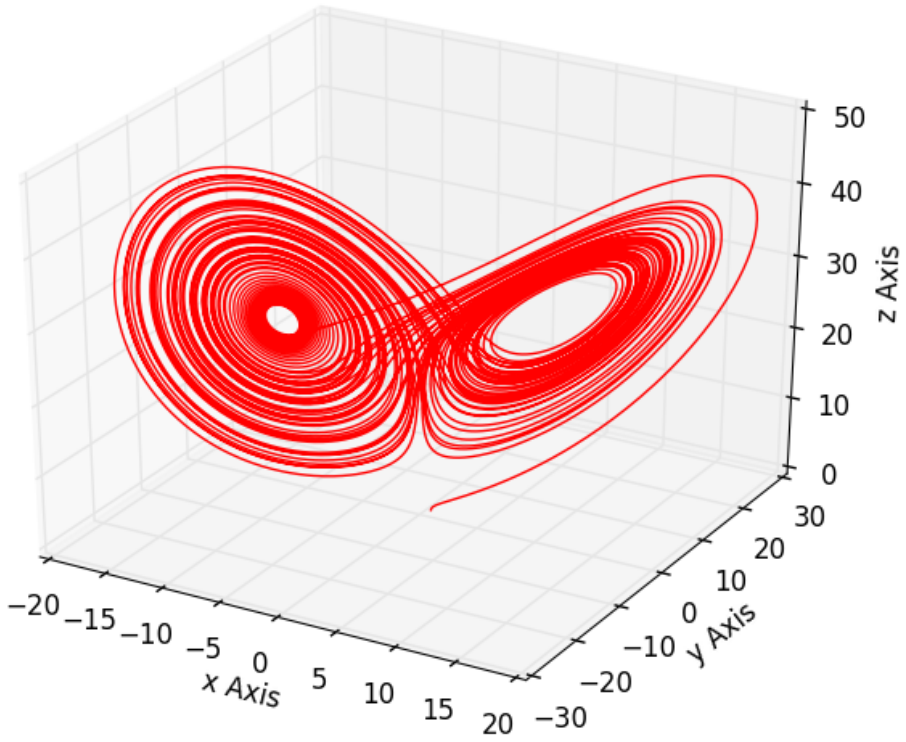
University of Leeds

MATH3001: Project in Mathematics

Computational Modelling of Nonlinear Dynamics: Chaos in the Lorenz Equations

Richard Cotton (201008138)



Supervised by
Dr. Stephen Griffiths

UNIVERSITY OF LEEDS

School of Mathematics

Declaration of Academic Integrity
for Individual Pieces of Work

I am aware that the University defines plagiarism as presenting someone else's work as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance.

I promise that in the attached submission I have not presented anyone else's work as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance.

I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.

I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the School's policy on mitigation and procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Student Signature _____ Date _____

Student Name _____ Student Number _____

Please note.
When you become a registered student of the University at first and any subsequent registration you sign the following authorisation and declaration:
"I confirm that the information I have given on this form is correct. I agree to observe the provisions of the University's Charter, Statutes, Ordinances, Regulations and Codes of Practice for the time being in force. I know that it is my responsibility to be aware of their contents and that I can read them on the University web site. I acknowledge my obligation under the Payment of Fees Section in the Handbook to pay all charges to the University on demand.
I agree to the University processing my personal data (including sensitive data) in accordance with its Code of Practice on Data Protection http://www.leeds.ac.uk/dpa . I consent to the University making available to third parties (who may be based outside the European Economic Area) any of my work in any form for standards and monitoring purposes including verifying the absence of plagiarised material. I agree that third parties may retain copies of my work for these purposes on the understanding that the third party will not disclose my identity.'"

Contents

1 Introduction to the Lorenz Equations

1.1 Edward Lorenz

The Lorenz Equations were named after American meteorologist and mathematician Edward Norton Lorenz. He received degrees in mathematics from Dartmouth College and Harvard University and worked in weather forecasting with the US Air Army Corps. Post World War II, he attended MIT where he received a masters and a doctorate in meteorology in 1943 and 1948 respectively.

In early 1960s, Lorenz discovered the weather follows a phenomenon called "sensitive dependence on initial conditions". He developed his now famous equations as a model for thermal convection flows within the Earth's atmosphere. Lorenz explained his model to the public as the "butterfly effect"; a butterfly flapping its wings on one side of the planet having unpredictable and drastic effects on the other side.

For his work published in a paper called "Deterministic Nonperiodic Flow" (1963), Lorenz was awarded the 1983 Crafoord Prize and the 1991 Kyoto Prize. [1]

1.2 Introduction

In this paper I shall discuss the Lorenz Equations given by

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - y - xz \\ \dot{z} = xy - bz. \end{cases} \tag{1}$$

I will take an in-depth look into the intrinsic properties belonging to it and some analysis of numerical solutions for said equations. The research carried out in writing this paper includes a number of topics within mathematics. I shall investigate the intrinsic properties belonging to the set of equations including a thorough exploration of the fixed points and their stability. A number of other properties including boundedness and dissipativity will be discussed.

The Lorenz Equations are of great importance within the mathematical community due to the inherent chaos they behold. For example, with small changes of initial conditions, the final outcomes can behave wildly differently. The equations also exhibit the property of strange attractiveness, where a chaotic system is bounded and has a clear pattern but is not a periodic oscillation.

We will move on to an analysis of two numerical methods used to solve the system of equations. The system will be solved exactly by removing non-linearities and instantiating the parameter values. The solutions generated by the time stepping methods can be compared to the exact solution to find the error and thus, the accuracy for the numerical method. The analysis will be completed by using advanced techniques such as 3-dimensional graphing, Poincaré Sections and the Lorenz Map to further investigate the numerical analysis.

