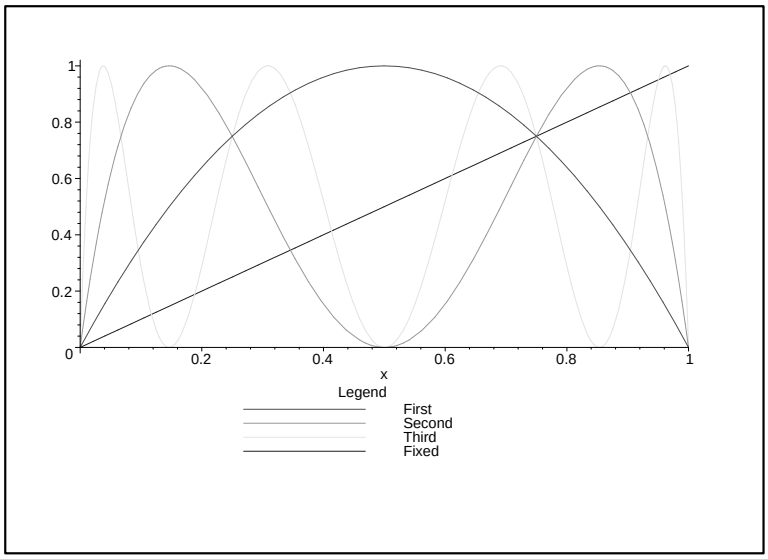

Figure 1.11: The Logistic Diagram Multiple Return Maps

since they reveal the kneading sequence, and may also be used to approximate the sequences of allowed orbits. With one stable coordinate point found one returns to the Poincaré section to determine the corresponding second reference point. Now one must decern the stability of these points.

1.5 Bifurcation

Bifurcation diagrams are maps that plot the output of the system against one or more of its parameters. This type of diagram is useful for determining regions of stability and instability. As seen in Figure 1.14, we see that there are a few regions of stability and several regions of chaotic dynamics. These diagrams are often used for simple analysis of dynamical systems, and are a good indication that chaos may exist in a system. For systems with more than one parameter several bifurcation diagrams

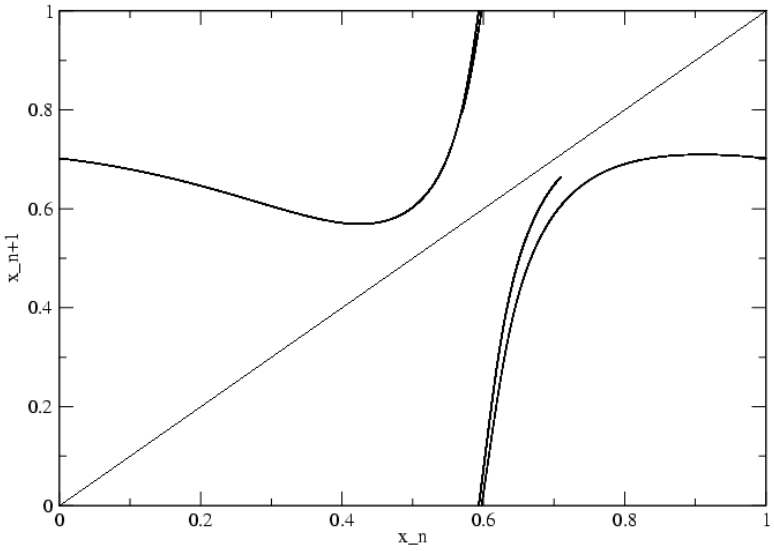

Figure 1.12: First return map for Q=0.76

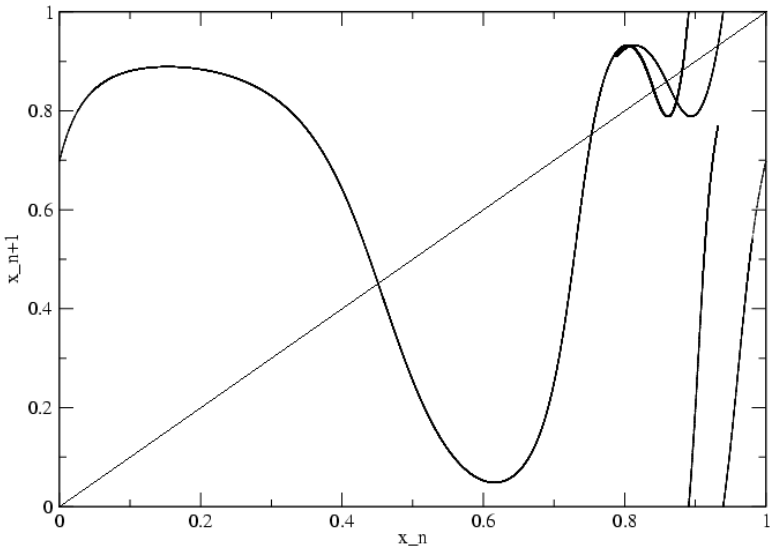

Figure 1.13: Second return map for Q=0.76

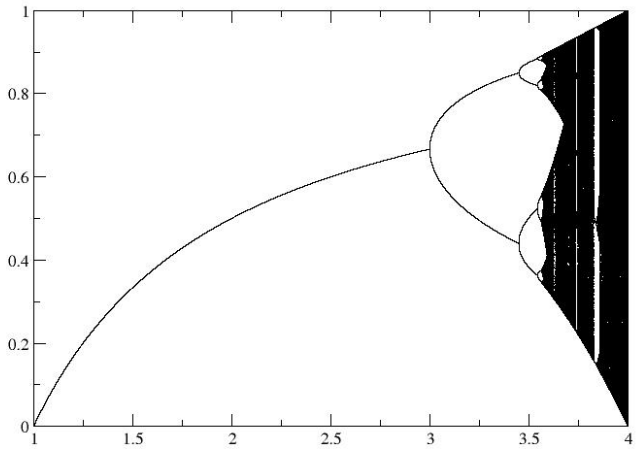

Figure 1.14: Logistic Bifurcation Diagram

are necessary and may be cross linked. Thus, to truly determine if a system with at least three free parameters such as the forced damped oscillator, is chaotic one must investigate the coupled dependence in several bifurcation maps. There are several interesting features of any bifurcation diagram, such as the pitchfork bifurcation, and the convergence from chaos. In 1964, A.N. Sharkovsky, while studying the Logistic bifurcation diagram, created a method of ordering the periodic orbits found in a bifurcation diagram, that became a universal ordering for all bifurcation diagrams. One starts by observing a bifurcation diagram, and begins counting the periodic orbits from the largest value of the tested parameter to the smallest value of the parameter.

He started with the third period, and moves along the following trend;

$$\begin{gathered} 2^0 \cdot 3 \prec 2^0 \cdot 5 \prec 2^0 \cdot 7 \prec \cdots \prec 2^1 \cdot 3 \prec 2^1 \cdot 5 \prec 2^1 \cdot 7 \prec \cdots \prec \\ 2^n \cdot 3 \prec 2^n \cdot 5 \prec 2^n \cdot 7 \prec \cdots \prec 2^n \prec 2^{n-1} \prec 2^{n-2} \prec \cdots \prec 2^2 \prec 2 \prec 1 \end{gathered} \tag{1.5.1}$$

The pitchfork bifurcation is not the origin of chaos itself, it is seen on the interval [3.0..3.5] and [3.5..3.55] in figure 1.14. In these regions we have oscillations between two stable solutions for the given parameter value. As these regions evolve they can begin to experience further pitchforking, or higher order splitting.[33] The pitchforking is an onset of an avalanche of allowed periodic orbits, the system is simply becoming more and more complicated, but countably so. The onset of chaos occurs when the periodic orbits begin to take on odd values. That is on a bifurcation diagram one can pick regions where there are an odd number of stable points. This occurs on a fine line just after the 2^∞ point. [36, 68] As the parameters evolve further we may find that the chaos is continuous, or contracting back to the smaller oscillations until it returns to a single solution, as is seen in figures 1.15, and 1.16.

The work of Fiegenbaum found more constants of bifurcation diagrams, known as the first and second Fiegenbaum constants. These are measurements of the bifurcations of the system. The first constant is a measure of the ratio of the distance between bifurcations along the parameter coordinate from the previous bifurcation. Graphically this appears as seen in Figure 1.17

From a formula standpoint this is simply;

$$\delta = \lim_{n \to \infty} \frac{r_{n-1} - r_n}{r_n - r_{n+1}} \tag{1.5.2}$$

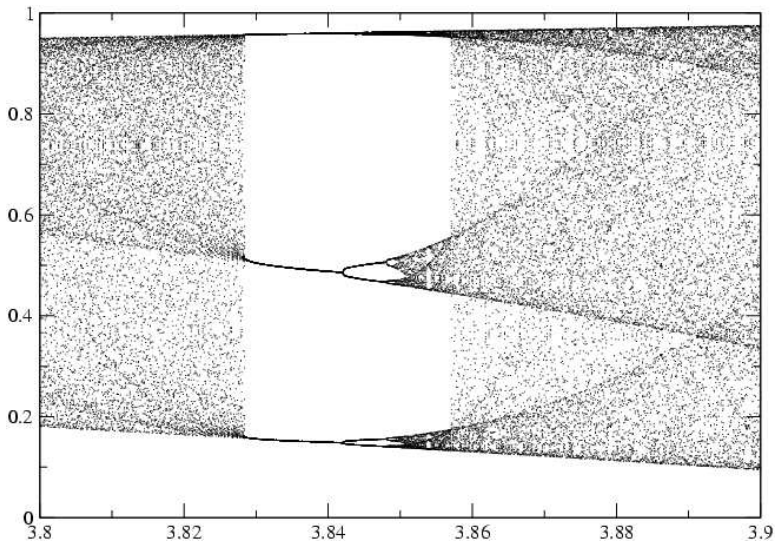

Figure 1.15: Logistic Bifurcation Diagram zoomed within regions $\mu = 3.8 \rightarrow 3.9$

The second constant is a measure of the separation of the different bifurcations from the complimentary branch. Graphically this appears as seen in figure 1.18 and the ratio is;

$$\alpha = \lim_{n \to \infty} \frac{w_{n+1}}{w_n} \tag{1.5.3}$$

[36]

This is good for the 1 dimensional maps exhibited by the logistic map. However, this is slightly more complicated for higher order differential equations. With these more common systems, one needs to determine the bifurcation through a Poincaré section.

From an experimental point of view, in our system the gear ratio is fixed, as is the rod length, so the only parameters that may be actively altered once the experiment has been set up is the quality factor "Q", and the effective drive frequency.

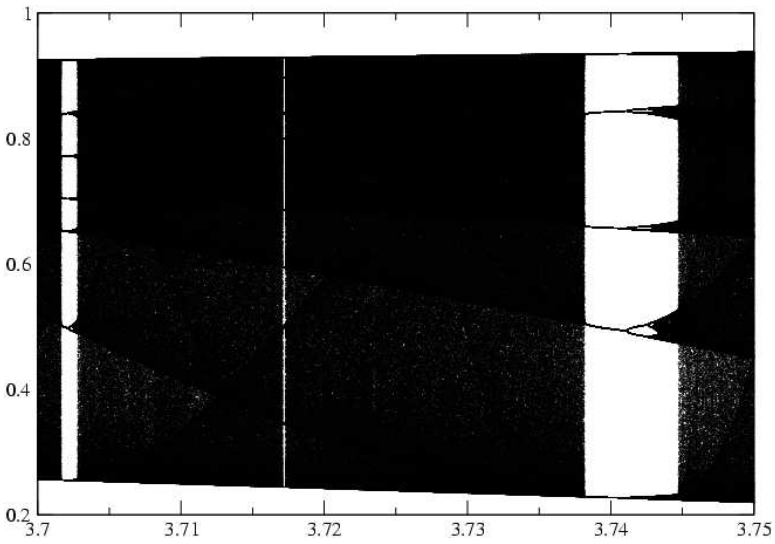

Figure 1.16: Logistic Bifurcation Diagram zoom within regions $\mu = 3.7 \rightarrow 3.75$

See equations 1.2.11. A simple relation was used between the two free variables, $a = 0.8 + 0.3Q$ and from this a bifurcation diagram for the system at hand was created.[55] From here two specific regions were selected since they exist in a chaotic region of the bifurcation diagram. [55] These regions were Q = 0.76, a = 1.028, and Q = 1.25771, a = 1.17731. We focus the bifurcation plot in figures 1.20 and 1.21

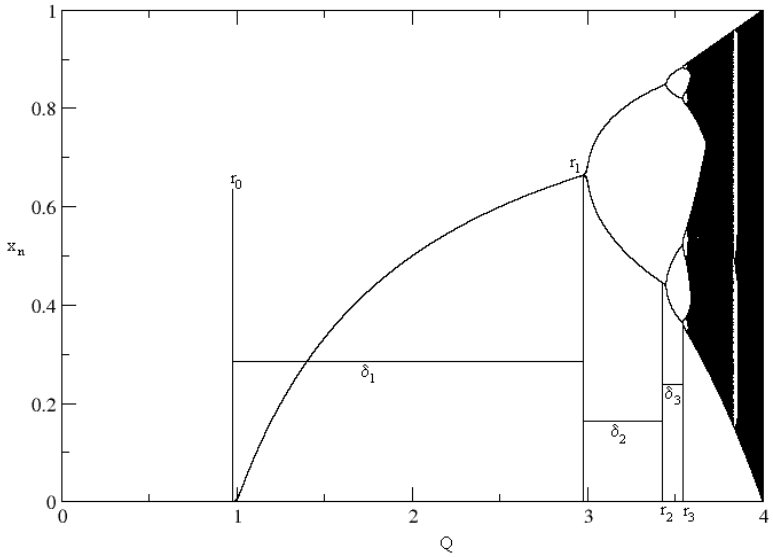

Figure 1.17: The First Feigenbaum Constant

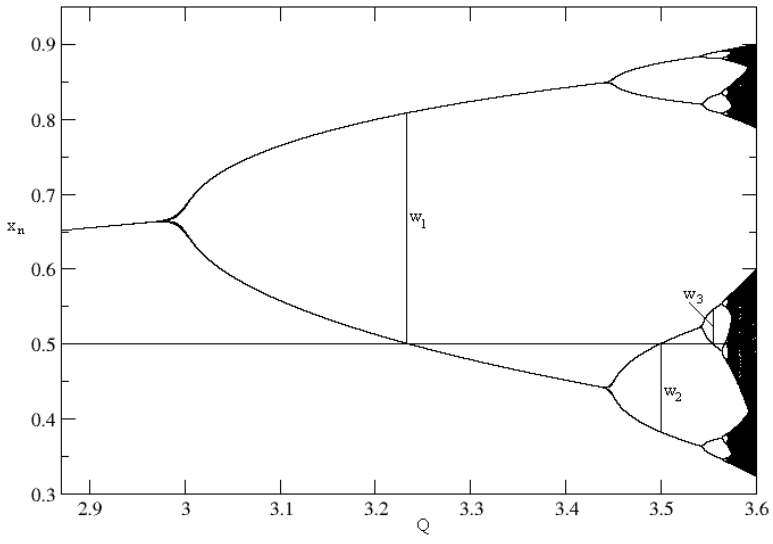

Figure 1.18: The Second Feigenbaum Constant

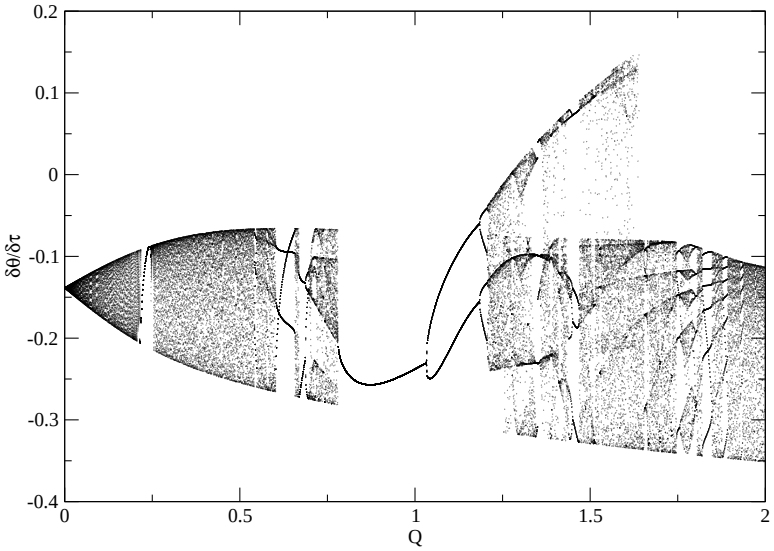

Figure 1.19: 2 Gears and Rod Full Bifurcation Diagram, a = 0.8 + 0.3Q

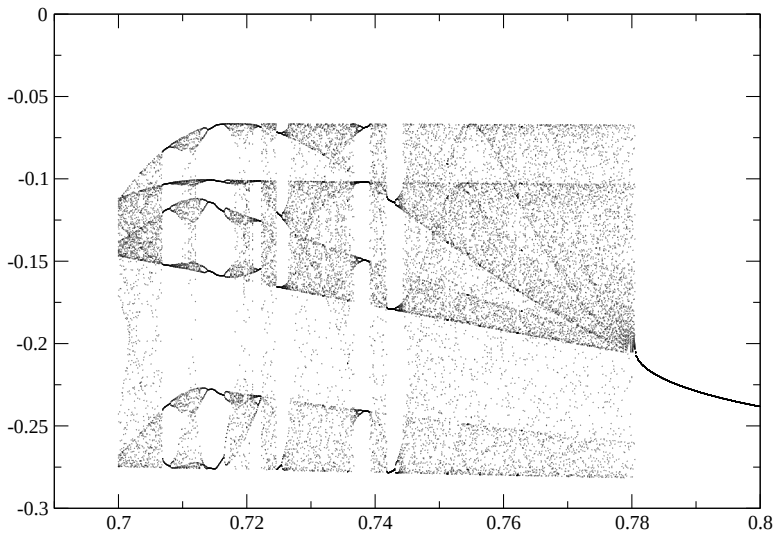

Figure 1.20: 2 Gears and Rod Bifurcation Diagram, Q = 0.76 a = 1.028

This depth of study in chaos is a relatively new topic since only in the last 20 years calculations have been fairly easy to perform, and it has wide applications. So then how may we be able to determine if a system is chaotic or at least has the potential to be? Well, we have already discussed the bifurcation diagram which provides an excellent graphical display of chaos. Other indications of chaos arise from the Poincaré-Bendixon theorem that states that a system with fewer than 3 differential equations cannot exhibit chaos.

Let f be a smooth vector field of the plane, for which the equilibrium points of $\dot{\mathbf{v}} = \mathbf{f}(\mathbf{v})$ are isolated. If the forward orbit $\mathbf{F}(t, \mathbf{v}_0)$, $t \geqslant 0$, is bounded, then either

- $\omega(\mathbf{v}_0)$ is an equilibrium, or
- $\omega(\mathbf{v}_0)$ is a periodic point, or

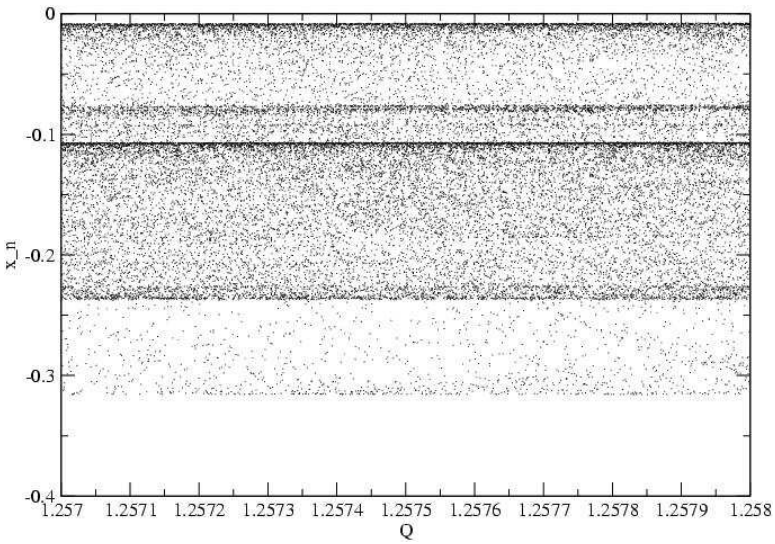

Figure 1.21: 2 Gears and Rod Bifurcation Diagram, Q = 1.2577 a = 1.17731

- For each $\mathbf{u}$ in $\omega(\mathbf{v}_0)$, the limit sets $\alpha(\mathbf{u})$ and $\omega(\mathbf{u})$ are equilibria

Otherwise, it takes a deeper analysis of a system to determine the levels of chaos exhibited.[35]

1.6 Fractals

Fractals are a feature of many nonlinear systems. They are geometric shapes that may come in both integer and non-integer dimension, depending on the system being studied. Fractals have a least one common distinguishable feature, which is that they are everywhere non-differentiable, they consist solely of singularities. The original definition of a fractal is that its Hausdorff dimension strictly exceeds the topological dimension. Simply put the measured dimension is greater than the apparent dimension. Fortunately this definition was not strictly adhered to, since further studies of

what are now considered fractals would have been ignored, an example of which is the Mandelbrot set. These shapes occur frequently in nature – one of the oldest physical examples studied is the coast of Britain.

Since fractals are a geometric structure we must concern ourselves with the dimension of such a feature. There are three common standards, the box dimension, the information or entropy dimension and, the correlation dimension. All of these dimensions have a similar form, they are an asymptotic behaviour of the system over the scale of each measurement. That is they are the log of a measurement divided by the log of the scale at which they did the measuring. The easiest to perform and the easiest to describe is the box dimensions so we shall do that here. The box dimension is a counting dimension, where one only counts the boxes containing information.

For a quick demonstration we can take the bifurcation diagram. The first step is to choose a scale for the dimension. Say we take the entire length of the box to be 1, which is a box that covers the entire diagram. The next series of boxes may be reduced to half their original dimension, thus leaving us with 4 boxes in the same space as the first. Now we look at the boxes we have created and remove any that do not contain any piece of the bifurcation diagram. Now we still have 4 boxes to now scale to a quarter of the original size. Thus each of the three boxes contain 4 smaller boxes. Again we remove any box that does not contain the bifurcation diagram, and down-scale the boxes, making sure that at each step we count the number of boxes

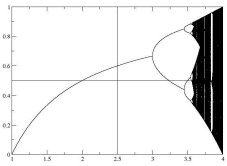

Figure 1.22: The logistic bifurcation diagram with a covering of 4 boxes

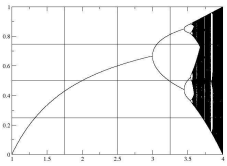

Figure 1.23: The logistic bifurcation diagram with a covering of 16 boxes

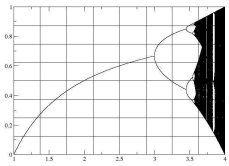

Figure 1.24: The logistic bifurcation diagram with a covering of 64 boxes

remaining. We see when considering 16 possible boxes only 11 contain the image, and with 64 boxes only 28 contain the image. We do this until a physical limit is reached, such as the width of a pixel on a computer screen.

The calculation for the box dimension is simple:

$$D_{HM} = \lim_{k \to \infty} \frac{\log(N_k)}{\log(r^k)} \tag{1.6.1}$$

where the subscript HM stands for Hausdorff Mesh, which is named after the inventor. The N_k is the number of volume units (vels) that still contain the attractor, and the r^k is the scale of the vel. When using an algorithm to perform this calculation, it is best to operate with the data used to plot the attractor, rather than the plotted attractor itself, since information may be added or removed from a system artificially by graphical output.

The other two dimensions are described in more detail in chapter 5. In truth there are a possible infinity of dimensions allowed to measure a system. Fractals also come in several different varieties such as compound, simple, and multifractal.

(see chapter 5). The topological dimension will simply ignore some of the attractor if it becomes disjoint, for a good example see [36] where Kinsner discusses the box counting dimension as it acts on a disjoint set of a line and a point. In this case Kinsner claims that if we were to only consider the box counting dimension we would have only the dimension of the line. Therefore, other dimensions are required for proper analysis.

The physical nonlinearity of nature is most obvious when one observes the physical shapes that nature is comprised of. It is not common in nature that one will find a stray cube, a pyramid, or even a perfect sphere. In fact many of these standard geometric shapes are only approximations to natural objects. For example the moon's geometric shape appears to be a sphere, however as one gets closer to the moon they find craters and ridges which are fractals. What we usually find are almost disorganized shapes such as coastal regions and mountain ranges. These cannot be properly approximated as geometric shapes, but we all agree they are physical and do have some kind of shape. We have now defined that shape to be of the fractal class. The study of fractals to understand the real world is important, not to mention useful in industry for real image manipulation.

Chapter 2

Symbolic Dynamics

2.1 Introduction

Symbolic dynamics began with J. Hadamard and H.M. Morse in 1898, where they applied this idea to geodesic flows on surfaces of negative curvature.[16] The term "Symbolic dynamics" was coined by Morse and Hedlund in 1938 [28]. This new creation simplified the studying of the geodesic flows by producing symbolic sequences that were forbidden, and that any sequence is possible as long as it doesn't contain any forbidden subsequences. 20 years later Morse and Hedlund continued work on the topic and showed that finite descriptions of dynamics were possible. Symbolic dynamics was then used to study dynamical systems which were made up of differential equations. These systems are commonly studied by discretizing time, however Symbolic dynamics is an attempt to discretize both space and time.[16] The intent was to divide the phase space into a finite number of pieces and keep track of which piece the system was in after each passing unit of time. Each state is labelled by a

symbol and thus as the states evolve one is able to observe the trends of the evolving system. [16] Basically it is the study of dynamical systems generated by the shift map, described below, in the spaces of some finite alphabetic sequence denoted w_k. A *shift* is the evolution that occurs with a symbolic sequence. We will briefly describe the orbits in phase space in terms of a symbolic sequence.

With our 3 dimensional, invertible, dissipative system, we expect three Lyapunov exponents, one of which is zero, one in an unchanging direction, and one is expected to be negative, a contracting dimension. Therefore since the sum of the Lyapunov exponents must be positive, we will have the remaining exponent being positive. So we have only one stretching direction, and thus we will have 1 dimension like mappings. [38]

Consider an orbit $x_i = f^i x$ in phase space, denoted as P. We must assume that the phase space may be partitioned into subsets $\bigcup_k P_k = P$ and $P_i \bigcap P_j = 0$ for $i \neq j$ If this assumption is invalid the method of partition will not work. To keep things entirely general we take the approach offered by [68], where we relate the value $f^i x$ to a symbol j if $f^i x \in P_j$. Basically, if the ith iteration of a function takes the value of x which is within the P_j partition the resulting value takes on the symbol j, rather than some value that may have an infinite mantissa. This is very important to chaotic systems since the infinite mantissa is essential for the absolute knowledge of the resulting point on the attractor. Recall the property of transitivity. Ultimately we

receive a sequence $\{j_0 j_1 j_2 \ldots\}$ that entirely codes the orbit determined by $\{f^i x\}$ with x being the points of the orbit that intersect the Poincare section. If the map is one-to-one, as it is in our case, we are left with the bi-infinite sequence $\{\ldots j_{-1} j_0 \ldots j_n \ldots\}$ which is determined by the rule: if $f^i x \in P_{j^*}$ then $j_i = j^*$, $i \in \mathbb{Z}$. This comes about since we have a point on the manifold, and invertibility tells us that we may know about the history of the trajectory. However one must keep in mind that chaos implies a limited knowledge of the history, to perhaps only a few steps.[14]

The ease of the sequence analysis is that an arbitrary starting point on the periodic orbit may be selected. The symbol sequencing will adjust by shifting the sequence until the initial point is the first symbol in the sequence. The sequences may not all be admissible but the dynamics of the system $\{f^n, P\}$ are reflected by the dynamics in the shift map.[68] Admissibility conditions are described later.

The 2-Sided Bernoulli shift is the shift type that one is concerned with for a diffeomorphism.[68] This type of shift deals with the bi-infinite sequences. The phase space for this system is;

$$\Omega = \{0,1\}^{\mathbb{Z}_+} = \{\underline{w} = (\ldots w_{-1} w_0 \ldots w_n \ldots), w_i \in \{0,1\}\} \tag{2.1.1}$$

with the metric

$$dist\{\underline{w}', \underline{w}"\} = \sum_{i=-\infty}^{\infty} \frac{|w_i' - w_i"|}{3^{|i|}} \tag{2.1.2}$$

The map generating the dynamical system is the shift map $\sigma : \Omega \to \Omega, \sigma\underline{w} = \underline{w}'$ where $w_i' = w_{i+1}$ and σ is the shift operator.

As a simple example take a bi-infinite sequence $\sigma(\underline{w}) = \sigma(\ldots abcd \bullet efgh \ldots) \rightarrow \underline{w'} = \ldots abcde \bullet fgh \ldots$ where it is obvious that the first symbol in the forward sequence becomes the last symbol in the backward sequence. The $\bullet$ is the current position in the orbit j_0. This makes logical sense since the present location of any orbit changes over time. If one is dealing with a periodic orbit the forward and backward sequences are easy to determine since any backward sequence is just a reverse cyclic permutation to the orbit, and the forward sequence is just a forward cyclic permutation. The difficulty with dissipative system is that given the present location, finding the past locations is not a trivial task. For a full treatment of this topic see Lind and Marcus [16].

2.2 Partitions

With aid from the symbolic dynamics of a system we can map the trajectory of a dynamical system through space. Symbolic dynamics requires a separation of the phase space into logical breaks, that are generally based on some finite knowledge of the attractor at hand. I say "finite knowledge" since we only know a small level of accuracy for the decimal value of the break. Problems occur as we approach these breaks too closely. In the standard example for this system we look at the logistic map. See figure 2.1 With the logistic map, the break is actually very well defined to be the peak of the quadratic, or more generally the critical point of the attractor. Since we are using a two symbol system, we will use the symbols L and R. As we

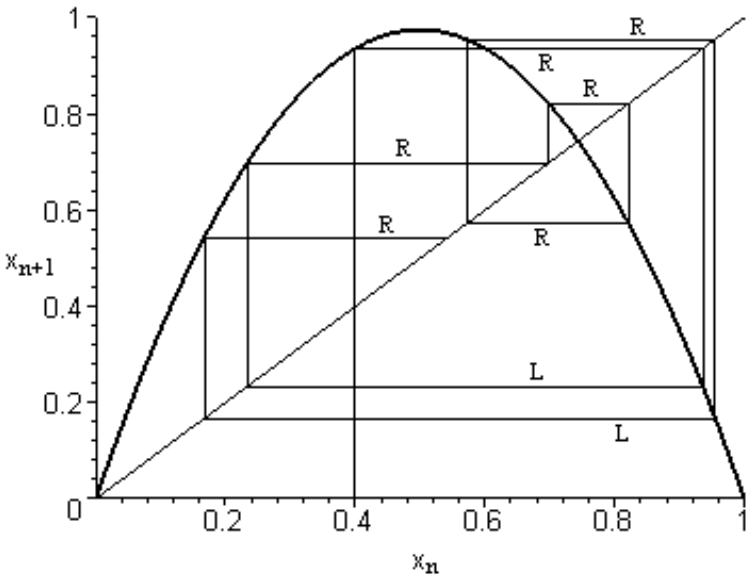

Figure 2.1: Logistic Diagram, here we se the sequence is RLRRRRLR...

track a trajectory we label it as it moves through the phase space. As an example we start with the initial position $x_0 = 0.4$ on the logistic map again see figure 2.1 with $\mu = 4$ we then follow the trajectory starting in the L region to the R region back to the R region, and *etcetera*. This is same procedure as outlined in chapter 1 in the section on Kneading. What we find are general patterns that emerge depending on which region, L or R, we use for a starting condition. These trends are described below.

