Exploring Fractal Dimension Calculations for Chaotic Systems.

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
John Aufiero

Annandale-on-Hudson, New York
December 2012

Abstract

This work discusses fundamental tools for the analysis and understanding of chaotic dynamical systems, develops numerical routines for the computation of these methods. Various examples of know chaotic systems are reviewed and used to develop the programs using Mathematica. New examples are investigated and analyzed. In particular, various dimension concepts of fractals have been studied. Further, Lyapunov exponents and Lyapunov spectrum is carried out to insure that we are studying chaotic systems. New notions of dimension have been developed, and new areas for application of the methods studied have been outlined, including the analysis of a special class of complex network called T-networks.

Dedication

To my Mom and Grandma who never stopped believing in me. I wish you both could see me now! To my father and brother for always being there for me.

Acknowledgments

Thank you to my family. To Professor G. Gospodinov who made this project possible and inspired me to create T-networks. To Professor A. Zito for all his help early in my mathematic career. To Max Kenner and Dan Karpowitz for believing in a person that most of the world has given up on. To the B.P.I. staff for all your help. The Bard college campus Professors for all their help in making this possible. Thanks to Professor Stone for all his help completing this project.

Contents

List of Figures

List of Tables

Chapter 1

Preliminary Concepts and Results

1.1 What is Chaos?

Chaos deals with long–term evolution, how something changes over a long time. A reasonable definition of chaos is sustained and disorderly looking long-term evolution that satisfies certain mathematical criteria and that occurs in a nonlinear system. The mathematical requirements for chaos are:

1. Sensitivity to initial conditions,
2. a fractal dimension,
3. and a positive Lyapunov exponent.

The focus of this work centers around the last two requirements for chaos. Specifically, we will be looking at geometric structures and testing them for fractal dimensions and positive Lyapunov exponents.

For the Lyapunov exponent, there is a straight forward method for computing it that will be discussed in Section 2.1. Unlike the Lyapunov exponent, there are several methods for computing the fractal dimension of a system. Some of the methods that we will be studying in Sections 2.2 through 2.5 include the box-counting dimension, the information dimension, and the correlation dimension. With so many different methods for calculating the dimension of a fractal, we are left with a question.

Question 1. *Given a geometric structure produced by a system of equations can we say that the system truly behaves chaotically, and can we determine which dimension calculation method offers the best results?*

In order to confirm that a system is chaotic, we will use the Lyapunov exponent calculation. When discussing the best results for a dimension calculation, we are referring to the fastest method computationally with accurate results.

Our findings indicate that the box-counting dimension procedure offers results that are unreliable. The cluster dimension procedure suffers from a necessity to have a scaler region in order to provide reliable results. No such scaler region exist of yet, and may be considered for future work. The information dimension procedure offers the best results for a dimension calculation out of all the procedures. All figures and data was generated using Mathematica. Before beginning our research, we will introduce the environment in which chaos lives and the terms associated with chaos.

1.2 Introduction to Chaos and the definitions surrounding it.

Most of the definitions and ideas in this section can be found in standard introductory text on chaos theory such as [1, 7, 14, 3]. The problem of understanding chaotic behavior boils down to reconciling the apparently conflicting notions of randomness and predictability. The key element to understanding chaotic behavior is the notion of nonlinearity. A **nonlinear system** is a system whose time evolution equations are nonlinear. That is, the dynamical variables describing the properties of the system (for example, position, velocity, pressure, etc.) appear in the equations in a nonlinear form. We can develop an intuitive idea of nonlinearity by characterizing the behavior of a system in terms of stimulus and response: First we give the system a "kick" and observe a certain response to that kick. Next we ask the question what happens if we kick the system twice as hard. If the response is twice as large, then the system's behavior is said to be linear. On the other hand, if the response is not twice as large (larger or smaller), then we say the system's behavior is nonlinear. The unpredictability of a chaotic system is better explained by considering the evolution of the same system, but started twice,

with slightly different initial conditions. The error in the initial conditions can be due to measurement discrepancies. For nonchaotic systems, this error will grow linearly with time, whereas for chaotic systems the error will grow exponentially with time so after a short time the state of the system is unknown. Prediction becomes impossible after a certain moment (this is also known as sensitivity to initial conditions). This means that the system is subject to some irregularity. This irregularity, or randomness, however, is quite different from that of a stochastic system (a system subject to random external forces) because it is a part of the nature of the system and its evolution dynamics.

A **dynamical system** is a set of equations specifying how certain variables change over time. The equations specify how to determine (compute) the new values as a function of their current values and control parameters (parameters in the equations of a dynamic system are allowed to change, thus the behavior of the system would also change). The functions, when explicit, are either difference (or algebraic) equations or differential equations. A **difference equation** is a function specifying the change in a variable from one discrete point in time to another. A **differential equation** is a function that specifies the rate of change in a continues variable over changes in another variable (usually time). Dynamical systems may be stochastic or determinist. In a stochastic system, new values come from a probability distribution. In a deterministic system, a single new value is associated with any current value, that is, precise knowledge of the conditions of the system at one time allows us, at least in principle, to predict exactly the future behavior of that system. Examples include the mathematical models that describe the swinging of a clock pendulum, the flow of water in a pipe, and the number of fish in a lake. At any given time, a dynamical system has a state given by a set of real numbers (a vector) that can be represented by a point in an appropriate state space or phase space—an abstract space used to represent the behavior of a system. Its dimensions are the variables of the system. Thus, a point in the phase space defines a potential state of the system. The points on the **state orbit** or **trajectory** achieved by a system depend on its iterative function and initial conditions (starting point). Loosely speaking, the trajectory is the succession of points of the geometric object in a space derived by the solution to an equation.

The state that a dynamical system eventually "settles down to" is called an **attractor.**

An attractor is a set of values in the phase space to which a system migrates to over time or iterations. An attractor can be a single fixed point, a collection of points regularly visited, a loop, a complex orbit, or a infinite number of points. Attractors can have as many dimensions as the number of variables that influences its systems. In more general terms, the attractor is the set of points to which trajectories approach as the number of iterations goes to infinity. As we shall see for more complicated systems, the system may have more than one attractor for a given parameter value. **The basin of attraction** for a particular attractor consists of the initial points, each of which gives rise to a trajectory that approaches the attractor as the number of iterations approaches infinity. A **limit cycle** is a particular type of attractor that is periodic in time, that is, it cycles periodically through an ordered sequence of states. In general, there are three kinds of limit points: attractors, repellers, and saddle points. A system moves away from repellers and towards attractors. A **saddle point** is both an attractor and a repeller, it attracts a system in certain regions, and repels the system in other regions.

For a system described by a set of first-order differential equations, a **fixed point** is a point in the state space for which all of the time derivatives of the state space variables are zero for that system (also called an equilibrium point, a critical point, or a singular point). If the system starts at one of these fixed points, it stays at that fixed point for all time. Since the time derivatives of the state space variables are zero at the fixed point, those variables cannot change in time.

The crucial importance of chaos is that it provides an alternative explanation for the apparent randomness of a dynamical system that is chaotic, meaning it provides an explanation for behavior that depends on neither noise nor complexity. Chaotic behavior shows up in systems that are essentially free from noise and are also relatively simple–meaning only a few degrees of freedom are active. Of course, if we believe that chaos does play a role in any given experiment, we need to establish that noise is not a major factor, and we ought to know the number of active degrees of freedom. Chaos theory provides us with the tools necessary to develop this analysis.

Several means of distinguishing chaos from noise have been developed and used successful both in theoretical calculations and experimental studies. The one that seems to be the easiest

to use experimentally is based on the divergence of nearby trajectories. The **Lyapunov number** or **Lyapunov exponent** is the value of an exponent, a coefficient of time, that reflects the rate of departure of dynamic orbits. It is a measure of sensitivity to initial conditions, and, as such, it is one of the main ways of detecting chaotic or chaos-like behavior. The concept of the Lyapunov exponent will be dealt with in great detail in Section 2.1.

The importance of divergence of nearby trajectories is the following: if a system displays divergence of nearby trajectories for some range of its parameter values, then the behavior of that system becomes essentially unpredictable. The system is still deterministic, so if we know the initial conditions of a trajectory exactly, then we could predict the future behavior of that trajectory by iterating the system. But if we make the smallest change to those initial conditions, the trajectory quickly follows a completely different path. Since there is always some imprecision in specifying initial conditions in any real experiment or real numerical calculation, we see that the actual future behavior is in fact unpredictable for a chaotic system. To make this point more forcefully, we say that the future of a chaotic system is indeterminable even though the system is deterministic.

In Section 2.2 we will study geometric structures called fractals in great detail. As a brief introduction, we will say that fractals are irregular geometric shapes that exhibit self-similarity. It has infinite detail, and cannot be differentiated. **Self-similarity** is an infinite nesting of structure on all scales. Strict self-similarity refers to a characteristic that a form exhibits when a substructure resembles a superstructure exactly. This means that each subpart, when appropriately magnified, looks just like the larger part. Self-similar objects are called fractals because their geometric dimension (suitably defined) is often a fraction, not an integer. If a geometric structure exhibits self-similarity, then it has no inherent size scale. That is, if we look at some subsection of this structure, at some level of magnification, it looks like any other subsection at some level of magnification. This remarkable feature means that many features of the geometric structure must be independent of the details of the model that gave rise to it.

In Sections 2.3 through 2.5 we will explore different calculation methods for fractal dimension. **Fractal dimension** is a measure of a geometric object that can take on fractional

values. At first, fractal dimension was synonymous with Hausdorff dimension; however, fractal dimension is now used as a general term for a measure of how fast length, area, or volume increases with decrease in scale. We will explore the concept of dimension very closely.

1.3 Approximating Solutions

Given the Initial Value Problem, abbreviated IVP,

$$y'(t) = f(t, y),\ y(t_0) = y_0,$$

by the Existence and Uniqueness Theorem (below), we know that the IVP has a unique solution (curve) providing the rate function $f(t, y)$ is well-behaved.

Theorem 1.3.1. *[2, Theorem 3.1] Let $a, b > 0$ and suppose f and df/dy are continuous on the rectangle R given by $|x - x_0| < a$ and $|y - y_0| < b$. Then there exist an $h > 0$ so that the initial value problem*

$$y' = f(x, y), \qquad y(x_0) = y_0$$

has one and only one solution $y = y(x)$ for $|x - x_0| \leq h$.

Remark 1.3.2. Due to Theorem 1.3.1, we know that different trajectories will never intersect no matter how many iteration are preformed. If two trajectories did intersect, then there would be two solutions starting from the same point (the intersection point), and this would violate the uniqueness part of the theorem. In general terms, a trajectory cannot move in two directions at once. Attractors always have a well-groomed look to them because trajectories cannot intersect.

This remark will be very important when we calculate the Lyapunov exponent and dimension of a system because it guarantees that we are not missing a point because it coincides with another point. This simply means there can be no point (x, y) on the Cartesian Plane that possess two trajectory points generated by the system of equations.

1.3.1 Euler's Method

The basic principle of numerically solving an IVP consist of integrating the rate function along an interval. The simplest method for generating approximations for the solution curves of an IVP of the above type is Euler's method, which is a first-order numerical method using the linear approximation to a curve at a point via its derivative at that point.

To formulate the method, we first partition the interval $t_0 \leq t \leq T$ into N equal steps of step size h

$$h = \frac{T - t_0}{N},$$

$t_n = t_0 + nh,$ where $n = 1, 2, 3 \ldots, N$.

The initial point $(t_0, y(t_0))$ is on the solution curve, so we find the Euler approximation (t_1, y_1) for the next point $(t_1, y(t_1))$ by following the tangent line to the solution curve at $(t_0, y(t_0))$. We obtain

$$y_1 = y_0 + hf(t_0, y(t_0)),$$

which, for small enough h, is an approximation to $y(t_1)$. Similarly, we use the new point (t_1, y_1) to construct an approximate y_2 for $y(t_2)$:

$$y_2 = y_1 + hf(t_1, y_1).$$

Definition 1.3.3. In general, **Euler's method** for the IVP $y'(t) = f(t, y), y(t_0) = y_0$ is defined by the recursive formula

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) \tag{1.1}$$

where $t_n = t_{n-1} + h$ and $1 \leq n \leq N$.

Connecting the points $(t_0, y_0), (t_1, y_1), \ldots, (t_N, y_N)$ produces the Euler approximation to the solution curve at $(t_0, y(t_0))$ for the IVP. Euler's method is called a **one-step method** because we only need a single evaluation of the slope (given by the differential equation) to obtain the value of y_n (y from the value of y_{n-1}).

To put Euler's method into practice, we need a formula for determining (t_{k+1}, y_{k+1}) from (t_k, y_k). Finding t_{k+1} is easy. We specify the step size Δt at the outset, so

$$t_{k+1} = t_k + \Delta t.$$

To obtain y_{k+1}, from (t_k, y_k), we use the differential equation. We know that the slope of the solution to the equation $dy/dt = f(t, y)$ at the point (t_k, y_k) is $f(t_k, y_k)$, and Euler's method uses this slope to determine y_{k+1}. In fact, the method determines the point (t_{k+1}, y_{k+1}) by assuming that it lies on the line through (t_k, y_k) with slope $f(t_k, y_k)$.

Now we can use our basic knowledge of a slope to determine y_{k+1}. The formula for the slope of a line gives

$$\frac{y_{k+1} - y_k}{t_{k+1} - t_k} = f(t_k, y_k).$$

Since $t_{k+1} = t_k + \Delta t$, the denominator $t_{k+1} - t_k$ is just Δt, and hence we have

$$\frac{y_{k+1} - y_k}{\Delta t} = f(t_k, y_k).$$

$$y_{k+1} - y_k = f(t_k, y_k)\Delta t$$

$$y_{k+1} = y_k + f(t_k, y_k)\Delta t$$

which is Euler's method exactly.

Example 1.3.4. Euler's method can be used for approximating a solution to the IVP

$$y' = y + x, \qquad y(1) = 0,$$

at the x values

$$1.1, 1.2, 1.4, 1.5.$$

Substituting the values of x into the equation we have

$$y_1 = 0 + f(1, 0)(1.1 - 1) = 0.1,$$

$$y_2 = 0.1 + f(1.1, 0.1)(1.2 - 1.1) = 0.22,$$

$$y_3 = 0.22 + f(1.2, 0.22)(1.4 - 1.2) = 0.504,$$

$$y_4 = 0.504 + f(1.4, 0.504)(1.5 - 1.4) = 0.6944.$$

Thus, the first four points (x, y) on the graph of the approximate solution are $(1.1, 0.1)$,$(1.2, 0.22)$, $(1.4, 0.504)$,$(1.5, 0.6944)$.

In general, Euler's method is fast, but not very accurate, and also suffers from stability problems. For these reasons, the Euler method is not often used in practice. However, it serves as the basis to construct more advanced and efficient methods.

1.3.2 Runge-Kutta Methods

In 1895, Carle Runge used Taylor's formula to develop techniques for generating numerical solutions to initial value problems. In 1901, M.W. Kutta improved on Runge's work [2, p.175]. The technique that these two individuals developed are now known as Runge-Kutta methods and they are very popular for approximating solutions to IVPs becuase of there accuracy ([4, 5, 3, 2]). We will employ an advanced version of a 5th-order Runge-Kutta method in our work here, so we outline it below, and show a few numerical comparisons between its various refinements.

The 4th-order Runge-Kutta method involves a weighted average of the slopes at the points t_{n-1}, and $t_{n-1} + \dfrac{h}{2}$ to approximate the value of the solution curve at the point t_n.

Definition 1.3.5. Given the IVP $y'(t) = f(t, y), y(t_0) = y_0$, the **RK4** for the approximate value $y_n \approx y(t_n)$ from $y_{n-1} \approx y(t_{n-1})$ is as follows:

$$y_n = y_{n-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{1.2}$$

where h is the time step and

$$\begin{aligned}
k_1 &= f(t_{n-1}, y_{n-1}),\\
k_2 &= f\Big(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_1\Big),\\
k_3 &= f\Big(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_2\Big),\\
k_4 &= f\Big(t_{n-1} + h, y_{n-1} + hk_3\Big).
\end{aligned}$$

The slopes k_i for the RK4 are determined as follows:

1. Slope k_1 is given as in Euler's method. $k_1 = f(t_{n-1}, y_{n-1})$.

2. Slope k_2 is the slope at the point obtained by moving halfway along the slope line at (t_{n-1}, y_{n-1}) to the intermediate point $(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_1)$, so that $k_2 = f\Big(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_1\Big)$.

3. Slope k_3 is the slope at the point obtained by moving halfway along a different straight line at (t_{n-1}, y_{n-1}), where the slope is now k_2 rather than k_1 as before. Hence, $k_3 = f\Big(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_2\Big)$.

4. Lastly, slope k_4 is the slope at point (t, y) where we use a line with slope k_3 to determine this point. Hence, $k_4 = f\Big(t_{n-1} + h, y_{n-1} + hk_3\Big)$.

   Thus, the step from $y(t_{n-1})$ to $y(t_n)$ is given by moving along a straight line whose slope is a weighted average of the four values: $\frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Example 1.3.6. We will use the forth order Runge–Kutta method with step size $h = 0.1$ and iteration count $N = 2$ to obtain a numerical solution to the IVP

$$y' = 2y - x, \qquad y(0) = 1$$

For $h = 0.1$ and $N = 1$ we have

$$\begin{aligned}
k_1 &= f(0, 1) = 2, \\
k_2 &= f\Big(0 + \frac{0.1}{2}, 1 + \frac{0.1}{2}(2)\Big) = 2.15, \\
k_3 &= f\Big(0 + \frac{0.1}{2}, 1 + \frac{0.1}{2}(2.15)\Big) = 2.165, \\
k_4 &= f\Big(0 + 0.1, 1 + 0.1(2.165)\Big) = 2.333.
\end{aligned}$$

Thus, we get

$$y_1 = 1 + \frac{0.1}{6}\Big(1.2 + 2(2.15) + 2(2.165) + 2.333\Big) = 1.21605.$$

The first point (x, y) on the graph of the approximate solution is $(0.1, 1.21605)$.

For $h = 0.1$ and $N = 2$ we have

$$\begin{aligned}
k_1 &= f(0.1, 1.21605) = 2.3321, \\
k_2 &= f\Big(0.1 + \frac{0.1}{2}, 1.21605 + \frac{0.1}{2}(2.3321)\Big) = 2.51531, \\
k_3 &= f\Big(0.1 + \frac{0.1}{2}, 1.21605 + \frac{0.1}{2}(2.51531)\Big) = 2.533631, \\
k_4 &= f\Big(0.1 + 0.1, 1.21605 + 0.1(2.533631)\Big) = 2.7388262.
\end{aligned}$$

Thus, we get

$$y_2 = 1 + \frac{0.1}{6}\Big(2.3321 + 2(2.51531) + 2(2.533631) + 2.7388262\Big) = 1.46886.$$

Thus, we have calculated y_1 and y_2. The second point (x, y) on the graph of the approximate solution is $(0.2, 1.46886)$. Further approximate solutions to the IVP are produced with more iteration of the RK4.

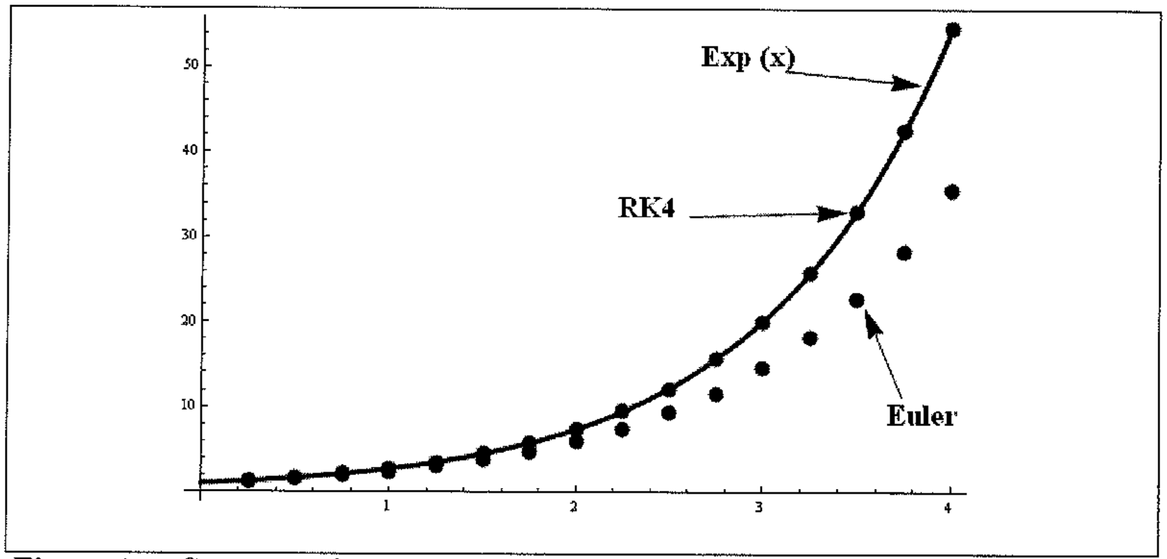

Figure 1.1: Comparison between Euler and RK4 for $y = e^x$. The interval step h is equal to 0.25. Presented here are the first 16 evaluations of Euler's method and the Rk4 method (solid points), and the function e^t (solid line) for $0 \leq t \leq 4$.

Figure 1.1 offers a visual comparison between Euler's method and the RK4 for approximating the solution to the IVP $y'(t) = y(t), y(0) = 1$ which is solved analytically by $y(t) = e^t$. Clearly, RK4 is much more accurate. Table 1.1 shows the global error of Euler's method and the RK4 method for up to 50 iterations with step size $h = 0.25$. Each error estimate is given by the absolute value of the difference between the exponential function and the approximation given by each method at t_n. At the 50th iteration, the Euler error is about 75%, while the RK4 error is about 0.03%.

Number of iterations n	e^{t_n}	**Euler error** $\lvert e^{t_n} - y_n^E(t_n)\rvert$	**RK4 error** $\lvert e^{t_n} - y_n^{RK4}(t_n)\rvert$
10	12.18255	2.869276	0.000805447
20	148.4136	61.67712	0.019624176
30	1808.047	1000.253	0.358593941
40	22,026.58	14,503.38	5.824552344
50	268,337.1	198,272.2	88.69411567

Table 1.1: Comparison between Euler and RK4 for $y = e^t$. The interval step h is equal to 0.25 and $t_n = 0.25n$.

We look at two refinements of the RK4 method. The Runge–Kutta–Gill and the Runge–

Kutta–Fehlberg offer better precision and round-off error prevention.

Definition 1.3.7. Given the IVP $y'(t) = f(t, y), y(t_0) = y_0$, the **Runge-Kutta-Gill** for approximate value $y_n \approx y(t_n)$ from $y_{n-1} \approx y(t_{n-1})$ is given by:

$$y_n = y_{n-1} + \frac{h}{6}\left(k_1 + (2-\sqrt{2})k_2 + (2+\sqrt{2})K_3 + k_4\right), \tag{1.3}$$

where

$$\begin{aligned}
k_1 &= f(t_{n-1}, y_{n-1}),\\
k_2 &= f\left(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}k_1\right),\\
k_3 &= f\left(t_{n-1} + \frac{h}{2}, y_{n-1} + h\Big(\frac{\sqrt{2}-1}{2}\Big)k_1 + h\Big(1 - \frac{1}{\sqrt{2}}\Big)k_2\right),\\
k_4 &= f\left(t_{n-1} + h, y_{n-1} + h\Big(-\frac{1}{\sqrt{2}}\Big)k_2 + h\Big(1 + \frac{1}{\sqrt{2}}\Big)k_3\right).
\end{aligned}$$

Definition 1.3.8. For the IVP $y'(t) = f(t, y), y(t_0) = y_0$ the **Runge-Kutta-Fehlberg** (RK5) for the approximate value $y_n \approx y(t_n)$ from $y_{n-1} \approx y(t_{n-1})$ is given by:

$$y_n = y_{n-1} + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right), \tag{1.4}$$

where

$$\begin{aligned}
k_1 &= f(t_{n-1}, y_{n-1}),\\
k_2 &= f\left(t_{n-1} + \frac{1}{4}h, y_{n-1} + h\frac{1}{4}k_1\right),\\
k_3 &= f\left(t_{n-1} + \frac{3}{8}h, y_{n-1} + h\Big(\frac{3}{32}k_1 + \frac{9}{32}k_2\Big)\right),\\
k_4 &= f\left(t_{n-1} + \frac{12}{13}h, y_{n-1} + h\Big(\frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\Big)\right),\\
k_5 &= f\left(t_{n-1} + h, y_{n-1} + h\Big(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\Big)\right),\\
k_6 &= f\left(t_{n-1} + \frac{1}{2}h, y_{n-1} + h\Big(\frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\Big)\right).
\end{aligned}$$

There are a number of other RK approximation methods such as the RK8. They all essentially function as the RK4 but with additional steps and different coefficients. Table 1.2 shows a comparison between the global errors in the two RK4 methods and the Rk5 method. The RK4 methods do not differ in precision, while the RK5 method offers a vast improvement precision.

Number of iterations n	e^{t_n}	**Euler error** $\|e^{t_n} - y_N^E\|$	**RK4 error** $\|e^{t_n} - y_n^{RK4}\|$	**RK4Gill error** $\|e^{t_n} - y_n^{RK4G}\|$	**RK5Felberg error** $\|e^{t_n} - y_n^{RK5F}\|$
10	2.71828	0.12458	2.08×10^{-6}	2.08×10^{-6}	2.28×10^{-8}
30	20.0855	2.63645	4062×10^{-5}	4.62×10^{-5}	5.06×10^{-7}
50	148.413	31.0224	0.00056934	0.00056934	6.23×10^{-6}
70	1096.63	306.893	0.00588612	0.0058861	6.45×10^{-5}
90	8103.08	2790.12	0.0559194	0.0559194	6.13×10^{-4}
110	59,874.1	24,130.8	0.505011	0.505011	0.00553158
130	442,413	201,950	4.41002	4.41002	0.0483047

Table 1.2: Comparison of Runge-Kutta methods with time step $h = 0.1$.

We will employ the Rk5 method in our numerical computations, as it shows a hundredfold improvement to the 4-step Runge-Kutta methods yielding an error of 0.0001% after 130 iterations, which is a very significant stability and precision improvement. The built-in numerical solvers in mathematical programs such as Mathematica, use adaptive-step RK4 variations. We have not implemented adaptive-step algorithms, which is a study in its own right, due to the limited availability of computational resources and the overall focus of this work. In particular, for dimension calculations, we recommend a fixed time step in order to generate an attractor structure as uniformed and detailed as possible.

1.4 Known and New Examples

In this section, we will build a library of basic dynamical system models that offer different perspectives such as dimension, equation formulation, and chaotic behavior characteristics.

1.4.1 Logistic Equation

The simple Logistic Equation [3, p.17] is a formula for approximating the evolution of an animal population over time. The system is better described by a discrete difference equation than a continuous differential equation, and the population increase will be some fraction of the present population. Let x_n denote the number of animals in a given year, then $x_n = Ax_{n-1}$, where A is the growth rate. This model produces exponential growth without limit. If there is a carrying capacity of the environment, then the population may not exceed that capacity, or it would become extinct. This can be modeled by multiplying the population by a number that approaches zero as the population approaches its limit. The resulting logistic equation becomes

$$x_n = Ax_{n-1}(1 - x_{n-1}). \tag{1.5}$$

The Logistic equation is a one-dimension dynamical system with chaos-like behavior that is parabolic. Varying the parameter changes the height of the parabola but leaves the width unchanged. All initial conditions eventually settle into one of three different types of behaviors. Firstly, the population can approach a stable value or a fixed point. It can do so by approaching asymptotically from one side or from both sides. The second possibility is that the population alternates between two or more fixed values, exhibiting a periodic behavior. Likewise, it can do so by approaching asymptotically in one direction or from both sides in an alternating manner. Periodic orbits can be either stable or unstable. The periodicity is discrete; that is, there are no intermediate values. The third case, and the one of interest to us is when the population eventually visits every neighborhood in a subinterval of $(0, 1)$, exhibiting behavior that we will call chaos-like behavior or quasi-chaotic, as the system lacks the dimensionality and capacity to exhibit chaos in the full dynamical sense. Nested among the points it does visit, there is a countably infinite set of fixed points and periodic points of every period. The chaotic orbits exhibit sensitive dependence on initial conditions such that any two nearby points will eventually diverge in their orbits to any arbitrary separation one chooses.

1.4.2 Hénon Map

The first two-dimensional system we will work with is the Hénon map (figure 1.2) [3, p.610] which is not defined by a system of differential equations, but by a system of nonlinear algebraic equations

$$F(x, y) = \Big(y + 1 - ax^2, bx\Big).$$

The discrete version is given as

$$\begin{aligned} x_n &= y_{n-1} + 1 - ax_{n-1}^2, \\ y_n &= bx_{n-1}, \end{aligned} \tag{1.6}$$

where a and b are adjustable parameters.