I shall assume that the reader has no prior knowledge of dynamical systems but is fluent with differentiation, algebra and basic vector calculus. Thus, this paper is aimed at a readership of latter undergraduate level mathematics. This paper will see significant use of the programming language Python; however, this will not be of hindrance for those not familiar as the paper focuses on the mathematics side of analysis, rather than the programming aspect.

2 Fixed Points

2.1 Fixed Points

We begin our analysis by finding the fixed points of the 3 equation system. This is useful as it allows us to find points at which the value of each direction remains constant and thus, the trajectory remains at the point. The point (0,0,0) is a clear fixed point, regardless of the values of the parameters. Looking deeper, we find from the $\dot{x}$ equation that $x = y$ is a fixed point of the x direction. Substituting this into equations $\dot{y}$ and $\dot{z}$, we obtain

$$\begin{cases} \dot{y} = x(r - 1 - z) = 0 \\ \dot{z} = x^2 - bz = 0. \end{cases} \tag{2}$$

Then from $\dot{y}$, $z = r - 1$, which can be substituted into equation $\dot{z}$ to find $x^2 - b(r - 1) = 0$. Thus $x = \pm\sqrt{b(r-1)}$ which is equivalent to y.

Therefore, we have two cases depending on whether $r > 1$ or $r < 1$. If $r < 1$, there is only one fixed point, the origin. This corresponds to the state of no convection.

However in the case that $r > 1$, we have three fixed points as follows:

$$\begin{cases} x_- = (-\sqrt{b(r-1)}, -\sqrt{b(r-1)}, r-1) \\ x_+ = (\sqrt{b(r-1)}, \sqrt{b(r-1)}, r-1) \\ x_0 = (0, 0, 0). \end{cases}$$

2.2 Stability of the Fixed Points

We are interested in determining the stability of the fixed points inherent to the system as this allows us to predict how the trajectories move around 3-dimensional space. This will be integral to later parts of the report as we may check whether our python code is working correctly when calculating and plotting our results. We wish to linearise the system around the fixed points as the behaviour for the original system is very similar to that of the behaviour of the linearised system close to the fixed points.

Consider the general system given by $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ and a fixed point x_*. We may take a Taylor series around x_*, allowing us to linearise the system.

$$\mathbf{F}(\mathbf{x}) \approx \mathbf{F}(\mathbf{x}_*) + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_*} (\mathbf{x} - \mathbf{x}_*) + \dots$$

$$\text{where } \frac{\partial \mathbf{F}}{\partial \mathbf{x}} = J(\mathbf{x}) = \begin{pmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial y} & \frac{\partial \dot{x}}{\partial z} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial y} & \frac{\partial \dot{y}}{\partial z} \\ \frac{\partial \dot{z}}{\partial x} & \frac{\partial \dot{z}}{\partial y} & \frac{\partial \dot{z}}{\partial z} \end{pmatrix}.$$

After computing the Jacobian matrix, we may calculate the eigenvalues of the matrix evaluated at the fixed point x_*. This is done by computing $\mid J(\mathbf{x}_*) - \lambda I \mid = 0$.

The equations can be written in matrix form given by $\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$ with general solution $\mathbf{x}(t) = e^{tA}\mathbf{x}_0$ where $\mathbf{x}_0$ is the initial value.

Taking a general eigenvalue $\lambda = a + bi$ in one of these terms, we find $e^{a+bi} = e^a(\cos(b) + i\sin(b))$. Thus the real part of the eigenvalue determines the stability while the complex part affects the oscillatory behaviour.

In the case of 3 dimensions, we have

$$\mathbf{x}(t) = C_1 e^{\lambda_1 t}\mathbf{v}_1 + C_2 e^{\lambda_2 t}\mathbf{v}_2 + C_3 e^{\lambda_3 t}\mathbf{v}_3 \tag{3}$$

where λ_i are the eigenvalues and $\mathbf{v}_i$ are the eigenvectors of the Jacobian matrix.

We shall only consider 2 cases for varying eigenvalues. If all eigenvalues have negative real part, then the fixed point is stable. This is due to all terms in the sum above reducing to 0 exponentially quickly. If at least one eigenvalue has positive real part, then the fixed point is unstable, due to the value of this term approaching ∞ while all others tend to 0.

We shall now study the stability of the fixed points by constructing the Jacobian matrix and computing the eigenvalues of said system. We calculate the general Jacobian for the system

$$J = \begin{pmatrix} -\sigma & \sigma & 0 \\ r - z & -1 & -x \\ y & x & -b \end{pmatrix}.$$

Starting our analysis with the fixed point at the origin

$$J_{(0,0,0)} = \begin{pmatrix} -\sigma & \sigma & 0 \\ r & -1 & 0 \\ 0 & 0 & -b \end{pmatrix}.$$

Thusly, we obtain the characteristic equation:

$$(\lambda + b)(\lambda^2 + \lambda(\sigma + 1) + \sigma(1 - r)) = 0.$$

Therefore we find the eigenvalues at the origin are $\lambda_0 = -b$ and $\lambda_\pm = \frac{-(\sigma+1)\pm\sqrt{(\sigma-1)^2+4r\sigma}}{2}$.

Using this information, we find that in the case $r < 1$, all three of the eigenvalues take negative real values. Therefore, the fixed point (0,0,0) is stable, thus unaffected by small changes in position.

If $r > 1$, we find 2 negative eigenvalues and one positive. This corresponds to the case of an unstable fixed point.

Now we shall analyse the fixed points x_+ and x_-. As noted previously, the system is symmetric in (x, y). Therefore, we shall only consider x_+. This time, we will fix the values of $\sigma = 10$ and $b = \frac{8}{3}$ and use python to evaluate the eigenvalues over differing values of the parameter r. The main focus will be on the real part of the eigenvalues, as this determines whether the fixed point is stable or unstable.