For higher dimensional systems a different form of partition may be necessary. The previously mentioned method of using critical points on the return map may not work if the critical points are saddle points, since these are neither maxima or minima points on the map. The question then is, what do we do in systems with

saddle points for critical points? The method used for this type of system is the method of homoclinic tangencies, where one plots the forward iterations on the same plot as the backward iterations which is simply the attractor of the system. The point(s) where these two iterates are tangent is called a homoclinic point(s). These homoclinic points are used as the partition points. If the attractor is multisheeted, the tangency points are connected. That is, the partitions are not necessarily straight lines in the phase space. These tangency points are easily mapped to the return map where they are of more use for future analysis, since this is where the periodic orbits may be found manually.

2.2.1 Pruning

Pruning rules are necessary for the symbolic dynamics. Pruning rules act as the name suggests: they control the direction of growth of symbolic patterns by setting up boundary conditions. These were evident even with Hadamard's negative curvature analysis, when he found forbidden sequences. To find these pruning rules we observe some form of boundary condition and apply that as a rule for future orbits. The pruning rules are in general difficult to determine, only in a few cases are they trivial, but clever methods of separation may lead to more obvious pruning rules.

For the rod and gear system we opt to use kneading sequences which are an inherent property of the system. A kneading sequence is a symbol string that is created by choosing the initial condition to be the same as the critical value of the

map, or the homoclinic point. In n-modal maps, which are maps with n critical points/homoclinic points, this sequence would be created by starting at a point along the attractor (return map) that it known to be at (or near) the partition line. In multisheeted maps there may be more than one kneading sequence for a given critical or homoclinic point. This means that if we start from the same section in phase space we may choose to extend the iterate to any of the available sheets, thus producing several kneading sequences from one homoclinic point. The pre-image is also useful and is created by using the tangency point as the initial point of iteration on the return map. However we go in the opposite direction as the kneading sequence iterations. Simply iterate the map in the opposite direction as that described in logistic map, to get the pre-image for the logistic map. The symbolic sequences may be infinite non-repeating, in such a case we only need to follow the forward iterations to the required precision. That is, based on trinary arithmetic we only need to pursue a finite number of symbols along a sequence. For the kneading sequence to be of any computational use we must have a metric for the symbol space, and since the attractor is normalized between 0 and 1 the kneading sequence must also be normalized. Using methods like those found in [50, 4] we define the metric for the system. The kneading sequences can readily be converted, by using the definition of the trinary arithmetic.[60] The metric plane may be defined by the following metrics;

first we define an integer ϵ_i for each symbol s_i

$$\epsilon_i = \begin{cases} -1, & \text{if } s_i = N \\ 1, & \text{otherwise} \end{cases} \tag{2.2.1}$$

Then the forward sequences are replaced by their metric:

$$\alpha = \sum_{i=0}^{\infty} \frac{\mu_i}{3^i} \tag{2.2.2}$$

where we use 3^i because we are using a 3 symbol system. For the system to be well ordered the μ is defined as

$$\mu_i = \begin{cases} 0, & \text{if } s_i = R \\ 1, & \text{if } s_i = otherwise \end{cases} \tag{2.2.3}$$

We make a backwards sequence by defining a similar metric using

$$e_i = \begin{cases} -1, & \text{if } s_i = L \\ 1, & \text{otherwise} \end{cases} \tag{2.2.4}$$

Then the backward sequences are replaced by their metric:

$$\beta = \sum_{i=0}^{\infty} \frac{\nu_i}{3^i} \tag{2.2.5}$$

Again considering well ordering ν is defined as

$$\nu_i = \begin{cases} 0, & \text{if } s_i = N \\ 1, & \text{if } s_i = rest \end{cases} \tag{2.2.6}$$

This leaves us with some test sequences, when considering limits in the metric

$$\alpha(\bullet NL^\infty) = \beta(L^\infty\bullet) = 1 \qquad \alpha(\bullet L^\infty) = \beta(R^\infty\bullet) = 0$$
$$\alpha(\bullet NNL^\infty) = \alpha(\bullet RNL^\infty) = 2/3 \quad \beta(R^\infty L\bullet) = \beta(R^\infty N\bullet) = 2/3$$
$$\alpha(\bullet RL^\infty) = \alpha(\bullet LNL^\infty) = 1/3 \quad \alpha(L^\infty N\bullet) = \beta(L^\infty R\bullet) = 1/3$$

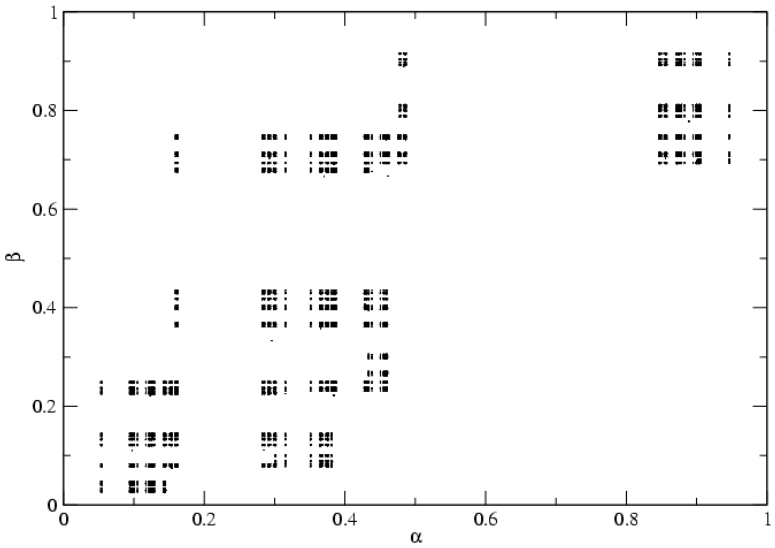

Figure 2.2: Symbol Plane, Q = 0.76

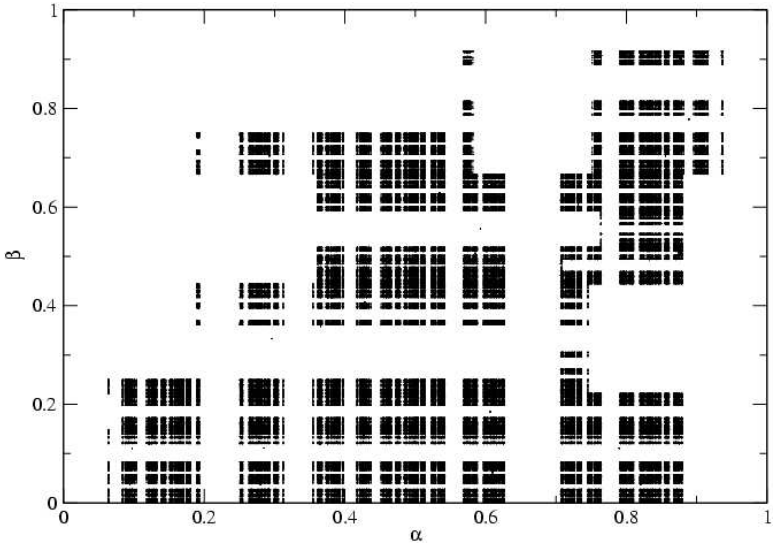

Figure 2.3: Symbol Plane, Q = 1.2577

Now, with the trinary arithmetic defined, we can continue with the orbit hunting. The first thing to realize are the boundary conditions that have been freshly imposed. These act in two directions for our bimodal map, forward iterations and backward iterations. When these are plotted on iterative space we see they form boxes and these are referred to as fundamental forbidden zones (FFZ). They are shown in figures 2.2 and 2.3, the white space are the FFZ's. Basically, no orbit, or part of an orbit, is permitted to cross into these regions thus ruling them out from the list of possibilities. This corresponds to Hadamard's original ruling that any sequence containing a forbidden subsequence must also be forbidden.[25] We have added conditions that as long as any shift of a given orbit does fall within the FFZ's it is also inadmissible. Graphically these FFZ's may be seen on the symbolic plane. One needs only to plot the allowed orbits against the symbol plane and they may find cutoff points that the system will naturally avoid, these are FFZ's. With a 2 dimensional map, there are an infinite number of tangencies, which may be seen if one blows up a filled portion of the symbol plane. However since we are interested in only short orbits, a finite number of tangencies may be used. Basically the longer the orbit the more tangencies are needed for pruning.

When trying to calculate the backward sequences for a general chaotic system, one must use the current position and guess the previous position, based on a backwards iterate. However in a case, like the logistic diagram, we have that the current position,

when iterated backwards, has at least 2 possible paths, for the first iterate, and 2^n possible paths for the nth backwards iterate. This is to be expected since in chaotic systems one can at best hope to predict the short time changes in the system.

2.3 1D Approximation

In some cases, the 2 dimensional maps may take on strong resemblance to 1 dimensional maps. For this case there may be approximations made to use only the basic information about a system for pruning. The 2 dimensional map that we are observing does have some relation to the 1 dimensional map. In fact, with the rod and gear system we have maps that are fairly well approximated by the 1D maps. So using a method developed by Zheng [4] we use the forward foliations of the tangencies as initial boundaries, greatly reducing the number of allowed orbits. Effectively, if one observes the symbol plane this procedure would consist of looking at only the α coordinates, leaving the boundaries uncapped so that the forbidden zones are blocks of symbol space not allowed on a single axis. This approximation works very well for short orbits, however when put to practical use for larger orbits, it may still permit several orbits that should not exist. This would be due to the 2 dimensional approximation breaking down for longer orbits. What is needed now is the expansion that leads the system back into the 2D case, where we have to consider the FFZ's. Since for any analysis we expect that the α terms are the same, we have to expand our analysis to include the backward sequences. When we do this we have the regions that

were once inadmissible are now admissible within a small boundary. From this we know that adding conditions to the restrictions we can slowly test the admissibility of the larger orbits. Ultimately, this process will end with a 2 dimensional analysis for a 2 dimensional map, but as mentioned before, shorter orbits are all that is needed for the scope of this project. A fair analogy for this would be the Taylor series expansion, since for small values of the argument we can simply consider the linear term, but for larger values of the argument we need to expand to the quadratic term and so on. For higher dimensional analysis it is possible that the maps will go to higher dimension as well. However since symbolic dynamics still has not been properly defined for 3D systems, one cannot use symbolic dynamics for higher dimensional systems directly, and approximations would have to be made. One must also be careful in the accuracy of the kneading sequence, usually the researcher determines some cutoff where any further contribution to the kneading sequence is negligible so the sequence is truncated. If the orbit being searched is longer than the truncation, the results of any admissibility test are questionable. Fortunately, in many cases the truncation is trivial due to a repetition found in the kneading sequence. However, this may not always be the case.

2.4 Admissibility

Now, with the boundary conditions understood, one can produce an exhaustive list of the allowed orbits with a minimal probability of producing orbits that are inadmissible. I say "minimal probability" since there are several fundamental forbidden zones of varying sizes in the symbol space. By removing the tangencies as described above we are removing the largest FFZ's from the symbol plane and thus removing the largest amounts of space available for an orbit to possibly occupy. Since there are an infinite number of FFZ's catching them all is fruitless since for finding the lower period orbits we are required to use only the larger FFZ's. In fact, we found that for up to period 8, for table II parameters, see the end of this chapter, the only restrictions required were for the 1D map. Other restrictions did not make any difference in the results. So, we find that the orbits up to at least period 17 are valid with only a few FFZ's. Since the symbolic sequencing was not followed beyond period 17, higher periods were not verified for correctness. Now, with the reduced number of allowed orbits we find that the orbit hunting is made much easier, reducing the number of possibilities from N! to around 2^N, which for large orbits still is not great, but is still manageable.

Some of the benefit of finding the symbolic orbits is the information about a system that may be established before any root finding calculations, such as the winding number, which was defined in chapter 1. This can be determined by taking

the ratio of the number of R's and N's in an orbit to the number of symbols in the orbit.

Basically, what has been described here is a destructive procedure for determining allowed orbits. First you assume all permutations and combinations (except the repetitive cyclic permutations) are allowed and then from that list perform all shifts possible and remove any combination that violates the kneading ordering, or enters FFZ's. The remaining list, by deduction, must be all the allowed orbits. There is also a constructive method of creating the list, in short it uses already existing orbits and the idea of a rotation interval. This method may be found in the Thesis [55].

If one considers time requirements for symbolic orbit hunting, both methods may potentially be useful. The constructive method works well, especially if not all shorter period orbits have been found. However this method begins to break down for large periods of orbit since the original created list will have N! allowed combinations for testing. It is when the destructive method becomes too time consuming that the constructive method may be used, where one only takes a short list of allowed orbits, and generates the longer list of higher period orbits. Since we are only dealing with orbits with a period less than 18 the destructive method works fairly efficiently.

2.5 Prime Cycles

Another important fact to consider with orbit hunting is the single traversal of a cycle, also referred to as a prime cycles. These are orbits that do not repeat themselves or,

more formally, may not be broken into identical subsets. For example RN is a prime cycle, but RNRN is not a prime cycle, but is rather made up of a repeat of the RN subset. It is defined as a non-repeating symbol string of n_p symbols [52] Where n_p is the period of the orbit. A cyclic permutation of a prime cycle is the same prime cycle. An orbit hunter should be aware of this since it may reduce the amount of work done in the future.

With the methods for finding the symbolic orbits defined, from here we need to calculate the position of these allowed orbits in the Poincare section.

2.6 Numerical Orbit Hunting

Once the allowed orbits have been established, we are now left with the task of calculating the actual positions of the orbital trajectories. The symbolic approach to this method will be discussed here. The method that was predominantly used in the orbit searches for this project was a method developed by Divakar Viswanath. [70]

There are a few methods that may be used to find the numerical values for the short orbits, these will be discussed a in the next chapter. These methods can become tedious for larger orbits so there have been several methods created to make this work a lot easier. To find the period 1 orbits, assuming they exist, one only needs to look at the return map for the system. Where the diagonal crosses the attractor, as seen in figure 1.12 and 1.13 we find a fixed point as described in chapter 1, these are the period 1 values x component. Comparing them to the Poincare section we easily find

the y component. If the Poincare section is of higher dimension more testing may be required. With the period 1 orbits defined, one takes linear combinations of them as initial guesses for the larger orbits. This method may also work for higher order orbits and again will produce seeds that may be used for further symbolic hunting. [57, 26]

The first method operates on the assumption that all the smaller orbits have been found. Take for example the values found in table II at the end of this chapter, we would find the period 1 orbits N and R, the base orbits. From here we could come up with a fairly good initial guess for period 2 RN since it depends solely on the base orbits. For period 2 orbits a similar method may be used, In these cases we use the second return map $x = f^{(2)}(x)$ and the diagonal on this map crosses the period 1 orbits and the period 2 orbits. [26] This method may continue for the higher order orbits. Once these are found the calculations for further orbits are readily available. Given the metric values for any symbol in each trajectory, finding even the most obscure trajectory is greatly simplified since each letter in any of the above orbits may be broken away from the rest of the orbit and also used as an initial guess. We see there are fairly dramatic differences between the allowed orbits with table I and table II. Table I is very restricted and does not allow period 1 orbits, as seen on the first return map in figure 1.11. Since the diagonal does not contact the attractor on the first return map we know there are no period 1 orbits. So the combinations of

smaller orbits to determine the larger orbits is not ideal. Another method is needed to simplify the task, so we turn to Divakar's method. This method consisted of comparing the hunted orbit to existing orbits, but in a slightly different manner. If we were looking for an orbit RNNLN, Divakar's method would suggest we use an orbit that contains RNNL as it would compare the hunted orbit with 4 or more matching points.

```
RNNLNRNNLN ...
RNNLRNNLRN ...
```

Since that orbit, or any cyclic permutation thereof, does not exist we then back down to a period 3 orbit that might also match with 4 or more symbols. This orbit is RNN.

```
RNNLNRNNLN ...
RNNRNNRNNR ...
```

Now with this match we take the Poincaré section coordinate for the symbol R as the initial guess for the R in RNNLN. Now we take a cyclic permutation of RNNLN to NNLNR, and repeat the test, first looking for an orbit that matches at 5 points and so on. For our purposes this method was found to work well, however it was slow, especially for longer orbits, thus a modified method was used, where one could combine the 2 methods described above.

The modified method uses the linear combinations of shorter orbits method with Divakar's method. As an example we can take the orbit RRLLN, which exists in table

I and II and would take the first three coordinates to be the period 3 orbit RRL. Then, we would use the period 2 orbit LN for the remaining coordinates. This method does not always work well. Often trends were found between these 2 methods that would allow for rapid orbit finding with the larger orbits, such as the use of orbit RRRLL in table 2.1 for any initial guess containing 'LL'. These methods do not guarantee a successful return, they are simply methods of improving the guess, and have a much higher probability of success over random guesses. The advantage of Divakar's method is that it requires the use of some of the already known orbits. So that if some of the smaller period orbits are still missing, it does not prevent the hunt from continuing for larger period orbits. This is useful if an orbit is particularly difficult to find, which may indicate that the orbit is very unstable and thus we do not need to find it, this will be described in detail in chapter 3.

When dealing with smaller period orbits, say ten or less, the above mentioned methods work well enough but are still time consuming. Methods using automated search techniques had to be developed. This automated search method had to be broad in its search spectrum but specific enough to be efficient. This was done by selecting points that were known to be along the attractor up to the 14th decimal place.

To find initial guesses on the attractor, one needs to run a Poincaré iterate of the period of orbit being investigated, with a large repeat of the orbit count, say 30

to 50. To ensure the broad spectrum is being covered, one needs to make a wide range of starting points for the Poincaré iterate. This is fairly easy to do, since we know the shape and range of the attractor we could cover the attractor with a grid of points with two decimal accuracy and the system would evolve appropriately for each initial guess. With the consideration that transient behaviour is expected for the system while it is evolving from a two decimal initial guess to a point on the attractor, we can easily obtain a data set that is full of points as they evolve along the given differential equations and rest on the chaotic attractor. These points are all suitable guesses for the multishoot program and since this process may produce millions of initial guesses, probability theory indicates that many of these points will end up being convergent to some orbit of the same period as that of the Poincaré iterate. All we really need is around n^2 as many initial guesses as we have expected orbits, with n being the number of expected orbits. Say for a period 12 orbit from table II we expect around 600 orbits, so we need to have at least $600^2 = 360000$ initial guesses. This process would have to be repeated several times for the different periods of orbit for a given set of parameters. This is another method that would be next to impossible without computers. This method was much more efficient than the program that uses 2 decimal initial guesses for the multishoot method. This would take around 3 years to complete a period 5 search. Higher decimal accuracy on the initial guesses would result in an exponentially longer search time. After about eight hours most of the

needed orbits were found. Furthermore, the larger the period of orbit, the more orbits could be found in a shorter period of time. Difficulties with this method are twofold. The first difficulty is that the hunter cannot be entirely sure that they have found all the orbits belonging to a given period. The second difficulty is finding the exact same orbit several times, but this simply requires filtering to remove the repeated orbits. As for the issue of not finding all of the allowed orbits in an automated search, it has been found that the automated searches find the least unstable of the unstable orbits faster than the very unstable orbits. In the end the following tables 2.1, 2.2, and 2.3 were constructed for the particular parameter values used.

2.7 Conclusion

Now that we have established a method of determining a maximal list of allowed orbits, and a prescription for calculating the phase space location of these orbits based solely on the symbolic dynamics, we are left with the need for algorithms to find the numerical values of the allowed symbolic orbits on the Poincaré section. This will be discussed in detail in chapter 3. The methods discussed above are generic to the level of 2 dimensional maps. Extending these ideas to 3 dimensional maps is an open area for research, but for now we can easily use Poincaré sections to reduce the mappings to 2 dimensions, allowing the use of the above methods.

Table 2.1: Allowed Orbits for Q = 0.76

n	Allowed Orbits
2	RL NL
3	RRL RLN
4	RLLN
5	RRRLL RRLRL RRLNL RRLLN RLRLN RLNLN
6	RRRLRL RRLRLN RRLNRL RLRLLN RLLNLN
7	RRRLRLL RRRLLNL RRLRLRL RRLRLLN RRLLNRL RRLLNLN RLRLRLN RLRLNLN RLNRLLN RLNLNLN
8	RRRLRLRL RRRLRLNL RRRLLNRL RRLRRLRL RRLRLRLN RRLRLNRL RRLRLNLN RRLNRLRL RRLNLNRL RRLLNRLN RLRLRLLN RLRLNRLN RLRLLNLN RLNRLNLN RLLNLNLN
9	RRRLRLRRL RRRLRLRLL RRRLRLNRL RRRLRLLNL RRRLLNRLL RRRLLNLNL RRLRRLNRL RRLRLRRLN RRLRLRLRL RRLRLRLLN RRLRLNRLN RRLRLLNRL RRLRLLNLN RRLNRLNRL RRLNLNRLL RRLNLNLNL RRLLNRLRL RRLLNRLLN RRLLNLNRL RRLLNLNLN RLRLRLRLN RLRLRLNLN RLRLNRLLN RLRLNLNLN RLRLLNRLN RLNRLLNLN RLNLNRLLN RLNLNLNLN
10	RRRLRLRRLL RRRLRLRLRL RRRLRLRLNL RRRLRLNRLL RRRLRLNLNL RRRLRLLNRL RRRLLRRLNL RRRLLNRRLL RRRLLNRLRL RRRLLNRLNL RRRLLNLNRL RRLRRLRLRL RRLRRLLNRL RRLRLRRLLN RRLRLRLRLN RRLRLRLNRL RRLRLRLNLN RRLRLNRLRL RRLRLNRLLN RRLRLNLNLN RRLRLLNRLN RRLNRLRLRL RRLNRLLNRL RRLNLNRRLL RRLNLNRLRL RRLNLNRLNL RRLNLNLNRL RRLLNRLRLN RRLLNRLNRL RRLLNRLNLN RRLLNLNRLN RLRLRLRLLN RLRLRLNRLN RLRLRLLNLN RLRLNRLNLN RLRLNRLLNL RLRLNLNRLN RLRLLNRLLN RLRLLNLNLN RLNRLNRLLN RLNRLNLNLN RLLNRLLNLN RLLNLNLNLN
11	RRRLRLRRRLL RRRLRLRRLRL RRRLRLRRLNL RRRLRLRLRRL RRRLRLRLRLL RRRLRLRLNRL RRRLRLRLLNL RRRLRLNRRLL RRRLRLNRLRL RRRLRLNRLNL RRRLRLNLNRL RRRLRLLNRLL RRRLRLLNLNL RRRLLNRRLRL RRRLLNRLRRL RRRLLNRLRLL RRRLLNRLNRL RRRLLNRLLNL RRRLLNLNRLL RRRLLNLNLNL RRLRRLRRLRL RRLRRLRLNRL RRLRRLNRLRL RRLRRLNLNRL RRLRLRRLRLN RRLRLRRLNRL RRLRLRRLNLN RRLRLRLRRLN RRLRLRLRLRL RRLRLRLRLLN RRLRLRLNRLN RRLRLRLLNRL RRLRLRLLNLN RRLRLNRRLLN RRLRLNRLRLN RRLRLNRLNRL RRLRLNRLNLN RRLRLNLNRLN RRLRLLNRLRL RRLRLLNRLLN RRLRLLNLNRL RRLRLLNLNLN RRLNRRLLNRL RRLNRLRLNRL RRLNRLNRLRL RRLNRLNLNRL RRLNLNRLRLL RRLNLNRLNRL RRLNLNRLLNL RRLLNRLRLRL RRLLNRLRLLN RRLLNRLNRLN RRLLNRLLNRL RRLLNRLLNLN RRLLNLNRLRL RRLLNLNRLLN RRLLNLNLNRL RRLLNLNLNLN RLRLRLRLRLN RLRLRLRLNLN RLRLRLNRLLN RLRLRLNLNLN RLRLNRLRLLN RLRLNRLNRLN RLRLNRLLNLN RLRLNLNRLLN RLRLNLNLNLN RLRLLNRLNLN RLRLLNLNRLN RLNRLNRLNLN RLNRLLNRLLN RLNRLLNLNLN RLNLNRLLNLN RLNLNLNRLLN RLNLNLNLNLN

Table 2.2: Allowed Orbits for Q = 1.2577

n	Allowed Orbits
1	R N
2	RN RL NL
3	RRN RRL RNN RLN
4	RRRN RRRL RRNN RRLN RNNN RLLN
5	RRRRN RRRRL RRRNN RRRLN RRNRN RRNRL RRNNN RRLRL RRLLN RNRNN RNRLN RNNRL RNNNN RLRLN RLNNN RLNLN
6	RRRRNN RRRRLN RRRNRN RRRNRL RRRNNN RRRLRL RRRLLN RRNRRL RRNRNN RRNRLN RRNNRN RRNNRL RRNNNN RRLRNN RRLRLN RRLNRN RRLNRL RRLNNN RRLNLN RNRNNN RNRLLN RNNRLN RNNNNN RLRLLN RLLNNN RLLNLN
7	RRRRNRN RRRRNRL RRRRNNN RRRRLRL RRRRLLN RRRNRRN RRRNRRL RRRNRNN RRRNRLN RRRNNRN RRRNNRL RRRNNNN RRRLRRN RRRLRRL RRRLRNN RRRLRLN RRRLNRN RRRLNRL RRRLNNN RRRLNLN RRNRRNN RRNRRLN RRNRNRN RRNRNRL RRNRNNN RRNRLRL RRNRLLN RRNNRRL RRNNRNN RRNNRLN RRNNNRN RRNNNRL RRNNNNN RRRLRRNN RRRLRRLN RRRLRLRL RRRLRLLN RRRLNRRN RRRLNRRL RRRLNRNN RRRLNRLN RRRLLRNN RRRLLNRN RRRLLNRL RRRLLNNN RRRLLNLN RRNRRNRN RRNRRNRL RRNRRNNN RRNRRLRL RRNRRLLN RRNRNRRL RRNRNRNN RRNRNRLN RRNRNNRN RRNRNNRL RRNRNNNN RRNRLRRL RRNRLRNN RRNRLRLN RRNRLNRN RRNRLNRL RRNRLNNN RRNRLNLN RRNNRRLN RRNNRNRN RRNNRNRL RRNNRNNN RRNNRLRL RRNNRLLN RRNNNRRL RRNNNRNN RRNNNRLN RRNNNNRN RRNNNNRL RRNNNNNN RRLRRLRL RRLRRLLN RRLRNNRN RRLRNNRL RRLRLRNN RRLRLRLN RRLRLNRN RRLRLNRL RRLRLNNN RRLRLNLN RRLNRNRN RRLNRNRL RRLNRNNN RRLNRLRL RRLNRLLN RRLNNNRN RRLNNNRL RRLNLRNN RRLNLNRN RRLNLNRL RRLNLNNN RRLNLNLN RRLLNRNN RRLLNRLN RNRNRNNN RNRNRLLN RNRNNRNN RNRNNRLN RNRNNNNN RNRLRLLN RNRLNRNN RNRLNRLN RNRLLNNN RNRLLNLN RNNRNNRL RNNRNNNN RNNRLRLN RNNRLRLL RNNRLNRL RNNRLNNN RNNRLNLN RNNRLLNL RNNNRLLN RNNNNRLN RNNNNNNN RLRLRLLN RLRLNRLN RLRLLNNN RLRLLNLN RLNRLNNN RLNRLNLN RLLNNNNN RLLNLNNN RLLNLNLN
8	RRRRNRRN RRRRNRRL RRRRNRNN RRRRNRLN RRRRNNRN RRRRNNRL RRRRNNNN RRRRLRRN RRRRLRRL RRRRLRNN RRRRLRLN RRRRLNRN RRRRLNRL RRRRLNNN RRRRLNLN RRRNRRRL RRRNRRNN RRRNRRLN RRRNRNRN RRRNRNRL RRRNRNNN RRRNRLRL RRRNRLLN RRRNNRRN RRRNNRNL RRRNNNRN RRRNNNNL RRRLRRNN RRRLRRLN RRRLRLRL RRRLRLLN RRRLNRRN RRRLNRRL RRRLNRNN RRRLNRLN RRRLLRNN RRRLLNRN RRRLLNRL RRRLLNNN RRRLLNLN RRNRRNRN RRNRRNRL RRNRRNNN RRNRRLRL RRNRRLLN RRNRNRRL RRNRNRNN RRNRNRLN RRNRNNRN RRNRNNRL RRNRNNNN RRNRLRRL RRNRLRNN RRNRLRLN RRNRLNRN RRNRLNRL RRNRLNNN RRNRLNLN RRNNRRLN RRNNRNRN RRNNRNRL RRNNRNNN RRNNRLRL RRNNRLLN RRNNNRRL RRNNNRNN RRNNNRLN RRNNNNRN RRNNNNRL RRNNNNNN RRLRRLRL RRLRRLLN RRLRNNRN RRLRNNRL RRLRLRNN RRLRLRLN RRLRLNRN RRLRLNRL RRLRLNNN RRLRLNLN RRLNRNRN RRLNRNRL RRLNRNNN RRLNRLRL RRLNRLLN RRLNNNRN RRLNNNRL RRLNLRNN RRLNLNRN RRLNLNRL RRLNLNNN RRLNLNLN RRLLNRNN RRLLNRLN RNRNRNNN RNRNRLLN RNRNNRNN RNRNNRLN RNRNNNNN RNRLRLLN RNRLNRNN RNRLNRLN RNRLLNNN RNRLLNLN RNNRNNRL RNNRNNNN RNNRLRLN RNNRLRLL RNNRLNRL RNNRLNNN RNNRLNLN RNNRLLNL RNNNRLLN RNNNNRLN RNNNNNNN RLRLRLLN RLRLNRLN RLRLLNNN RLRLLNLN RLNRLNNN RLNRLNLN RLLNNNNN RLLNLNNN RLLNLNLN