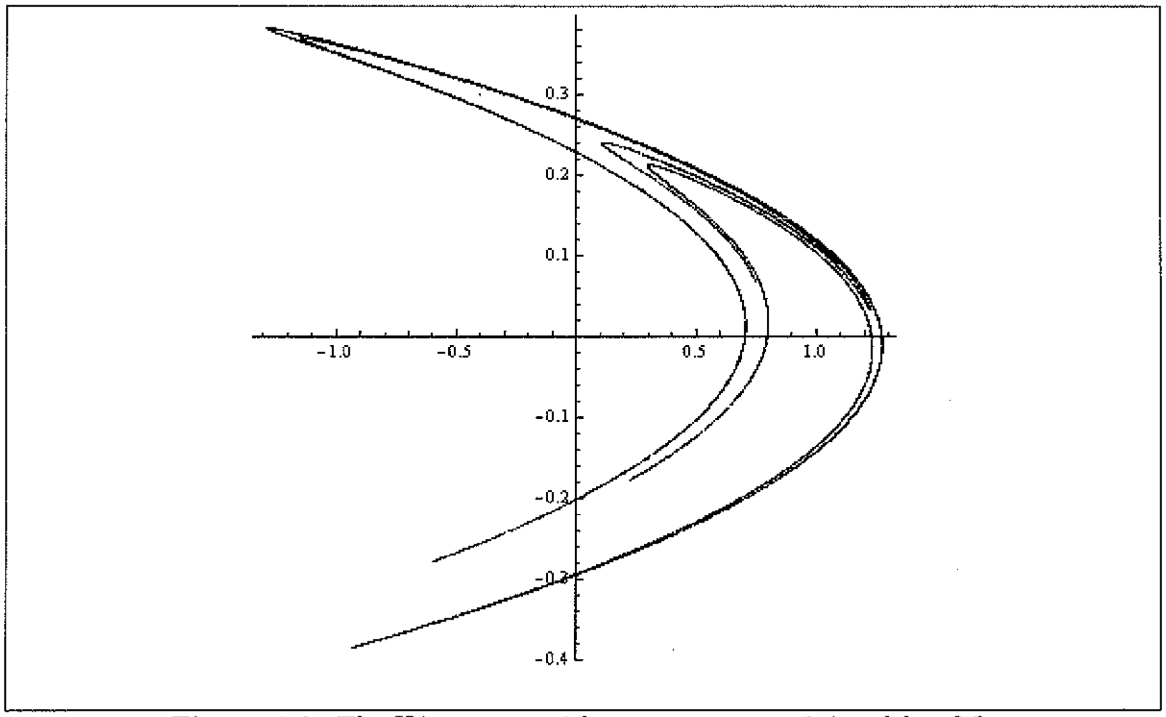

Figure 1.2: The Hénon map with parameters $a = 1.4$ and $b = 0.3$.

Hénon was trying to study another famous system we will include in our library, the Lorenz system, but instead of dealing with the difficult task of resolving the tightly packed sheets of the Lorenz attractor, he developed a simplified map that behaved in essentially the same way as the Lorenz system. The Hénon map stretches and folds the plane, and then contracts,

then reflects it across a line, which creates the transformation for a map that is nonlinear and invertible, so it exhibits behavior similar to the uniqueness characteristic of an IVP. The Hénon map is dissipative, that is, it contracts areas in phase space. In addition, for certain parameter values, the Hénon map has an attractor region, that is a region that gets mapped inside itself by the flow, yet has trajectories that escape to infinity since the quadratic term does dominate behavior away from the origin.

1.4.3 Lozi Map

Our next map is the Lozi map (figure 1.3)[14, p.451] which also exhibits a structure similar to the Hénon map.

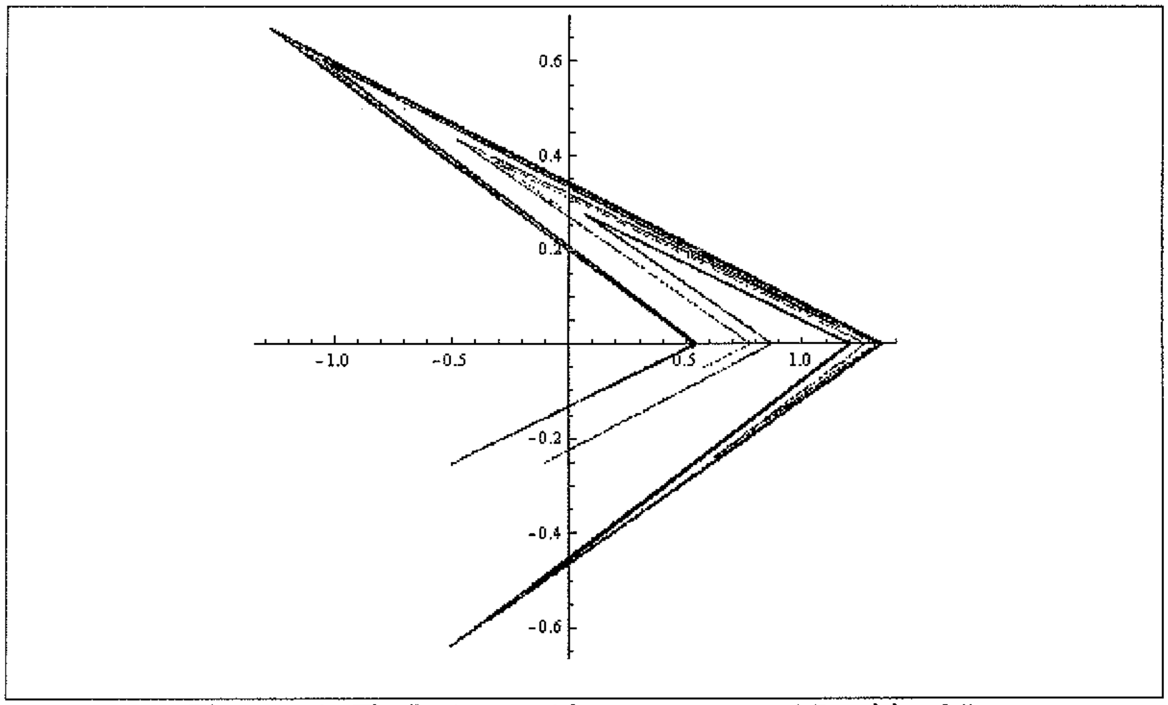

Figure 1.3: The Lozi map with parameters $a = 1.7$ and $b = 0.5$.

The Lozi map is given by the system of equations

$$x_n = 1 + y_{n-1} - a|x_{n-1}| \tag{1.7}$$

$$y_n = bx_{n-1},$$

where a, b are parameters.

1.4.4 Ikeda Map

Another map that has a pair of algebraic equation instead of a system of differential equations is the Ikeda map (figure 1.4) [3, p.202].

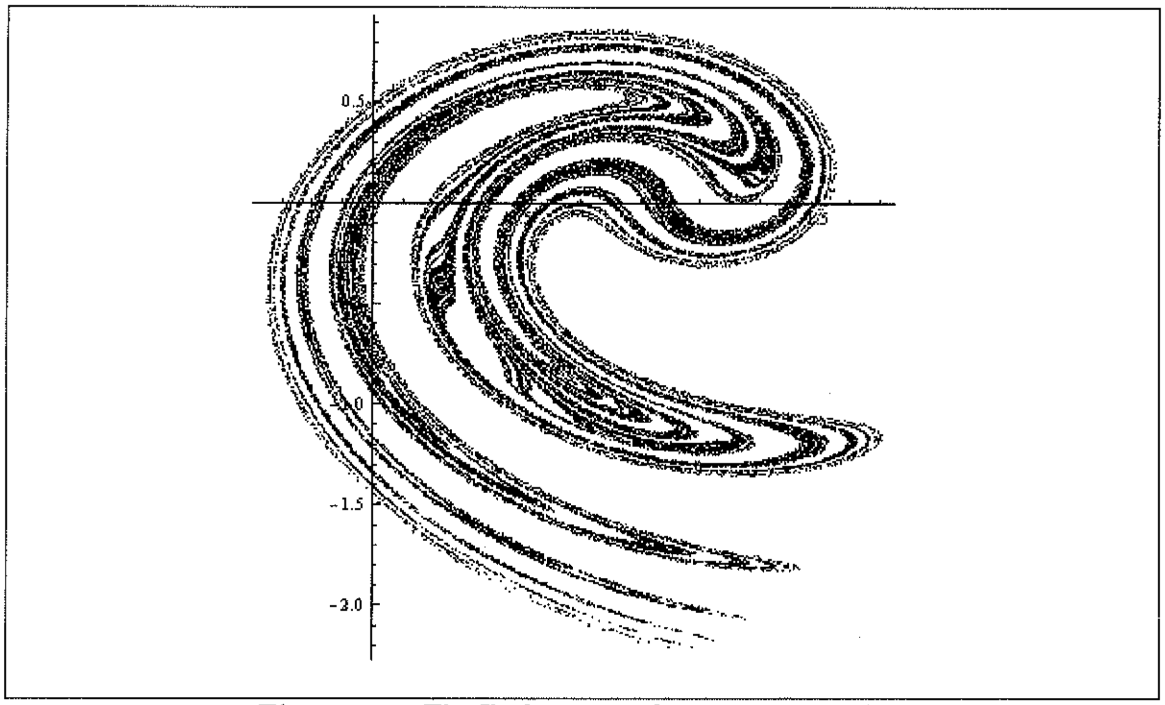

Figure 1.4: The Ikeda map with parameter $c = 0.9$.

The equations for the Ikeda map is given by

$$x_n = 1 + C\Big(x_{n-1}\cos(\tau_{n-1}) - y_{n-1}\sin(\tau_{n-1})\Big) \tag{1.8}$$
$$y_n = C\Big(x_{n-1}\sin(\tau_{n-1}) + y_{n-1}\cos(\tau_{n-1})\Big),$$

where C is a parameter and

$$\tau_n = 0.4 - \frac{6}{1 + x_n^2 + y_n^2}.$$

This map was proposed as a model, under some assumptions, of the type of cell that might be used in an optical computer. For values of the parameter C that are near $C = 1$, the map has quasi-chaotic behavior.

1.4.5 Sprott 2D maps

One of the simplest systems that can produce a large variety of such images is the general time-delayed quadratic map family (figure 1.5). These were created by Sprott [6]. The equations are give as

$$x_n = a_1 + a_2 x_{n-1} + a_3 x_{n-1}^2 + a_4 x_{n-1} y_{n-1} + a_5 y_{n-1} + a_6 y_{n-1}^2 \tag{1.9}$$
$$y_n = x_{n-1},$$

where a_i are the parameters that govern the behavior.

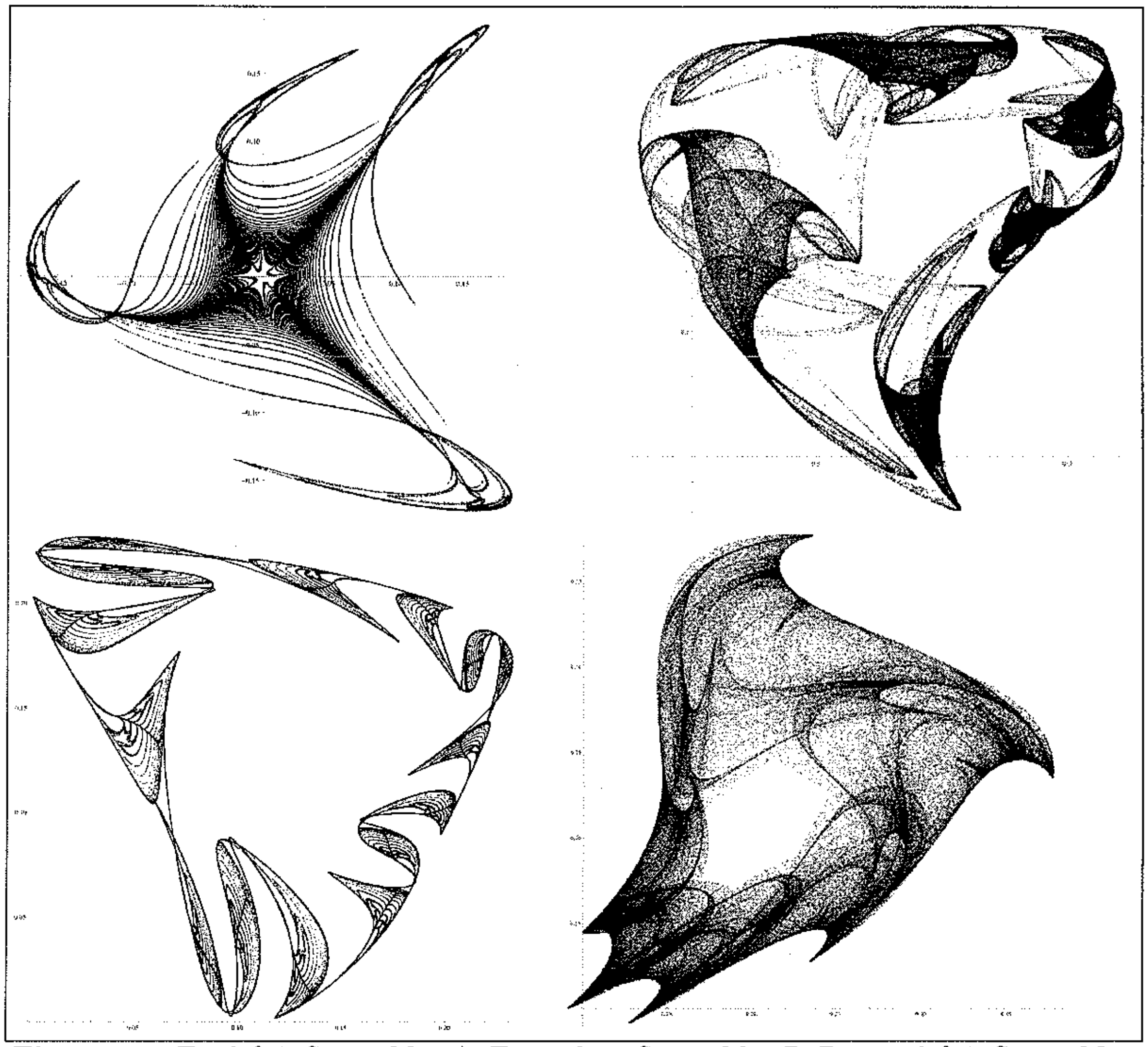
Figure 1.5: Top left is Sprott Map A. Top right is Sprott Map B. Bottom left is Sprott Map C, and bottom right is Sprott Map D.

We will use these four Sprott examples to verify our algorithms and computational routines, and to optimize our methods. This six-dimension parameter space is vast and admits an enormous variety of forms. Some examples with integer values of the parameters have been studied by Sprott, and have proved to provide visually interesting and geometrically important features. Non-chaotic solutions are excluded in the usual way by testing for sensitive dependence on initial conditions. Formally, this is done by calculation the Lyapunov exponent for each system and discarding cases for which it is not clearly positive. For the examples here, Sprott performed two simultaneous calculations in which the initial conditions (typically taken as $x_0 = y_0 = 0.05$) differed by some small amount such as 10^{-6}, and discarded cases for which any subsequent iterate differed by less than this amount. It was also possible to discriminate against attractors that were too thin (line-like) or too thick (area-filling) by calculating their fractal dimension ([6]).

1.4.6 Ueda Map

The first two-dimensional flow defined as a pair of differential equations we will encounter is the Ueda map (figure 1.6)[14, p.453],

$$\begin{aligned} x'(t) &= y(t) \\ y'(t) &= -0.1y(t) - x(t)^3 + 11.5\cos t. \end{aligned} \tag{1.10}$$

which we include here as the first physical experiment that not only exhibited but also was proven to have clear chaotic behavior. The Ueda map is in fact a 3-dimensional system in disguise, by letting t be an independent variable and adding the equation $t' = 1$ to the system.

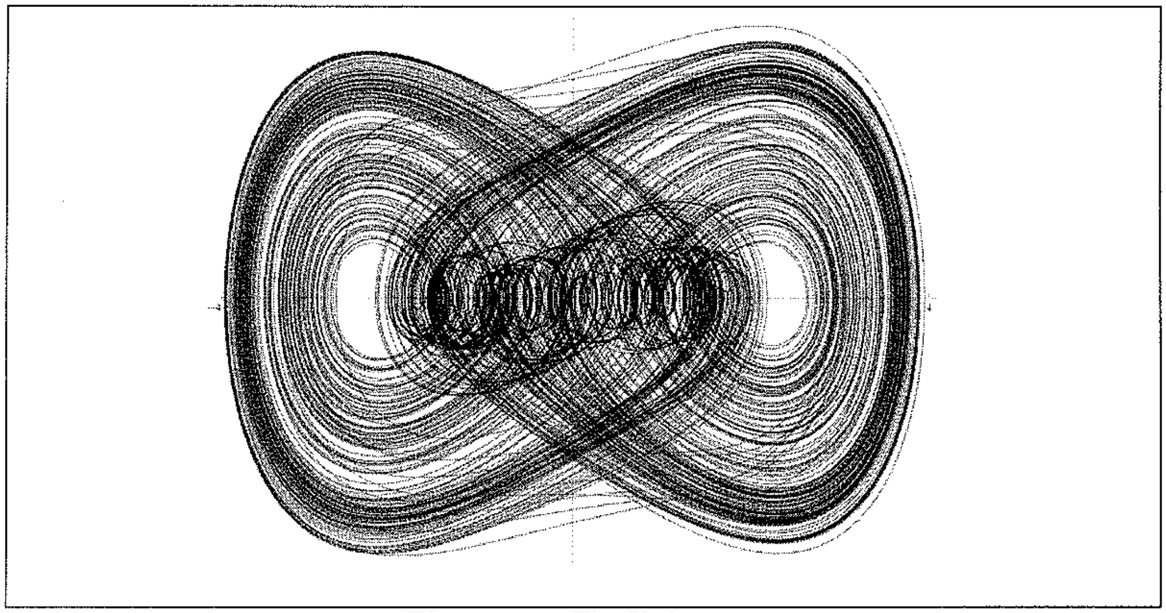

Figure 1.6: The Ueda map.

1.4.7 Lorenz Map

The next example on our list is the Lorenz map (figure 1.7), a set of three first-order non-linear differential equations discovered by E. Lorenz in his search for a model that explains atmospheric behavior [3, p.362]. This simple deterministic system turns out to exhibit very complicated behavior over a wide range of parameters. Solutions have irregular oscillatory behavior, are bounded to a region in space, and seem to intersect (but do not in reality).

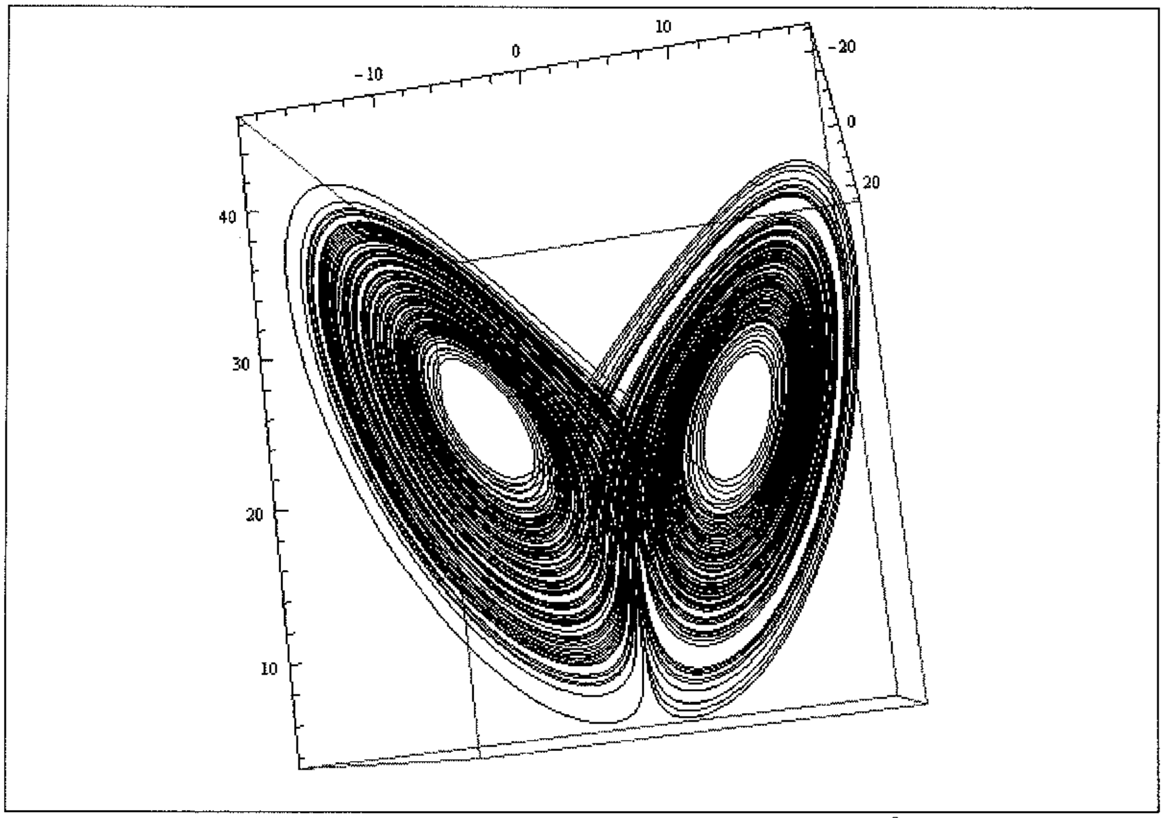

Figure 1.7: The Lorenz map with parameters $\sigma = 10, b = \frac{8}{3}, r = 28$.

The equations for the Lorenz map are given by

$$
\begin{aligned}
x' &= \sigma(y - x) \qquad (1.11)\\
y' &= rx - y - xz\\
z' &= xy - bz,
\end{aligned}
$$

where x, y, z are all functions of t, and $\sigma, r, b > 0$ are all parameters.

The Lorenz system is nonlinear and contains the quadratic terms xy and xz. It is volume contracting, that is, the system evolves the points in a region of space into another region of smaller surface area or volume. The Lorenz system also has fixed points, one of them is the origin, which is stable for all values of the parameter. The other two points lie in the center of the two "wheels" of he Lorenz map, but are subjected to the values of the parameters. In fact, for certain values of the parameter r, one of the fixed points coincides with the origin. The most fundamental feature of the Lorenz system is its sensitivity to initial conditions, which

manifest itself as exponential divergence of nearby trajectories. That is, solutions with almost identical initial conditions exhibit drastically different long-term behavior ([7, 14, 3]).

1.4.8 The A2DM and A3DM Systems

Lastly, we would like to add a family of nonlinear dynamical systems that have not yet been investigated. No published results are known about the family of differential equations of the following type. No know parameter values or initial conditions are known for chaotic behavior. We will report our findings on the investigation of these systems as we go along. We will call this family the A2DM discrete family of dynamical systems

$$\begin{aligned} x_{n+1} &= Ay_n + x_n, \\ y_{n+1} &= Bf(y_n) + g(x_n, y_n) + x_n. \end{aligned} \tag{1.12}$$

The functions $f(y_n)$ and $g(x_n, y_n)$ are certain discrete nonlinear function of x_n and y_n, such as $\cos y_n, y_n^m, \sin y_n$, etc, and A and B are parameters. We allow only two parameters to facilitate our search. We will explore different values of the parameter A and different functions $f(y_n)$.

We will also consider the continuous case, the three-dimensional continuous family of dynamical systems called the A3DM systems

$$\begin{aligned} x' &= Ay, \\ y' &= Bz, \\ z' &= Cf(x, y) + g(x, z) + h(y, z) + x, y, z, \end{aligned} \tag{1.13}$$

where x, y, z are functions of t, and A, B, C, f, g, h are as above. We will use the RK5 algorithm to approximate and study this family of nonlinear dynamical systems.

Chapter 2

Fundamental Ideas and Calculations

In this chapter, we will outline some fundamental tools for the analysis of chaotic systems, the Lyapunov exponent, and various dimension concepts. Their main purpose is understanding the behavior of a dynamical system and the structure of the corresponding trajectory in phase space.

Additionally, we reproduce known calculation using original Mathematica routines, then we use them to study the examples from Section 1.4. We have generated all images, table data, and results, with a few exceptions where explicitly stated.

Thus, we prepare an arsenal of tools we can use to investigate the new two and three-dimensional dynamical systems produced by Equations 1.12 and 1.13. We develop these fundamental ideas further, and apply them to new realms, such as complex network analysis in Section 3.3.

2.1 Lyapunov Exponent Calculations

Before we can truly understand what the Lyapunov exponent represents and how it is derived, we first need to develop some concepts. The first definition we will develop is that of a fixed point.

Definition 2.1.1. Let x be a point in $\mathbb{R}^n$ and let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a map. The **orbit** of x under f is the set of points $\{f^i(x)|i \in \mathbb{Z}_{\geq o}\}$. The starting point x for the orbit is called the

initial value of the orbit. A point $p \in \mathbb{R}^n$ is a **fixed point** of the map f if $f(p) = p$.

Example 2.1.2. Let $g(x) = 2x(1-x)$ be the function mapping from the real line $\mathbb{R}$ to itself. The orbit of $x = 0.01$ under g is $\{0.01, 0.0198, 0.0388, \ldots\}$. The initial point is $x = 0.01$, and the fixed points of $g(x)$ are $x = 0$ and $x = 1/2$. The fixed points are found by solving the equation $g(x) = 0$.

We also need to develop the concept of "near" or "close". We do this so that when we speck of the Lyapunov exponent as a measure of divergence or convergence of nearby points, we will have a clear understanding of the concept of near. The concept of "near" or "close" is made precise by referring to all real numbers within a distance ϵ of p as the ϵ **neighborhood** $N_\epsilon(p)$.

Definition 2.1.3. Let $p \in \mathbb{R}$ such that $\epsilon > 0$. The $N_\epsilon(p)$ is the interval of numbers $\{x \in \mathbb{R} : |x - p| < \epsilon\}$.

Now that we have defined the notion of a fixed point and the concept of "near", we are ready to develop the concept of a sink and a source. We do this in order to show how a point affects nearby points and what it means to the overall dynamics of the system.

Definition 2.1.4. Let f be a real valued map on $\mathbb{R}$ and let p be a real number such that $f(p) = p$. If all points sufficiently close to p are attracted to p, then p is called a **sink** or an **attracting fixed point**. More precisely, if there is a $\epsilon > 0$ such that $N_\epsilon(p)$, $\lim_{k\to\infty} f^k(x) = p$ for all $x \in N_\epsilon(p)$, then p is a sink. If all points sufficiently close to p are repelled from p, then p is called a **source** or a **repelling fixed point**. More precisely, p is a source if there is an epsilon neighborhood $N_\epsilon(p)$ such that for all x in $N_\epsilon(p)$ $f^k(x) \notin N_\epsilon(p)$ for k sufficiently large except for p itself.

In the limit of the above definition $f^k(x)$ is the composition of the function f that represents the evolution of the dynamical process. It is defined $f^2(x) = f(f(x))$ and in general, defined $f^k(x)$ to be the results of applying the function f to the initial state k times. Given an initial value of x, we want to know about $f^k(x)$ for large k.

Theorem 2.1.5. *[3, Theorem 1.5] Let f be a real value map on $\mathbb{R}$, and assume that p is a fixed point of f.*

1. If $|f'(p)| < 1$, then p is a sink.

2. If $|f'(p)| > 1$, then p is a source.

Proof. [3, p.10] For Part 1, let a be any number between $|f'(p)|$ and 1. For example, a could be chosen to be $(1 + |f'(p)|)/2$. Since

$$\lim_{x\to p} \frac{|f(x) - f(p)|}{|x - p|} = |f'(p)|,$$

there is a neighborhood $N_\epsilon(p)$ for some $\epsilon > 0$ so that

$$\frac{|f(x) - f(p)|}{|x - p|} < a,$$

for x in $N_\epsilon(p)$, $x \neq p$.

In other words, $f(x)$ is closer to p then x is by at least a factor of a (which is less than 1). This implies two things: first, if $x \in N_\epsilon(p)$, then $f(x) \in N_\epsilon(p)$; that means that if x is within ϵ of p, then so is $f(x)$, and by repeating the argument, so are $\{f^2(x), f^3(x)\ldots\}$. Second, it follows that

$$|f^k(x) - p| \leq a^k|x - p|$$

for all $k \geq 1$. Thus, p is a sink.

Part two of Theorem (2.1.5) is proven the same way except we take a greater than 1 and reverse the inequality. □

In the above proof, we take for granted that

$$G(x) = \lim_{x\to p} \frac{|f(x) - f(p)|}{|x - p|} = |f(x)| = \left| \lim_{x\to p} \frac{f(x) - f(p)}{x - p} \right|.$$

Now we will proof it. To do so we will need the definition of the limit.