The Jacobian matrix at the fixed point x_+ is:

$$J_+ = \begin{pmatrix} -\sigma & \sigma & 0 \\ 1 & -1 & -\sqrt{b(r-1)} \\ \sqrt{b(r-1)} & \sqrt{b(r-1)} & -b \end{pmatrix}.$$

By using the python code found in Appendix A, we may evaluate the eigenvalues of the matrix, upon varying r. Our focus will be on the real part of the eigenvalue for the reason highlighted above.

r	Fixed Points	λ	Stability
$0 < r < 1$	x_0	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
	x_+	-	Non-Existence
	x_-	-	Non-Existence
$r = 1$	x_0	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
	x_+	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
	x_-	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
$1 < r < r_h \approx 24.74$	x_0	$\lambda_1, \lambda_2 < 0, \lambda_3 > 0$	Unstable
	x_+	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
	x_-	$\lambda_1, \lambda_2, \lambda_3 < 0$	Stable
$r > r_h$	x_0	$\lambda_1 > 0, \lambda_2, \lambda_3 < 0$	Unstable
	x_+	$\lambda_1 < 0, \lambda_2, \lambda_3 > 0$	Unstable
	x_-	$\lambda_1 < 0, \lambda_2, \lambda_3 > 0$	Unstable

Table 1: Eigenvalues for varying r

For the values $r < 1.346$, we notice that all eigenvalues of the matrix are entirely real. For values $r > 1.346$, we have one real eigenvalue and two complex conjugate roots.

Our analysis agrees with those found in a wide range of papers on the subject - for example, in Sparrow. [2]

As seen in table 1, at the point $r = 1$, the fixed point at the origin undergoes a change in stability, from attractiveness to repelling. The two fixed points x_+ and x_- come into existence. Due to the transition from one to three fixed points, a pitchfork bifurcation occurs. This bifurcation is of the supercritical type as the fixed point x_0 becomes unstable while the fixed points x_+ and x_- become stable.

The value of r_h can be generalised for all parameter values of r, b and σ. From the Jacobian matrix at the fixed point x_+, we get the characteristic polynomial

$$\lambda^3 + (\sigma + b + 1)\lambda^2 + b(\sigma + r)\lambda + 2b\sigma(r - 1).$$

The instability points occur when the largest eigenvalue has zero real part. This is because as the value of r increases crossing over r_h, the real part of the eigenvalue increases and switches sign from negative to positive. Thus, the fixed points change stability from stable to unstable.

Consider a general eigenvalue of the form $\alpha + i\omega$. With zero real part, we get the eigenvalue $\lambda = i\omega$. Substituting into the characteristic equation

$$(i\omega)^3 + (\sigma + b + 1)(i\omega)^2 + b(\sigma + r)(i\omega) + 2b\sigma(r - 1) = 0.$$

After some simplification

$$-\omega^2(\sigma + b + 1) + 2\sigma b(r - 1) + i(-\omega^3 + \omega b(\sigma + r)) = 0.$$

Thus we require both real and imaginary parts to be zero. We may assume $\omega \neq 0$ as otherwise we get the trivial solution $\lambda = 0$. Thus,

$$\omega^2 = \tfrac{2\sigma b(r-1)}{\sigma+b+1} \text{ and } \omega^2 = b(\sigma + r).$$

These must be equal so

$$\tfrac{2\sigma b(r-1)}{\sigma+b+1} = b(\sigma + r) \Rightarrow r_h = \tfrac{\sigma(\sigma+b+3)}{\sigma-b-1}.$$ [3]

2.3 Global Behaviour

We may use Python to show how the trajectories of the Lorenz Equations are affected upon varying the parameter r. In this section, we will use two sets of initial conditions: $(1, 0, 0)$ shown in red and $(0, -1, r-1)$ shown in blue. Both trajectories will be plotted on the same axes to allow the dynamics to be seen clearly.

The first property of interest occurs in the case of $r = 5$ and $r = 10$. In these cases, the trajectories spiral toward the fixed points x_- and x_+ as discussed in Section 3.1.

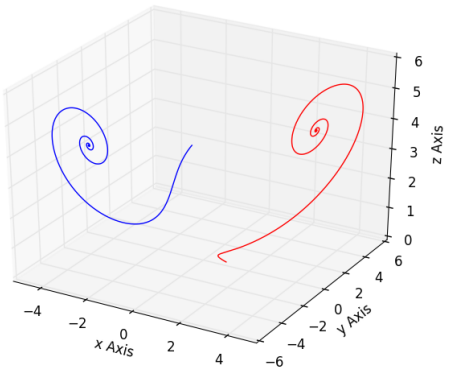


Figure 1: Plots for $r = 5$

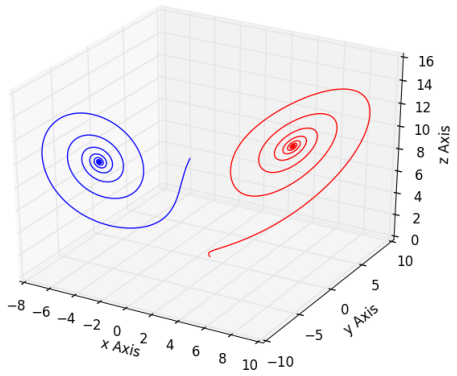


Figure 2: Plots for $r = 10$

These spirals are clearly stable as the trajectories converge to the fixed points.