Table 2.3: Additional Allowed Orbits for Q = 1.2577

n	Allowed Orbits
9	RRRRRRRRN RRRRRRRRL RRRRRRRNN RRRRRRRLN RRRRRRNRN RRRRRRNRL RRRRRRNNN RRRRRRLRL RRRRRRLLN RRRRRNRRN RRRRRNRRL RRRRRNRNN RRRRRNRLN RRRRRNNRN RRRRRNNRL RRRRRNNNN RRRRRLRRN RRRRRLRRL RRRRRLRNN RRRRRLRLN RRRRRLNRN RRRRRLNRL RRRRRLNNN RRRRRLNLN RRRRNRRRN RRRRNRRRL RRRRNRRNN RRRRNRRLN RRRRNRNRN RRRRNRNRL RRRRNRNNN RRRRNRLRL RRRRNRLLN RRRRNNRRN RRRRNNRRL RRRRNNRNN RRRRNNRLN RRRRNNNRN RRRRNNNRL RRRRNNNNN RRRLRRRLN RRRLRRNRN RRRLRRNRL RRRLRRNNN RRRLRLRRN RRRLRLRRL RRRLRLRNN RRRLRLRLN RRRLRLNRN RRRLRLNRL RRRLRLNNN RRRLRLNLN RRRLLRRNN RRRLLRNRN RRRLLRNRL RRRLLRNNN RRRLLRNLN RRRLLNRRN RRRLLNRRL RRRLLNRNN RRRLLNRLN RRRLLNNRN RRRLLNNRL RRRLLNNNN RRRLLNLRN RRRLLNLNN RRRLLNLNL RRRNRRRNN RRRNRRRLN RRRNRRNRN RRRNRRNRL RRRNRRNNN RRRNRRLRL RRRNRRLLN RRRNRNRRN RRRNRNRRL RRRNRNRNN RRRNRNRLN RRRNRNNRN RRRNRNNRL RRRNRNNNN RRRNRLRRN RRRNRLRRL RRRNRLRNN RRRNRLRLN RRRNRLNRN RRRNRLNRL RRRNRLNNN RRRNRLNLN RRRNNRRRL RRRNNRRNN RRRNNRRLN RRRNNRNRN RRRNNRNRL RRRNNRNNN RRRNNRLRL RRRNNRLLN RRRNNNRRN RRRNNNRRL RRRNNNRNN RRRNNNRLN RRRNNNNRN RRRNNNNRL RRRNNNNNN RRRLRRRLN RRRLRRNRN RRRLRRNRL RRRLRRNNN RRRLRLRRN RRRLRLRRL RRRLRLRNN RRRLRLRLN RRRLRLNRN RRRLRLNRL RRRLRLNNN RRRLRLNLN RRRLNRRNN RRRLNRRLN RRRLNRNRN RRRLNRNRL RRRLNRNNN RRRLNRLRL RRRLNRLLN RRRLNNNRN RRRLNNNRL RRRLNLRNN RRRLNLNRN RRRLNLNRL RRRLNLNNN RRRLNLNLN RRRLLNRRN RRRLLNRRL RRRLLNRNN RRRLLNRLN RRNRRNRRL RRNRRNRNN RRNRRNRLN RRNRRNNRN RRNRRNNRL RRNRRNNNN RRNRRLRRL RRNRRLRNN RRNRRLRLN RRNRRLNRN RRNRRLNRL RRNRRLNNN RRNRRLNLN RRNRNRRNN RRNRNRRLN RRNRNRNRN RRNRNRNRL RRNRNRNNN RRNRNRLRL RRNRNRLLN RRNRNNRRL RRNRNNRNN RRNRNNRLN RRNRNNNRN RRNRNNNRL RRNRNNNNN RRNRLRRNN RRNRLRRLN RRNRLRLRL RRNRLRLLN RRNRLNRRL RRNRLNRNN RRNRLNRLN RRNRLLNRN RRNRLLNRL RRNRLLNNN RRNRLLNLN RRNNRRNNN RRNNRRLRL RRNNRRLLN RRNNRNRRL RRNNRNRNN RRNNRNRLN RRNNRNNRN RRNNRNNRL RRNNRNNNN RRNNRLRRL RRNNRLRNN RRNNRLRLN RRNNRLNRN RRNNRLNRL RRNNRLNNN RRNNRLNLN RRNNNRRLN RRNNNRNRN RRNNNRNRL RRNNNRNNN RRNNNRLRL RRNNNRLLN RRNNNNRRL RRNNNNRNN RRNNNNRLN RRNNNNNRN RRNNNNNRL RRNNNNNNN RRLRRLRNN RRLRRLRLN RRLRRLNRN RRLRRLNRL RRLRRLNNN RRLRRLNLN RRLRNNRNN RRLRNNRLN RRLRLRRLN RRLRLRLRL RRLRLRLLN RRLRLNRNN RRLRLNRLN RRLRLLRNN RRLRLLNRN RRLRLLNRL RRLRLLNNN RRLRLLNLN RRLNRRLLN RRLNRNRNN RRLNRNRLN RRLNRNNRN RRLNRNNRL RRLNRNNNN RRLNRLRNN RRLNRLRLN RRLNRLNRN RRLNRLNRL RRLNRLNNN RRLNRLNLN RRLNNNRNN RRLNNNRLN RRLNLNRNN RRLNLNRLN RRLLRNNRN RRLLRNNRL RRLLNRNRN RRLLNRNRL RRLLNRNNN RRLLNRLRL RRLLNRLLN RRLLNNNRN RRLLNNNRL RRLLNLRNN RRLLNLNRN RRLLNLNRL RRLLNLNNN RRLLNLNLN RNRNRNRNN RNRNRNRLN RNRNRNNNN RNRNRLRNN RNRNRLRLN RNRNRLNNN RNRNRLNLN RNRNNRNNN RNRNNRLLN RNRNNNRNN RNRNNNRLN RNRNNNNNN RNRLRLRNN RNRLRLRLN RNRLRLNNN RNRLRLNLN RNRLNRNNN RNRLNRLLN RNRLNLRNN RNRLNLNNN RNRLNLNLN RNRLLNRNN RNRLLNRLN RNNRNNRLN RNNRNNNRL RNNRNNNNN RNNRLRLRL RNNRLRLNL RNNRLRLLN RNNRLNRLN RNNRLNRLL RNNRLNLNL RNNRLLNRL RNNRLLNNN RNNRLLNLN RNNNRNNNN RNNNRLRLN RNNNRLNNN RNNNRLNLN RNNNNRLRL RNNNNRLLN RNNNNNRLN RNNNNNNNN RLRLRLRLN RLRLRLNNN RLRLRLNLN RLRLNRLLN RLRLNNNNN RLRLNLNNN RLRLNLNLN RLRLLNRLN RLNRLLNNN RLNRLLNLN RLNNNRLLN RLNLNRLLN RLNLNLNNN RLNLNLNLN

Chapter 3

Periodic Orbit Theory

3.1 Introduction

3.1.1 Evolution

Periodic Orbit Theory, as the name suggests studies the set of periodic orbits that exist in a system. As previously mentioned, a set of linear differential equations may involve a singular solution or a collective solution, as in a periodic orbit, such as the case of the simple harmonic oscillator. In nonlinear differential equations, the solution set usually involves an infinite number of periodic orbits.

Periodic solutions occur when the system exhibit properties like $\mathbf{x} = f^n(\mathbf{x})$ for $n < \infty$, where n is the period of the orbit, as mentioned in chapter 2. If the function f was to take on the operator form what we are actually looking for are eigenvectors of a system.

3.1.2 Trajectory Displacements

When looking at a chaotic system one generally tries to find Lyapunov exponents and fractal dimensions since, as mentioned in chapter 1, these are indicative of a chaotic system. These properties also happen to be properties that are coordinate independent, and thus a topological invariant.

As a trajectory evolves it takes on the form $x(t) = f^t(x_0)$, where x_0 is the initial position in phase space, and f^t is a time evolution operator. If we linearize this equation by moving the system through a very small section of time, we have $x_i(t) + \delta x_i(t)$, and the evolution of the system would be determined by the small change $\delta x_i(t)$. This small change is defined by;

$$\delta x_i(t) = \frac{\delta x_i(t)}{\delta x_{0j}} \delta x_{0j}$$

where the fraction is simply, J^t, the time evolved Jacobian matrix of the system.

As points on the manifold flow through a trajectory they distort the region of space around them. The deformation is commonly understood to be some initial position on the manifold that undergoes a small displacement $\delta x(0)$, and the flow transport is $\delta x(t)$ along a trajectory $x(x_0; t) = f^t(x_0)$. Now, considering the equation of variations, this becomes $x_i(x_0, t) + \delta x_i(x_0, t)$ which then leads to, via a Taylor series expansion;

$$\dot{x} + \delta\dot{x} \approx v_i(x) + \sum_j \frac{\partial v_i}{\partial x_j} \delta x_j \tag{3.1.1}$$

Taking into consideration the equation of flow, $v(x) = \dot{x}$, we are left with $\delta\dot{x} = \sum_j \frac{\partial v_i}{\partial x_j}\delta x_j = \sum_j A_{ij}\delta x_j$ with $A_{ij}(x) = \frac{\partial v_i(x)}{\partial x_j}$ where A_{ij} is the matrix of variations.

3.2 Nonlinear Flows

The equation of motion for the rod and gear system, due to large oscillation capability of the rod, is a nonlinear flow. Thus, we have a lot of work to do to the system before we can calculate anything meaningful from it. We still start with equations

$$\dot{x}_i = v_i(x) \quad \dot{\delta x}_i = A_{ij}(x)\delta x_j \tag{3.2.1}$$

where A_{ij} describes the instantaneous rate of shearing of the infinitesimal neighbourhood of x by the flow [52]. It is actually this matrix which is critical for the proper analysis of the system since the eigenvalues of this matrix determine the local behaviour of neighbouring trajectories. For maps this is referred to as the local stability, since this measures the reaction of the system to a perturbation. If the eigenvalue is larger than 1, then there exists a perturbation that will grow over a given time interval [37]. It is here that we actually divide the system into two classes of trajectory, hyperbolic for the eigenvalues greater than 1, and convergent for the eigenvalues less than or equal to 1.

This leads us through the topological analysis, which is not immediately applicable for real problems. However if we were to linearize the flow using a Taylor series

expansion:

$$f_i^t(x_0 + \delta x) = f_i^t(x_0) + \frac{\partial f_i^t(x_0)}{\partial x_{0j}} \delta x_j + \dots \tag{3.2.2}$$

, then the Jacobian matrix comes into play. We see that the Jacobian is instrumental in analyzing the deformation of a neighbourhood for finite time, since the Jacobian is just the functional rate of change over each of the parameters. This is directly analogous to the matrix of variations. Where the deformations are determined by the classification of the eigenvalues [52].

Since the variation equations are linear, we are left with a formal definition of the Jacobian matrix

$$J_{ij}^t(x_0) = \mathbf{T} \left[\exp^{\int_0^1 d\tau \mathbf{A}(x(\tau))} \right]_{ij} \tag{3.2.3}$$

for time ordered integration. After a Taylor series expansion around a fixed point of x we find that equation 3.2.3 reduces to

$$\mathbf{J}^t(x_q) = e^{\mathbf{A}t} \tag{3.2.4}$$

The Euler limit definition of the exponential may be used now, so we may expect a more convenient computational form of the Jacobian

$$J^t(x_q) = \lim_{m \to \infty} \left(1 + \frac{t}{m} A \right)^m \tag{3.2.5}$$

Since $\mathbf{A}$ is not constant, we may further reduce the equation to discrete time steps over a reversed product

$$J^t(x_q) = \lim_{m \to \infty} \prod_{n=m}^{1} \left(1 + \frac{t}{m} A(x_n) \right) \tag{3.2.6}$$

however, if we were to take a usual trick of letting t/m become Δt and the limit becomes $\lim_{\Delta t \to 0}$, we have successfully linearized the flow. Now to further simplify things we consider the nature of the time ordered product along the flow

$$J^{t+t'}(x_0) = J^{t'}(x(t))J^{t}(x_0) \tag{3.2.7}$$

which becomes, to first order,

$$\frac{d}{dt}J^{t}(x) = A(x)J^{t}(x) \tag{3.2.8}$$

Since 3.2.8 is linear in A(x), this is also known as first variation, meaning it is a function that may be readily programmed.

Now, what we are really interested in is finding the periodic orbits, as they are the topological invariants. So, we need to calculate the stability matrices, which are now just successive multiplications of the Jacobian matrices, evaluated at the points along the trajectory. Considering that periodic points have a fixed period of repeat, all we need to do for a proper length of multiplication is to evaluate the Jacobians at each point in the periodic trajectory that falls on the Poincaré section and multiply them together. Following from equation 3.2.7 we see that the stability of any full repeat of a prime cycle will simply be the Jacobian of the prime cycle taken to the power of the number of repeats. This does not lead to any new information since the value of this structure will simply return the same prime Jacobian, by the periodicity assumption. The usefulness of this is reinforced by the fact that the point at which

you evaluate the position of the Jacobian matrix may be any point along the orbit. Now, with the stability matrix determined we find the stability eigenvalues, and then we rank them, as seen in earlier in this section.

$$|\Lambda| = \begin{cases} > 1, & \text{Expanding, Hyperbolic;} \\ = 1, & \text{Marginal, Stable;} \\ < 1, & \text{Contracting.} \end{cases} \tag{3.2.9}$$

with Λ being the eigenvalue. Another important fact that we need to consider is that the periodic orbits may be considered stable if all of the stability eigenvalues are less than 1, in which case the overall orbit has no contribution to the final calculations due to the hyperbolicity assumption; The system is nowhere convergent.

3.3 Periodic Orbit Calculations

The first step in solving the dynamical averages for the system is to find numerical solutions to the periodic orbits.

First of all, we are aware of the definition of the periodic orbits $f^{t+T_p}(x) = f^t(x)$, where T_p is the period of orbit of the prime cycle. When looking at maps, the cycle crosses the Poincaré section n_p times where each crossing is at a fixed point in the Poincaré section return map. Therefore it is common to refer to the crossings as fixed points.

As is common in calculus, when we are looking for fixed points we really need the roots of the equations. With numerical problems one must solve numerically for these roots. The first known method is Newton's iterative procedure for finding the

roots of a one dimensional function f(x)

$$x_1 = x_0 - f(x_0)/f'(x_0). \tag{3.3.1}$$

This method works well on simple one dimensional functions with only a few widely spaced roots. Almost any initial guess is a good guess for such functions. However, as the functions become more complicated, better and better initial points are needed so that the proper roots are found.

Newton's procedure is great for one dimensional functions but with the periodic orbits we are trying to find the roots of some very complicated functions with several roots. This situation requires an extremely good initial guess, otherwise the roots that will be found will be random.

To rectify this situation one needs to use a more sophisticated root finding system. There is at least one such method available, the multipoint shooting method, henceforth referred to as multishoot. This method also requires a decent initial guess, however, as long as the phase partitions are good, a lower accuracy guess will do. We began with a definition of the periodic function [52]

$$F\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 - f(x_n) \\ x_2 - f(x_1) \\ \vdots \\ x_n - f(x_{n-1}) \end{pmatrix} \tag{3.3.2}$$

and we are left with a function and a procedure to solve for the roots. The derivative of the given function is an $[n \times n]$ matrix. We are more interested in the inverse of

this matrix to solve the Newton-Raphson iterative equation. Again, in this situation we are expecting a minimal solution due to the difference in equation 3.3.2. What becomes more interesting is the case when we are dealing with higher dimensional flows. Many nonlinear dynamic problems are flows and may be studied as maps using a Poincaré section, this is the case for the rod and gear system. We have an autonomous flow $\dot{x} = v(x)$ with a corresponding Jacobian Matrix J which may be defined by linearizing equation 3.2.8:

$$\dot{\mathbf{J}} = \mathbf{AJ}, \tag{3.3.3}$$

where A is again the matrix of variations. Both the flow $\dot{x} = v(x)$ and equation 3.3.3 are solved simultaneously using the same routines by compounding them into one matrix. Integrating the initial conditions on the Poincaré map until a later crossing of the Poincaré map and linearizing around the flow we get

$$f(x') \approx f(x) + \mathbf{J}(x' - x) \tag{3.3.4}$$

Here we know that x, $f(x)$ and x' are on the Poincaré section, however it is most likely that f(x') is not on the Poincaré section. This is due to the fact that **J** corresponds to a specific integration time, and has no relation to the choice of Poincaré section.[52] To find a fixed point of the flow near a starting guess x, we have to solve the linearized equation

$$(\mathbf{I} - \mathbf{J})(x' - x) = -(x - f(x)) = -\mathbf{F}(x) \tag{3.3.5}$$

This problem may be solved by using a linear equation that demands that x' be on the Poincaré section, and by adding a small vector along the flow. Take for example the equation

$$(x - x_0) \cdot \mathbf{a} = 0 \tag{3.3.6}$$

which claims that the Poincaré section is a plane, that $\mathbf{a}$ is a vector that is perpendicular to the Poincaré section, and that x_0 is a point within the section. This leads to the Poincaré bound equation

$$\begin{pmatrix} 1 - \mathbf{J} & v(x) \\ \mathbf{a} & 0 \end{pmatrix} \begin{pmatrix} x' - x \\ \delta T \end{pmatrix} = \begin{pmatrix} -\mathbf{F}(\mathbf{x}) \\ 0 \end{pmatrix} \tag{3.3.7}$$

where $v(x)\delta T$ is the small vector addition to the flow. This equation may easily be generalized to n dimensions if one takes $1-\mathbf{J}$ to be $I-\mathbf{J}$ and a, v to be diagonal $[n \times n]$ matrices. The vector we are interested in solving is the left side vector of differences. Therefore, an inversion of teh matrix would be necessary to find the computational solution. [52] The inverted matrix will be seen in chapter 4.

3.4 Statistics

We are all familiar with the concept of integral averages from statistical mechanics. In that case all we are doing is calculating the value of the observable over all of the system space. Here, we are only concerned with the manifold space, and thus we have;

$$\langle a \rangle(t) = \frac{1}{|\mathcal{M}|} \int_{\mathcal{M}} a(x(t))dx \tag{3.4.1}$$

where $|\mathcal{M}|$ is the absolute volume of the manifold (surface area in our case) and 'a' is the observable being examined. Since the volume of the manifold is broken into small strips in the periodic orbit theory on a Poincaré section, we may approximate the infinitesimal values $\mathcal{M}_i = \Lambda_i$ so that $\mathcal{M} = \sum_i \mathcal{M}_i = \sum_i \Lambda_i$, where Λ_i is the eigenvalue of the ith periodic orbit. What is of more immediate use is the time average of the system, where one defines the integrated observable;

$$A^t(x_0) = \int_0^t a(f^\tau(x_0))d\tau \tag{3.4.2}$$

However, since our system is over an infinite series of periodic orbits on a Poincaré section, we immediately obtain a sum rather than an integral.

$$A^n(x_0) = \sum_{k=0}^{n-1} a(f^k(x_0)) \tag{3.4.3}$$

where the time average drops out of the equation by taking the average of equation 3.4.3 over time, and taking the limit asthe time go to infinity. We may perform the averaging using discrete steps over prime cycles. We can define:

$$A_p = a_p n_p = \sum_{i=0}^{n_p-1} a(f^i(x_0)) \tag{3.4.4}$$

where p is the prime cycle and n_p is the discrete time period, or the period of orbit. This is only true over periodic orbits since $T_p \propto n_p$. What is really important to pay attention to is the fact that the system is defined for closed loops. If a loop it retraced q times, the final average for that loop is qA_p The average we are interested in is only over a single traversal of the orbit.[52]

Now, since equation 3.4.4 is only true for well behaved orbits we may also need to consider the possibility that each of the values A_p are different depending on the period p. The original function, equation 3.4.1, must now be taken into consideration. We may replace the integral with a discrete sum, and since we are observing periodic orbits over all phase space, we are left with:

$$\langle a_p \rangle (n_p) = \sum_{i}^{n_p - 1} \frac{A_p}{n_p \left| \Lambda_i \right|} \tag{3.4.5}$$

Once again we are left with a simplified form of averaging, leaving the system completely dependent on the number of orbits within a given period. The overall average is then a sum of all orbits in the sum of all periods. This leaves us with the sum:

$$\langle a \rangle (n_p) = \sum_{n_p=1}^{\infty} \langle a_p \rangle (n_p) = \sum_{n_p=1}^{\infty} \sum_{i}^{n_p - 1} \frac{A_p}{n_p \left| \Lambda_i \right|} \tag{3.4.6}$$

which is just the asymptotic limit. This equation is the spatial average of the time averages, with all allowed periodic points considered. Now that we have the system where we want it, with all known values relatively easy to determine, once all the periodic orbits are located. However, as stated in Predrag [52], the average value for a is not tractable in practice. So, instead of calculating the average of the system, one calculates the average of the exponential of the observable to be averaged. This forces the average to be well behaved. Since the limit for the time average exists for all x_0, and since the system is assumed to meet the hyperbolic assumption, the time average of the system tends to one value $\bar{a}$. The summed system then goes as $\bar{a}n$ and

ultimately

$$\langle e^{\beta A^n} \rangle \propto e^{ns(\beta)} \tag{3.4.7}$$

where in the asymptotic limit $n \to \infty$ we get

$$s(\beta) = \lim_{n\to\infty} \frac{1}{n} \ln e^{\beta A^n}. \tag{3.4.8}$$

Finally, we have exactly what we are looking for:

$$\left.\frac{\partial s}{\partial \beta}\right|_{\beta=0} = \lim_{n\to\infty} \frac{1}{n} \langle A^n \rangle = \langle a \rangle \tag{3.4.9}$$

which is analogous to the thermodynamic formalism for entropy in the grand canonical system Pathria [54]. For the methods used in a continuous system please see Predrag's [52] the chapter on Dynamical Averaging.

3.4.1 Hyperbolicity & Trace

One may review the concepts of the trace operator in the webbook [52]. Here we simply state the more relevant material. In reality, the trace operator actually refers to the full 3 dimensional flow, however for our purposes a Poincaré section was used, and thus reduces the system to 2 dimensions. The trace operations may then be replaced with a sum. A simple operator that captures the effects of periodic orbits is the trace operator [52]. However for our purposes it can be simply stated that the trace operator depends on the periodicity condition described in the previous section. Since our system is a flow reduced to a map, the equations simplify to discrete sums

or products. The contribution of an isolated prime cycle can be evaluated by taking the integral for a region around the cycle,

$$tr_p\mathcal{L}^{n_p} = n_p \prod_{i=1}^{d} \frac{e^{A_p}}{1-\Lambda_{p,i}} \tag{3.4.10}$$

where the p indexes the pth prime cycle, and A_p is the observable we are interested in.

Clearly, the sum will blow up to infinity if $\Lambda \to 1$. This is where the hyperbolicity assumption comes in, it states that the stability eigenvalues are bounded away from unity. If this condition failed then the entire method would be invalid, and a new approach is necessary. Since the system we are interested in has three possible eigenvalue states, $|\Lambda|$ is either equal to, greater than, or less than 1. We only keep those calculations that ensure the satisfaction of the hyperbolicity assumption, that is $|\Lambda| > 1$.

Equation 3.4.10 can be put into a more recognizable form since we do not know the stability eigenvalues, but rather the matrix that produces them. A more useful approach would be to consider;

$$tr\mathcal{L}^n = \sum_{x_i \in f^n(x)=x} \frac{e^{\beta A_i}}{|det(\mathbf{1}-\mathbf{J}^n(x_i))|} \tag{3.4.11}$$

where the index i runs over all periodic orbits in period n. Since we are dealing with periodic orbits, the time period in which we are dealing with the system is arbitrary as long as we are taking integer units of the period of the orbit. That is to say that

time $t = qT_p$, and from there we can see that to take the limit $t \to \infty$ we need to take the limit as $q \to Large$ for the value $\left|det(\mathbf{1} - \mathbf{J}_p^q)\right| \to |\Lambda_p|^q$. This is the point at which the calculations are easiest to perform since all components of the trace are well known, and all the transient behaviour of the system has been completely worked out. It is the time asymptote that we are concerned with when we speak of the asymptotic trace condition.

With the sum in equation 3.4.11 we run the risk of encountering repeated smaller orbits that are not prime cycles. According to periodic orbit theory an observable is additive along the cycle. That is, if a prime cycle repeats itself q times, as mentioned in the previous section, the resulting observable A_p is repeated q times. $A_i = A^n(x_i) - qA_p, x_i \in p$ Thus we have to be cautious of double counting an earlier found value. To do this we introduce a Kronecker delta δ_{n,n_r} into the sum 3.4.11 and add over all prime cycle, thus leaving us with

$$tr\mathcal{L}^n = \sum_p n_p \sum_{q=1}^{\infty} \frac{e^{q\beta A_p}}{|det(\mathbf{1} - \mathbf{J}_p^q)|} \delta_{n,n_p q} \tag{3.4.12}$$

This equation as stated in Predrag [52] is awkward due to the dirac delta, so to remove it the common procedure is to take a Laplace transform of the sum leaving us with,

$$\sum_{n=1}^{\infty} z^n tr\mathcal{L}^n = \sum_p n_p \sum_{q=1}^{\infty} \frac{z^{n_p q} e^{q\beta A_p}}{|det(\mathbf{1} - \mathbf{J}_p^q)|} \tag{3.4.13}$$

which after the substitutions are made and a geometric series is observed we have:

$$\sum_{\alpha=0}^{\infty} \frac{z e^{s_\alpha}}{1 - z e^{s_\alpha}} = \sum_p n_p \sum_{q=1}^{\infty} \frac{z^{n_p q} e^{q\beta A_p}}{|det(\mathbf{1} - \mathbf{J}_p^q)|} \tag{3.4.14}$$

Now we are finally in a position to use the periodic orbit information.

3.4.2 Lyapunov exponents

Now that we know how to formally use the information from the periodic orbits, we can use it to determine the Lyapunov exponent for the system. We have defined the Lyapunov exponent to be $|\delta x(t)| \approx e^{\lambda t} |\delta x(0)|$ from chapter 1, where λ is the mean rate of separation between trajectories. To formally calculate λ we can go through the procedure of taking limits of a logarithm

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{|\delta x(t)|}{|\delta x_0|} \tag{3.4.15}$$

however, there are two things that simplify this procedure. The first is that we are dealing with periodic orbits, and therefore taking the asymptotic limit is the same as calculating the time at the period of the orbit, where we know for the individual orbits that;[37]

$$\lambda_i = \log \Lambda_i. \tag{3.4.16}$$

The second fact is that for an infinitesimal δx we know the ratio in equation 3.4.15 is exactly the Jacobian matrix. So equation 3.4.15 becomes:

$$\lambda = \lim_{q\to\infty} \frac{1}{qT_p} \ln(\left|J^{qT_p}(x_0)\right|) \tag{3.4.17}$$

however, since we are dealing with maps and periodic orbits, the time period T_p is reduced to a simpler form, $T_p = \frac{n_p a}{2\pi}$. Here, a is as defined in chapter 1. Finally we

can evaluate $\left|J^{qT_p}(x_0)\right|$ where x_0 is a point on the periodic orbit, using the leading eigenvalue of the Jacobian to be;

$$\bar{\lambda_i} = \lim_{q\to\infty} \frac{n_p a}{2\pi q} \ln(|\Lambda_i^q|) \tag{3.4.18}$$

This is in a form that we are readily capable of solving, for each located periodic orbit. Since it is the average value, one should expect to simply replace the observable of equation 3.4.9 with $\ln(|\Lambda_i|)$. We are interested in the asymptotic limit, and thus only the largest eigenvalues contribute to the sum in 3.4.18, since the smaller eigenvalues are negligible. As well, we must consider that as q increases we multiply the Jacobian by itself q times, and thus we have the determinant as the qth power of the product of the eigenvalues for the Jacobian. This leaves us with;

$$\bar{\lambda_i} = \frac{n_p a}{2\pi} \ln(|\Lambda_i|) \tag{3.4.19}$$

3.4.3 Escape Rate

The escape rate for a system is defined as the average time required for a trajectory to diverge to infinity[52]. What we generally find is that the survival of a given trajectory is a function of the area of phase space occupied by each of the elements of the trajectory. In our case, for a survival probability of a period 2 orbit,

$$\hat{\Gamma}_1 = \frac{|\mathcal{M}_0|}{|\mathcal{M}|} + \frac{|\mathcal{M}_1|}{|\mathcal{M}|} \tag{3.4.20}$$

and a period n orbit becomes;

$$\hat{\Gamma}_n = \frac{1}{|\mathcal{M}|} \sum_i^n |\mathcal{M}_i| \tag{3.4.21}$$

where $\mathcal{M}$ is the total area of phase space, and $|\mathcal{M}_i|$ is the area of the ith strip. This is expected to drop off exponentially with n and thus produce the limit;

$$\frac{\Gamma_{n+1}}{\Gamma_n} \to e^{-\gamma} \tag{3.4.22}$$

where γ is the escape rate of the system.[52] According to Predrag [52] the periodic points also keep track of the size of the partition. Each section of the partition contains a periodic point. As the periods increase, the section of phase space becomes well defined, until the width of the partition goes as $l_i = a_i / |\Lambda_i|$, where $|\Lambda_i|$ is the stability eigenvalue of the ith periodic point. Due to the hyperbolicity assumption we have that a_i are taken to the first order to be unity, in effect we ignore them. With this in place we have that the area of the ith strip of the phase space is calculated by $|\mathcal{M}_i| = Ll_i$, with L being the width of the strip of phase space containing each orbit. Rewriting 3.4.21, we have

$$\Gamma_n = \sum_i^n \frac{1}{|\Lambda_i|}. \tag{3.4.23}$$

Now, for large n we have the limit $\Gamma_n \to e^{-n\gamma}$ and so there is a great simplification

$$\gamma = \lim_{n\to\infty} \frac{1}{n} \ln \Gamma_n = \lim_{n\to\infty} \frac{1}{n} \ln \sum_i^n \frac{1}{|\Lambda_i|} \tag{3.4.24}$$

3.4.4 Measure

Now that we have defined the system in terms of the periodic orbits and their stability eigenvalues, another idea must be investigated for completeness. This concept is the measure of the system. The measure of the system must have one basic property, it must sum to unity. Considering the description of the escape rate, it may be useful to consider the function;

$$\mu_i = \frac{e^{n\gamma}}{|\Lambda_i|} \tag{3.4.25}$$

which is the measure of the system for each periodic orbit.

For bounded systems such as ours, the escape rate is zero, or tends to zero in the asymptotic limit. The consequences of this are a further simplification of equation 3.4.25 to match that of equation 3.4.23, since $\gamma = 0$.[8] We further expect that the sum of the measure is unity. This will have significant consequences in dimensional analysis in chapter 5.

3.5 Cycles

3.5.1 Zeta

Using the methods for finding the periodic orbit trajectories in the Poincaré section as defined in chapter 1. We are interested in, due to the hyperbolicity assumption, the sum in equation 3.4.23. Here, we are calculating the Γ_n for each period of orbit. To continue the analysis, we must define a generating function for sums over all periodic

orbits of all lengths to be:

$$\Gamma(z) = \sum_{n=1}^{\infty} \Gamma_n z^n. \tag{3.5.1}$$

When observing the periodic orbits, we have a geometric consideration about the expanding eigenvalue and the area encompassed by the attractor, where we may think about 3.4.23 to produce:

$$\Gamma(z) = \sum_{n=1}^{\infty} \sum_{x=f^n(x)} \frac{z^n e^{\beta A^n(x_i)}}{|\Lambda_i|}, \tag{3.5.2}$$

where both the number of periods and the number of orbits within a period are considered. This falls directly from equation 3.4.6. When dealing with the periodic orbits, the eigenvalues are restricted to prime cycles, and thus the sum is also restricted. Each orbit of length n_p contributes n_p terms, reducing 3.5.2 to;

$$\Gamma(z) = \sum_p n_p \sum_{r=1}^{\infty} \left(\frac{z^{n_p} e^{\beta A^n(x_i)}}{|\Lambda_p|} \right)^r = \sum_p \frac{n_p t_p}{1-t_p}, \quad t_p = \frac{z^{n_p} e^{\beta A^n(x_i)}}{|\Lambda_p|}. \tag{3.5.3}$$

After some calculus this leads to the relation

$$\Gamma(z) = -z \frac{d}{dz} \sum \ln(1 - t_p) \tag{3.5.4}$$

and, due to the euler relation, we end up with the product relation;

$$1/\zeta(z) = \prod_p (1 - t_p) \tag{3.5.5}$$

where $\zeta(z)$ is the dynamical zeta function.