Definition 2.1.6. Let f be a function defined on some open interval that contains the number p, except possibly at p itself. Then we say that the **limit of f(x) as x approaches a is L,**

and we write

$$\lim_{x \to p} f(x) = L$$

if for every number $\epsilon > 0$ there is a corresponding number $\delta > 0$ such that if

$$0 < |x - p| < \delta \text{ then } |f(x) - L| < \epsilon.$$

Furthermore, to proof that $G(x) = |f(X)|$, we will need the triangle inequality definition.

Definition 2.1.7. By the Triangel inequality, we have

$$\Big||a| - |b|\Big| \leq \Big|a - b\Big|,$$

where a is the larger-sized number.

Proof. Let $G(x) = \dfrac{f(x) - f(p)}{x - p}$. We want to show that $G(x) \to |f(x)|$ as $x \to p$ such that for all $\epsilon > 0$ there exists $\delta > 0$ when $|x - p| < \delta$

We have $\left|\dfrac{f(x) - f(p)}{x - p} - f'(p)\right| < \epsilon$.

$$|G(x) - f'(p)| \leq \left|\left|\frac{G(x) - G(p)}{x - p}\right| - \left|\lim_{x \to p} \frac{f(x) - f(p)}{x - p}\right|\right| < \epsilon \tag{2.1}$$

$$\leq \frac{G(x) - G(p)}{x - p} - \frac{f(x) - f(p)}{x - p} - \epsilon \tag{2.2}$$

We drop the absolute values in equation 2 because the function will be positive. furthermore, as $x \to p$ both $G(x) \to 0$ and $f(x) \to 0$ for all $\epsilon > 0$. Thus, we have shown that

$$G(x) = \lim_{x \to p} \frac{|f(x) - f(p)|}{|x - p|} = |f(x)| = \left|\lim_{x \to p} \frac{f(x) - f(p)}{x - p}\right|,$$

for all $\epsilon < 0$.

□

There are two more concepts that we have to develop before diving into the Lyapunov exponent. The first is that of a periodic point of period k.

Definition 2.1.8. Let f be a real value map on $\mathbb{R}$. We call p a **periodic point of period k** if $f^k(p) = p$, and if k is the smallest such positive integer. The orbit with initial point p (which consist of k points) is called a **periodic orbit of period** k.

Example 2.1.9. Consider the map defined by $f(x) = -x$ on $\mathbb{R}$. This map has one fixed point, at $x = 0$. Every other real number is a period–two point, because f^2 is the identity map.

Definition 2.1.10. Let f be a real value map and assume that p is a period–k point. The period–k orbit of p is a **period sink** if p is a sink for the map f^k. The orbit of p is a **period source** if p is a source for the map f^k.

At this point it will be helpful to review the chain rule of calculus, which shows us how to expand the derivatives of a composition of functions. Let f and g be real value functions on $\mathbb{R}$. Then

$$(f \circ g)'(x) = f'(g(x))g'(x). \tag{2.3}$$

For a period–two, the chain rule is simply saying $(f^2)'(x) = f'(f(x))f'(x)$. The derivative of f^2 at a point of a period–two orbit is simply the product of the derivatives of f at the two points in the orbit.

Remark 2.1.11. In general, the chain rule for higher periods of iterate functions is as follows. Let $\{p_1, \ldots, p_k\}$ denote a period–k orbit of f. The chain rule says

$$\begin{aligned}(f^k)'(p_1) &= (f(f^{k-1}))'(p_1)\\ &= f'(f^{k-1}(p_1))(f^{k-1})'(p_1)\\ &= f'(f^{k-1}(p_1))f'(f^{k-2}(p_1))\cdots f'(p_1)\\ &= f'(p_k)f'(p_{k-1})\cdots f'(p_1).\end{aligned} \tag{2.4}$$

This formula tells us that the derivative of the k^{th} iterate of f^k of f at a point of a period–k orbit is the product of the derivatives of f at the k points of the orbit.

2.1.1 Lyapunov Exponents of One-Dimensional Maps

Remark 2.1.12. Discrete dynamical systems' stability is heavily influenced by the derivative of the map. If x_1 is a fixed point of a one–dimensional map f and $f'(x) = \mathbf{a} > 1$, then the orbit of each point x near x_1 will separate from x_1 at a multiplicative rate of approximately $\mathbf{a}$ per iteration until the orbit of x moves significantly far away from x_1. That is, the distance

between $f^n(x)$ and $f^n(x_1) = x_1$ will be magnified by approximately $a > 1$ for each iteration of f.

Example 2.1.13. Let f be a map on $\mathbb{R}$ given by $f(x) = (3x - x^3)/2$. The map has three fixed points, namely $-1, 0,$ and 1. However, for the purpose of the magnification factor, we are only interested in the fixed point 0, and the points in the neighborhood surrounding 0. For instance, we take the initial value $x = 0.1$ and iterate it in our map. To the fourth decimal, we have $f(0.1) = 0.1495, f^2(0.1) = 0.2226, f^5(0.1) = 0.6587$. The point $x = 0.1$ is moved by f to approximately 0.1495, a magnification factor of 1.495. This magnification factor turns out to be approximately the derivative $f'(0) = 1.5$. That is, the derivative of the map of f evaluated at the fixed point $f(0) = 0$. We are not interested in the fixed points $x = 1$ and $x = -1$ because $f'(1) = f'(-1) = 0$. They both have no magnification factor due to the fact that the derivative of the system evaluated with these values equals zero.

Before we give a formal definition for the Lyapunov exponent, we will build a case for it. For a periodic point of period-k, we have to look at the derivative of the k^{th} iterate of the map, which, by the chain rule (see Remark 2.1.9) is the product of the derivatives at the k^{th} point of the orbit. Suppose this product of derivatives is $A > 1$. Then the orbit of each neighbor x of the periodic point x_1 separates from x_1 at a rate of approximately A after each k iterate. This is a cumulative amount of separation—it takes k iterations of the map to separate by a distance of A. It makes sense to describe the average multiplicative rate of separation as $A^{1/k}$ per iteration.

The term Lyapunov number is introduced to quantify this average multiplicative rate of separation of points x very close to x_1. The Lyapunov exponent is simply the natural logarithm of the Lyapunov number. A Lyapunov exponent of $\ln 2$ for the orbit of x_1 means that the distance between the orbit of x_1 and the orbit of a nearby point x doubles each iteration on average. For a function $f : \mathbb{R} \to \mathbb{R}$ and a periodic point x_1 of period-k, this is the same as saying

$$|(f^k)'(x_1)| = |f'(x_1)||f'(x_2)| \cdots |f'(x_k)| = 2^k.$$

But we want to consider this concept even when x_1 is not a fixed point or a periodic point.

Additionally, a real value function whose Lyapunov exponent is between 1 and 0 would mean the distance between the two points converges, and the orbits of the two points x and x_1 would move rapidly closer.

The significance of the concept of the Lyapunov exponent is that it can be applied to non-periodic orbits. In order to formally define the Lyapunov number and the Lyapunov exponent for a general orbit, we follow the above analogy for the periodic case, and consider the product of the derivatives at points along the orbit. We begin by restricting our attentions to one-dimensional maps.

Definition 2.1.14. Let f be a real value map of the real line $\mathbb{R}$. The **Lyapunov number** $L(x_1)$ of the orbit $\{x_1, x_2, x_3, \cdots\}$ is defined as

$$L(x_1) = \lim_{n\to\infty}(|f'(x_1)| \ldots |f'(x_n)|)^{1/n} \tag{2.5}$$

if the limit exists.

The **Lyapunov exponent** $h(x_1)$ is defined as

$$h(x_1) = \lim_{n\to\infty}(1/n)\left[\ln|f'(x_1)| + \cdots + \ln|f'(x_n)|\right] \tag{2.6}$$

if the limit exists.

Remark 2.1.15. Notice that h exists if and only if L exists and is nonzero, and $\ln L = h$.

It follows from the definition that the Lyapunov number of a fixed point x_1 for a one–dimensional map f is $|f'(x_1)|$, or equivalently, the Lyapunov exponent of the orbit is $h = \ln|f'(x_1)|$. If x_1 is a periodic point of period–k, then it follows that the Lyapunov exponent is

$$h(x_1) = \frac{\ln|f'(x_1)| + \cdots + \ln|f'(x_k)|}{k} \tag{2.7}$$

The point is that for a periodic orbit, the Lyapunov number $e^{h(x_1)}$ describes the average local stretching, on a per–iterate basis, near a point on the orbit. This concept is applied directly to a non–periodic orbit.

Additionally, equation (2.4) can be rewritten as

$$h(x_i) = \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n} ln|f'(x_i)|$$

and substituting λ for $h(x_i)$, we have

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} ln|f'(x_i)|. \tag{2.8}$$

where λ is defined as the Lyapunov exponent.

Now that we understand the concept of the Lyapunov exponent and how it works, we can formalize a definition for a chaotic orbit.

Definition 2.1.16. Let f be a real value map of the real line $\mathbb{R}$, and let $\{x_1, x_2, x_3, \ldots\}$ be a bounded orbit of f. The orbit is **chaotic** if

1. $\{x_1, x_2, x_3, \ldots\}$ is not periodic,
2. The Lyapunov exponent λ is greater than zero.

Example 2.1.17. Consider the tent map

$$T(x) = \begin{cases} 2x & \text{if} \quad x \le 1/2, \\ 2(1-x) & \text{if} \quad x \ge 1/2 \end{cases} \tag{2.9}$$

on the unit interval $[0,1]$. Using the Lyapunov exponent definition, we compute λ as follows

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln|f'(x_i)| = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln 2 = \ln 2.$$

Since there is a cusp at $x = 1/2$, $T(x)$ is not differentiable. So we restrict our attention to orbits that never map to the point $x = 1/2$. A Lyapunov exponent of $\lambda = \ln 2$ corresponds to a chaotic orbit. It displays the stretching rate of orbits that do not map to the point $x = 1/2$. In fact, there are many chaotic orbits of the tent map.

Theorem 2.1.18. *[3, Theorem 3.9] The tent map has infinitely many chaotic orbits.*

Proof. Since the absolute value of the slope of T is 2 whenever it exists (for $x \neq 1/2$), the Lyapunov exponent of an orbit of T is $\ln 2$ whenever it is defined. Any orbit that avoids $1/2$ and is not periodic is therefore a chaotic orbit.

Any periodic orbit of the tent map must be eventually periodic. The reason is that the derivative of T^k at a period-k orbit is 2^k, so all periodic orbits must have an eventually

repeated iterates. There are infinitely many non-repeating iterates that correspond to distinct chaotic orbits.

This also demonstrates that the Lyapunov exponent evaluates the function correctly. □

Example 2.1.19. For the logistic equation produced by Equation 1.5 in Section 1.4 is

$$x_i = Ax_{i-1}(1 - x_{i-1}),$$

where A is a constant. We have $f(x_i) = Ax_i - Ax_i^2$, so $f'(x_i) = A - 2Ax_i$. Therefore, the Lyapunov exponent for the logistic equation is

$$\lambda \approx \frac{1}{n}\sum_{i=1}^{n} \ln|f'(x_i)| = \frac{1}{n}\sum_{i=1}^{n} \ln\Big|A(1-2x_i)\Big|,$$

for large n.

The parameter A in the Logistic Equation makes it hard to calculate the Lyapunov exponent exactly; however, it is easy to graph a representation of the Lyapunov exponent for the map. We do this in the figure below. It is a published fact that the Logistic Equation produces values of points that display sensitivity to initial conditions for values of A in the range $(3.55699, 4]$ [14, p.369]. Figure 2.1 shows the output of our program for various values of the parameter A. It verifies the range of the A values for which the Lyapunov exponent has a positive sign.

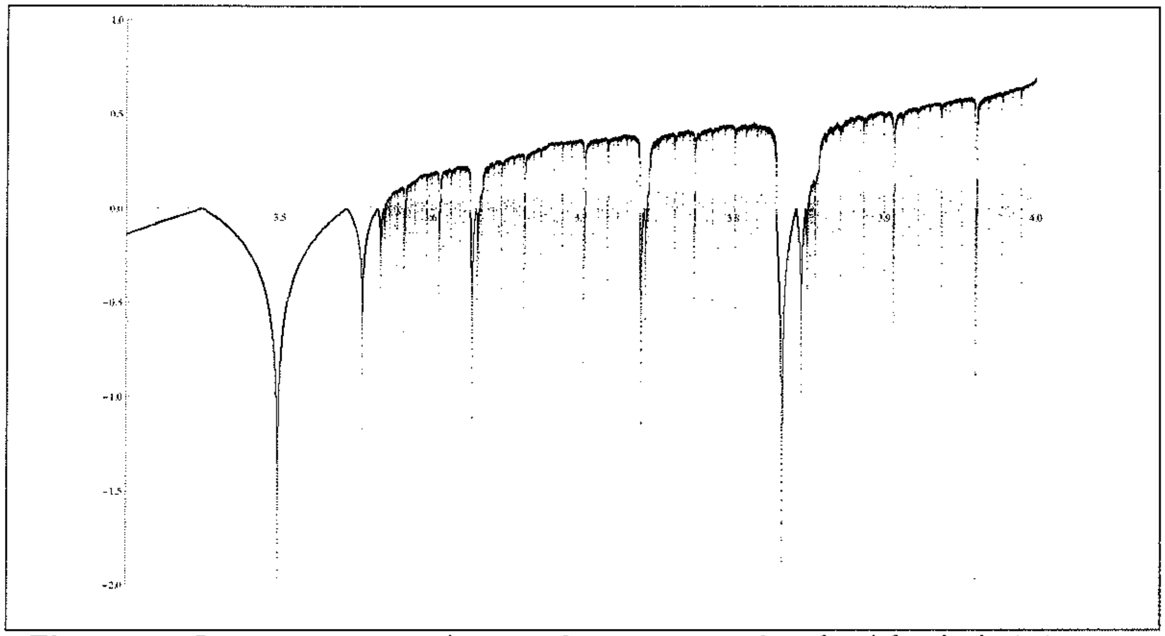

Figure 2.1: Lyapunov exponent λ versus the parameter values for A for the logistic map.

In Figure 2.1, the value of the parameter A begins with $A = 2.8$ and incrementing by 0.00001 up to $A = 4$ is considered. The initial x_0 is chosen at random in the interval $[0, 1]$. We ran the logistic equation for 1000 iterations first then discarded them, and then for $10,000$. We computed the above geometric average of the logarithm of the divergence of trajectories. One can see that beyond approximately $A = 3.56$ the positive λ corresponds to chaotic behavior.

2.1.2 Lyapunov Exponent for Two–Dimensional Maps

Let us now consider the Lyapunov exponent for a two-dimensional map. The reason to look at the Lyapunov exponent for maps of different dimension is to build up to the multi-dimensional case when we come to three or higher dimensional maps, then we will give the full and accurate definition of Lyapunov exponents and Lyapunov spectrum. Also, we will present the accurate algorithm for calculating Lyapunov exponents. Since one and two-dimensional maps could be thought of as projections of a three-dimensional phase space onto the line or plane, we present them separately and give different interpretations and algorithms for computing the exponents.

The Lyapunov exponent for a one-dimensional map can be interpreted as the average

stretching rate along the line averaged over the trajectory. We are going to interpret the Lyapunov exponent for a two-dimensional map in terms of a two-dimensional stretching (stretching of some part of the phase plan). Let us suppose that around our initial point on the trajectory we have a unit disc with a diameter d. After one iteration, the new point on the attractor will have a new disc around itself: the transformed image of the unit disc under the mapping. It will likely be slightly distorted, that is, it will resemble an ellipse. The diameter of the ellipse that is in the direction of greatest perturbation will correspond to the direction of greatest (rate of) divergence of the nearest trajectories. The short axis measures how many points are attracted in one iteration.

There are two general ways for computing the Lyapunov exponent: (1) by allowing a small finite error and renormalizing the new error to the same value (and same direction) after each iteration. (2) The second method is by assuming the error is infinitesimally small and then the notion of derivatives is used to iterate the error vectors and their average rate of stretching per iteration is measured.

We will present a Mathematica program which employs the second method (the more stable and accurate method) to compute the exponent for the Hénon map. As found in [4, p.663], the basic outline of the algorithm is as follows.

1. *Initialization.* Iterate the initial point 1,000-10,000 times to arrive at a point (x, y) on the attractor. Initialize a storage variable (an accumulator).

2. *Initial error.* For an arbitrary angle ϕ, consider $(\cos\phi, \sin\phi)$, the direction of the error, or the error vector, which is defined as the difference from the two nearby trajectories.

3. *Transformation.* To find the amplification factor of an infinitesimally small error, we need to define a tangent space to the phase space.

4. *Error amplification.* The error has increased (or changed) by the factor

$$d = \sqrt{(-2ax_n\cos\phi + \sin\phi)^2 + (b\cos\phi)^2},$$

and we accumulate the logarithm of this factor to calculate the logarithm of the average geometric mean of the error growth.

5. *Renormalization.* The old point (x_n, y_n) is replaced by the new point $(x_{n+1}, y_{n+1}) = f(x_n, y_n)$ in the iterator. The error $(\cos\phi, \sin\phi)$ is replaced by the new normalized directional vector
$$\frac{1}{d}(-2ax_n\cos\phi + \sin\phi, b\cos\phi).$$

6. *Loop.* Go back to step three until a given number of iterations have been performed.

7. *Result.* Divide the contents of the accumulated quatity by the number of iterations. This gives us the equivalent of the logarithm of the geometric mean of divergence of the trajectory originating at (x_0, y_0):
$$\lambda \approx \frac{1}{n}\sum_{i=0}^{n-1}\ln(\text{error amplification factor}).$$

Step three of the above algorithm can be some what tricky, so we will break it down further now. We need the derivatives of the functions defining the trajectory. Consider a two-dimensional autonomous system of the form
$$x' = f(x, y),$$
$$y' = g(x, y).$$
Here the differentiation is with respect to t, and x and y are functions of t. Then the 2×2 **Jacobian matrix** of partial derivatives of the system is given by
$$D = \begin{bmatrix} \frac{\partial}{\partial x}f & \frac{\partial}{\partial y}f \\ \frac{\partial}{\partial x}g & \frac{\partial}{\partial y}g \end{bmatrix}.$$

In the above, the partial derivatives are to be evaluated at a particular point $(x^\star, y^\star) = (x + dx, y + dy)$ on a trajectory of the system. The Jacobian matrix D is the multivariable equivalent of the derivative $f'(x^\star)$, the linearization of the flow at a point $x^\star$ of a one-dimensional system $x' = f(x)$ or, in the discrete case, $x_{n+1} = f(x_n)$. For the Hénon attractor, a two-dimensional flow is given by the transformation
$$F(x_{n+1}, y_{n+1}) = \begin{bmatrix} 1 + y_n - ax^2 \\ bx_n \end{bmatrix},$$

where $a = 1.4$ and $b = 0.3$, so the Jacobian JF is given by

$$JF(x_{n+1}, y_{n+1}) = \begin{bmatrix} \frac{\partial}{\partial x}(1 + y_n - ax^2) & \frac{\partial}{\partial y}(1 + y_n - ax^2) \\ \frac{\partial}{\partial x}(bx_n) & \frac{\partial}{\partial y}(bx_n) \end{bmatrix} = \begin{bmatrix} -2ax_n & 1 \\ b & 0 \end{bmatrix}.$$

Using the Jacobian, we can compute how an infinitesimally small error in a point of the attractor is transformed by an iteration of the system of equations. If the error is given by the vector (dx, dy), we multiply this vector by the Jacobian:

$$\begin{bmatrix} -2ax_n & 1 \\ b & 0 \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} -2ax_n dx + dy \\ bdx \end{bmatrix}.$$

The amplification factor in the error is given by the ratio of the lengths of the two vectors $(dx, dy) = (\cos\phi, \sin\phi)$ and $(-2ax_n dx + dy, bdx)$. So, given the initial direction vector, the Jacobian transforms it. Next we iterate the point (x_n, y_n) under the flow, and the error (dx, dy) under the Jacobian to obtain the transformed error $(-2ax_n \cos\phi + \sin\phi, b\cos\phi)$.

The published results for the Lyapunov exponent for the Hénon map gives us a value of 0.4192 ± 0.001 for $1,000,000$ iterations using the method described above [4, p.663]. After 1000 iterations, we ran this algorithm for $100,000,000$ iterations, and obtained a value of 0.41926, in agreement with the published value.

Below, we summarize our findings for two-dimensional discrete and continuous systems. For the continuous ones, we used an RK5 algorithm to iterate the system within the Lyapunov exponent algorithm.

System	**Lyapunov exponent λ**	**Number of iterations**
Logistic equation	$0 < \lambda, 1$ for $3.5569, A, 4$	$10,000$ for each $3.4 \leq A \leq 4$
Hénon	$\lambda = 0.419257$	$100,000,000$
Lozi	$\lambda = 0.470487$	$1,000,000$
Sprott Map A	$\lambda = 0.362897^*$	$1,000,000$
Sprott Map B	$\lambda = 0.514662^*$	$1,000,000$
Sprott Map C	$\lambda = 0.067355^*$	$1,000,000$
Sprott Map D	$\lambda = 0.312275^*$	$1,000,000$
Ikeda	$\lambda = 0.299563$	$1,000,000$
Ueda	$\lambda = 1.911861$	$500,000$

Table 2.1: Lyapunov exponents calculations of various systems (*new result)

2.1.3 Lyapunov Exponent for Three-Dimensional Maps

Let us finally consider the Lyapunov exponent for a three-dimensional system. We need to define and discuss the Lyapunov spectrum first. Since the Lyapunov exponents are orthogonal quantities representing the average divergence of nearby trajectories in phase space, there is a Lyapunov exponent in the direction of each of the axis (since we can choose our second trajectory to start from a point which denotes some perturbation along each of the different dimensions of the phase space). This is not technically true. Given a continuous dynamical system in a n-dimensional phase space, the long-term behavior of the system is being monitored by the expansion/contraction of an n-dimensional sphere of infinitesimally small volume in phase space. After some period of time, the sphere will distort into an ellipsoid due to the deforming nature of the flow. So the Lyapunov exponents are related to the expanding or contracting of the flow of the system in different directions. Since the position (orientation) of the ellipsoid changes as it develops, the directions associated with each exponent vary throughout the attractor. Thus, we cannot speak of an exponent associated with each axis. Note, however, that the volume of the ellipsoid grows as $e^{(\lambda_1+\lambda_2+\cdots+\lambda_n)t}$ ([4, p.665]). Each of the lambdas represents the measure of the growth or contraction rate of the function, so there is a Lyapunov exponent associated with each of the dimensions of the phase space, regardless of the orientation of the ellipsoid. The set of all Lyapunov exponents represents the **Lyapunov spectrum**. A system is referred to as chaotic if its spectrum contains at least one positive Lyapunov exponent. One should also note that the presence of a positive Lyapunov exponent does not necessarily imply chaos, as we saw, for instance, the simple one and two-dimensional flows do possess positive Lyapunov exponents but they lack enough dimensions to exhibit chaotic behavior (in those cases, we have pseudo-chaotic behavior). Since this is not the focus of this work, we will not explore this point further, but it is still important to keep it in mind.

We can define flow as the change in n-dimensional phase space volume and its rate of change turns out to be

$$V(t) = V_0 e^{(\lambda_1+\lambda_2+\cdots+\lambda_n)t}.$$

For a dissipative system, the change in volume is nonzero and a "flow-out"denotes a positive

change in volume (volume expansion). A “flow-in” denotes a negative change in volume (volume contraction) for dissipative systems. A conservative system would have a constant volume (no flow in or out).

Remark 2.1.20. There are two Lyapunov exponents for attractors of systems in two dimensions. If a transformation is defined in three dimensions, then there are three Lyapunov exponents $\lambda_1, \lambda_2, \lambda_3$. We can define them by requiring that

1. e^{λ_1} is the maximal average factor by which an error is amplified.

2. $e^{\lambda_1+\lambda_2}$ is the maximal average factor by which an area changes.

3. $e^{\lambda_1+\lambda_2+\lambda_3}$ is the maximal average factor by whxcih a volume changes.

The main idea behind calculating the Lyapunov exponent of a flow in a n-dimensional phase space is that in a chaotic system each vector tends to follow the direction of maximal growth. So, no matter how we start off, if we follow the trajectory long enough, we would end up in the direction of greatest change.

Another aspect of the computational procedure is that methods involving a small finite initial error ϵ cannot guarantee small enough time separations after a certain amount of iterations that we perform for the system to converge.

Therefore, we are going to define a tangent plane to our phase space, which is essentially the linearization of our phase space.

Definition 2.1.21. Suppose f has continuous partial derivatives. The **tangent plane** to the surface $z = f(x, y)$ at the points $p(x_0, y_0, z_0)$ is given by the equation

$$z - z_0 = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0). \tag{2.10}$$

The linear function whose graph is this tangent plane, namely

$$L(x, y) = f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

is called the **linearization** of f at (a, b) and the approximation

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

is called the **linear approximation** of f at (a, b). For three variables or higher, just add another variable term to the equation.

A **fiducial trajectory** is defined as the action of nonlinear differential equations on some initial conditions during the evolution of the system. The Jacobian of the system of differential equations defines a set of linearized equations that in turn define the tangent space to the phase space. Trajectories of the points on the surface of the sphere of a small volume we discussed above are defined by the action of the linearized equations on points infinitesimally small distance away from the fiducial trajectory. The **principle axes** are defined by the evolution via the linearized equations of an initially orthonormal vector frame attached to the fiducial trajectory. By definition, principle axes defined by the linear system are always infinitesimally small relative to the attractor. As the fiducial trajectory is iterated using the differential equations, the linearized equations iterate the set of orthonormal vectors. Due to the finite precision of numerical methods and calculations, and because vectors on the trajectory fall into direction of maximal growth, after some amount of time, all the vectors will converge in the direction of maximum growth. This can be avoided by repeatedly applying the Gram-Schimdt orthonormalization procedure. It is a standard linear algebra tool, and the idea behind it involves keeping one of the initially orthonormal vectors unchanged and constantly adjusting the rest to become pairwise orthonormal.

Definition 2.1.22. Vector projection of v onto u is defined by

$$\mathbf{proj_u}(\mathrm{v}) = \left(\frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}|}\right)\frac{\mathbf{u}}{|\mathbf{u}|} = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}|^2}\mathbf{u},$$

where $\mathbf{u}$ and $\mathbf{v}$ are vectors in $\mathbb{R}^n$ and $\mathbf{u}\cdot\mathbf{v}$ denotes the **inner product** (or **dot product**) of the vectors $\mathbf{u}$ and $\mathbf{v}$. This operator projects the vector $\mathbf{v}$ orthogonally onto the line spanned by vector $\mathbf{u}$.

Definition 2.1.23. Given k vectors $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}\}$ in a nonzero subspace W of $\mathbb{R}^n$, we can orthonormalize them using the **Gram-Schmidt Process** as follows. First we produce an orthogonal set of vectors $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k}\}$:

$\mathbf{u_1} = \mathbf{v_1}$,

$$\mathbf{u_2} = \mathbf{v_2} - \mathbf{proj_{u_1}}(\mathbf{v_2}),$$
$$\mathbf{u_3} = \mathbf{v_3} - \mathbf{proj_{u_1}}(\mathbf{v_3}) - \mathbf{proj_{u_2}}(\mathbf{v_3}),$$
$$\mathbf{u_4} = \mathbf{v_4} - \mathbf{proj_{u_1}}(\mathbf{v_4}) - \mathbf{proj_{u_2}}(\mathbf{v_4}) - \mathbf{proj_{u_3}}(\mathbf{v_4}),$$
$$\vdots$$
$$\mathbf{u_k} = \mathbf{v_k} - \sum_{i=1}^{k-1} \mathbf{proj_{u_i}}(\mathbf{v_k}).$$

Now $\{u_1, u_2, \ldots, u_k\}$ is an orthogonal basis for W. In addition, $\text{span}\{u_1, u_2, \ldots, u_p\} = \text{span}\{v_1, v_2, \ldots, v_p\}$ for $1 \leq p \leq k$.

Example 2.1.24. Apply the Gram–Schmidt Process on the vectors

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \text{and} v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

Then $\{v_1, v_2, v_3\}$ is clearly linearly independent and thus is a basis for a subspace W of $\mathbb{R}^4$ spanned by v_1, v_2, v_3. Construct an orthogonal basis for W.

Step 1. Let $u_1 = v_1$ and $W_1 = \text{Span } \{u_1\}$.