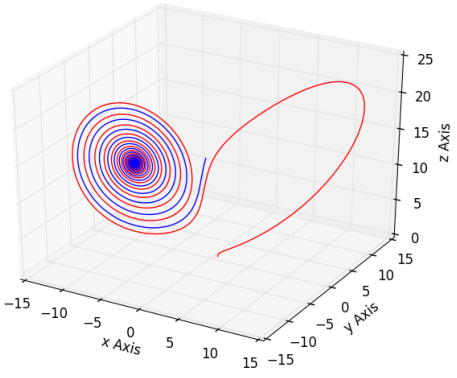


Figure 3: Plots for $r = 15$

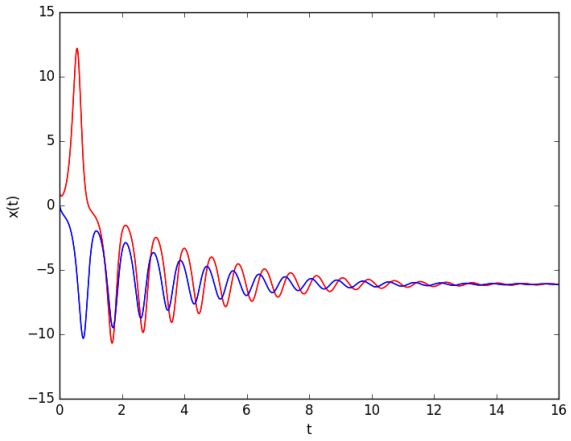


Figure 4: Plot of x versus t

As we increase r to 15, we see the red trajectory begins its first cycle around the fixed point x_+. Before it can complete its cycle, it is attracted towards the fixed point x_- in a spiral.

The trajectory plotted in blue is immediately attracted toward the fixed point x_- at all points in time. As seen in Figure 4, the value of the x-coordinates diverge away from each other to a distance of over 20 at around $t = 1$. After this initial deviation, the two trajectories approach one another. We note the blue trajectory looks to be a dampened, lagged version of its red counterpart. After around $t = 14$, the solutions seem to converge, with a final x value of around -6.

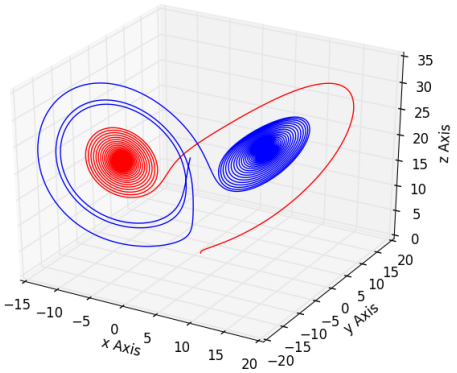


Figure 5: Plots for $r = 20$

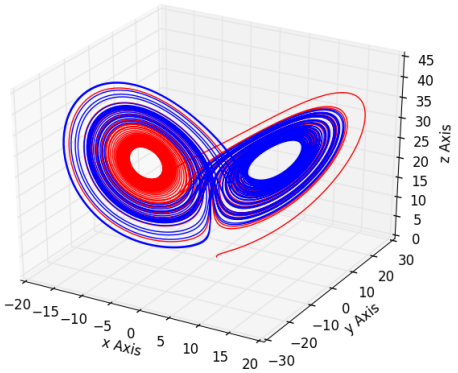


Figure 6: Plots for $r = 25$

For the value of $r = 20$, we see the red trajectory again circles the fixed point x_+ before spiralling to convergence to the fixed point x_-. However, for the blue trajectory, the beginnings of some chaotic behaviour emerges. The trajectory begins its path encircling away from x_- before converging rapidly toward the fixed point x_+. As we increase to $r = 25$, the chaotic pattern becomes clearer and we can see the "Lorenz butterfly" pattern much more clearly. The trajectories begin by converging toward the fixed points, completing a number of rotations spiralling away from them before switching over to the other fixed point.

Between $r = 20$ and $r = 25$, the inequality sign changes from $r < r_h$ to $r > r_h$. As noted in section 2.2, the fixed points x_+ and x_- change from stable to unstable across this region. This can be seen clearly from figures 5 and 6. For values above $r_h = 24.74$ the trajectories lie on the so called "strange attractor". This will be discussed in detail in section 4.

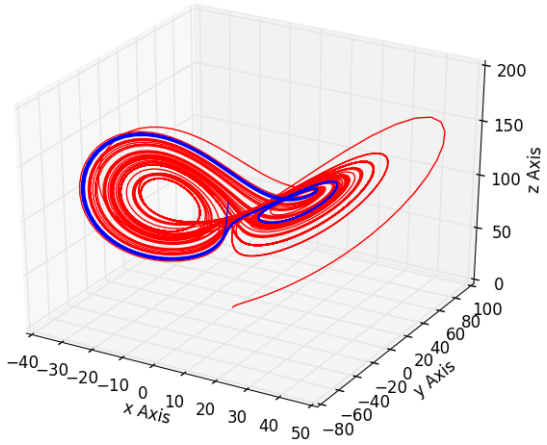


Figure 7: Plots for $r = 100$

Increasing the value of r to 100, we find a trajectory which no longer shows a chaotic solution. This blue solution in figure 7 shows an almost exact periodic cycle upon which the trajectory follows. Notice that the red trajectory also follows a similar, yet less strict cycle which is not upon the Lorenz attractor.

3 Properties of the Lorenz Equations

We return to the set of equations and take a deeper look at its properties. In this section, we shall assume that the parameters σ, r and b take positive real values.

3.1 Symmetry

A basic property of the system of equations is that they are invariant under the transformation of $(x, y) \to (-x, -y)$. Therefore, if there exists a solution of the form (x, y, z) this implies there is another solution of the form $(-x, -y, z)$.