The usefulness of the dynamical zeta function is that the $1/\zeta(s) = 0$ disappears for s = s_0 = s(0). where s is defined in equation 3.4.7.

If we take $z = e^{-s(\beta)}$ the eigenvalue condition becomes

$$\frac{1}{\zeta(s,\beta)} = 0 \tag{3.5.6}$$

where s and β are parameters of the zeta function. Now, we take the total derivative of zeta with respect to β at $\beta = 0$ and we get:

$$0 = \frac{d}{d\beta}\left(\frac{1}{\zeta(s,\beta)}\right)_{\beta=0} = \frac{\partial}{\partial\beta}\left(\frac{1}{\zeta(s,\beta)}\right)_{\beta=0} + \frac{ds}{d\beta}\frac{\partial}{\partial s}\left(\frac{1}{\zeta(s,\beta)}\right)_{\beta=0} \tag{3.5.7}$$

which leads us to a natural conclusion from the averages worked out above

$$\frac{ds(\beta)}{d\beta} = -\frac{\langle A\rangle_\zeta}{\langle n\rangle_\zeta} \tag{3.5.8}$$

where

$$\langle A\rangle_\zeta = \frac{\partial}{\partial\beta}\prod_p (1 - t_p)_{\beta=0} \tag{3.5.9}$$

and

$$\langle n\rangle_\zeta = \frac{\partial}{\partial z}\prod_p (1 - t_p)_{\beta=0}. \tag{3.5.10}$$

For more properties of the dynamical zeta functions see [51].

3.5.2 Cycle Expansion

Now that we have a working equation for the periodic orbit theory we can exploit it to find what we need to know about the system. One method of doing so is to use cycle expansions. We first expand equation 3.5.5 and group the terms in order of the period of orbit. We must note that the cross products will result in values

besides the original orbit. These are the pseudo-orbits, and they are important for the calculations. As seen below we have groupings of pseudo-orbits and prime cycles, this is referred to as shadowing since each of the orbits longer than period 1 will have at least one pseudocycle that follows it mathematically and symbolically. As mentioned earlier any cyclic permutation of a prime cycle is the same prime cycle. This matter becomes important for the grouping of pseudo-orbits as any combination of smaller orbits that returns the same prime cycle, as seen at the end of the example below. Now considering the grouping individually one finds that pseudocycle shadowing can actually disrupt the calculation tremendously. If one considers the meaning of t_p from equation 3.5.5, when looking at the shadowed terms you get

$$t_{ab} - t_a t_b = t_{ab} \left(1 - \left| \frac{\Lambda_{ab}}{\Lambda_a \Lambda_b} \right| \right) \tag{3.5.11}$$

where the term inside the brackets, if the cycles are weighted equally, reduces to zero. Only the unshadowed (first few) terms contribute to the overall calculation.

In real cases, the shadowing usually does not actually reduce the higher order terms to zero, but rather to a very small number. The reasons for this is that the prime cycle and the pseudocycles lie close to each other in phase space. This becomes significant when dealing with higher order periods, since the number of pseudocycles is much greater and there are terms that get closer and closer to zero. This is why the dynamics of the system can be found using only the smallest unstable eigenvalues.

3.5.3 Pseudocycles

The prime cycles make up only part of the orbital spectrum that we are interested in finding. Each real orbit beyond some lower limit begins to experience pseudocycles, which are combinations of existing orbits to make up artificial orbits of the same length. When these orbits begin to symbolically match existing larger orbits the pseudocycles are said to shadow the larger orbits. The mathematical behaviour of this shadowing effectively acts to cancel the effects of the larger orbits.

First, we describe the dynamical zeta function as a power series, by recombining the product into leading order terms.

$$1/\zeta = \prod_{p}(1 - t_p) = 1 - \sum_{p_1 p_2 \ldots p_k}{}' (-1)^{k+1} t_{p1} t_{p2} \ldots t_{pk} \tag{3.5.12}$$

Then for simplicity we denote $\pi = p_1 p_2 \ldots p_k$, and the weights of each pseudocycle becomes

$$t_\pi = (-1)^{k+1} \frac{1}{|\Lambda_\pi|} e^{\beta A_\pi - s T_\pi} z^{n_\pi} \tag{3.5.13}$$

where Λ_π is the product of the unstable eigenvalues for each sub-cycle in the pseudocycle. Since we know that the cumulative effect of retracing an orbit q times is to add the observable q times, or have qA_p, we can extend this idea to pseudocycles. If we trace the first cycle in the pseudocycle we have A_{p1}, then continuing to the second cycle in the pseudocycle we have A_{p2} and so the net observable can be thought of as $A_\pi = A_{p1} + A_{p2}$. The net period of orbit will then be $n_\pi = n_{p1} + n_{p2}$, and the net

eigenvalue will be $\Lambda_\pi = \Lambda_{p1}\Lambda_{p2}$ since this system will be the product of two different Jacobian matrices.

From this we may extend to any pseudocycle with the simple relations;

$$\begin{aligned} n_\pi &= n_{p1} + \cdots + n_{pk} \quad & T_\pi &= T_{p1} + \cdots + T_{pk} \\ A_\pi &= A_{p1} + \cdots + A_{pk} & \Lambda_\pi &= \Lambda_{p1}\Lambda_{p2}\ldots\Lambda_{pk} \end{aligned} \tag{3.5.14}$$

The important contributions come from the shorter cycles, since the longer cycles offer exponentially decaying corrections.

Now we have the information on the prime cycles and the pseudocycles. To apply this information to the dynamical averaging, the first realization is that we are looking for roots, or singularities of the dynamical zeta function. These occur when $\beta = 0$, as well as when $s = s_0 = 0$ in the dynamical function for t, leaving $t_\pi = (-1)^{k+1}\frac{1}{|\Lambda_\pi|}z^{n_\pi}$. Technically, we know two different aspects of the system thus far, the level sum, and the dynamical zeta function. We must use these to the best of our knowledge in analyzing the remaining system properties.

We see from the definition of t_π in equation 3.5.13 that most of the important information comes from the value t_i, that is, the averages we are looking for fall from the first and second derivatives of the functions defined in equation 3.4.8. The observables in question are easily obtained from equation 3.4.9.

When considering the cycle expansions to calculate the corrected averages one only needs to expand the definition of each average as a collective sum, over all of the cycle space. Since we are looking for zeros of the dynamical zeta function so the

calculations may be made when $\beta = 0$ or $s(\beta) \to s(0)$. Even here the shadowing takes effect, offering small corrections for larger orbits.

3.6 Stability Ordering

For the gears and rod system, we have no finite grammar rules therefore the sequences may go on indefinitely and organization of the cycles must change slightly, which depends on a stability cutoff.

3.6.1 Ordering

Before making any mathematical calculations for the system constants that have been generated thus far, there needs to be some form of ordering for the values so that their evolution may be monitored. There are two methods, for ordering. The first is ordering based on their symbolic dynamics, which would make sense for systems that are well ordered and have a finite grammar. The second method is to order the cycles based on their stability, leaving the symbolic dynamics behind and concentrating solely on the stability eigenvalues. This type of ordering is useful for the many dynamical systems that do not have a finite grammar, as well as for cases where finding all of the orbits in a given period is arduous, such is the case for many larger periods. This ordering reduces the finding of all the orbits to finding only the more stable of the unstable orbits. Basically, any orbit that may have an eigenvalue "fairly" close to 1. This is a relative concept, and must be considered carefully. For

the application we had, this meant that all orbits with an eigenvalue less than 1000 had to be found. This number was selected due to the inverse relationship between the averages and the eigenvalues. For four digit accuracy an eigenvalue of 1000 was needed, for greater accuracy, larger eigenvalues will be needed.

3.6.2 Smoothing

Smoothing data is a method of data manipulation that was implemented to correct for flaws in the data, such as missing values, and, in this case, was a compensation for truncating data. This is a very common technique for controlling the noise output due to data. This method is a necessity when one uses stability ordering over topological ordering. When we truncate the data set we run the risk of creating partial shadowing where some of the shadowing terms may exist but the cycle they shadow may be removed.

The compensation goes as follows; for each eigenvalue in a function we multiply the element by a smoothing function $f(\Lambda)$, where $f(\Lambda)$ is a monotonically decreasing function with boundary conditions $f(0) = 1$ and $f(\Lambda_{max}) = 0$. Clearly, there are several functions that one can choose from, but for the purposes of the data used, only 4 were tested, two of which came from Predrag's [52];

$$f_1(\Lambda) = 1 - \left(\frac{\Lambda}{\Lambda_{max}}\right)^2 \tag{3.6.1}$$

$$g_1(\Lambda) = \frac{e^{f_1(\Lambda)} - e^{-f_1(\Lambda)}}{e - \frac{1}{e}} \tag{3.6.2}$$

$$f_2(\Lambda) = 1 - \left(\frac{|\Lambda|}{\Lambda_{max}} \right) \tag{3.6.3}$$

and

$$g_2(\Lambda) = \frac{e^{f_2(\Lambda)} - e^{-f_2(\Lambda)}}{e - \frac{1}{e}} \tag{3.6.4}$$

The effects of smoothing can be tested by varying the cutoff value. If the variations are large then smoothing is required. One can adjust the extent of the smoothing by changing the smoothing function. See figure 3.1 for the plots of the four smoothing function used.

3.6.3 Cutoff

The choice of a cutoff value is important to this project. For an appropriate choice of cutoff we must consider the data set as a whole and pick a point which looks smooth, such as a plateau. With this choice of cutoff, we avoid the transient section of the data, where output would be utterly meaningless, and we can also avoid the regions where the data becomes falsified by the partial shadowing. Ordinarily we are only interested in order of magnitude cutoffs. Further considerations to obtain a proper cutoff would be that it maximizes usable data. That is to say, when we do our orbit hunting one cannot be entirely sure if all the cycles from a given period are found, or that all of the orbits with a particular magnitude of stability have been found. Contributions from larger stabilities will be harmful for our system since the missing orbits may have provided the necessary shadowing to remove the cycles

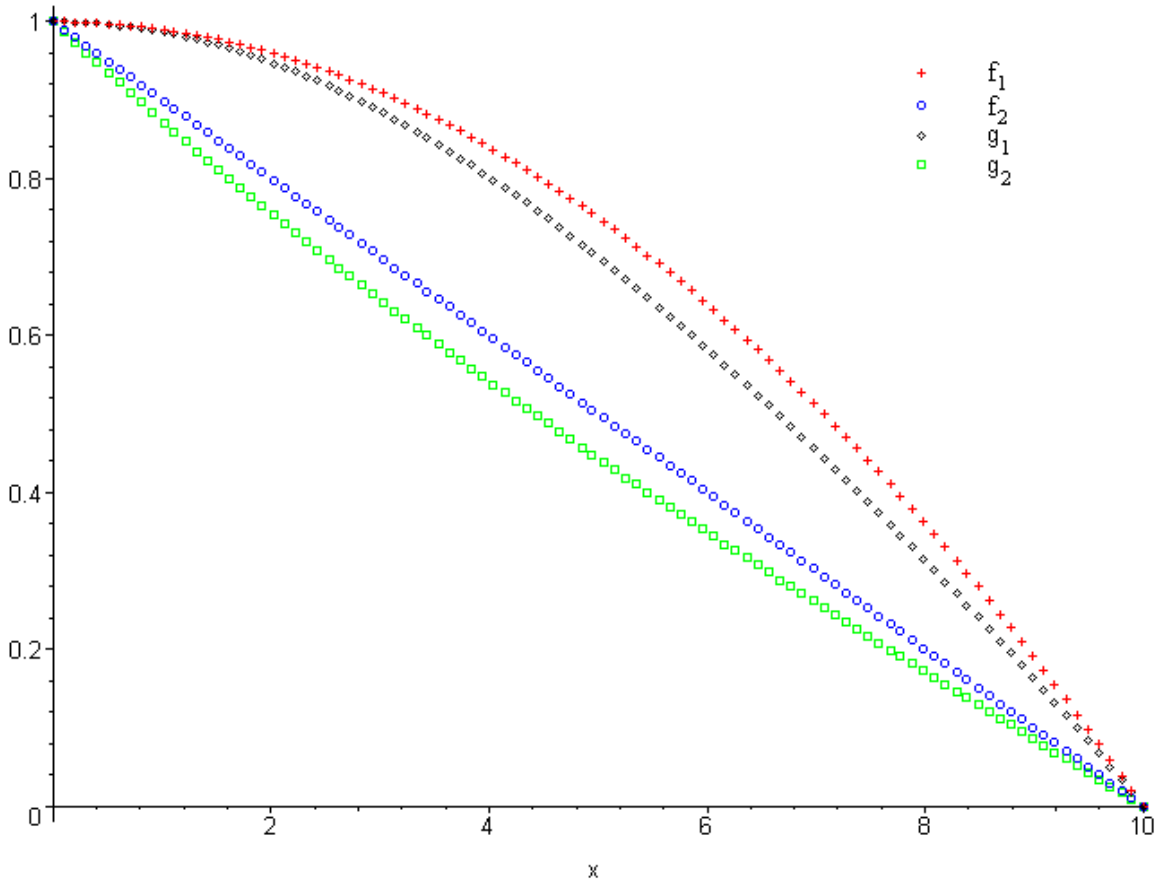

Figure 3.1: The Smoothing Functions

from consideration. Since the zeta functions are sensitive to only the most stable of the eigenvalues the larger stabilities may be removed to prevent false data output. However, to be sure that this is the effect of the cutoff, the plateau technique is also required.

3.7 Conclusion

I have presented here a complete method of both calculating the periodic orbits, as well as a method of calculating the dynamical averages in the system. These methods are broad in their scope as they can be applied to any system of differential equations, of any order. One still has to be cautious since random processes will produce results, but their meaning will be null. Computational implementation of these ideas will be described in the chapter 4, since not all of the methods are immediately obvious.

Chapter 4

Computation and Calculation

4.1 Introduction

We have been through the theoretical approach to orbit finding, the root finding techniques and the analysis of the observables of each system. However, as is often the case, the theoretical approach is clean and easy to follow whereas the real solutions for these systems rely on the numerical calculations, which are cumbersome, and are worthy of mention.

4.2 Finding The Periodic Points

We have already described the Poincaré section calculation of the system as the time step evolution of the ordinary differential equation in some detail. We will restate it here for clarity.

The Poincaré section of ODE solving is best implemented for a driven system, since it is this driving frequency that we will exploit as a natural stroboscopic frequency

of the system. In our system this driving frequency is the parameter a as defined in chapter 1. Effectively, we are altering our output time step to correspond to the inverse driving frequency. It is a minor change to the coding, but it has rather dramatic effects on the output.

The calculation of the actual orbits, based on the initial guesses, is less than obvious. The first step was to develop an ODE solver for the system, fortunately most of the ground work was done in the code used for the Poincaré section, as described in chapter 1, and seen in Appendix I, we just alter the program slightly. The next step was to create two functions, one to calculate the vector F found in the following equation [52]

$$\begin{pmatrix} 1 & & & & -J_n & v_1 & & \\ -J_1 & 1 & & & & & & \\ & \ddots & & & & & \ddots & \\ & \cdots & 1 & & & & & \\ & & -J_{n-1} & 1 & & & & v_n \\ a & & & & & 0 & & \\ & \ddots & & & & & \ddots & \\ & & & & a & & & 0 \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \vdots \\ \delta_n \\ \delta t_1 \\ \vdots \\ \delta t_n \end{pmatrix} = \begin{pmatrix} -F_1 \\ -F_2 \\ \vdots \\ \vdots \\ -F_n \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.2.1}$$

which is of the form,

$$\mathbf{DF} \cdot \delta = \mathbf{F} \tag{4.2.2}$$

This equation is found and proven in Predrag's [52]. The other function, to calculate the differential matrix, DF. Each of these calculations were done in two parts. To produce these functions; First we setup the shell of each matrix, in more detail this

went as follows;

- Initialize F to zero

- Load y vector with initial guess vector line by line

for i = $1 \rightarrow 3$

- Use the Poincare ODE solver to evolve the initial guess y[i] $\rightarrow$ y'[i]

- If the current guess, y[k], is one less than the full Period of Orbit

  - Calculate y[i] = y'[i] - Initial Guess[i]

  - Modulate the vector y to keep problem bounded

  for j = $1 \rightarrow 3$

  - set F[j] = -y[j]

- Otherwise Calculate y[i] = y'[i] - Initial Guess[3*(k+1)+i]

  - Modulate the vector y to keep problem bounded

  for j = $1 \rightarrow 3$

  - set F[3*[k+1]+j] = -y[j]

Now to calculate the differential matrix DF

- Initialize DF to zero

- create and load the Poincare hypersurface, since it is in the x-y plane, the vector (0,0,1) is sufficient

- set top left of DF to identity

- Load y vector with initial guesses line by line

```
   - Load v, the flow vector by using initial points in the
original differential equations
   - Use the Poincare ODE solver to evolve the initial guess thus
producing the Jacobian matrix
   - Load DF with the Jacobian matrix in the appropriate locations
as seen in equation 4.2.1
   - Load DF with the flow vector according to 4.2.1
   - Load DF with the Poincare hypersurface according to
```

Now we have the form of the solution form as seen in equation 4.2.1. This method works very quickly, and is easily implemented using predefined libraries such as the Gnu Scientific Library. The rate of convergence to a root depends on two things, the first being the density of roots. If two roots are closely spaced on the attractor, it may be difficult to find the individual roots without an incredibly accurate initial guess. The second issue to consider is the sensitivity of the system. As was seen earlier, different systems can have varying degrees of sensitivity. The more sensitive the system, the more difficult it is to find the roots. This led us to the desire to find an automated search technique.

4.2.1 Automation

The most obvious method would be to use points on the attractor for the initial guesses, this was fairly easy to implement. The first step was to create the attractor

with as many points as possible. This required running the Poincaré section program using an array of initial guesses:

```
- Cover the phase space with a one or two decimal precision grid of points
- Use this grid as locations of the initial positions of the points and
run the Poincare section
- Let the code run long enough to skip the transient points, in our case
30 points were sufficient
- Save the output which should contain millions of points that are on
the attractor
```

Now all one needs to do is read in points generated from the above code, and use them for initial guesses. Our method required two levels of approach for efficiency. The first level required that the initial guesses carried an error on the order of 1E6, this will guarantee that the output points have a converging trend. Since many point combinations taken from the attractor will not easily converge, this weeds out the useless point combinations. This method requires several hours to run, on an Intel Pentium p4 desktop computer, for around 2 million points on the attractor. The next level of approach is to take the information received from the previous level, and take the error down to 1E-13 by performing more iterations. This method needed to be separate from the previous since the root finding method can become stuck on the bad orbit guesses while trying to reduce their error. By refining the filtered orbits,

the process takes only a few seconds per orbit.

Now we have a list of orbits that are of a specific period, with a small error. This is the list of orbits that may be used for calculating the zeta function, and consequent values as described in chapter 3. There is a disadvantage of this method, one cannot be entirely sure that all orbits have been found. However, when one also considers the symbolic dynamic techniques, one can be confident in the results. For larger period orbits, the number of initial guesses that are needed for the search of all orbits, is tremendous and often unnecessary, since, as stated earlier, we are only interested in the smaller stability eigenvalues, because they offer the largest contribution to the zeta functions. Fortunately, with this automated search method, the smaller stability eigenvalue orbits are found very quickly, and most of the larger stability eigenvalues take longer to find. For expediency, there was no attempt to find all orbits of a larger period orbit, which could potentially take months. With the orbits we have at hand we must find the stability eigenvalues, winding number and Lyapunov exponents.

4.3 Eigenvalues

To calculate the eigenvalues of the periodic orbits requires reading the orbit values line by line for each orbit found:

```
- Read orbit values in line by line (x,y,z)
- Copy the Jacobian matrix into a temporary matrix
- Run the orbit vector in the Poincare iterate function to obtain the
```

Jacobian matrix for that orbit location

- Multiply the new Jacobian matrix by the temporary matrix and store the result in the Jacobian matrix

This is repeated for the entire orbit, satisfying equation 3.2.8. Now with the Jacobian, we find the trace and discriminant of that matrix to solve for the eigenvalues by three methods;

$\Lambda = \frac{1}{2}\sqrt{Trace^2 - Discriminant}$, if Discriminant < 0;

$\Lambda = \frac{1}{2}Trace - \frac{1}{2}\sqrt{Discriminant}$, if Discriminant > 0 and Trace < 0;

$\Lambda = \frac{1}{2}Trace + \frac{1}{2}\sqrt{Discriminant}$, if Discriminant > 0 and Trace > 0.

4.4 Winding Number

The calculation of the winding number is also fairly straightforward, following the instructions in chapter 1.

- Read in the orbit values in line by line (x,y,z)

- Save the orbit vector as a temporary vector

- Perform a Poincare iterate of the orbit

- Compare the difference between the new orbit x value and the temporary vector x value

- Sum the previous step over all values obtained within an orbit

- Add the period of orbit to the result

- Divide the result by the period of orbit

Now we have all the information we need from the orbits. We now may proceed to the zeta function analysis of the system.

4.5 Pseudocycles

Calculating the pseudocycles effects requires at least one previous step, we must calculate all possible pseudocycles. To do this one must load all existing orbits into some kind of list, and then produce all combinations of orbits. All combinations of existing orbits are possible, this effectively acts like $n!$, which is easy to calculate for a small number of orbits; however it quickly gets out of hand for even 10 orbits. Therefore a more intelligent method is required. What we did is one of two things, the first would be to sort the orbits in terms of their period of orbit, then find all pseudocycles depending on their period. The drawback to this method is that all of the orbits of one period are necessary in order to calculate higher periods. As we have discussed in chapters 2 and 3, one cannot be entirely certain if all periods have been found, (this problem may occur as early as period 10, depending on the stability of the system.) In order to circumvent this issue we rely on the second method, which involves sorting the entire orbit list in terms of their stability eigenvalues. This is more complicated, however since one only has to be sure of finding all eigenvalues up to a specific magnitude, one can be more certain of the results. Our method was to use linked lists, which was a very fast method of sorting, since all the work done was in the computer memory, only outputting to file when completed. Another benefit

of using the eigenvalues for sorting is that one can introduce a cutoff eigenvalue. Since all of the information we are interested behaves as $1/\Lambda$ a large Λ can make the contribution of that orbit negligible. A cutoff may be used to analyze the data more efficiently, by removing the unshadowed information. Again, since we can only use eigenvalues that are complete we must also use cutoff to remove unknown values, for instance with table 2.1 values we may only know the orbits up to 1E4, and thus that has to be our cutoff. Another technique that was the assignment of negative values. All elements of pseudocycle space that is created by an odd multiple of prime cycles is negative as seen in [52], including the prime cycles themselves. This was easiest to handle while sorting the cycles, rather than trying to keep the information on the multiples of orbits in the pseudocycle.

4.6 Results

When the shadowing was done we were left with only the most relevant information was retained, which was used in the zeta evaluations. Here the coding was very much like the theory in chapter 3, due to the discrete infinite sums. Therefore, we need only a truncation for the data and further description of the algorithm is unnecessary. However, we are now able to describe the results in a little more detail.

What we expect for our system as discussed in chapter 3 is that the escape rate tends to zero as we include more data from higher orbits. Since this is a requirement for our system, we need to select a cutoff that reflects that property. With that

cutoff selected, we must also select a smoothing function that will also allow the escape rate to be a minimal. This is basically done by trial and error, using the four given smoothing functions. As we see in figures 4.1 and 4.2 the escape rate does change slightly depending on the cutoff selected. Since the pseudocycles offer the only negative contribution to the escape rate average we may conclude that if the Gamma function goes below zero, we have an over compensation for the system, that is at that point we can say that we are missing orbits, and so that particular cutoff is too high.

What we see in the plots in 4.1, 4.2, and 4.3 is that the escape rate is closest to zero when we use a cutoff at 1E3. This is strong evidence that we are missing orbits of stability 1E4 and higher. Thus further calculations with table 2.1 values must also use a cutoff at 1E3, otherwise the numerical calculations are not true to our system. Likewise we see that in 4.6, 4.6, and 4.6 the largest cutoff allowed is 1E3, although the escape rate with the cutoff at 1E4 is only slightly larger.

With the cutoff for the systems, defined we may now pursue the appropriate smoothing function. Keeping in mind that we must use monotonic functions so that the data is not distorted, just smoothed. What we have used are the four functions

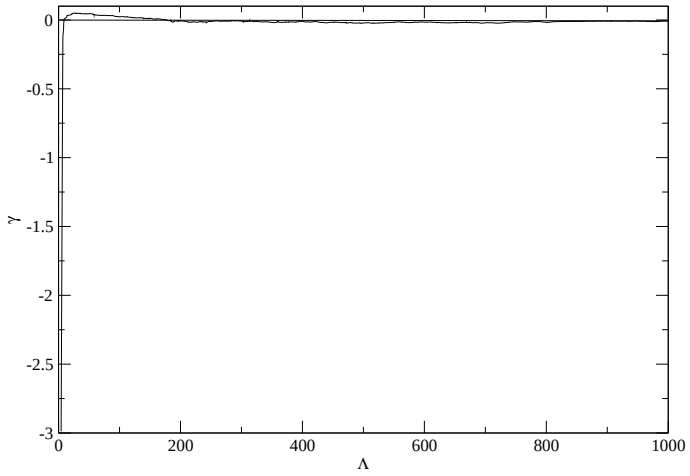

Figure 4.1: T = -0.008721

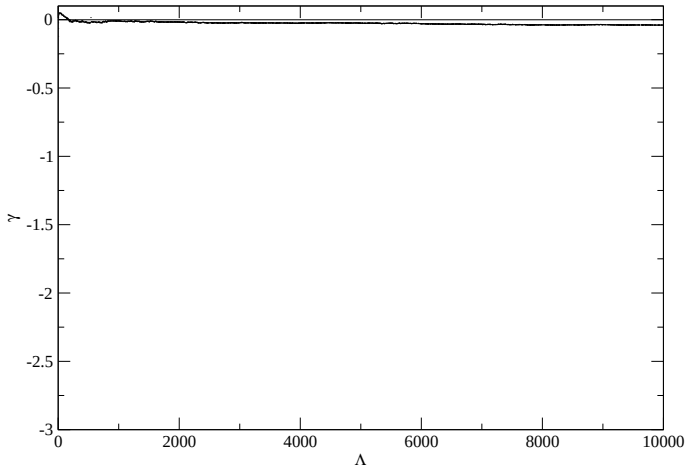

Figure 4.2: The escape rate for table 2.1 with a fourth order cutoff, γ = -0.040234

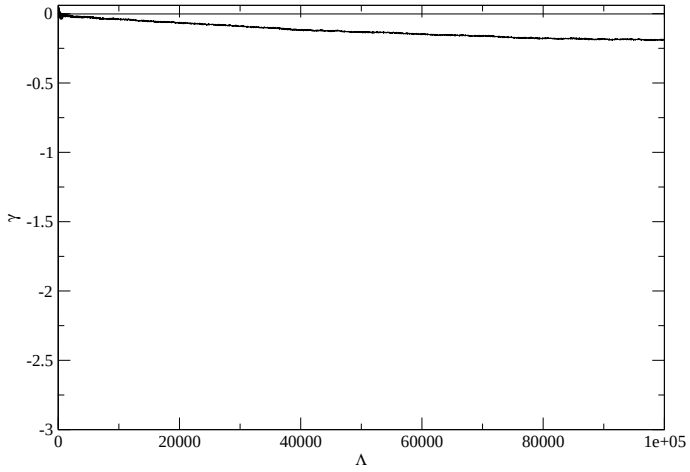

Figure 4.3: The escape rate for table 2.1 with a fifth order cutoff, γ = -0.096006

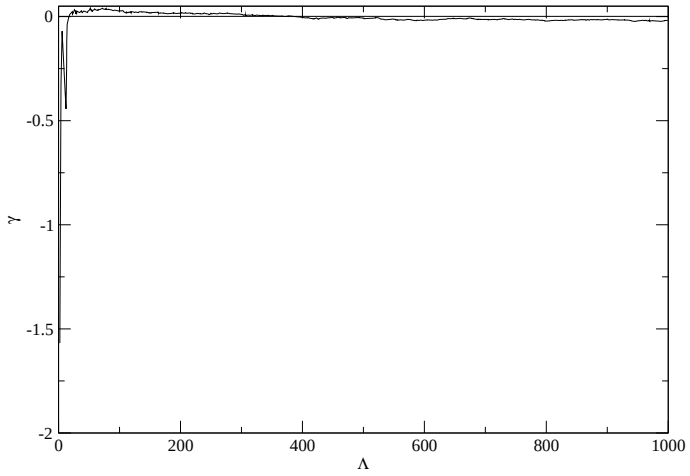

Figure 4.4: T = -0.019135

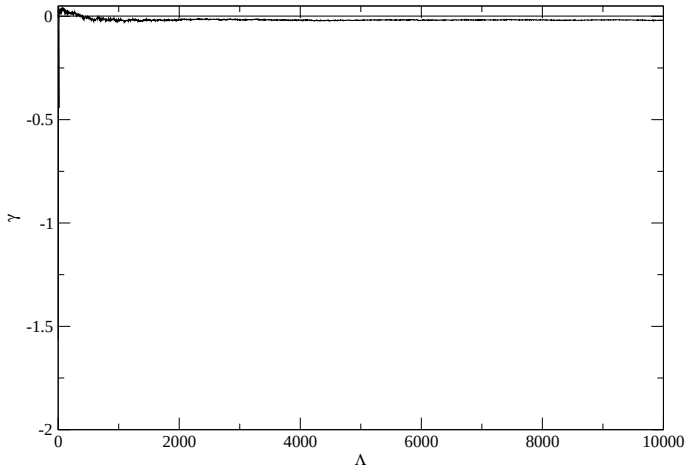

Figure 4.5: The escape rate for table 2.2 with a fourth order cutoff, γ = -0.096006

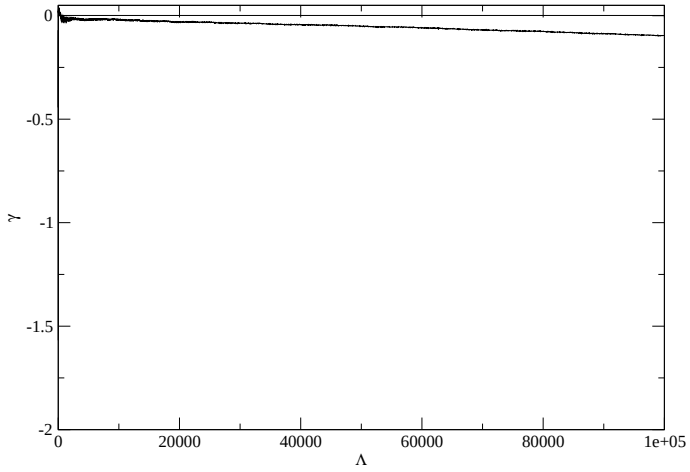

Figure 4.6: The escape rate for table 2.2 with a fifth order cutoff, γ = -0.065199

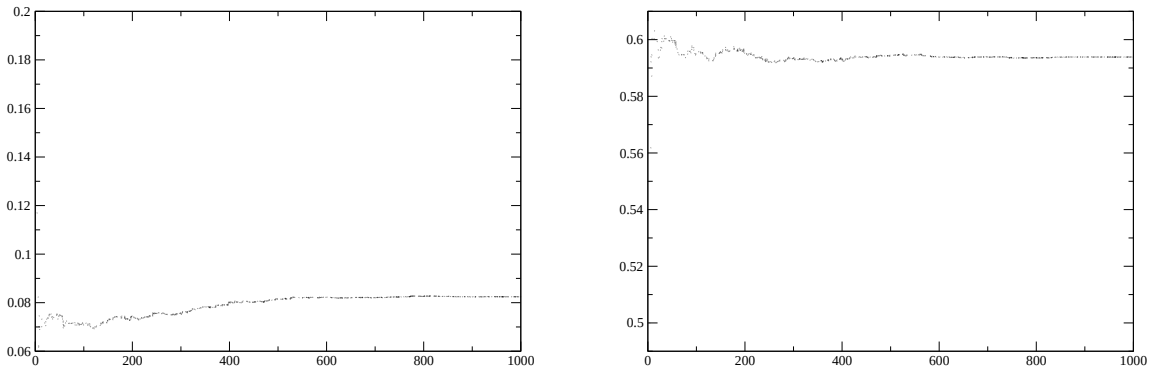

Figure 4.7: Observables for Table 2.1, Q = 0.76, a)$\lambda = 0.082489$ b)Winding Number = 0.593845

as described in chapter 3. To recap we have;

$$\begin{aligned} f_1(\Lambda) &= 1 - \left(\frac{\Lambda}{\Lambda_{max}}\right)^2 \\ f_2(\Lambda) &= 1 - \frac{|\Lambda|}{\Lambda_{max}} \\ g(\Lambda) &= \frac{e^{f_i} - e^{-f_i}}{e - 1/e}, i = 1, 2 \end{aligned} \tag{4.6.1}$$

From this we have that the smoothing function used for tables 2.1 and 2.2 is $g(\Lambda) = (e^{f_2} - e^{-f_2})/(e - 1/e)$, since this smoothing function allowed the escape rate to be closest to zero.

