Applications of Graph Theory to the Analysis of Chaotic Dynamical Systems and Complex Networks

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

by
Anderson Grant

Annandale-on-Hudson, New York
December, 2012

Abstract

This work investigates and create applications of graph theory in the study of chaotic dynamical systems. The visibility algorithm, which provides a way of generating a graph from a 1-dimensional time series, is generalized to higher-dimensional time series, where different notions of multi-dimensionla visibility are defined and studied. New methods of analysis, such as Poincaré decomposition of time series, are defined and studied. Mathematica routines are developed for calculating and visualizing some of these new definitions in the context of known chaotic systems.

Contents

List of Figures

Dedication

Acknowledgments

1
Background

1.1 Introduction to Chaos

Chaos, a term that is still not clearly defined, is part of a larger field known as dynamics. The field of dynamics was initially a branch of physics, which dates back to the mid-1600s and deals with systems and their evolution through time. Although many renowned scientists, such as Isaac Newton, have studied this field, no real breakthroughs came until the 1800s when Poincaré developed a novel geometric method to answer the three-body problem. Poincaré realized that certain deterministic systems exhibited non-periodic behavior that depended heavily on the initial conditions of the dynamical system. From this finding, Poincaré also realized that within such systems, long-term predictions were impossible. Despite this remarkable discovery, the field of chaos languished until the second half of the twentieth century ([11]).

With the advent of the high-speed computer, researchers possessed a tool that enabled them to experiment with equations that were impossible to evaluate before. These experimental calculations led to new discoveries such as the Lorenz strange attractor. The most surprising of these discoveries came from a physicist named Feigenbaum. This physicist

discovered that the transition from regular behavior to chaotic behavior in a system was governed by certain universal laws ([9, 13]).

Although chaos is often thought to refer to randomness and lack of order, it is more accurate to think of it as an apparent randomness that results from complex systems and interactions among systems. According to James Gleick, author of *Chaos: Making a New Science*, chaos theory is "a revolution not of technology, like the laser revolution or the computer revolution, but a revolution of ideas. This revolution began with a set of ideas having to do with disorder in nature: from turbulence in fluids, to the erratic flows of epidemics, to the arrhythmic writhing of a human heart in the moments before death. It has continued with an even broader set of ideas that might be better classified under the rubric of complexity."

Very recently, advanced techniques from graph theory that transform time series into networks have offered a view of chaos and its genesis in low-dimensional maps from an unusual perspective favorable for the discovery of novel features and new understanding that we explore further in this work.

1.2 Chaos and Dynamical Systems

Nonlinear dynamics is an area of science that provides the necessary tools, language, and context that enable us to analyze and talk about dynamical systems. We give brief definitions of the basic terms of nonlinear dynamics, and illustrate them in terms of classical examples of dynamical systems. Most of these can be found in [9–11, 13, 14].

A **dynamical system** is a part of the world which can be seen as a self-contained entity with some temporal behavior. In nonlinear dynamics, speaking about a dynamical system usually means considering and studying an abstract mathematical system which is a model for such a system. Mathematically, a dynamical system is defined by its state and by its dynamics.

The **state of a dynamical system** is a number or a vector (i.e., an ordered list of real numbers) defining the state of the dynamical system uniquely at each point in time.

The **phase space** associated with a dynamical system is the set of all possible states of the system. Each point in the phase space corresponds to a unique state.

The **dynamics** or the **equation(s) of motion** of a dynamical system is the causal relation between the present state and the next state in the development of the system. The deterministic rule (or the equations) of the system tells us what happens in the next time step. In the case of continuous time, the time step is infinitesimally small, and the equation of motion is a differential equation or a system of differential equations of the form $d\mathbf{v}/dt = \mathbf{F}(\mathbf{v})$, where $\mathbf{v}$ is the state variable and t is time. In the case of discrete time, the dynamics is given by an algebraic map of the form $\mathbf{v}_{n+1} = \mathbf{F}(\mathbf{v}_n)$, where time is discrete. We will consider a collection of examples of 1-dimensional, 2-dimensional, and 3-dimensional discrete and continuous systems.

What happens if the next state isn't uniquely defined by the present one? In general, this is an indication that the phase space is not complete, that is, there are important variables determining the state which are not accounted for. This is a crucial point while modeling real-life systems. There are two important classes of systems where the phase space is incomplete: the **non-autonomous systems** and **stochastic systems**. A non-autonomous system has an equation of motion which depends explicitly on time. Thus, the dynamical rule governing the next state not only depends on the present state but also on the time. In this case, we complete the phase space (and turn the system into an **autonomous system**) by simply including time into the definition of the state variables.

In a stochastic system, the number and the nature of the variables necessary to complete the phase space is usually unknown so the next state can not be deduced from the present one. The deterministic rule is replaced by a stochastic one. Instead of the next state, it gives only the probabilities of all points in the phase space to be the next state.

The **orbit** or the **trajectory** of a dynamical system is a solution of the equation(s) of motion associated with the system. In the case of continuous time, it is a curve in phase space parametrized by the time variable. For a discrete system it is an ordered set of points in the phase space.

The **flow** associated with a dynamical system is the mapping of the whole phase space of a continuous dynamical system onto itself for a given time step dt. If dt is an infinitesimally small time step, the flow is just given by the right-hand side of the equation of motion (i.e., $\mathbf{F}$). In general, the flow for a finite time step is obtained by numerical approximation methods.

A **characteristic set of a system** is a set of points in the phase space with the property that all trajectories that originate from a point of this set come arbitrarily close and arbitrarily often to any point of the set. There are four types of characteristic sets. The first are **fixed points**, they correspond to steady-state solutions. The second type are **limit cycles**, which correspond to periodic solutions of the system. The next type are the **quasi-periodic orbits**, which correspond to periodic solutions with mutually irrational frequencies (i.e., the ratio of the frequencies is an irrational number). Of particular interest is the fourth type, the **chaotic orbits**, which correspond to bound non-periodic solutions. These solutions occur in the driven pendulum if the driving force is strong enough. The first three types can also occur in linear dynamical systems. The fourth type appears only in nonlinear dynamical systems. Its existence was first anticipated by Henri Poincaré, and this irregular behavior was termed deterministic chaos.

In linear dynamics, we seek the fundamental solutions that generate all other solutions. In nonlinear dynamics, we want to find the qualitative behavior of the system, the special characteristic sets in the system phase space (fixed points, limit cycles, quasi-periodic or chaotic orbits). Which of them are stable? How they change when we vary a parameter of the system (a control parameter)? The appearance and disappearance of a characteristic

set is called a **bifurcation**. The change of the stability of a characteristic set element coincides with the emergence of a bifurcation ([13, 14]).

A fundamental feature of a chaotic dynamical system is the **sensitive dependence on the initial conditions**. This means that if two identical mechanical systems are started at initial conditions $\mathbf{x}$ and $\mathbf{x}+\epsilon$, respectively, where ϵ is a very small quantity, their dynamical behavior diverges very rapidly in phase space, their separation increasing exponentially on the average. The directions of divergence and shrinkage are different at different points in phase space. The net effect is that two closely spaced points are later found quite far apart. The exponential divergence of adjacent phase points has a further consequence for the chaotic attractor. In order that the trajectories of two adjacent phase points remain bounded without intersecting. They must fold back on themselves, producing a three-dimensional chaotic attractor with many layers (actually an infinite number) ([9–11]).)

How predictable is a chaotic system? As any deterministic system, it is predictable on a small time scale. But a large time scale, chaotic systems become unpredictable. Predictability is lost at a point in time that depends only logarithmically on the uncertainty of the initial conditions. There is an ultimate point beyond which we cannot foresee the future, it depends on the system and the inherent error in precision. For the weather, it may be roughly a week, and for our solar system with all planets, the prediction horizon lies a few million years in the future.

Sensitivity to initial conditions leads to chaos only if the attractor is bounded in phase space. Linear systems can have either sensitivity on the initial conditions or bounded trajectories, but not both, whereas nonlinear systems can exhibit both. The systems deforms phase space by folding it and rolling it, stretching it or contracting it, thereby sending nearby points further and further apart.

Chaotic motion is not a rare phenomenon. For a dynamical system described by a set of first order differential equations, necessary conditions for chaotic motion are that the

system has at least three independent variables, and that the equations of motion contain a nonlinear term. Three dimensions are sufficient to allow for divergence of trajectories, boundedness of the attractor to a finite region of phase space, and uniqueness of the trajectory. The effect of a nonlinear term is often to render a periodic solution unstable for certain parameter choices. While these conditions do not guarantee chaos, they do make its existence possible ([13, 14]).

The nonlinearity condition has probably been responsible for the late historical development of the study of chaotic systems. With the exception of some first order equations, nonlinear differential equations are either difficult or impossible to solve analytically, so the solution of nonlinear differential equations generally requires numerical methods. We use *Mathematica* routines to generate all of our diagrams, plots, tables, and graphs.

1.3 Methods of Dynamical Systems Analysis

There are a few fundamental techniques for analyzing dynamical systems that will be of particular use in this work.

The **Poincaré section** or **Poincaré map** is a carefully chosen (possibly curved) plane in the phase space that is crossed by the trajectory of the dynamical system. Even the term plane is being used loosely here. The Poincaré sections in this work will mostly consist of planes in phase space in 3-dimensional phase space, the Poincaré, and for 2-dimensional phase space, the Poincaré sections will be 1-dimensional curves, mostly lines. In general, they are subspaces of phase space that are one dimension smaller than phase space. Henri Poincaré developed this tool for the visualization of flows in higher-dimensional phase spaces. A Poincaré section turns a continuous dynamical system into a discrete one by plotting only points where the trajectory intersects the Poincaré section (in one direction). If the Poincaré section is carefully chosen, no essential information is lost concerning the qualitative behavior of the dynamics.

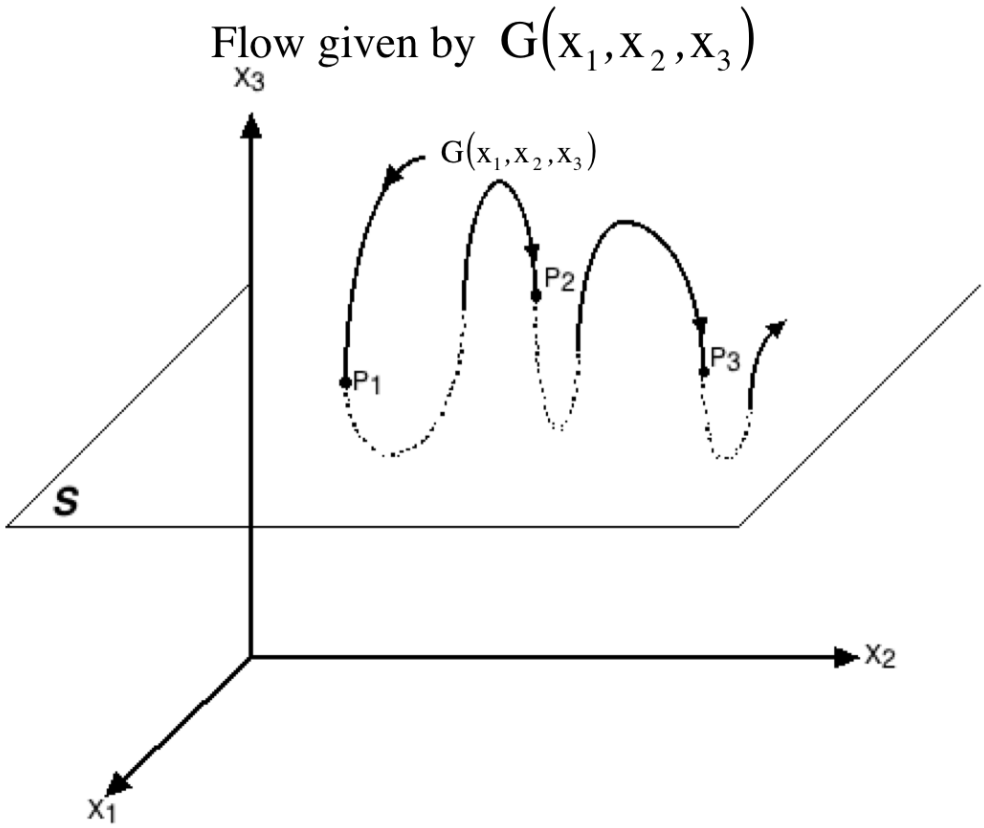


Figure 1.3.1: Poincaré section example, each time the trajectory intersects the Poincaré plane in one direction, the intersection point is recorded.

Note that Poincaré sections are a tool that takes a continuous system and turns it into a discrete system of equations of one or more lower dimensions. For systems that are discrete, they are already in a discrete form and Poincaré sections do not apply as a tool for analysis. More on Poincaré sections can be found in most standard texts on chaos, for example [9–11, 13, 14].

Another very important tool for the analysis of dynamical systems is the **bifurcation**. The structure of the attractors in a nonlinear dynamical system can change when a system parameter is changed, the change accompanied by a change in the stability of the attractor set. A bifurcation captures the nature of this change. In the examples in the following

section, we compute Poinaré sections and bifurcation diagrams, and demonstrate how they work.

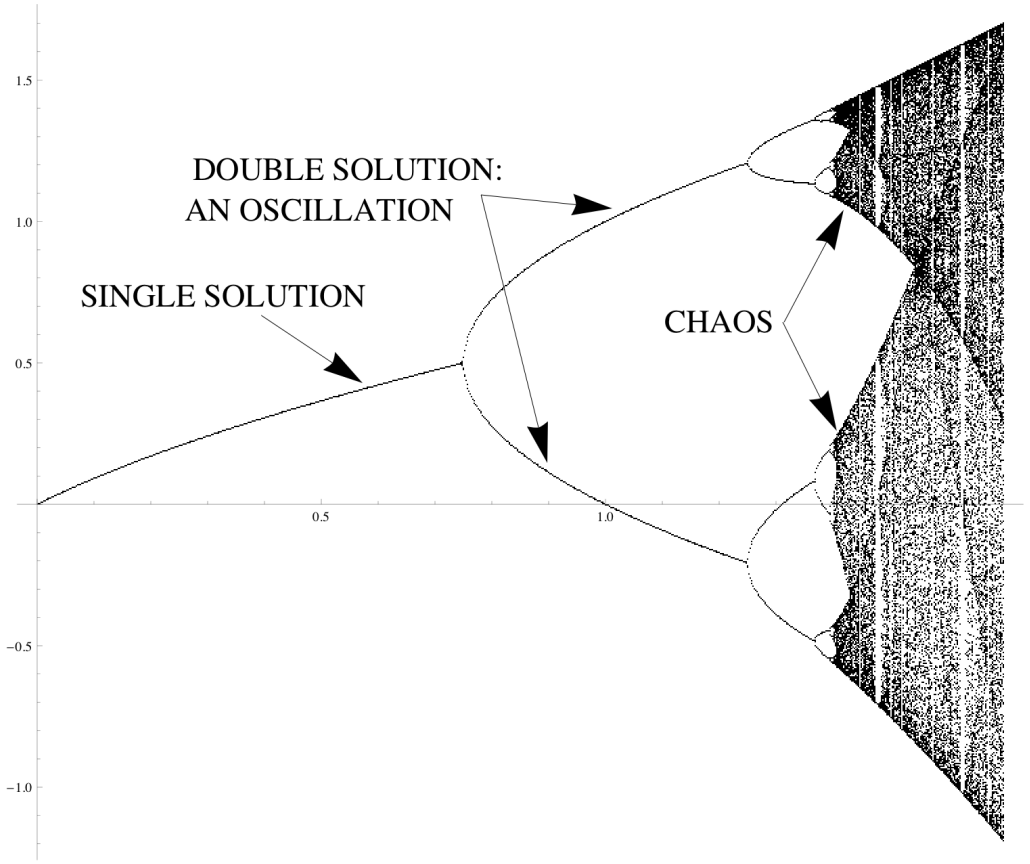


Figure 1.3.2: An example of a bifurcation diagram for the 1-dimensional quadratic equation $x_{n+1} = c - x_n^2$.This figure shows a plot of x_n (y-axis) versus c (x-axis), which is varied from 0 to 1.7 at a step of 0.002. For each value of c, the system is iterated some residual number of times until a steady behavior is reached, and then the next 500 iterations of the system are plotted. For instance, for values of c smaller than about 0.75, there is a unique solution. For values of c in the range $0.75 < c < 1.25$, there are two solution (so a period-doubling has occurred) and the diagram shows two solutions that the system oscillates between. Then another period-doubling gives four solutions that the system oscillates between. Eventually, the solutions grow to an infinite number, the onset of chaos.

Phase diagrams, Poincaré sections, and bifurcations provide information about the dynamics of a chaotic system. The dynamics may also be viewed more globally over a range of parameter values, thereby allowing simultaneous comparison of periodic and chaotic

behavior. The bifurcation diagram provides a summary of the essential dynamics of the system. More on bifurcations can be found in in most standard texts on chaos, for example [9–11, 13, 14].

Bifurcation diagrams and Poincaré sections are essential classical tools that we employ in combination with more recent techniques from graph theory and complex networks theory to further investigate the dynamics of a given system.

1.4 1-Dimensional Examples

We build a library of examples that we will use in order to test our methods. Where appropriate, we include a bifurcation diagram example and/or a Poincaré section example.

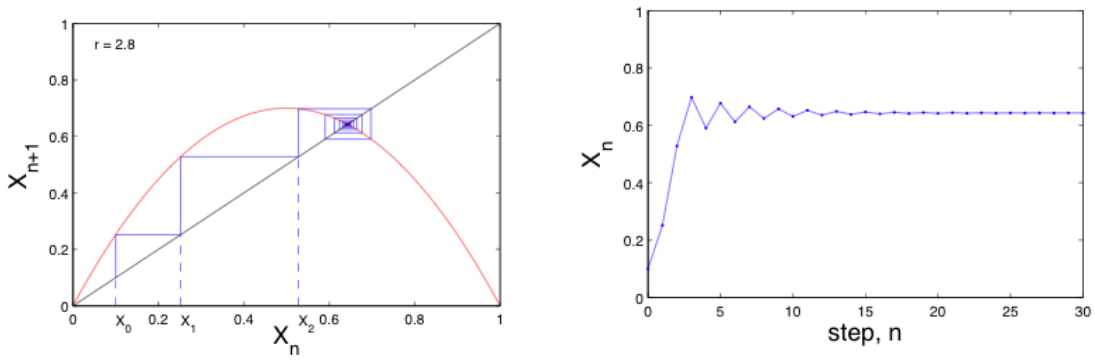
The **logistic map** inherits its name from the right-hand side of the **logistic differential equation** $dy/dx = rx(1-x)$. So it is given by the map $y = rx(1-x)$, where the parameter a satisfies $0 \leq a \leq 4$. This map is also known as an example of a **quadratic map**. The function $f(x) = rx(1-x)$ maps the unit interval into itself for $0 \leq x \leq 1$, which ensures that the system is bounded. We will use **the discrete version of the logistic map**, where the difference equation is given by

$$x_{n+1} = rx_n(1 - x_n).$$

The solutions to the logistic differential equation are determined by the fact that $x = 1$ is a stable equilibrium point, while $x = 0$ is unstable, that is, solution curves converge to $x = 1$ and are repelled from $x = 0$. This is true for any positive value of the parameter r. Unlike the case of the logistic differential equation, whose behavior is relatively insensitive to the particular value of a, the behavior of the discrete logistic map changes dramatically with r. This is the subject of our study here. More on the logistic equation can be found in [9–11, 13, 14].

We have an equilibrium point at $x = 1 - 1/a$. The derivative of f at that point is $\frac{df}{dx}(x_0) = r(1 - 2x_0) = r(1 - 2(1 - 1/r)) = -r + 2$ so $|f'(x_0)| < 1$ if $|2 - r| < 1$. So this equilibrium point is stable if $1 < r < 3$ and it is unstable for $r \in (-\infty, 1) \cup (3, \infty)$. This can be derived for the logistic map by setting $x_{n+1} = x_n = rx_n(1 - x_n) = x_s$, where x_s denotes the stable solution, and then solving $x_n = rx_n(1-x_n)$ to obtain $x_s = x_n = 1-1/r$. To find the fixed points numerically, we fix a value for the parameter a, then start with a random initial point, and iterate the discrete system enough until we reach stability.

We examine a simple way of visualizing solutions of first-order difference equations like the logistic map. In Figure 1.4.1a, depicting x_{n+1} versus x_n, we first note that the logistic map is a concave-down parabola $ax_n(1-x_n)$ passing through $x = 0$ and $x = 1$ and with a maximum at $x = 1/2$. For any given initial value x_0, we use the line $x_{n+1} = x_n$ to find the next value x_1 by going vertically to the quadratic curve, then going horizontally to the line $x_{n+1} = x_n$ and then vertically to the parabola for the next value. This is a **recursive graphical method** for the determining the value at each step n ([11, 13, 14]).



(a) The curve $x_{n+1} = ax_n(1 - x_n)$ and a sequence of points $\{x_0, x_1, \dots\}$ starting at x_0.

(b) The sequence $\{x_0, x_1, \dots\}$ versus the time step n, the sequence approaches a fixed value.

Figure 1.4.1: A graphical method for generating solutions from a recursive (discrete) equation, the logistic equation $x_{n+1} = rx_n(1 - x_n)$. Starting with an initial value x_0, and bouncing between the parabola$x_{n+1} = rx_n(1 - x_n)$ and the line $x_{n+1} = x_n$, the sequence approaches a fixed value, which is the point that satisfies both the logistic map and the equation $x_{n+1} = x_n$ ([11, 13, 14]).

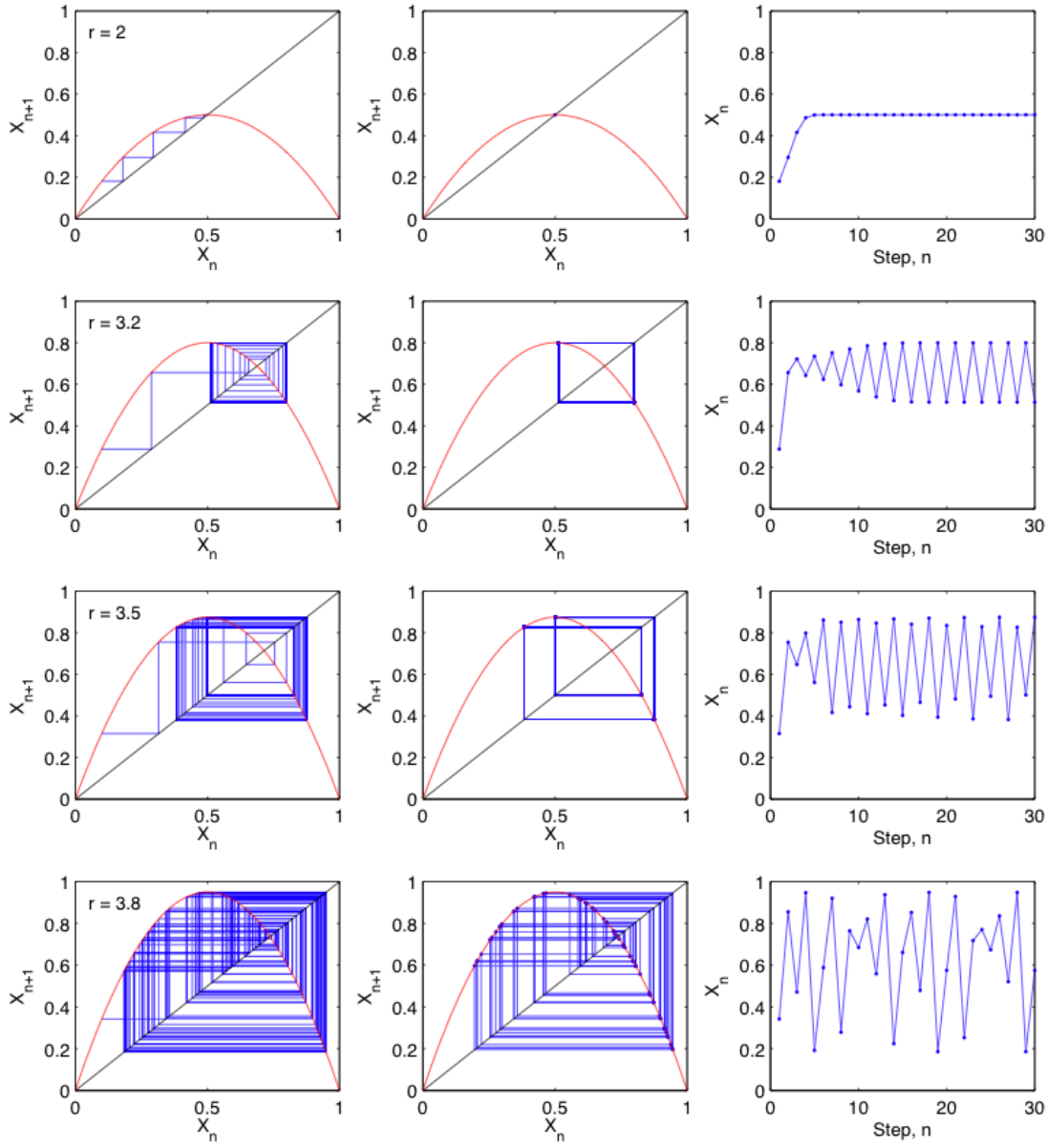
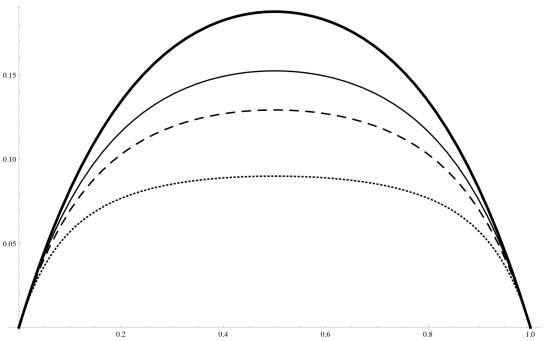


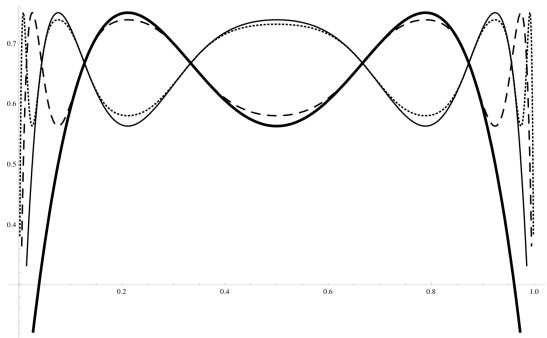
Figure 1.4.2: Graphical view of several sequences corresponding to different values of the parameter r are shown. Notice that as r increases from 2 to 4, the system's fixed points change from a single point, which is constant long-term behavior to a double fixed point which is an oscillation, then higher-order period oscillations, and finally, chaos. (reprinted from [13]).

When the parameter r increases, this effectively increases the steepness of the parabola, which makes the slope of this tangent steeper, so that eventually the stability condition is

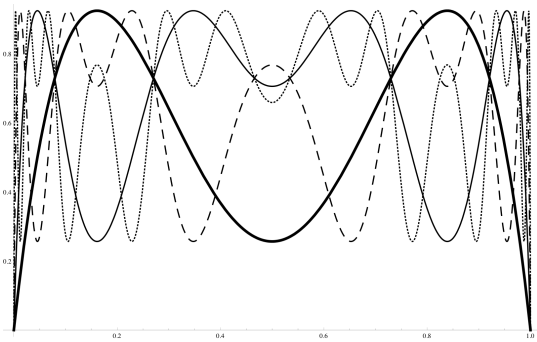
violated. The steady state then becomes unstable and the system undergoes oscillations, leading to chaotic behavior, as mentioned above.



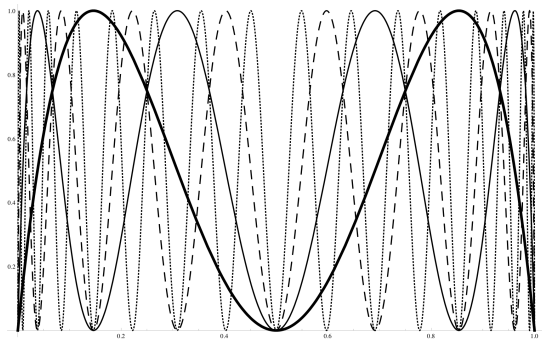
(a) One, two, three, and four iterations of the system $x_{n+1} = rx_n(1 - x_n)$ for $r = 1$. Each is a function of x_0 on the unit interval.



(b) One, two, three, and four iterations of the system $x_{n+1} = rx_n(1 - x_n)$ for $r = 3$. Each is a function of x_0 on the unit interval.



(c) One, two, three, and four iterations of the system $x_{n+1} = rx_n(1 - x_n)$ for $r = 3.7$. Each is a function of x_0 on the unit interval.



(d) One, two, three, and four iterations of the system $x_{n+1} = rx_n(1 - x_n)$ for $r = 4$. Each is a function of x_0 on the unit interval.

Figure 1.4.3: The first four iterations of the system $x_{n+1} = rx_n(1 - x_n)$. $g_0 = rx_0(1 - x_0)$, $g_1 = r^2(1 - x_0)x_0(1 - rx_0 + rx)0^2)$, etc. Each is a function of x_0, and is plotted in different format over the unit interval. With the increase of r, even the first few iterations become very unstable.

Figure 1.4.4 below shows a bifurcation diagram of the logistic map obtained by plotting x_n as a function of r for a series of values for obtained by starting with a random value x_0, iterating many times, and discarding the first points corresponding to values before the iterates converge to the attractor. In other words, the set of fixed points of x_n corresponding to a given value of r are plotted for values of r increasing to the right.