Proof: We define a map $(x, y, z) \to (X, Y, Z) = (-x, -y, z)$. Then:

$$\begin{cases} \dot{X} = -\dot{x} = -\sigma(y - x) = \sigma(x - y) = \sigma(Y - X) \\ \dot{Y} = -\dot{y} = -rx + y + xz = rX - Y - XZ \\ \dot{Z} = \dot{z} = xy - bz = XY - bZ. \end{cases} \tag{4}$$

This property will be useful in our analysis of the stability of fixed points.

3.2 Dissipative

The Lorenz equations have the property of being dissipative; that is, volumes contract when moving along the flow of the system.

Proof:

We begin by defining $\mathbf{n}$ as the outward normal for the volume in phase space.
With a small increment in time V, we get

$$V(t + \delta t) = V(t) + \int_S (\mathbf{F} \cdot \mathbf{n} dt) dA.$$

Thus

$$\frac{V(t+\delta t) - V(t)}{dt} = \int_S (\mathbf{F} \cdot \mathbf{n}) dA.$$

Therefore, by using the Divergence Theorem and the definition of the derivative

$$\dot{V} = \int_V \nabla \cdot \mathbf{F} \, \mathrm{d}V.$$

Let $\mathbf{F} = (\sigma(y - x), rx - y - xz, xy - bz)$. Then,

$$\begin{aligned} \nabla \cdot \mathbf{F} &= \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \cdot (\sigma(y - x), rx - y - xz, xy - bz) \\ &= -\sigma - 1 - b \\ &= -(\sigma + b + 1). \end{aligned} \tag{5}$$

Thus $\nabla \cdot \mathbf{F} < 0$ for all values of $\sigma > 0$ and $b > 0$.

Using the above integral, we find

$$\dot{V} = -(\sigma + b + 1)V.$$

Solving this differential equation, we get the solution

$$V(t) = V(0)e^{-(\sigma+b+1)t}.$$

Therefore, trajectories reduce to zero volume with an exponential rate. [4]

3.3 Bounded Solutions

First, we define the notion of a Lyapunov function.

Definition: A function $V(\mathbf{x})$ is called a Lyapunov function if it satisfies the following 3 conditions:

1. $V(\mathbf{x}_*) = 0$
2. $V(\mathbf{x}) > 0$ for all $\mathbf{x} \in \bar{R}, \mathbf{x} \neq \mathbf{x}_*$ (V is positive definite)
3. $\dot{V} < 0$ for all $\mathbf{x} \in \bar{R}, \mathbf{x} \neq \mathbf{x}_*$ ($\dot{V}$ is negative definite)

where $\bar{R}$ is some neighbourhood of the point. [6]

We will now show that the solutions of the Lorenz system are bounded. We shall investigate this property in two cases, depending on whether $r < 1$ or $r > 1$. This is motivated by Strogatz [4] and exercise 9.2.2.

In the former, consider the Lyapunov function $V = \frac{x^2}{\sigma} + y^2 + z^2$. Differentiating with respect to time,

$$\begin{aligned}\frac{1}{2}\dot{V} &= \frac{x}{\sigma}\dot{x} + y\dot{y} + z\dot{z} \\ &= x(y-x) + y(rx - y - xz) + z(xy - bz) \\ &= (r+1)xy - x^2 - y^2 - bz^2 \\ &= -\left(x - \frac{r+1}{2}y\right)^2 - \left(1 - \left(\frac{r+1}{2}\right)^2\right)y^2 - bz^2.\end{aligned} \tag{6}$$

As $0 < r < 1, 0 < \frac{r+1}{2} < 1$ so $\dot{V}$ is negative definite. Thus the chosen Lyapunov function satisfies the conditions outlined above. Due to the existence of such a function, we note that V($\mathbf{x}$) decreases monotonically along the trajectories. This shows that the origin is a global attractor for all trajectories in the case $r < 1$.

In the the latter case of $r > 1$, we may consider the Lyapunov function $V = rx^2 + \sigma y^2 + \sigma(z-2r)^2$. Upon taking a time derivative:

$$\begin{aligned}\frac{\dot{V}}{2\sigma b r^2} &= -\frac{y^2 + bz^2 + r(x^2 - 2bz)}{br^2} \\ &= -\frac{x^2}{br} - \frac{y^2}{br^2} - \frac{(z-r)^2}{r^2} + 1\end{aligned} \tag{7}$$

The Lyapunov function has level surfaces on $V = V_0$ which are ellipsoids. The above equation is constant on the level surface. Taking the surface of $V = 0$ we get the expression

$$\frac{x^2}{br} + \frac{y^2}{br^2} + \frac{(z-r)^2}{r^2} = 1. \tag{8}$$

We may now choose $V = V_0$ such that the level surface lies wholly outside the ellipsoid given above. Any trajectories inside in the level surface will remain inside and any starting outside will cross into the level set. Thus, the trajectories are always bounded, regardless of the value of r.

4 Chaos and the Strange Attractor

4.1 Chaos

Now we are familiar with the general properties with the Lorenz equations, we are able to examine in depth the attractive and chaotic nature of the system. Although there is not a universally accepted definition, we use a working definition.

Defintion:
"Chaos is aperiodic long-term behaviour in a deterministic system that exhibits sensitive dependence on initial conditions." [4]

We break this down as follows:

- "Aperiodic long-term behaviour" requires the system in question to have trajectory solutions which do not follow any periodic paths or tend to any fixed points.
- "Deterministic system" requires the parameters used in the system to be fixed while the system is running. For example, the parameters should not change when a trajectory is being followed.
- "Sensitivity to initial conditions" means trajectories which start initially within a small range should diverge away from each other as the system progresses.

The Lorenz system exhibits the property of chaos for the reasons below.

The system has aperiodic solutions. This can be seen by taking the standard parameter values of $\sigma = 10, b = \frac{8}{3}$ and $r = 28$ as seen in figure 8.