Step 2. Let u_2 be the vector produced by subtracting from v_2 its projection onto the subspace W_1. That is, let

$$\begin{aligned} u_2 &= v_2 - \mathbf{proj}_{w_1} v_2 \\ &= v_2 - \frac{v_2 \cdot u_1}{u_1 \cdot u_1} u_1 \\ &= \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \frac{3}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -3/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}. \end{aligned}$$

Here u_2 is the component of v_2 orthogonal to v_1, and $\{u_1, u_2\}$ is an orthogonal basis for the subspace W_2 spanned by v_1 and v_2.

Step 2′ (optional) If appropriate, scale u_2 to simplify later computations. Since u_2 has fractional entries, it is convenient to scale it by a factor of 4 and replace $\{u_1, u_2\}$ by the orthogonal basis

$$u_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, u_2' = \begin{bmatrix} -3 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Step 3 Let u_3 be the vector produced by subtracting from v_3 its projection onto the subspace W_2. We use the orthogonal basis $\{u_1, u_2'\}$ to compute this projection onto W_2:

$$\mathbf{proj}_{W_2} v_3 = \frac{v_3 \cdot u_1}{u_1 \cdot u_1} u_1 + \frac{v_3 \cdot u_2'}{u_2' \cdot u_2'} u_2' =$$

$$\frac{2}{4}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \frac{2}{12}\begin{bmatrix} -3 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2/3 \\ 2/3 \\ 2/3 \end{bmatrix}.$$

Then u_3 is the component of v_3 orthogonal to W_2, namely,

$$u_3 = v_3 - \mathbf{proj}_{W_2} v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 2/3 \\ 2/3 \\ 2/3 \end{bmatrix} = \begin{bmatrix} 0 \\ -2/3 \\ 1/3 \\ 1/3 \end{bmatrix}.$$

Now that we have a set of orthogonal vectors, we normalize them to have unit length:

$$\{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_3}\} = \left\{ \frac{\mathbf{u_1}}{||\mathbf{u_1}||}, \frac{\mathbf{u_2}}{||\mathbf{u_2}||}, \ldots, \frac{\mathbf{u_3}}{||\mathbf{u_3}||} \right\}$$

The calculation of the sequence $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k}\}$ is known as Gram-Schmidt orthogonalization, while the calculation of the sequence $\{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_k}\}$ is known as Gram-Schmidt orthonormalization as the vectors are normalized.

We see that Gram-Schmidt orthonormalization never affects the direction of the first vector, so this vector tends to seek out the direction in tangent space which is most rapidly growing (components along other directions are either growing less rapidly or shrinking). The second vector has its component along the direction of the first vector removed, and is then normalized. Because we are changing its direction, vector $\mathbf{v_2}$ is not free to seek out the most rapidly growing direction. The same holds for all the other vectors. Note however that the vectors $\{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_k}\}$ and $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}\}$ span the same k-dimensional subspace in $\mathbb{R}^n$. In spite of repeated vector replacements, the space these vectors define continually seeks out the k-dimensional subspace that is most rapidly growing. As we pointed out above, the volume defined by these vectors is proportional to $e^{(\lambda_1+\lambda_2+\ldots,\lambda_n)t}$.

If a system is given by an equation $\dot{\mathbf{x}} = F(\mathbf{x})$, where $\mathbf{x}$ is a function of time, then for two nearby trajectories the equivalent of our one-dimensional approximation in higher dimensions becomes

$$\dot{\mathbf{x}} - \dot{\mathbf{y}} = F(\mathbf{x}) - F(\mathbf{y}) \approx JF(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{y}),$$

and at a very small later time dt,

$$\mathbf{x}(t+dt) - \mathbf{y}(t+dt) =$$

$$= \mathbf{x}(t) - \mathbf{y}(t) + dtJF(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{y})$$

$$= \Big(I + dtJF(\mathbf{x})\Big) \cdot (\mathbf{x} - \mathbf{y}).$$

So the local expansion is measured by the quantity $\Big(I + dtJF(\mathbf{x})\Big)$ and the Lyapunov exponent associated with this expansion (or the associated eigenvalue) can be computed by the approximation (which is exact in the limit)

$$\lambda \approx \frac{1}{dt} \ln \left(\frac{\mathbf{x}(t+dt) - \mathbf{y}(t+dt)}{\mathbf{x} - \mathbf{y}} \right).$$

Recall [4, p.667]also that for the Lorenz system with

$$F(x,y,z) = \begin{bmatrix} 10(y-x) \\ x(28-z) - y \\ xy - 8z/3 \end{bmatrix},$$

the Jacobian is given

$$JF(x,y,z) = \begin{bmatrix} -10 & 10 & 0 \\ 28-z & -1 & -x \\ y & x & -8/3 \end{bmatrix}.$$

Below is a sample of the Mathematica routine that we made to compute the Lyapunov spectrum for the Lorenz system. We do a preliminary run of 1000 iterations to get on the attractor. Then in the main loop, we first get the new point on the trajectory, evaluate the quantity $I + dtJF[x_1]$, which gives us the expansion rate at this point, then we compute the expansion along each direction of the unit cube. Next, we re-orthogonalize, measure the divergence, and store the value with the λ_i, renormalize the cube, and move to the next point. At the end, we print out the Lyapunov exponents. Note that the results confirm the known values for the Lyapunov spectrum

```
(*defining the Lorenz equations*)
Timing[
 F[{x_, y_, z_}] := {10 (y - x), x*(28 - z) - y, x*y - 8/3*z};
 (*defining the RK5 algorithm, it is a mathematica module, which \
takes on a list of functions fcns_, each one is a function of a list \
of variables vars_, a list of initial values x0_, and a time step \
dt_, and integrates them all to give us the next value on the \
trajectory of the system of equations starting at x*)
 RK5[fcns_, vars_, x0_, dt_] :=
  Module[{k1, k2, k3, k4, k5, k6},
   (*thread means evaluate all functions by replacing the variables \
as the arrow shows*)
   k1 = dt* N[fcns /. Thread[vars -> x0]];
   k2 = dt *N[fcns /. Thread[vars -> x0 + k1/4]];
   k3 = dt* N[fcns /. Thread[vars -> x0 + (3*k1 + 9*k2)/32]];
   k4 = dt *
     N[fcns /.
       Thread[vars ->
         x0 + 1932/2197*k1 - 7200/2197*k2 + 7296/2197*k3]];
   k5 = dt *
     N[fcns /.
       Thread[vars ->
         x0 + 439/216*k1 - 8*k2 + 3680/513*k3 - 845/4104*k4]];
   k6 = dt *
     N[fcns /.
       Thread[vars ->
         x0 + 8/27*k1 + 2*k2 - 3544/2565*k3 + 1859/4104*k4 -
          11/40*k5]];
   x0 + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5 +
    2/55*k6];
 (*define the Jacobian matrix by using the Outer command which \
combined with th ederivative D command gives the derivative of each \
function by each variable in a list*)
```

```
 JacobianMatrix[fcns_, vars_] := Outer[D, fcns, vars];
 (*defining the Jacobian of F which we will need to evaluate along \
the trajectory*)
 JF[{x_, y_, z_}] := Evaluate[JacobianMatrix[F[{x, y, z}], {x, y, z}]];
 (*define the time step*)
  dt = 0.001;
 (*define the time interval along which to compute*)
  T = 100;
 (*define the unit ball vectors*)
 w1 = {1, 0, 0};
 w2 = {0, 1, 0};
 w3 = {0, 0, 1};
 (*initialize the Lyapunov exponent variables*)
 \[Lambda]1 = 0;
 \[Lambda]2 = 0;
 \[Lambda]3 = 0;]


Timing[
 (*we use a Random[] function which guarantees a random number between \
0 and 1 and we magnify it because the Lorenz attractor ranges between \
-50 and 50, roughly*)
 x0 = {10 Random[], 10 Random[], 30 Random[]}; Print[x0]
  (*Preliminary run to ensure we are on the attractor first before we \
begin measuring divergence*)
  Do[
   x1 = RK5[F[{x, y, z}], {x, y, z}, x0, dt]; x0 = x1, {i, 10000}];
 ]
{3.3990517003553107`, 0.2788704875394655`, 9.968737426916572`}
{2.828`, Null}
Print[x1]
{14.84224909492229`, 17.759372669125128`, 32.51993924789539`}
Timing[
 Do[
  (*compute next value on the trajectory*)
  x1 = RK5[F[{x, y, z}], {x, y, z}, x0, dt];
  (*computing the quantity I+dtKF,
  which gives us the expansion quantity for the unit ball*)
  IJF = IdentityMatrix[3] + dt*JF[x1];
  (*calculate the image of the unit ball under IJF*)
  w1 = IJF.w1;
  w2 = IJF.w2;
  w3 = IJF.w3;
  (*turn into an orthogonal set, Gram-Schmidt1*)
  w1 = w1;
  w2 = w2 - Projection[w2, w1];
  w3 = w3 - Projection[w3, w1] - Projection[w3, w2];
  (*calculate the divergence along each column*)
  \[Lambda]1 = \[Lambda]1 + Log[Norm[w1]];
  \[Lambda]2 = \[Lambda]2 + Log[Norm[w2]];
  \[Lambda]3 = \[Lambda]3 + Log[Norm[w3]];
  (*renormalize back to unit vectors, Gram-Schmidt2*)
  w1 = w1/Norm[w1];
  w2 = w2/Norm[w2];
  w3 = w3/Norm[w3];
  (*reinitialize*)
  x0 = x1, {i, IntegerPart[T/dt]}];
```

```
 (*print out the Lyapunov exponents*)
 Print[{N[\[Lambda]1/T], N[\[Lambda]2/T], N[\[Lambda]3/T]}]
 ]
{0.9759626264403718‘, 0.029996276609986605‘, -14.711565071410973‘}
{65.90599999999999‘, Null}
```

We also compared our routine to a more sophisticated one that we created based on the theoretical outline in [10, p.80], and improved by using RK5 instead of RK4 algorithm. That routine considers an initial point $\mathbf{x_0}$ and an initial error vector $\mathbf{u_0}$ which defines a "cube" U_0 with edges specified by its components in $\mathbb{R}^n$. Then after time t, we have our system run along the fiducial trajectory $F_t(\mathbf{x_0})$ and the cube has become $JF_t(\mathbf{x_0}(U_0))$. It is beyond the scope of this project to discuss the details of using the fiducial trajectory at a given point in time to integrate the tangent vectors in order to measure the volume of the "cube". Essentially, the tangent vector $\mathbf{u_t}$ evolves according to a **variational equation** [10] which is a linear system of time-varying differential equations whose coefficients come from the fiducial trajectory, and which consider the tangent vector along each trajectory as they vary with respect to time and the trajectory.

Our algorithm for the Lyapunov spectrum of the Lorenz system produced a positive Lyapunov exponent of 0.97563. This confirms that the Lorenz system is sensitive to initial conditions and is chaotic. It also confirms that our algorithm works correctly. Another, even more recent research result has produced an algorithm for the largest Lyapunov exponent and the Lyapunov spectrum [8], and we used it to confirm our results.

Our Lyapunov routines can be used in the study of other dynamical systems. This is one of our future goals.

2.2 Self-similarity Dimension

Our goal in this section is to become familiar with the fractals (self–similar) in order to begin to understand the concept of fractal dimension. In subsequent section, we will use the dimension calculation methods of box-counting, information dimension, and correlation dimension, and compare their results in order to answer the question of which method offers the best results.

Definition 2.2.1. Fractals are geometric shapes with fine structure at arbitrarily small scales. Usually they have some degree of self–similarity. For example, if we magnify a tiny part of a fractal, we will see features reminiscent of the whole. The self–similarity can be exact; however, more often it is only slightly similar.

2.2.1 Similarity Dimension

Consider a square of unit side length. If we shrink the side length by a factor of 2, it takes 4 small squares to cover the original square. If we scale the unit length side by a factor of 3, it takes 9 small squares to cover the original square. It is easy to show that scaling the square side by a factor of r, it takes r^2 smaller squares, each with side length $1/r$, to cover the original square. So m denotes the number of smaller squares that are needed to cover the original square, and r is the scaling factor.

Definition 2.2.2. Suppose that a self–similarity set is composed of m copies of itself scaled down by a factor of r. Then the similarity dimension d is the exponent defined by

$$m = r^d,$$

equivalently,

$$d = \frac{\ln m}{\ln r}, \tag{2.11}$$

For a square, which is a piece of the plane, the similarity dimension is clearly 2.

In order to carry out similarity dimension calculations, one needs to have a precise idea of how the fractal is structured and what its basic building blocks look like. In the cases of the Cantor set, and the von Koch curve this is fairly easy to do.

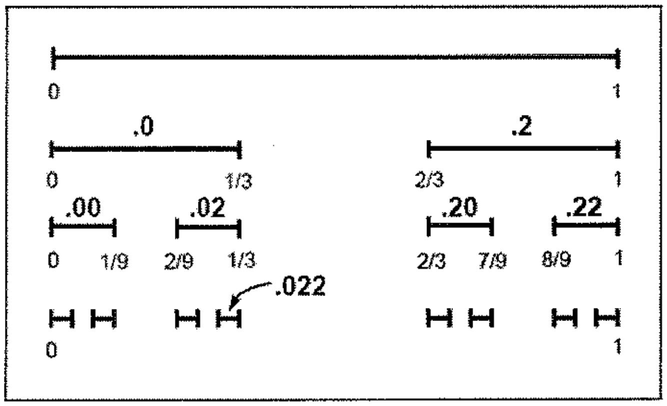

Figure 2.2: The Cantor set.(Reproduced from [3].

Example 2.2.3. The Cantor set is generated by the iteration of a single operation on a line of unit length. With each iteration, the middle third from each line segment of the previous set is removed. See figure 2.2.

As the number of iterations increases, the number of separate line segments tends to infinity while the length of each line segment approaches zero. Under magnification its structure is essentially indistinguishable from the whole, which makes it self-similar. After the first scaling, the left half of the Cantor set is a scaled down copy of the Cantor set, with a scaling factor of 3. So for the Cantor set, $m = 2$ since we have a total of 2 copies of the set, and $r = 3$ for the scaling factor. Therefore, for the Cantor set we have

$$d = \frac{\ln 2}{\ln 3} \tag{2.12}$$

Example 2.2.4. The Von Koch curve is made up of a straight line from which the middle third is removed and replaced by the two sides of an equilateral triangle. Every step produces four equal pieces, each of which is similar to the original curve but scaled down by a factor of 3 in both directions. See figure 2.3.

So $m = 4$ since we have a total of 4 copies of the Von Koch curve, and $r = 3$ for the scaling factor. Therefore, for the Von Koch curve we have

$$d = \frac{\ln 4}{\ln 3} \approx 1.26 \tag{2.13}$$

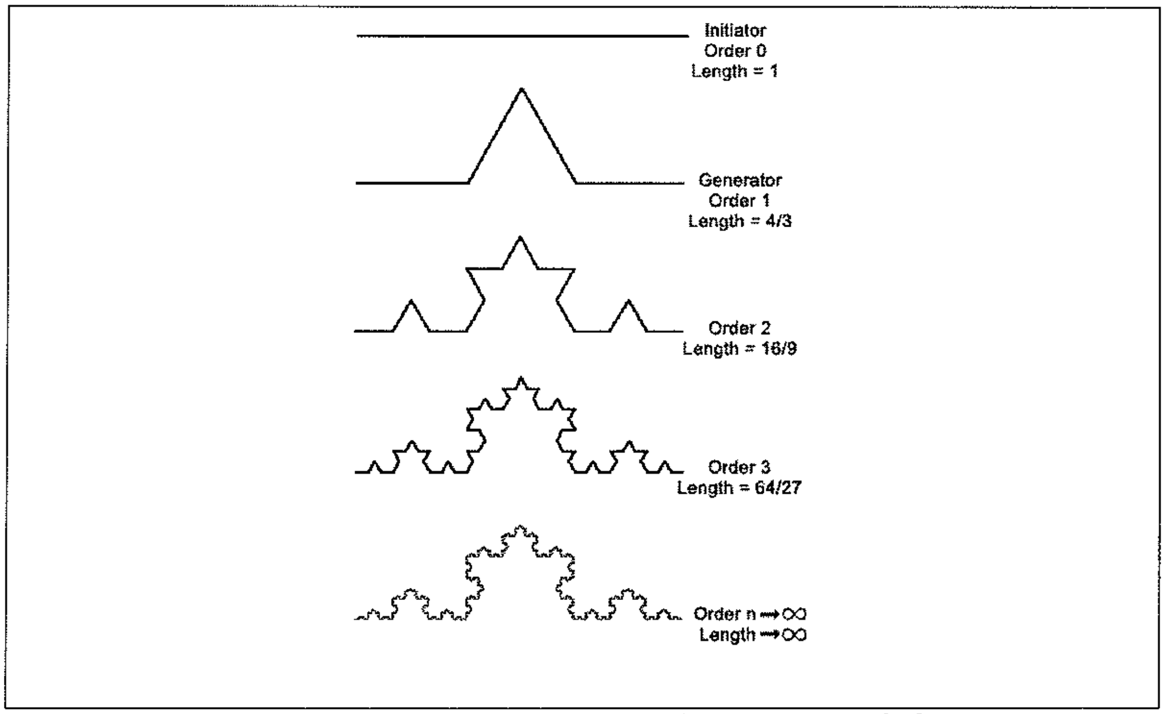

Figure 2.3: The Von Koch curve. (reproduce from [14]

2.3 Box-counting Dimension

The next type of dimension concept we will consider is the box-counting dimension. This is a general notion of the self-similarity dimension, as it is able to deal with non-self-similar fractal structures. Let S be a subset of n-dimensional Euclidean space $\mathbb{R}^n$, and let $N(\epsilon)$ be the minimum number of n-dimensional boxes/cubes of side–length ϵ needed to cover S. To get an idea of the relationship between ϵ and $N(\epsilon)$, consider that for a smooth curve of length L, $N(\epsilon) \propto L/\epsilon$. For a planar region of area A bounded by a smooth curve, $N(\epsilon) \propto L/\epsilon^2$. The key observation that we arrive at is that the dimension of the set equals the exponent d in the power law

$$N(\epsilon) \propto L/\epsilon^d.$$

For fractal sets, this law holds; however, d takes on non-integer values.

Definition 2.3.1. A bounded set S in $\mathbb{R}^n$ has **box-counting dimension**

$$d = \lim_{\epsilon \to 0} \frac{\ln N(\epsilon)}{\ln(1/\epsilon)}. \tag{2.14}$$

when the limit exist.

Example 2.3.2. The box-counting dimension of the Cantor set is calculated as follows. Recall that the Cantor set is covered by each of the sets S_n, the intervals that are left after removing certain pieces of the unit length interval S_0 at each subsequent step. Note that the Cantor set C is really the limit of this process! Each S_n consists of 2^n intervals of length $(1/3)^n$, so select $\epsilon = (1/3)^n$, then let $N = 2^n$. Since $\epsilon \to 0$ when $n \to \infty$, we have

$$d = \lim_{\epsilon \to 0} \frac{\ln N(\epsilon)}{\ln(1/\epsilon)} == \lim_{\epsilon \to 0} \frac{\ln(2^n)}{\ln(3^n)} = \frac{n \ln 2}{n \ln 3} = \frac{\ln 2}{\ln 3}$$

equal to the self-similarity dimension of the Cantor set. In general, self-similarity and box-counting dimensions have different values.

Remark 2.3.3. This calculation illustrates an important point. The discrete sequence for ϵ can be used to evaluate the limit as $n \to \infty$ even though the definition requires a continuous limit evaluation. So even if ϵ takes on some values where the number of boxes covering the structure $N(\epsilon)$ varies, or even if the boxes are rectangular (as we shall see later), in the limit, the estimation will be exact [14, p.410].

Example 2.3.4. Suppose for the equation $F(x, y) = (y + 1 - ax^2, bx)$ where $a = 1.4$, $b = 0.3$. Allow x and y such that $f(x, y)$ is contained in the square area of $-2 \leq x \leq 2$, and $-2 \leq y \leq 2$. Next, we superimpose a grid of boxes with side length of $\epsilon = \frac{1}{4}$. Furthermore, suppose there are a total of 256 boxes that make up our grid, and of those 256 boxes, 76 contain at least one piece of the attractor. A visual representation of the attractor before and after the grid is superimposed is given in figure 2.4.

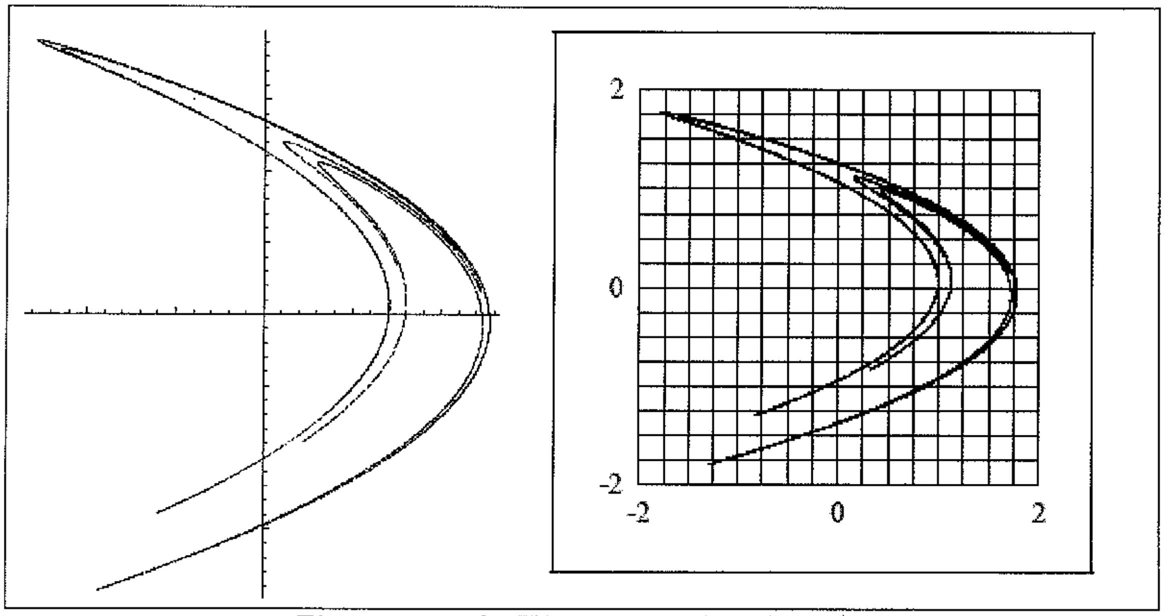

Figure 2.4: The Hénon map with grid of 1/4.

We compute the box-counting dimension as follows

$$d = \frac{\ln N(\epsilon)}{\ln(1/\epsilon)} = \frac{\ln(76)}{\ln\left(\dfrac{1}{1/4}\right)} = \frac{\ln(76)}{\ln(4)} = 3.124$$

This is not the full and accurate definition of the box-counting dimension of the attractor. The number 3.124 only represents a value for that particular size of box. To get the full representation of the box-counting dimension, we have to take a range of different box sizes. After we obtain the values for the different box sizes, we then graph the quantities in a $\log_2 N(\epsilon)$ versus $\log_2(1/\epsilon)$ graph. We can do this because the quantity $\log_2 N(\epsilon)$ versus $\log_2(1/\epsilon)$ has the same ratio as $\frac{\ln N(\epsilon)}{\ln(1/\epsilon)}$ which defines the box-counting dimension in the limit as $\epsilon \to 0$. Thus, the box-counting dimension corresponds to the slope in the graph. We will go into more detail in the next example.

Example 2.3.5. We will compute the box-counting dimension for the Hénon map produced by equation 1.6. We used 10,000,000 points and scaled the box size as a power of 2, starting at 2^{-1} and going through to 2^{-20}. Table 2.2 presents the results of the number of boxes that have been found to contain points of the map.

Box size	Number of boxes	Box size	Number of boxes
2^{-1}	4	2^{-11}	29,185
2^{-2}	10	2^{-12}	69,416
2^{-3}	28	2^{-13}	170,789
2^{-4}	70	2^{-14}	398,134
2^{-5}	170	2^{-15}	881,508
2^{-6}	396	2^{-16}	1,683,903
2^{-7}	918	2^{-17}	3,048,299
2^{-8}	2,181	2^{-18}	4,667,559
2^{-9}	5,050	2^{-19}	6,582,188
2^{-10}	12,059	2^{-20}	8,556,845

Table 2.2: Box size and corresponding box count with elements of the Hénon map in them. Produce with 10,000,000 points on the trajectory.

In order to compute the box-counting dimension, we plot the natural logarithm of the number of boxes, $N(\epsilon)$,versus the natural logarithm of the reciprocal of the box size, $\ln(1/\epsilon)$, and we look for a linear fit (see figure 2.5). The slope of the linear fit is the value of the scaling constant, which is what the box-counting dimension essentially represents. The slope in the linear fit on the left (a) is 1.12867, which is about a 12% deviation from the estimate of the slope on the right, (b), where the last 5 points have been disregarded.

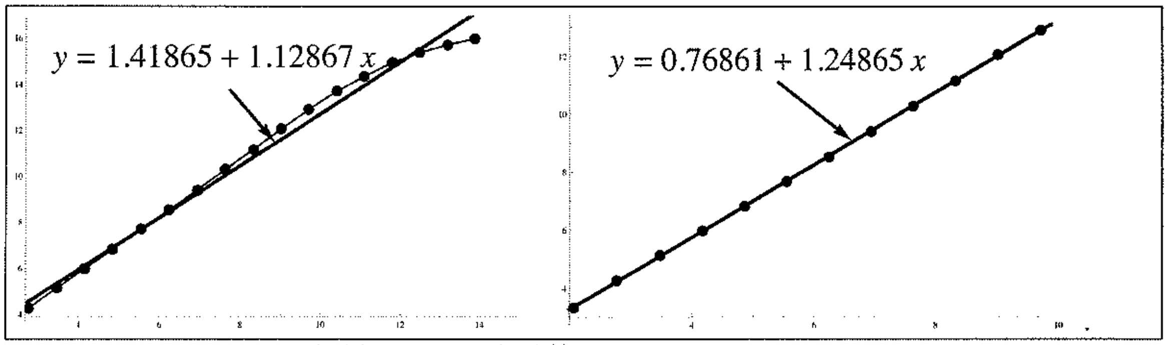

Figure 2.5: Plot of $\ln(1/\epsilon)$ versus $\ln(N(\epsilon))$ for the Hénon map with 10,000,000 points, a linear fit to part of the data (right), and another fit to the full data set(left)

Remark 2.3.6. The slope of the linear fit representing the box-counting dimension depends on the number n of points on the attractor.

Note in figures 2.5, 2.6, and 2.7 for the complete data fit model how the data points flatten off for small values of ϵ, just where the dimension, or the scaling coefficient, could be calculated

relatively precisely. This is the effect of having a limited number of points, n, the number of iterations of the system of equations. So as soon as we scale down to where the boxes are small enough to contain individual points, i.e., the box size is on the order of the average distance between points on the attractor, we lose our ability to measure scaling. After this point, every time we scale down, reducing the box size, there is no more structure to uncover. That is, if a given box contains only a single point, making it smaller will have no effect because there is only that one point in the box.

Lemma 2.3.7. *There is a definite small size $\epsilon^\star$ such that any box count $N(\epsilon)$ with $\epsilon \leq \epsilon^\star$ will yield the same number of boxes counted.*

The box-counting dimension formula is not concerned with how many points are inside individual boxes, only that there is at least one point inside of a box in order to count that box for the dimension computation. Therefore, we uncover important structure information by scaling the box size down; however, the scaling is limited by the number of iterations that the system has taken. This is a weakness of the box-counting formula.

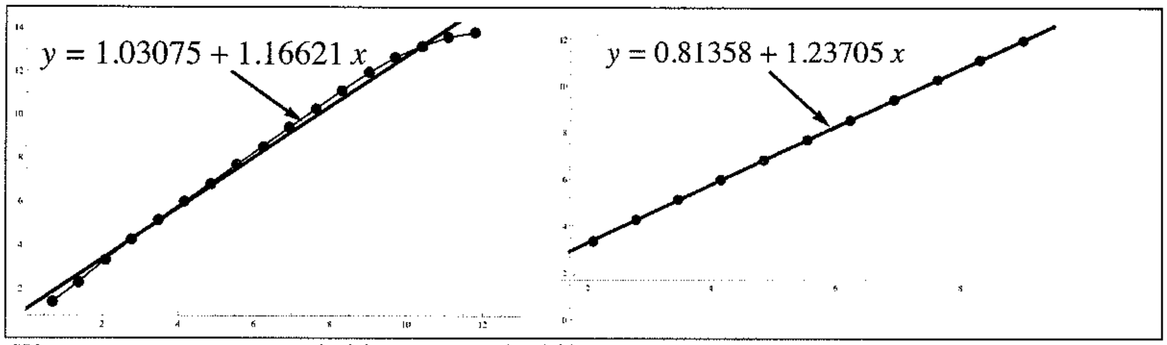

Figure 2.6: Plot of $\ln(1/\epsilon)$ versus $\ln(N(\epsilon))$ for the Hénon map with 1,000,000 points, a linear to part of the data is on the right, and another fit to the full data set is on the left for comparison.