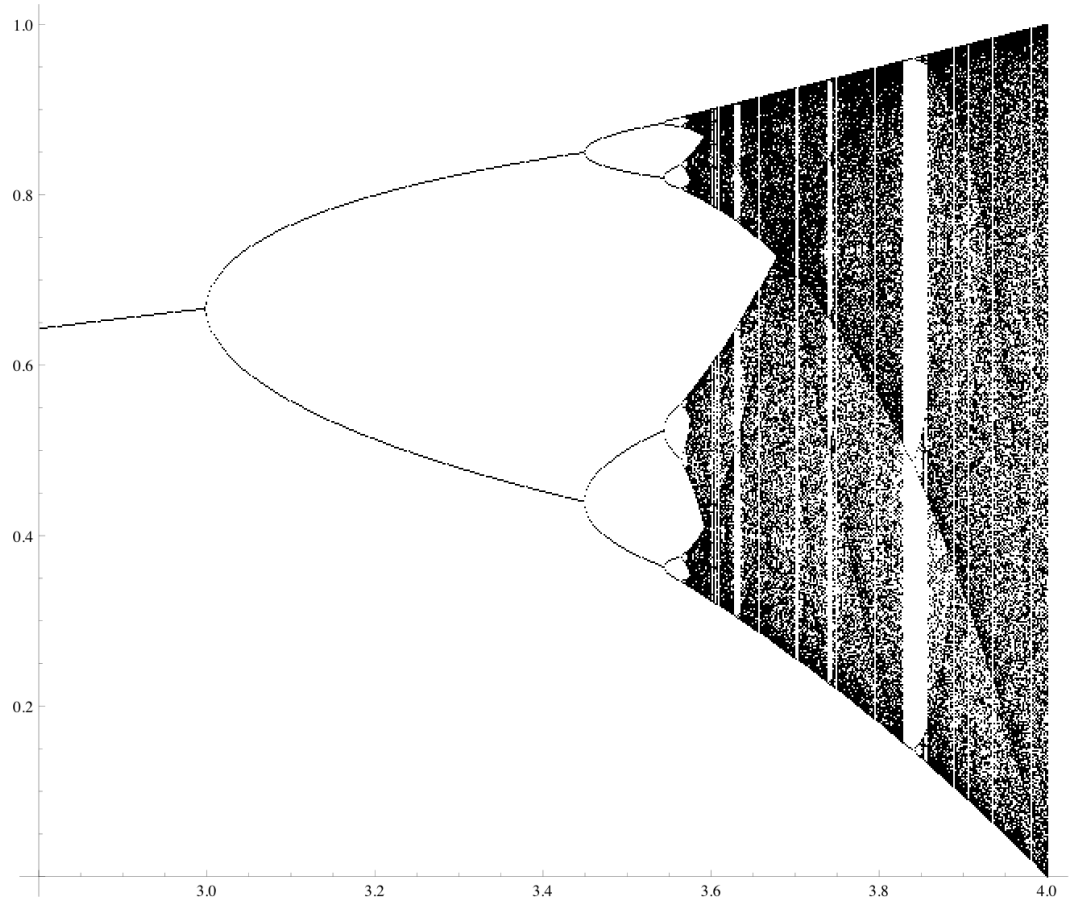


Figure 1.4.4: Bifurcation diagram for the fixed points of the **logistic map** $x_{n+1} = rx_n(1 - x_n)$ versus r generated with *Mathematica*.

Another map we will consider is the **cubic map** ([13]) defined by

$$f(x) = rx(1 - x^2),$$

where r is the parameter and the critical point at $r = 1$ is the first period-doubling instance, where the map bifurcates from the stable trivial solution $x = 0$ at $r = 1$. We will use the discrete version of this map,

$$x_{n+1} = rx_n(1 - x_n^2).$$

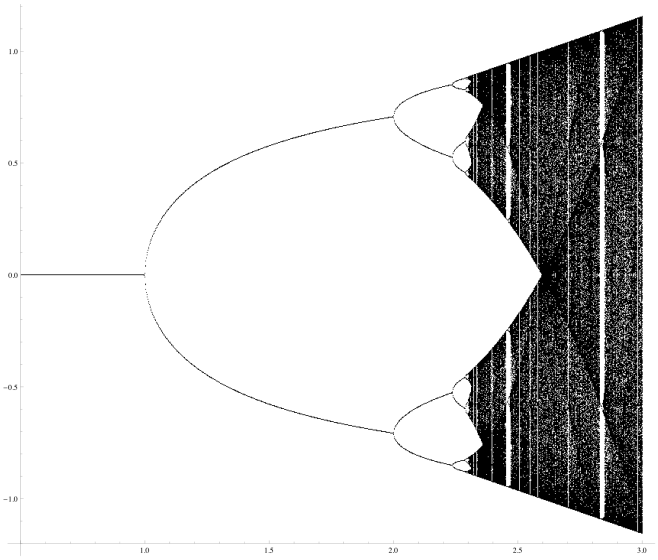


Figure 1.4.5: Bifurcation diagram for the fixed points of the **cubic map** $x_{n+1} = rx_n(1-x_n^2)$ versus r generated with *Mathematica*.

Our next example is the **quartic map** ([13]) given by the equation $x_{n+1} = rx_n(1-x_n^3)$.

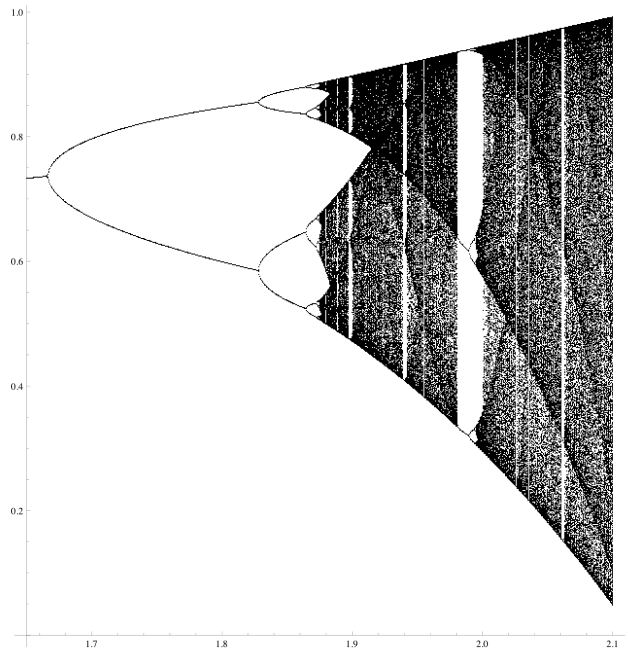


Figure 1.4.6: Bifurcation diagram for the fixed points of the **quartic map** $x_{n+1} = rx_n(1-x_n^3)$ versus r generated with *Mathematica*.

1.5 2-Dimensional Examples.

The first 2-dimensional map we will consider is the **Burgers map** ([11]) defined by the following (discretized) equations.

$$x_{n+1} = (1-a)x_n - y_n^2, \quad y_{n+1} = (1+b)y_n + x_n y_n,$$

where a and b are non-zero parameters.

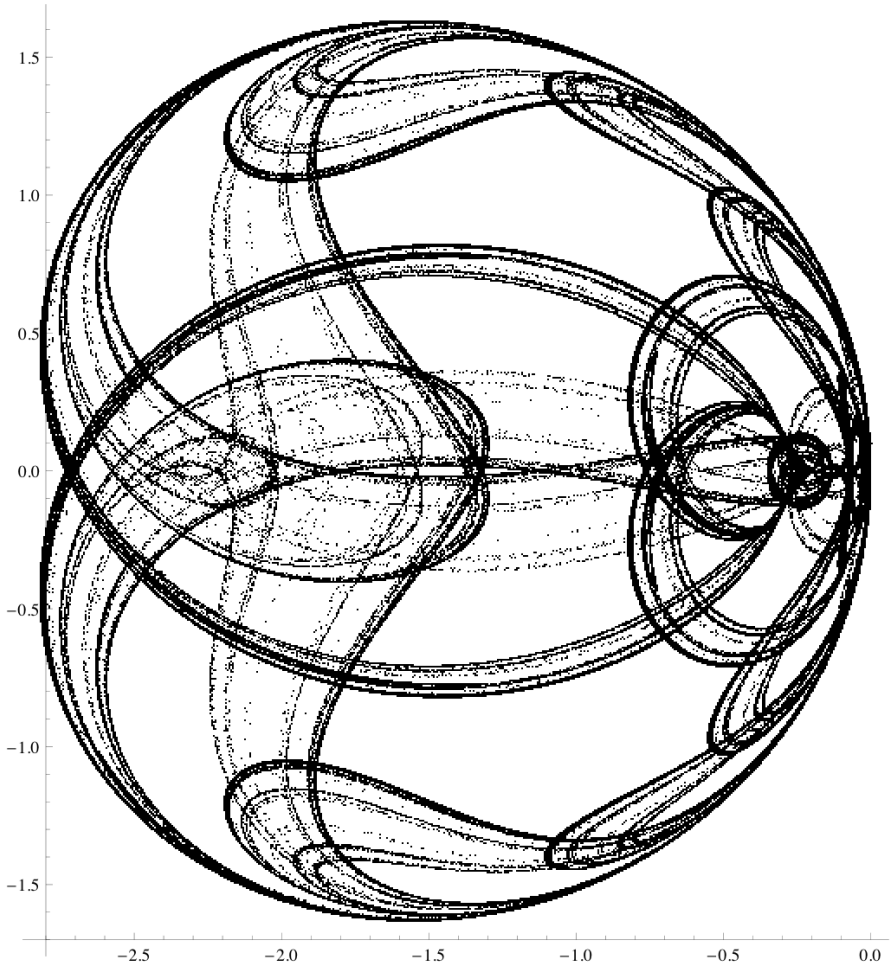


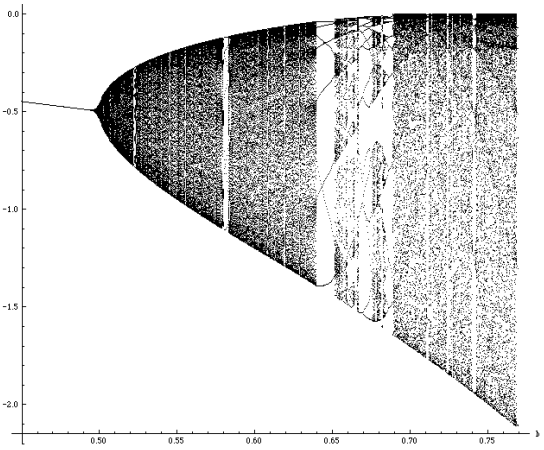
Figure 1.5.1: The Burgers map with $a = 0.9$ and $b = 0.865$.

There are two fixed points $(-b, \pm\sqrt{ab})$, which are stable for $b < 5$. For $a = 0.9$ and $b = 0.856$, the system exhibits quasi-chaotic behavior.

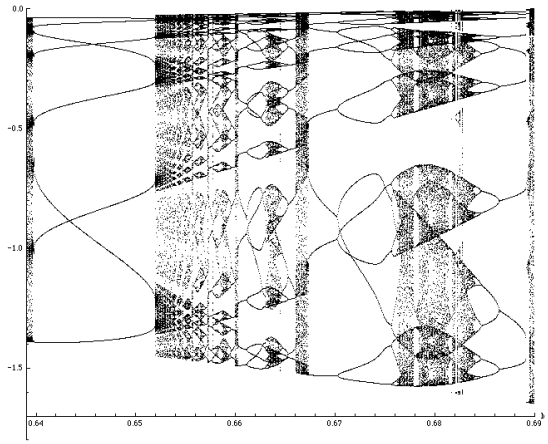
We will be interested in considering the continuous analogue of the Burgers map, the **continuous Burgers map** given by

$$x' = (1-a)x - y^2, \quad y' = (1+b)y + xy.$$

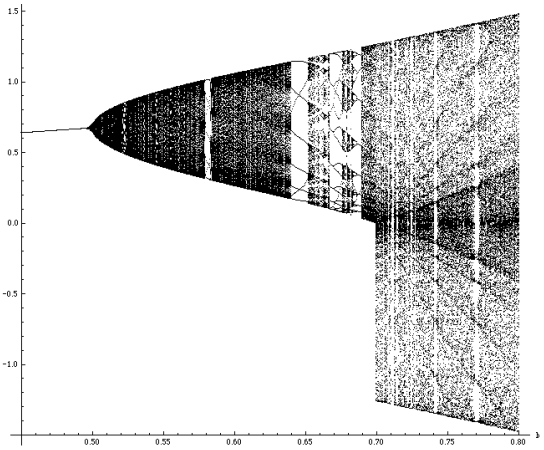
To integrate the system, we use a 4-step Runge-Kutta approximation algorithm ([2]).



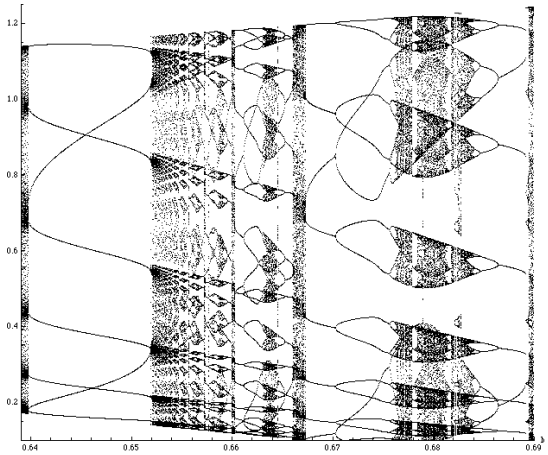
(a) Bifurcation diagram for the Burgers map x_n versus b with $a = 0.9$ and b varying between 0.45 and 0.8.



(b) Bifurcation diagram for the Burgers map x_n versus b with $a = 0.9$ and b varying between 0.63 and 0.69.



(c) Bifurcation diagram for the Burgers map y_n versus b with $a = 0.9$ and b varying between 0.45 and 0.8.



(d) Bifurcation diagram for the Burgers map y_n versus b with $a = 0.9$ and b varying between 0.63 and 0.69.

Figure 1.5.2: Bifurcation diagrams for Burgers map, with one parameter fixed $a = 0.9$ and the other varying $0.45 \leq b \leq 0.8$, the top two figures show x_n versus b and the bottom two show y_n versus b. The first 1000 iterations are thrown out and then 300 points are recorded as b is varied with a step size of 0.00001.

The next map in our library is the well-studied **Hénon map**, one of the most famous examples of dynamical systems that exhibit quasi-chaotic behavior ([14]). It is a discrete dynamical system. The system is given by the following set of equations.

$$x_{n+1} = y_n + 1 - ax_n, \quad y_{n+1} = bx_n.$$

The map exhibits sensitivity to initial conditions for $a = 1.4$ and $b = 0.3$. For these values, the Hénon map is chaotic (more precisely, quasi-chaotic). The parameter b is a measure of the rate of area contraction (dissipation), and this contraction is the same everywhere on the attractor. For $b = 0$, the Hénon map reduces to the quadratic map.

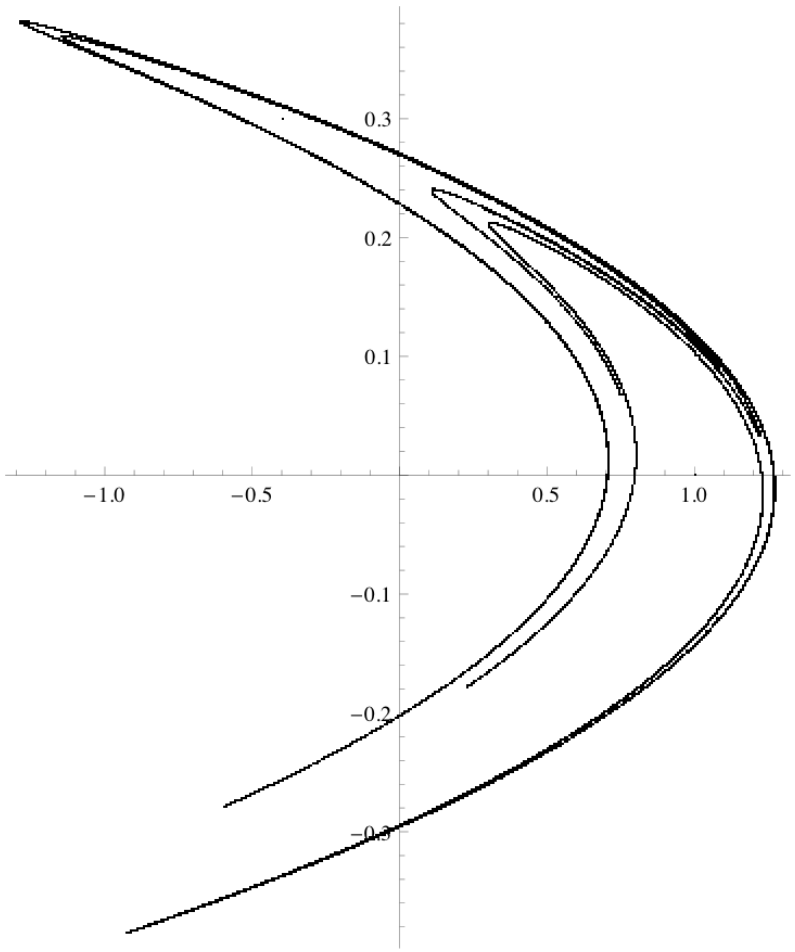


Figure 1.5.3: The Hénon attractor with $a = 1.4$ and $b = 0.3$.

Bounded solutions exist for the Hénon map over a range of a and b values, and some yield chaotic solutions as in the case of $a = 1.4$ and $b = 0.3$.

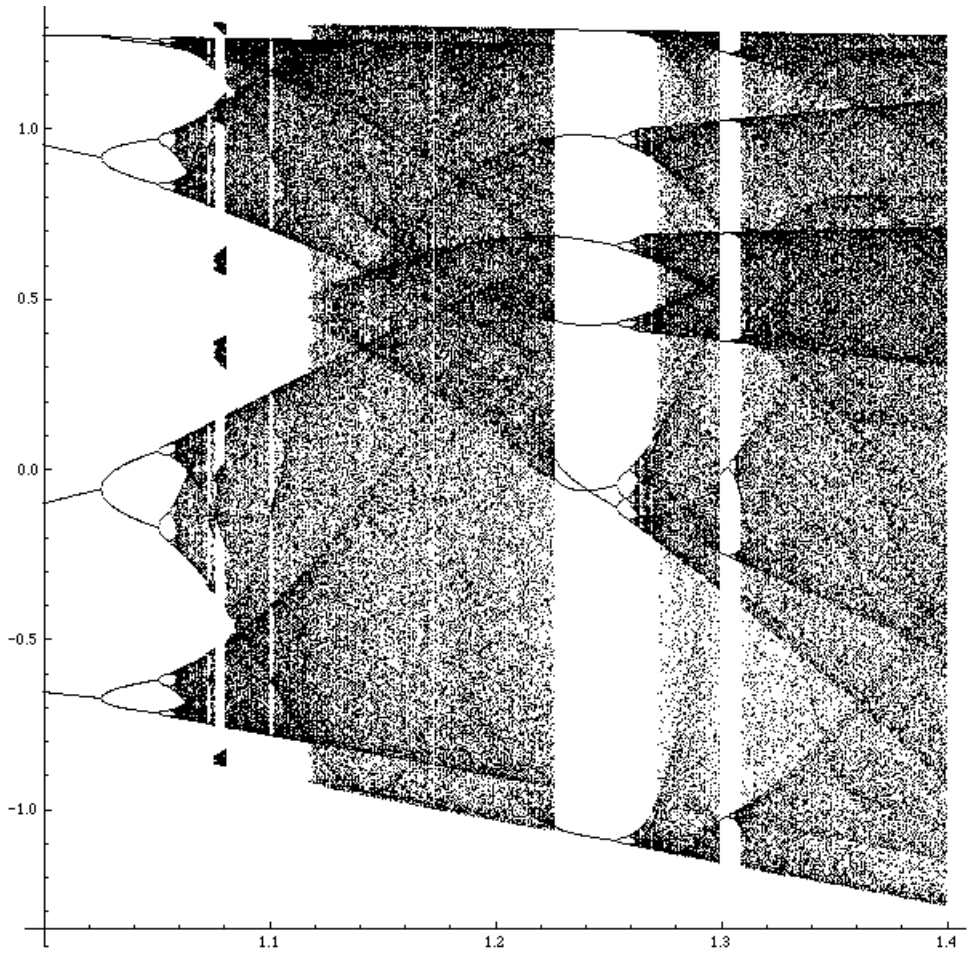


Figure 1.5.4: Hénon bifurcation diagram in $a - x_n$ plane for $b = 0.3$.

We will be interested in considering the continuous analogue of the Hénon map, **the continuous Hénon map** given by

$$x' = y + 1 - ax, \quad y' = bx.$$

To integrate the system, we use the 4-step Runge-Kutta approximation algorithm.

1.6 3-Dimensional Examples

The first 3-dimensional system we will be working with is the Lorenz system, one of the most well-known systems of ordinary differential equations ([9, 11, 13, 14]).

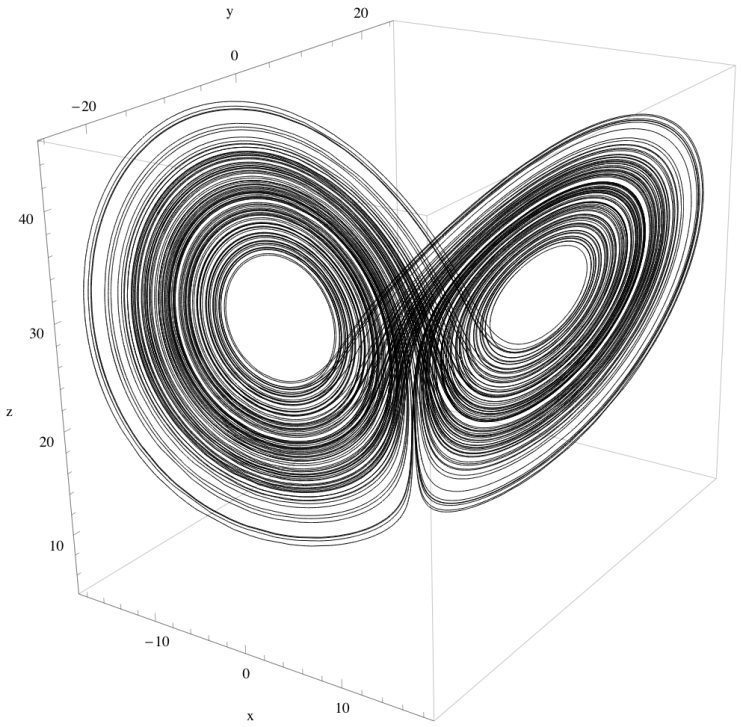


Figure 1.6.1: The Lorenz map with parameters $\sigma = 10, b = \frac{8}{3}, r = 28$.

It exhibits chaotic solutions for certain parameter values and initial conditions. The equations for the Lorenz map are given by

$$x' = \sigma(y - x), \quad y' = rx - y - xz, \quad z' = xy - bz,$$

where x, y, z are all functions of t, and $\sigma, r, b > 0$ are all parameters. Higher-dimensional chaotic systems give us a chance to study higher-dimensional bifurcations, which is one of out current projects. We consider an example of a planar Poincare section of Lorenz.

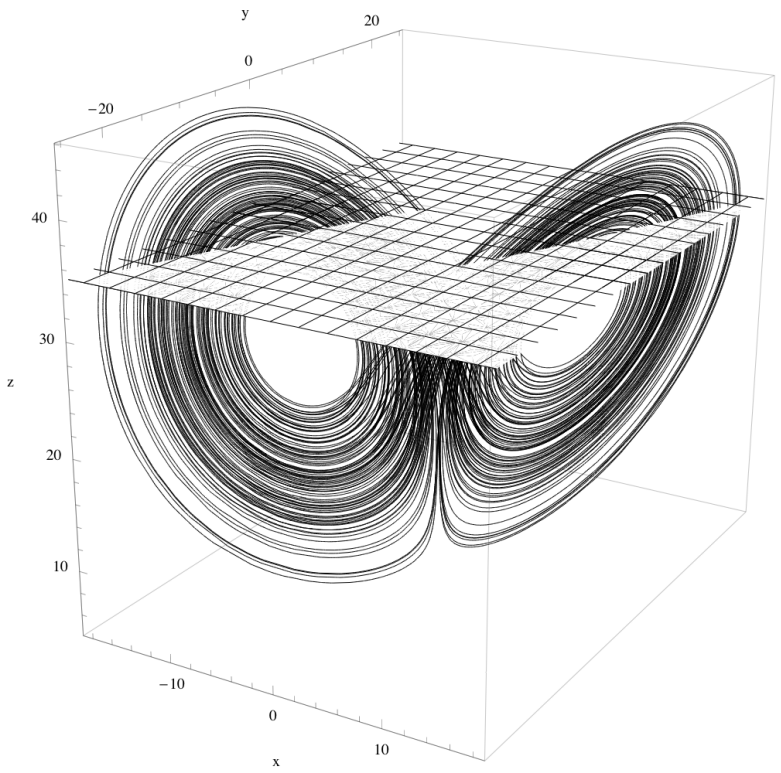
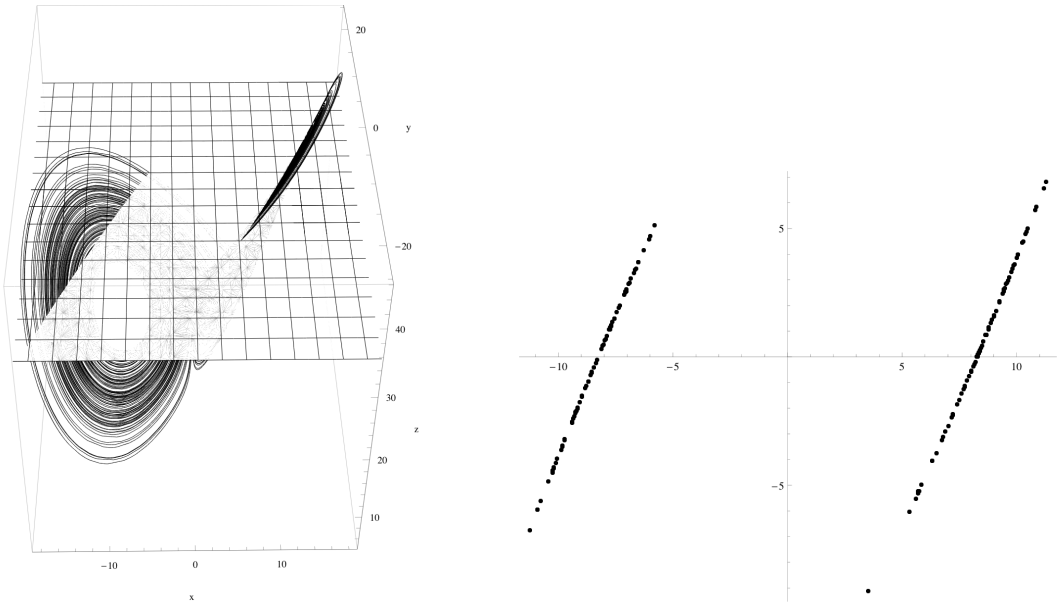


Figure 1.6.2: Lorenz with a Poincaré plane.

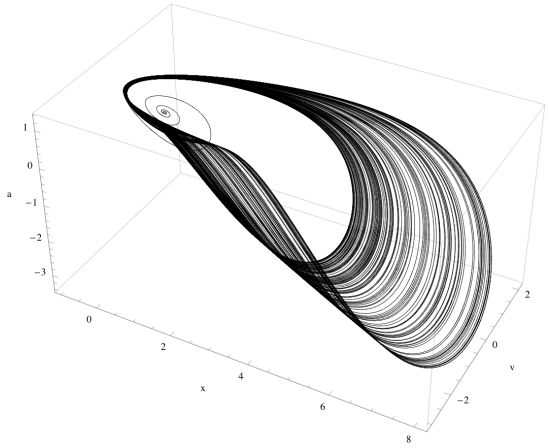


(a) Different view of the Poincaré section.

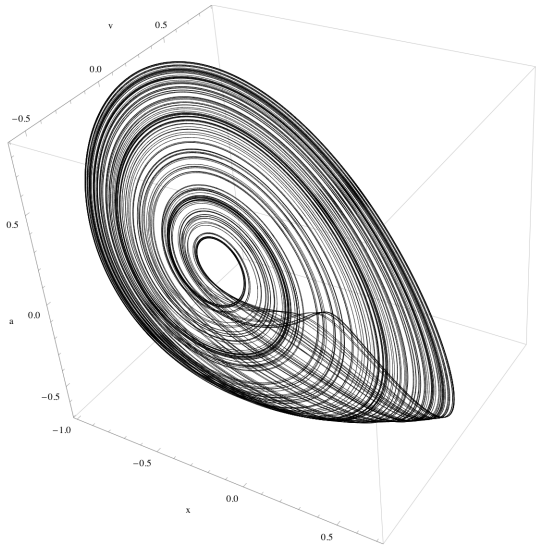
(b) The Poincareé section

Figure 1.6.3: The image of the actual Poincaré section was generated using a *Mathematica* routine that integrated the Lorenz system using a 4-step Runge-Kutta system and plotted each place the trajectory crossed the Poincaré plane in a given direction.

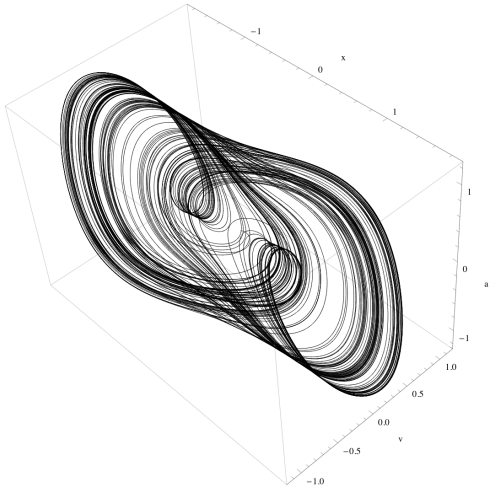
Lastly, we consider four 3-dimensional systems discovered by Sprott in [12]. In his search, he looked at nonlinear polynomial systems with one or two parameters.