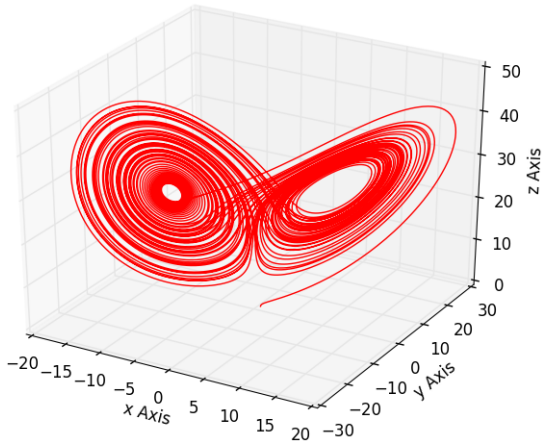


Figure 8: Plot showing aperiodicity

The system is clearly deterministic as the values of the parameters do not change while the trajectories are running.

The Lorenz attractor also has solutions which diverge away from each other after a short period of time. For example, figure 9 shows the x values of two trajectories with initial conditions $(1, 1, 1)$ and $(1.01, 1, 1)$.

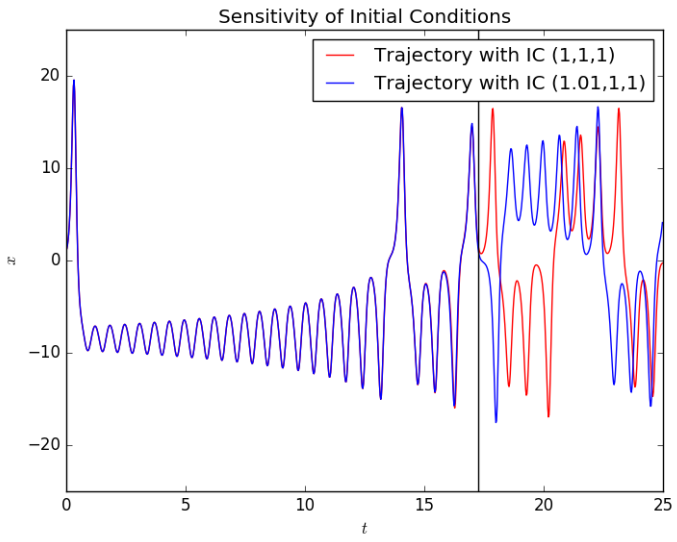


Figure 9: Sensitivity of Initial Conditions

Here, the first time at which the two trajectories diverge is highlighted by the black line at time $t = 17$. As the system satisfies all three conditions required for chaos, the system can be called chaotic.

4.2 Strange Attractor

We also discuss the property of strange attractiveness as defined in Strogatz. [4]

Defintion:
An attractor is a closed set A with three properties:

- A is an invariant set: any trajectory $\mathbf{x}(t)$ that starts in A stays in A for all time.
- A attracts an open set of initial conditions: there is an open set U containing A such that if $\mathbf{x}(0) \in U$, then the distance from $\mathbf{x}(t)$ to A tends to zero as $t \to \infty$.
- A is minimal: there is no proper subset of A that satisfies the two conditions above.

Further, a strange attractor is an attractor which exhibits sensitivity to initial conditions.

With no prior knowledge, it may not be clear that the equations have the property of attractiveness. This is due to "trajectories being attracted to a bounded set of zero volume which may not be minimal". It can be seen from figure 8 with the standard parameter values that all points tend to the attractor. Within the mathematical community, it is assumed that the Lorenz equations are an attractor. This was proved to be the case by Tucker in his paper 'A Rigorous ODE Solver and Smale's 14th Problem'. [5]

As the equations also hold the property of sensitive dependence to initial conditions shown in figure 9, our attractor may also be called a strange attractor.

5 General Considerations for Numerical Solutions

5.1 Reduced Set of Equations

We now wish to analyse the approximations carried out when computing the solutions of the Lorenz equations numerically. The purpose of this analysis is to determine the effectiveness of the numerical scheme chosen. For example, we wish to keep numerical errors to a minimum as there are a large number of calculations needed to compute the solutions after each time step. If at each step we have small deviations from the true value, these errors compound due to the multiplication operations occurring when computing the next values of $\dot{x}, \dot{y}$ and $\dot{z}$.

The main problems when using such schemes are highlighted to a greater effect when considering a reduced set of equations. This is done by omitting the nonlinear terms within equations (1). We may instantiate the equations with the values $\sigma = b = 1$ and $r = 4$. This gives the equations:

$$\begin{cases} \dot{x} = y - x \\ \dot{y} = 4x - y \\ \dot{z} = -z. \end{cases} \tag{9}$$

From this, we have a set of coupled differential equations for x and y and a linear differential equation for z.

Firstly, we solve the system of equations to find solutions for x and y. We write the equations in matrix form:

$$\underline{\dot{x}} = A\underline{x} = \begin{pmatrix} -1 & 1 \\ 4 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

We find the eigenvalues of the matrix A are $\lambda_1 = 1$ and $\lambda_2 = -3$, with corresponding eigenvectors

$$\underline{v_1} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \text{ and } \underline{v_2} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

Therefore we obtain the general solutions for x and y

$$\underline{x} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = C_1 e^{t} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + C_2 e^{-3t} \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

Solving the z equation using separation of variables, we find the general solution $z(t) = Ae^{-t}$.

We now impose initial conditions $x(0) = 1, \quad y(0) = -1, \quad z(0) = 2.$

The solution for z is simple as $z(0) = A = 2 \quad \text{so} \quad z(t) = 2e^{-t}$.