The box-counting limitation phenomenon is not particular only to the Hénon attractor (see figures 2.5,2.6) or the Lozi map (see figure 2.7). We carried out similar analysis for all of our examples from section 1.4, and we summarized our findings in table 2.3.

To be more precise, the numbers in table 2.3 have been computed using rectangular boxes. The value $\epsilon = 1/128$, for example, means that we have subdivided the rectangular region S

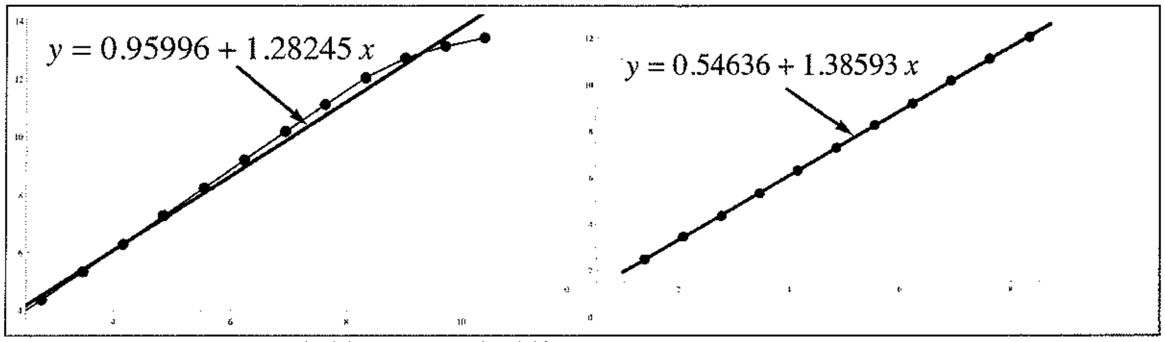

Figure 2.7: Plot of $\ln(1/\epsilon)$ versus $\ln(N(\epsilon))$ for the Lozi map with 1,000,000 points, a linear fit to all of the data (a) is on the left, and another fit to part of the data set (b) is on the right.

System name	Box-counting dimension (complete data fit)	Box-counting dimension (partial data fit)	Number of points n	Box size scaling
Hénon system	1.12867	1.24865	10,000,000	2^{-1} to 2^{-20}
Lozi system	1.28245	1.38593	1,000,000	2^{-1} to 2^{-15}
Ikeda system	1.56645	1.71449	1,000,000	2^{-1} to 2^{-15}
Sprott map A	1.42396*	1.56532*	3,000,000	2^{-3} to 2^{-16}
Sprott map B	1.55061*	1.71173*	3,000,000	2^{-3} to 2^{-15}
Sprott map C	1.41198*	1.56732*	3,000,000	2^{-3} to 2^{-16}
Sprott map D	1.57011*	1.81102*	3,000,000	2^{-3} to 2^{-14}
Ueda system	1.72807	1.87097	500,000	2^{-1} to 2^{-10}

Table 2.3: Box-counting dimensions of partial and complete data linear fit. (* indicates new results.)

containing the given map into rectangles with length and height equal to $1/128$ of S. This leads to exactly the same box-counting value in the limit, with generally minor deviations in the intermediate values. Furthermore, it makes the programming a bit simpler. The calculations in table 2.3 were carried out using different box sizes scaled as powers of 2, for example, from 2^{-1} to 2^{-16}. The Ueda map was generated using the RK5 method, with 500,000 points and a time step of $h = 0.001$.

Question 2. *How do we treat the box scaling limitation in order to get an accurate dimension calculation?*

We have three options, all of which fail to resolve the problem. If we ignore the flat part of the curve stemming from the boxes with very small box size, we will get a systematic error

because we are using only the very large boxes (see figure 2.5, and figure 2.6). Fitting the line only to the dated for the smaller boxes yields a slope of about zero, implying the measured fractal dimension is also 0. However, this is the correct box-counting dimension only for the collection of points we have generated and not for the infinite number of points that form an attractor. Thus, for any number we obtain for a dimension, we do not know whether we are really measuring the dimension of the attractor or just seeing an artifact of the limitations of the method.

Table 2.4 examines this dependence. $N(\epsilon, n)$ is the number of boxes that contain one or more points of the Hénon attractor computed for a number of n iterations. At box size $\epsilon = 2^{-3}$, we get 26 boxes after 100 iterations, only two more boxes in the next 900 iterations, and no more in the next 10 million iterations! At $\epsilon = 2^{-6}$ (which is not yet a small enough value for a serious study), we find 394 boxes in the first 100,000 iterations and only two more boxes in the next 9,900,000 iterations. At $\epsilon = 2^{-12}$, however, we find more than 60,000 new boxes between 10,000 and 10,000,000 iterations!

Iterations n	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-3}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-4}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-5}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-6}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-7}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-8}}$	$\mathbf{N(\epsilon, n)}$ $\epsilon = \mathbf{2^{-12}}$
100	26	42	60	77	90	95	100
1,000	28	61	148	304	516	721	990
10,000	28	69	166	380	853	1,879	8,898
100,000	28	69	169	394	911	2,128	43,716
1,000,000	28	69	169	395	915	2,175	66,253
10,000,000	28	70	170	396	918	2,181	69,416

Table 2.4: Dependence of the box count $N(\epsilon, n)$ on the box size ϵ and the number of points (iterations) n for the Hénon attractor.

Remark 2.3.8. It seems that to make the box-counting algorithm work, we have to adapt the number of iterations to the limitations of the box size. The smaller the size, the more iterations we should perform to measure the total number of boxes.

How many iterations are enough for a particular box size? Assume that after 10,000,000 iterations of box size 2^{-12} we find a box not previously visited. Is it the last one we will find if we continue to iterate? There is no way to tell. We could propose to perform another 100,000

iterations, and if we do not find any more new boxes to add to our count, we would stop the iteration. But the next iteration may be a hit, and we would have missed it. We conclude that we are not able to count $N(\epsilon)$ directly, and our count $N(\epsilon, n)$ will always depend on the number of iterations performed.

The numbers in table 2.4 confirm our observation about the box-counting dimension method. The picture becomes worse when we improve the resolution, that is, consider even smaller box sizes. How many more iterations should we perform in order to reach an accurate representation of the box-counting dimension? In the case of the Hénon attractor, iterations are very easy to carry out, but for many other systems one million iterations may be well above any reasonable computational time frame. For the Hénon transformation, even 100 million iterations are probably insufficient when we require results for even smaller resolutions. Thus, this is a fundamental problem of the box-counting method. This problem has been the subject of a detailed research study ([12]).

2.3.1 Grassberger's Method

Question 3. *Can we estimate how the number of boxes with points in them $N(\epsilon, n)$ depends on the number of iterations n?*

If we know the dependence of $N(\epsilon, n)$ up to some precision, then we can extrapolate from our count $N(\epsilon, n)$ to arrive at an estimate for

$$N(\epsilon) = \lim_{n \to \infty} N(\epsilon, n).$$

which is needed for the dimension calculation.

Recall that $N(\epsilon)$ is the minimum number of n-dimensional boxes of side-length ϵ needed to cover a given set. In essence we are looking for the optimum number of iterations necessary for a given box size ϵ so that $N(\epsilon)$ functions as a true scaler constant of the dimension instead of a product of decreasing box size. These issues were addressed in a 1983 paper by Peter Grassberger ([13]). His data suggested a general behavior

$$N(\epsilon, n) \approx N(\epsilon) - A\epsilon^{-\alpha} n^{-\beta}, \tag{2.15}$$

Where A is a constant, and n is very large. Based on this formula, we obtain the approximation

$$\frac{\triangle N(\epsilon,n)}{\triangle n} \propto \epsilon^{-\alpha} n^{-\beta-1}.$$

In table 2.4, given the values of $N(\epsilon,n)$ for the different box sizes ϵ and iteration counts n, we can compute the growth rates of $N(\epsilon,n)$ as n increases. We do this by plotting the data in a graph of ln − ln of $\dfrac{\triangle N(\epsilon,n)}{\triangle n}$ versus ϵ and n. A straight line in the plots would support the conjecture, and the exponents α and β can be obtained from the corresponding slopes. The exponent β measures the convergence rate towards the asymptotic value, while α shows how the under estimation error grows as $\epsilon \to 0$.

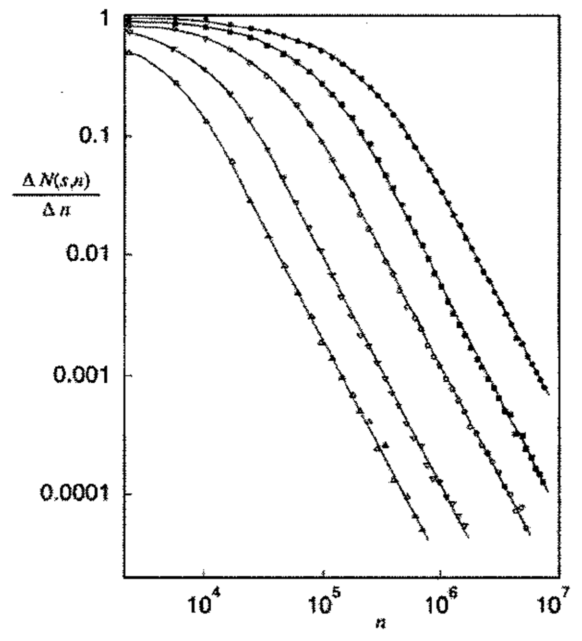

Figure 2.8: Figure adapted from P. Grassberger, *On the Fractal dimension of the Hénon attractor*. Physics letters 97A (1983) pp.224-26. Change of $\triangle N(\epsilon,n)/\triangle n$ average over five different box sizes. Clearly a line fit is appropriate for large n verifying the power law conjecture in Grassberger's work ([12]).

Grassberger performed such an analysis with five runs (each for a different value of ϵ) of 7.5 million iterations each and using five different box sizes ([12]). His findings showed clear linear behavior for large values of n, and his estimate for the exponents is

$$\alpha = 2.42 \pm 0.15, \ \beta = 0.89 \pm 0.03.$$

We also used the data we generated in table 2.4 to reproduce the behavior in the plots displayed in Grassberger's plot using our data ([12]). We are able to observe a linear fit for large values of n, as you can see in figure 2.9, where each curve represents calculations for a different box size. The plot is a ln-ln graph of $\triangle N(\epsilon, n)/\triangle n$ versus ϵ and n, where each axis uses a logarithmic scale. Eliminating initial points and looking at the linear fit to the data, the exponents α and β are obtained from the corresponding slopes. For α, we got 2.513 and for β we obtain a value of 0.886, which is well within the error range that Grassberger obtained ([12])!

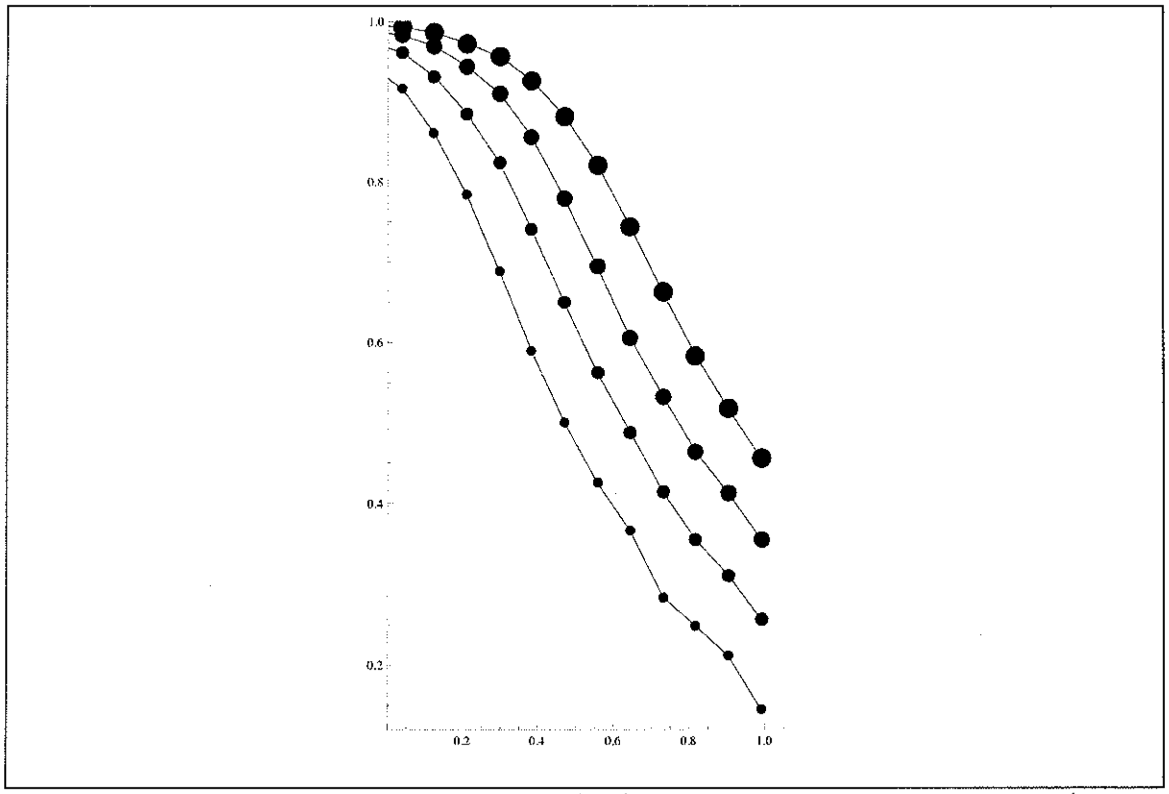

Figure 2.9: Dependence of the box count $N(\epsilon, n)$ on the box size ϵ and iterations n (Hénon attractor). Each plot line corresponds to a different box size, $\epsilon = 2^{-1}$, 2^{-11}, 2^{-12}, 2^{-13}, respectively, displayed with increasing point size. This is a ln-ln scale graph of the $\dfrac{\triangle N(\epsilon, n)}{\triangle n}$ versus n.

We can now compute an estimate for $N(\epsilon)$ using only two measurements, for example,

$N(\epsilon_1, n_1)$ and $N(\epsilon_1, 2n_1)$ as follows ([13]). Let

$$N(\epsilon_1, n_1) = N(\epsilon_1) - A\epsilon_1^{-\alpha} n_1^{-\beta}$$

$$N(\epsilon_1, 2n_1) = N(\epsilon_1) - 2A\epsilon_1^{-\alpha} n_1^{-\beta}$$

Solving these equations for A,

$$A = \frac{N(\epsilon_1, 2n_1) - N(\epsilon_1, n_1)}{(1 - 2^{-\beta})\epsilon_1^{-\alpha} n_1^{-\beta}}$$

we obtain the equation

$$N(\epsilon) = N(\epsilon, n) + A\epsilon^{-\alpha} n^{-\beta}, \tag{2.16}$$

where α, β, A are known.

This allows us to compute $N(\epsilon)$ from a given value for $N(\epsilon, n)$ with some specific range of value of $N(\epsilon)$ due to the range for the constants α and β. From that, we can compute the box-counting dimension D as follows. Given another constant of proportionality K, we obtain

$$N(\epsilon) = K\epsilon^{-D}$$

$$N(2\epsilon) = K2^{-D}\epsilon^{-D},$$

so,

$$\frac{N(\epsilon)}{N(2\epsilon)} = 2^D.$$

Therefore,

$$D = \frac{\ln N(\epsilon) - \ln N(2\epsilon)}{\ln 2}. \tag{2.17}$$

We create a list of such values in order to estimate the value of D when $\epsilon \to 0$.

These estimate values for α and β and the measurements for $N(\epsilon, n)$ are sufficient to first compute $N(\epsilon)$ and then to calculate the box-counting dimension D for these values. Grassberger reported ([12]) the number

$$D = 1.28 \pm 0.01$$

for the dimension of the Hénon attractor.

The above algorithm is very complex and computationally demanding, even in the relatively simple two-dimensional cases considered here. Most attractors arise from more complex systems where even the mere computation of trajectories requires extensive numerical approximation techniques, as in the case of the Ueda map with the RK5 algorithm. Moreover, in other cases the dimension may be much larger, of value five or more, which implies that a much more significant computational endeavor is necessary. So the box-counting dimension is useful, but there are too many problems with its computation for the numerical results to be dependable. A lot of care is required in its computation. We will consider variations of the box-counting dimension.

We carried out similar analysis to Grassberger for other two-dimensional maps, but limited computational resources prevent us from generating the same data as in the Hénon case. Our partial data does confirm the values for α and β.

Overall, we discredit the box-counting technique because of its limitations and flimsy results. Furthermore, while Grassberger's method produces accurate results, the method is too time consuming, and is subject to numerous computational errors due to its complexity. Thus, Grassberger's method does not provide a useful computational procedure for the box-counting dimension. We will continue to seek better results from other dimension calculations methods.

There is another shortcoming of the box-counting dimension algorithm. For an orbit which densely fills up the attractor, we notice that it ends up within some parts of the attractor much more frequently than others. For example, we have 170,789 boxes of box size $1/2^{13}$ that cover the Hénon attractor for 10,000,000 iterations. But this number is not evenly distributed along the iterations, that is, the last few million iterations hit only a few of these boxes, and any new boxes visited during those counts have a much lower "density" than the boxes hit early on. The box-counting dimension ignores such differences. If a box is visited, it is counted only once no matter how many million more times the trajectory passes through that box. In other words, the box-counting dimension does not reflect the distribution of points from a trajectory on the attractor. One way to address this problem is to assign a certain weight to each box accounting for how many times the box is visited by the trajectory. This will be discussed in the section 2.4.

2.4 Natural Measure and Information Dimension

In the definition below, we will introduce a generalized concept which essentially assigns weights to different boxes covering an attractor based on how frequently a box is visited by the trajectory. It is the idea of a **measure**.

2.4.1 Natural Measure

Let B be an open subset of a space X which contains the attractor we want to study. The trajectory we generate seems eventually to visually cover the subset B densely. We can count the number of times a trajectory $x_0, x_1, \ldots, x_n$ in X enters the subset B, and it makes sense to assume that the percentage of all points which are in B converges to a fixed value as we perform more and more iterations [4, p.676]. That is, the fraction of the trajectory that is contained in B is a fixed number in the limit as the system is iterated infinitely along the trajectory. This percentage gives the natural measure for the system.

Definition 2.4.1. For a bounded subset B_k of $\mathbb{R}^n$, the **natural measure** $\mu(B_k)$ is given by

$$\mu(B_k) = \lim_{n\to\infty} \frac{1}{n+1} \sum_{k=0}^{n} I_B(x_k), \tag{2.18}$$

where $I_B(x_k)$ is 0 or 1 depending on whether $x_k \in B$ or $x_k \notin B$:

$$I_B(x_k) = \begin{cases} 1, & \text{if } \ x_k \in B \\ 0, & \text{otherwise} \end{cases}$$

and

$$\sum_{k=0}^{n} I_B(x_k)$$

is the number of points from the (finite) orbit $x_0, x_1, \ldots, x_n$ that belong to the set B_k.

The natural measure can also be defined for continuous dynamical systems. In this case the measure is the relative time that a trajectory $x(t)$ spends in the region B:

$$\mu(B) = \lim_{t\to\infty} \frac{1}{t} \int_0^t I_B(x(s))ds.$$

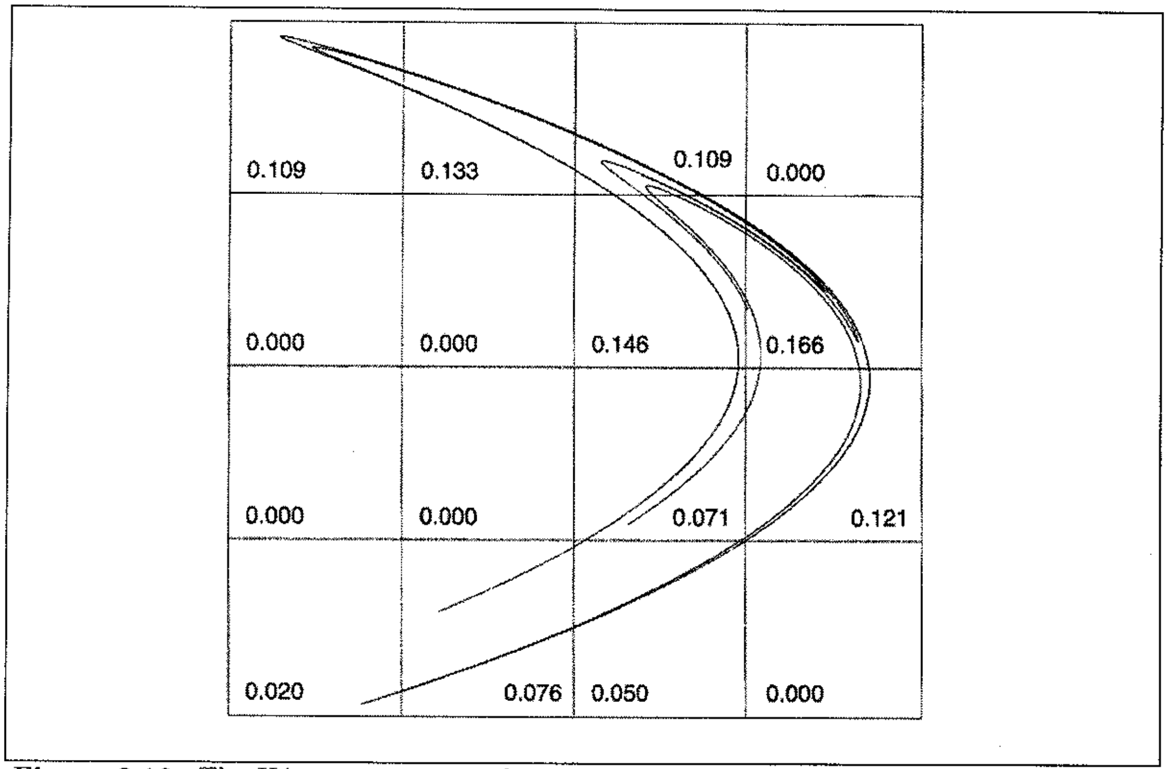

Figure 2.10: The Hénon attractor with 10,000,000 iteration points and subdivided into 16 boxes. The Mathematica routine we devised essentially keeps track of how many times a given box B_k is visited. (the figure reprinted from reference [7].)

Remark 2.4.2. The natural measure is an attempt to formulate the concept of weight for a portion of the attractor. Suppose that for the Hénon attractor, produced by equation 1.6, that as the trajectory is iterated we place a stone at every point of the orbit. In some regions of the attractor the stones will accumulate faster than in other places. The weight of the stones collected in the subset B is approximated by the product of the fraction of total stone mass that B contains (that is, $\mu(B)$) and the total weight of the distributed stones. While following the orbit and distributing more and more stones, this approximation becomes more and more exact. In figure 2.10 the mathematica routine used essentially keeps track of how many times a given box B_k is visited, the quantity $\sum_{k=0}^{n} I_B(x_k)$, and then divides it by 10,000,001 in order to obtain $\mu(B_k)$. Our calculations reproduce the numbers in the figure exactly.

Question 4. *What does the percentage values of the boxes covering the attractor tell us about the dimension?*

The natural measure captures the long-term statistics of the iteration process, and it can be used to define a new kind of fractal dimension. The approach is to replace the simple box-counting algorithm by a counting procedure in which each box is weighted according to its natural measure. So regions that are visited frequently by the trajectory have a larger contribution to the dimension value than the ones which the trajectory rarely visits.

2.4.2 Information Dimension

Definition 2.4.3. We replace $\ln(N(\epsilon))$ in the box-counting algorithm by

$$I(\epsilon) = \sum_{k=1}^{N(\epsilon)} \mu(B_k) \log_2 \frac{1}{\mu(B_k)}. \tag{2.19}$$

Here the sum ranges over the number $N(\epsilon)$ boxes B_k of size ϵ that cover the attractor. The quantity $I(\epsilon)$ gives the amount of information necessary to specify a point of the attractor to within an accuracy of ϵ (or to within an error range of ϵ). As the box size decreases, the information must increase.

Definition 2.4.4. Then the **information dimension** D_I is defined by

$$I(\epsilon) \approx I_o + D_I \log_2(1/\epsilon), \tag{2.20}$$

where I_o is a constant and D_I is the slope in the plot $I(\epsilon)$ versus $\log_2(1/\epsilon)$.

We reproduce Grassberger's table but we only do it with 1,000,000 iteration points as the 10,000,000 point calculation requires computation time that is beyond reasonable scale. See table 2.5.

Example 2.4.5. We give an example of the formula as applied to the natural measure of a strange attractor and how it becomes the information dimension. In Figure 2.10, the subdivision of the Hénon attractor into 16 boxes is shown. We have computed a long iteration sequence and collected its statistics resulting in the probabilities attached to the boxes as shown. In this figure, the information that a point is in the upper left box has a value of

$\log_2 1/0.109 \approx 3.198$bits. The average amount of information per measurement is

$$\begin{aligned}
&0.109\log_2 1/0.109 + 0.133\log_2 1/0.133+\\
&0.109\log_2 1/0.109 + 0.146\log_2 1/0.146+\\
&0.166\log_2 1/0.166 + 0.071\log_2 1/0.071+\\
&0.121\log_2 1/0.121 + 0.020\log_2 1/0.020+\\
&0.076\log_2 1/0.076 + 0.050\log_2 1/0.050 = 3.168
\end{aligned} \tag{2.21}$$

This result is the same as the second iteration for the number $I(\epsilon)$ in table 2.5 below.

This is only one step in calculating the information dimension. From table 2.5, we plot the Information $I(\epsilon)$ versus the logarithmic inverse scale $log_2(1/\epsilon_2)$ and a linear fit to the data is made. The slope of the linear fit is the value of the information dimension (see figure 2.11).

Iteration	Scale ϵ_2	No. of Boxes (ϵ)	I(ϵ)	D$_\text{I}$
1	1/2	5	1.83907	
2	1/4	11	3.16783	1.32876
3	1/8	29	4.47989	1.31206
4	1/16	70	5.62322	1.144334
5	1/32	170	6.87692	1.25369
6	1/64	396	8.1444	1.26748
7	1/128	916	9.36906	1.22466
8	1/256	2176	10.57832	1.20927
9	1/512	5029	11.79271	1.21437
10	1/1024	11927	13.0298	1.23707

Table 2.5: Using 1,000,000 iterations and grids of up to 1024 by 1024 boxes covering the region. We compute the number of nonempty boxes and the information. The last column contains estimates for the information dimension. The entry in row k of the last column is computed by the difference $I(\epsilon) - I(\epsilon_{-1})$.