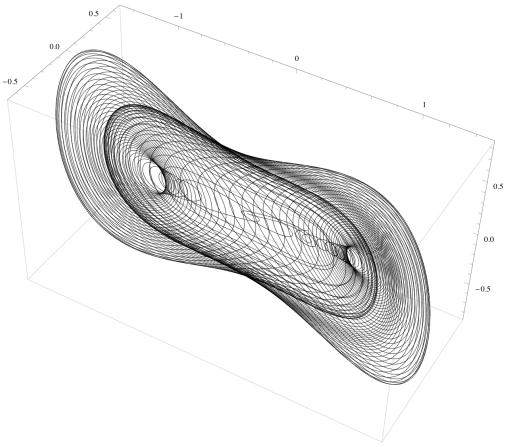
(a) Sprott Eq. 9 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa + v^2 - x$, where $A = 2.017$.



(b) Sprott Eq. 15 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa - v + x^2 - B$, where $A = 0.5$ and $B = 0.25$.



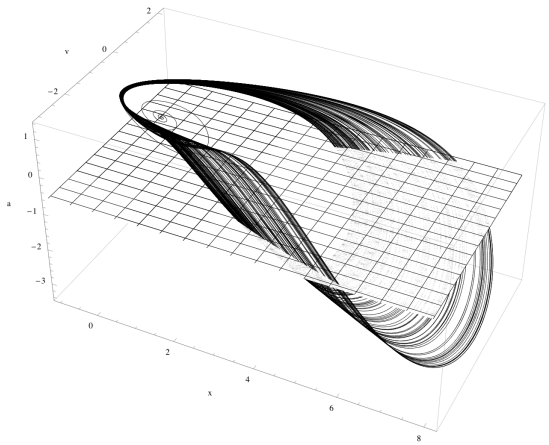
(c) Sprott Eq. 21 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -a^3 - vx^2 - Ax$, where $A = 0.25$.



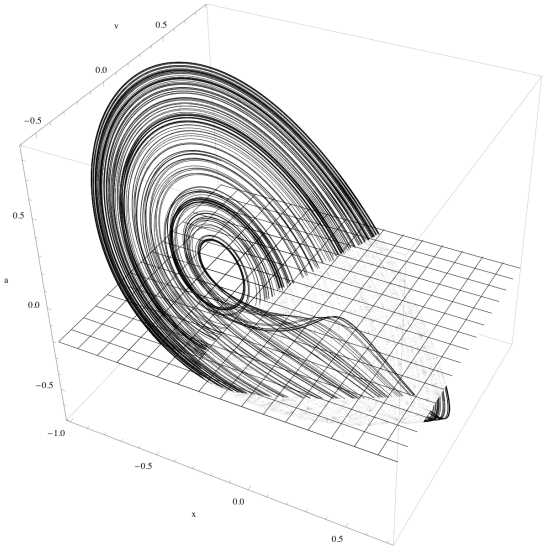
(d) Sprott Eq. 23 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -va^2 + A(1 - x^2)x$, where $A = 0.01$.

Figure 1.6.4: Four 3-dimensional chaotic systems discovered by Sprott ([12]). We label them Sprott Eq. 9, Sprott Eq. 15, Sprott Eq. 21, and Sprott Eq. 23, according to their labels in the original paper. We generated trajectories using a 4-step Runge-Kutta method with *Mathematica*.

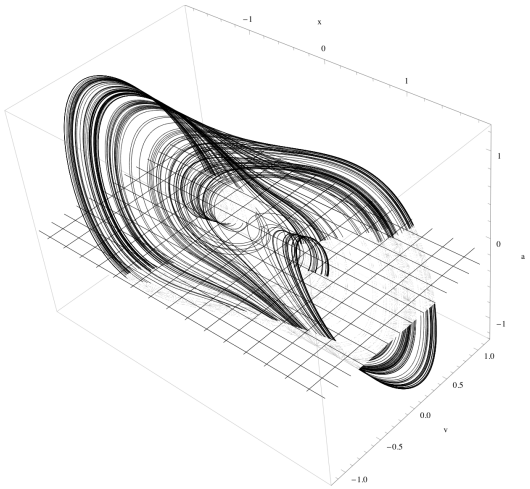
We take a look at Poincaré sections of the Sprott systems. For convenience, we have given examples of horizontal Poincaré planes, i.e., with equations of the form z=constant.



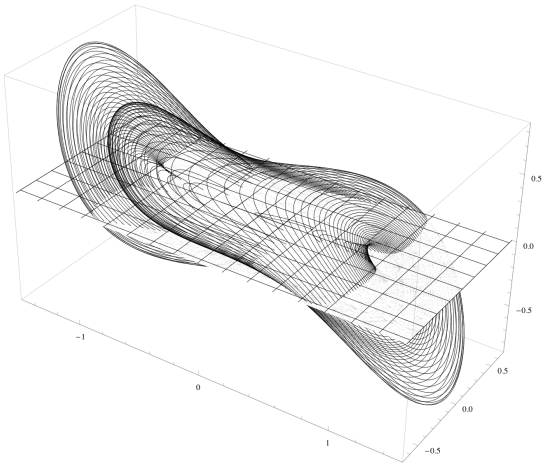
(a) Sprott Eq. 9 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa + v^2 - x$, where $A = 2.017$ with a Poincaré plane.



(b) Sprott Eq. 15 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa - v + x^2 - B$, where $A = 0.5$ and $B = 0.25$ with a Poincaré plane.



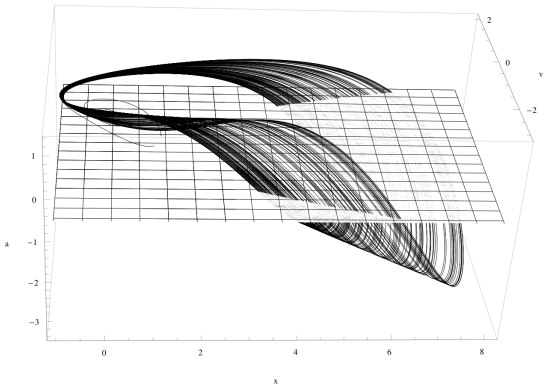
(c) Sprott Eq. 21 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -a^3 - vx^2 - Ax$, where $A = 0.25$ with a Poincaré plane.



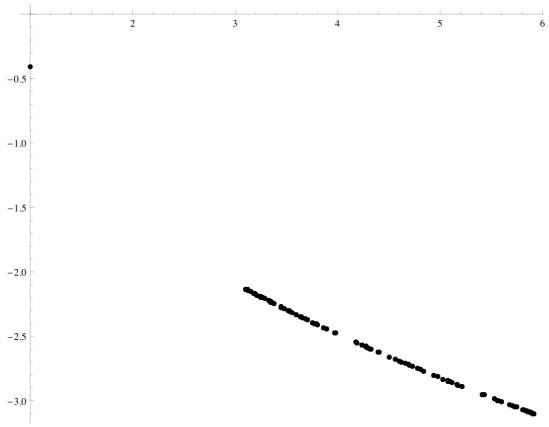
(d) Sprott Eq. 23 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -va^2 + A(1 - x^2)x$, where $A = 0.01$ with a Poincaré plane.

Figure 1.6.5: The Sprott systems with Poinacré planes.

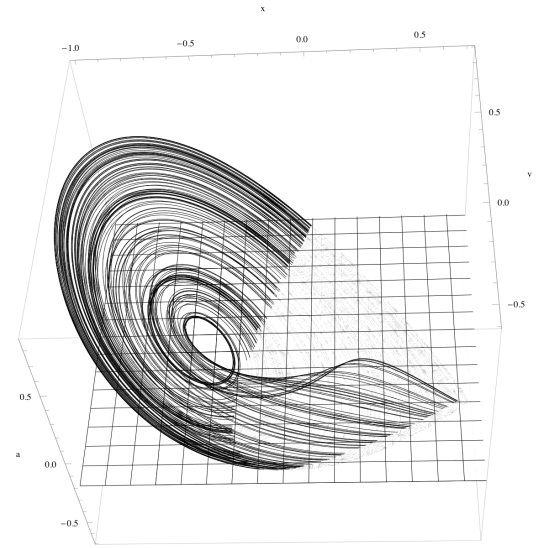
Lastly, we compute the Poincaré sections and show them next to the corresponding Sprott attractor map displayed at an angle that facilitates matching with the Poincaré section.



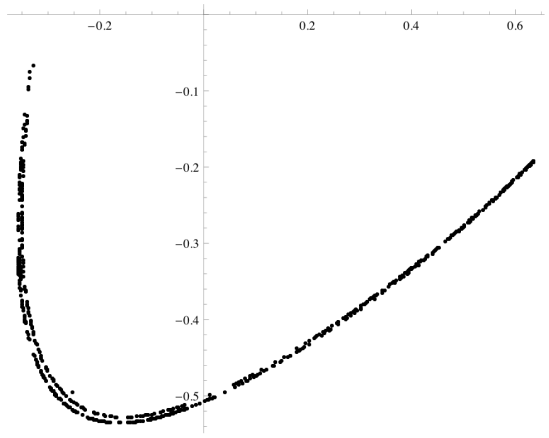
(a) Sprott Eq. 9 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa + v^2 - x$, where $A = 2.017$ with a Poincaré plane.



(b) Poincaré section for Sprott Eq. 9 attractor map.



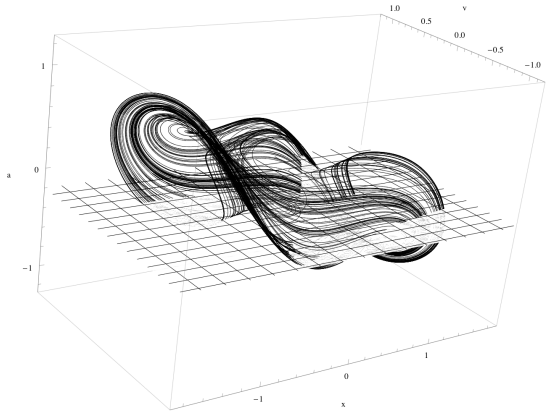
(c) Sprott Eq. 15 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -Aa - v + x^2 - B$, where $A = 0.5$ and $B = 0.25$ with a Poincaré plane.



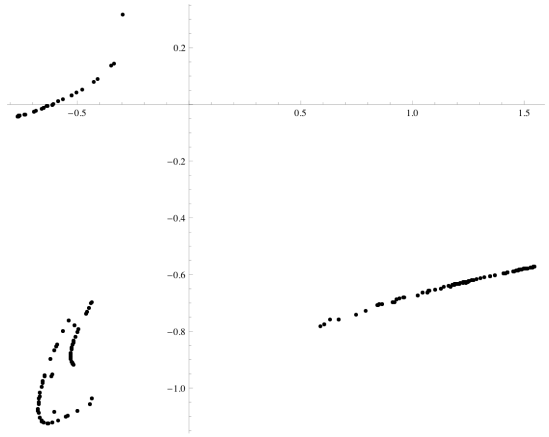
(d) Poincaré section for Sprott Eq. 15 attractor map.

Figure 1.6.6: Poincaré planes and Poincaré sections with the Sprott Eq. 9 system and the Sprott Eq. 15 system.

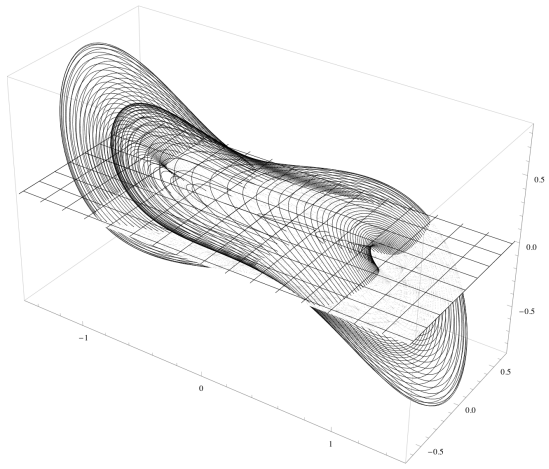
These are the Poincaré sections of the remaining two maps.



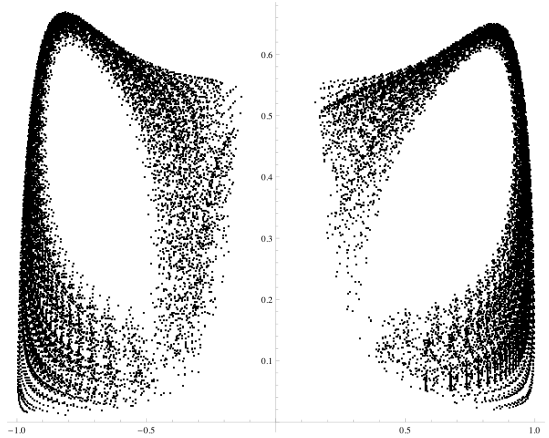
(a) Sprott Eq. 21 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -a^3 - vx^2 - Ax$, where $A = 0.25$ with a Poincaré plane.



(b) Poincaré section for Sprott Eq. 21 attractor map..



(c) Sprott Eq. 23 attractor map $\dot{x} = v,\ \dot{v} = a,\ \dot{a} = -va^2 + A(1 - x^2)x$, where $A = 0.01$ with a Poincaré plane.



(d) Poincaré section for Sprott Eq. 23 attractor map, here we have displayed 50 superimposed Poincaré sections, by taking 50 different trajectories and finding the section for each one.

Figure 1.6.7: Poincaré planes and Poincaré sections with the Sprott Eq. 21 system and the Sprott Eq. 23 system.

1.7 Graph Theory and Complex Networks

Graph theory was born in 1735 when the Swiss mathematician Leonhard Euler solved the famous Königsberg bridge problem. The problem was to answer whether one could walk around the whole bridge system by going along each bridge only once. Euler proved conclusively that such a walk was not possible by modeling the complicated bridge structure at Königsberg by assigning dots for the parts of land and segments to the different bridges connecting the land [1].

Modeling structures from real life using dots or **vertices** and segments or **edges** connecting them has become a well-developed mathematical field, and has provided many useful tools for understanding the world. Parallel with the mathematical development of the concept of a graph, many models of systems with large number of vertices or nodes, were called **networks**. Examples of networks are the internet, the Facebook network, any social network, or business network, or any kind of biological network, including neural networks in the brain, cardiovascular system, delivery routes, etc.

In this work, we will use the terms graph and network interchangeably. Most of what follows in this section are standard definitions from graph theory that can be found in most introductory texts on the subject (see [1, 4]).

While graphs have been studied extensively in mathematics, much of the study has centered on simple graphs with finitely many (a few, in fact) vertices and edges, which has enabled many visualization methods. In networks, the number of vertices and edges is too large, and many statistical techniques have been employed. The research has expanded the analysis from local description of simple graphs and graph clusters, to a more global approach to understanding complex networks.

Formally, a **graph** G is an ordered pair $G = (V_G, E_G)$, where $V = \{v_1, v_2, \ldots, v_n\}$ for some positive integer n is the **vertex set** of G, and $E = \{e_1, e_2, \ldots, e_m\}$ for some

nonnegative integer m is the **edge set** of G, with each e_i of the form (v_j, v_k) for some vertices in V_G. Edges in E_G are identified by their endpoints, or by the pair of vertices that each edge connects.

Edges may also be assigned a direction or orientation, making G a **directed graph**. In this case, E_G is a set of ordered pairs.

If an edge joins the same vertices, it forms a **loop**. Also, two vertices may be joined by multiple edges. A graph that does not have loops or multiple edges is called **simple**.

A trivial graph with no vertices and/or no edges is called **empty**. Another special case is formed by a simple graph with each vertex being adjacent to every other vertex. This graph is also known as **complete**. The complete graph with n vertices is denoted by K_n.

The **complement of a graph** G, denoted by G^C, is obtained from G by removing all its edges and joining exactly those vertices that were not adjacent in G. The union of a graph G and its complement G^C (where the union is given by taking unions of edges and vertices, respectively) yields a complete graph.

The **neighbor set (or neighborhood) of a vertex** is formed by vertices that are adjacent to that vertex.

In an undirected graph, each vertex has a **degree** given by the number of edges incident on it. In a directed graph, the degree may be defined as a difference between the number of edges ending at the vertex and the number of edges originating at the vertex. One can define a probability function on a graph, the **degree distribution function**, which takes on vertices and assigns to them the probability that their degree is a given number. This function can also be used as an estimate of the fraction of vertices in the graph having a given degree.

Graphs can be represented by using matrices and vertex degrees. The **adjacency matrix** A associated with a graph $G = (V_G, E_G)$ with n vertices is an $n \times n$ matrix whose elements a_{ij} give the number of edges between vertices v_i and v_j (for undirected graphs).

A is a symmetric matrix if the corresponding graph is undirected, not symmetric if it is directed. Note that diagonal elements a_{ii} give (twice) the number of loops at the vertex v_i.

We will pay particular attention to distance and paths in graphs. The **shortest path** between two vertices is given by the smallest number of edges connecting them in an unweighted undirected graph. This is also called the **geodesic distance** between the two vertices. For a **weighted network**, edges are assigned a given value, or **weight**, and the shortest distance is the path of minimum weight. If no path connecting two vertices exists, then their distance is infinite.

Given a graph, the average path length between any two vertices is called **the characteristic path length**, given by the mean of the geodesic length over all possible pairs of vertices.

$$CL = \frac{1}{n(n-1)} \sum_{i \neq j} d_{ij},$$

where d_{ij} is the geodesic distance between v_i and v_j in a connected component of the graph.

If we build a matrix D indicating all the geodesic distances d_{ij}, this matrix is called the **graph distance matrix**. The length of the maximum geodesic in a given graph (the maximum value d_{ij}) is called the **graph diameter**, and the length of the minimum geodesic is called the **graph radius**.

Most recent developments in graph theory have presented a few local properties describing the microstructure of a graph ([`?diester`]). A clustering coefficient in a graph measures the extent to which vertices tend to cluster together. In most real-world networks, and in particular social networks, vertices seem to create tightly knit groups with a relatively high density of connections. Two versions of this measure exist: the **global clustering coefficient** or just **the clustering coefficient** and the **local clustering**

coefficient. The global version was designed to give an overall indication of the clustering in the graph, whereas the local gives an indication of the way single vertices are placed in the graph.

The global clustering coefficient is based on triangles (or triplets) of vertices. A **triangle of vertices** consists of three vertices that are connected by either two (open triangle) or three (closed triangle) edges. A **triad** consists of three closed triangles, one centered on each of the nodes. The global clustering coefficient is the number of closed triads over the total number of triangles (both open and closed).

The local clustering coefficient of a vertex is computed as the proportion of connections among its neighbors which are actually realized compared with the number of all possible connections. The local clustering coefficient of a vertex quantifies how close neighbors are to being a clique (a complete graph). Duncan J. Watts and Steven Strogatz introduced the measure in 1998 ([13]) to determine whether a graph is a small-world network. A graph is considered **small-world** if its average clustering coefficient is significantly higher than a random graph constructed on the same vertex set, and if the graph has approximately the same mean-shortest path length as its corresponding random graph.

The clustering coefficient for the whole graph is computed as the average of the local clustering coefficients of all the vertices.

1.8 Visibility and Horizontal Visibility Algorithms

There are many complex systems in nature and in technology such as neural networks, coupled biological and chemical systems, social interacting species, and the internet. The first approach to developing global properties of such systems is to model them as graphs whose vertices represent elements of the system, and whose edges represent specific interactions between these elements. This representation of a system aims to capture structures that exhibit time-dependent or other types of dynamics. Even though edges and vertices

are unable to adequately capture the characteristics of the system completely, there are many tools such as weights, vertex coloring, and clustering, that allow for an effective study of the system. Apart from statistical and geometric studies of such real-world systems, the most recent developments have incorporated analytical and topological methods as well (see [5, 6, 8]).

One of the main tools we will employ in this study is the **visibility graph** ([6]), which inherits some fundamental properties of the time series that it comes from.

The **visibility algorithm** ([6]) maps a time series into a graph, and we can use techniques from graph theory to study its properties. Here is a brief outline of the algorithm.

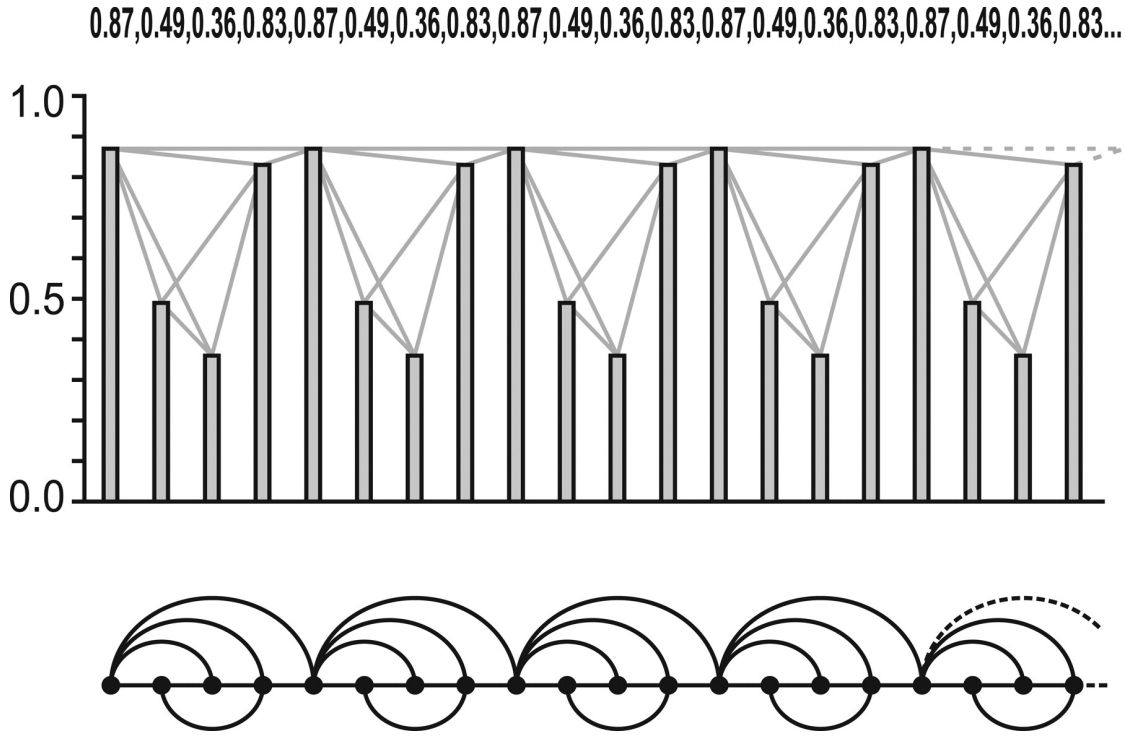


Figure 1.8.1: Time series (periodic) of 20 data points, depicted as bars and the associated visibility graph derived from the visibility algorithm. In the graph, every vertex corresponds, in the same order, to a data from the time series, and edges are defined by the visibility criterion (reprinted from [6]).

Figure1.8.1 shows a plot of the first 20 values of a (periodic) time series (represented with vertical bars), and we link every bar with all the ones that can be seen from the top of this bar, obtaining the associated visibility graph, which is shown in the lower part of the figure.

In the visibility graph, vertices correspond to data points, and two vertices are connected if **visibility** exists between the corresponding data points, that is, the line of visibility between the corresponding bars does not intersect any intermediate bar. So we have the **visibility criterion** ([6]): two data points x_i and x_k will have visibility, and consequently will become two connected vertices of the associated visibility graph, if any other data point x_j between them (that is, in the time series, $\{\ldots,\ x_i, \ldots,\ x_j, \ldots,\ x_k, \ldots\}$ or $i < j < k$) fulfills:

$$x_j < x_i + (x_k - x_i)\frac{j-i}{k-i}.$$

In order to understand this criterion, we consider the three possible cases:

- $x_i > x_j$ and $x_j < x_k$, in which case x_k is represented by a bar taller than x_j and is visible from both x_i, which is also taller than x_j, and x_j,

- $x_i > x_j$ and $x_j > x_k$, in which case the relative slopes of the segments $x_i x_j$ and $x_i x_k$ and $x_j x_k$ come into play,

- $x_i < x_j$ and $x_j < x_k$, which is not as obvious as it seems, so the relative slopes of the segments $x_i x_j$ and $x_i x_k$ and $x_j x_k$ come into play again.

These are easily understood as the only scenarios in which visibility is possible. In the remaining fourth case, $x_i < x_j, x_k$, yet $x_j > x_k$ (since we have two inequalities with a total of four options), x_j would be represented by a bar taller than the bar representing x_i or x_k, thereby blocking visibility between them. So in that case, we run into trouble.

Therefore, the criterion contains the first three cases, and eliminates the fourth. Note that

$$x_j < x_i + (x_k - x_i)\frac{j-i}{k-i}.$$

is equivalent to

$$x_j - x_i < (x_k - x_i)\frac{j-i}{k-i},$$

which can also be rewritten as

$$\frac{x_j - x_i}{j-i} < \frac{x_k - x_i}{k-i}.$$

Essentially, this means that the slope of the line between the points x_j and x_i is smaller than the slope of the line between the points x_i and x_k, ensuring that x_k is visible from x_i so x_j does not "block" the visibility of x_k from x_i. In other words, the line segment $x_i x_k$ is steeper than the line segment $x_i x_j$, thus the latter remains "below" the former, except for the common endpoint x_i.

Remark 1.8.1. It will be very important for us to observe the fact that the visibility criterion inequality

$$x_j < x_i + (x_k - x_i)\frac{j-i}{k-i}$$

holds true if we exchange x_i and x_k, that is,

$$x_j < x_k + (x_i - x_k)\frac{j-k}{i-k}$$

which means that if x_i i visible from x_j, then x_j is visible from x_i (see Lemma 1.8.2 below). So visibility is a **symmetric relation**. It is easy to see that it is **not reflexive** and **not necessarily transitive** (for the latter, imagine three tall bars with many very short bars between them and the middle bar taller than all of them).

Lemma 1.8.2. *Given three data points* $\{\ldots,\ x_i, \ldots,\ x_j, \ldots,\ x_k, \ldots\}$ *(with* $i < j < k$*) the visibility criterion*

$$x_j < x_i + (x_k - x_i)\frac{j-i}{k-i}$$

is symmetric with respect to i and k, that is, the above inequality is equivalent to

$$x_j < x_k + (x_i - x_k)\frac{j-k}{i-k}.$$

Proof. Given that $x_j < x_i + (x_k - x_i)\frac{j-i}{k-i}$, we have the following equivalent expressions

$$x_j < x_i + x_k\left(\frac{j-i}{k-i}\right) - x_i\left(\frac{j-i}{k-i}\right)$$

$$x_j < x_i + x_k\left(\frac{j-i}{k-i}\right) + (x_k - x_k) - x_i\left(\frac{j-i}{k-i}\right)$$

$$x_j < x_k + x_k\left(\frac{j-i}{k-i} - 1\right) + x_i\left(1 - \frac{j-i}{k-i}\right)$$

$$x_j < x_k + x_k\left(\frac{j\cancel{-i} - k\cancel{+i}}{k-i}\right) + x_i\left(\frac{k\cancel{-i} - j\cancel{+i}}{k-i}\right)$$

$$x_j < x_k + x_k\left(\frac{j-k}{k-i}\right) + x_i\left(\frac{k-j}{k-i}\right)$$

$$x_j < x_k - x_k\left(\frac{j-k}{i-k}\right) - x_i\left(\frac{k-j}{i-k}\right)$$

$$x_j < x_k - x_k\left(\frac{j-k}{i-k}\right) + x_i\left(\frac{j-k}{i-k}\right)$$

$$x_j < x_k + (x_i - x_k)\left(\frac{j-k}{i-k}\right).$$

□

It has been shown ([6]) that time series structure is inherited in the associated graph, which suggests that the visibility graph may capture the dynamical fingerprints of the process that generated the series. We are particularly interested in exploring this question in detail here.

There is a refinement of the above technique, called the **horizontal visibility algorithm** which generates a **horizontal visibility graph** associated to a time series or data

series ([5]). The horizontal visibility algorithm assigns each point x_i of a given time series to a vertex in the horizontal visibility graph according the following rule.

Two vertices v_i and v_j in the graph (corresponding to points x_i and x_j in the time series) are connected if one can draw a horizontal line in the time series joining x_i and x_j that does not intersect any intermediate data height (see Figure 1.8.2). So v_i and v_j are two connected vertices if the following geometrical criterion is fulfilled within the time series:

$$x_i, x_k > x_j \text{ for all } j \text{ such that } i < j < k$$

This algorithm is a simplification of the visibility algorithm, and the horizontal visibility graph is always a subgraph of the corresponding visibility graph.