Imposing the first two conditions, we get

$$\begin{cases} x(0) = C_1 + C_2 = 1 \\ y(0) = 2C_1 - 2C_2 = -1. \end{cases} \tag{10}$$

Solving these simultaneously, we find $C_1 = \frac{1}{4}$ and $C_2 = \frac{3}{4}$. Thus, the final set of solutions are given by

$$x = \tfrac{1}{4}e^{t} + \tfrac{3}{4}e^{-3t}, \quad y = \tfrac{1}{2}e^{t} - \tfrac{3}{2}e^{-3t}, \quad z = 2e^{-t}.$$

For our comparison of numerical methods, we shall use the solutions at time $t = 1$. Thus we get the analytic values:

$$x(1) = \tfrac{e}{4} + \tfrac{3}{4e^3} \approx 0.717, \quad y(1) = \tfrac{e}{2} - \tfrac{3}{2e^3} \approx 1.284, \quad z(1) = \tfrac{2}{e} \approx 0.736.$$

5.2 Runge-Kutta 2

We now run the code found in appendix B. This uses the numerical method Runge-Kutta 2 (RK2) to find a numerical solution and investigates the error between the analytic and numerical values upon varying values of Δt.

The formula for the RK2 scheme is given by

$$\begin{aligned} k_1 &= \Delta t f(x_n, y_n) \\ k_2 &= \Delta t f(x_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2}) \\ y_{n+1} &= y_n + k_2 + O(\Delta t^3). \quad [7] \end{aligned}$$

Our error is calculated using the formula

$$E = \sqrt{(x(1) - x_f)^2 + (y(1) - y_f)^2 + (z(1) - z_f)^2} \tag{12}$$

where (x_f, y_f, z_f) is the exact solution at time $t = 1$.

The errors for varying Δt are given in Table 2.

Δt	E
0.001	5.79030×10^{-7}
0.005	1.45403×10^{-5}
0.01	5.84924×10^{-5}
0.05	1.53833×10^{-3}
0.1	6.65137×10^{-3}

Table 2: Error Depending on Time Step Using RK2

These errors may be plotted on a logarithmic scale as shown in Figure 10. Here we note that the points may be fitted with a line of best fit as shown in blue. Thus, we may assume that there is a relationship between the error and time increment of the form $E = c\Delta t^n$ for some constant c. We may take the logarithm of both sides of this postulated relationship to get an equation of the form

$$\log(E) = n \log \Delta t + \log(c).$$

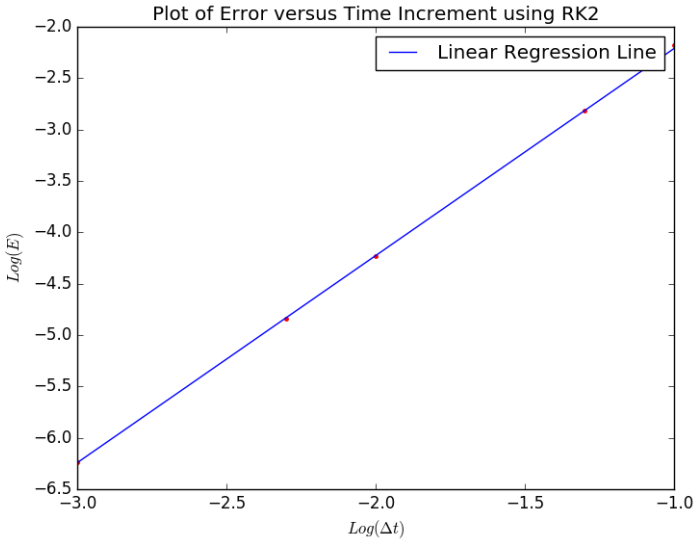


Figure 10: Logarithmic plot of error defined in (12) vs Δt using RK2 numerical method

This implies that slope of the straight line as given in the graph in Figure 10 determines the order of the numerical scheme chosen. When computing the slope of the regression line in our graphs, we will ignore the time stepping values for $\Delta t = 0.1$. This is done as this value is outside the range where the logarithmic scaling works effectively. In our case of RK2, the slope $n \approx 2.015$ suggests the scheme is second-order. This means the local error (error caused by a single iteration) is of order $O(\Delta t^3)$, which agrees with our formula for the RK2 scheme.

5.3 Runge-Kutta 4

We will perform similar analysis using a higher order scheme to obtain a more accurate set of numerical results. Our choice of scheme will be RK4, an altered version of RK2 with four increments of the interval used instead of two.

The formula for the RK4 scheme is given by

$$
\begin{aligned}
k_1 &= \Delta t f(x_n, y_n) \\
k_2 &= \Delta t f(x_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2}) \\
k_3 &= \Delta t f(x_n + \frac{\Delta t}{2}, y_n + \frac{k_2}{2}) \\
k_4 &= \Delta t f(x_n + \Delta t, y_n + k_3) \\
y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5). \quad [7]
\end{aligned}
$$

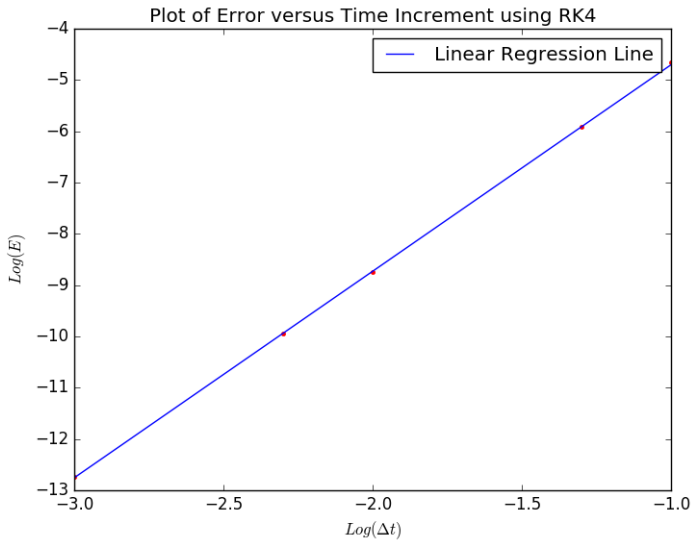


Figure 11: Logarithmic plot of error vs time using RK4 numerical method

Δt	E
0.001	1.79959×10^{-13}
0.005	1.11976×10^{-10}
0.01	1.81279×10^{-9}
0.05	1.24539×10^{-6}
0.1	2.24601×10^{-5}