Our Mathematica routine for calculating the information dimension of the Hénon map is given below.

```
Timing[
 OpenWrite["LHI"];
 D1x = ReadList["Henonx"];(*read in the Henon data*)
 D1y = ReadList["Henony"];
 Do[
  ex = 3/2^i;(*set limits for the box covering area,
  the attractor area, the x range*)
  ey = 0.8/2^i;(*set the y-range, we eyeballthese*)
```

```
  BOXLIST = {{Quotient[(D1x[[1]] + 1.5), ex],
     Quotient[(D1y[[1]] + 1.5), ey]}};(*keep track of the boxes*)
  BOXMEASURE = {{Quotient[(D1x[[1]] + 1.5), ex],
     Quotient[(D1y[[1]] + 1.5), ey],
     1}};(*keep track of how many times each box is visited,
  its measure*)
  Do[
   t1 = Quotient[(D1x[[m]] + 1.5),
     ex];(*identify each box by its palace in the rectangular area*)
   t2 = Quotient[(D1y[[m]] + 0.4), ey];
   If[(*and if-then-else statement, the IF PART FOLLOWS*)
    FreeQ[
     BOXLIST, {t1,
      t2}],(*test if the box is already on the list of boxes that \
contain elements of Henon,
    FreeQ tests if the lsit is "free" of the box, so if it is,
    this is a new box, so we need to add it to the list*)
    BOXLIST = Append[BOXLIST, {t1, t2}];(*THEN part:
    add the box to the list*)
    BOXMEASURE =
     Append[BOXMEASURE, {t1, t2, 1}],(*add the measure count*)
    (*ELSE part, in the case when the IF part is false, do this*)
    p = Position[BOXLIST, {t1, t2}][[1,
      1]];(*tells us where in the box we are looking at is located in \
the box list, in the case it is not a new box*)
    BOXMEASURE[[p, 3]] = BOXMEASURE[[p, 3]] + 1],(*
   increase the measure of this box by 1,
   since this is another visit to that box*)
   {m, 1,
    1000000}];(*we do this 1000000 times for each point that we have \
from the trajectory*)
  (*number of boxes that have been counted*)
  LL = Length[BOXMEASURE];
  (*information dimension for the current box size*)
  II = N[
    Sum[BOXMEASURE[[k, 3]]*
      Log[2, 1000001/BOXMEASURE[[k, 3]]]/1000001, {k, LL}]];
  Print[{"Box scale=", 1/2^i , "No. of Boxes=", Length[BOXLIST],
    "Information I=", II}];
  Write["LHI", {i, 1/2^i, Length[BOXLIST], II}],
  {i, 10}];
 Close["LHI"];
```

In the above Mathematica algorithm, we plot information versus the logarithmic inverse scale. We observe that $I(\epsilon)$ increases logarithmically with $1/\epsilon$. In other words, there is a constant and a slope in the plot of $I(\epsilon)$ versus $\log_2(1/\epsilon)$ that characterizes the information growth. It is the additional amount of information obtained by doubling the resolution of the subdivision of the attractor area (i.e. reducing ϵ or increasing the box number). This number is then called the information dimension. The information dimension is a lower bound for the fractal (box-counting) dimension ([4, p.685]). This is confirmed by our results of 1.24 for the

```
(*calcualte the information dimension by
 differnece of consecutive information values*)
Table[ReadList["LHI"][[i + 1, 4]] - ReadList["LHI"][[i, 4]],
 {i, Length[ReadList["LHI"]] - 1}]
{1.3287590240972622, 1.3120592917453413, 1.143338509042902,
 1.2536926326924958, 1.2674799945860533, 1.2246648918297023,
 1.2092748230296415, 1.214367266083089, 1.237071052405243}
(*create a list of I versus Log[2,1/epsilon]*)
L5 = Table[{Log[2, 1 / ReadList["LHI"][[i, 2]]], ReadList["LHI"][[i, 4]]},
  {i, Length[ReadList["LHI"]] - 1}]
{{1, 1.8390673151799173}, {2, 3.1678263392771795}, {3, 4.479885631022521},
 {4, 5.623224140065423}, {5, 6.876916772757919}, {6, 8.144396767343972},
 {7, 9.369061659173674}, {8, 10.578336482203316}, {9, 11.792703748286405}}
(*creating a linear fit*)
Clear[x];
Fit[L5, {1, x}, x]
0.6791353579690469 + 1.23909334741308 x
Show[ListPlot[L5, PlotStyle → {Black, AbsolutePointSize[7]}],
 Plot[0.6791353579690469 + 1.239093347413086 * x, {x, 0, 9.5},
  PlotStyle → {Black, AbsoluteThickness[2]}]]
```

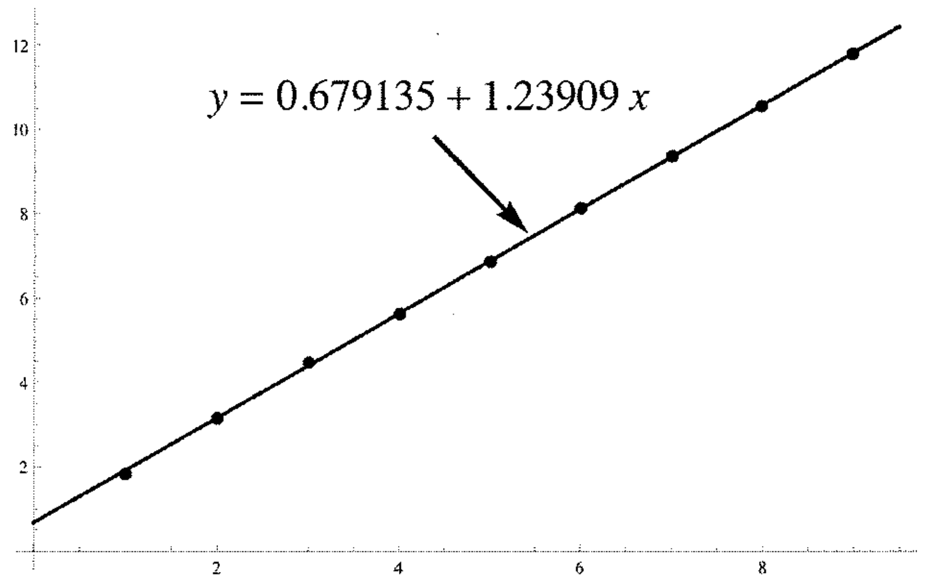

Figure 2.11: Part 2 of our Mathematica routine for calculating the information dimension of the Hénon map, with a linear fit of the slope 1.23909.

information dimension and 1.25 for the box-counting dimension of the Hénon map.

In table 2.6, we have assembled the dimension calculations for the known systems. The table offers a side by side comparison of the box-counting dimension for the partial and complete linear fit as well as the information dimension for the given systems. Notice that the information dimension is always slightly smaller than the box-counting dimension for the partial

System name	**Box-counting dimension (complete data fit)**	**Box-counting dimension (partial data fit)**	**Number of points n**	**information Dimension**	**Number of points**
Hénon system	1.12867	1.24865	10,000,000	1.23909	1,000,000
Lozi system	1.28245	1.38593	1, 000, 000	1.34454*	100,000
Ikeda system	1.56645	1.71449	1,000,000	1.6953*	100,000
Sprott map A	1.42396*	1.56532*	3,000,000	1.45582*	100,000
Sprott map B	1.55061*	1.71173*	3,000,000	1.50602*	100,000
Sprott map C	1.41198*	1.56732*	3,000,000	1.43689*	100,000
Sprott map D	1.57011*	1.81102*	3,000,000	1.46369*	100,000
Ueda system	1.72807	1.87097	500,000	1.86331*	100,000

Table 2.6: Box-counting versus Information dimension of known systems. (* indicates new results.)

linear fit which is the more accurate estimation of the two box-counting calculations.

In conclusion, the information dimension procedure is more time consuming computationally wise on a per iteration basis than the box-counting procedure because the information dimension algorithm keeps track of both the boxes and the points within them. However, due to the information dimension's calculation precision it does not require the extensive number of iterations that the box-counting dimension procedure does. In table 2.6 above, we can see that the information dimension for the Hénon attractor differs by less than one one-hundredth of the box-counting dimension calculation for the partial fit. What is important and should be noted is that the information dimension achieved its results using 9,000,000 less iterations. Due to the smaller number of iterations necessary to achieve an accurate calculation, the information dimension procedure actually turns out to be faster then the box-counting procedure. Thus, the information dimension offers the best results.

2.5 Correlation Dimension

Applying the conceptually simple box-counting algorithm to higher-dimensional data reveals many shortcomings of the method, particularly the number of computations increases exponentially with the phase space dimension. To provide a computationally simpler dimension calculation for an attractor, the correlation dimension was introduced ([14, p.411]). It has a computational advantage because it uses the trajectory points directly and does not require a separate partitioning of phase space.

Given a trajectory of a system with n attractor points, for each point x_i on the trajectory, we want to find the number of points $N_r(x_i)$ on the trajectory within a given distance r of the x_i.

Definition 2.5.1. The **correlation sum** $C(r)$ is defined by

$$C(r) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{n-1} N_r(x_i).$$

If all of the trajectory falls within distance r of each other, $C(r) = 1$. Conversely, if r is smaller than the shortest distance between any two points on the trajectory, $N_r(x_i) = 0$ for all x_i, so $C(r) = 0$.

Using the **Heaviside step function** θ defined by

$$\theta(x) = 0 \text{ if } x < 0$$

$$\theta(x) = 1 \text{ if } x \geq 0,$$

we can write the term

$$\frac{N_r(x_i)}{n-1} = \frac{1}{n-1}\sum_{j\neq i}^{n} \theta(r - |x_i - x_j|),$$

where the sum ranges over all points $x_j \neq x_i$ on the trajectory, and $\theta(r - |x_i - x_j|) = 1$ for each point x_j that is within distance r of the point x_i otherwise, it equals 0. So $C(r)$ is then given by

$$C(r) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}^{n} \theta(r - |x_i - x_j|).$$

Definition 2.5.2. The **correlation dimension** D_c is defined by

$$C(r) = \lim_{r \to 0} r^{D_c},$$

equivalently,

$$D_c = \lim_{r \to 0} \frac{\log_2 C(r)}{\log_2 r}$$

The main question in this approach is how to compute the correlation. We propose a **correlation algorithm** according to which we can carry out a clustering for a given distance r. Given a trajectory of n points of our system $\{x_1, x_2, \ldots, x_n\}$, consider the first point x_1 and consider all points x_i that are distance r or less from x_1. Call this collection $N_r(x_1)$. Then follow along the trajectory and check if subsequent points are elements of $N_r(x_1)$. Find the point x_k on the trajectory such that $x_k \notin N_r(x_1)$ and $x_{k-1} \in N_r(x_1)$, that is, the first point on the trajectory we encounter after x_1 that is not in the cluster x_1. Then construct $N_r(x_k)$. Proceed in this manner until we have clustered the whole trajectory.

Observe that we may cluster points that are far ahead on the trajectory with a point at the beginning. Once a point has been clustered, it is no longer considered for a cluster center.

So we have constructed a collection $\{N_r(x_1), N_r(x_i), \ldots, N_r(x_j), \ldots, N_r(x_m)\}$, where x_m is the last point that acquired a clustering. Each cluster $N_r(x_i)$ also comes with a weight or a measure, given by the number of points, including x_i, that belong to $N_r(x_i)$.

Next, collapse the cluster to their cluster points, and form a new sequence $\{x_1(w_1), \ldots, x_i(w_i), \ldots, x_m(w_m)\}$, where w_i indicates the weight of each cluster. Now, we use the sequence to carry out a box-counting dimension estimate.

For each value of r, we carry out this procedure. It generates a lot of data, but we hope to be able to extract a much more detailed picture of the attractor structure than with the usual tools.

Remark 2.5.3. The above idea is called the **cluster growing method** for which one **seed vertex** is chosen. A cluster of vertices separated by at most the given distance r from the seed vertex can be formed. The procedure is repeated by choosing many seeds until the cluster covers the whole attractor.

The Cluster algorithm takes a map like the Hénon map in figure 2.12, and transforms it into the map in figure 2.13. Further examples of a clustering for a correlation dimension can be seen in figures 2.14, and 2.15.

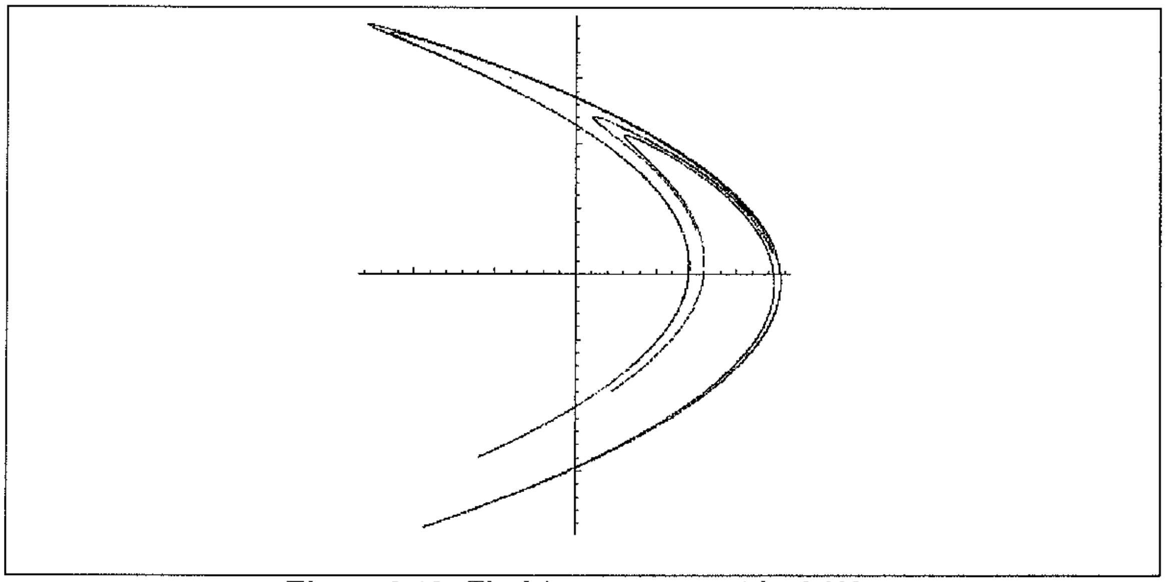
Figure 2.12: The hénon attractor with 10,000,.

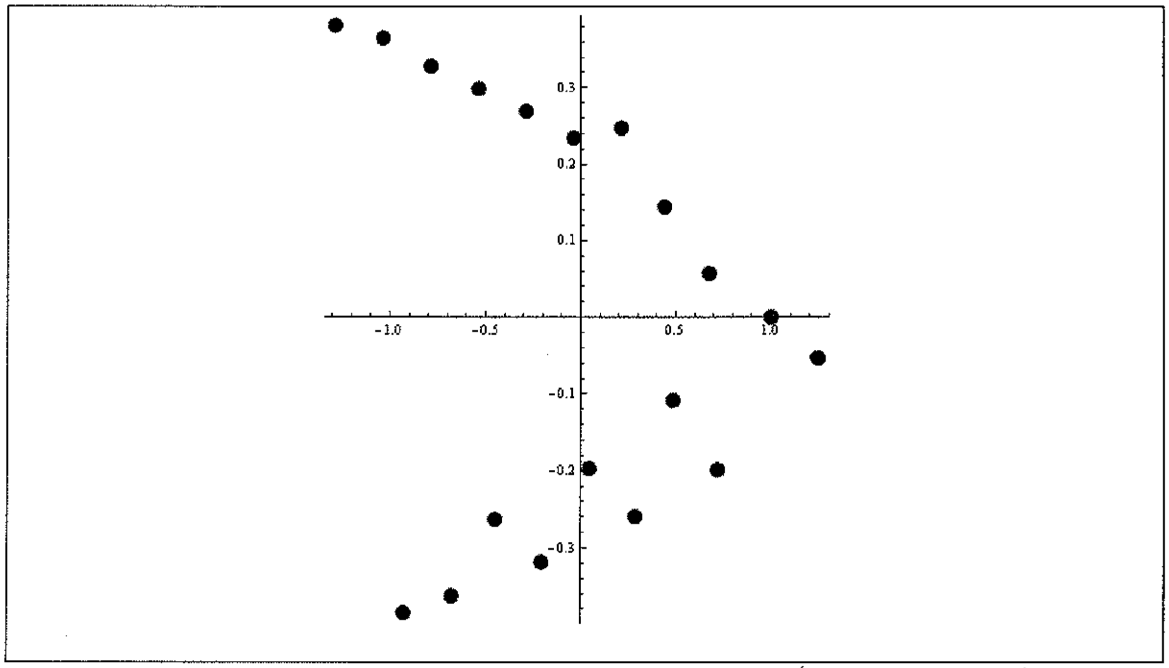

Figure 2.13: The hénon attractor with 10,000, and a cluster radius of 1/4.

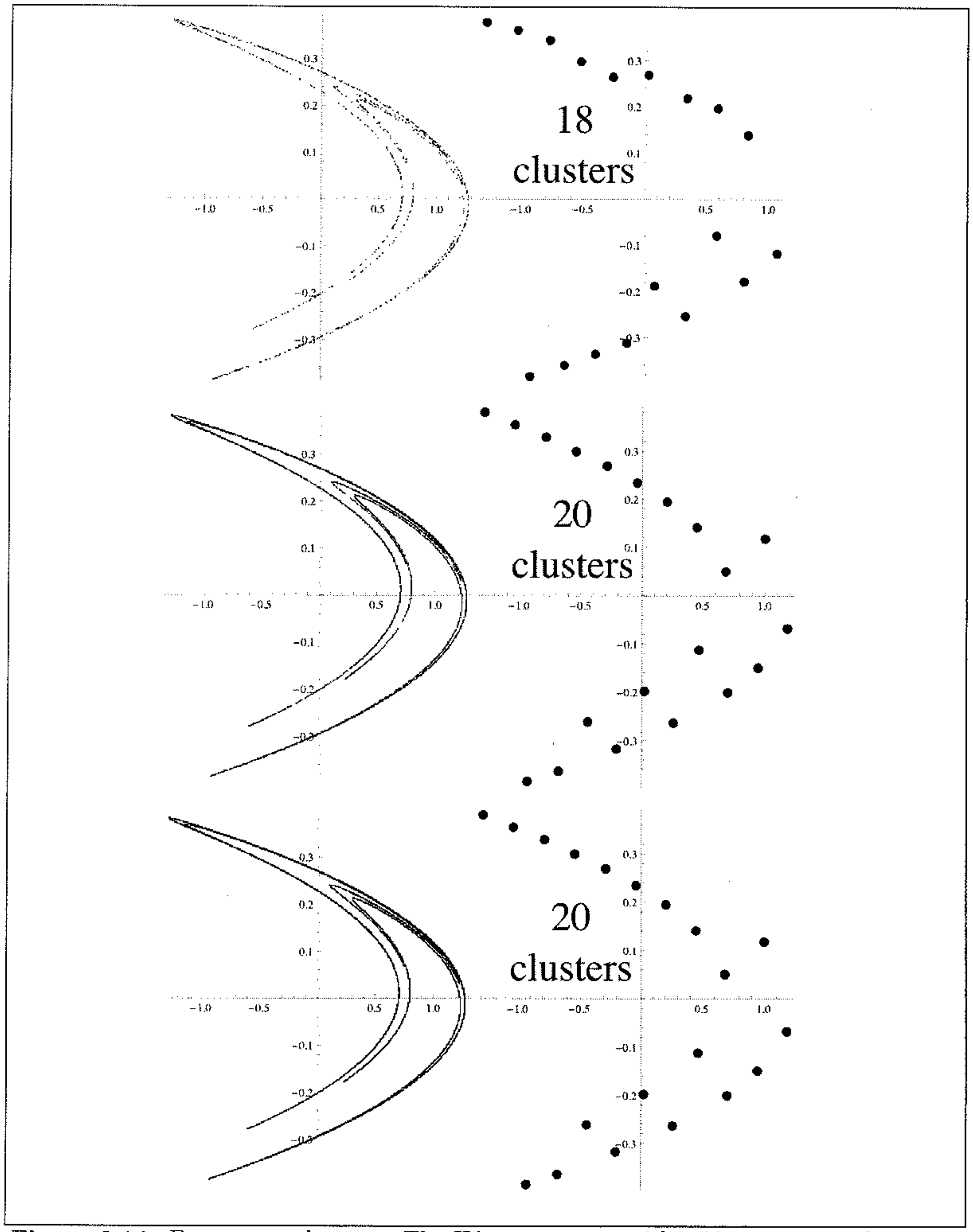

Figure 2.14: From top to bottom. The Hénon attractor with 1,000, 10,000, and 100,000 points and a cluster radius of 1/4.

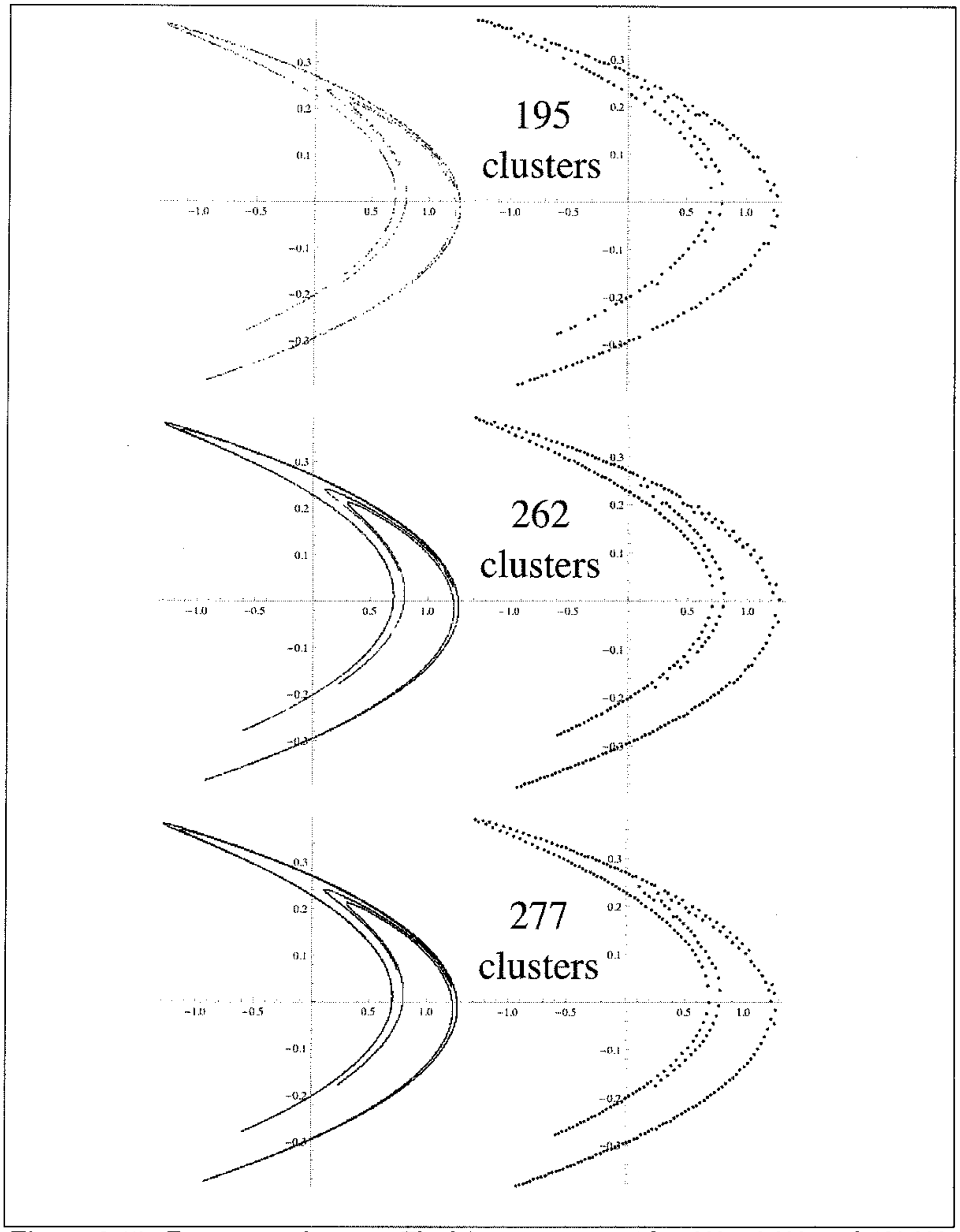

Figure 2.15: From top to bottom. The hénon attractor with 1,000, 10,000, and 100,000 points and a cluster radius of $1/2^5$.

Remark 2.5.4. Very clearly, we are observing a similar behavior as we did with the box-counting algorithm in the dependence of the boxes needed to cover the trajectory of a system on the number of points of the trajectory and the box scale. Note that these are not equivalent notions because clusters do not aim to create higher resolution on the trajectory, just the opposite. Otherwise, clearly, we could keep making the clusters smaller and smaller, until the radius of each cluster is smaller than the minimum distance between any two points on the trajectory, thereby making each individual point x_i into a cluster. This, however, would be impractical, as we are looking for a semi-global way of observing the trajectory, so the cluster radius should be small enough so as to preserve important features of the system (a cluster radius that clusters everything to one point would be just as useless as the minimal cluster radius described above), yet large enough so that it offers a useful simplification of the trajectory. In essence, we are seeing the same "flatting" of the slope as we did for the box-counting dimension wehn the box sizes were very small. This suggest that there is a scaling region for the correlation dimension where

$$(\text{Minimium seperation of points on A}) \ll r \ll (\text{diameter of A})$$

where A is the attractor.

System name	**r=8**	**r=12**	**r=26**	**r=36**	**r=51**	**Number of Points**
Hénon system	1.34248	1.28813	1.24881	1.23416	1.2055	10,000
Lozi system	1.4071	1.38506	1.34338	1.31894	1.28976	10,000
Ikeda system	1.63169	1.57696	1.47184	1.40336	1.31477	10,000
Sprott map A	1.60257	1.58898	1.55515	1.54725	1.53774	10,000
Sprott map B	1.58453	1.59114	1.57232	1.55844	1.54609	10,000
Sprott map C	1.70797	1.54825	1.54291	1.53448	1.5228	10,000
Sprott map D	1.44537	1.52013	1.59638	1.62191	1.63141	10,000

Table 2.7: The correlation dimension algorithm was run on all systems using 10,000 iteration points. The r in the table is in fact $1/2 \cdot r$ which is the radius that we used. As far as we know, all the results are new results.

In Table 2.7, we present the different dimension calculations for the known systems. We used 10,000 iteration points for all values of r (radius). We did this hoping to find an accurate dimension calculation using little computational time. In essence, we did this because we were

looking for fast accurate results for a dimension calculation. What we found is that the size of the attractor in phase space plays an important role in the computational time necessary to calculate the dimension.

Remark 2.5.5. A system occupying a 100X100 space on a Cartesian plan will take longer to compute than a system that occupies only a 3X4 area on the same Cartesian plan.

Since both systems have the same number of points, one would think that the system more densely compacted would take longer to calculate because of the amount of information that has to be unraveled. However, this is not the case. A system occupying a 100X100 space takes longer to compute due to the spacing of each individual point. Remember, both systems have the same number of n points, so the system occupying the greater area will have the greater number of seed vertices because the radius will encompass fewer of the n points for the system with the larger area. The increased number of seed vertices increases the number of calculations necessary to estimate the correlation dimension; thereby slowing down the computation time.

The benefit of the correlation dimension procedure is that it requires significantly less data to compute compared to the box-counting or information dimension procedure. To be more precise, the number of ϵ–boxes in the unit cube of $\mathbb{R}^n$ is ϵ^{-n}. If $\epsilon = 0.01$ and $n = 10$, then there are potentially 10^{20} boxes that need to be tracked using the box-counting algorithm [3, p.181]. This lead to a significant data structure problem.

In conclusion, the correlation dimension is computationally the fastest method that we have looked at; however, it suffers from inaccuracy problems. If a fast, accurate method is devised to determine r by the scaling region (see remark 2.5.4), then the correlation dimension algorithm would be the most practical of the the dimension calculation methods. As of now, our findings suggest that the information dimension calculation method offers the best results.

Below is an example of our Mathematica algorithm for the Correlation Dimension of the Hénon attractor.

```
Timing[
 OpenWrite["CTh"];
 Nu = 10^4;
 Do[
```

```
 r = 1/(2*i);
 x1 = 0.1;
 y1 = 0.1;
 A = 1.4;
 B = 0.3;
 OpenWrite["h"];
 Do[
  x2 = y1 + 1 - A x1^2;
   y2 = B x1;
  Write["h", {x2, y2}];
  x1 = x2;
  y1 = y2, {i, Nu}];
 Close["h"];
 DH = ReadList["h"];
 OpenWrite["Ch"];
 While[
  Length[DH] > 1,
  x1 = DH[[1]];
  COUNT = 0;
  DH1 = {};
  Do[
   If[
    EuclideanDistance[x1, DH[[k]]] < r,
    COUNT = COUNT + 1;
    DH1 = Append[DH1, DH[[k]]]], {k, Length[DH]}];
  DH = Complement[DH, DH1];
  Write["Ch", x1]];
 Close["Ch"];
 Print[{Nu, r, Length[ReadList["Ch"]]}];
 Write["CTh", {Nu, r, Length[ReadList["Ch"]]}], {i, 2, 51}];
Close["CTh"];
]
```

Chapter 3

Applications

In this chapter, we will study the new two and three–dimensional systems discussed in section 1.4. We will use all of our methods—Lyapunov exponent calculation, box-counting dimension, information dimension, and correlation dimension—to determine if the geometric structures that we chose truly possess chaotic behavior, and which dimension calculation offers the best results.

In section 3.3, we introduce future work called T–networks. T–networks stems directly from the work we have done thus far with dimension calculation but applied to real world problems.

3.1 Investigation of the two-dimensional systems

Recall the A2DM discrete dynamical system

$$x_{n+1} = Ay_n + x_n,$$

$$y_{n+1} = Bf(y_n) + g(x_n, y_n) + x_n,$$

where A and B are parameters, and $f(y_n)$ and $g(x_n, y_n)$ are certain continuous nonlinear function of x_n and y_n, such as $\cos y_n, y_n^m, \sin y_n$, etc.