Using the above information, we are now prepared to move on to the rest of the observables in the system.

We also expect that stability ordering and period ordering will offer the same results. The plots are seen in 4.9 and 4.10, here we drop the smoothing and the cutoff, since these are calculations for the system as it is for each orbit.

What is expected is that if all orbits were found properly the period ordering plots

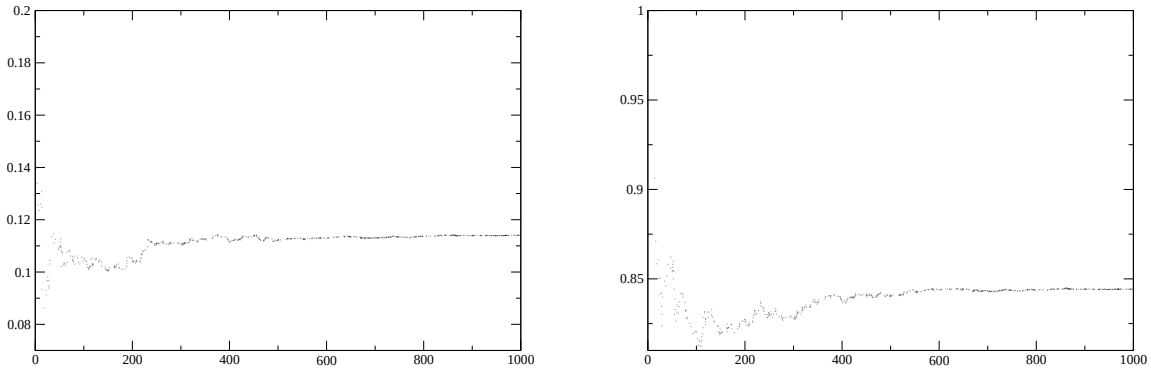

Figure 4.8: Observables for Table 2.2. Q = 1.2577, a)$\lambda = 0.114000$ b)Winding Num-

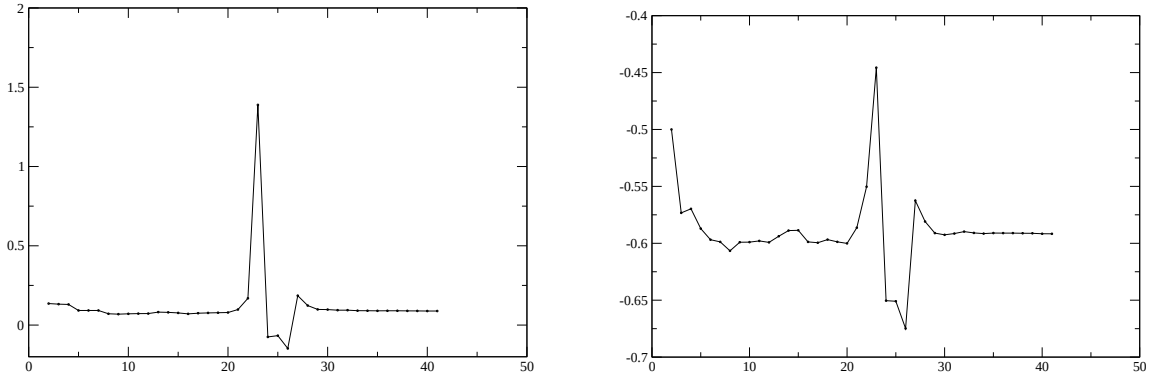

Figure 4.9: Observables for Table 2.1, Q = 0.76, a)$\lambda = 0.088563$ b)Winding Number

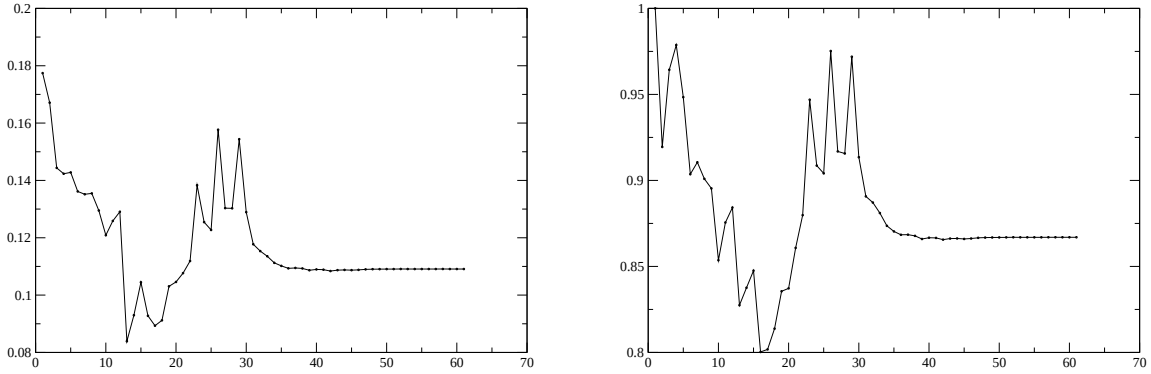

Figure 4.10: Observables for Table 2.2, Q = 1.2577, a)$\lambda = 0.109098$ b)Winding Number = 0.866934

would result with a smooth transition from one period to the other, in some form of convergence to the final result. However, what we find is a plot with several spikes in regions where smooth transitions are expected. This is a rough indication as to how many orbits are really missing. We see in figures 4.9, as an example, at or around period 26 we find an incredible spike, which tells us that we are missing some least unstable orbits, since this orbit, in our case, is only comprised of pseudocycles. Soon after that period we do find the system converging again, thus indicating that the pseudocycles in period 26 are actually shadowed by higher period orbits. Due to the complexity of the cycle expansions this type of behaviour is expected. However, in a proper period orbiting plot these spikes would not exist, and validating our stability ordering method. If we had a complete set of data, that is, all orbits from each period, there would be no difference between the stability ordering and the period ordering.

For a technical comparison, we take the calculation for the largest Lyapunov exponent, which is easily calculated using well studied methods, [52] [24] [57]. It was also discussed in detail in chapter 4. Using this method, we expect an Lyapunov exponent to be 0.083044 for table 2.1, and 0.104560 for table 2.2. Using Haken's theorem we can easily find the other two expected Lyapunov values. By comparison, we see that the values obtained through stability ordering are tending towards the expected values. This means that if we were to include more orbits, the numbers would tend closer to the expected value, just as the theory predicts.

Further analysis of the system requires an observation of the winding number. Our brute force method produces the values 0.593573, and 0.850911 for Q = 1.2577 and Q = 0.76, respectively. The data set produced by periodic orbit theory, produces 0.593845 and 0.844202, respectively. By inspection we see yet again that the values obtained in Q = 0.76 are at least an order of magnitude closer to the expected value of the system. Again, I attribute this to the escape rate value. If more orbits from Q = 1.2577 were located the agreement would be much better.

It has been established that the escape rate must tend to zero as the stabilities are added to the system. If this is not the case, the data may be incomplete and no further computations are necessary until the missing orbits are located. The closer the gamma plot comes to zero, the more credible the final results of the other plots will be. Clearly, our first plot figure 4.1 approaches zero, thus allowing us to pursue other analysis methods. The second plot figure 4.6 also comes close to zero, but not as fast as the escape for Q = 0.76. This shows that we may proceed with the calculations, however, the results will not be as good as for Q=0.76. Ultimately more orbits of a higher period may be needed, but since our results come within the expected value within error, further orbit hunting is unnecessary.

4.7 Conclusion

We have shown in this chapter methods of finding the periodic orbits as they are presented in chapter 2. A method of automating this method was also presented.

More work in this area is still necessary to make the programs utilize better search algorithms and thereby much more efficient. However, this extended beyond the scope of the project, and is best left for further study in this field. Although the code is quite complicated as a whole, each of the parts easily correspond to the theory described in chapter 3. The resulting effect was a program that could take a moderately good guess, i.e. the methods used in chapter 2, and form an accurate prediction of the allowed orbit in relatively short time. The orbits were used for averaging, and easily produced results for the system that came within a few decimal places of the brute force calculations. The benefit of this method is that long term prediction becomes possible since one only observes the known periodic orbits of the system, and calculates the chaotic propertied from there. This leaves the original chaotic system alone, which is where most of the difficulties in long term calculation lie.

Chapter 5

Fractals

"Big gaskets are made of little gaskets,

The bits into which we slice 'em.

And little gaskets are made of lesser gaskets

And so ad infinitum."

5.1 Introduction

Some of the most intricate pictures in existence are fractals.

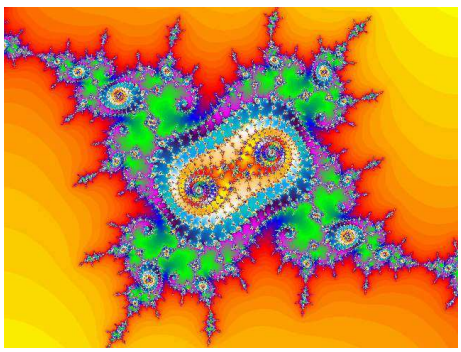

Figure 5.1: One of an infinity of fractal sets

A fractal may be described to be a space filling object that is completely made out of corners. A more mathematical description of a fractal is that it is made completely out of singularities, but these singularities fall close to each other, and never actually repeat themselves. [36] Another definition is a complicated geometric structure that does not simplify upon magnification. [35] Although there is no specific definition of a fractal, they are incredibly interesting structures to study. In traditional mechanics, standard geometric shapes are used to describe trajectories such as periodic orbits. Fractals are the geometric standard for chaotic mechanics. Basically this is the connection between chaos and fractals. A better visual relation between the two will be offered shortly. The study of fractals goes back to Gaston Maurice Julia who was a French mathematician that studied iterative functions and the space they existed in. He wrote a paper on the topic, "Memoire sur l'iteration des fonctions" in 1918. These findings were then forgotten for several years until a fellow mathematician, and former student, Benoit Mandelbrot, came along and started a more general investigation. Mandelbrot's name is often synonymous with fractals, as he actually coined the term "Fractals", meaning broken or fractional dimension. His set is said to be comprised of all possible Julia sets. Through the work of Mandelbrot and Hausdorff the dimension of a fractal could be calculated. What has been established is that fractals may have non-integer dimension, and so-called fat fractals, such as the Mandelbrot set, have integer dimension. One could easily lose themselves in the mathematical

richness of such objects, however for realistic uses of these objects one only needs to know how they behave in general and are classified.

5.1.1 Dimension

Dimensionality is a critical piece of information about a fractal, since this is the method by which a fractal may be readily identified. Effectively, it is like fingerprinting a fractal. There are several methods to determine the dimension, and there are several different definitions of dimension. We will only concentrate on the three mentioned in chapter 1, the Box dimension, the information dimension, and the correlation dimension. Many books for example [36, 22, 24], summarize the basics of each dimension discussed here. The dimension of a fractal needs to exhibit a power law relationship between the measurement scale and the type of counting used, for example counting every point in a measurement region, or counting every pair of points in the measurement region, before it may be considered a fractal. A standard geometric shape, such as a square exhibits an integer power relationship, and thus we are left with the standard integer dimensions for these shapes. For a while it was thought that fractals could only consist of fractional dimensions, however it was shown that fractals may also exhibit integer dimension. Table 5.1 shows the various dimensions for several different shapes.

As we can see, even the logistic bifurcation diagram has a fractional dimension, and

Table 5.1: Fractal Dimensions of Various Fractals

Shape	Box Dimension	Entropy Dimension	Correlation Dimension
Square	2.00	1.997 ± 0.003	1.998 ± 0.001
Logistic Bifurcation Diagram	1.52 ± 0.02	1.12 ± 0.02	1.04 ± 0.01
Henon Map	1.259 ± 0.007	1.233 ± 0.003	1.197 ± 0.005
Table 2.1	1.03 ± 0.01	1.021 ± 0.009	0.95 ± 0.01
Table 2.2	1.2 ± 0.1	1.04 ± 0.001	0.81 ± 0.02

therefore we see a clear connection between chaos and fractals. Further observations from table 5.1 is that the topological dimension of the two Poincaré sections are close to one. Basically, table 2.1 (Q=0.76) Poincaré section appears to be almost a line, which has box dimension one. Since the fractal shape is almost a line, one may expect that the degree of chaos exhibited by the system will be very similar to a one dimensional system, whereas the fractal dimension of table 2.2 (Q=0.12577) shows a stronger deviation from a line, and consequently one expects the system to act somewhere between a one dimensional attractor and a two dimensional attractor, but closer to the one dimensional case.

Another dimension that has not found its way into a geometric standard is the Lyapunov dimension. This dimension was discovered by Kaplan and York. The Lyapunov dimension is calculated based on the Lyapunov exponents of the system. For a simple 2 dimensional map the Lyapunov dimension, D_{Ly} is defined as;

$$D_{Ly} = 1 + \left(\frac{\lambda_1}{|\lambda_2|}\right) \tag{5.1.1}$$

with λ_1 being the largest Lyapunov exponent, which is positive, and λ_2 being the negative Lyapunov exponent. A second constraint is that the relation $\lambda_1 < |\lambda_2|$ must also be observed. This may be generalized to higher dimensional systems as;

$$D_{Ly} = K + \left(\sum_{j=1}^{K} \frac{\lambda_j}{|\lambda_{K+1}|} \right) \tag{5.1.2}$$

[48] where K is the largest index that makes a sum of Lyapunov exponents non-negative.[36] However, since we have been dealing with maps, we will only be concerned with the primary case in equation 5.1.1.

From our system we found that the largest Lyapunov exponent for Q = 0.76 was 0.082489, therefore by equation 1.2.22 we have at one of the Lyapunov exponents equal to zero, and the second is -1.233300. This leads to a Lyapunov dimension of 1.07 ± 0.02. For Q = 1.2577 we have the largest Lyapunov exponent equal to 0.114000, and likewise the second is -0.681102, so we get a Lyapunov dimension 1.16 ± 0.02. Kaplan and York conjectured that the Lyapunov exponent spectrum is related to the morphological dimension, and that its value is between the box dimension and the information dimension, which is held true in our case. The q value for the Lyapunov exponent is expected to fall between 0 and 1, which is allowable since the restriction to integer dimensions, or integer counting is lifted by the Renyi proof for fractal dimensions. [36]

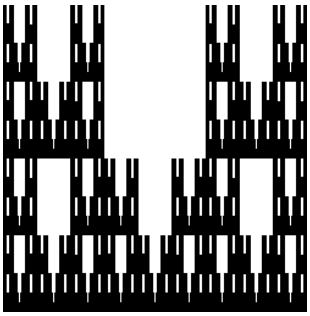

Figure 5.2: A Self Affine Fractal since the scaling in x is different than in y

5.2 Fractal Classes

There are 2 major classes that fractals may fit into.

- Self Affine - where the fractal has some dependence on scale, and is anisotropic. These fractals are generally distorted at some scales, and may not exactly repeat. See figure 5.2 for an example.

- Self Similar - This type is the more well known fractal. A self-similar fractal is defined by the similarity of the object to itself regardless of the scale observed. Simple examples of these may be found in Peitgen [24], but for quick reference see figure 5.3.

A fractal generally starts with what is called an initiator, which is followed by a generator. The initiator is any geometric shape, the generator is a procedure that acts on the initiator changing it by either adding or removing structure from the initiator. The most common examples are the Cantor set, the last line in (figure 5.4) and the

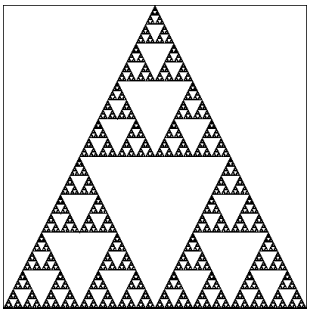

Figure 5.3: A Self Similar Fractal since the scaling in x is the same as in y

Koch curve (figure 5.5). The initiator is a line in both cases, and is included in the figures for comparison. The cantor set is an example of structure being removed from the initiator, as at each step we remove the middle third of the line from the remaining lines in the set. The Koch curve is an example of adding structure, since the middle third is removed, like in the Cantor set, however the middle third is then replaced by two lines as long as the middle third to make a triangle over the missing third from the initiator.[36] The dimension of these objects may be determined by the specific traits of the set. One pattern is consistent: if you are using the structure removing method, the dimension of the resulting set will be less than the initiator. Thus, in figure 5.4 we expect a dimension less than one, and the structure adding method will result in a dimension greater than the initiator dimension, figure 5.5 will have a dimension greater than one. Since the Cantor set has a known form of reduction we can easily calculate the dimensions. The reduction is always removing the middle third, so after each step we have two pieces remaining for each piece removed. The

Figure 5.4: The Cantor Set

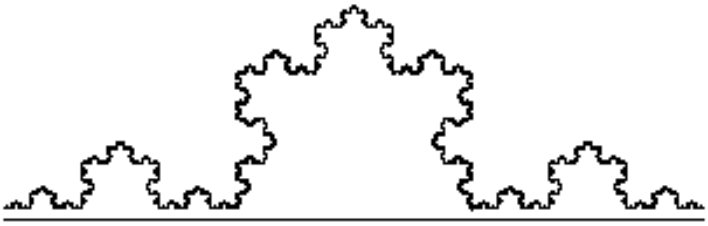

Figure 5.5: The Koch Curve

number of segments remaining then goes as $N_k = 2^k$, and the scale goes as $r_k = 1/3^k$ thus taking the log of each of these qualities and the resulting ratio we can find the box dimension, as seen in equation 5.2.1.

$$D_{Box} = -\lim_{k\to\infty} \frac{\log(N_k)}{\log(r_k)} = -\lim_{k\to\infty} \frac{\log(2^k)}{\log(1/3^k)} = \frac{\log(2)}{\log(3)} \approx 0.631 \tag{5.2.1}$$

Likewise, for the Koch curve we have the segment count at each of the curves as $N_k = 4^k$ and the scale is again $r_k = 1/3^k$ so in finding the box dimension we have;

$$D_{Box} = -\lim_{k\to\infty} \frac{log(N_k)}{log(r_k} = -\lim_{k\to\infty} \frac{log(4^k)}{log(1/3^k)} = \frac{log(4)}{log(3)} \approx 1.262 \tag{5.2.2}$$

which is exactly double the Cantor set box dimension.

5.2.1 Fractal Dimension & Stability Eigenvalues

The dimension of a system is invariant under change of coordinates, and in order to complete our analysis of the pendulum equations, we must calculate the fractal dimensions. More specifically, we look at the box and entropy dimensions.

We have established in chapter 3 that we can create a metric for a given system and that in our system the metric is the same as the Γ function. Recall;

$$\mu_i = \frac{1}{|\Lambda_i|} \tag{5.2.3}$$

where

$$\sum_i^n \mu_i = 1 \tag{5.2.4}$$

Remarkably, this definition of a metric corresponds to the probability of the trajectory of the system entering a specific region of phase space. Since the fractal dimensions are dependent on probability, we have an immediate connection. Effectively we have;

$$D_q = \frac{1}{1-q} \lim_{n\to\infty} \frac{\mu^q}{\log r^n} \tag{5.2.5}$$

where q is an index indicating the dimension: $q = 1$ for the box dimension, $q = 2$ for the Entropy dimension and so on. This generalized description of the dimension of a system is known as the Renyi dimension. Now we may define the entropy of the system in the original way;

$$Entropy = K = \sum_i^n \mu_i \log \mu_i. \tag{5.2.6}$$

The Kalmogorov entropy calculated for the systems in our case are $K = 1.193307$ for $Q = 1.2577$, and $K = 1.335254$ for $Q = 0.76$.

5.2.2 Fat Fractals

A fat fractal is defined as any fractal with a nonzero Lebesgue measure. Effectively, what this is claiming is that a fat fractal contains subsets that are measurable – they are not a set of singularities. The mathematical definition of this is: a set lying in an N-dimensional Euclidean space that for every $\mathbf{x}$ in the set and every $\epsilon > 0$, a ball of radius ϵ centered at the point $\mathbf{x}$ contains a nonzero volume of points in the set, as well as a nonzero set outside the volume element (vel).[48] We define the fat fractal since it may be useful in IFS theory, however, our systems do not use them directly.

5.2.3 MultiFractals

Simply put, a multifractal is a compilation of several fractals. That is, the multifractal in itself is not self affine or self similar, but subsets of the multifractal are self replicating. An example of such a set is the Henon Map. As seen in figure 5.6, if one looks at the smooth flow of the attractor and zoom in to a small subsection of this attractor, one would not see a repeat of the full attractor, as would be expected if the Henon map were a standard fractal, but rather one sees a series of lines. If one were to zoom in on these lines, they would find yet another series of lines, much like the original zoomed set (figure 5.7). When one really wishes to investigate IFS

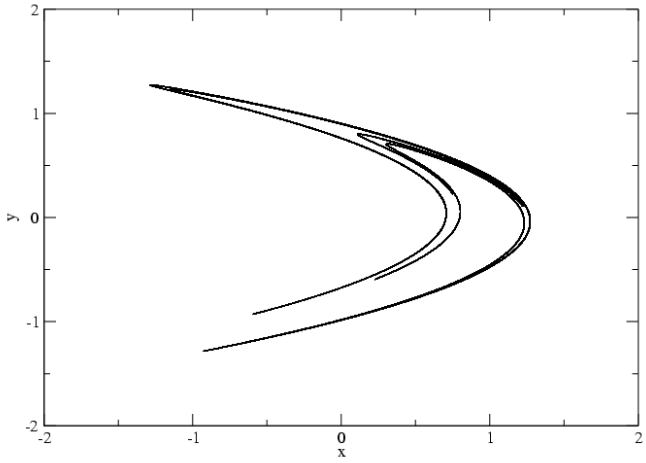

Figure 5.6: The Henon Map

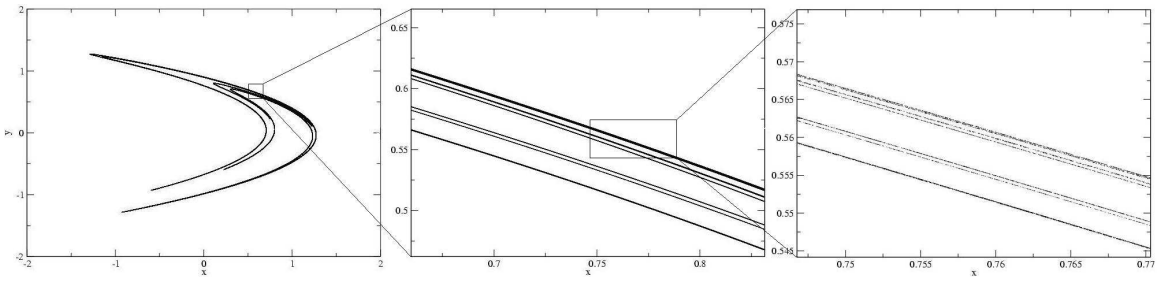

Figure 5.7: The Zoomed Henon Map

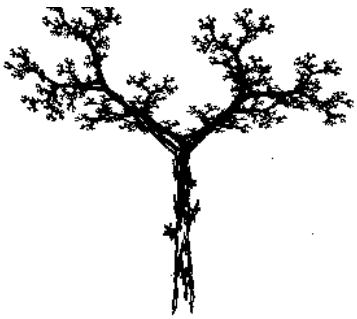

Figure 5.8: The Fractal Tree

compression, they have to look at the multifractal nature of any natural surface.

Fractals are very natural structures and come up everywhere from mountain ranges to living structures such as the fern. Even the human body is riddled with fractals, the cardiovascular system and the pulmonary system are two prime examples. Fractals can be space filling curves, and thus they are an optimal method of relaying information, or maximizing (hyper) surface area in a finite amount of (hyper) volume (especially needed in the lungs). Due to these features, the fractal became heavily studied.

5.3 Iterated Function Systems

Iterated Function Systems (IFS's) are a method of breaking down fractal images to their most basic parts and using this data for compression and image manipulation. As the name suggests, they are an iterative process that depends on a countable system of functions. From a mathematical standpoint this basically means the union

of all homeomorphisms mapping an image set onto itself. [27] An IFS is defined as a set of n functions, denoted as Ω, defined on a compact space, and the corresponding set of n probabilities which sum to unity.[34]

$$\mathcal{F} = \{\Omega, f_i : \Omega \to \Omega, p_i : \Omega \to [0,1]\} \forall i = 1, ..., n \tag{5.3.1}$$

Before we can go on to describe the iterated function systems there are a few basics that are required for understanding. The first is the idea of a metric, defined as a method by which one can determine the distance between two points in a set. We have already come across this idea when we needed to plot symbolic space, for the periodic orbits. When one is dealing with an IFS they are likely to come across the Hausdorff metric, [6], which is used to define the distance between two sets. Before we can define that, we need to know how to find the distance from a point to a set. From [57], we have the distance from a point to a set defined as;

The distance, d(x,A), between the point x $\in \mathbb{X}$ and set A$\subset \mathbb{X}$ in the metric space is given by;

$$d(x, A) = \inf_{y \in A} d(x, y) \tag{5.3.2}$$

Basically what this is saying is that the distance between a given point and the set is the minimal distance between the given point and any point in the set.

By similar logic we can define the Hausdorff metric as;

The distance between two sets for two sets A,B $\subset \mathbb{X}$ and two points $x \in A$, $y \in B$

we can define the distance between the sets A and B as;

$$h(A, B) = \sup \{d(x, B), d(y, A)\} \tag{5.3.3}$$

This says that the distance between two sets is equal to the maximum distance between points in each set. So, whichever set has the largest distance from a point in the other set, is the distance between the two sets. [27]

With the metrics defined we may now look at the idea of a contraction.

5.3.1 Contraction Mapping Principle

Let (𝕏,d) be a complete metric space and let w:$\mathbb{X} \to \mathbb{X}$ be a contraction on 𝕏. Then w has a unique fixed point in 𝕏. The proof for this is found in several sources [6] [27] and consists of induction and contradiction.

5.3.2 The Hutchinson Operator

The Hutchinson operator is defined to be a finite set of maps $\{w_i\}_{i=1}^{N}$ on a set A, which is commonly written as $\mathbf{w}$(A) or more precisely;

$$\mathbf{w}(A) = \bigcup_{i=1}^{N} w_i(A) \tag{5.3.4}$$

The definition of an IFS is a collection of contractive Hutchinson operators that are homeomorphic.

5.3.3 Fundamental Theorem of Iterated Function Systems

For any IFS $\mathbf{w} = \{w_i\}_{i=1}^{N}$ there exists a unique non-empty compact set $A \in \mathbb{R}^n$, the invariant attractor of the IFS, such that

$$A = \mathbf{w}(A) \tag{5.3.5}$$

The proof may be found in Hart [27], Barnsley [6, 43], Welstead [72]. Basically, what is being shown here is that the set must be eventually convergent, and therefore the system of equations result in an observable image set.

Possibly the most important feature of the IFS is the corollary that may be drawn from the above theorem;[27]

Corollary 5.3.1. *Let* $\mathbf{w}$ *be an IFS with attractor A, and let B be a bounded set in* $\mathbb{R}^n$. *Then* $\lim_{i\to\infty} \mathbf{w}^{\circ i}(B) = A$

This simply states that given an IFS, operating on any space of the same dimension will result in the attractor A being generated. [36, 6, 43]

Classical fractals may be obtained as attractors of appropriate IFS's. As an initial example we will turn to the 1 dimensional Cantor set. As we saw in Chapter 4, the Cantor set has a single generator, the removal of the middle third of any remaining line segments. One may characterize the generator by a set of three equations

$$w_0(x) = \tfrac{1}{3}x \quad w_1(x) = \tfrac{1}{3}x + \tfrac{1}{3} \quad w_2(x) = \tfrac{1}{3}x + \tfrac{2}{3} \tag{5.3.6}$$

where one can easily verify the fixed points of the system are 0, 1/2, and 1 respectively. For these simple equations to contain all of the information of the Cantor set, all that

is needed is a recursive calling to each of these functions. Each function will map the entire unit line into a smaller version of itself. Thus, recursively taking the last step and reducing the length by a third. If we omit the w_1 function we will obtain the Cantor set. So, the infinite nature of the cantor set can easily be captured using two simple functions, thus leading to the usefulness of IFS's for image compression. As we have done for the cantor set we may do for all fractals. The trick is to understand the mapping structure of the fractals, or rather the path by which the fractals contract and grow. These methods are applicable to higher dimensional objects, just as easily. It is all a matter of understanding the generator. Due to corollary 5.3.1, these functions may be applied to any 1 dimensional set to produce the cantor attractor, regardless of the initial length.

As seen in the Fundamental theorem of IFS, the idea of attractors is paramount to an understanding the IFS's. They require that a system has a convergence throughout the iterative process, or evolution. For the Cantor set the attractor is a infinite set of disjoint points, for the Sierpinski triangle it is a Sierpinski triangle. One of the easiest transforms to capture the convergence and evolution is the affine transformation.