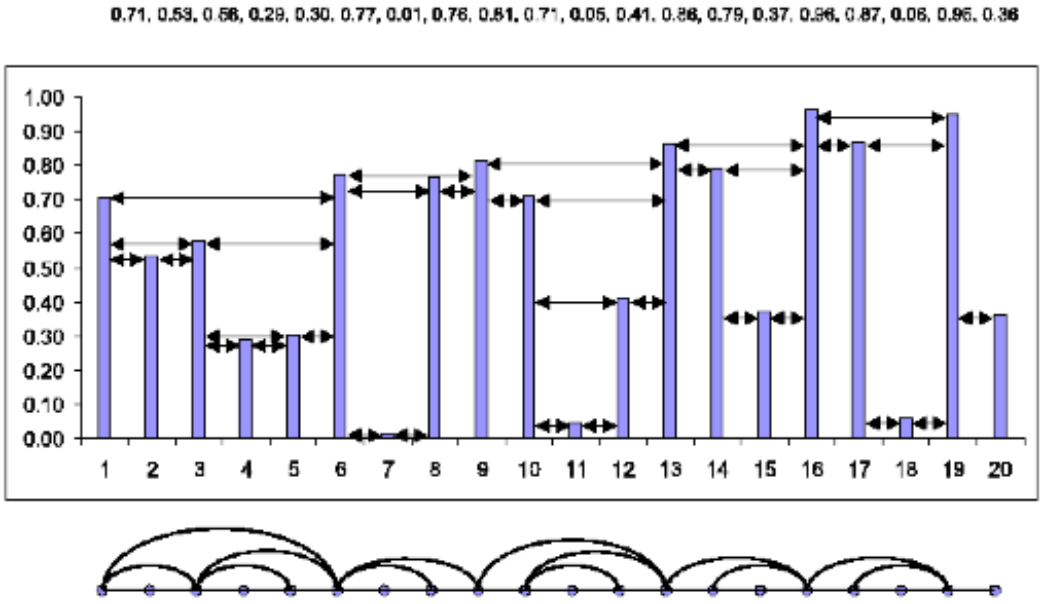


Figure 1.8.2: Illustrative example of the horizontal visibility algorithm. In the upper part we plot a time series and in the bottom part we represent the graph generated through the horizontal visibility algorithm. Each point in the series corresponds to a vertex in the graph, such that two vertices are connected if their corresponding data heights are larger than all the data heights between them (reprinted from [5]).

There are a few obvious properties of the horizontal visibility graph that the horizontal visibility algorithm readily implies. The horizontal visibility graph associated to a time series is always connected since each point "sees" at least its nearest neighbors on the left or on the right. The graph is also undirected by virtue of the way the algorithm is contructed. We will be interested in modifying or extending this in our work. The graph is also invariant under affine transformations of the series data: the visibility criterion is invariant under rescaling of both horizontal and vertical axes, and under horizontal and vertical translations (see [5]).

We will be interested in a few other characteristics of horizontal visibility graphs. One of them is their reversibility, that is, note that some information regarding the time series is inevitably lost in the process of the application of the horizontal visibility algorithm from the fact that the graph structure is completely determined in the (binary) adjacency matrix, while time series carry much more information. For instance, two distinct time series series can easily produce the same visibility graph.

One way to address this problem is to generalize the visibility method by using weighted graphs (where the adjacency matrix is not binary and the weights determine the height difference of the associated data).

We will explore this further and study the efficiency of using weighted graphs in the various visibility techniques.

Another issue we would like to address is the undirected graphs that the algorithm generates. In this work, we show a few methods that could help one also extract a directed graph.

Note that the geometric criterion defined for the horizontal visibility algorithm is more restrictive than its analogue for the general case. That is to say, the vertices within the horizontal visibility graph will be less "visible" than their counterparts within the visibility

graph. While this fact does not have an impact on the qualitative features of the graphs, quantitatively speaking, horizontal visibility graphs will have typically "less statistics".

We will be interested in the following properties of the visibility graph.

The first one is the **degree distribution of the visibility graph** ([7,8]), which is one of the most important properties of complex networks. It has been shown that the degree distribution for stochastic and chaotic time series is exponential. In order to compute the degree distribution $P(k)$ of the associated visibility graph to a given time series, we need to calculate the probability that an arbitrary point x_i has visibility of exactly k other data points. The general expression for $P(k)$ has been proved to be

$$P(k) = \frac{1}{3}\left(\frac{2}{3}\right)^{k-2}.$$

An interesting aspect worth exploring is the relation between data height and the vertex degree, that is, to study whether a functional relation between the value of a point x_i and the degree of its associated vertex v_i exists.

Another interesting relationship directly related to the horizontal visibility graphs of chaotic time series is the connection between the degree distribution $P(k)$ and the **Poincaré recurrence time (PRT)**, which measures the time interval between two consecutive visits of a trajectory to a finite size region of phase space. It has been shown that Poincré recurrence times are exponentially distributed in some known chaotic systems, such as the logistic equation and the Hénon system. Several conjectures have been made about the relationship between $P(k)$ and PRT ([7,8]).

1.9 Feigenbaum Graphs

The recently formulated theory of horizontal visibility graphs transforms time series into graphs and allows the possibility of studying dynamical systems through the characterization of their associated networks. Feigenbaum graphs come in this subject of study study

in the period-doubling and band-splitting attractor cascades that characterize certain systems. This classic setup is treated in terms of the horizontal visibility graphs associated with the dynamics within the attractors, that are called **Feigenbaum graphs**, independent of map nonlinearity or other particulars. Exact results have been derived for their degree distribution and related quantities ([7,8]).

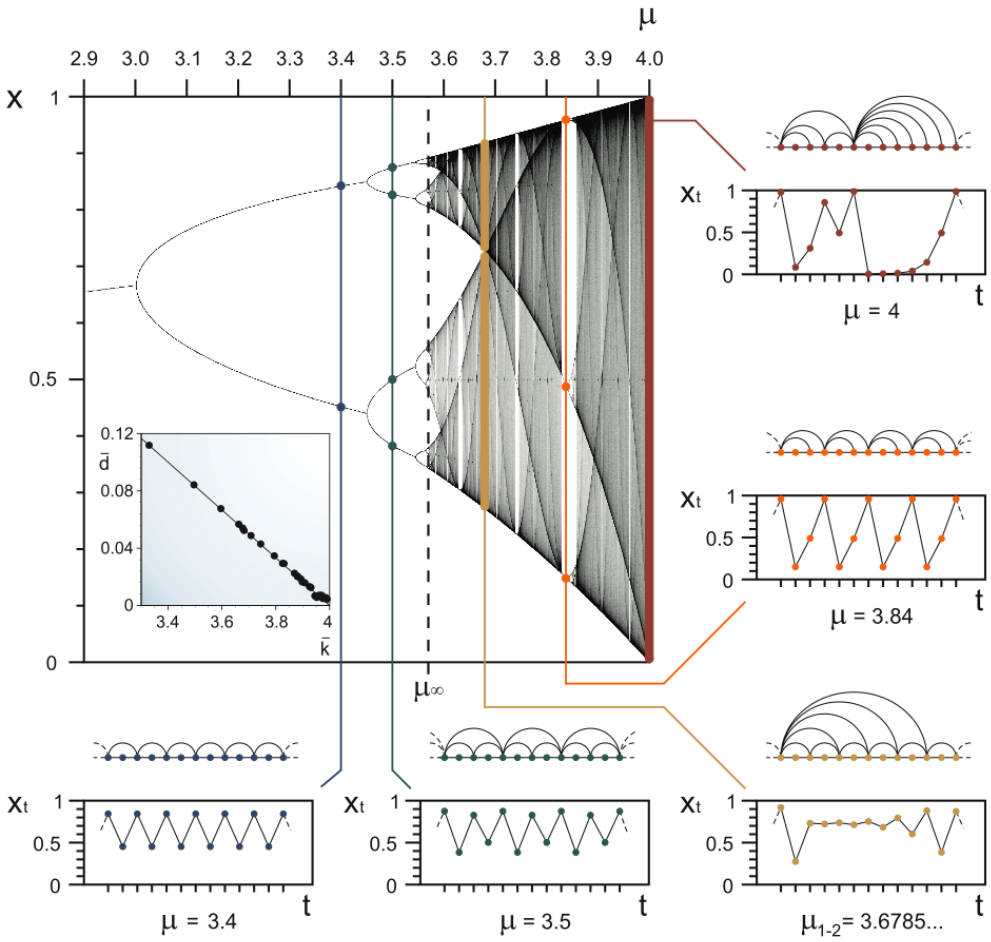


Figure 1.9.1: Feigenbaum graphs from the logistic map. The main figure indicates a transition from periodic to chaotic behavior at $a_\infty = 3.569946\ldots$ through period-doubling bifurcations. For $a > a_\infty$, attractors merge where aperiodic behavior appears interrupted by windows that, when entered from their left-hand side, display periodic motion that subsequently develops into period-doubling cascades with new accumulation points. Adjoining the main figure, we show time series and their associated Feigenbaum graphs according to the horizontal visibility algorithm for several values of a where the map evidences both regular and chaotic behavior. (reprinted from [8]))

Horizontal visibility graphs represent the time evolution of all trajectories that take place within the attractors of the given system.

The authors ([7]) use the logistic map as an example of illustrating the construction of the Feigenbaum graphs associated with the time series and the horizontal visibility graph of the logistic map. A time series of the logistic map for a particular value of its parameter a is converted into a Feignebaum graph as Figure 1.9.2 shows.

The construction of Feigenbaum graphs follows the standard horizontal visibility graph construction. Given a time series $\{x_i\}_{i=1,\ldots,\ N}$ of N real data points, a vertex v_i is assigned to each element x_i of the time series and then two vertices v_i and v_j are connected if the corresponding data elements fulfill the horizontal visibility criterion $x_i, x_j > x_n$ for all n such that $i < n < j$.

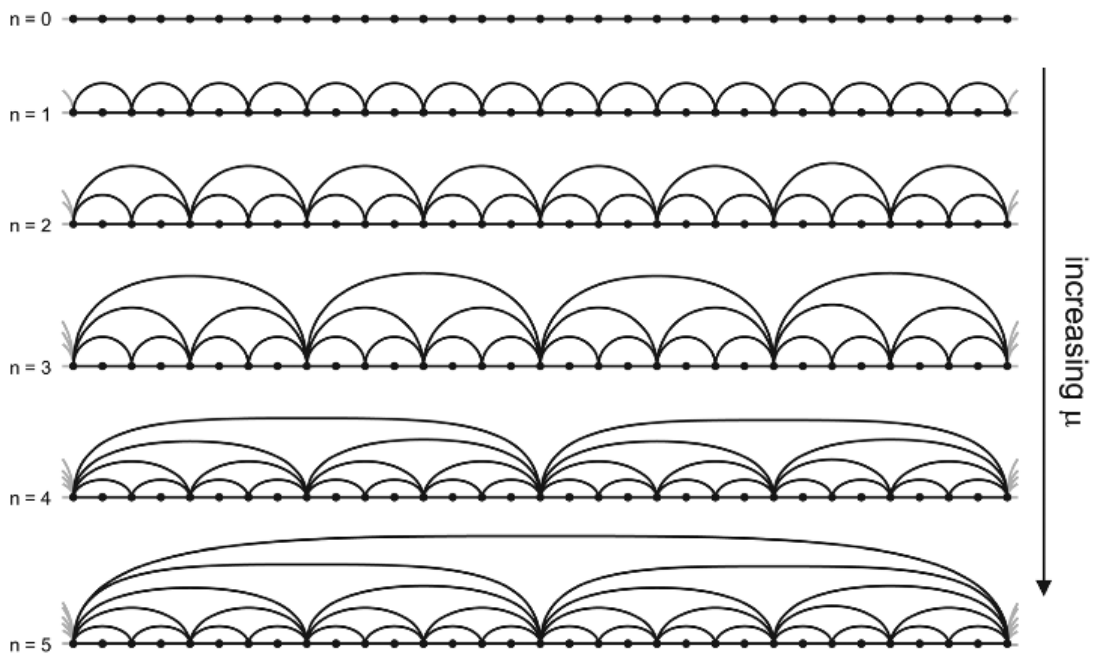


Figure 1.9.2: Feigenbaum graphs from the logistic map for $\mu < 3.569946$. The pattern that occurs for increasing values of the period $T = 2^n$ due to period-doubling is a consequence of the universal ordering of orbit visits by the system. The self-similarity of these graphs requires that the graph for n is a subgraph of the one for $n + 1$. (reprinted from [7])

In Figure 1.9.2, for each value of the parameter a (in the notation of [7], μ), a time series is generated, that is, a particular trajectory for the (logistic) system corresponding to the particular variant of the logistic system specified by the value of the parameter. In that sense, understanding the horizontal visibility algorithm for periodic time series is essential, since the logistic system exhibits constant or periodic solutions for values of a between 2.9 and about 3.57.

It has been shown [5, 8] that the horizontal visibility graph is a **planar graph**, that is, there are no edge intersections except at vertices. Moreover, a horizontal visibility graph is also **outerplanar**, that is, each vertex is on the boundary of the complement of the planar region enclosed by the graph (so no vertex is in the interior of the region of the plane bounded by the graph.

A fundamental characteristic of the period-doubling cascade is that the order in which the values of an oscillation are ordered is very specific. This order is essential for the construction of the corresponding Feigenbaum graph. Figure 1.9.3 below illustrates the process.

After the first period-doubling, we have two values x_1 and x_2, and the system oscillates between them. To generate the Feigenbaum graph, label the larger value x_1 and the smaller x_2, without loss of generality. The corresponding Feigenbaum graph is periodic and is constructed form the repeated configuration of three vertices (see Figure 1.9.3). Upon the next period-doubling occurrence, each point splits up into two points. The labeling here depends on our initial choice of labeling as follows: each point x_i splits into two points labeled as x_i and x_{i+T}, where T is the period of the oscillation; after the first period-doubling, $T = 2^1$, after the second one, it is 2^2, and so on. This means that after the first doubling with $T = 2$, x_1 splits into x_1 and x_3, and x_2 splits into x_2 and x_4 (Figure 1.9.3). So the period-doubling bifurcation of an orbit of period $T = 2^k$ generates two identical copies of the $T = 2^k$ root motif of the graph, which are now concatenated by the vertex

v_{1+2^k} and linked by bounding vertices x_1 and x_1 (on the other end), which becomes the root motif for the $T = 2^{k+1}$ Feigenbaum graph.

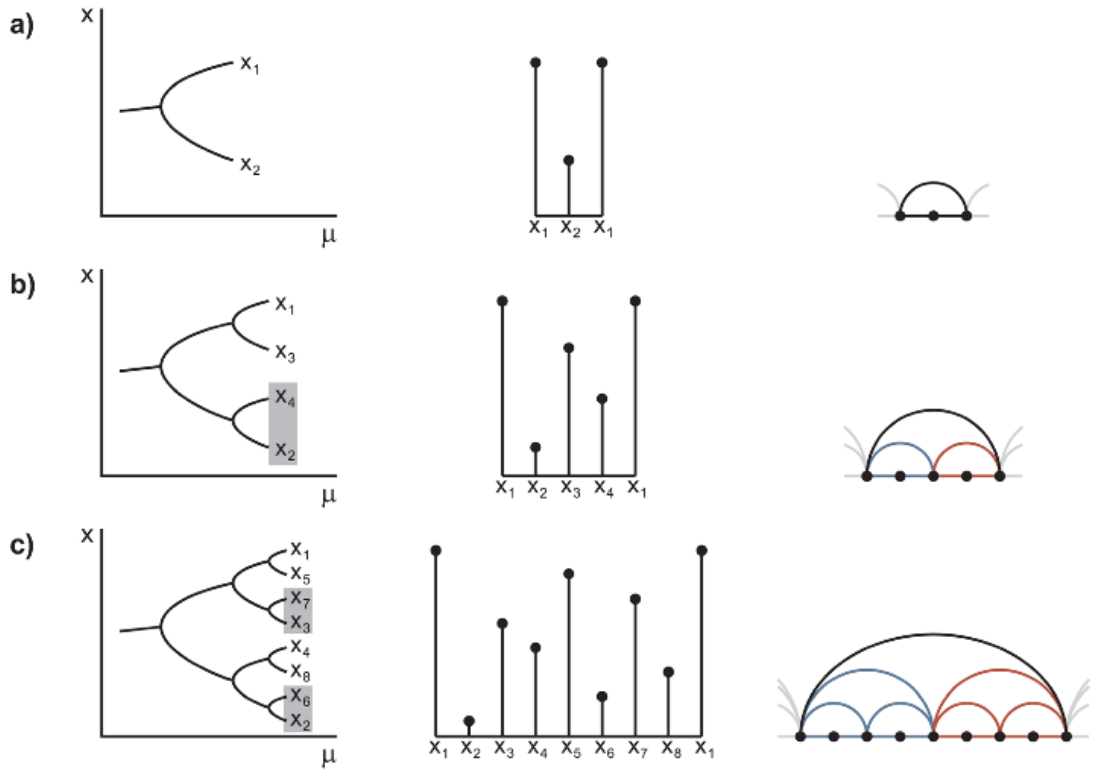


Figure 1.9.3: Graphical illustration of the order in which stable values are visited, and the induced Feigenbaum structure. (reprinted from [7]).

We will be particularly interested in the **degree distribution** $P(n, k)$ for a Feigenbaum graph ([7]). Recall that the degree distribution of a graph is defined as a discrete probability distribution that expresses the probability of finding a node with degree k. By construction, the degree distribution of a Feigenbaum graph for a series of period $T = 2^n$ where $n = 0,\ 1,\ 2, \ldots,$ is given by

$$P(n,k) = \left(\frac{1}{2}\right)^{k/2},\ k = 2,\ 4,\ 6, \ldots,\ 2n$$

$$P(n,k) = \left(\frac{1}{2}\right)^{n},\ k = 2(n+1)$$

$$P(n,k) = 0,\ k = 3,\ 5,\ 7, \ldots,\ \text{or } k > 2(n+1).$$

At the accumulation point $\mu_\infty = 3.5699456\ldots$, $n \to \infty$, and the degree distribution of the Feigenbaum graph at the onset of chaos becomes an exponential for even values of the degree.

$$P(\infty,k) = \left(\frac{1}{2}\right)^{k/2},\ k = 2,\ 4,\ 6, \ldots,\ 2n$$

$$P(\infty,k) = 0,\ k = 3,\ 5,\ 7, \ldots,\ \text{or } k > 2(n+1).$$

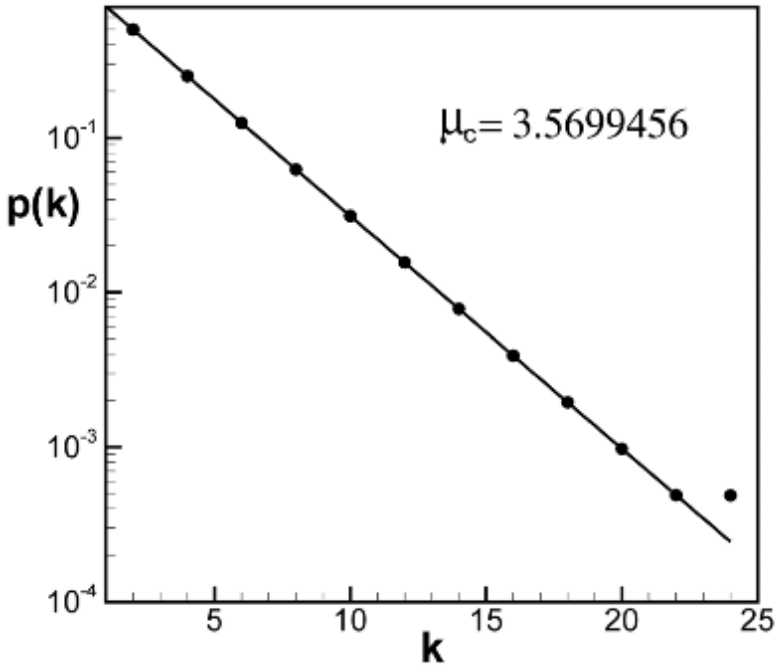


Figure 1.9.4: Logarithmic plot of the degree distribution of a Feigenbaum graph associated with a time series of 1,000,000 data points extracted from a Logistic map at the onset of chaos, where r, or μ_c, as the authors have labeled it, is equal to 3.5699456... The straight line corresponds to the power law above in agreement with the numerical calculation (the deviation for large values of the degree are due to finite size effects).(reprinted from [8]).

Another property of Feigenbaum graphs we will be interested in is their **mean degree** ([7,8]). Consider a periodic orbit of a time series of period T, where the orbit is represented by the recursively defined sequence $\{x_0,\ x_1, \ldots,\ x_T\}$ with $x_0 = x_T$. Then the associated

Feigenbaum graph consists of a concatenation of identical motifs of $T+1$ vertices. Figure 1.9.5 shows an inductive argument for calculating the mean degree $\bar{k}$ by removing data points in increasing order of height. For a motif of $T+1$ vertices, this results in $2(T-1)$ deleted edges.

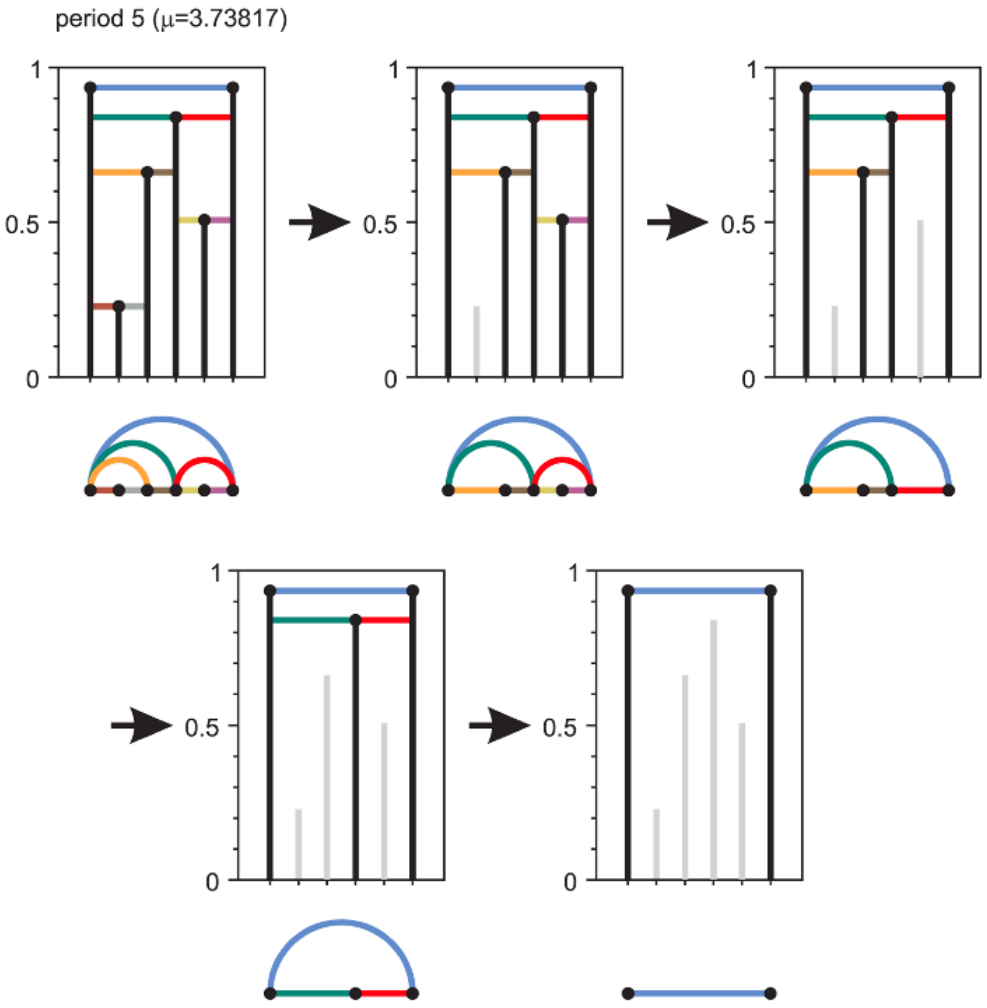


Figure 1.9.5: Feigenbaum graphs from the logistic map. The main figure indicates a transition from periodic to chaotic behavior at $a_\infty = 3.569946\ldots$ through period-doubling bifurcations. For $a > a_\infty$, attractors merge where aperiodic behavior appears interrupted by windows that, when entered from their left-hand side, display periodic motion that subsequently develops into period-doubling cascades with new accumulation points. Adjoining the main figure, we show time series and their associated Feigenbaum graphs according to the horizontal visibility algorithm for several values of a where the map evidences both regular and chaotic behavior (reprinted from [8]).

The mean degree $\bar{k}$ of the graph is given by the mean degree o the motif made of T vertices (excluding $x_0 = x_T$, which contribute a value of 1 to the degree), therefore $\bar{k} = 2\frac{\#\ edges}{\#\ vetices} = \frac{2(2(T-1)+1)}{T} = 4\Big(1 - \frac{1}{2T}\Big)$, which is true even in the case when $T \to \infty$. The maximum value for the mean degree is $\bar{k} = 4$ in the case when the time series is aperiodic ($T \to \infty$).

Another property of Feigenbaum graphs we will be looking at is the **mean distance** $\bar{D}$ ([8]) which is given by the average of the shortest path between a pair of connected vertices, averaged over all pairs of connected vertices). The **normalized mean distance** $\bar{d}$ is given by $\bar{d} = \bar{D}/N$, where N is the number of data points, hence the number of vertices. For a horizontal visibility graph associated with an infinite time series ($N = \infty$), the mean degree and the normalized mean distance are related by

$$\bar{d}(\bar{k}) = \frac{1}{6}(4 - \bar{k}).$$

The local clustering coefficient is another quantity we will be looking at, and for a Feigenbaum graph corresponding to an attractor of period $T = 2^n$, we have from [8] that

$$c(n,k) = \frac{2}{k},\ k = 2,\ 4,\ 6, \ldots,\ 2n.$$

$$c(n,k) = \frac{2(k-2)}{k(k-1)}\ k = 2(n+1).$$

With the approximation

$$c(n, 2(n+1)) \approx 1/(n+1)\ P(n,c) = \frac{1}{2}^{1/c}.$$

It can be shown that the mean clustering coefficient $\bar{c}(n)$ converges to $\ln 2$ as $n \to \infty$ very rapidly.

2
New Ideas and Development

2.1 Extensions of the Visibility Algorithm

In this section, we define and explore a few generalizations of the visibility algorithm.

2.1.1 Weighted and Directed Visibility Graphs

The first question we would like to address is the loss of information during the visibility process. Namely, one can easily generate different time series that give rise to the same identical visibility graph.

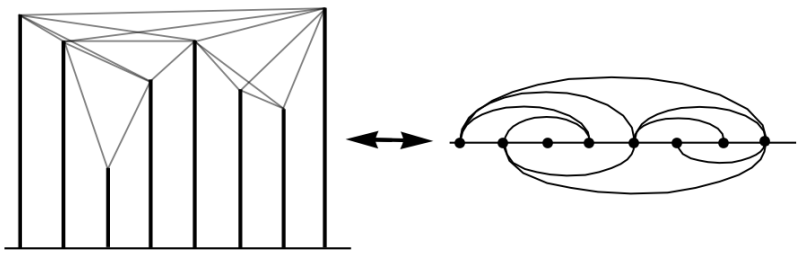


Figure 2.1.1: Bijective correspondence between a time series and its visibility graph.

In order to easily create a **bijective correspondence** between time series (in this section, we will be dealing with 1-dimensional time series) and the corresponding visibility

graph (so that the time series can be recovered from its visibility graph), we claim that it is sufficient to add weights to the vertices of the visibility graph, where each weight is given by the value of the corresponding point in the time series. Thus, x_i is mapped to the vertex v_i by the visibility algorithm, where v_i is assigned a weight given by the value of x_i.

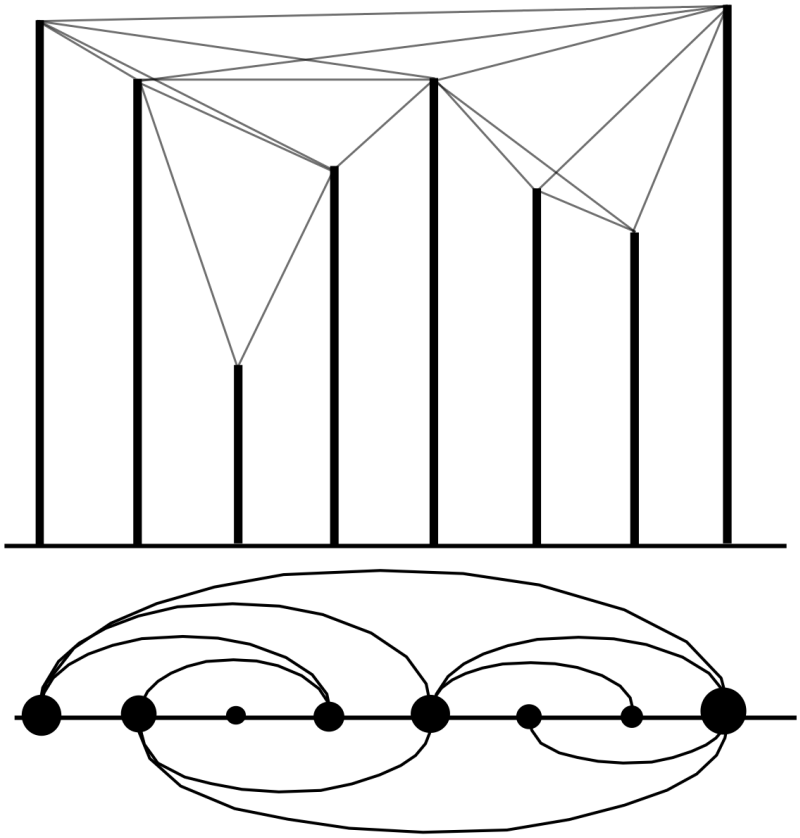


Figure 2.1.2: An example of a weighted visibility graph.

We would like to note that this is a rather trivial modification of the visibility algorithm since the vertices of the visibility graph are already arranged on a line, so the order (or "time line") of the time series is already there, so the weights of the vertices just give us the numerical values of x_i. This idea will be particularly relevant to us when we begin considering higher-dimensional time series.

Another addition to the visibility algorithm we recall is the ability to construct a **directed visibility graph** associated with a time series. One can distinguish two different values associated with a vertex v_i in the visibility graph: an ingoing degree k_{in}, related to how many vertices see the vertex v_i, and an outgoing degree k_{out}, that is the number vertices that the vertex v_i sees.