The first subclass of systems that we look at from this family is

$$x_{n+1} = Ay_n + x_n,$$

$$y_{n+1} = By_n^{k_1/k_2} + x_n^{k_3/k_4} y_n^{k_5/k_6} + x_n,$$

where A and B are given values in the range $[-100, 100]$ with increments of 0.01. k_i is given integer values (excluding cases where the fractions were not defined) in the range $[1, 20]$. Below is an example of one of our mathematica searches.

```
Timing[
  Do[
   Do[
    Do[
     Do[
      Do[
       Do[
        Do[
         Do[

          If[IntegerQ[2*t/n] && IntegerQ[2*k/m] &&
             IntegerQ[2*r/s] ||
            (2*t/n == 0 && 2*k/m == 0) ||
            (2*t/n == 0 && 2*r/s == 0) ||
            (2*k/m == 0 && 2*r/s == 0)], Break[]];
         x1 = Random[];
         y1 = Random[];
         DeleteFile["A1x"];
         DeleteFile["A1y"];
         OpenWrite["A1x"];
         OpenWrite["A1y"];
         Do[
          x2 = A*y1 + x1;
          y2 = B*(y1^(2*t))^(1/
                n) + ((x1^(2*k))^(1/m))*((y1^(2*r))^(1/s)) + x1;
          If[Abs[x2] > 10^2 || Abs[y2] > 10^2, Break[]];
          Write["A1x", x2]; Write["A1y", y2];
          x1 = x2; y1 = y2, {i, 20000}];
         Close["A1x"]; Close["A1y"];
         If[Length[ReadList["A1x"]] == 20000,
          Print[{"A=", A, "B=", B, "y^", 2*t/n, "x^y^", 2*k/m, 2*r/s,
            ListPlot[Transpose[{ReadList["A1x"], ReadList["A1y"]}],
             PlotStyle -> { AbsolutePointSize[0.1], Black},
             AspectRatio -> 1.0]}]],
         {A, -100, 100, 0.02}],
        {B, 3, 100, 0.01}],
       {k, 10}],
      {m, 20}],
     {r, 10}],
    {s, 20}],
   {t, 10}],
  {n, 20}]
 ]
```

Our initial search was carried out with a Mathematica routine that was essentially an 8-fold parallel loop, and the search was for bounded non-periodic, non-divergent solutions, looking

at each trajectory after the first 10,000 iterations. If the trajectory was still bounded (to a 100×100 square in the plane), non-periodic, and not exhibiting a slowly diverging behavior (such as spiraling outward) the trajectory was recorded.

We also excluded the cases where both equations were just polynomials in x_n and y_n because Sprott has already performed such searches, even if the constants were given only integer values in his study (see [6]). We were more interested in non-polynomial cases, so we required at least one non-integer exponent in the search. We selected several systems to test for chaotic behavior using the Lyapunov exponent calculation method from section 2.1. We ran the box-counting dimension algorithm from section 2.3, the information dimension algorithm form section 2.4, and the correlation dimension algorithm from section 2.5, to test which dimension calculation offers the most accurate measure of the system with the least amount of computational time necessary.

3.1.1 Box-counting Results for A2D Maps

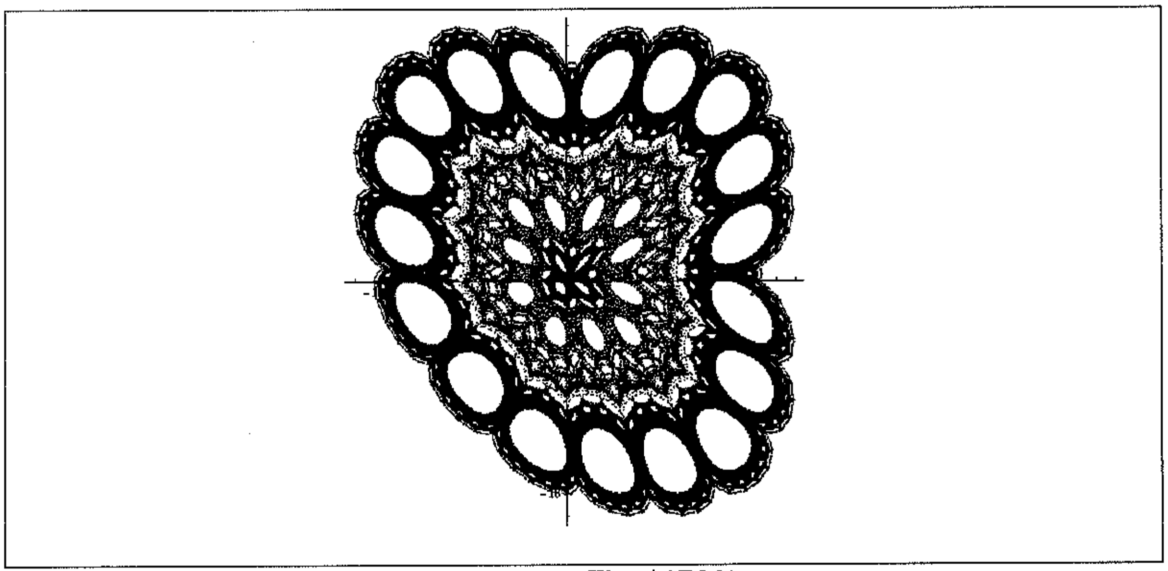

Figure 3.1: The A2DM1.

Example 3.1.1. For the A2DM1, we have the equation

$$x_{n+1} = y_n, \tag{3.1}$$
$$y_{n+1} = -0.4y_n + y_n^{2/3} - x_n$$

with the initial values $x_0 = 0.58064$ and $y_0 = 0.90151$.

We ran two separate box-counting procedures for two separate iteration limits. One was set for 1,000,000 iteration and the other was set for 100,000 iteration of the equation. We did this in order to demonstrate how the number of points on a map affects the scaling for the box-counting dimension of a map. For 1,000,000 iterations, we obtained a box-counting dimension of 1.7553 for the complete data fit, and for 100,000 iteration of the same map, we obtained a box-counting dimension of 1.7766—again for a complete data fit. For the complete data fit, 900,000 iteration produces a discrepancy of almost two–one–hundredths. The results can be seen in figure 3.2.

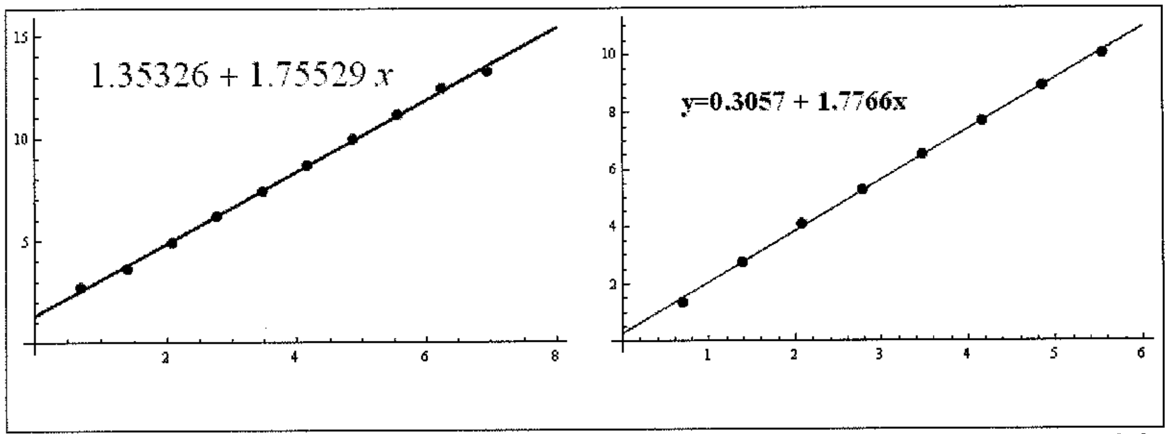

Figure 3.2: Plot of $\ln(1/\epsilon)$ versus $\ln(N(\epsilon))$ for the A2DM1 with 1,000,000 points on the left and the same map with 100,000 points on the right for a linear fit to the full data.

We can only imagine how the discrepancy will continue to grow as the number of iterations increases. Additionally, for 1,000,000 points, we obtain a box-counting dimension of 1.8099 for the partial data fit, and 1.8160 for the partial data fit to 100,000 points of the same map. The discrepancy between the two dimension calculations for the partial fit is within one–one–hundredth. The partial data fit for the box-counting dimension not only produces closer approximations over large variations in number of iterations, but it is also a more accurate

estimate of the dimension because it represents a smoother linear fit to the data. A visual representation of the dimension for the linear fit to the partial data can be seen in figure 3.3. Notice that the points conform to the line better than those in figure 3.2.

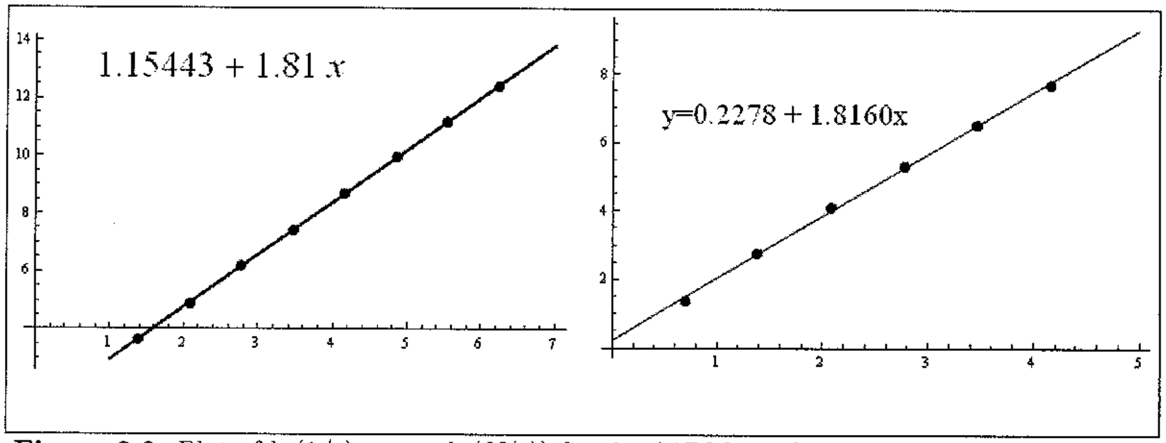

Figure 3.3: Plot of $\ln(1/\epsilon)$ versus $\ln(N(\epsilon))$ for the A2DM1 with 1,000,000 points on the left and the same map with 100,000 points on the right for the linear fit to the partial data.

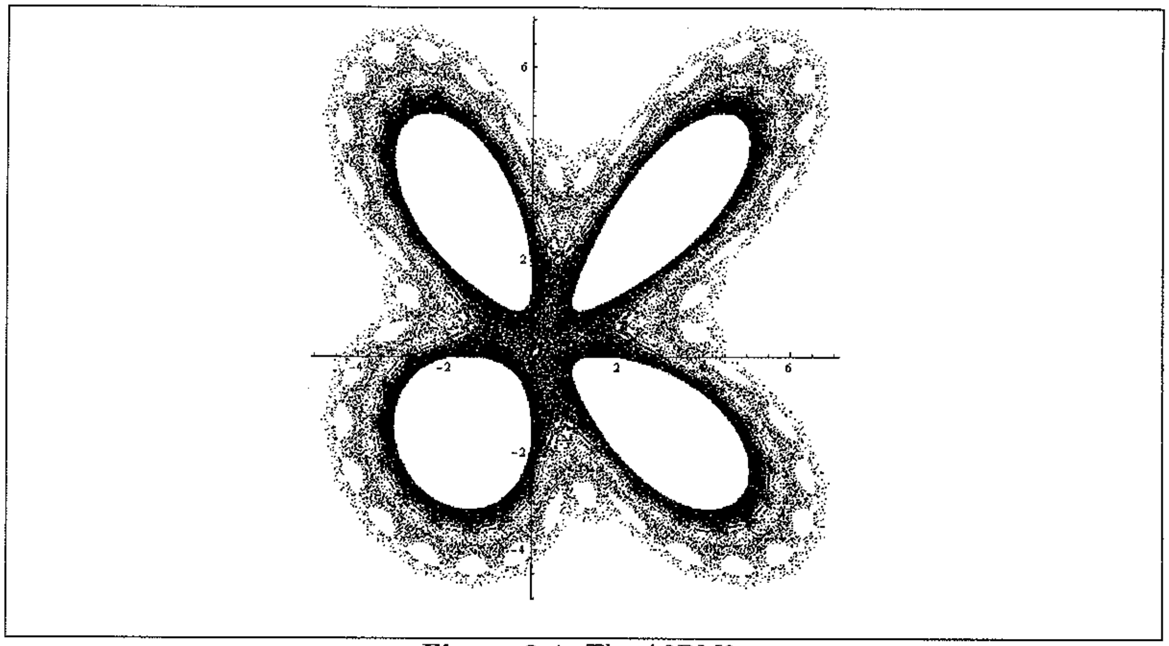

Figure 3.4: The A2DM2.

Example 3.1.2. For the A2DM2, we have the equation

$$\begin{aligned} x_{n+1} &= y_n, \\ y_{n+1} &= -0.175y_n + y_n^{2/3} - x_n \end{aligned} \tag{3.2}$$

with the initial values $x_0 = 0.98349$ and $y_0 = 0.56363$.

We ran the box-counting procedures for an iteration limit of 100,000 points on the A2DM2. At this point in the project we had very limited computation abilities time wise; thus, the reason for the small amount of iterations performed. For 100,000 iterations, we obtained a box-counting dimension of 1.60913 for the complete data fit, and 1.6758 for the partial data fit of the same map. The discrepancy between the two dimension calculations is almost seven–one–hundredths. Again, the partial data fit for the box-counting dimension is the more accurate estimate of the dimensions because it represents a smoother linear fit to the data.

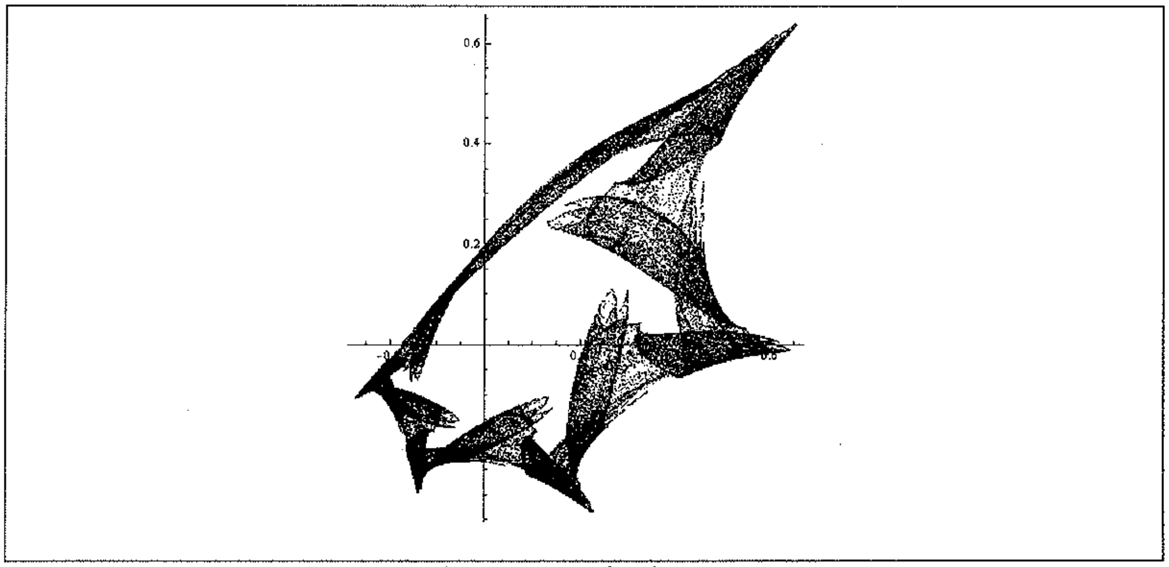

Figure 3.5: The A2DM3.

Example 3.1.3. For the A2DM3, we have the equation

$$\begin{aligned} x_{n+1} &= -1.239y_n + x_n, \\ y_{n+1} &= -2.519y_n^2 + x_n^{2/3}y_n^2 + x_n \end{aligned} \tag{3.3}$$

with the initial values $x_0 = 0.564125$ and $y_0 = 0.465134$.

We ran the box-counting procedures for an iteration limit of 100,000 points and obtained a box-counting dimension of 1.5798 for the complete data fit. For the partial data fit of the same map, we obtained a dimension of 1.6291. The discrepancy between the two dimension calculations is almost five–one–hundredths. Again, the partial data fit for the box-counting dimension is the more accurate estimate of the dimensions because it represents a smoother linear fit to the data.

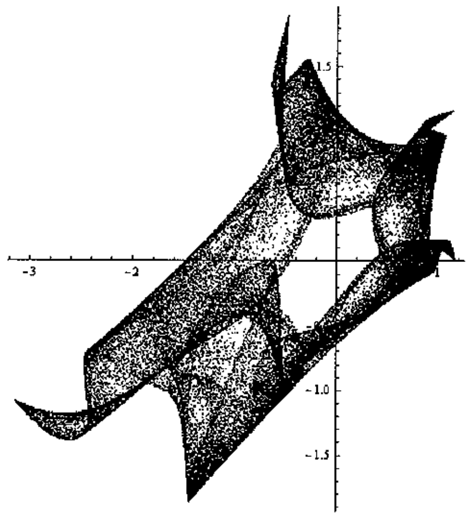

Figure 3.6: The A2DM4.

Example 3.1.4. For the A2DM4, produced by the equation

$$x_{n+1} = -1.4y_n + x_n, \qquad (3.4)$$
$$y_{n+1} = -0.1y_n + x_n \sin(y_n)^3 + x_n$$

with the initial values $x_0 = 0.85317$ and $y_0 = 1.06121$.

We ran the box-counting procedures for an iteration limit of 100,000 points. The complete data fit produced a box-counting dimension of 1.6127, and 1.7088 for the partial data fit of the same map. The discrepancy between the two dimension calculations is almost nine–one–hundredths. Again, the partial data fit for the box-counting dimension is the more accurate estimate of the dimensions because it represents a smoother linear fit to the data.

System name	Box-counting dimension (complete data fit)	Box-counting dimension (partial data fit)	Number of points n	Box size scaling
A2DM1	1.7553	1.8099	1,000,000	2^{-1} to 2^{-9}
A2DM1	1.7928	1.8110	100,000	2^{-1} to 2^{-10}
A2DM2	1.6091	1.6758	100,000	2^{-1} to 2^{-10}
A2DM3	1.5798	1.6291	100,000	2^{-1} to 2^{-10}
A2DM4	1.6127	1.7088	100,000	2^{-1} to 2^{-10}

Table 3.1: Results of the box-counting dimension calculations for the 2D systems.

Over all, the box-counting dimension for the complete data fit is a lower dimensional estimate to the scaling calculation for the linear fit to the system. This simply means that when a partial fit to the data is performed, the box-counting dimension increases by some number. The results can be seen in table 3.1. For the A2DM1 with 1,000,000 points there is an increase of almost five–one–hundredths. For the same map with 100,000 points the increase is only about four–one–hundredths. For maps 2,3, and 4 respectively we have increases of seven, five, and nine–one–hundredths. This same increase in scaling calculation was seen in section 2.3, table 2.3 , so our data for the A2D maps is consistent and expected.

3.1.2 Information Dimension and Lyapunov Exponent of A2D maps

Additionally, we calculated the information dimension as well as the Lyapunov exponent for the A2D systems.

Example 3.1.5. For the A2DM1 produced by equation 3.1, we ran two seperate information dimension algorithm for 1,000,000 and another for 100,000 point on the trajectory. For 1,000,000 points we obtained a information dimension of 1.79395, and 100,000 points on the same map we calculated a dimension of 1.7253. Again, we see that the number of iterations produces a discrepancy for the calculation of the dimension. In this case, the discrepancy is by almost seven–one–hundredths.

Remark 3.1.6. It is important to note that for the information dimension calculation the lower iteration limit produced the lower dimension calculation, whereas for the box-counting

algorithm, the lower iteration limit produced the higher dimension calculation.

This may have to do with the fact that the information dimension calculation produces a lower limit for the dimension. A graphical representation of the linear fit for the A2DM1 for 1,000,000 iteration versus 100,000 iteration can be seen in figure 3.7.

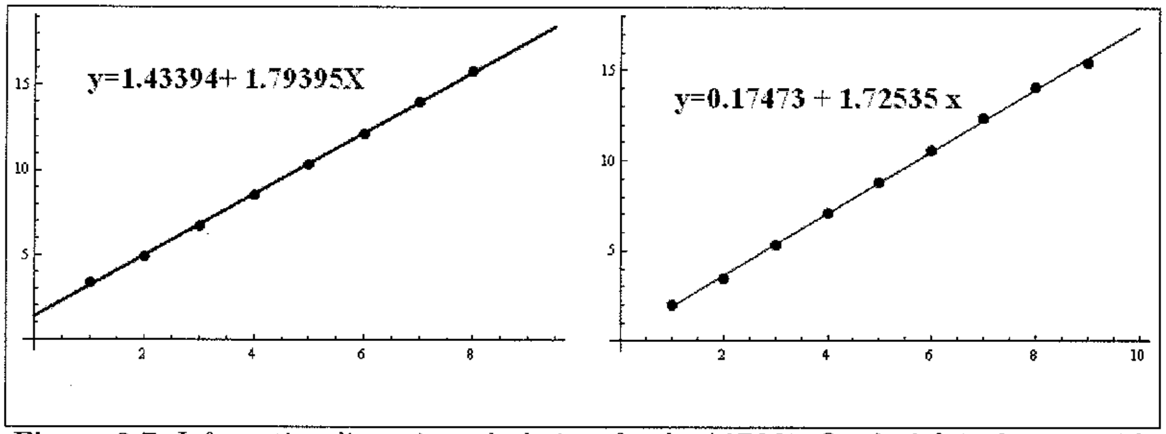

Figure 3.7: Information dimension calculations for the A2DM1. On the left is the map with 1,000,000 iterations of the system, and on the right is the same map with 100,000 iterations of the system.

Additionally, the Lyapunov exponent for the A2DM1 is 0.0910897. This is a positive Lyapunov exponent which implies that the system is sensitive to initial conditions. The positive Lyapunov exponent coupled with the presence of a fractal dimension, as verified by the box-counting dimension and the information dimension, indicates clear conclusive chaotic behavior of the system.

System name	**Information dimension**	**Number of points n**	**Box size scaling**
2D map 1	1.79395	1,000,000	2^{-1} to 2^{-9}
2D map 1	1.7253	100,000	2^{-1} to 2^{-10}
2D map 2	1.6404	100,000	2^{-1} to 2^{-10}
2D map 3	1.5776	100,000	2^{-1} to 2^{-10}
2D map 4	1.5825	100,000	2^{-1} to 2^{-10}

Table 3.2: Results of the information dimension calculations for the A2D maps.

Example 3.1.7. For the A2DM2 produced by equation 3.2, we ran the information dimension algorithm for 100,000 iterations as it was very time consuming and beyond our limited ability

to run it for a higher iteration count. We calculated the amount of information for box sizes ranging from 1/2 to 1/1024. For the A2DM2, we obtained an information dimension of 1.6404. Recall that our box-counting dimension of the system for the partial data fit was 1.6758. The information dimension is about three–one–hundredths less than the box-counting dimension. This confirms our belief that the information dimension algorithm produces a lower dimensional limit.

Additionally, The Lyapunov exponent for the A2DM2 is 0.101707. This is a positive Lyapunov exponent which implies that the system is sensitive to initial conditions. The positive Lyapunov exponent coupled with the presence of a fractal dimension, as verified by the box-counting dimension and the information dimension, indicates clear conclusive chaotic behavior of the system.

System	**Lyapunov exponent λ**	**Number of iterations**
2D Map 1	$\lambda = 0.0910897$	$1,000,000$
2D Map 2	$\lambda = 0.101707$	$1,000,000$
2D Map 3	$\lambda = 0.422653$	$1,000,000$
2D Map 4	$\lambda = 0.395895$	$1,000,000$

Table 3.3: Lyapunov exponents calculations for the A2D systems

Example 3.1.8. For the A2DM3 produced by equation 3.3, we ran the information dimension algorithm for 100,000 iterations as it was very time consuming and beyond our limited ability to run it for a higher iteration count. We calculated the amount of information for box sizes ranging from 1/2 to 1/1024. We obtained an information dimension of 1.5776. Recall that our box-counting dimension of the system for the partial data fit was 1.6291. The information dimension is about five–one–hundredths less than the box-counting dimension. This confirms our belief that the information dimension algorithm produces a lower dimensional limit.

Additionally, The Lyapunov exponent for the A2DM3 is 0.422653. This is a positive Lyapunov exponent which implies that the system is sensitive to initial conditions. The positive Lyapunov exponent coupled with the presence of a fractal dimension, as verified by the box-counting dimension and the information dimension, indicates clear conclusive chaotic behavior of the system.

Example 3.1.9. For the A2DM4 produced by equation 3.4, we ran the information dimension algorithm for 100,000 iterations. We calculated the amount of information for box sizes ranging from 1/2 to 1/1024. We obtained an information dimension of 1.5825. Recall that our box-counting dimension of the system for the partial data fit was 1.7088. The information dimension is about two–one–hundredths less than the box-counting dimension. This confirms our belief that the information dimension algorithm produces a lower dimensional limit.

Additionally, The Lyapunov exponent for the A2DM4 is 0.395895. This is a positive Lyapunov exponent which implies that the system is sensitive to initial conditions. The positive Lyapunov exponent coupled with the presence of a fractal dimension, as verified by the box-counting dimension and the information dimension, indicates clear conclusive chaotic behavior of the system.

3.1.3 Correlation Dimension for the A2DM Systems.

System name	**r=8**	**r=12**	**r=26**	**r=36**	**Number of Points**
A2DM1	1.5882	1.5277	1.38223	1.27745	10,000
A2DM2	1.47368	1.38669	1.20597	1.09889	10,000
A2DM3	1.51023	1.51274	1.54652	1.54672	10,000
A2DM4	1.58973	1.5591	1.41566	1.32491	10,000

Table 3.4: The correlation dimension algorithm was run on the A2DM systems using 10,000 iteration points.

We also ran the correlation dimension algorithm for the A2DM systems. We used 10,000 iterations for all of the attractors, and the radius is given by $1/2 * r$. So if $r = 8$, the radius is really 1/16. Additionally, the dimension calculation for an r_i value represents all values in multiples of 2 leading up to r_i. When comparing the correlation dimension to that of the information dimension for the systems, the only system having a similar dimension calculation is the A2DM4 for r=8. The other systems vary greatly. We believe this is do to the fact that r_i should be adjusted according to the scaling region as discussed in Section 2.5. Our conclusion for the A2DM systems follows are conclusion from Chapter 2: The information dimension algorithm produces the best results.

3.2 Investigation of the three-dimensional systems

Recall the A3Dm Dynamical systems from Chapter 1 given by the equation

$$\begin{aligned} x' &= Ay, \\ y' &= Bz, \\ z' &= Cf(x,y) + g(x,z) + h(y,z) + x, y, z, \end{aligned} \tag{3.5}$$

where x, y, z are functions of t, and A, B, C, f, g, h are as above. We will use the RK5 algorithm to approximate and study this family of nonlinear dynamical systems.

We performed several different searches, running the system for 10,000 iterations and varying the parameter A from -100 to 100 with increments of 0.001, initially testing for boundedness. After the initial search was performed, the three-dimensional systems were chosen because they were aesthetically pleasing. Our dimension calculation algorithms were not performing as expected at this point, but we were able to perform the Lyapunov spectrum algorithm for the three-dimensional system with positive results.

The Lyapunov exponents for a three-dimensional system are quantities that measure the average divergence rate of nearby trajectories in phase space. In simple terms, what we are doing is starting the system twice with two different initial conditions. One system starts with the exact initial conditions that produce the geometric shape in phase space. The other system starts with initial conditions that are slightly different by some infinitesimally small error ϵ. This error ϵ can be due to rounding off decimal places of the original initial condition. The two systems are then ran simultaneously for an amount of time. At each step of the run, the error ϵ is measured and set back to the initial error length that the two systems started with. Over the entire length of the system run, the additional error lengths of ϵ are measured and accumulated. At the end of the system run, the accumulated additional error lengths ϵ are averaged. The quantities of the averages (there will be three for a three-dimensional system) represent our Lyapunov spectrum. If the first exponent of the spectrum is positive, than by the Lyapunov definition we have a system that displays sensitivity to initial conditions. Hence, we have a chaotic system. Table 3.5 shows the positive exponent results of our Lyapunov

algorithm for the three-dimensional systems.