Affine transformations are marked by the rotation and scaling of a system as well as a linear shift it its position. They have the general form $w(\vec{x}) = \mathbf{A}\vec{x} + \vec{b}$ These transformations are the method by which we carry out the iterated function systems, since they hold all of the possible transformation, depending on the value of the

constants.

$$w_i(x,y) = \begin{pmatrix} r_i & \theta_i \\ \phi_i & s_i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix} \tag{5.3.7}$$

where r_i and s_i are scale factors for the coordinates x and y respectively. θ_i and ϕ_i are skew factors, they induce rotations and skewness. e_i and f_i are simple linear shifts in image space. All of the mentioned values are constants in the affine transformation. These six parameters are often referred to as the IFS code for a given attractor. There are direct analogies for three, or higher dimensional affine transformations, these will be discussed in Chapter 6.

To continue a previous example, the Cantor set may be extended to a 2 dimensional analog, a cantor sheet. For this type of strange attractor, we again need four transformation like in the 1 dimensional case. Here we need

$$\begin{aligned} w_1(x,y) &= \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ w_2(x,y) &= \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 2/3 \end{pmatrix}, \\ w_3(x,y) &= \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2/3 \\ 0 \end{pmatrix}, \\ w_4(x,y) &= \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2/3 \\ 2/3 \end{pmatrix}. \end{aligned} \tag{5.3.8}$$

Which appears to be a simple expansion of the 1 dimensional system. Higher order cantor sets can be calculated using similar techniques. In this example there are no rotational properties in the matrix **A**.

Here is where the probabilities come into play. One decides the probability that

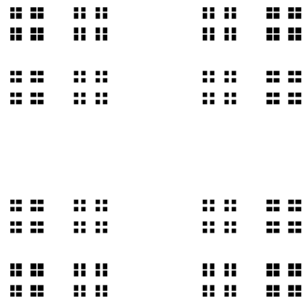

Figure 5.9: Output of equation 5.3.8

	a	b	c	d	e	f
A_1	0	0	0	0.16	0	0
A_2	0.85	0.04	-0.04	0.85	0	1.6
A_3	0.2	-0.26	0.23	0.22	0	1.6
A_4	-0.15	0.28	0.26	0.24	0	0.44

Table 5.2: IFS code for Barnsley's Fern [43]

any given portion of a fractal becomes iterated, and each section is produced at a different rate. Eventually the entire fractal will be reproduced, but the more important parts of the fractal will be given a higher probability, depending on the desired effect. For a common example one can look at the Barnsley Fern, which has a IFS transformation seen in table 5.2;

Now, as we see in table 5.2 the probability of each transformation is $p_1 \approx 0.0001$, which was actually zero, however the rules given by Barnsley [6] say that a zero probability will be given a comparatively small probability. This method is justified since rounding considerations will cause the sum of probabilities to be less than one.

Figure 5.10: Barnsley's Fractal Fern

Following the form $p_i = (A_i)(\det \mathbf{A})$ where A_i is the ith element in the matrix A. $p_2 = 0.7730$, $p_3 = 0.1108$, $p_4 = 0.1161$ and, as we scale down each sub-square with the same probability distribution, (our generator), we will have the desired effect.

This may be seen graphically through the histogram method described in [20] for the Sierpinski triangle. We start with the probability distribution;

$$T = \begin{array}{|c|c|} \hline 0.3 & 0.3 \\ \hline 0.1 & 0.3 \\ \hline \end{array} \tag{5.3.9}$$

Then we produce the second iterate of the system by taking the tensor product

of T with itself;

$$T \otimes T = \begin{array}{|c|c|c|c|} \hline 0.09 & 0.09 & 0.09 & 0.09 \\ \hline 0.03 & 0.09 & 0.03 & 0.09 \\ \hline 0.03 & 0.03 & 0.09 & 0.09 \\ \hline 0.01 & 0.03 & 0.03 & 0.09 \\ \hline \end{array} \tag{5.3.10}$$

We then continue to take the tensor products $T \otimes T \otimes T...$ for each iterate. The first five iterates may be seen in figure 5.11.

Obviously, the highest probability regions form the Sierpinski gasket, the lower probability regions form the removed sections. For a crisp image at the predefined truncation of the infinite calculation, one simply needs to remove everything except the highest probability regions. Often this is done naturally by extending the diminishing probabilities well below computational limits. For a well behaved system this may take only a few iterations.

Barnsley introduced the IFS procedure [6] and consequently found that many simple fractals that are made up of an uncountable set could be reproduced by preserving only a few numbers. A comprehensive list of the values may be found in several references [24, 43]. This type of setup is great for what may now be considered standard fractal shapes, but what of the more complicated fractals such as a fern, seen in figure 5.10? Barnsley's investigations have shown that the fractal fern, which does look somewhat like a real fern, can be completely captured using only 24 numbers, 4 sets of 6 values. Once again, we have seen that a fractal of great resolution may be reproduced in full with a very small amount of information. It is obvious from this

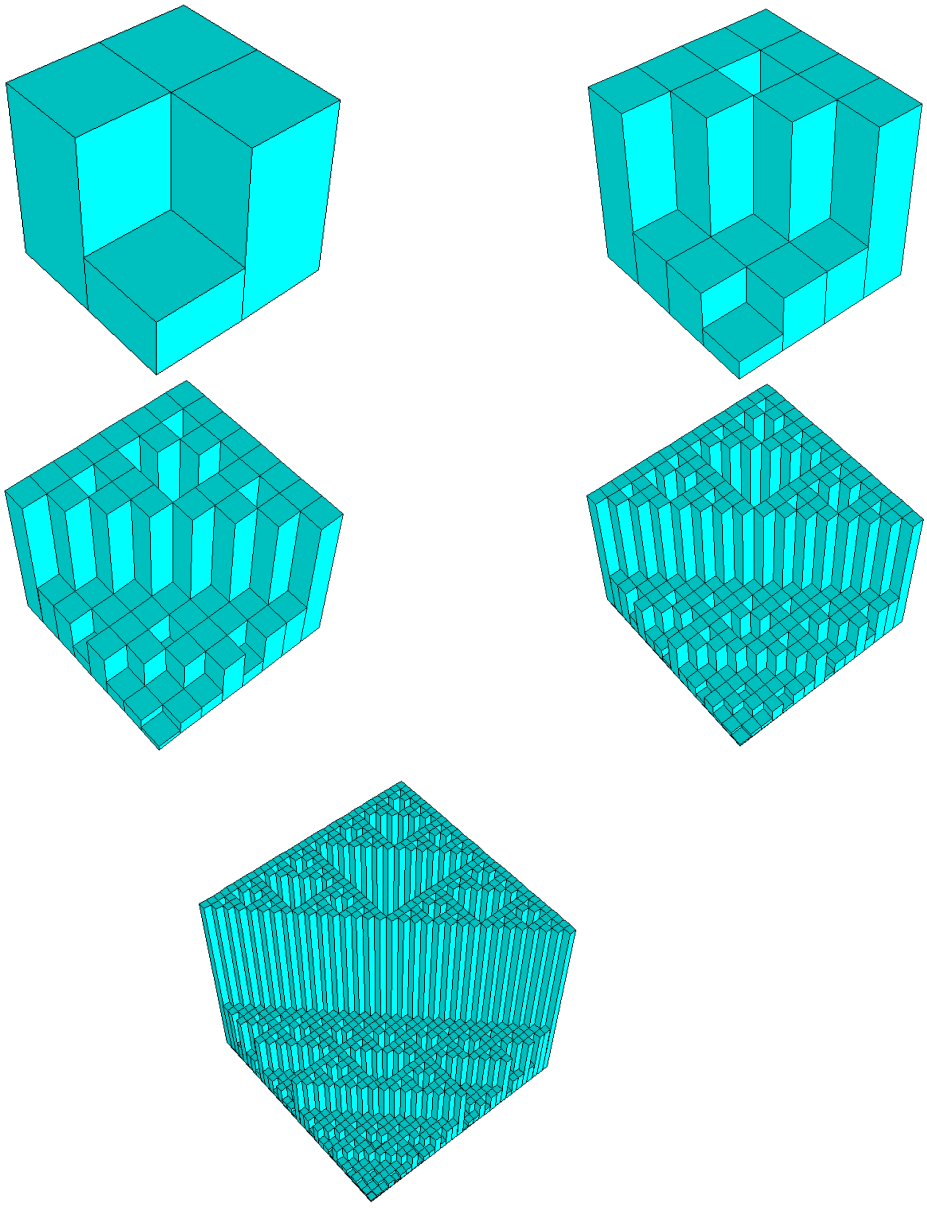

Figure 5.11: The first five iterates of the Sierpinski probability square

that fractal compression techniques are able to achieve incredible compression ratios. Another such "real" image is the fractal tree as seen in 5.8

Along with the IFS structure there comes a predefined natural measure. Simply put, this probability is calculated as;

$$p_i = \frac{|A_i|}{\sum_{i=1}^{n} |A_i|} \tag{5.3.11}$$

where $|A_i|$ is the determinant of A_i. It is clear that the sum of the probabilities goes to unity, as expected.

If the probability p_i is zero, we assign a small value to it. This small value may be defined by the sum of the probabilities of the remaining transformations. Due to rounding errors this will likely be less than 1, so the zero probability may be adjusted to fill in for this error. [6] With this done, we have that the probability for the four transforms that produce the 2 dimensional cantor set, have equal probability, at $p_i = 1/4$.

Probability has a firm footing in the practicality of the random reconstruction algorithm. The randomness is slightly skewed towards higher probability transformations. That is, in the cantor set all transformations have equal probability, so the algorithm may call these each equally. However, for systems with different probabilities, the weighting of the randomness must tend to call the higher probability transformations more frequently. [72, 43] This is easily done with proper random number generators.

The second issue follows the adage that you cannot get something for nothing. What we gain in compression we lose in time. To produce the fractal fern out of IFS's, with a good resolution, one would need a significant amount of time, 10^{10} years [24]. Thus, methods again must be developed to obtain the original picture in polynomial time. This is only true for the deterministic method of reconstruction. As stated earlier the random algorithm reproduces the attractor faster since the more significant parts, the ones with higher probabilities, are repeated more often. So for our perception of the attractor, the random algorithm is the better method.

The fact that one could make complicated geometric structures simplify to only a small amount of information is a great achievement, but how practical is this ability? How much information do we really have that is based on pure self-similar fractal structures such as the Sierpinski triangle? The truth is, not a lot. So then why are we still studying this material? What is of practical importance is the inverse problem. Taking a picture, finding its fractal nature, and then storing this information in a condensed form. Effectively, determining the IFS codes is like trying to determine a generator that will take a simple initiator to the desired attractor, the picture being compressed. This way we can record only a simple picture and the transformation codes to get there. The interesting part is that one does not know the components of the affine transformations, nor do they know how many are needed. Along with this are methods of interpolation so that the recovery of the stored images may be even

better. For work done on this matter, see [19], or Barnsley [6, 43, 5].

There are some fundamental requirements of determining IFS codes. Simply put, one must be able to see a recursive pattern in a picture. That is, take the original picture and segment it, only to see the segments are smaller versions of the whole picture. Once this has been performed, one must now find suitable fixed points in the picture. This process is basically best left to trial and error. Finding patterns in a picture is easily done, however finding the appropriate fixed points can be difficult. [72] Observe the figures 5.12,and 5.13, for a visual description. With this process there are two methods by which the attractor may be produced. The first is a deterministic algorithm, where each of the transforms recorded occur an equal number of times, and in a cyclic pattern. This method can take a long time to reproduce the attractor since it covers the attractor with greater and greater detail over each cycle of the transformations. The second algorithm is the random algorithm, where one takes each transformation, at basically random levels of detail, thus covering the entire attractor in fewer steps than the deterministic algorithm. However what is gained in speed is lost in resolution. If we consider again that human eye resolution does not require a great deal of detail to see an image, the random algorithm is often sufficient for reproduction of images, whereas in the deterministic algorithm the same number of steps may not cover enough of the attractor to be recognizable. The random number algorithm calls each of the IFS's at random, producing what is referred to as

a trajectory through the image space. This trajectory is seen as;

$$\mathbf{x}_n = w_i(\mathbf{x}_{n-1}) \tag{5.3.12}$$

so, as each transformation is called we get a set of points, referred to as an orbit. If the system is truly driven by a random sequence of values, we will have a non-repeating sequence [72].

Part of the idea for finding the fractal nature of any system comes from the collage theorem that simply states that any image may be created from subsections of its parts. A graphical display of this is seen in figures 5.12, 5.13, and 5.14. The different colour in the reconstructed image are only for emphasis.

Each part, if layered properly, would be completely reproduced. Thus the image space would consist of a small collection of sub pictures, and a mapping for the sub-pictures. From a topological standpoint we are trying to produce a minimal set of allowed structures, or rather we are finding a basis set. Mathematical definitions of the collage theorem may be found in several references [43, 27]. So, when we save a fractal image as only a few constants we save a multifractal as an independent set of single fractal constants, as well as a map joining the sets. This was evident in the examples in figure 5.13 and 5.14. The leaf itself could be treated as a self affine fractal but the stem was a separate entity. A decent reconstructed image will require a good choice of initial image. What is seen in 5.14 is a poor choice of partial image, since the reconstructed image looks very different from the original in a few steps, whereas

in figure 5.13 the choice reproduces the image very well over several steps.

The procedure laid out works well for two dimensional two colour image spaces. However, for practical use this would be very limited. An extension to at least grey-scale images is required. This may be seen in [67] where they describe the extension as the affine transformation;

$$W\begin{pmatrix} x \\ y \\ z(x,y) \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & p \end{pmatrix}\begin{pmatrix} x \\ y \\ z(x,y) \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ q \end{pmatrix} \tag{5.3.13}$$

The upper-left sub-matrix, with the upper sub-vector is the already introduced affine transformation, with identical properties. The values p and q that are introduced are defined to be the contrast and brightness, respectively which simply control the third vector entry, z, the shade of grey. There are several good papers on the topic, see [9], [17], [23] to name a few. It is this type of transformation that we hope to exploit in Chapter 6.

5.4 Fractal Interpolation

Interpolation is a common method for determining information between two plotted points. This method is very commonly employed in real data structures, where only a few pieces of information are well known and the data in between is unknown, thus the use of formulae for interpolation. Anyone that has performed data fitting, has effectively performed an interpolation. One must understand the set of points being

used, clearly a linear interpolation will provide poor results if the original set was most likely, say parabolic. So when we define our system, the types of interpolation will have to correspond to the way we treat the data points.

The IFS is reliant on an attractor of the system. Fortunately, one may easily assess the attractor of any given system simply because they can define it themselves. With the random IFS the attractor of a system may be defined as a function that passes through a few fixed points of the system. [59]

What may been seen here is that the regular IFS requires at least 3 fixed points to define the IFS code. Here we are simply applying the same requirements on the system. As mentioned above, these fixed points are necessities of the system as they are the points by which the system can begin the interpolation. In general, one expects that the higher the order of interpolation, the greater the number of fixed points needed. The simplest such affine transformation is

$$w_n \begin{pmatrix} t \\ x \end{pmatrix} = \begin{pmatrix} a_n & 0 \\ c_n & 0 \end{pmatrix} \begin{pmatrix} t \\ x \end{pmatrix} + \begin{pmatrix} e_n \\ f_n \end{pmatrix}, \tag{5.4.1}$$

which follows from the simplest fractal to model, the cantor set. Here we impose the endpoint, or fixed point, conditions;

$$w_n \begin{pmatrix} t_0 \\ x_0 \end{pmatrix} = \begin{pmatrix} t_{n-1} \\ x_{n-1} \end{pmatrix} \tag{5.4.2}$$

$$w_n \begin{pmatrix} t_N \\ x_N \end{pmatrix} = \begin{pmatrix} t_n \\ x_n \end{pmatrix} \tag{5.4.3}$$

where N is the total number of points.

What we have done in 5.4.1 is scale the x and replace y with another scale of x and given them some linear shift, e_i and f_i. All that is left is to combine the end points with the affine transformation, to determine the interpolation parameters as follows;

$$a_n = \frac{t_n - t_{n-1}}{t_N - t_0}, e_n = \frac{t_N t_{n-1} - t_0 t_n}{t_N - t_0}, c_n = \frac{x_n - x_{n-1}}{t_N - t_0}, f_n = \frac{t_N x_{n-1} - t_0 x_n}{t_N - t_0} \tag{5.4.4}$$

What is interesting here is that we have managed to produce a system of transformations with a zero determinant. Therein we know 2 things about the system. The first is that the probability for each transform is very small, according to the rule generated by Barnsley in [6]. Since the probabilities are all equal this system is a prime candidate for the random algorithm. The second is that the system in non-invertible, therefore decoding the system requires knowledge of the entire system, including initial conditions, not just the output of the transforms. Thus we may have a secure system for information passage.

What we may be interested in is the contractivity of the system. To determine this we take the definition of contractivity;

$$d(\omega(x), \omega(y)) \leq sd(x, y) \tag{5.4.5}$$

for $s \in \mathbb{R}$

So, we simply take two arbitrary points in the image space $\mathbb{S}$, (x,y) and a ball

of radius ϵ to find a second point. We receive the general distance formula for 2 dimensions, $p \geq 1$, depending on the metric selected.

$$d(x, y) = ((x_1 - x_2)^p + (y_1 - y_2)^p)^{1/p}$$

which, with our selected points, becomes

$$d(x, y) = \epsilon 2^{1/p}$$

and, after the affine transformation,

$$d(\omega(x), \omega(y)) = ((a\epsilon)^p + (c\epsilon)^p)^{1/p} = \epsilon(a^p + c^p)^{1/p}$$

so for comparison between the two distances as seen in equation 5.4.5

$$(a^p + c^p) \leq 2s \tag{5.4.6}$$

Now, clearly the contraction is independent of the original displacement between the two points in $\mathbb{S}$. Further, we must see that the value s has to be less than 1 for a contracting set. If the contractivity is 1, then our functions are similtudes, a set that may not contract any further. To check this feature we have to look at the values setup for our constants a and c, as seen in equations 5.4.4. We know that $\inf(\mathbb{S}) = (t_0, x_0)$ and that $\sup(\mathbb{S}) = (t_N, x_N)$ so that $t_N - t_0$ is a maximal displacement in the set. Consequently, we also have that the set $\mathbb{S}$ is well ordered, the points within are ordered such that $t_0 < t_{n-1} < t_n < t_N$ and likewise for x. Thus, the values for

a_n and c_n are less than 1. Since both values are less than 1, taking them to a power equal or greater than one, produces a smaller value. Thus the equation 5.4.6 is at a maximum when p=1, and since the sum of two values less than one will always be less than 2, we have a contraction for all values p. This validates our claim that the system defined above is an IFS.

Thus with only a small piece of information one may determine the interpolation of nearest neighbour points. Further extensions may be made to higher order interpolations using K+1 initial points. That is the linear (1D) required two points, quadratic (2D) requires three points etcetera.[59] Again using similar solution methods as the linear case, one can find the equation of transformation in terms of the starting points. The graphs of the interpolating functions may be produced by the random iterative function[59]

- `initialize` (t, x) `to a point in the interval of interest`
- `for a set number of iterations`
- `randomly select a transformation` $w_n(t, x)$
- `plot` $(t_0, x_0) = w_n(t, x)$
- `set` $(t, x) = (t_0, x_0)$
- `end for`

The interpolation functions for 1 dimensional systems requires 1 parameter variable, ie a variable that is well defined. A 2 dimensional system requires two parameter

variables. This trend is obvious but incredibly important to formulating the theory for higher dimensional systems. The higher dimensional systems have a further level of complication. They require an exponential number of fixed points for interpolation. That is for the 1D system we required 3 fixed points for the quadratic interpolation, however for the 2D system we require 12 fixed points with the same interpolation. This pattern may be met with $3 \cdot 2^2$ where the 3 is the "standard" 1D number of fixed points, and 2^n are the requirements of an n-dimensional interpolation. This leads us to a tradeoff, the better the interpolation the more points are needed, likewise with higher dimensions, even the simplest interpolation could require a large number of points. The variables that may be solved for the quadratic system may be solved to be as seen in [58], which is not necessarily the nicest solution but it is far less complicated than the alternative. What must be keep in mind is the fact that we cannot get more out of the picture than was originally there [59]. Basically, if the details of the picture were non-existent in the original picture they can not show up in scaled versions, despite what popular crime shows may exhibit. A popular example of this is using a mirror in a picture to fully identify an assailant, especially when they mix it with a 3 dimensional rotation to see around the frame of the mirror. With the interpolation functions one may not be able to get the exact picture quality when blown up but they will be able to determine a fair approximation. This is seen in Chapter 6 with the standard picture of Lena.

5.5 Conclusion

In this chapter we have presented the ideas and observations that lead to the concepts of fractal geometry. We have discussed the link between the concepts of chaos and fractals, and given an example of such a relationship through the system observed in chapters 2, 3 and 4. In the following chapter we will take the work done on fractals to yet another form of image manipulation, magnification.

Figure 5.12: A Canadian Maple Leaf

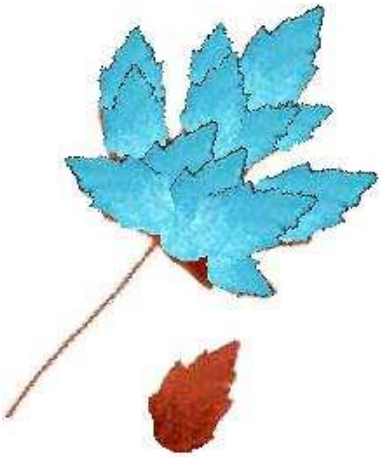

Figure 5.13: A Canadian Maple Leaf using a small tip of the main lobe to reconstruct. A second image would be necessary for the stem.

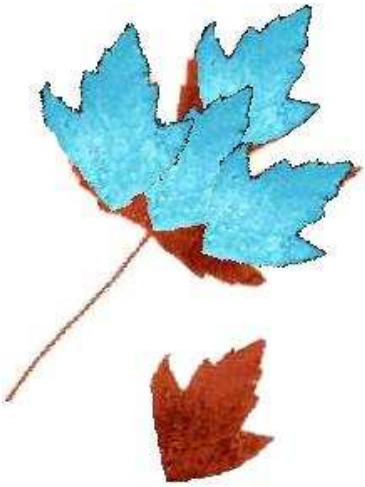

Figure 5.14: A Canadian Maple Leaf using a larger piece of the main lobe to reconstruct. Again a second image is needed to reconstruct the stem.

Chapter 6

Iterated Interpolation Function Systems

6.1 Introduction

An Iterated Function System (IFS), as defined in Chapter 5, may be used to construct fractal interpolation functions for a given set of data. This same system may be used for more immediate practical purposes. That is, to magnify or enlarge a picture, with moderate to high definition. By the same principle, it is also possible to shrink a picture. The important feature of this method is that it requires real image spaces for a high quality output. Artificial spaces, such as cartoons and geometric shapes will work but with very limited quality output.

A two dimensional interpolation acting via a two dimensional IFS is necessary to map a two dimensional black and white picture onto a two dimensional map. Since grey-scale images have RGB values all equal, any calculation that happens to one colour coordinate, is treated as happening to them all, and the complexity

of calculations easily reduce to two dimensional mappings. If this greyscale IFS is expanded to three dimensions, using the same two dimensional interpolation, a picture with colour may be properly mapped. If the third dimension is itself treated as a three component vector representing the red, green, and blue components of a colour, a full colour picture may be properly transformed.

With the use of the 2 dimensional IFS, one can easily produce a 3 dimensional IFS. Likewise, with the use of the 2 dimensional case and the 2 dimensional higher order interpolation, a generalization may be drawn for k - degree interpolation accuracy.

6.2 Linear Two Dimensional Interpolation Functions

First of all, we understand that the two dimensional linear interpolation between points may be written as follows:

$$y = \frac{x - x_0}{x_1 - x_0} y_1 + \frac{x_1 - x}{x_1 - x_0} y_0 \tag{6.2.1}$$

with initial conditions $y(x_0) = y_0$ and $y(x_1) = y_1$.

The three dimensional linear interpolation between points would then be written as:

$$\begin{array}{c} z = \frac{(x-x_0)(y-y_0)}{(x_1-x_0)(y_1-y_0)} z_{1,1} + \frac{(x_1-x)(y-y_0)}{(x_1-x_0)(y_1-y_0)} z_{0,1} + \\ \frac{(x-x_0)(y_1-y)}{(x_1-x_0)(y_1-y_0)} z_{1,0} + \frac{(x_1-x)(y_1-y)}{(x_1-x_0)(y_1-y_0)} z_{0,0} \end{array} \tag{6.2.2}$$

The 1 dimensional linear fractal interpolation involves solving an IFS where

$$w_n \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_n & 0 \\ b_n & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_n \\ f_n \end{pmatrix}, \tag{6.2.3}$$

given the initial, or fixed point, conditions

$$w_n \begin{pmatrix} x_N \\ y_N \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} \quad w_n \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix}.$$

When we consider that the dimension of the image space is $M \times N$ there are 3 times as many initial conditions to be considered. M is the number of pixels in the width of the image, and N is the number of pixels in the height of the image.

The 2 dimensional linear interpolation involves solving an IFS where

$$\begin{aligned} W_{m,n}(x) &= a_m x + e_m \\ W_{m,n}(y) &= b_n y + f_n \& \\ W_{m,n}(z) &= A_{m,n}x + B_{m,n}y + C_{m,n}xy + D_{m,n}, \end{aligned} \tag{6.2.4}$$

where A,B,C, and D are coefficients to be determined, given the initial conditions

$$\begin{aligned} W_{m,n} \begin{pmatrix} x_M \\ y_N \end{pmatrix} = \begin{pmatrix} x_m \\ y_n \end{pmatrix} \quad W_{m,n} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_{m-1} \\ y_{n-1} \end{pmatrix} \\ W_{m,n}(z_{M,N}) = z_{m,n} \qquad W_{m,n}(z_{M,0}) = z_{m,n-1} \\ W_{m,n}(z_{0,N}) = z_{m-1,n} \qquad W_{m,n}(z_{0,0}) = z_{m-1,n-1} \end{aligned} \tag{6.2.5}$$

where we have used $z_{m,n} = z(x_m, y_n)$ for compactness.

Here x_m takes on the linear interpolation

$$x_m = \frac{x_{n-1} - x_{n-2}}{x_n - x_{n-2}} x_N + \frac{x_{n-1} - x_n}{x_{n-2} - x_n} x_0 \tag{6.2.6}$$

of the fixed point between the 2 known end points. The definition for y_m is similar.

The coefficients in the 1 dimensional case were solved for in terms of initial conditions, and likewise for the 2 dimensional case coefficients. After some simple mathematics the coefficients were solved to be:

$$\begin{aligned} a_m &= \tfrac{x_m - x_{m-1}}{x_M - x_0} & b_n &= \tfrac{y_n - y_{n-1}}{y_N - y_0} \\ e_m &= \tfrac{x_0 x_m - x_{m-1} x_M}{x_0 - x_M} & f_n &= \tfrac{y_0 y_n - y_{n-1} y_N}{y_0 - y_N} \end{aligned} \tag{6.2.7}$$

and the z component coefficients were:

$$A_{m,n} = \frac{(z_{m,n-1} - z_{m-1,n-1})y_N + (z_{m-1,n} - z_{m,n})y_0}{(x_M - x_0)(y_N - y_0)} \tag{6.2.8}$$

$$B_{m,n} = \frac{(z_{m-1,n} - z_{m-1,n-1})x_M + (z_{m,n-1} - z_{m,n})x_0}{(x_M - x_0)(y_N - y_0)}$$

$$C_{m,n} = \frac{z_{m-1,n-1} - z_{m,n-1} + z_{m,n} - z_{m-1,n}}{(x_M - x_0)(y_N - y_0)}$$

$$D_{m,n} = \frac{z_{m-1,n-1}x_M y_N - z_{m,n-1}x_0 y_N + z_{m,n}y_0 x_0 - z_{m-1,n}x_M y_0}{(x_M - x_0)(y_N - y_0)}$$

6.2.1 Simplify

The problem of working out the above fixed points for the $z' = W_{m,n}(z)$ transformation is quite tedious, and highly prone to mistakes. Fortunately, a shortcut was determined, it may not make the coding any simpler, but does prove to make solving the systems of equations rather trivial. Basically, all one needs to do is calculate z'=x'y', where $x' = W(x)$ and $y' = W(y)$ as seen above. Effectively, what will be worked out is that the following relations will be evident, say for the linear transformation,

$$A_{m,n} = a_m f_n \quad B_{m,n} = b_n e_m \quad C_{m,n} = a_m b_n \quad D_{m,n} = e_m f_n \tag{6.2.9}$$

with a_m etcetera as seen in 6.2.7. When using this method one must keep in mind that the resulting product substitution must be made before programming $x_m y_n = z_{m,n}$ for all values of x and y that are not endpoints. Furthermore, use of this information, and the contraction proof for the 2 dimensional grey scale picture proof in Chapter 5, we

have that the transformation above is also contracting. We may then assume without loss of generality that the product of two contracting systems is also contracting.

Another observation of the values calculated for the coefficients of the transformation is that we know the resulting value for the x coordinate will be located between x_0 and x_M and likewise for y between y_0 and y_N. Thus, we expect the indices on z to reflect this behaviour. In particular, we have that the scaling value, say for the coefficient $B_{m,n}$, that the x_M will not allow the present position to be taken into consideration, for the first index on any of the multiplying z values. This also applies to the y multipliers. The x_0 multiplier appears to force the restriction that only the current position may be considered for the first index on z. This takes care of the positions allowed to be considered, however there is still the issue of the sign. What can be done for the sign is that the z values, based on their index, can be written as a matrix. Considering the restrictions above, one may remove the disallowed z values. Now, if there are an even number of multipliers (remember that zero is even) the elements of the remaining sub-matrix will take on a negative of the sign matrix. If there are an odd number of multipliers then the elements of the remaining sub matrix will take on the sign from the sign matrix.

6.3 Quadratic Two Dimensional Interpolation Functions

The interpolation technique may also be extended to the quadratic interpolation. One method of doing this might follow a quadratic interpolation of the position components, which follow;

$$x' = a_m x_m^2 + b_m x_m + c_m \tag{6.3.1}$$

$$y' = d_n y_n^2 + e_n y_n + f_n$$

and the quadratic interpolation of the colour,

$$z' = A_{m,n}x_m^2 y_n^2 + B_{m,n}x_m^2 y_n + C_{m,n}x_m y_n^2 + D_{m,n}x_m^2 + E_{m,n}y_n^2 + F_{m,n}x_m y_n + G_{m,n}x_m + H_{m,n}y_n + K$$

This method may very well be useful for image analysis, however it falls away from iterated function systems, specifically it does not follow an affine transformation. Clearly a different method is needed to attempt an IFS approach.

6.4 Partial Quadratic Two Dimensional Interpolation Functions

The interpolation techniques described above may be mixed resulting in a linear interpolation for the coordinates of the image space, and a quadratic interpolation of the colour at each location.

$$x' = a_m x_m + e_m \tag{6.4.1}$$

$$y' = b_n y_n + f_n$$

$$z' = A_{m,n}x_m^2 y_n^2 + B_{m,n}x_m^2 y_n + C_{m,n}x_m y_n^2 + D_{m,n}x_m^2 + E_{m,n}y_n^2 + F_{m,n}x_m y_n + G_{m,n}x_m + H_{m,n}y_n + K$$

With this method we are still obeying an Affine transformation, since the coordinates are the only part of the vector that need to obey any such rules, in order to still be considered affine.