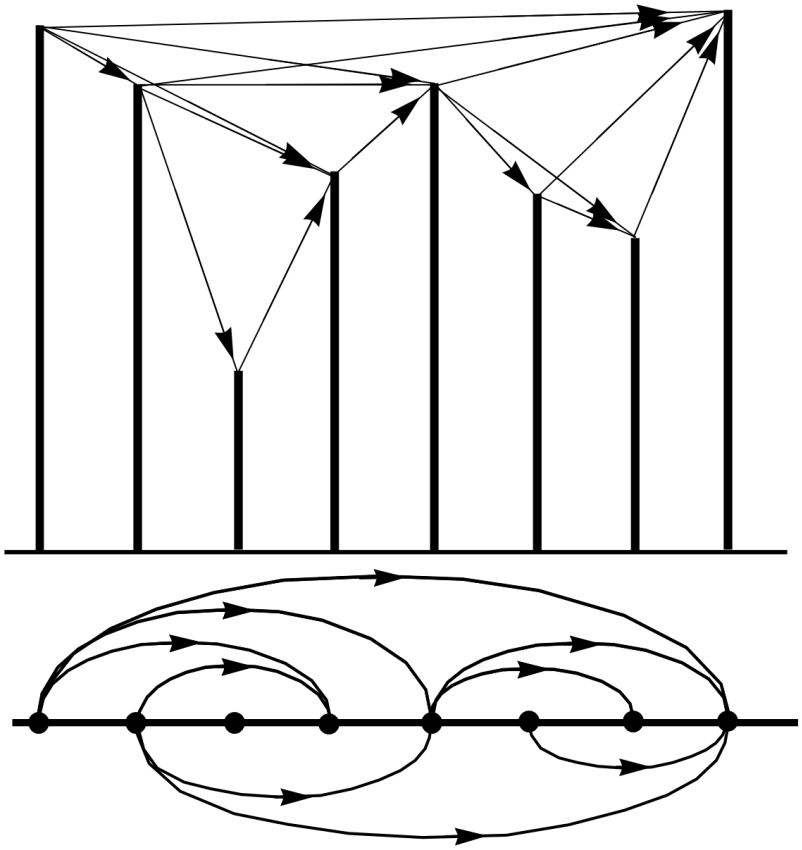


Figure 2.1.3: An example of what would be a standard way of constructing a directed visibility graph. Notice, however, that the visibility graph's edges are all oriented in the same direction.

In this work, we propose a different method of deriving a directed visibility graph as follows. We can assign a direction to each edge by considering the value of the slope of the segment connecting the two corresponding values x_i and x_j. If the slope is negative, we

assign a direction that points from v_i to v_j, and if the slope is positive, the direction of the edge is reversed: from v_j to v_i. One can certainly reverse this convention. In the way we have defined it, however, edges always point "downward", that is, always run from vertices with higher weight to vertices with lower weight. In that sense, vertices with heavier weight are "sources" for all the visible vertices of lower weight. This will be particularly significant for us when we explore a variant of the visibility algorithm below.

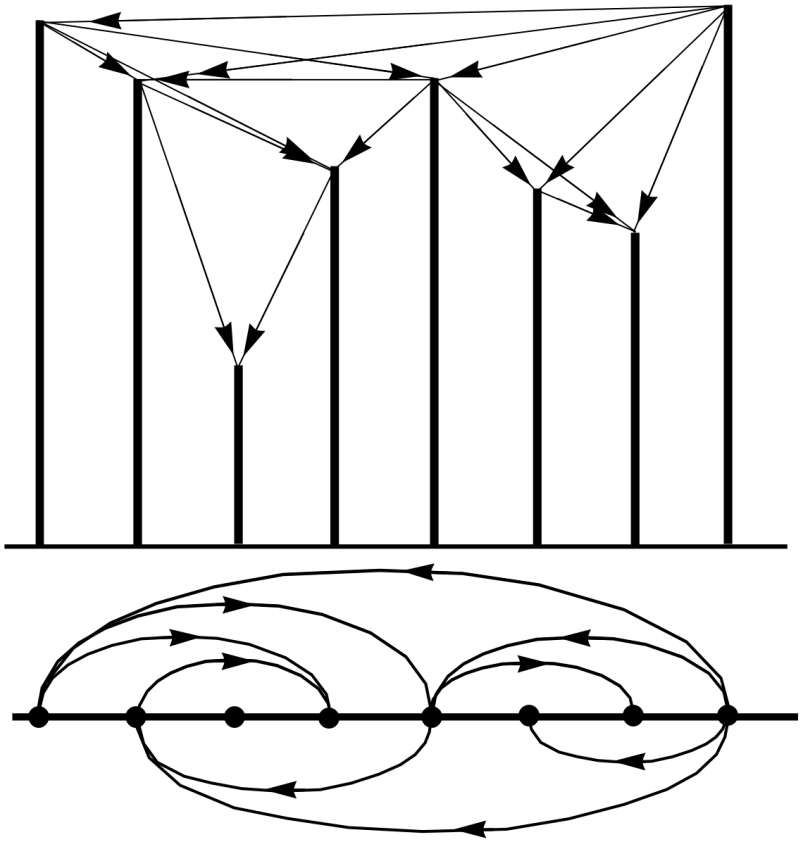


Figure 2.1.4: An example of our definition of a visibility graph.

Note that both the idea of weights, as well as the idea of assigning direction to the edges, extend to horizontal visibility graphs.

A brief modification we will use for the representation of the Feigenbaum graph of periodic time series of high enough period is the representation of the Feigenbaum graph on a

circle as opposed to a periodic line (see Figure 2.1.5). Note that in the usual representation of a periodic time series, the first point of the time series, call it x_1, occurs multiple times in the Feigenbaum graph representation on the line.

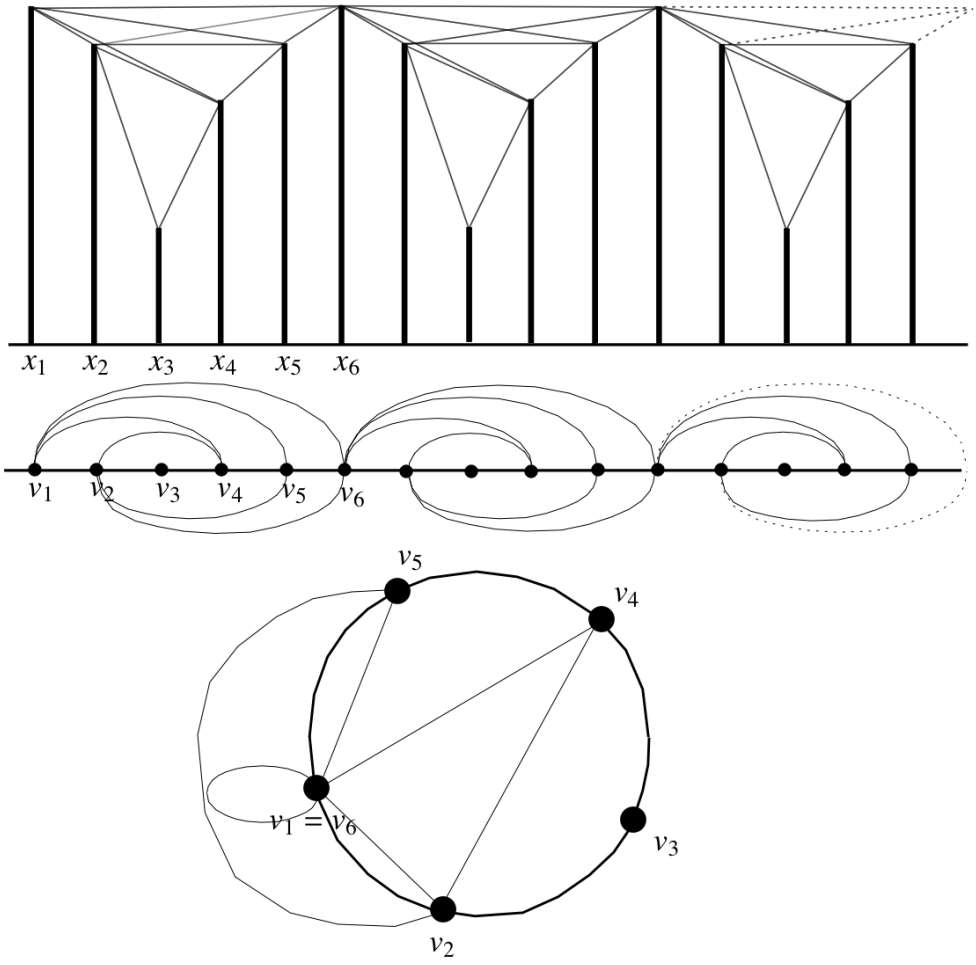


Figure 2.1.5: A circle representation of the visibility graph associated with a periodic time series. The repeated pattern is of the first 5 data points, so the visibility graph repeats a pattern after the first 5 vertices, so the circle shows 5 points with the sixth point identified with the first one, running in a counterclockwise way.

This representation of the Feigenbaum graphs, invites some tools from geometry, combinatorics, and number theory to be applied in the analysis of the properties of the data.

Firstly, notice that the periodic representation of the data on a circle, requires that we identify the vertices of the graph in a manner of modular arithmetic, where the repeated pattern in the data determines the structure. In the example above, we have a motif of 5 data points that repeats, so we work modulo 5, and identify v_1 and v_6. In the case, as in our example, that these two particular vertices are connected by an edge, we would get a loop in our circle representation. So our graph on a circle is not simple.

Secondly, given a collection of non-intersecting chords on a circle (as in Figure 2.1.5 above), define a **median chord** or a **median chord pair** (depending on the total number of chords) in a circle to be a chord (or chord pair) such that there are an equal number of chords on both sides of the chord (or chord pair). Clearly, such a chord does not necessarily exist. We would like to better understand the implications of the existence (or absence) of such a chord (or chord pair). We would then like to investigate the position of these median chords in the degree distribution and other properties of the Feigenbaum graph.

This and other questions arise with regards to the visibility graph of periodic time series.

2.1.2 Poincaré Decompositions of Visibility Graphs

We now turn our analysis to the visibility graph of a general time series. Much attention has been given to the question whether the visibility graph can distinguish between a time series from a Brownian motion (a stochastic time series) and a chaotic time series (a deterministic time series). The authors of [7, 8] have conjectured (based on some unpublished current work) that there is a clear and deep relation between the degree distributions of the visibility graph and the properties of the time series it came from.

We propose a couple of extensions to the visibility method (horizontal visibility will be addressed as well) which allows us to further analyze a time series that is not periodic. Consider a general time series $X = \{x_1,\ x_2, \dots\}$, and its visibility graph $G(X)$.

The first extension applies to 1-dimensional time series only (refer to Figure 2.1.6). Consider a general time series $X = \{x_1,\ x_2, \dots\}$, where the original data is represented by bars whose height is equal to the value of the corresponding data point. We define a **Poincaré cascade** $P_f(X)$ (analogous to a Poincré section) to be the following construction. Consider an increasing continuous function $f : \mathbb{N} \to \mathbb{R}^+$ such that, where $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{R}^+$ denotes the set of real positive numbers. Then the function f produces an increasing sequence $\{r_1,\ r_2,\ \dots\}$ of real positive numbers, which we may choose to terminate at some value. In particular, if the maximum $\max_X = \max\{x_1,\ x_2, \dots\}$ exists, then we may choose to terminate the sequence of real numbers at the largest number r_N such that $r_N \leq \max_X$.

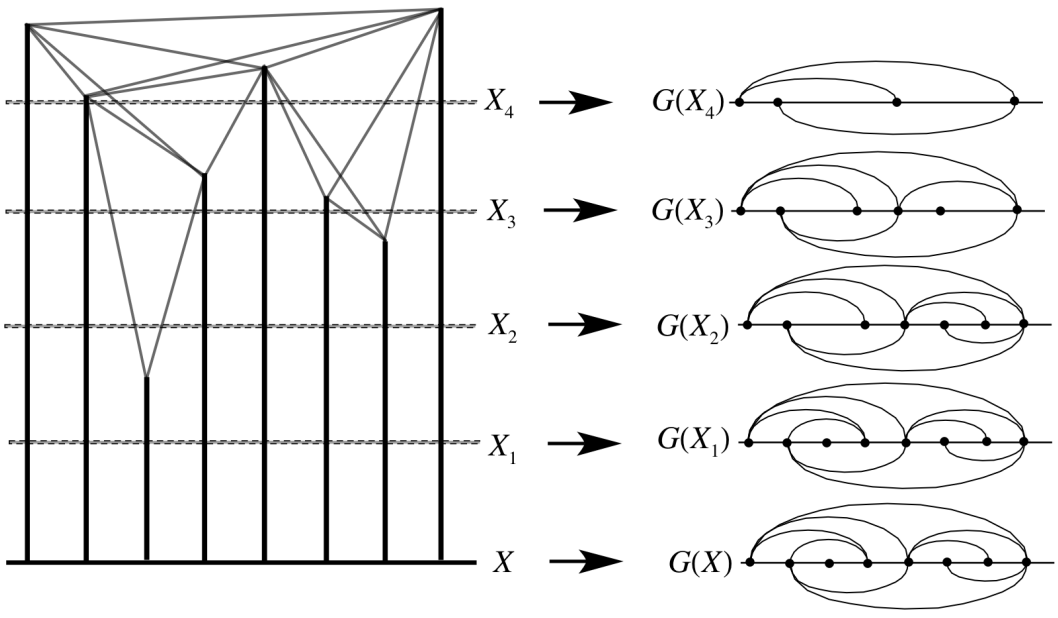


Figure 2.1.6: A circle representation of the visibility graph associated with a periodic time series. The repeated pattern is of the first 5 data points, so the visibility graph repeats a pattern after the first 5 vertices, so the circle shows 5 points with the sixth point identified with the first one, running in a counterclockwise way.

In the real world, our time series is bounded, so we will certainly have a supremum of X, which we could use to find N. So the function f has defined for us the sequence of

heights $r_1 \leq r_2 \leq \cdots \leq r_N$, at which we raise the x-axis in our time series. We disregard all points below the height r_i and consider the remaining data as defining a new series X_i (which is essentially a sub-series of the original time series X). So the Poincaré cascade $P_f(X)$ produces a sequence of time series $X \supseteq X_1 \supseteq X_2 \supseteq \cdots \supseteq X_N$ of inclusions of different time sub-series of the original time series. Then we apply the visibility algorithm to each X_i and obtain a sequence of corresponding visibility graphs $G(X_i)$ for $1 \leq i \leq N$.

Lemma 2.1.1. *Given a time series $X = \{x_1,\ x_2, \dots\}$, its visibility graph $G(X)$, a Poincaré cascade $P_f(X)$, and the sequence of sub-series $X \supseteq X_1 \supseteq X_2 \supseteq \cdots \supseteq X_N$ produced by $Pf(X)$, the corresponding visibility graphs $G(X_i)$ form the following sequence of subgraphs*

$$G(X) \supseteq G(X_1) \supseteq G(X_2) \supseteq \cdots \supseteq G(X_N).$$

Proof. While the Poincaré cascade does alter the vertex and edge sets for each subgraph by eliminating elements from both from the original graph, it does not affect the way the remaining vertices are connected. So, consider an element of $G(X_{i+1})$, we want to show that it is also an element of $G(X_i)$.

To prove that, we need to show that the vertex set $V_{G(X_{i+1})}$ is a subset of the vertex set $V_{G(X_i)}$ and the edge set $E_{G(X_{i+1})}$ is a subset of the edge set $E_{G(X_i)}$. Consider a vertex $v_k \in V_{G(X_{i+1})}$. In the Poincaré cascade process, we remove vertices from the visibility graph as we pass through a data point in the cascade, so vertices are discarded, so every vertex $v_k \in V_{G(X_{i+1})}$ is a vertex of $v_k \in V_{G(X_i)}$ that was not discarded, so in particular, $v_k \in V_{G(X_i)}$ and

$$V_{G(X_{i+1})} \subset V_{G(X_i)}.$$

Similarly, an edge $(v_k, v_j) \in E_{G(X_{i+1})}$, identified by the vertices it connects, means that the corresponding data point x_k and x_j are visible in X_{i+1}. The question whether (v_k, v_j) is an element of $E_{G(X_i)}$ is equivalent to asking whether x_k and x_j are visible in X_i. Again,

looking at the Poincaré cascade algorithm, at each sep of the algorithm, visibility is not created, only possibly removed by passing through points between two visible data points, which discards those data elements and the previously visible data elements would now be next to each other, thereby becoming invisible. So, if x_k and x_j are visible in X_{i+1}, this means they satisfy the visibility criterion in X_{i+1}, which means that x_k and x_j exist as elements of X_i and the points between them in X_{i+1} are also points between them in X_i, with the same relative height differences (see Figure 2.1.6), so they are also visible in X_i, so (v_k, v_j) is an element of $E_{G(X_i)}$ and

$$E_{G(X_{i+1})} \subset E_{G(X_i)}.$$

□

This process produces a way of decomposing a visibility graph, and thus one gets a lot more information about the structure of the graph. For example, calculating the degree distribution of each subgraph can shed light on the way the degree distribution is constructed. Essentially, we obtain a series of cross-sections of the visibility algorithm.

Of course, this decomposition, call it **Poincaré decomposition of a visibility graph** depends on the function f. In our examples, we chose f to be fairly simple, evenly spaced. Certainly, it would be very interesting to examine the connection between f and the degree distribution.

Note that this kind of bottom-up decomposition will certainly preserve inclusion, as it will not create any new edges or introduce new vertices. It would be interesting to study a top-down way of decomposing a visibility graph. This is one of our future projects.

Remark 2.1.2. Note also that the same lemma holds for horizontal visibility graphs.

2.1.3 Bundled Visibility Graphs

The second extension to the visibility algorithm we would like to propose makes use of the weighted directed version of the visibility graph. Again, consider a time series $X = \{x_1,\ x_2, \dots\}$.

Consider the weighted, directed generalization of G, with the directions of edges assigned according to our definition, and not what we called a standard definition, which is according to slope. In this section, we introduce the idea of **bundles**, which allow us to combine the weighted visibility graph, the directed visibility graph, and the Poincaré decomposition of a visibility graph all in one. Recall that our idea of orienting the edges in a visibility graph came from the corresponding connections in the time series pointing "downward", that is, directed edges point in such a way that they originate from "heavier weighted" vertices (corresponding to taller bars in the time series) to lighter ones.

Now, the Poincaré cascade in the Poincaré decomposition scans a time series and essentially throws away data points in a gradual way, and similarly for the associated visibility graph. We would like to change this process in a way that takes into account the data points instead of throwing them away. The idea is to bundle them together with others as we go through the cascade in such a way that they are still accounted for in the end.

We will need to use the weights of vertices and the orientations of edges for this purpose.The question is how to bundle up the data points that we pass through. Our criterion for this depends on the way the time series bars are positioned. If the cascade passes through a bar, we look at the bar(s) that it is connected to, and find the best candidate that the bar will be bundled to. The criterion takes into account the horizontal distance to that bar, as well as the vertical distance (that is, the relative weight of the data point).

More formally, the **bundle algorithm** can be described in the following way. Consider a data point x_i that a given Poincaré cascade has passed over and that needs to be bundled.

Take the collection $C(x_i)$ of **all points that the point is visible from**. Note that if this is an empty set, the point x_i will not be bundled but merely discarded as we pass across it. This will affect the visibility of the remaining data. In our definition, $C(x_i)$ is the set of all vertices that connect with x_i, in downward direction (we would by then have eliminated any data points that x_i connects to in the cascade steps prior to the current one. For each element $x_j \in C(x_i)$, we have **the horizontal distance** h_j between the bars for x_i and x_j in the time series, as well as **the vertical distance** v_j which is the difference in the heights of x_i and x_j (since each cascade step is a horizontal slice, the relative height difference remains the same, whether measured from the first or current cascade step).

At this point, it is our decision how to use h_j and v_j to decide to which x_j we will bundle x_i. We believe that the relative weight difference (measured by the relative height difference v_j) is far more significant, since the horizontal distance between x_i and x_j is not as significant in the end, if they are visible. Here, h_j is just a positive integer, counting how many data slots are there between x_j and x_i, while v_j is a real number.

Note that v_i and v_j are heights measured with respect to the current heights as a result of the cascade step!

Therefore, we form the following set over all elements in $C(x_i)$:

$$S = \left\{ \frac{v_j}{h_j} \;\middle|\; \text{for all } x_j \in C(x_i) \right\}$$

and then take the element(s) x_j such that $\dfrac{v_j}{h_j} = \max(S)$. This is a finite set, so there will be a maximum, and in the case there is more than one element achieving the maximum (there can be more than two, by the way!), we "split" x_i evenly among them all.

Now, the process of bundling an element x_i to an element x_j essentially removes x_i from the new (bundled) time series, and adjusts the weight of x_j to account for the presence of x_i.

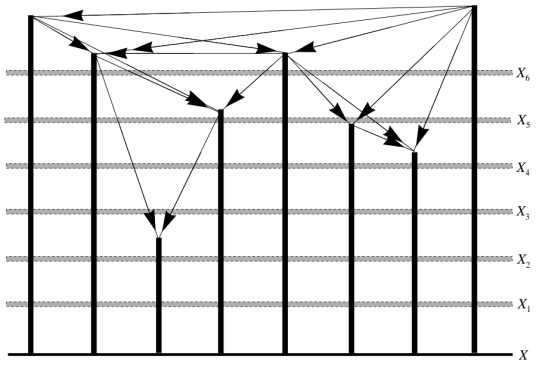
Again, we have to use our discretion here, how exactly do we change the height of x_j? Do we just add the height of x_i? This would create a significant increase in the height of the data bar for x_j, which would force more elements to be bundled with x_j. Preliminary estimates seem to show that this creates an unbalanced growth in a few elements that receive bundled elements, and this alters the overall characteristics of the data significantly. Since we are trying to study the data, and not modify it, we choose to adjust the height of x_j as follows.

If x_j is the sole element achieving a maximum for the set S, that is, the one point that x_i would be bundled to, we set the new height x_j to equal

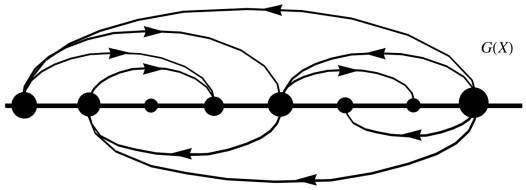
$$v_j + \frac{v_i}{h_i}\frac{v_i}{v_j},$$

thereby taking into account the relative position of x_i to x_j, as well as the relative height difference of $_i$ and x_j. One can easily see that data points of similar height next to each other will get bundled right away and significantly combine their heights, while points of differing heights would get bundled together but the resulting height will not be as significant. Since all heights are measured relative to the lat cascade step, in fact, the bundle algorithm is determined by the cascade. Importantly, once an element is bundled, its spot in the time series is destroyed, so that the elements adjust not to leave a gap. This clearly has an effect on subsequent bundles. To complete the bundle algorithm, we set the rule that we bundle all points that are just below the last cascade step, that is the ones that fall below the last but above the second-to-last step. Moreover, we bundle points in increasing order, as it is possible to bundle a point to another, and then bundle that to another, so it is important to actively update the heights of the bars at each bundle event. In a discrete cascade, we would have a certain number of data points between cascade steps to bundle. In a continuous cascade, for f continuous in $P_f(X)$, we would have individual points to bundle each time, and the bundle algorithm extends to this process as well.

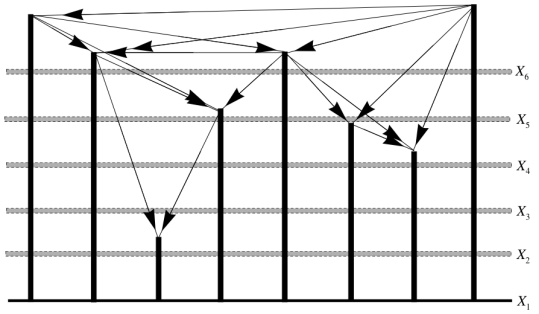
If we compare the result of a standard Poincaré cascade that we described above (see Figure 2.1.6) in which we simply discard data, and a Poincaré cascade with a bundle algorithm, in which we go through an elaborate construction to bundle up data instead of discarding it, we see that after 5 steps, we would essentially have a trivial visibility graph in the first case, while in the second case, we have some more information preserved, in the form of visibility or a different weights configuration.



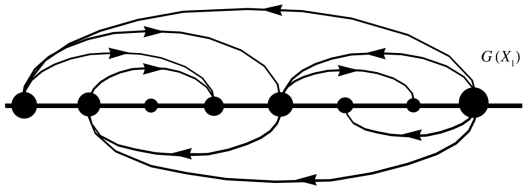
(a) The initial time series with a Poincaré cascade.



(b) Associated weighted oriented visibility graph.



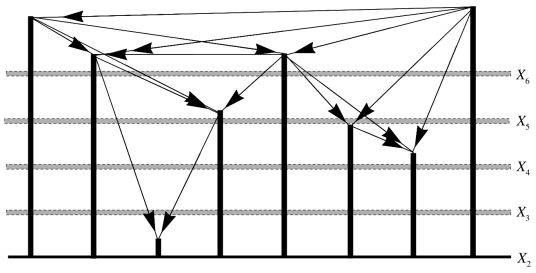
(c) Moving along the Poincaré cascade. No points have been crossed, no changes in the visibility structure.



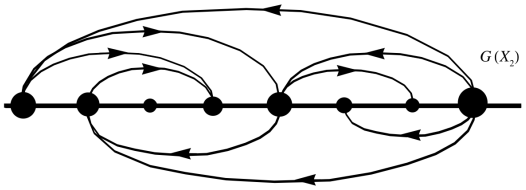
(d) Associated weighted oriented visibility graph.

Figure 2.1.7: Bundle algorithm with a Poincaré cascade, first 2 steps. At each step, we move up along the cascade levels, indicated by equally spaced horizontal bars. Notice that if we do not pass through a point, there is no change in the associated visibility graph, even though the values of the weights do change, essentially the same amount is subtracted from each at each cascade level we pass.

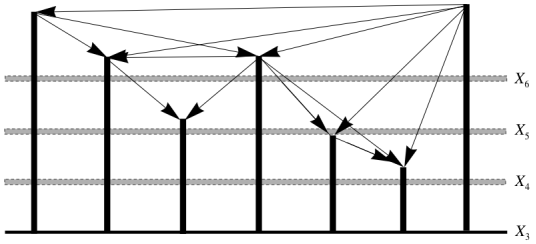
Next, we observe the changes in the visibility graph as we progress further along the cascade.



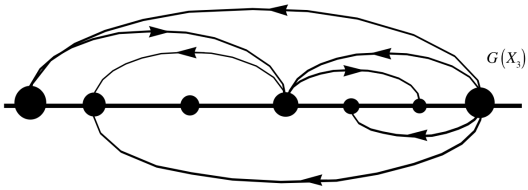
(a) Moving along the Poincaré cascade.



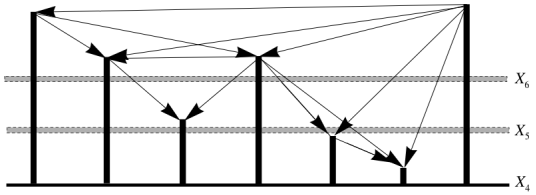
(b) Associated weighted oriented visibility graph.



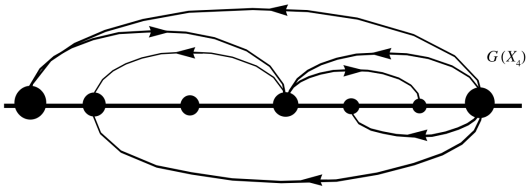
(c) Moving along the Poincaré cascade.



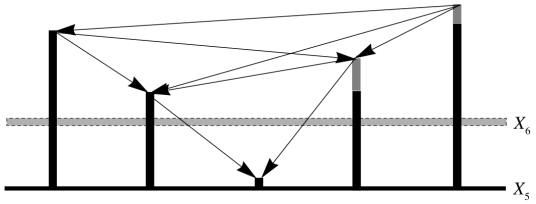
(d) Associated weighted oriented visibility graph.



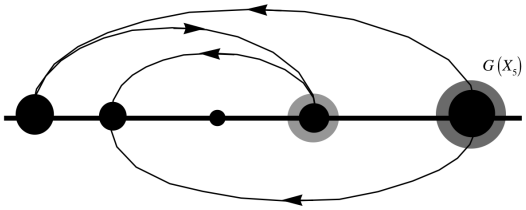
(e) The initial time series with a Poincaré cascade.



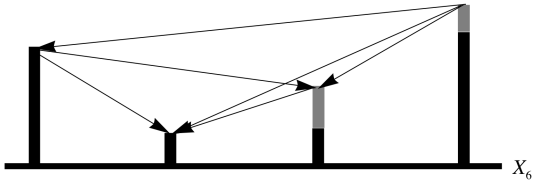
(f) Associated weighted oriented visibility graph.



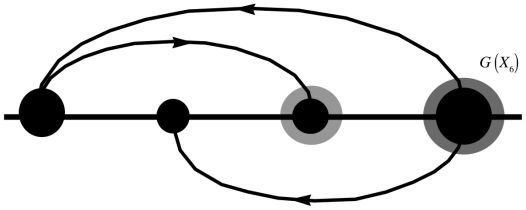
(g) Moving along the Poincaré cascade.



(h) Associated weighted oriented visibility graph.



(i) Moving along the Poincaré cascade.



(j) Associated weighted oriented visibility graph.

Figure 2.1.8: Bundle algorithm with a Poincaré cascade, remaining 5 steps.

Remark 2.1.3. The above definitions and constructions directly carry over to Feigenbaum graphs.

At this point, we are ready to implement our ideas and use the data generated in Chapter 1 to test our ideas on real chaotic time series.