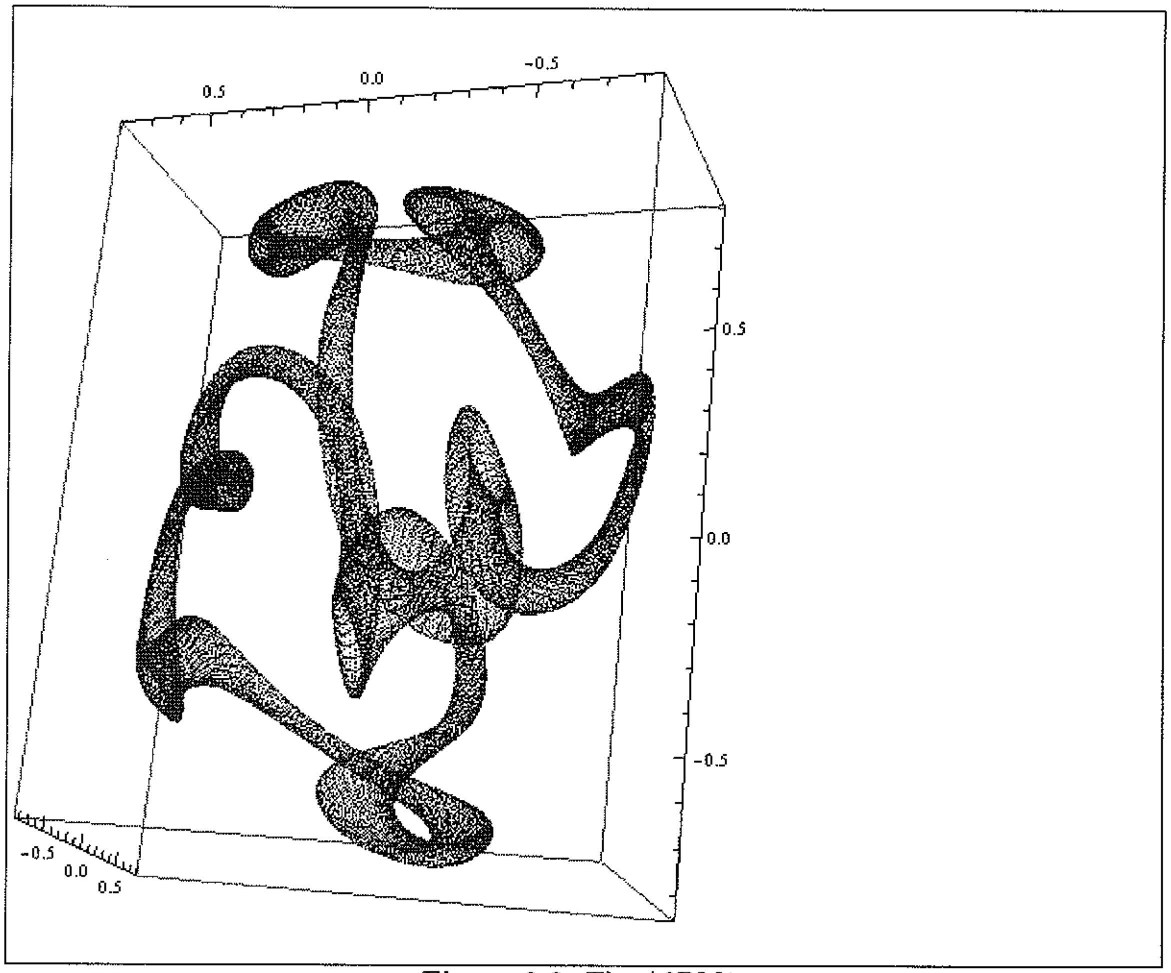

Figure 3.8: The A3DM1.

Example 3.2.1. For the A3DM1, we have the equation

$$x_1 = y_o \tag{3.6}$$

$$y_1 = z_0$$

$$z_1 = 0.403z_0 + y_0^3 - x_0$$

with initial conditions of $x_0 = 0.0593$, $y_0 = 0.58647$, and $z_0 = 0.05069$. We ran this system for 1,000,000 time steps and obtained a positive Lyapunov exponent of $\lambda = 0.610616$. This system clearly displays divergence of nearby trajectories, and is therefore chaotic.

System	Lyapunov exponent λ	Number of iterations
A3DM1	$\lambda = 0.610616$	$1,000,000$
A3DM2	$\lambda = 0.550468$	$1,000,000$
A3DM3	$\lambda = 0.671025$	$1,000,000$
A3DM4	$\lambda = 0.679372$	$1,000,000$

Table 3.5: Lyapunov exponents calculations for the A3DM systems

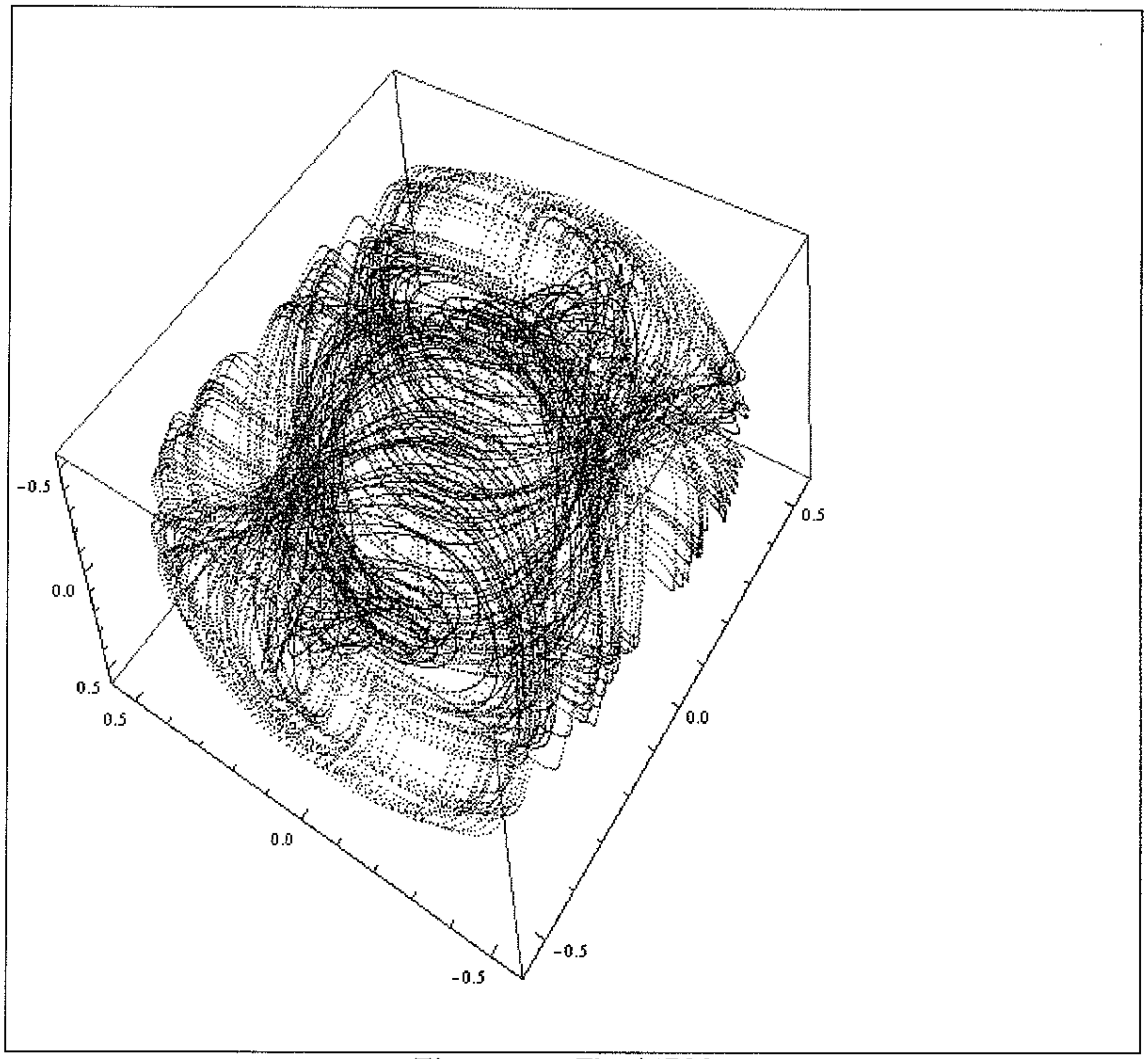

Figure 3.9: The A3DM2.

Example 3.2.2. For the A3DM2, we have the equation

$$\begin{aligned} x_1 &= y_o \\ y_1 &= z_0 \\ z_1 &= 0.007z_0 + y_0^7 - x_0 \end{aligned} \tag{3.7}$$

with initial conditions of $x_0 = 0.43402$, $y_0 = 0.23861$, and $z_0 = 0.07611$. We ran this system for 1,000,000 time steps and obtained a positive Lyapunov exponent of $\lambda = 0.550468$. This system clearly displays divergence of nearby trajectories, and is therefore chaotic.

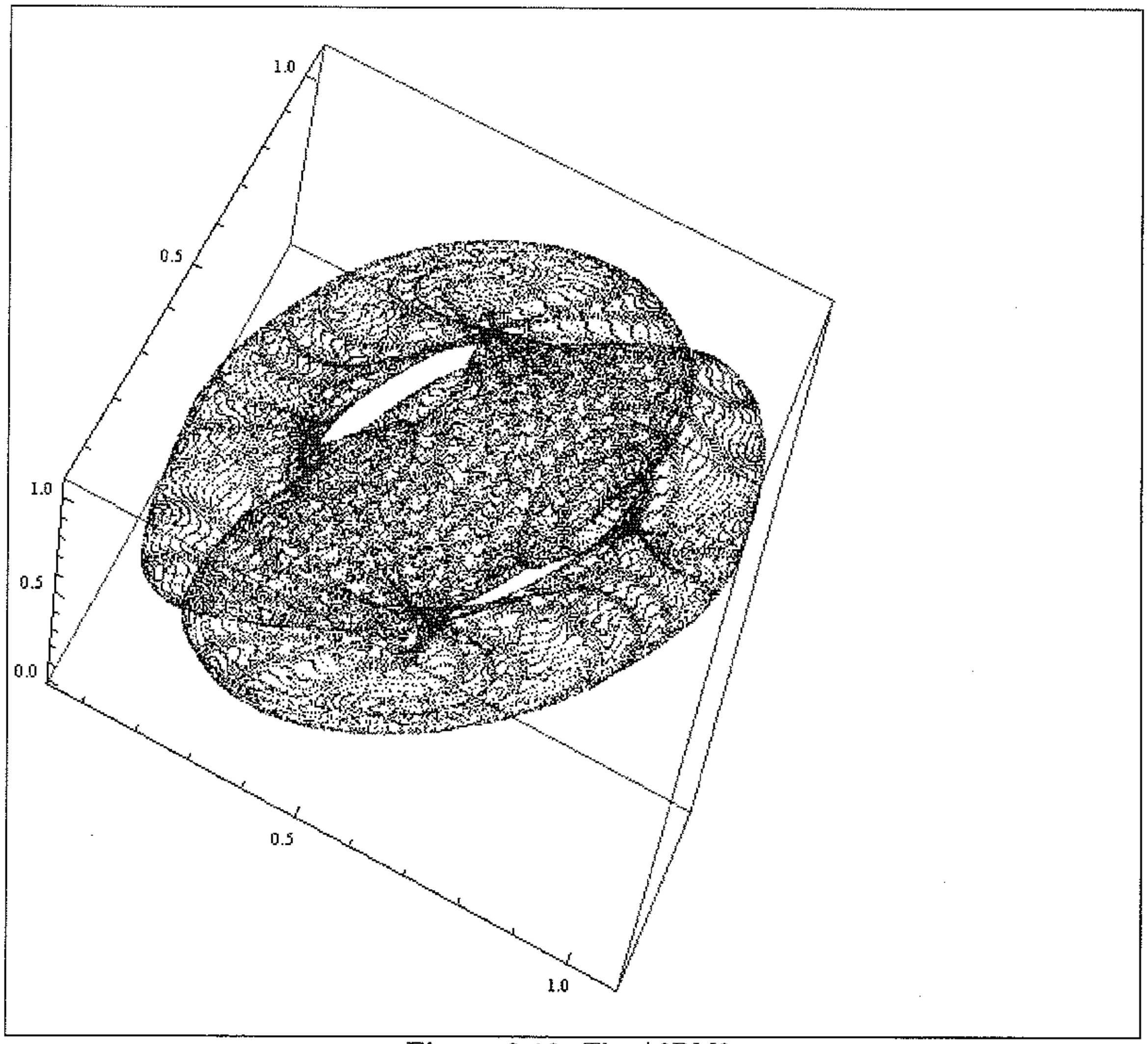

Figure 3.10: The A3DM3.

Example 3.2.3. For the A3DM3, we have the equation

$$\begin{aligned} x_1 &= y_o \\ y_1 &= z_0 \\ z_1 &= 0.621z_0 + y_0^{0.4} - x_0 \end{aligned} \tag{3.8}$$

with initial conditions of $x_0 = 0.50956$, $y_0 = 0.11749$, and $z_0 = 0.72381$. We ran this system for 1,000,000 time steps and obtained a positive Lyapunov exponent of $\lambda = 0.671025$. This system clearly displays divergence of nearby trajectories, and is therefore chaotic.

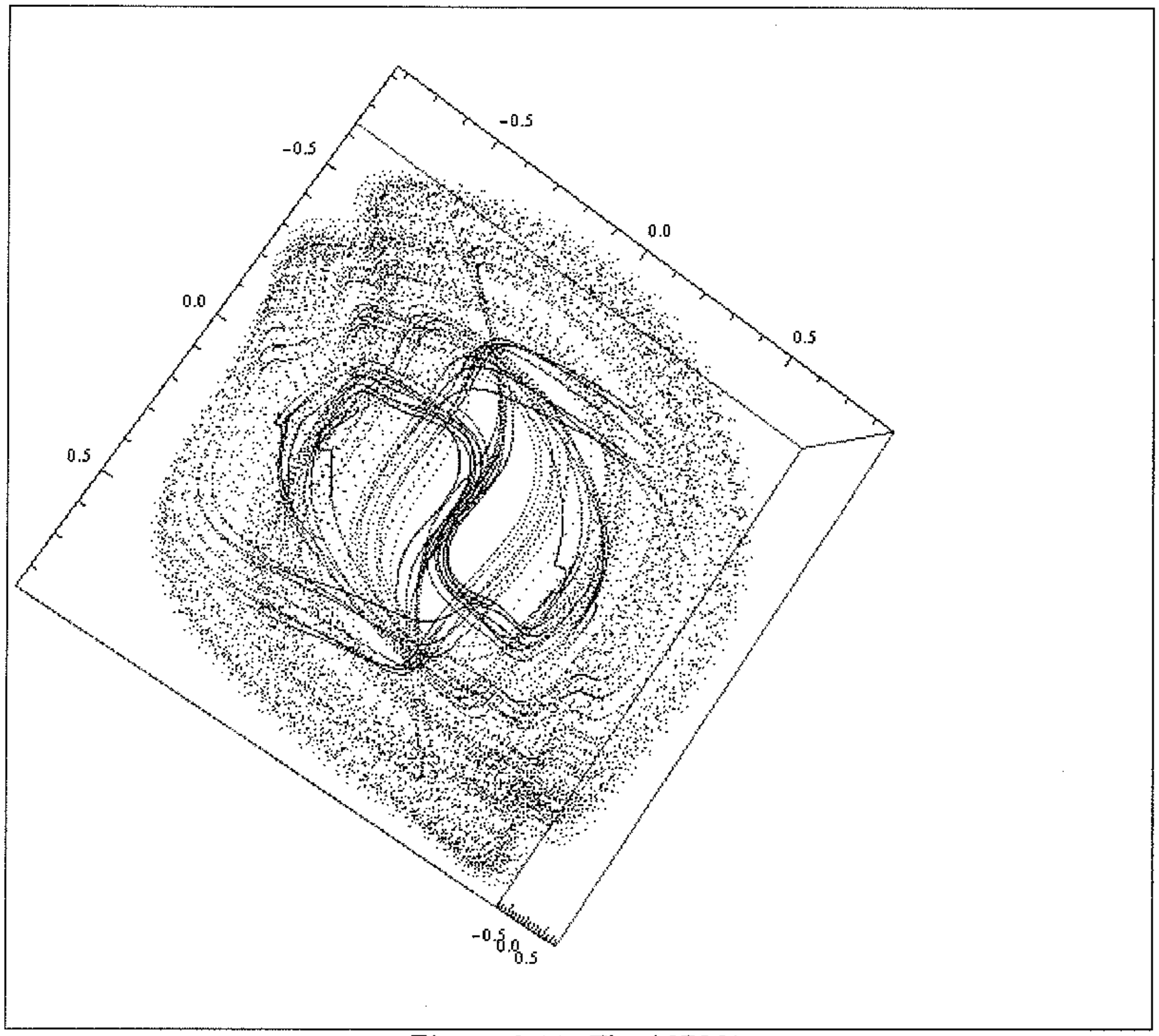

Figure 3.11: The A3DM4.

Example 3.2.4. For the A3DM4, we have the equation

$$\begin{aligned} x_1 &= y_o \\ y_1 &= z_0 \\ z_1 &= 0.00833z_0 + y_0^{2/3} - x_0 \end{aligned} \tag{3.9}$$

with initial conditions of $x_0 = 0.43402$, $y_0 = 0.23861$, and $z_0 = 0.07611$. We ran this system for 1,000,000 time steps and obtained a positive Lyapunov exponent of $\lambda = 0.679372$. This system clearly displays divergence of nearby trajectories, and is therefore chaotic.

3.3 T–Networks

Box-counting dimension algorithms are set-up in such way that they are minimalist inducing programs. This means that the algorithm is structured in such a way that it counts many components as a single entity (as in the normal box-counting method), or at best the algorithms count each component on a one for one bases as in the information dimension algorithm. The algorithms employ a method using the philosophy of taking a localized incident and looking at it globally. This is a sufficient method for a number of problems; however, many real world situations and networks possess valuable information on the localized level that is neglected when employing globalized methods.

One of our current projects, which utilizes many of the techniques for analyzing chaotic dynamical systems that we have learned so far, is the study of t-networks. These are a special class of networks that can be used to model disease spread, information flow, social networks, and, in particular terrorist networks, hence the name. Using a terrorist model as the bases for our model, we will display the short comings of the dimension calculation methods and offer a solution to these short comings.

Let us say we are interested in a network model based on a terrorist organization. Furthermore, let there be 10 terrorist in this one particular cell of terrorist labeled $T_1, T_2, \ldots, T_{10}$ such that we allow connections between different T_n, but no T_n is completely connected—meaning that no individual T_i is connected to every other T_n. Now the normal dimension procedure will either count each T_n separately or group all the T_n of this particular cell into one box and link it to a larger network consisting of multiple cells of terrorist. This method of treating a netwrok leads to important structural questions.

Question 5. *Given a complex network identified as a T-network, can we determine at what stage of its evolution it is in?*

Question 6. *Given a partial view of a complex network identified as a T-network, can we construct a mathematical measure that will allow us to discover the part we are not able to see? That is, can we detect that what we are seeing is only a partial view, and then decide what is missing?*

Definition 3.3.1. A T-network is a non-trivial network (that is, the edge set and the vertex set are both non-empty), with directed edges, and possibly weighted vertices. A T-network undergoes changes with time, so we define the evolution of a T-network to be the sequence of events of the network, indexed by a discrete or continuous time function. In its evolution, a T-network always begins connected, that is, at the initial point in time, the network has a single connected component. The evolution of a T-network is marked by fundamental changes in its structure which changes it from a connected component to a non-connected, non-trivial network. We are interested in understanding the process of this evolution as it holds important information of how the system develops.

Definition 3.3.2. A **network bifurcation** is a fundamental change in the network structure, such as change in its connectedness description (introducing additional components or disconnecting through a removal of an edge or a larger substructure), changes in its vertex or edge set, or in the case of oriented networks, a change in orientation of some of its edges. If the network is weighted, a change in the weight of a vertex would also qualify as a bifurcation. The T-network begins its life cycle as a trivial network; however, as it grows it can develop or bud into separate additional networks. Each of those separate networks begin their life cycle as a trivial network.

Definition 3.3.3. We would like to distinguish **trivial T-networks**, which consist of complete graphs of a few vertices, as well as other simple cycle graphs. Trivial T-networks will be the building blocks of larger T-networks. Trivial T-networks will also be referred to as cells. Examples of trivial networks can be seen in figure 3.12.

Definition 3.3.4. We would like to distinguish particular vertices in the case when the T-network is directed, which will likely be the most frequent instance. A **source vertex** or simply a **source** is a vertex with a positive outgoing degree and zero incoming degree, that is, all edges connected to the vertex are directed away from it. We also define a **receiver** or an **end vertex** a vertex for which, conversely to the source vertex, the incoming degree is positive and the outgoing degrees is zero, so that all edges connected to the end vertex point towards it.

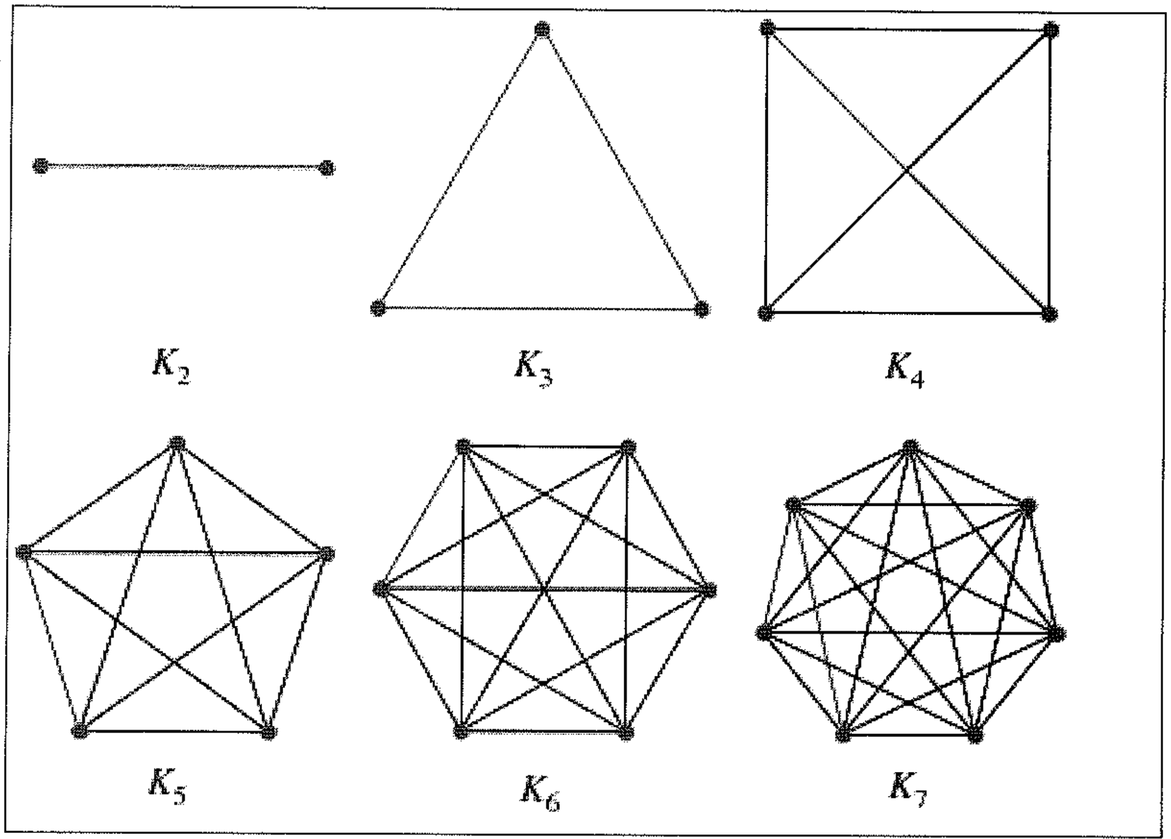

Figure 3.12: A sample search of trivial networks.

Lemma 3.3.5. *A non-trivial T-network cannot form a complete graph.*

In a terrorist network where each terrorist is denoted as a node labeled T, we will assume there is a minimum amount of nodes such that $T \geq 3$. We assume 3 to be the minimum number of nodes for T due to the logical assumption that there will be a "source" who issues orders, an intermediate who goes between the source and the other parties, and lastly the receiver who is the individual that carries out the orders and normally has minimal contact with all other parties within the cell. The reason a complete graph does not exist is due to the logical assumption that there would be no need for an intermediate.

If the T-network is complete, then there exist direct connections between all parties in the network. This means that there exists a graph of the T-network such that each node in the graph is connected to every other node. However, if there are direct connections to all nodes, then there is no need for an intermediate. Thus, the network becomes a trivial network.

Lemma 3.3.6. *At conception, the T-network has to form a complete graph.*

The critical piece of information that is overlooked by the normal dimension calculation methods is what happens on the local level. By expanding or zooming out, we miss important connections that exist for the cell members with individuals outside the cell. We label these individuals outside of the cell O_n. These individuals are not recognized as part of the terrorist organization for various reasons such as there is no information linking them to the cell. However, it is reasonable to assume that each T_n of the cell has at least one O_n if not multiple O_n. In order to discover the value of a member of the cell that includes his or her connections outside the cell we set up a random variable that fluctuates between a lower and upper bound such as $O \leq x \leq O_n$. Then we run a dimension calculation algorithm that counts individual nodes in conjunction with shifting variables for their connection outside the cell and see which node is most critical to the functioning of the network. Thus, we have a more accurate method for discovering critical nodes in a real world network model.

T-Network also has the benefit of being able to possibly predict the future. This seems doubtful at first until further consideration is given. Because T-Network is set up in a manner that allows for analysis of the visual entities as well as the invisible entities, it allows for possible scenarios that can take place in the future.

For instance, in a terrorist network $T_1, T_2, \ldots, T_n$ there is a leader who we will call the source and label T_1. The source issues all major commands and plans to be carried out by the other Ts. These orders might be issued directly to the other Ts, or the order could be handed down a chain of command through an intermediate so as to protect the identity of the source. Now, common belief would be that the source is the most important T in the network. This belief stems from the fable that if you remove the head of the snake, it can no longer bite you. Organizations still function when a leader is eliminated, although, it is granted that their progress is slowed. The best option is to eliminate the whole network; however, that is nearly impossible if not completely impossible.

New leaders pop-up to replace old ones and often once a new leader is established it is hard to eliminate him due to the obvious reason that he seclude himself for his own protection.

Sometimes these new leaders are not the second or third person in the chain of command, but can be the least likely individual and/or the last link in the command chain. This low level individual labeled T_L may get his position as the new leader because of the important connections he has made while being in the position of T_L. Once the T_L receives his new position as the source, it becomes very hard to eliminate him. Case in point, Bin-Ladden. It took many years for the U.S. Government to find and eliminate him, yet there is still more terrorist activity taken place, which points to the fact that eliminating a leader does not mean the organization will dissolve. However, what if there was a way to determine how important each T_n in the organization will be once the source is eliminated? Well, that is a possibility that T-Network allows for.

Because T-Network is set up with a random variable in conjunction with existing visible nodes, it allows for possible future scenarios to be played out. Iterating the system a number of times and observing the structural transformations that take place allows the analysis to view the shifting dynamics of the system. Theoretically, the T_P that produces that greatest shift in the network structure, after the source, holds the potential to become the new source when the existing one is eliminated.

The terrorist model is just one of the many models that the T-network model can be used to study. For instance, T-networks can also be used to study the development of a virus with in the body, or a viral outbreak among communities.

It is our goal to create a working algorithm that functions on the principles of the T-network model.

Bibliography

[1] E. Ott, *Chaos in Danamical Systems*, New York: Cambridge University Press, New York, NY, 1993.

[2] Gary L. Peterson, James S. Sochacki. *Linear Algebra and Differential Equations*, Addison Wesley, NY, 2002.

[3] J.A. Yorke, K.T. Alligood, and T.D. Sauer, *Chaos: An Introduction to Dynamical Systems*, Springer, New York, NY, 1996.

[4] J.C. Butcher, *The numerical analysis of ordinary differential equations, Runge-Kutta and general linear methods*, Springer,New York, NY,1996.

[5] J.C. Butcher, *The numerical analysis of ordinary differential equations*, Wiley, Chichester and New York, New York, NY, 2003.

[6] J.C. Sprott, *Automatic Generation of Strange Attractors*, Comput. and Graphics 17, No. 3 (1993), 325-332.

[7] Heinz-Otto Peitgen, Hartmut Jürgens, Dietmar Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer, New York, NY,2004.

[8] H.Binous,, N.Zakia, *An Improved Method for Lyapunov Exponents Computation*, Department of Chemical Engineering National Institute of Applied Science and Technology.

[9] Morsis W. Hirsch, Stephen Smale, Robert L. Devaney. *Differential Equations, Dyanmical Systems and An Introdution to Chaos Second Edition.* Elservies Academic Press, New York, 2004.

[10] M.Sandri, *Calculation of Lyapunov Exponents*, The Mathematical Journal 6 Issue 3 (1996), 78-84.

[11] O. Shanker *Defining Dimension of a Complex Network* Modern Physics Letters B21 (2007), 321-326

[12] P. Grassberger, *On the Fractal Diemnsion of the Hénon Attractor*, Physics Letters 97A (1983), 224-226.

[13] P. Grassberger and I. Procaccia, *Characterization of Strange Attractors*, Phys.REV.Lett¿ 50 (1083), 364.

[14] S.H. Strogatz, *Nonlinear Dynamics and Chaos: with Applications to Physics, Biology, Chemistry, and Engineering, L.L.C.*, New York, NY, 1994.