As with the linear case the 1 dimensional solutions will be utilized to work out a solution for the two dimensional case. For the 1 dimensional case the coefficients for the x and y components of the transformation are the same as in 6.2.7.

The resulting coefficients for the colour transformation are a little more complicated to work out. A simplification arises when one looks at the z subscripts and the multiplying coordinate subscripts. Again we consider the multiplier index. If the index is the maximal position, we have that the corresponding z index cannot be the current position. If the index is minimal, the corresponding z index is not allowed to be 2 less than the current position. Finally, if the index of the multiplier is the current position, then 1 less than the current position may not be considered. This generalization of the system can greatly reduce the amount of work to be done in calculating the coefficients. But as in the linear interpolation we also must consider the sign of the z values. Just as in the linear case we can write the z values into a

matrix based on the subscripts. Then considering the restrictions stated above, we will be left with a sub-matrix consisting of only allowed positions for z. If there are an even number of multipliers, and the indices of the multipliers are the same, the off diagonal values are negative. If the indices differ by one, the off diagonals become positive. Finally, if the indices differ by two then the off diagonals are negative. Further relations for odd number of multipliers show that the opposite signs holds true. This can be easily stated in an equation if we allow the following to be true;

Let us order the fixed points in the image space 0→0, m or n→1, and M or N→2. Furthermore, we write the z values in matrix form, based on their subscripts. Now using standard matrix form we are left with;

$$\sum_{k,w=0}^{l} z_{m-k,n-w}(-1)^{i+j+q+p+h} \tag{6.4.2}$$

where h is the order of interpolation, q is the number of multipliers, k is the fixed point index for the x coordinates, w is the same for the y coordinates, and p is defined as

$$p = \begin{cases} 1, & \text{if the index on the multipliers are different;} \\ 0, & \text{if the index on the multipliers are the same.} \end{cases}$$

Again, we have to keep in mind that the multiplier subscripts will determine which of the matrix elements are not allowed. This method is general for any order interpolation. It is a useful tool for determining the coefficients, since the actual computation based on the initial conditions is lengthy and is incredibly error prone when coding. This may be used as a check against potential errors.

As an example we take the coefficient $E_{m,n}$ for the quadratic case, where we expect four multipliers, y, and 3 x's. So we are left with;

$$E_{m,n} = \sum_{a,b,c=0}^{2} x_a^2 y_b x_c z_{i,j} (-1)^{i+j+4+p+2} = \sum_{a,b,c=0}^{2} x_a^2 y_b x_c z_{i,j} (-1)^{i+j+p} \tag{6.4.3}$$

where in this case $z_{i,j}$ is written as;

$$z = \begin{pmatrix} z_{m-2,n} & z_{m-1,n} & z_{m,n} \\ z_{m-2,n-2} & z_{m-1,n-1} & z_{m,n-1} \\ z_{m-2,n-2} & z_{m-1,n-2} & z_{m,n-2} \end{pmatrix} \tag{6.4.4}$$

and we will end up with terms where the multipliers are, for example $x_M^2 y_0 x_m$ so we cannot allow terms that have a two step separation in the y index, (for y_0 we do not allow $n-2$ subscripts). We also do not allow a one step separation in the x index, nor do we allow a zero step separation in the x index (due to x_m, and x_M respectively.)

As complicated as this method is, it is less complicated than programming the results from the endpoint mappings. Since there are nine constants in this case, higher order interpolations will push the calculation limits beyond a computer's computational ability and other methods of solution would be necessary.

The program as described above was implemented using the linear interpolation method and then the quadratic method, and the resulting images were compared for resolution. Further comparisons are made when one looks at the traditional linear extrapolation. The fractal interpolation methods resolution is much superior to the traditional linear interpolation method. These are seen in figures 6.6 to 6.4.

It should also be mentioned that the magnification of the original picture is limited to the hardware of the PC used. Several magnifications were attempted on a test picture and the resolution of the magnifications were acceptable. When comparing the linear method to the quadratic it was easily seen that the difference between these methods became more apparent at larger magnifications. It was assessed that with a higher degree interpolation the resolution could be improved even further. All of these pictures were not included here to save space.

In figures 6.11,6.14, and 6.17, clearly one can see that the higher the order of interpolation the better looking the resulting picture. The fuzzyness exhibited by the quadratic magnification is a feature of the interpolation, and would be remedied by further iterations. Obviously the linear interpolation does not have the same degree of fuzzyness, but the tradeoff here is the pixelation of the magnified picture makes the image unrecognizable at higher magnifications.

6.5 Higher Degree Interpolation Techniques

For the two dimensional systems, in a higher order interpolation we need to consider the same transforms for the spatial dimensions as was done in the past two techniques. The colour dimension will be the only dimension changed. We will end up with a system where;

$$w_n(x) = a_n x + e_n \tag{6.5.1}$$

$$w_n(y) = c_n y + f_n$$

$$w_n(z) = C_n^{(0,0)} + C_n^{(1,0)} x + C_n^{(1,1)} xy + C_n^{(2,1)} x^2 y + ... + C_n^{(k,k)} x^k y^k$$

where k denotes the degree of interpolation.

$$w_n \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \begin{pmatrix} x_{n-k} \\ y_{n-k} \\ z_{n-k,n-k} \end{pmatrix} \text{ and } w_n \begin{pmatrix} x_{m+1} \\ y_{m+1} \\ z_{m+1} \end{pmatrix} = \begin{pmatrix} x_{n-k+1} \\ y_{n-k+1} \\ z_{n-k+1,n-k+1} \end{pmatrix} \text{ \& ... \&}$$

$$w_n \begin{pmatrix} x_N \\ y_N \\ z_N \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ z_{n,n} \end{pmatrix}$$

Calculating this system of equations will devastate a computer, and basically anyone trying it by hand. The method of solution shown in general for the quadratic interpolation method will have to be implemented. However, there is a cost of high level interpolations that is the number of fixed points needed. Such high level interpolations will require a larger image space to operate on, as well as a high correlation between pixels. If this correlation is not met the picture will appear distorted regardless of the number of iterations performed.

6.6 Potential Applications

As an application of this work, in this section we consider the task of re-scaling a given colour space. This is a natural problem for an interpolating function of two variables (x, y), the pixel coordinates. The function $\mathbf{z}(x, y)$ in this case is a vector-valued function having three components representing the RGB value of the pixel

specifying the amount of red, green, and blue present. The procedure used to scale an image of size M pixels wide by N pixels high is as follows. We first read in the RGB values of each pixel of the image, and use that as the data to construct a fractal interpolating function $\mathbf{z}(i,j)$, where i = 1, 2, ... ,M and j = 1, 2, ... ,N. Then to resize the image so that the resulting image is of size $s_xM \times s_yN$, we construct a new fractal interpolating function $z'(i,j) = z(s_xi, s_yj)$. Applying the random iteration algorithm to $z'(i,j)$, choosing independently a transformation index (i, j) at each stage, will then result in the re-scaled image. Simply we re-arrange the information from the fractal interpolations so to take on the appearance as seen in Kobes [58], which is re-capitulated here,

$$w_{mn}(x) \equiv x^* = \frac{x - x_0}{x_M - x_0}x_m + \frac{x - x_M}{x_0 - x_M}x_{m-1} \tag{6.6.1}$$

$$w_{mn}(y) \equiv y^* = \frac{y - y_0}{y_M - y_0}y_m + \frac{y - y_M}{y_0 - y_M}y_{m-1}$$

$$w_{mn}(z) \equiv z^* = \frac{(x^* - x_{m-1})(y^* - y_{n-1})}{(x_m - x_{m-1})(y_n - y_{n-1})}z_{m,n} - \frac{(x^* - x_m)(y^* - y_{n-1})}{(x_m - x_{m-1})(y_n - y_{n-1})}z_{m-1,n} - \frac{(x^* - x_{m-1})(y^* - y_n)}{(x_m - x_{m-1})(y_n - y_{n-1})}z_{m,n-1} + \frac{(x^* - x_m)(y^* - y_n)}{(x_m - x_{m-1})(y_n - y_{n-1})}z_{m-1,n-1}$$

What we have done here is re-ordered the coefficients found in section 6.2 to produce a functional form for the colour transformation. Since the x^* and y^* components are the same under our affine transformations, we can re-write the z transformation for

the quadratic fractal interpolation as;

$$
\begin{aligned}
w_{mn}(z) \equiv z^* = {} & \frac{(x^*-x_{m-2})(x^*-x_{m-1})(y^*-y_{n-2})(y^*-y_{n-1})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m,n} - \\
& \frac{(x^*-x_{m-2})(x^*-x_{m-1})(y^*-y_{n-2})(y^*-y_n)}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m,n-1} + \\
& \frac{(x^*-x_{m-2})(x^*-x_{m-1})(y^*-y_n)(y^*-y_{n-1})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m,n-2} - \\
& \frac{(x^*-x_m)(x^*-x_{m-2})(y^*-y_{n-1})(y^*-y_{n-2})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-1,n} + \\
& \frac{(x^*-x_m)(x^*-x_{m-2})(y^*-y_{n-2})(y^*-y_n)}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-1,n-1} - \\
& \frac{(x^*-x_m)(x^*-x_{m-2})(y^*-y_n)(y^*-y_{n-1})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-1,n-2} + \\
& \frac{(x^*-x_m)(x^*-x_{m-1})(y^*-y_{n-2})(y^*-y_{n-1})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-2,n} - \\
& \frac{(x^*-x_m)(x^*-x_{m-1})(y^*-y_n)(y^*-y_{n-2})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-2,n-1} + \\
& \frac{(x^*-x_m)(x^*-x_{m-1})(y^*-y_n)(y^*-y_{n-1})}{(x_m-x_{m-1})(x_m-x_{m-2})(y_n-y_{n-1})(y_n-y_{n-2})} z_{m-2,n-2}
\end{aligned}
\tag{6.6.2}
$$

where the value $z_{m,n}$ are vectors containing the colour components of the pixels whose location is determined by $\{m, n\}$. These vectors in the code may be 4-vectors, where the fourth element in the vector is the transparency, α, of the value. The transparency of the image is the ability to see through the image. The resulting image with the use of transparency will have smoothed edges and true colour forms since the artificial colours will have high alpha values. The image may be transformed only considering the RGB values, however artificial colouring will become very apparent, especially in brighter areas.

The generalization of this procedure to re-scale a portion of an image is straightforward. As examples of the results of this procedure, see the figures 6.9 to 6.17. We start with the image appearing in 6.10, and zoom in on the area of the face. The results appears in figure 6.14 and 6.12, together with a comparison done using a simple linear interpolation scheme. Zooming further into the area of the eye results in 6.17 and 6.15, again with a comparison of the result of a simple linear interpolation. Generally, the number of iterations needed in the random iteration algorithm to produce acceptable images is of the order of $s_x M \times s_y N$, where the original image is of size M $\times$ N. Also, while slower, the quadratic fractal interpolating function typically produces, for the same number of iterations, a "smoother" looking image than the corresponding linear interpolating function. However, as with all interpolation schemes, there comes a point where such higher-order interpolating formulas actually start to produce worse results due to an artificially high sensitivity to fluctuations in the data. Evidence of this is seen in the fractal zoom around the eye, artifacts of the high sensitivity cause discolouration on the boundary between the iris and the sclera. Some informal tests of this procedure seem to indicate that better results are obtained for images of people, natural scenery, etc., as opposed to those containing lettering, simple geometric shapes, and similar constructs, as seen in figures 6.1, 6.2, 6.3, 6.4. The Dilbert pictures are courtesy of Scott Adams. The Dilbert cartoon is copyright Scott Adams.

Figure 6.1: Full size Dilbert reconstruction using linear fractal interpolation

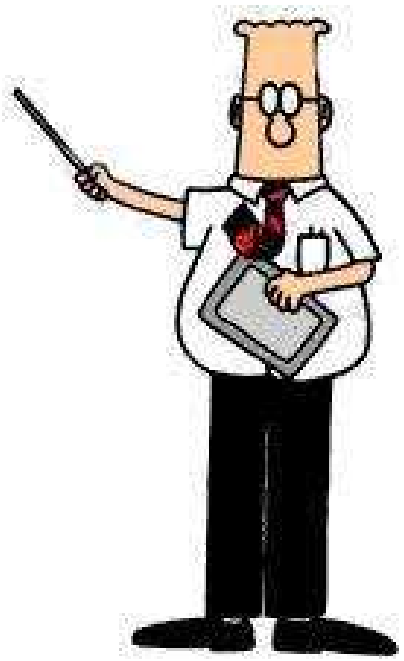

Figure 6.2: Full size Dilbert reconstruction using linear interpolation

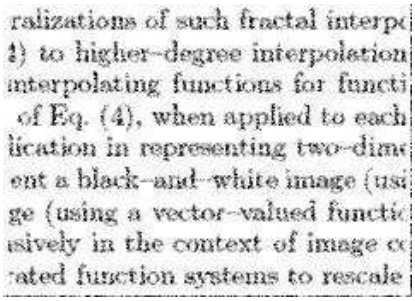

Figure 6.3: Full size text reconstruction using linear fractal interpolation

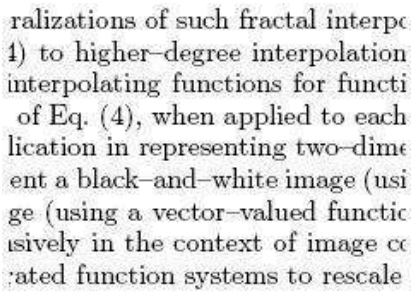

Figure 6.4: Full size text reconstruction using linear interpolation

Figure 6.5: 4x zoom Dilbert reconstruction using linear fractal interpolation

Figure 6.6: 4x zoom Dilbert reconstruction using linear interpolation

However during the zoom process the smoothing effects seen in the real image space picture, exists in the artificial image space produced by a cartoon or a block of text. This is seen in figures 6.5, 6.6, 6.7, 6.8.

The preceding method demonstrates that these non-linear fractal interpolating functions of two variables can be used in principle to represent images.

A potentially useful extension of these considerations is to the method of partitioned iterated function systems, which is the currently best method of image compression. This method breaks an image into several parts and attempts to create an IFS for each sub-image. This has a higher probability of producing a highly compact

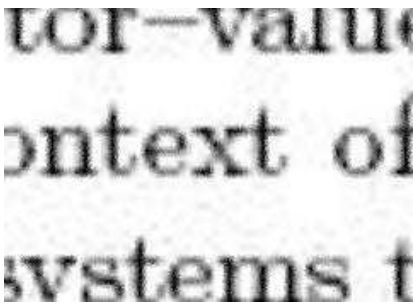

Figure 6.7: 4x zoom text reconstruction using linear fractal interpolation

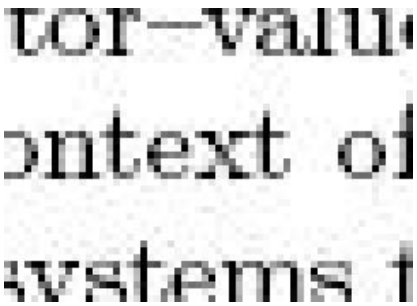

Figure 6.8: 4x zoom text reconstruction using linear interpolation

Figure 6.9: The quadratic fractal reconstruction of entire image

Figure 6.10: The linear reconstruction of entire image

Figure 6.11: The linear fractal reconstruction of entire image

Figure 6.12: The quadratic fractal magnification around the face

Figure 6.13: The linear magnification around the face

Figure 6.14: The linear fractal magnification around the face

Figure 6.15: The quadratic fractal magnification around the eye

Figure 6.16: The linear magnification around the eye

Figure 6.17: The linear fractal magnification around the eye

image, rather looking at the image as a whole to produce one IFS. [30]

6.7 Conclusion

Here we have presented the full usefulness of the fractal interpolation methods. There are several good industrial applications of these methods. Since current video capture methods usually result in fuzzy images upon magnification, better methods of image magnification are needed. This becomes rather critical when it comes time for police to identify suspects from a surveillance video.

Chapter 7

Conclusions

7.1 Symbolic Dynamics

We established a method of determining a maximal list of allowed orbits, and a prescription for calculating the phase space location of these orbits based solely on the symbolic dynamics. The methods discussed in Chapter 2 are generic to the level of 2 dimensional maps. Extending these ideas to 3 dimensional maps is an open area for research, but for now we can easily use Poincaré sections to reduce the mappings to 2 dimensions, allowing the use of the presented methods.

7.2 Numerical Periodic Orbit Methods

I have presented in Chapter 3 a complete method of both calculating the periodic orbits, as well as a method of calculating the dynamical averages in the system. The method described are broad in their scope as they can be applied to any system of differential equations, of any order. One still has to be cautious since random

processes will produce results, but their meaning will be null, and general trends will not likely be seen.

7.3 Numerical Periodic Orbits

I have shown in Chapter 4, methods of numerically finding the periodic orbits as they are found symbolically in Chapter 2. A method of automating this method was also presented. More work in this area is still necessary to make the programs utilize better search algorithms and thereby much more efficient. However, this extended beyond the scope of the project, and is best left for further study in this field. Although the code is quite complicated as a whole, each of the parts easily correspond to the theory described in Chapter 3. The resulting effect was a program that could take a moderately good guess, i.e. the methods used in Chapter 2, and form an accurate prediction of the allowed orbit in relatively short time. The orbits were used for averaging, and easily produced results for the system that came within a few decimal places of the brute force calculations. The benefit of this method is that long term prediction becomes possible since one only observes the known periodic orbits of the system, and calculates the chaotic propertied from there. This leaves the original chaotic system alone, which is where most of the difficulties in long term calculation lie.

7.4 Fractals

In this chapter we have presented the ideas and observations that lead to the concepts of fractal geometry. We have discussed the link between the concepts of chaos and fractals, and given an example of such a relationship through the system observed in chapters 2, 3 and 4. Specifically the most important concept from this chapter was the use of fractal geometry rather than the traditional Euclidean geometry in chaotic dynamics. The link between the fractal nature of real systems and chaotic analysis is fundamental to nature, and needs to be investigated in more physical systems.

7.5 Image Manipulation

Finally in Chapter 6 we presented the full usefulness of the fractal interpolation methods. There are several good industrial applications of these methods. Since current video capture methods usually result in fuzzy images upon magnification, better methods of image magnification are needed. This becomes rather critical when it comes time for police to identify suspects from a surveillance video.

7.6 Future Projects

The concepts studied here have all been for classical systems. A natural extension of both of these methods would be to quantum systems. The beauty of the IFS is that the methods may have applications in several different fields, it does not have to

be solely associated with image compression. They may also have more immediate physics applications through quantum mechanics. Currently IFS's are being applied to the topological study of quantum systems, in a paper by Artur Loziński, Karol Zyczkowski and Wojciech Slomczyński (2003). In this paper they describe quantum iterated functions systems or as they dub it QIFS. QIFS are described as functions acting randomly in Hilbert space, where each function acts according to a prescribed probability. These are designed to describe specific nonunitary quantum dynamics. Therein any research into this field may have effects on the study of their research.

Further investigation into QIFS resulted into the discovery that they are also being used to describe event enhanced quantum theory, EEQT. Quantum fractals also play an important role in this

Further applications of periodic orbit theory may be applied to quantum chaos, see Steiner [65] where he discusses the general trace formula as a sound mathematical basis for the semiclassical quantization of chaos. He presents two conjectures where it is argued that there are unique fluctuation properties in quantum mechanics which are universal. He states that these properties constitute the quantum mechanical analogue of the phenomenon of chaos in classical mechanics. His overall claim is that quantum chaos has been found.

Periodic orbit theory also has applications to relativistic mechanics, as shown by Carl Dettmann [12, 11]. Although, my speculation is that the higher dimensional

symbolic dynamics have to be developed long before periodic orbit theory will be of any use, since relativistic dynamics is usually described in hyperspace. Poincaré sections may not be able to reduce the system to 2 dimensions without significant loss of information about the system.

An immediate computer science application of iterated function systems is that shown in [66] since it extracts the contour of an image given an outline, thus singling out that part of the image space where our fractal zoom method may be used to enhance that particular part of the image. Of course we have to admit there is a long way to go for image compression using the interpolating methods.

As briefly mentioned in chapter 6 the method presented have applications in video image shots. What may be a logical next step would be to apply this magnification procedure to video. We have seen that the linear magnification for images is weak when compared to fractal methods, it only stands to reason that the same would be true for motion pictures.

7.7 Final Remarks

When all is said and done, it appears that chaos may be rigorously studied, as our system is a random choice for a test of the validity of symbolic dynamics, and periodic orbit theory. Our system has been able to match to a small level of accuracy the well tested brute force methods of determining the Lyapunov exponent of a system, the winding number, as well as the topological and entropy dimensions of the system.

To obtain a higher level of accuracy one only needs to continue the orbit searching methods to higher periods, and possibly spend more time trying to find all of the orbits within a given period. Creating the programs as suggested in this thesis will lead to faster methods of determining allowed orbits, and ultimately more accurate results.

As for the interpolated IFS methods, we have shown that the methods developed in [59] could easily be extended to higher dimensions as well as higher order extrapolations. We have successfully shown that the outcome of these methods does allow one a secure method of resizing the image with minimal information loss. With further optimization it is felt that this method may also be useful for image compression.

AMDG

Appendix A

Periodic Orbit Hunting Code

```
/*
   Multishooting program v0.3.2.
   A collaborative effort between S. Peles, Georgia Institute
   of Technology, School of Physics, A.J. Penner, University
   of Manitoba, and R. Kobes, University of Winnipeg.

   This program takes guesses for periodic orbits from a file that
   contains the points of a Poincare section of a given set of
   ODE's, processes them and attempts to generate periodic orbits,
   if the guesses are close enough to an expected orbit.

   Approximate running time on an intel Pentium IV 2.4 GHz
   processor is 8 hours, with an input file on the order of 7
   million Poincare section points.
*/

#include <stdio.h>
#include <math.h>
#include "gsl/gsl_errno.h"
#include "gsl/gsl_matrix.h"
#include "gsl/gsl_odeiv.h"
#include"gsl/gsl_linalg.h"
#include <gsl/gsl_blas.h>
#include <gsl/gsl_multiroots.h>

#define ODE_DIM 3
```

```
struct ODE_params {
  double Q, drive_freq, r;
  int period;
};

// this function sets up the ODE's appropriate to our system
int func (double t, const double y[], double f[],
      void *params)
{
  struct ODE_params *p
    = (struct ODE_params *) params;

  double Q = p->Q;
  double a = p->drive_freq;
  double r = p->r;

  f[0] = y[1];
  f[1] =  - y[1]/Q +
    ( -sin(2.*M_PI*y[0]) - r*a/Q + (1.0+r)/r*sin(2.*M_PI*y[0] - y[2]))/2./M_PI;
  f[2] = a;
  return GSL_SUCCESS;
}

// this function allows for the multiplication of the individual
// Jacobian functions
// this is necessary for the calculation of the appropriate
// eigenvalues for each orbit

int linfunc (double t, double J[], double dJdt[], double A[]) {

  // Set the matrix of variations of the flow
  gsl_matrix_view A_mat
    = gsl_matrix_view_array (A, ODE_DIM, ODE_DIM);

  // Set NxN linearized equations
  gsl_matrix_view J_mat
    = gsl_matrix_view_array (J, ODE_DIM, ODE_DIM);

  // Initialize derivatives of linearized equations
  gsl_matrix_view dJdt_mat
```

```
    = gsl_matrix_view_array (dJdt, ODE_DIM, ODE_DIM);


  // Calculate derivatives of linearized equations: dJ/dt = A*J
  gsl_blas_dgemm (CblasNoTrans, CblasNoTrans,
                  1.0, &A_mat.matrix, &J_mat.matrix,
                  0.0, &dJdt_mat.matrix);

  return GSL_SUCCESS;
}


// This function sets up the Jacobian matrix of our system
int jac (double t, const double y[], double *dfdy,
     double dfdt[], void *params)
{
  struct ODE_params *p
    = (struct ODE_params *) params;

  double Q = p->Q;
  double a = p->drive_freq;
  double r = p->r;

  gsl_matrix_view dfdy_mat
    = gsl_matrix_view_array (dfdy, ODE_DIM, ODE_DIM);
  gsl_matrix * m = &dfdy_mat.matrix;
  gsl_matrix_set (m, 0, 0, 0.0);
  gsl_matrix_set (m, 0, 1, 1.0);
  gsl_matrix_set (m, 0, 2, 0.0);
  gsl_matrix_set (m, 1, 0, -cos(2.*M_PI*y[0]) + (1.0+r)/r*cos(2.*M_PI*y[0] - y[1
  gsl_matrix_set (m, 1, 1, -1.0/Q);
  gsl_matrix_set (m, 1, 2, -(1.0+r)/r*cos(2.*M_PI*y[0] - y[2])/2./M_PI);
  gsl_matrix_set (m, 2, 0, 0.0);
  gsl_matrix_set (m, 2, 1, 0.0);
  gsl_matrix_set (m, 2, 2, 0.0);
  dfdt[0] = 0.0;
  dfdt[1] = 0.0;
  dfdt[2] = 0.0;
  return GSL_SUCCESS;
}
```

```
int func1 (double t, const double x[], double f[],
      void *params)
{
  int i;
  double y[ODE_DIM], dy[ODE_DIM], J[ODE_DIM*ODE_DIM], dJdt[ODE_DIM*ODE_DIM];
  double dfdy[ODE_DIM*ODE_DIM], dfdt[ODE_DIM];

  // Set equations of the flow
  for(i = 0; i < ODE_DIM; i++)
    y[i] = x[i];

  // Evaluate derivatives of the flow
  func(t, y, dy, params);
  for(i = 0; i < ODE_DIM; i++)
    f[i] = dy[i];

  // Calculate the matrix of variations of the flow and store it in dfdy
  jac(t, y, dfdy, dfdt, params);

  // Set linearized equations
  for(i = 0; i < ODE_DIM*ODE_DIM; i++)
    J[i] = x[ODE_DIM + i];

  // Evaluate derivatives of the linearized equations
  linfunc(t, J, dJdt, dfdy);
  for(i = 0; i < ODE_DIM*ODE_DIM; i++)
    f[ODE_DIM + i] = dJdt[i];

  return GSL_SUCCESS;
}

int jac1 (double t, const double y[], double *dfdy,
     double dfdt[], void *params)
{ /*   Still to implement. Right now you cannot use  */ /*   bsimp
integrator. */
  return GSL_SUCCESS;
}
```

```
// this function calculates the Poincare iterate at each step of
// the ODE solver

int Poincare_map_strob_wjac (double *y, double *Jacobi_Matrix,
void *params) {
  const int STEPS_PER_SECTION = 1;
  int i,j;
  const gsl_odeiv_step_type * T
    = gsl_odeiv_step_rk8pd;

  gsl_odeiv_step * s
    = gsl_odeiv_step_alloc (T, ODE_DIM*(1+ODE_DIM));
  gsl_odeiv_control * c
    = gsl_odeiv_control_y_new (1e-12, 0.0);
  gsl_odeiv_evolve * e
    = gsl_odeiv_evolve_alloc (ODE_DIM*(1+ODE_DIM));

  struct ODE_params *p
    = (struct ODE_params *) params;
  double STROBE_PERIOD = 2.0*M_PI / (p->drive_freq);


  double tin=0.0, deltat=STROBE_PERIOD/STEPS_PER_SECTION;
  double t, t1;
  double h = 1e-6;
  double x[ODE_DIM*(1+ODE_DIM)];

  gsl_odeiv_system sys = {func1, jac1, ODE_DIM*(1+ODE_DIM), params};
  gsl_ieee_env_setup();


  // Set initial conditions of the flow + linearized equations
  for(i = 0; i < ODE_DIM; i++)
    x[i] = y[i];

  for(i = 0; i < ODE_DIM; i++)
    for(j = 0; j < ODE_DIM; j++)
      {
    if(i == j)
      x[ODE_DIM + ODE_DIM* i + j] = 1.0;
```

```
    else
      x[ODE_DIM + ODE_DIM* i + j] = 0.0;
      }


  t = tin;

  do
    {
      t1 = t + deltat;
      if ( t1 > STROBE_PERIOD ) t1 = STROBE_PERIOD;

      while (t < t1)
    {
      int status = gsl_odeiv_evolve_apply (e, c, s,
                           &sys,
                           &t, t1,
                           &h, x);
      if (status != GSL_SUCCESS)
        break;
    }

      t = t1;
    }
  while ( t < STROBE_PERIOD );

  for(i = 0; i < ODE_DIM; i++)
    y[i] = x[i];

  for(i = 0; i < ODE_DIM*ODE_DIM; i++)
    Jacobi_Matrix[i] = x[ODE_DIM + i];

  gsl_odeiv_evolve_free(e);
  gsl_odeiv_control_free(c);
  gsl_odeiv_step_free(s);
  return 0;
}

// this function prints to file
void print_orbits(const gsl_vector *F, const gsl_vector *x,
```

```
          void *param, FILE * out, const int count)
{
  int i,j;
  double error=0.0;
  struct ODE_params *p
    = (struct ODE_params *) param;
  int PERIOD_OF_ORBIT = p->period;

      /* Print vector F */
      for( i=0; i < PERIOD_OF_ORBIT; i++)
    {
      for( j=0; j < ODE_DIM; j++)
        {
          printf(" %18.14f",gsl_vector_get( F, ODE_DIM* i + j));
        }
      printf("\n");
    }

      for( i=0; i < ODE_DIM*PERIOD_OF_ORBIT; i++ )
    error += fabs(gsl_vector_get(F,i));
if(error < 1E-13){
      fprintf(out,"%d %g\n",count,error);

      /* Print orbits */
      for( i=0; i < PERIOD_OF_ORBIT; i++)
    {
      for( j=0; j < ODE_DIM; j++)
        {
          fprintf(out," %18.14f", gsl_vector_get(x,ODE_DIM* i + j));
        }
      fprintf(out,"\n");
    }
fprintf(out,"\n"); } // ends new if control

printf("----------------------------------------\n");

  return;
}

// this function crates the matrix DF as described in chapter 3
```

```
void ConstructDF(gsl_matrix *DF, int d, int n, int k,
         double J[], double v[], double a[])
{
  int i,j;

      // Add Jacobi matrix in the upper right block.
      if (k == n-1)
    {
      for( i=0; i<d; i++)
        for( j=0; j<d; j++)
          gsl_matrix_set( DF, i, (n-1)*d + j, -J[i*d +j] );
    }
      // Add other Jacobi matrices. Index k denotes point at the orbit.
      else
    {
      for( i=0; i<d; i++)
        for( j=0; j<d; j++)
          gsl_matrix_set( DF, (k+1)*d + i, k*d + j, -J[i*d +j]);
    }

      // Add flow vector. Index k denotes point at the orbit.
      for( i=0; i<d; i++)
    {
      gsl_matrix_set( DF, k*d + i, n*d + k, -v[i]);
    }

      // Add Poincare (hyper)surface vector.
      for( i=0; i<d; i++)
    {
      gsl_matrix_set( DF, n*d + k, k*d + i, a[i]);
    }

  return;

}

// this function produces the vector F as described in chapter 3
void ConstructF(gsl_vector *F, int d, int n, int k, double
value[]) {
  int i;
```

```
  for( i=0; i<d; i++)
    {
      gsl_vector_set( F, d * k + i, -value[i]);
    }

}

void SPmod(double *x) {
  if ( fabs(x[2]) > M_PI ) x[2] -= 2.0*M_PI*x[2]/fabs(x[2]);
  while ( fabs(x[0]) > 0.5 ) x[0] -= x[0]/fabs(x[0]);
  //  if ( x[2] >= 2.0*M_PI ) x[2] -= 2.0*M_PI;
  return;
}

// this function completes the previous DF function, it is
// separate for debugging purposes only

int CompleteDF(const gsl_vector *Guess,
           void *param, gsl_matrix *DF )
{
  int i,k;
  struct ODE_params *p
    = (struct ODE_params *) param;
  int PERIOD_OF_ORBIT = p->period;
  double J[ODE_DIM*ODE_DIM], y[ODE_DIM], v[ODE_DIM];
  // Poincare (hyper)plane vector
  double a[ODE_DIM] = { 0.0, 0.0, 1.0 };

  // Initialize matrix DF
  gsl_matrix_set_zero (DF);
  for(i=0; i < ODE_DIM*PERIOD_OF_ORBIT; i++)
    gsl_matrix_set( DF, i, i, 1.0); // makes the top left the identity matrix

  for ( k=0; k < PERIOD_OF_ORBIT; k++)
    {
      // Take initial conditions for a point on the orbit
      for( i=0; i < ODE_DIM; i++ )
    {
      y[i] = gsl_vector_get( Guess, ODE_DIM * k + i);
```

```
    }

      // Dynamical flow.
      func( 0.0, y, v, param);

      // Poincare map step.
      Poincare_map_strob_wjac( y, J, param );

      ConstructDF(DF, ODE_DIM, PERIOD_OF_ORBIT, k,J,v,a);
    }
  return GSL_SUCCESS;
}

// this function completes the F vector, again separate for
// debugging purposes

int CompleteF(const gsl_vector *Guess, void *param, gsl_vector *F)
{
  int i,k;
  struct ODE_params *p
    = (struct ODE_params *) param;
  int PERIOD_OF_ORBIT = p->period;
  double J[ODE_DIM*ODE_DIM], y[ODE_DIM];

  // Initialize vector F
  gsl_vector_set_zero(F);

  for ( k=0; k < PERIOD_OF_ORBIT; k++)
    {
      // Take initial conditions for a point on the orbit
      for( i=0; i < ODE_DIM; i++ )
    {
      y[i] = gsl_vector_get( Guess, ODE_DIM * k + i);
    }

      // Poincare map step.
      Poincare_map_strob_wjac( y, J, param );

      // Calculate -( x - f(x))
      if(k == PERIOD_OF_ORBIT - 1)
```

```
    {
      for( i=0; i < ODE_DIM; i++ )
        y[i] -= gsl_vector_get(Guess, i);
      SPmod(y);
      ConstructF(F, ODE_DIM, PERIOD_OF_ORBIT, 0, y);
    }
      else
    {
      for( i=0; i < ODE_DIM; i++ )
        y[i] -= gsl_vector_get(Guess, ODE_DIM * (k+1) + i);
      SPmod(y);
      ConstructF(F, ODE_DIM, PERIOD_OF_ORBIT, k+1, y);
    }

    }

  //  print_orbits(F, Guess, param);
  return GSL_SUCCESS;
}

int CompleteFDF(const gsl_vector *Guess, void *param,
        gsl_vector *F, gsl_matrix *DF)
{
  CompleteDF( Guess, param, DF);
  CompleteF( Guess, param, F);
  return GSL_SUCCESS;
}

int main(void) {
  const gsl_multiroot_fdfsolver_type *T;
  gsl_multiroot_fdfsolver *s;
  FILE *in, *out;

  int status;
  size_t iter = 0;

  const size_t n = (ODE_DIM+1)*2; // multiplied by the period of orbit!

  int i, j, k;
  struct ODE_params rod_param={ 0.76,    // Quality factor "Q"
```

```
                1.028,   // Drive frequency "a"
                1.088,     // Radii ratio "r"
                15 };      // PERIOD_OF_ORBIT

  in = fopen("Period2II.txt","rb");
  out = fopen("p2valuesII.dat","wb");

  int PERIOD_OF_ORBIT=rod_param.period;
  int count=0;

  gsl_multiroot_function_fdf f = {&CompleteF,
                                  &CompleteDF,
                                  &CompleteFDF,
                                  n, &rod_param};

  gsl_vector * Guess = gsl_vector_alloc ( PERIOD_OF_ORBIT * (ODE_DIM+1) );
  double InitialGuess[ODE_DIM*15];

// the following few lines reads the initial guesses from the
// Poincare section file
while(fscanf(in,"%lf%lf%lf
%lf%lf%lf",
&InitialGuess[0],&InitialGuess[1],&InitialGuess[2],
&InitialGuess[3],&InitialGuess[4],&InitialGuess[5] )!=EOF){

// these lines keep the orbit points in the correct normalization
for(i=0;i<PERIOD_OF_ORBIT;i++){
while(InitialGuess[i*ODE_DIM]<0.0){InitialGuess[i*ODE_DIM]+=1.0;}
while(InitialGuess[i*ODE_DIM]>1.0){InitialGuess[i*ODE_DIM]-=1.0;}
}

for(i=0;i<PERIOD_OF_ORBIT;i++){
while(InitialGuess[i*ODE_DIM+1]>0.0){InitialGuess[i*ODE_DIM+1]-=1.0;}
}

iter =0; // this allows the multishoot method to look at multiple
         // guesses

  gsl_vector_set_zero(Guess);
  for( i=0; i < PERIOD_OF_ORBIT*ODE_DIM; i++)
```

```
    gsl_vector_set(Guess,i,InitialGuess[i]);

// Setup GSL variables for the multiple root finding algorithms

  T = gsl_multiroot_fdfsolver_gnewton;
  s = gsl_multiroot_fdfsolver_alloc (T, n);
  gsl_multiroot_fdfsolver_set (s, &f, Guess);

  do
    {
      iter++;
        // iterates the multiroot solver one step
      status = gsl_multiroot_fdfsolver_iterate (s);

      if (status) // if the root solver method does not converge,
                  // or diverges, then it will return a failure,
                  // and this process must end.
        break;

      status = gsl_multiroot_test_delta (s->f, s->x, 1e-14, 0.0);
      // check the error calculation to within a specific range,
      // if the error is larger, the status is set to continue.
    }
  while(iter < 15 && status == GSL_CONTINUE);

// send the information to a print function for proper output
      print_orbits(s->f, s->x, &rod_param,out,count);
count++; } // ends while

fclose(in);
fclose(out);

  gsl_vector_free( Guess);
  gsl_multiroot_fdfsolver_free (s);

  return 0;
}
```