2.2 Applications to 1-Dimensional Time Series

We have two fronts of work here: understanding how our ideas from Section 2.1 apply to general 1-dimensional time series, such as chaotic time series (possibly random or stochastic time series) and understanding the Feigenbaum graphs scenario which particularly uses the horizontal visibility algorithm to understand a chaotic system through its bifurcation diagram. We develop *Mathematica* routines to test both on the examples we built in Chapter 1.

```
In[402]:= DATALIST = {};
          x1 = 0.28;
          NN = 5;
          Do[x1 = 3.7 * x1 * (1 - x1); DATALIST = Append[DATALIST, x1], {i, NN}];
          Show[{
            ListPlot[DATALIST, Joined → True, PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
            ListPlot[DATALIST, PlotStyle → {AbsolutePointSize[7], Black, AspectRatio → 0.7, Axes → True}]}]
```

Out[406]=

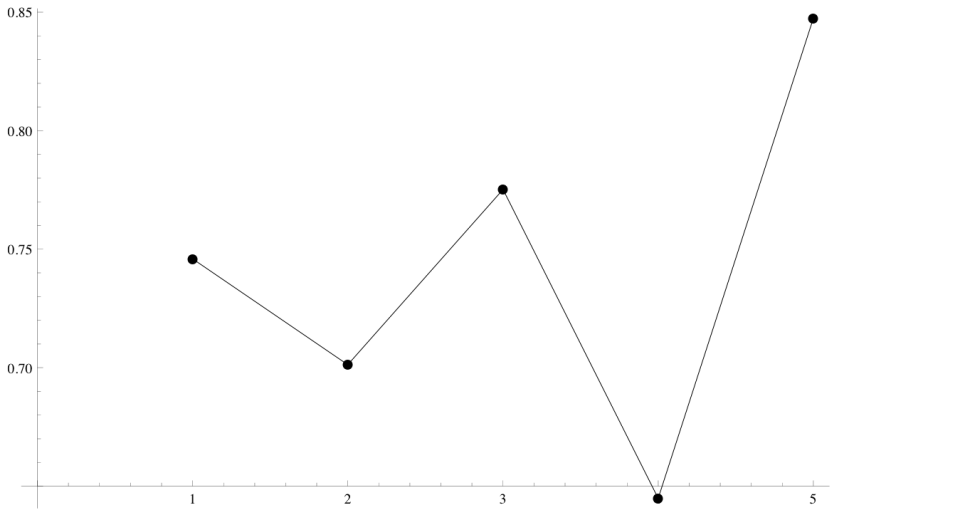


Figure 2.2.1: The figure shows 5 points generated for the logistic equation $x_{n+1} = rx_n(1 - x_n)$ with $r = 3.7$, combining two plots, a connected plot, and a discrete point plot. Note that this is not a visibility plot.

Consider the logistic equation $x_{n+1} = rx_n(1 - x_n)$ in its quasi-chaotic region with $r = 3.7$. Firstly, we create a data storage list, set the initial value, a counter for the number of points we will generate, and a recursive loop for the logistic equation.

```
T1[{i_, k_, LIST_}] :=
  If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, i + 1, k - 1}]],
   1,
   0];

T2[{i_, k_, LIST_}] :=
  If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, k + 1, i - 1}]], 1,
   0];
{T1[{1, 3, DATALIST}], T1[{1, 5, DATALIST}], T1[{3, 5, DATALIST}], T1[{4, 4, DATALIST}],
 T1[{2, 3, DATALIST}]}
{1, 1, 1, 0, 0}
{T2[{3, 1, DATALIST}], T2[{5, 1, DATALIST}], T2[{5, 3, DATALIST}], T2[{4, 4, DATALIST}],
 T2[{5, 4, DATALIST}]}
{1, 1, 1, 0, 0}
Table[
  ((i <= k) ⋀ T1[{i, k, DATALIST}]) ⋁ ((k < i) ⋀ T2[{i, k, DATALIST}]), {i, 5}, {k, 5}] //
 MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

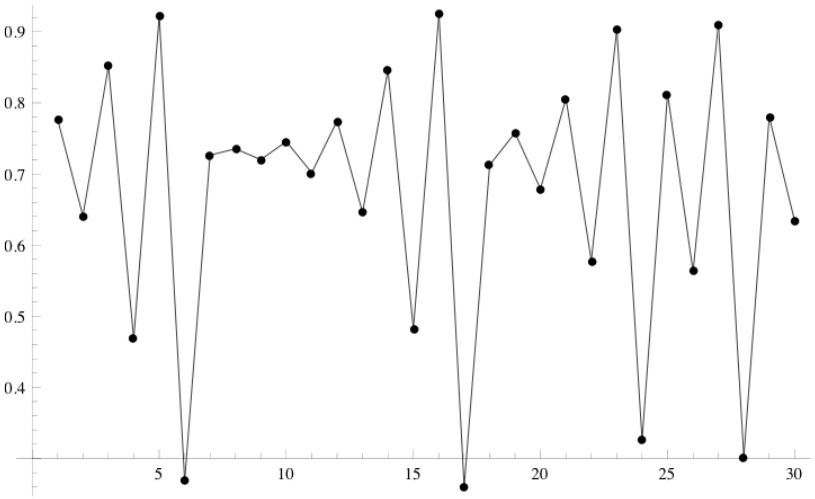
Figure 2.2.2: Then we define a visibility test $T1$ that takes on two numbers, finds the corresponding elements in the list, and checks the visibility criterion for all points between the two elements. It checks that the two elements are not equal, not consecutive (since visibility does not apply to neighbors), and then applies the visibility algorithm to all the points.

Note that Figure 2.2.8 reveals two important visibility connections: between the 5th and the 16th point and the 16th and 27th point. Points that are not visible are plotted to the side by the Mathematica routine.

In[475]:=
```
DATALIST = {}; x1 = 0.3; NN = 30;
Do[x1 = 3.7 * x1 * (1 - x1); DATALIST = Append[DATALIST, x1], {i, NN}];
Show[{
  ListPlot[DATALIST, Joined → True,
   PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[DATALIST,
   PlotStyle → {AbsolutePointSize[5], Black, AspectRatio → 0.7, Axes → True}]}]

GraphPlot[Table[((i <= k) ⋀ T1[{i, k, DATALIST}]) ⋁ ((k < i) ⋀ T2[{i, k, DATALIST}]),
  {i, NN}, {k, NN}], VertexLabeling → True, Method → "CircularEmbedding"]
```

Out[477]=



Out[478]=

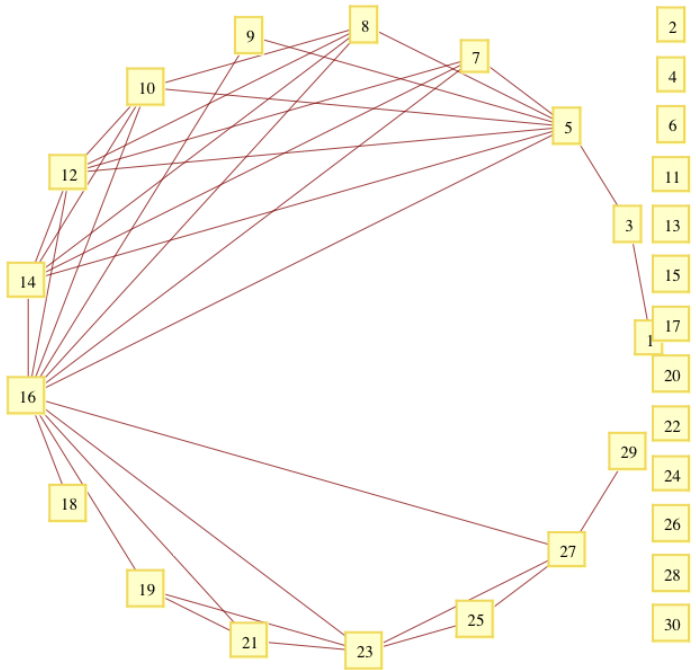
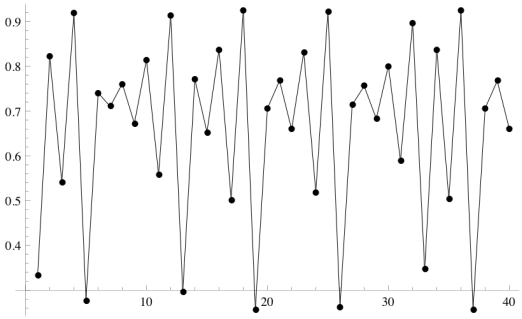


Figure 2.2.3: We suppress the $T1$ and $T2$ commands and show the circle representation of the 30 points.

It would be interesting to see more examples that can illustrate the strength of using a circle presentation of the visibility graph. In this case, the data is not periodic, but consisting of only a few points. In the case of very large non-periodic data sets, it may not be very useful to try to plot the points on a circle, but we may look at density of chords, if numerical tools are available.

```
DATALIST = {}; x1 = 0.1; NN = 40;
Do[x1 = 3.7 * x1 * (1 - x1); DATALIST = Append[DATALIST, x1], {i, NN}];
Show[{
  ListPlot[DATALIST, Joined → True,
   PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[DATALIST,
   PlotStyle → {AbsolutePointSize[5], Black, AspectRatio → 0.7, Axes → True}]}]
```

Out[34]=




```
T1[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And,
     Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, i + 1, k - 1}]], 1, 0];
T2[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And,
     Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, k + 1, i - 1}]], 1, 0];
MX = Max[DATALIST]; MN = Min[DATALIST];
h = (MX - MN) / 5;
Do[
  DATALIST1 = {};
  LL = Length[DATALIST];
  Do[If[DATALIST[[m]] - (MN + t * h) > 0,
    DATALIST1 = Append[DATALIST1, DATALIST[[m]]]], {m, LL}];
  Print[
   Show[{ListPlot[DATALIST1, Joined → True,
       PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
     ListPlot[DATALIST1,
      PlotStyle → {AbsolutePointSize[5], Black, AspectRatio → 0.7,
        Axes → True}]}]
   GraphPlot[Table[((i <= k) ⋀ T1[{i, k, DATALIST1}]) ⋁
      ((k < i) ⋀ T2[{i, k, DATALIST1}]), {i, Length[DATALIST1]},
     {k, Length[DATALIST1]}], VertexLabeling → False,
    Method → "CircularEmbedding"]], {t, 0, 4}];
```

Figure 2.2.4: Next, we show a Poincaré cascade for 40 points generated by the logistic equation. This is the Mathematica routine that generated the calculation, 5 equally spaced cascade steps between the minimum and maximum height in the set of 40 values..

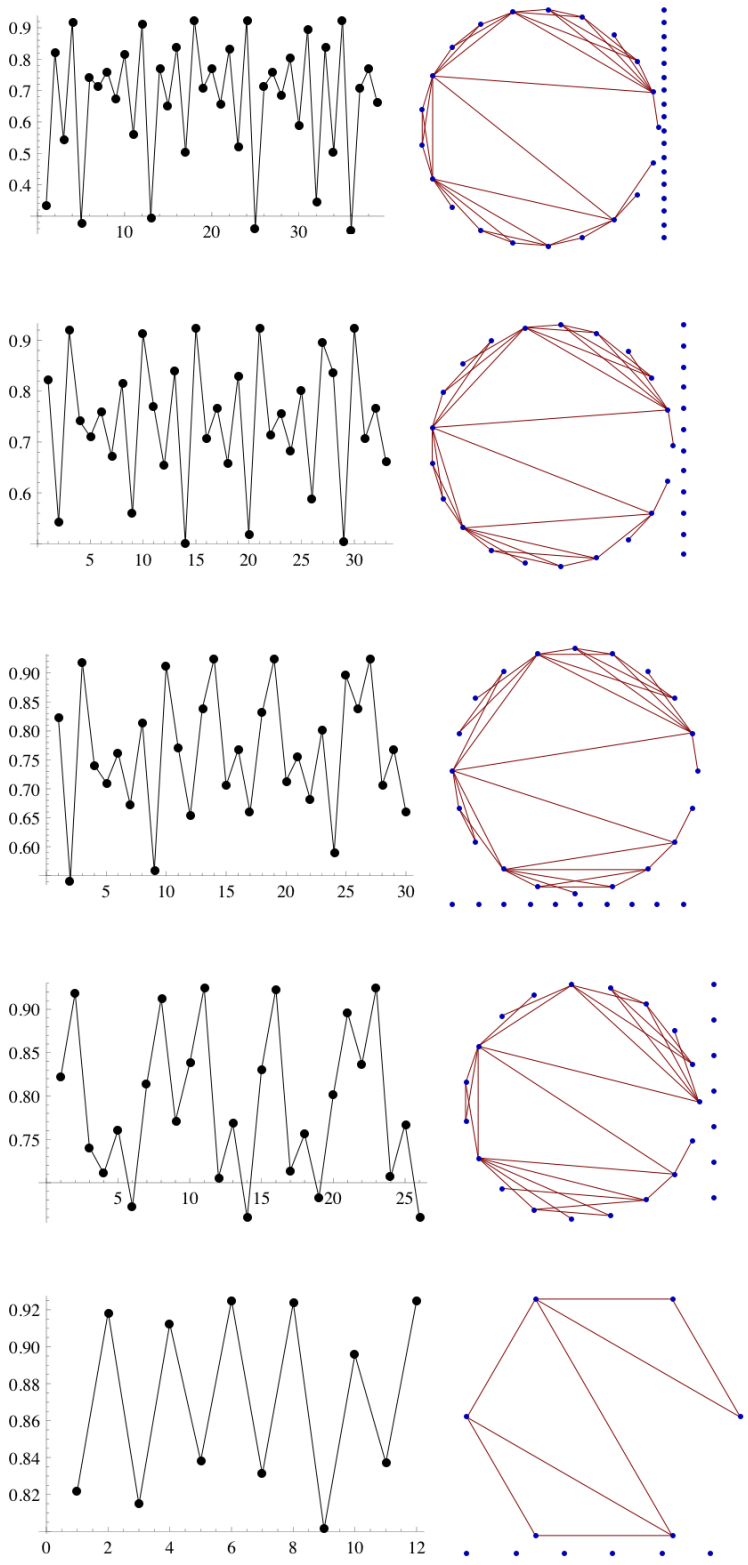


Figure 2.2.5: A Poincaré cascade, the horizontal values show the number of elements that have remained after each step, using the Mathematica routine to map a circular graph.

So far, we have no way of arranging the vertices in a line and having the edges be half-ellipses. There is an interesting observation that we can make, however. Taking the last graph (with only 12 data points) and overlaying it over the original data, we see that the larger behavior of the data is very well preserved.

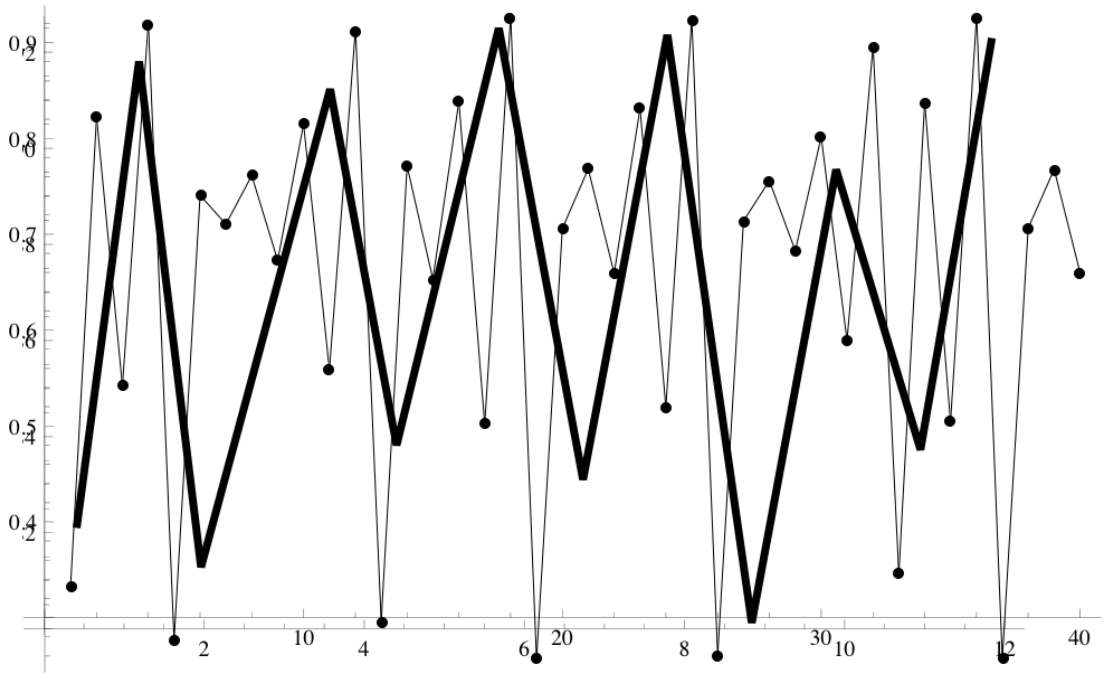


Figure 2.2.6: Comparison between original and final data after 5-step Poincaré cascade. Original data (40 data points) and final data (2 data points).

Also, note that this graph does not show visibility in the data, just connects the data points. We are working on a Mathematica routine that would accomplish that, and show the visibility lines between different data points. Also, we would like to be able to draw arrows, and add weights to the data points, and be able to depict that as well.

It is important to keep in mind, though, that this will become less and less useful as we increase the number of data points. Already with values on the order of 10,000 points, any kind of meaningful visualization of the original data is very challenging. Using a Poincaré cascade, or even combining it with a bundle algorithm, we can reduce the number of

points, and hopefully be able to visualize a significant representation of our data with fewer data points.

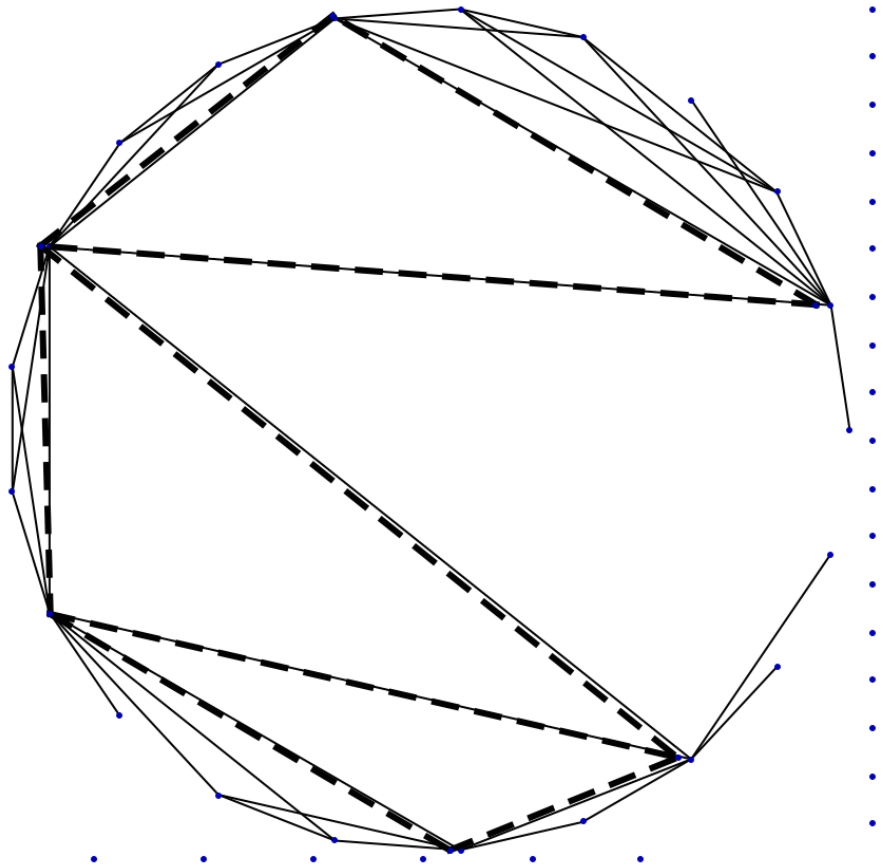


Figure 2.2.7: Comparison between original and final data after 5-step Poincaré cascade. Using the circular representation of the visibility graphs, we overlay (with a mall rotation) the original graph (thin solid lines) and the final graph (dashed lines) after a 5-step Poincaré cascade.

Next we show a few examples and further numerical experiments we have done.

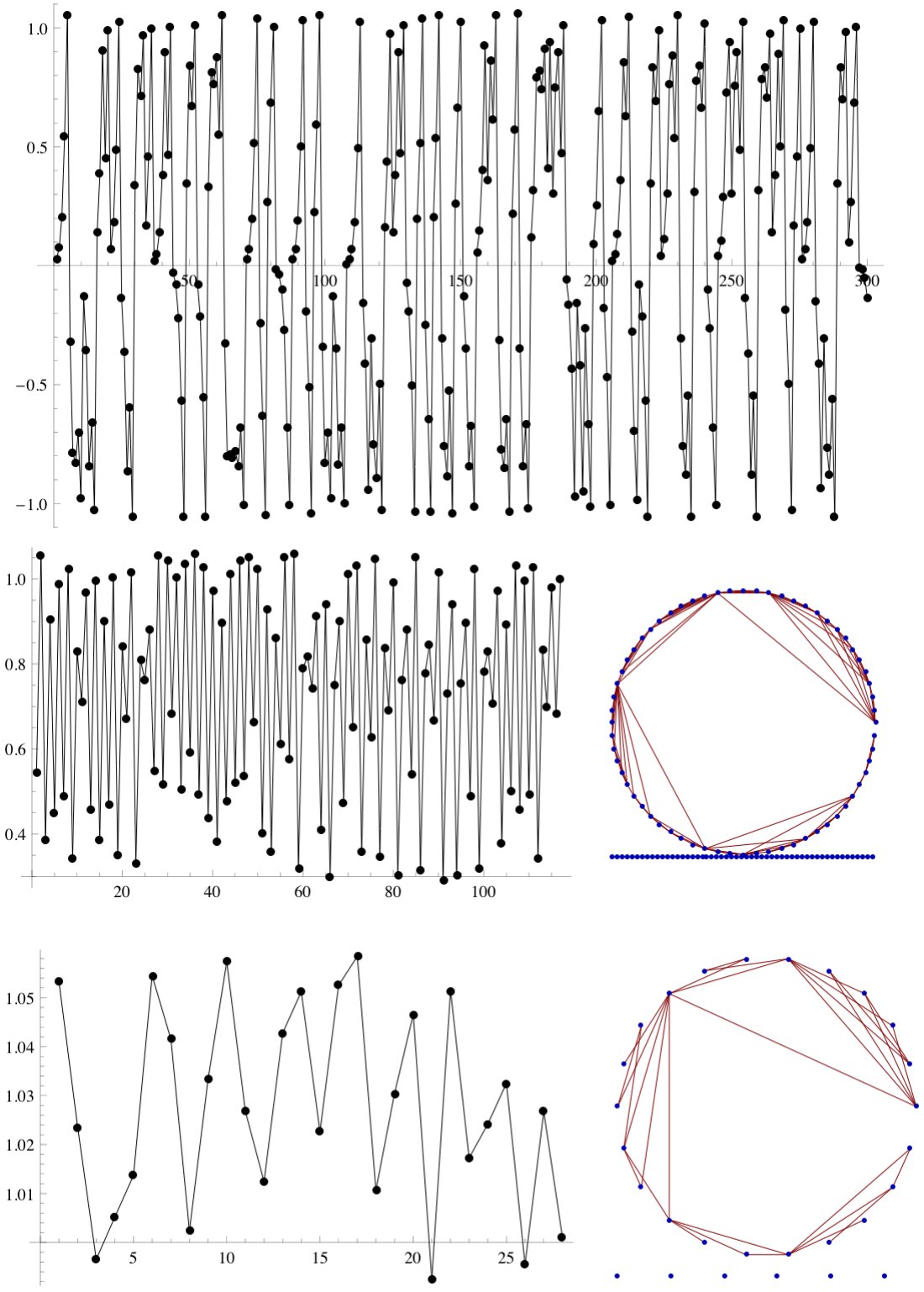


Figure 2.2.8: A 30-step Poicaré cascade for the cubic equation $x_{n+1} = 2.75x_n(1 - x_n^2)$, wiht the original data, the data after 20 steps, and the final data shown, with a circle representation of the visibility graph.

3
Multi-Dimensional Visibility Graphs

3.1 Higher-Dimensional Visibility Graphs

We would like to address the question of working with higher-dimensional time series. Consider an n-dimensional time series $\mathbf{X} = \{\mathbf{x_1},\ \mathbf{x_2},\ \dots\}$, where $\mathbf{x_i} = (x_i^1, x_i^2, \dots, x_i^n)$ is an element of $\mathbb{R}^n$.

There are several ways to define visibility and extend the visibility algorithm. We are first going to address the idea of **higher-dimensional visibility** for a higher-dimensional time series.

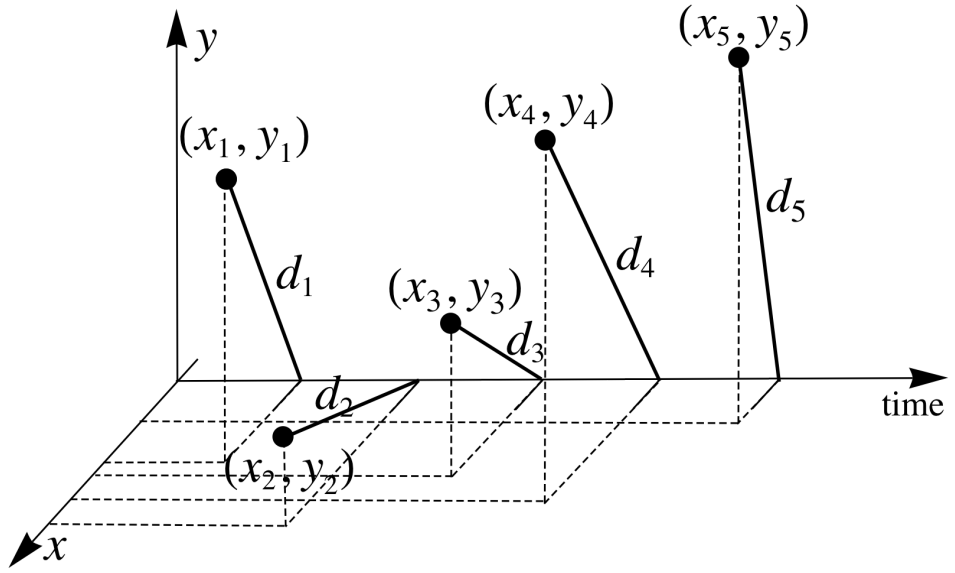
3.1.1 Magnitude Visibility Algorithm

One attempt might be just to convert this problem to the usual idea of visibility, where we define a new time series, call it $X = \{d_1,\ d_2, \dots\}$ consisting of the corresponding magnitudes of the vectors in $\mathbf{X}$.

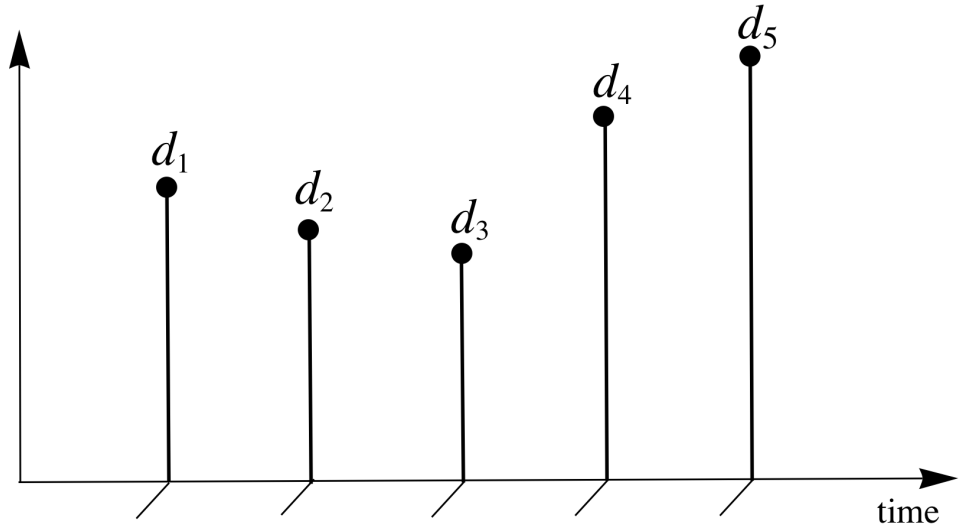
$$d_i = \sqrt{\mathbf{x_i} \cdot \mathbf{x_i}}$$

and apply the visibility algorithm to that series. In this case, we call this visibility **magnitude visibility**, and it clearly loses some information, even if we use the weighted

version of the visibility graph. Because different time series can produce the same series of magnitudes.



(a) 2-dimensional time series along a time line.



(b) Associated magnitude time series.

Figure 3.1.1: Magnitude visibility algorithm. Given a 2-dimensional time series, we obtain a 1-dimensional time series of magnitudes, and we apply standard visibility algorithms for its analysis.

Moreover, we can imagine different vector with the same magnitude occurring repeatedly, that is, the original n-dimensional time series might be exhibiting oscillatory behavior while the corresponding magnitude time series would be showing constant behavior. We would have to be careful then to interpret constant behavior in the magnitude time series to mean that elements in the original time series are restricted to the surface of an n-dimensional sphere. Similarly, oscillatory behavior in the magnitude time series represents quasi-oscillatory behavior in the original series, the best we could say would be that the data jump around among a number of concentric spheres in $\mathbb{R}^n$ in an oscillatory pattern, but we would have no idea of the exact values that the original time series takes. We could certainly carry the information along and assign n-tuples to the corresponding vertices in the magnitude visibility graph, still, the graph is generated according to visibility in the magnitude time series, and this is done already after some information is lost already.

Nevertheless, this is a valid way of accessing higher-dimensional time series via the well-established visibility (or horizontal visibility) method. There are a few questions that we are exploring. The first question is whether the magnitude visibility graph an help distinguish different behavior in the higher-dimensional time series, such as constant, periodic, chaotic, stochastic. We can clearly answer some of these, such as periodic behavior in the original time series would be detected in the magnitude visibility graph with the rare exception of oscillations between distinct vectors of precisely the same magnitude. The magnitude visibility graph will clearly distinguish between periodic and chaotic or stochastic behavior, as none of the later two are likely to be restricted to concentric spheres in $\mathbb{R}^n$. To answer these questions precisely, however, we need to understand the concept of chaotic and stochastic much better. In particular, we need to begin with the simpler question whether a 1-dimensional system restricted to a finite number of values can be chaotic or stochastic. Extending this to the 2-dimensional case, we ask whether a 2-dimensional system restricted to take on values only on a finite number of concentric circles can be

quasi-chaotic or stochastic. This extends analogously to higher-dimensional systems. It is clearly very very unlikely that a real-world n-dimensional system would exhibit chaotic or stochastic behavior while restricted to a finite number of concentric $(n-1)$-dimensional spheres, however, any counterexample would present a negative answer to the above questions, thus rendering the magnitude visibility algorithm blind to fundamental changes in the behavior of the original system.

As part of our current projects, we are carrying out a few examples to give us a better understanding of how much information we could extract from the magnitude visibility algorithm.

3.1.2 Higher-Dimensional Visibility Algorithm

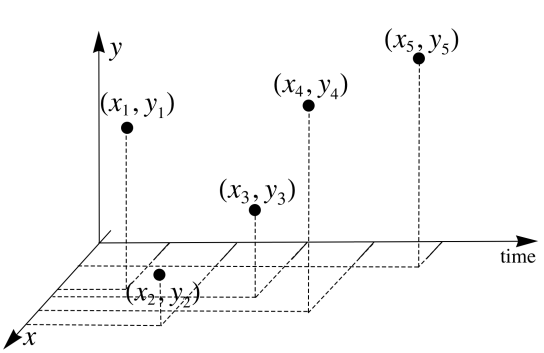
Next, we propose a **higher-dimensional visibility criterion** which we believe does address many of the shortcomings of the magnitude visibility. Consider again the n-dimensional time series $\mathbf{X} = \{\mathbf{x_1},\ \mathbf{x_2},\ \dots\}$ with $\mathbf{x_i} = (x_i^1, x_i^2, \dots, x_i^n)$. Once can consider this as an n-tuple of series, labeled $X^1 = \{x_1^1,\ x_2^1, \dots\}$, $X^2 = \{x_1^2,\ x_2^2, \dots\}$, $\dots$, $X^n = \{x_1^n,\ x_2^n, \dots\}$. We say that $\mathbf{x_i}$ is visible from $\mathbf{x_j}$ if x_i^k is visible from x_j^k in X^k in the usual sense for $1 \leq k \leq n$ (see Figure 1.8.1).

For convenience, we represent $\mathbf{X}$ in $\mathbb{R}^{n+1}$, where the extra dimension is time t. In Figure 3.1.2a, $n = 2$, so we represent the time series with the time direction along the x-axis, as in the 1-dimensional case. The two dimensions for the data are assigned along the y-axis and z-axis.

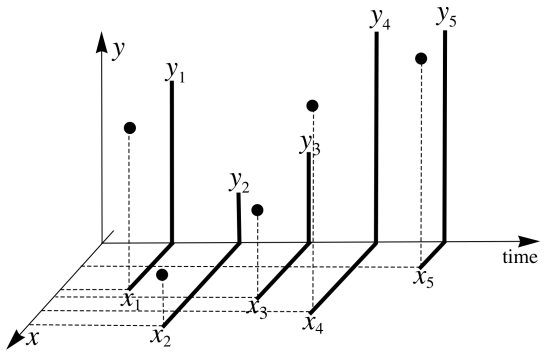
Notice that the concern that the projections data may not be in the same order as the time-ordered original data, but is instead ordered in terms of magnitude is wrong. In the example below, we have a 2-dimensional time series represented in 3-dimensional space. Since the third dimension is time, the projections of the data onto the x-time plane or

y-time plane yield another time series, ordered according to the original data so that the data set X^k can be analyzed in the standard way.

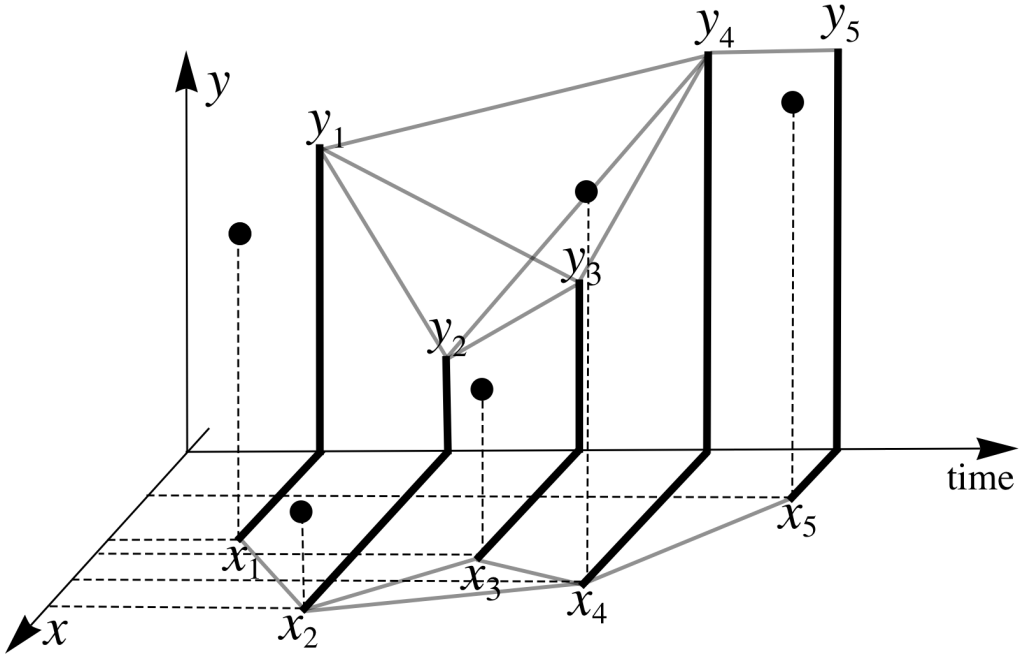
This condition, unlike magnitude visibility, is rather restrictive. We can consider **partial visibility** by requiring that x_i^k is visible from x_j^k in X^k in the usual sense for some (but not all) k in $\{1, 2, \ldots, n\}$.



(a) 2-dimensional time series and a time axis.



(b) The projections of the original data set



(c) Visibility reduces to componentwise visibility.

Figure 3.1.2: Projections of a time series.

In the example in Figure 3.1.2c, one can observe that there is partial visibility between the points (x_1, y_1) and (x_3, y_3). While y_3 is visible from y_1 due to y_2 being fairly small, x_3 is not visible from x_1 because x_2 is very large and obstructs visibility.

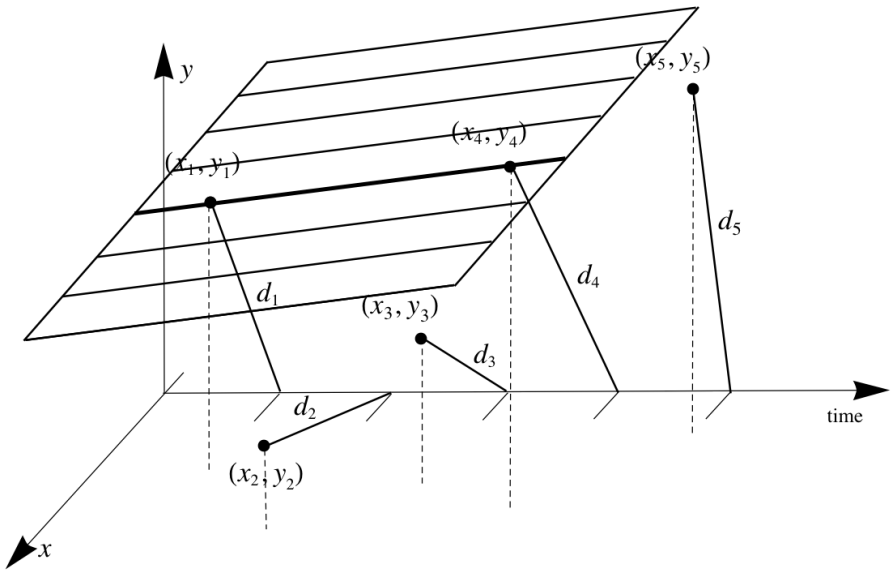
We can assign shades of gray to degrees of partial visibility and still construct a visibility graph. In the case of (x_1, y_1) and (x_3, y_3), we have 50% visibility.

3.1.3 Planar Visibility Algorithm

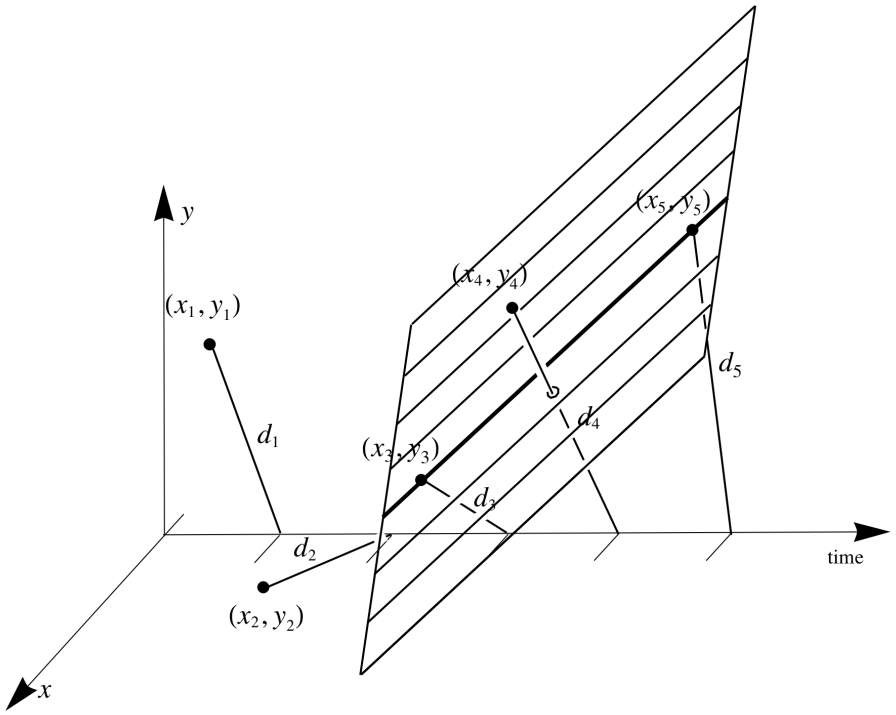
The last definition of visibility we would like to consider is **planar visibility**. Again, consider n-dimensional time series $\mathbf{X} = \{\mathbf{x_1},\ \mathbf{x_2},\ \dots\}$ with $\mathbf{x_i} = (x_i^1, x_i^2, \dots, x_i^n)$, represented in $\mathbb{R}^{n+1}$. For each point $\mathbf{x_i} \in \mathbf{X}$, consider its projection onto the "time" axis in $\mathbb{R}^{n+1}$, which, depending on the scaling, would have a value of i or the i-th point along the positive "time" direction. Denote by d_i the line segment connecting $\mathbf{x_i}$ with its projection along the time axis. Note that the d_i gives the magnitude of x_i in $\mathbb{R}^n$ (see Figure 3.1.3a).

We say that $\mathbf{x_i}$ is **plane visible** from $\mathbf{x_j}$ for $i < j$ if there exists an n-dimensional plane $T \subset \mathbb{R}^{n+1}$ containing the points $\mathbf{x_i}$ and $\mathbf{x_j}$ such that for all m with $i < m < j$, the subspace T does not intersect. In order to understand the idea, let us take a look at a few values for n. If $n = 1$, we have the standard case of 1-dimensional time series presented in $\mathbb{R}^2$. Then T is a line, and for each data point x_i, the d_i is the vertical bar to the time axis (in our case, the x-axis). The condition on T for any two points x_i and x_j to be planar visible is equivalent to the condition that x_i and x_j be visible in the standard sense.

The more interesting case of $n = 2$ concerns us with 2-dimensional time series, $\mathbf{X} = \{\mathbf{x_1},\ \mathbf{x_2},\ \dots\}$ with $\mathbf{x_i} = (x_i, y_i)$ for all t. Let us the represent this in the staircase scenario, so the time axis is the z-axis, and the positive direction is along the positive z-direction. So our data "grows" upward with time.



(a) 2-dimensional time series and a visibility plane that establishes visibility.



(b) An example of a visibility plane that violates planar visibility. Depending on the geometry and the data points, we can prove whether such a plane exists or not.

Figure 3.1.3: Planar visibility criterion for 2-dimensional time series. We consider the line connecting the two data points $\mathbf{x_i}$ and $\mathbf{x_j}$ and the infinitely many planes containing it (in higher dimensions, this does not apply in the same way, but the important idea is that the plane is 1 dimension smaller than $\mathbb{R}^{n+1}$. In 2 dimensions, we are looking for one of these plains to not intersect the d_k for $i \leq k \leq j$, notice that d_i and d_j cannot intersect the plane either!

The condition that T does not intersect any of the d_k for $i \leq k \leq j$. Note that this is a different approach from the magnitude visibility as there might be an intermediate data point with very large magnitude but well outside of the plane T.

Theorem 3.1.1. *The three notions of visibility: magnitude visibility, higher-dimensional visibility, and planar visibility all generalize the standard notion of visibility.*

Proof. In the magnitude visibility definition, if the data are 1-dimensional, their magnitudes as defined in the algorithm coincide with the height of the data bars in the standard setup, so we have the usual visibility (see Figure 3.1.1b).

In the higher-dimensional visibility definition, if the data are 1-dimensional, their projection onto the (only) x-time plane coincides with the height of the data bars in the standard setup, so we have the usual visibility (see Figure 3.1.2c).

In the planar visibility definition, if the data are 1-dimensional, the "plane" set reduces to a line since it is supposed to be n-dimensional and when $n = 1$, we obtain a line connecting the points in the x-time , so we again have the usual visibility (see Figure 3.1.3a). □

3.1.4 Further Remarks

The remaining questions concern developing these ideas and testing various aspects of them in the following sections.

The fist one is with regards to the next step: converting from a time series to a graph. What does it mean to have vertices in 3 dimensions?

One of the main questions we are concerned with is the difference between representing a time series in a space including an extra dimension for time, in essence, developing the series in time, instead of simply representing it in its natural phase space. For 1-dimensional time series, this means plotting points on the real line and it would be very difficult to extract any information from that. For higher-dimensional time series, however, this is a

very relevant question, especially considering bounded chaotic systems, for example, in 2 dimensions we are looking at systems such as the Hénon map or the Burgers map, and Poincaré sections of various 3-dimensional systems, such as the Lorenz system and the Sprott maps.

For now, we work with the extra time dimension and deal with higher-dimensional graphs, which in itself is a challenge. At some later stage, we will attempt to remove the time dimension and deal with visibility graphs.

3.2 Higher-Dimensional Examples

3.2.1 2-Dimensional Time Series Examples

We can study higher-dimensional time series now, and we have plenty of examples of 2-dimensional time series in particular.

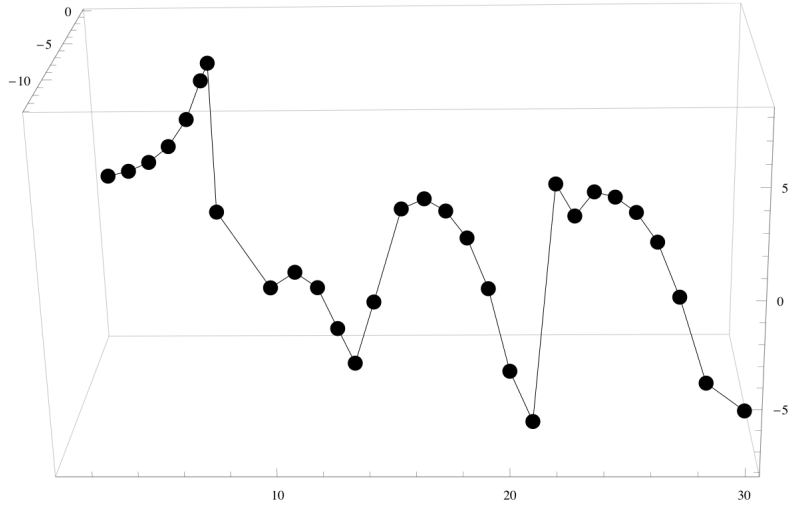
The first example we would like to look at is the Burgers map.

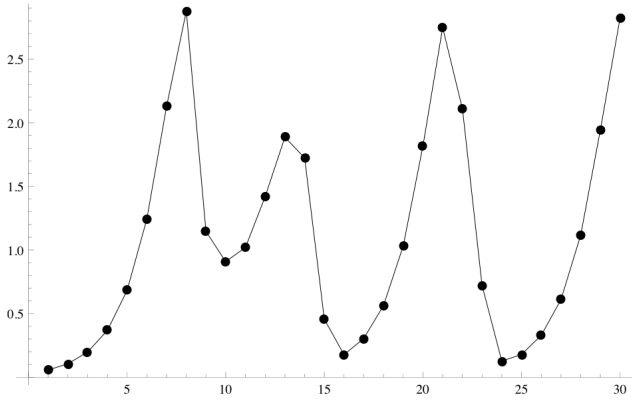
We have set up a Mathematica program that iterates the system of difference equations a given number of times (in the calculations below, we have 30 iterations) with a given set of initial values. Since we are still in the process of developing our tools, we are not dealing with large numbers of iterates.

We would like to first give a few example of magnitude visibility. This is a fairly straightforward generalization of the standard 1-dimensional visibility algorithm.

We generate the data points (with a scaling factor so that they are visible when plotted versus time). In the figure below, this is the top plot. Computing the magnitudes is immediate, in the second plot, and then we have the visibility graph, again plotted in a circular graph plot.

```
In[947]:= (*BURGERS MAP*)
DATALIST = {};
DATALIST1 = {};
x1 = 0.03;
y1 = 0.03;
NN = 30;
Do[
  x2 = 0.1 * x1 - y1^2;
  y2 = 1.865 * y1 + x1 * y1;
  DATALIST = Append[DATALIST, Sqrt[x2^2 + y2^2]];
  DATALIST1 = Append[DATALIST1, {i, 5 * x2, 5 * y2}];
  x1 = x2;
  y1 = y2, {i, NN}];
Show[{
  Graphics3D[Line[{DATALIST1}]],
  ListPointPlot3D[DATALIST1, PlotStyle → Directive[PointSize[0.02], Black]]}, Axes → True]
```





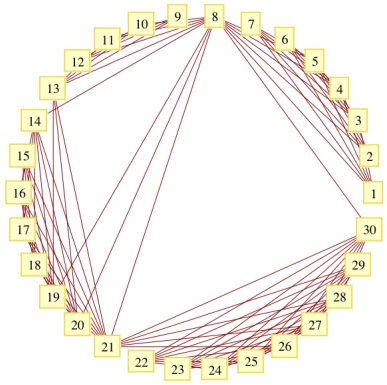


Figure 3.2.1: A data plot, a magnitude plot, and the associated visibility graph for the Burgers map, with 30 data points, generated by iterating the system of equations form a given set of initial values. The parameters for the Burgers map are chosen to be in its chaotic region (see the bifurcations in Section 1.5). One can easily identify the 8th, 21st, and 30th point as crucial, of highest degree and visibility. In the original data, interestingly, the 8th and 21st point correspond to rapid changes of behavior, a jump and change in direction.

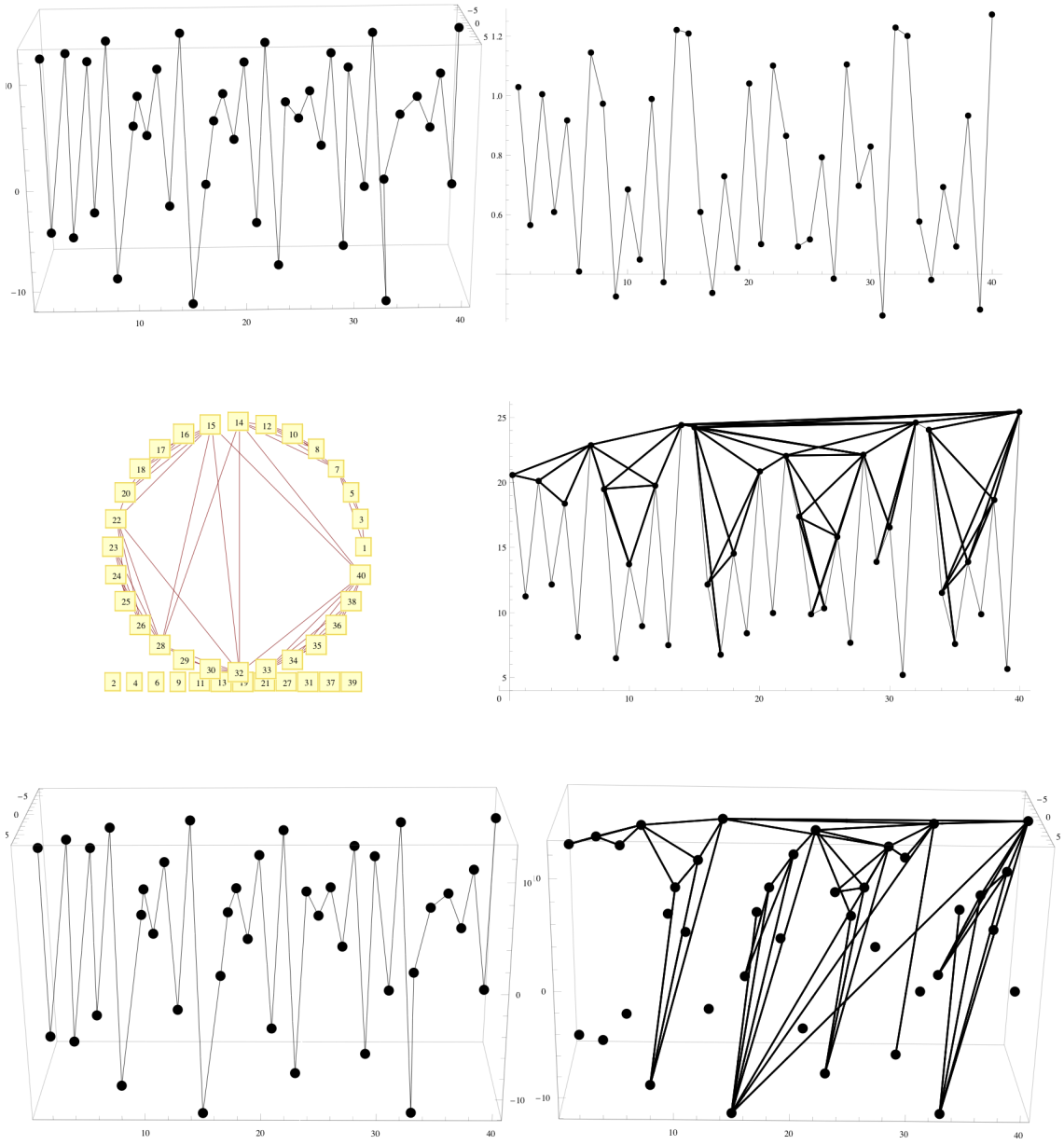


Figure 3.2.2: A data plot, a magnitude plot, and the associated visibility graph for the Hénon map, with 40 data points, generated by iterating the system of equations form a given set of initial values. The parameters for the Hénon map are chosen to be in its chaotic region (see the bifurcations in Section 1.5). The second row on the right shows in bold the visibility connections of the magnitude data. The bottom row shows the original data and its visibility, according to magnitude visibility, Note that we had to scale the vertical heigh of some of the plots in order to be able to display the plots next to each other and compare.

```
(*HENON MAP*)
DATALIST = {}; DATALIST1 = {};
x1 = 0.03; y1 = 0.03; NN = 40;
Do[x2 = y1 + 1 - 1.4 * x1 ^ 2; y2 = 0.3 * x1, {k, 1000}];
Do[x2 = y1 + 1 - 1.4 * x1 ^ 2; y2 = 0.3 * x1;
  DATALIST = Append[DATALIST, Sqrt[x2 ^ 2 + y2 ^ 2]];
  DATALIST1 = Append[DATALIST1, {i, 10 * x2, 20 * y2}];
  x1 = x2; y1 = y2, {i, NN}];
Show[{
  Graphics3D[Line[{DATALIST1}]],
  ListPointPlot3D[DATALIST1, PlotStyle → Directive[PointSize[0.02], Black]]},
 Axes → True]
```

In[1395]:=
```
Show[{
  ListPlot[DATALIST, Joined → True, PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[DATALIST, PlotStyle → {AbsolutePointSize[7], Black, AspectRatio → 0.7, Axes → True}]}]
```

In[1396]:=
```
T1NEW[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, i + 1, k - 1}]], {i, k}, {0, 0}];
T2NEW[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, k + 1, i - 1}]], {i, k}, {0, 0}];
```

In[1398]:=
```
TTT = Table[((i <= k) ⋀ T1NEW[{i, k, DATALIST}]) ⋁ ((k < i) ⋀ T2NEW[{i, k, DATALIST}]),
   {i, NN}, {k, NN}];
```

In[1430]:=
```
LIST = {};
Do[
  Do[m = TTT[[i, j]];
   If[m ≠ {0, 0},
    LIST = Append[LIST, {{TTT[[i, j, 1]], 20 * DATALIST[[TTT[[i, j, 1]]]]},
       {TTT[[i, j, 2]], 20 * DATALIST[[TTT[[i, j, 2]]]]}}]], {i, NN}], {j, NN}];
```

In[1432]:=
```
Show[{Graphics[{Thick, Table[Line[LIST[[i]]], {i, Length[LIST]}]}, Axes → True,
   AxesOrigin → {0.8, 4}],
  ListPlot[20 * DATALIST, Joined → True, PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[20 * DATALIST, PlotStyle → {AbsolutePointSize[7], Black, AspectRatio → 0.7, Axes → True}]}]
```

In[1426]:=
```
Show[{
  Graphics3D[{Thick, Table[Line[LIST[[i]]], {i, Length[LIST]}]}, Axes → True],
  ListPointPlot3D[DATALIST1, PlotStyle → Directive[PointSize[0.02], Black]]},
 Axes → True]
```

In[1416]:=
```
GraphPlot[Table[((i <= k) ⋀ T1[{i, k, DATALIST}]) ⋁ ((k < i) ⋀ T2[{i, k, DATALIST}]), {i, NN}, {k, NN}],
 VertexLabeling → True, Method → "CircularEmbedding"]
```

Figure 3.2.3: The Mathematica code that generated the figures above. A lot of the joined plots are built out of a plot for the points, for the segments, and for the visibility segments (the hardest ones to obtain using the T1NEW and T2NEW functions, as well as the two loops that generate the lists TTT and LIST.The algorithm to generate this required sophisticated programming and was developed in collaboration with my advisor, Dr. Gospodinov.