Appendix B

Iterated Function System Code

All of the following code was written as a collaboration between R. Kobes, University of Winnipeg, and A.J. Penner, University of Manitoba.

B.1 Draw.c

```
 #include "fractalResized.h"

int main (void) {
  gdImagePtr srcImg, dstImg, cmpImg; // create necessary image
                                     // pointers
  FILE *in, *out, *cmp;
  float min_factor = 0.999999, max_factor = 7;
  int width = 35, height = 57, neww, newh; // portion of picture
                                           // to be scaled
  float scale = 4;
  neww = (int) (scale * width);  // output image size
  newh = (int) (scale * height);

  in = fopen("lena.jpeg", "rb");
  if (!in) nrerror("Input file does not exist");
  srcImg = gdImageCreateFromJpeg(in);          // source input

  dstImg = gdImageCreateTrueColor(neww, newh);  // destination
                                                // output
```

```
  cmpImg = gdImageCreateTrueColor(neww, newh);  // comparison
                                                // image


  // this calls the resizing function
  gdIFSCopyResized(dstImg, srcImg, 0, 0, 80, 40, //110,120
                neww, newh, width, height, min_factor, max_factor);

  // this calls the image resizing function
  gdImageCopyResized(cmpImg, srcImg, 0, 0, 80, 40, //110,120
                neww, newh, width, height);

  out = fopen("lena1.jpeg", "wb");
  if (!out) nrerror("Output file cannot be created");

  gdImageJpeg(dstImg, out, -1); // finalize the destination image

  cmp = fopen("lena2.jpeg", "wb");
  if (!cmp) nrerror("Output file cannot be created");

  gdImageJpeg(cmpImg, cmp, -1); // finalize the comparison image

  fclose(in);
  fclose(out);
  fclose(cmp);
  return 0;
}
```

B.2 fractalResized.h

```
#include <gd.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  float x, y;
  int rgb[3];
} ifs;

void fractalResized (gdImagePtr dstImg, gdImagePtr srcImg,
```

```
                    int dstX, int dstY, int srcX, int srcY,
                    int dstW, int dstH, int srcW, int srcH,
            float min_factor, float max_factor);

void generate_ifs (gdImagePtr srcImg, ifs **z,
                   int srcX, int srcY, int dstX, int dstY,
                   int srcW, int srcH, int dstW, int dstH);

void generate_ifs_image (gdImagePtr dstImg, ifs **z, int **seen,
                         int srcX, int srcY, int dstX, int dstY,
                         int srcW, int srcH, int dstW, int dstH,
            int dstXend, int dstYend,
            float min_factor, float max_factor);

void fill_in_blanks (gdImagePtr dstImg, int **seen,
                     int dstX, int dstY, int dstXend, int dstYend);

void rgb_linear (float newx, float newy, ifs **z,
                 int m, int n, float x, float y, float xm1, float ym1,
                 int *rgb);

int nearest (int **seen, int i, int j,
              int xstart, int ystart, int width, int height);

ifs **ifsmatrix(long nrh, long nch);

int **imatrix(long nrl, long nrh, long ncl, long nch);

void free_ifsmatrix(ifs **m, long nrh, long nch);

void free_imatrix(int **m, long nrl, long nrh, long ncl, long
nch);

void nrerror(char error_text[]);
```

B.3 gdIFSCopyResized.c

```
#include "gdIFSCopyResized.h"

/* ************************************************************
```

```
copy an image from srcImg to dstImg via an IFS algorithm. The
source starts at (srcX, srcY), of size (srcW, srcH), and is copied
to the destination, starting at (dstX, dstY), of size (dstW,
dstH).

min_factor, between 0 and 1, determines the minimum fraction of
the destination points to be colored by the IFS algorthm. The
remainder simply use the nearest available pixel to determine the
colour. Values very close to 1 will produce better looking images,
but will take longer.

max_factor, greater than 1, determines the maximum number of
iterations that the IFS algorithm uses. A value of 1 will have
this iteration number equal to the number of pixels in the
destination; increasing this value will produce better looking
images, but at the expense of speed. Reasonable values are around
5. ********************************************************** */

static int gdImageGetTrueColorPixel (gdImagePtr im, int x, int y)
{
  int p = gdImageGetPixel (im, x, y);
  if (!im->trueColor)
    {
      return gdTrueColorAlpha (im->red[p], im->green[p], im->blue[p],
                   (im->transparent == p) ? gdAlphaTransparent :
                   gdAlphaOpaque);
    }
  else
    {
      return p;
    }
}

void gdIFSCopyResized (gdImagePtr dstImg, gdImagePtr srcImg,
                       int dstX, int dstY, int srcX, int srcY,
                       int dstW, int dstH, int srcW, int srcH,
                       double min_factor, double max_factor) {

  ifs **z;
  int **seen, i, j, dstXend, dstYend;
```

```
  if (min_factor < 0 || min_factor > 1) min_factor = 0.999;
  if (max_factor < 1) max_factor = 4;

  dstXend = dstX + dstW;
  dstYend = dstY + dstH;

  /* allocate some arrays */
  z = ifsmatrix(srcX, srcX + srcW, srcY, srcY + srcH);
  seen = imatrix(dstX, dstX + dstW, dstY, dstY + dstH);

  /* initialize some arrays */
  for (i=dstX; i<=dstXend; i++) {
    for (j=dstY; j<=dstYend; j++) {
      seen[i][j] = 0;
    }
  }

  generate_ifs(srcImg, z, srcX, srcY, dstX, dstY,
               srcW, srcH, dstW, dstH);

  generate_ifs_image(dstImg, z, seen, srcX, srcY, dstX, dstY,
                     srcW, srcH, dstW, dstH, dstXend, dstYend,
            min_factor, max_factor);

  fill_in_blanks(dstImg, seen, dstX, dstY, dstXend, dstYend);

  free_ifsmatrix(z, srcX, srcX+srcW, srcY, srcY+srcH);
  free_imatrix(seen, dstX, dstX+dstW, dstY, dstY+dstH);

}

void generate_ifs (gdImagePtr srcImg, ifs **z,
                   int srcX, int srcY, int dstX, int dstY,
                   int srcW, int srcH, int dstW, int dstH) {
  int srcXend, srcYend, i, j, index, srcIsTrue;
  double scaleX, scaleY;

  srcXend = srcX + srcW;
  srcYend = srcY + srcH;
```

```
  scaleX = (double) (dstW) / (double) (srcW);
  scaleY = (double) (dstH) / (double) (srcH);

  srcIsTrue = srcImg->trueColor;

  for (i=srcX; i<=srcXend; i++) {
    for (j=srcY; j<=srcYend; j++) {
      z[i][j].x = dstX + scaleX * (i-srcX);
      z[i][j].y = dstY + scaleY * (j-srcY);
      if (srcIsTrue) {
        index = gdImageGetTrueColorPixel(srcImg, i, j);
        z[i][j].rgb[0] = gdTrueColorGetRed(index);
        z[i][j].rgb[1] = gdTrueColorGetGreen(index);
        z[i][j].rgb[2] = gdTrueColorGetBlue(index);
        z[i][j].rgb[3] = gdTrueColorGetAlpha(index);
      }
      else {
        index = gdImageGetPixel(srcImg, i, j);
        z[i][j].rgb[0] = gdImageRed(srcImg, index);
        z[i][j].rgb[1] = gdImageGreen(srcImg, index);
        z[i][j].rgb[2] = gdImageBlue(srcImg, index);
        z[i][j].rgb[3] = gdImageAlpha(srcImg, index);
      }
    }
  }
}

// this function performs the IFS zoom on the original image
void generate_ifs_image (gdImagePtr dstImg, ifs **z, int **seen,
                         int srcX, int srcY, int dstX, int dstY,
                         int srcW, int srcH, int dstW, int dstH,
             int dstXend, int dstYend,
             double min_factor, double max_factor) {

  int count = 0, hits = 0, max, min, m, n, im, in, rgb[4], index,
    transparent, dstIsTrue;
  double oldx = 3.3, oldy = 4.7, newx, newy, x, y, xm1, ym1;

  transparent = gdImageGetTransparent(dstImg);
  dstIsTrue = dstImg->trueColor;
```

```
  max = (int) (max_factor * dstW * dstH);
  min = (int) (min_factor * dstW * dstH);

  for (count=0; count<max; count++) {
    m = 1 + srcX + (int) ((double) srcW * rand() / (RAND_MAX + 1.0) );
    n = 1 + srcY + (int) ((double) srcH * rand() / (RAND_MAX + 1.0) );
    x = z[m][n].x;
    y = z[m][n].y;
    xm1 = z[m-1][n].x;
    ym1 = z[m][n-1].y;
    newx = ( (oldx-dstX)*x + (dstXend-oldx)*xm1 ) / dstW;
    newy = ( (oldy-dstY)*y + (dstYend-oldy)*ym1 ) / dstH;
    im = (int) newx;
    in = (int) newy;
    if (seen[im][in] > 0) continue;
    rgb_linear(newx, newy, z, m, n, x, y, xm1, ym1, rgb);
    index = gdImageColorResolveAlpha(dstImg, rgb[0], rgb[1], rgb[2], rgb[3]);
    seen[im][in] = index;
    if (index == transparent || index <= 0) continue;
    if (hits++ > min) break;
    gdImageSetPixel(dstImg, im, in, index);
    oldy = newy;
    oldx = newx;
  }
}

// this function fills the blank spaces from the IFS zoom with a
// colour determined by neighbouring pixels
// this is necessary since the IFS is based on a random calling
// sequence, and not all pixels will be covered by this method in a
// reasonable amount of time.

void fill_in_blanks (gdImagePtr dstImg, int **seen,
                     int dstX, int dstY, int dstXend, int dstYend) {

  int transparent, i, j, index;

  transparent = gdImageGetTransparent(dstImg);
```

```
  for (i=dstX; i<=dstXend; i++) {
    for (j=dstY; j<=dstYend; j++) {
      if (seen[i][j] > 0) continue;
      index = nearest(seen, i, j, dstX, dstY, dstXend, dstYend);
      if (index <= 0 || index == transparent) continue;
      gdImageSetPixel(dstImg, i, j, index);
    }
  }
}

// this function performs a linear zoom on the original image
// using a classic zoom technique

void rgb_linear (double newx, double newy, ifs **z,
                 int m, int n, double x, double y, double xm1, double ym1,
                 int *rgb) {

  double denX, denY;
  int i;

  denX = x - xm1;
  denY = y - ym1;

  for (i=0; i<=3; i++) {
    rgb[i]=
      (int) ( ( (newx-xm1)*(newy-ym1)*z[m][n].rgb[i] -
                (newx-x)*(newy-ym1)*z[m-1][n].rgb[i] -
                (newx-xm1)*(newy-y)*z[m][n-1].rgb[i] +
                (newx-x)*(newy-y)*z[m-1][n-1].rgb[i]
                ) / denX / denY);
    if (rgb[i] < 0) rgb[i] = 0 ;
    if (rgb[i] > 255) rgb[i] = 255 ;
  }
}

int nearest (int **seen, int i, int j,
             int dstX, int dstY, int dstXend, int dstYend) {
  int m, n;

  for (m=i-1; m<=i+1; m++) {
```

```
      if (m<dstX || m>dstXend) continue;
      for (n=j-1; n<=j+1; n++) {
      if (n<dstY || n>dstYend || seen[m][n] == 0) continue;
      return seen[m][n];
      }
  }
  return -1;
}

/* allocate an ifs matrix with subscript range
m[nrl..nrh][ncl..nch] */ ifs **ifsmatrix(long nrl, long nrh, long
ncl, long nch) {
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    ifs **m;

    /* allocate pointers to rows */
    m=(ifs **) malloc((size_t)((nrow+1)*sizeof(ifs*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += 1;
    m -= nrl;


    /* allocate rows and set pointers to them */
    m[nrl]=(ifs *) malloc((size_t)((nrow*ncol+1)*sizeof(ifs)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += 1;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

/* allocate a int matrix with subscript range
m[nrl..nrh][ncl..nch] */ int **imatrix(long nrl, long nrh, long
ncl, long nch) {
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;
```

```
    /* allocate pointers to rows */
    m=(int **) malloc((size_t)((nrow+1)*sizeof(int*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += 1;
    m -= nrl;


    /* allocate rows and set pointers to them */
    m[nrl]=(int *) malloc((size_t)((nrow*ncol+1)*sizeof(int)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += 1;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

/* free a ifs matrix allocated by ifsmatrix() */ void
free_ifsmatrix(ifs **m, long nrl, long nrh, long ncl, long nch) {
    free((char*) (m[nrl]+ncl-1));
    free((char*) (m+nrl-1));
}

/* free an int matrix allocated by imatrix() */ void
free_imatrix(int **m, long nrl, long nrh, long ncl, long nch) {
    free((char*) (m[nrl]+ncl-1));
    free((char*) (m+nrl-1));
}

/* standard error handler */ void nrerror(char error_text[]) {
  fprintf(stderr,"Run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}
```

B.4 gdIFSCopyResized.h

```
#include <gd.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  double x, y;
  int rgb[4];
} ifs;

void gdIFSCopyResized (gdImagePtr dstImg, gdImagePtr srcImg,
                       int dstX, int dstY, int srcX, int srcY,
                       int dstW, int dstH, int srcW, int srcH,
              double min_factor, double max_factor);

void generate_ifs (gdImagePtr srcImg, ifs **z,
                   int srcX, int srcY, int dstX, int dstY,
                   int srcW, int srcH, int dstW, int dstH);

void generate_ifs_image (gdImagePtr dstImg, ifs **z, int **seen,
                         int srcX, int srcY, int dstX, int dstY,
                         int srcW, int srcH, int dstW, int dstH,
              int dstXend, int dstYend,
              double min_factor, double max_factor);

void fill_in_blanks (gdImagePtr dstImg, int **seen,
                     int dstX, int dstY, int dstXend, int dstYend);

void rgb_linear (double newx, double newy, ifs **z,
                 int m, int n, double x, double y, double xm1, double ym1,
                 int *rgb);

int nearest (int **seen, int i, int j,
              int xstart, int ystart, int width, int height);

int **imatrix(long nrl, long nrh, long ncl, long nch);

ifs **ifsmatrix(long nrl, long nrh, long ncl, long nch);
```

```
void free_imatrix(int **m, long nrl, long nrh, long ncl, long
nch);

void free_ifsmatrix(ifs **m, long nrl, long nrh, long ncl, long
nch);

void nrerror(char error_text[]);
```

Bibliography

[1] W. Slomczynski A. Lozinski, K. Zyczkowski, *Quantum iterated function systems*, PREprint (2003).

[2] H.L. Swinney J.A. Vastano A. Wolf, J.B. Swift, *Determining lyapunov exponents from a time series*, Physica D **16** (1984), 285–317.

[3] S.N. Fedotkin K. Matsuyanagi A.G. Magner, K. Arita, *Symmetry breaking and bifurcations in the periodic orbit theory. ii Spheroidal Cavity*, Progress in Theoretical Physics **108** (2002), no. 5.

[4] W. Zheng B. Hao, J.X. Liu, *Symbolic dynamic analysis of the lorenz equations*, Physical Review E **57** (1998), no. 5, 5378–5396.

[5] Micheal Barnsley, *Fractal image compression*, PREprint (2003).

[6] Micheal F. Barnsley, *Fractals everywhere*, Morgan Kaufmann, New York, 1993.

[7] David Betounes, *Differential equations, theory and applications with maple*, Springer-Verlag, New York, 2001.

[8] J.A. York C. Grebogi, E. Ott, *Unstable periodic orbits and the dimensions of multifractal chaotic attractors*, Physical Review A **37** (1988), no. 5, 1711–1725.

[9] W.C. Siu C.M. Lai, K.M. Lam, *A fast fractal image coding based on kick-out and zero contrast conditions*, IEEE transactions on Image Processing **12** (2003), no. 11.

[10] N.E. Frankel C.P. Dettman, *Chaos and fractals around black holes*, Fractals **3** (1995), no. 1, 161–181.

[11] N.E. Frankel C.P. Dettmann, *Fractal basins and chaotic trajectories in multi-black-hole spacetimes*, Physical Review D **50** (1994), no. 2.

[12] ———, *Stochastic dynamics of relativistic turbulence*, Physical Review E **53** (1996), no. 5.

[13] S.D. Prado C.P. Dettmann, J.P. Keating, *Stochastic stabilization of cosmological photons*, PREpaper.

[14] Predrag Cvitanović, *Periodic orbits as the skeleton of classical and quantum chaos*, Physica D (1991), 138–151.

[15] ———, *Periodic orbit theory in classical and quantum mechanics*, Chaos, An Interdisciplinary Journal of Nonlinear Science (1996).

[16] B. Marcus D. Lind, *An introduction to symbolic dynamics*, Cambridge University Press, Maryland, 1995.

[17] H. Yan D.C. Popescu, A. Dimca, *A nonlinear model for fractal image coding*, IEEE transactions on Image Processing **6** (1997), no. 3.

[18] R.L. Devaney, *Introduction to chaotic dynamical systems*, Addison-Wesley, New York, 1989.

[19] É. Tosan É. Guérin, *Fractal inverse problem: an analytical approach*, January 2004.

[20] Micheal Frame, *Fractal geometry*, Lecture Notes, Yale University, Department of Mathematics,http://classes.yale.edu/fractals/, 2004.

[21] Jr. F.S. Hill, *Computer graphics using open gl*, second ed., Prentice Hall, New Jersey, 1990.

[22] J.P. Gollub G.L. Baker, *Chaotic dynamics*, Cambridge, New York, 1996.

[23] D. Saupe H. Hartenstein, M. Ruhl, *Region-based fractal image compression*, IEEE transactions on Image Processing **9** (2000), no. 7.

[24] D. Saupe H. Peitgen, H. Jurgens, *Chaos and fractals new frontiers of science*, first ed., Springer-Verlag, New York, 1992.

[25] Jacques Hadamard, *Les surfaces a courbures opposees st leurs lignes geodesiques*, Journal De Mathematique Pures at Appliques (1898), 27–73.

[26] K. Hanson, *Symbolic dynamics*, Ph.D. thesis, University of Somewhere, 1999.

[27] J.C. Hart, *Computer display of linear fractal surfaces*, Ph.D. thesis, University of Illinois At Chicago, 1991.

[28] G.A. Hedlund H.M. Morse, *Symbolic dynamics*, I. American Journal of Mathematics **60** (1938), no. 4, 815–866.

[29] B. Hao J. Liu, W. Zheng, *From annular to interval dynamics: Symbolic analysis of the periodically forced brusselator*, Chaos, Solitons & Fractals **7**, no. 9.

[30] A. Jacquin, *Image coding based on a fractal theory of iterated contractive image transformations*, IEEE transactions on Image Processing **1** (1992), no. 1.

[31] A. Jadczyk, *Simultaneous measurement of non-commuting observables and quantum fractals on complex projective spaces.*, PREprint (2003).

[32] ———, *On quantum iterated function systems*, PREprint (2004).

[33] Stephen T. Thornton Jerry B. Marion, *Classical dynamics of particles and systems*, Saunders College Publishing, Toronto, 1995.

[34] A. Lozinski K. Zyczkowski, *Euroattractor: A brief introduction to iterated function systems*, PREprint (2002).

[35] James A. Yorke Kathleen T. Alligood, Tim D. Sauer, *Chaos an introduction to dynamical systems*, Springer-Verlag, New York, 1996.

[36] Witold Kinsner, *Chaos in engineering, lecture notes*, 2004.

[37] Lui Lam, *Introduction to nonlinear physics*, Springer-Verlag, New York, 1997.

[38] Bai lin Hao, *Symbolic dynamics and characterization of complexity*, Physica D (1991), 161–176.

[39] Ö. Stenflo M. Barnsley, J.E. Hutchinson, *A fractal valued random iteration algorithm and fractal hierarchy*, PREprint (2003).

[40] Ö Stenflo M. Barnsley, J.E. Hutchinson, *V-variable fractals and superfractals*, PREprint (2003).

[41] M. Nappi M. Polvere, *Speed-up in fractal image coding: Comparison of methods*, IEEE transactions on Image Processing **9** (2000), no. 6.

[42] Ronnie Mainier, *Arithmetical properties of dynamical zeta functions*, (2004).

[43] L.P. Hurd M.F. Barnsley, *Fractal image compression*, AK Peters, Massachusetts, 1993.

[44] Jorge Millan, *Studying a driven pendulum with a 1d-circle-map*, (2003).

[45] J.P. Reilly N.B. Tufillaro, T. Abbott, *An experimental approach to nonlinear dynamics and chaos*, Addison-Wesley, New York, 1990.

[46] P. Fisher O. Kiselyov, *Image compression with iterated function systems, finite automata and zerotrees: Grand unification*, Physical Review D **1** (2000), no. 1.

[47] ———, *Image compression with iterated function systems, finite automata and zerotrees: Grand unification*, PREprint (2003).

[48] Edward Ott, *Chaos in dynamical systems*, Cambridge University Press, Maryland, 1993.

[49] R. Olkiewiczy P. Blanchard, A. Jadczyk, *Completely mixing quantum open systems and quantum fractals*, PREprint (2000).

[50] I. Procaccia P. Cvitanović, G.H. Gunarante, *Topological and metric properties of hnon-type strange attractors*, Physical Review A **38** (August, 1988), no. 3, 1503–1520.

[51] M Sieber P. Cvitanović, E. Bogomolny, *On dynamical zeta function*, Chaos, An Interdisciplinary Journal of Nonlinear Science (1992), 5–13.

[52] R. Mainieri P. Cvitanović, R. Artuso, *Chaos: Classical and quantum*, Nield Bohn Institute, Copenhagen, 2003.

[53] U. Moenig P. Grassberger, H. Kantz, *On the symbolic dynamics of the henon map*, J. Phys. A **22**, 5217–5230.

[54] R.K. Pathria, *Statistical mechanics*, Butterworth Heinemann, Oxford, 1996.

[55] S. Pelěs, *Nonlinear phenomena in periodically driven classical systems*, Ph.D. thesis, University of Manitoba, 2001.

[56] Henri Poincaré, *Sur les courbes defines par les equations differentieles*, Journal De Mathematique Pures et Appliques (1885), 167–244.

[57] M. Lefranc R. Gilmore, *The topology of chaos*, John Wiley & Sons, New York, 2002.

[58] A.J. Penner R. Kobes, *Non-linear fractal interpolating functions of one and two variables*, PREprint (2002).

[59] H. Letkeman R. Kobes, *Nonlinear fractal interpolating functions*, PREprint (2001).

[60] S. Peles R. Kobes, J. Liu, *Analysis of a parameterically driven pendulum*, PREpaper (2000).

[61] C.D. Shaw R.H. Abraham, *Dynamics–the geometry of behavior*, Aerial Pree, Santa Cruz, California, 1983.

[62] K. Lee S. Kim, *Multiple transitions to chaos in a damped parametrically forced pendulum*, PREprint (1995).

[63] M. Sieber, *Applications of periodic orbit theory*, Chaos, An Interdisciplinary Journal of Nonlinear Science (1992).

[64] N.E. Frankel S.P. Drake, C.P. Dettman, *Chaos in special relativistic dynamics*, Physical Review E **53** (1996), no. 2.

[65] Frank Steiner, *Quantum chaos*, Chaotic Dynamics.

[66] Y. Sambonsugi T. Ida, *Self-affine mapping system and its application to object contour extraction*, IEEE transactions on Image Processing **9** (2000), no. 11.

[67] I.S. Reed P.C. Lee A.Q. Li T.K. Truong, J.H. Jeng, *A fast encoding algorithm for fractal image compression using the dct inner product*, IEEE transactions on Image Processing **9** (2000), no. 4.

[68] S. Hue V. Afraimovich, *Lectures on chaotic dynamical systems*, American Mathematical Society ·International Press, New York, 2003.

[69] Z. Zheng V. Franceschini, C. Giberti, *Characterization of the lorenz attractor by unstable periodic orbits*, Nonlinearity **6** (1993), 251–258.

[70] Divakar Viswanath, *Symbolic dynamics and periodic orbits of the lorenz attractor*, Nonlinearity **16** (2003), 10351056.

[71] ———, *The fractal property of the lorenz attractor*, Physica D **190** (2004), 115–128.

[72] Stephen Welstead, *Fractal and wavelet image comression techniques*, SPIE Optical Engineering Press, Washington, 1999.

[73] P. Cvitanović Y. Lan, *Variational method for finding periodic orbits in a general flow*, PREprint (2003).