```
(*HENON MAP*)
DATALIST = {}; DATALISTx = {}; DATALISTy = {}; x1 = 0.03; y1 = 0.03; NN = 20;
Do[x2 = y1 + 1 - 1.4 * x1^2; y2 = 0.3 * x1, {k, 1000}];
Do[x2 = y1 + 1 - 1.4 * x1^2; y2 = 0.3 * x1;
  DATALIST = Append[DATALIST, {i, 5 * x2, 20 * y2}];
  DATALISTx = Append[DATALISTx, {i, 5 * x2, -10}];
  DATALISTy = Append[DATALISTy, {i, -10, 20 * y2}];
  x1 = x2; y1 = y2, {i, NN}];
Show[{
  Graphics3D[{Thick, Line[{DATALIST1}]}],
  ListPointPlot3D[DATALIST1, PlotStyle → Directive[PointSize[0.02], Black]],
  Graphics3D[{Dashed, Thin, Line[{DATALISTx}]}],
  ListPointPlot3D[DATALISTx, PlotStyle → Directive[PointSize[0.01], Black]],
  Graphics3D[{Dashed, Thin, Line[{DATALISTy}]}],
  ListPointPlot3D[DATALISTy, PlotStyle → Directive[PointSize[0.01], Black]]
 }, Axes → True]

In[1650]:= T1NEW[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, i + 1, k - 1}]], {i, k}, {0, 0}];
T2NEW[{i_, k_, LIST_}] := If[i ≠ k ⋀ Abs[i - k] ≠ 1 ⋀
    Apply[And, Table[TrueQ[LIST[[j]] < LIST[[i]] + (LIST[[k]] - LIST[[i]]) * (j - i) / (k - i)],
      {j, k + 1, i - 1}]], {i, k}, {0, 0}];

In[1676]:= Lx1 = Table[DATALISTx[[i, 2]], {i, NN}];
Lx2 = Table[{DATALISTx[[i, 1]], DATALISTx[[i, 2]]}, {i, NN}];
TTT = Table[((i <= k) ⋀ T1NEW[{i, k, Lx1}]) ⋁ ((k < i) ⋀ T2NEW[{i, k, Lx1}]), {i, NN}, {k, NN}];

In[1681]:= LISTx = {};
Do[
  Do[m = TTT[[i, j]];
   If[m ≠ {0, 0}, LISTx = Append[LISTx, {5 * Lx2[[TTT[[i, j, 1]]]], 5 * Lx2[[TTT[[i, j, 2]]]]}]],
   {i, NN}], {j, NN}];

In[1683]:= Show[{Graphics[{Thick, Table[Line[LISTx[[i]]], {i, Length[LISTx]}]}, Axes → True,
    AxesOrigin → {0.8, 4}],
  ListPlot[5 * Lx2, Joined → True, PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[5 * Lx2, PlotStyle → {AbsolutePointSize[7], Black, AspectRatio → 0.7, Axes → True}]}]

In[1692]:= Ly1 = Table[DATALISTy[[i, 3]], {i, NN}];
Ly2 = Table[{DATALISTy[[i, 1]], DATALISTy[[i, 3]]}, {i, NN}];
TTT = Table[((i <= k) ⋀ T1NEW[{i, k, Ly1}]) ⋁ ((k < i) ⋀ T2NEW[{i, k, Ly1}]), {i, NN}, {k, NN}];

In[1694]:= LISTy = {};
Do[
  Do[m = TTT[[i, j]];
   If[m ≠ {0, 0}, LISTy = Append[LISTy, {20 * Ly2[[TTT[[i, j, 1]]]], 20 * Ly2[[TTT[[i, j, 2]]]]}]],
   {i, NN}], {j, NN}];

In[1696]:= Show[{Graphics[{Thick, Table[Line[LISTy[[i]]], {i, Length[LISTy]}]}, Axes → True,
    AxesOrigin → {0.8, 4}],
  ListPlot[20 * Ly2, Joined → True, PlotStyle → {Black, AspectRatio → 0.7, Axes → True}],
  ListPlot[20 * Ly2, PlotStyle → {AbsolutePointSize[7], Black, AspectRatio → 0.7, Axes → True}]}]
```

Figure 3.2.4: The following several pages explore higher-dimensional visibility and partial visibility. Recall that this essentially reduced multi-dimensional visibility to standard visibility in the projections. If the data points are visible in all their projections, then they are **HD-visible**, if they are visible in some of their projections, they are partially visible. In our case, we have 2-dimensional data from the Hénon map. First with 20, and then with 50 points. The algorithm to generate this required sophisticated programming and was developed in collaboration with my advisor, Dr. Gospodinov.

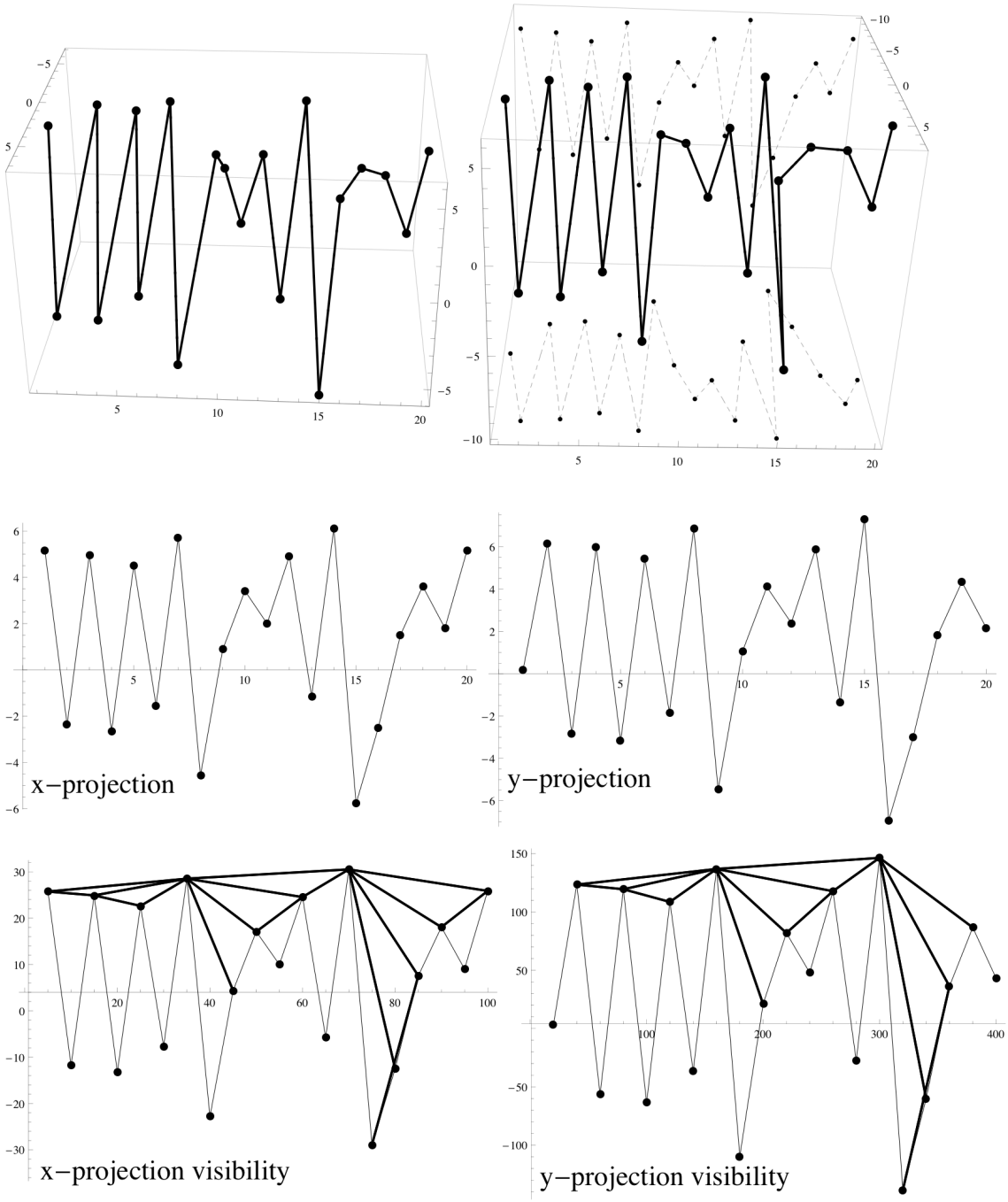


Figure 3.2.5: We have the original 20 data points on the top left, next to the data points shown with their projections. On the next row, the two projections are shown separately as 1-dimensional data sets. Lastly, standard visibility is shown in bold lines for both sets. The sets seem very similar, but are slightly shifted, and one can easily distinguish them by their last few elements.

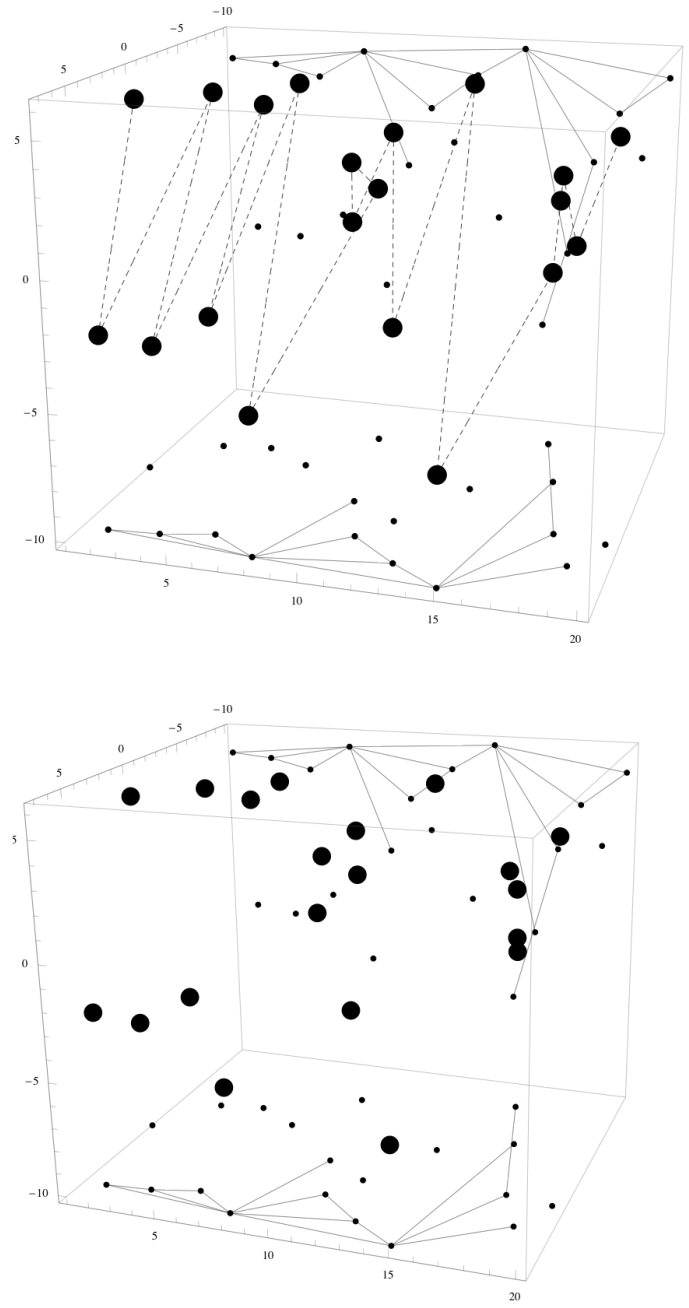


Figure 3.2.6: We show the original 20 data points with their projections on the top. The bottom shows the HD-visibility, along with the projections. Notice that NONE of the points are visible! There is no mistake (although it took us a while to verify this). None of these 20 points are simultaneously visible in their x- and y-projection (see Figure 3.2.5).

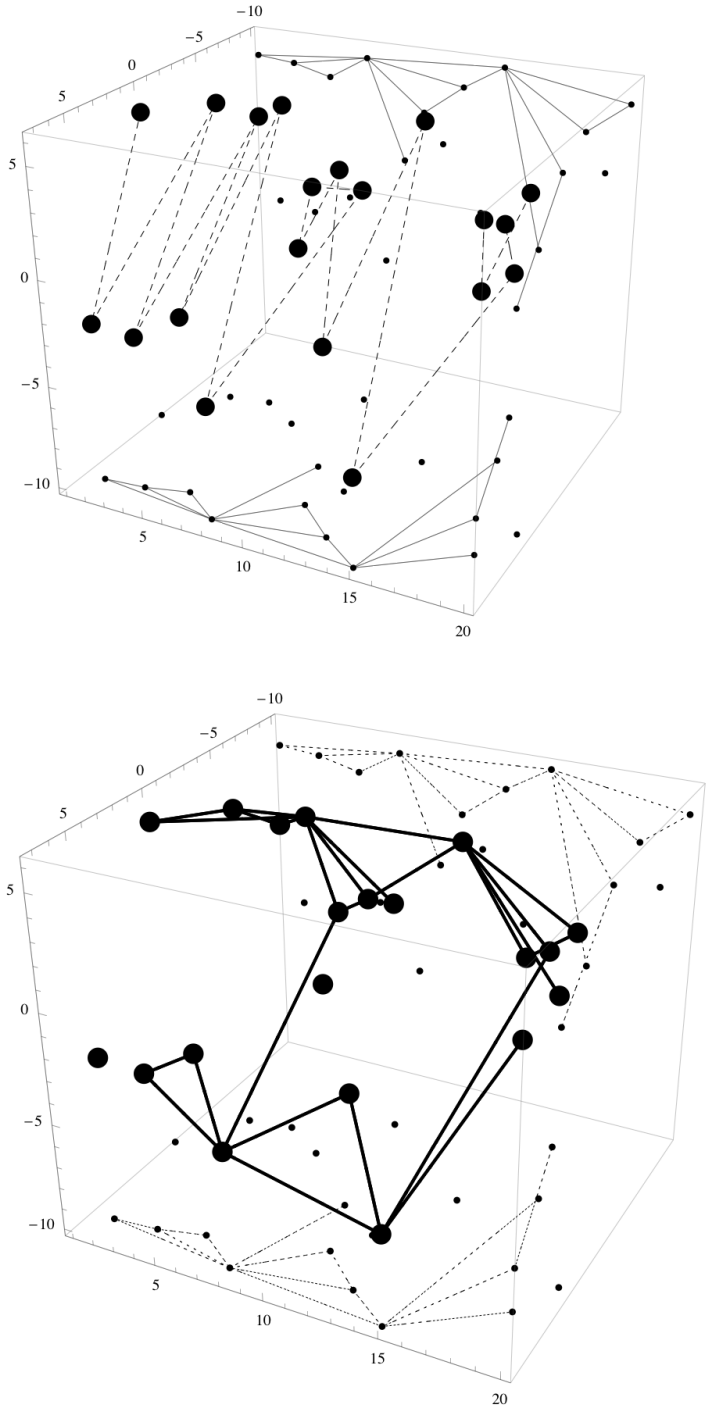


Figure 3.2.7: We show the original 20 data points with their projections on the top. The bottom shows the PARTIAL visibility (that is, original data points are visible if they are visible in at least one of their projections), along with the projections. Notice that many of the points are visible. Partial visibility seems to present a rich visibility structure on the data (see Figure 3.2.5).

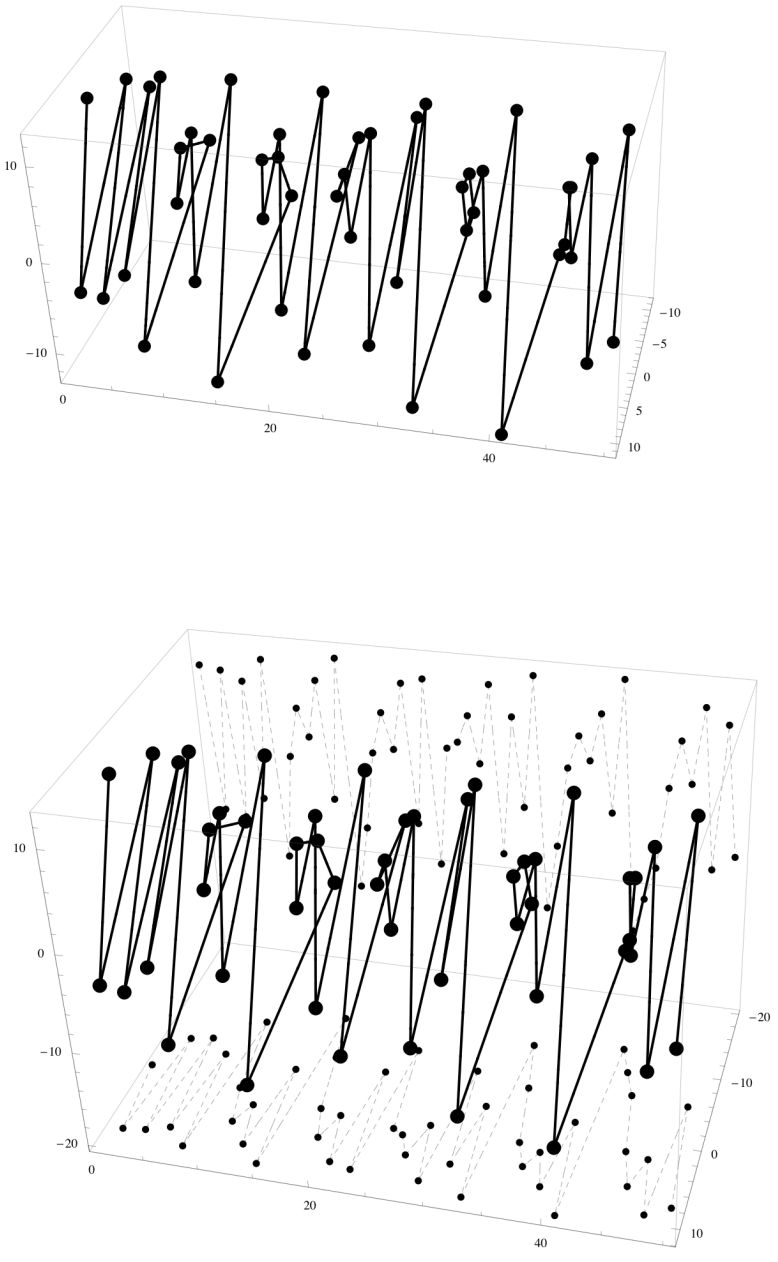


Figure 3.2.8: We show a data set of 50 points for the Hénon system (After a 1000-point preliminary iteration loop to land us on the attractor region). THe top plot shows the original data, and the bottom plot shows the original data along with the projections data. This study was generated by the same Mathematica routine (see Figure 3.2.4).

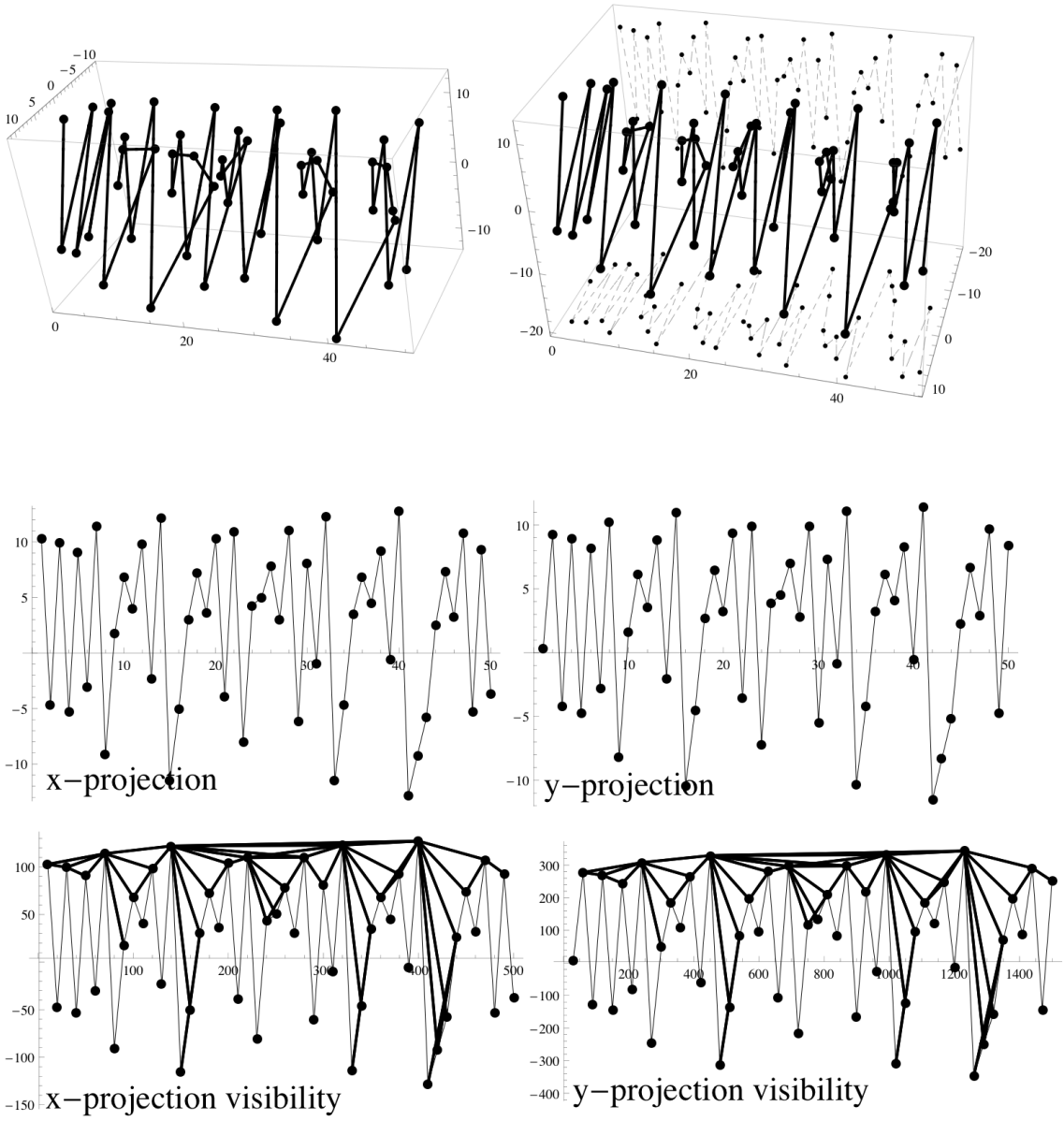


Figure 3.2.9: We have the original 50 data points on the top left, next to the data points shown with their projections. On the next row, the two projections are shown separately as 1-dimensional data sets. Lastly, standard visibility is shown in bold lines for both sets. The sets seem very similar, but are slightly shifted, and one can easily distinguish them by their last few elements.

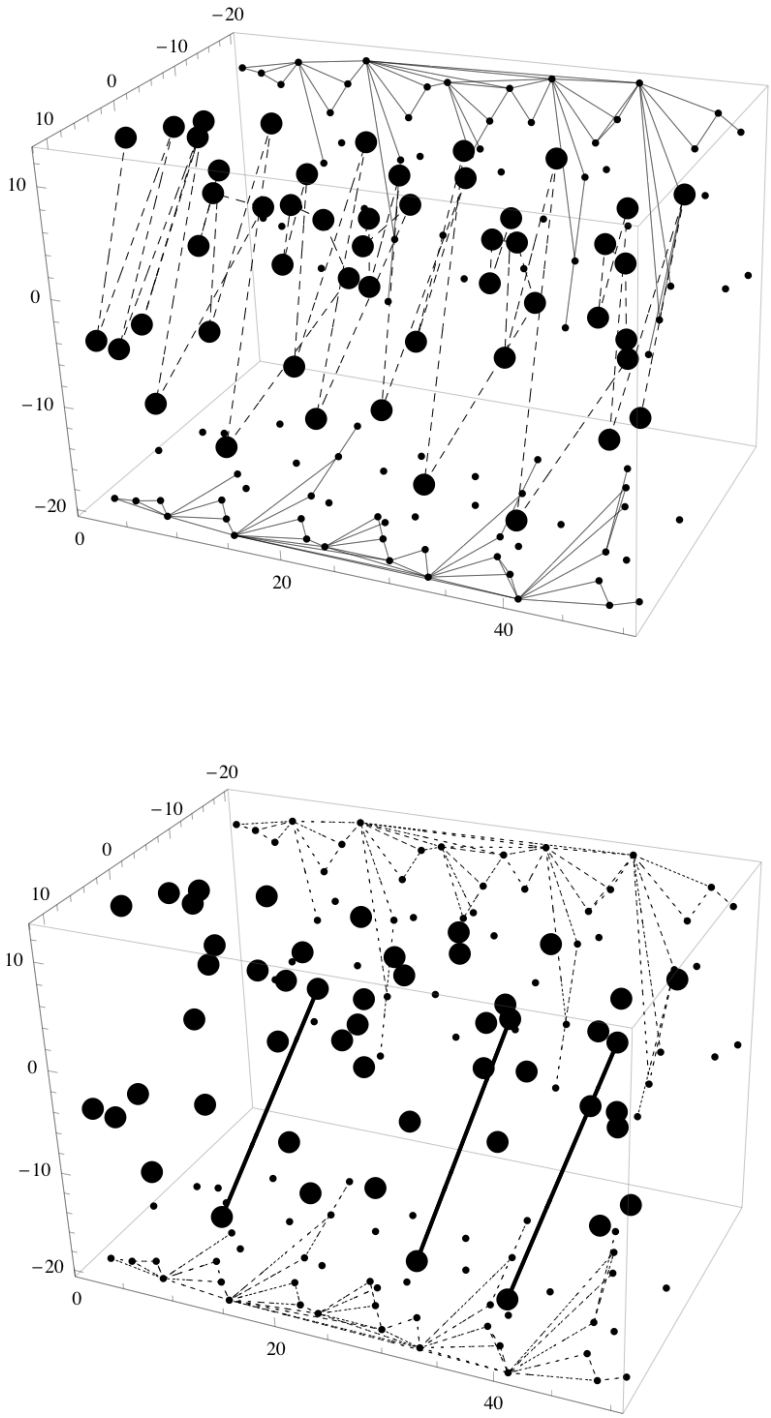


Figure 3.2.10: We show the original 50 data points with their projections on the top. The bottom shows the HD-visibility, along with the projections. Notice that ONLY A FEW of the points are visible! So HD-visibility seems to be very very restrictive, even in the 2-dimensional case.

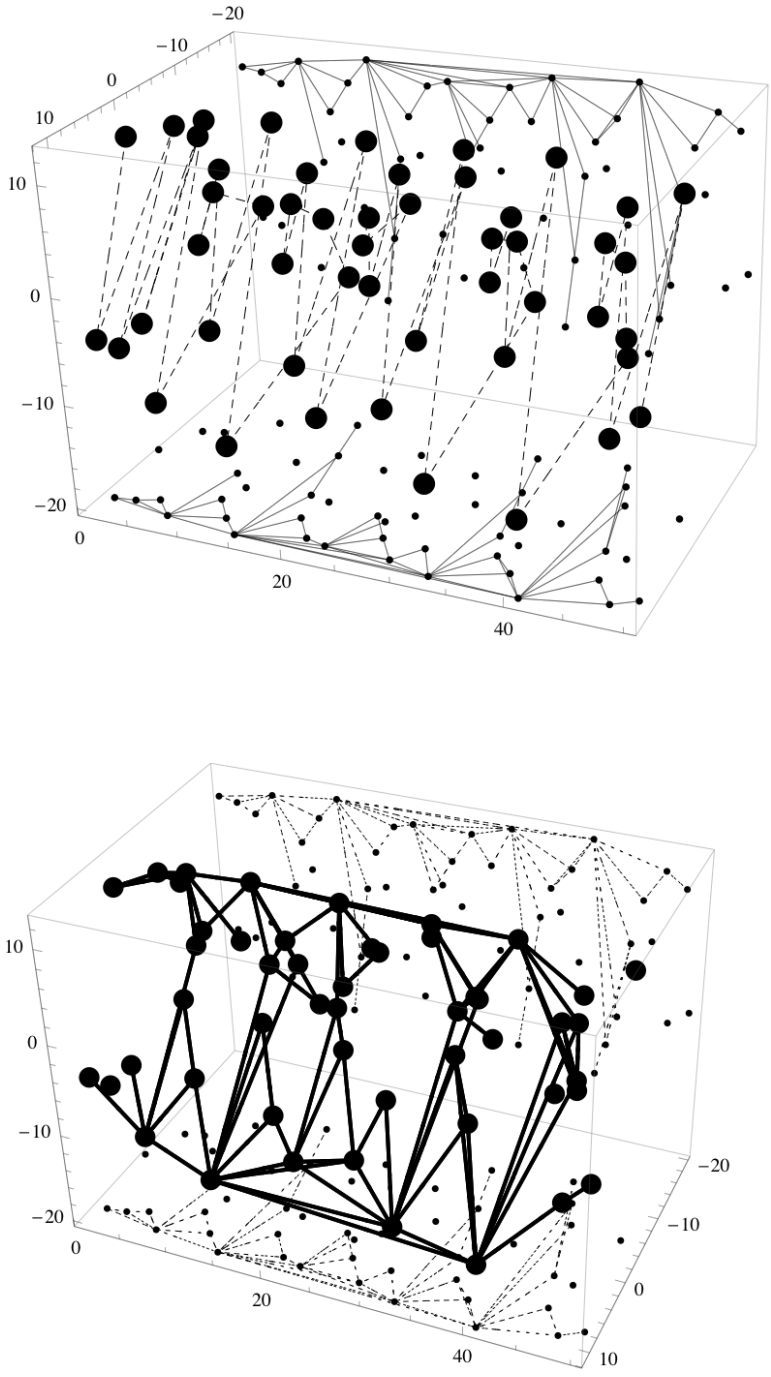


Figure 3.2.11: We show the original 50 data points with their projections on the top. The bottom shows the PARTIAL visibility (that is, original data points are visible if they are visible in at least one of their projections), along with the projections. Partial visibility seems to present a rich visibility structure on the data.

3.2.2 2-Dimensional Poincaré Sections

We can use 2-dimensional Poincaré sections to extend our study to even higher-dimensional time series. These can be fed in any of the 2-dimensional routines we have and studied. This is part of our future projects.

3.2.3 3-Dimensional Time Series Examples

Understanding how to deal with graphs of 3-dimensional time series can help us apply our higher-dimensional visibility ideas to the time series from our list of examples. Visualizing the visibility data may be difficult since the time dimension would require a 4-dimensional space, but we are working on developing techniques for visualizing 3-dimensional graphs.

4
Conclusions and Future Work

This has been an exciting project that has given rise to many new directions of work. Visibility of data seems to provide a rich new field of study and a variety of tools for the analysis of 1-dimensional, and now also higher-dimensional time data.

Our work continues with higher-dimensional examples, and some statistical calculations.

Bibliography

[1] J. A. Bondy and U. S. R. Murty, *Graph Theory With Applications*, Elsevier Science Publishing Co., Inc., New York, NY, 1976.

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

[3] ———, *Numerical Methods for Ordinary Differential Equations*, Wiley, Chichester and New York, New York, NY, 2003.

[4] R. Diestrel, *Graph Theory*, Springer-Verlag New York, New York, NY, 2000.

[5] B. Luque, L. Lacasa, F. Ballesteros, and J. Luque, *Horizontal visibility graphs: Exact results for random time series*, Phys. Rev. **E 80** (2009).

[6] B. Luque, L. Lacasa, F. Ballesteros, J. Luque, and J.C. Nuño, *From time series to complex networks: The visibility graph*, Proc. Natl. Acad. Sci. U.S.A. **105** (2008), 4972–4975.

[7] B. Luque, L. Lacasa, F. Ballesteros, and A. Robledo, *Horizontal visibility graphs: Exact results for random time series*, Phys. Rev. **E 80** (2009).

[8] ———, *Analytical properties of horizontal visibility graphs in the Feigenbaum scenario*, Chaos **22** (2012), 013109.

[9] F. C. Moon, *Chaotic and Fractal Dynamics: An Introduction for Applied Scientists and Engineers*, John Wiley & Sons, Inc., New York, NY, 2004.

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

[11] H. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer, New York, NY, 2004.

[12] J. C. Sprott, *Some simple chaotic jerk functions*, Am. J. Phys. **65, No. 3** (1997), 537–543.

[13] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, Perseus Books Publishing, L.L.C., New York, NY, 1994.

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