Table 3: Error Depending on Time Step Using RK4

These errors are plotted in Figure 11. Using the same relationship as proposed beforehand, we find the order of the numerical scheme is determined by the slope of the line of best fit. For the case of RK4, $n \approx 4.026$, suggesting the scheme is fourth-order. Thus we find the local error is of order $O(\Delta t^5)$.

We shall continue to use the RK4 numerical scheme in further analysis as it has a number of advantages over other schemes. For example, RK4 balances the computational cost and increased accuracy well, compared to simpler methods. Table 4 shows a comparison between the order of the method and the number of evaluations required at each time step. The RK4 method is the highest Runge-Kutta method for which the order of global accuracy is equivalent to the number of increments calculated for each time step. This is beneficial as it is comparatively more efficient compared to higher order schemes.

Evaluations per time step	2	3	4	5	6
Maximum order	2	3	4	4	5

Table 4: Runge-Kutta Method Efficiency

6 Analysis of Numerical Solutions

6.1 Poincaré Sections

As seen from the figure on the title page, some trajectories seem to settle down onto two planar surfaces. It is often difficult to interpret these images as we are seeing a three-dimensional image on a two-dimensional plane. Thus, it is smart to convert this 3-D image into an equivalent 2-D representation. This can be done by taking a planar slice through the trajectory and plotting the x and y values whenever the trajectory passes through the planar slice.

To begin, we shall plot a trajectory with the parameter values of $\sigma = 10, b = \frac{8}{3}$ and $r = 28$ and initial condition $(1, 0, 0)$. Also on our plot, we show the Poincaré section of $z = r - 1 = 27$. This is seen in figure 12.

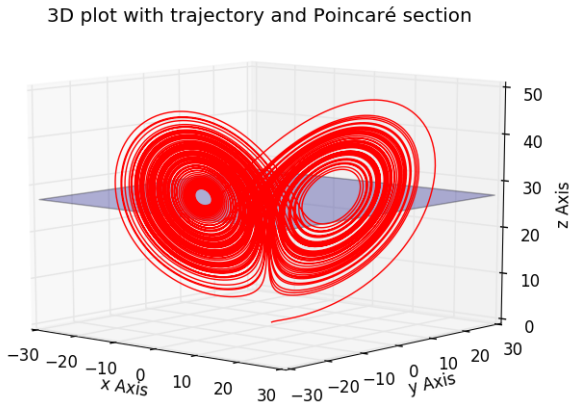


Figure 12: 3D Plot with Poincaré section of $z = r - 1$

We use a number techniques to get an accurate Poincaré section. For example, we remove the first twenty points of the Poincaré section which ensures that all points included in the plot are part of the Lorenz attractor. We also ensure that the trajectory is run to a long enough length to ensure enough points have been plotted on the section, which ensures a clear image has been plotted. In our case, we run the trajectory out to $t = 100$ with 30000 time steps.

In addition to this, whenever a candidate point for the section has been found, we interpolate between the current and previous z value. This ensures that all points from the trajectory would be found on the section, rather than just before or just after the trajectory passes through.

We mark the colour of the points on the Poincaré section with blue if the trajectory passes through the chosen plane from above and red if from below.

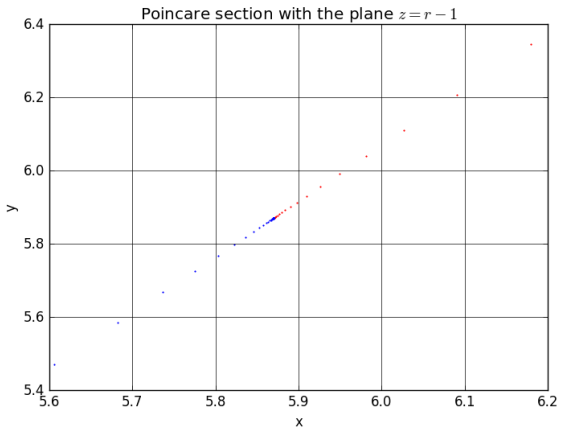


Figure 13: Plot for $r = 13.92$

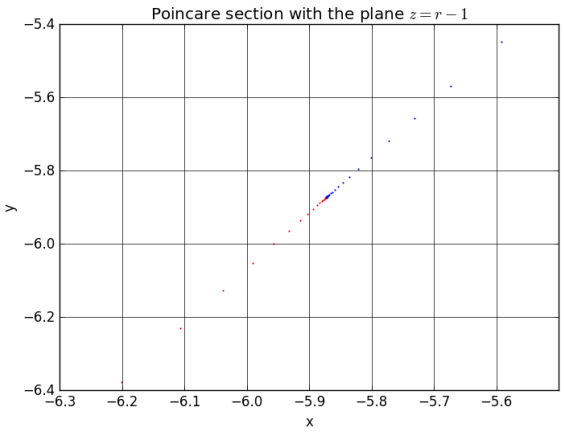


Figure 14: Plot for $r = 13.93$

For values of $r < 13.92$, all trajectories in the upper half plane pass through the plane $z = r - 1$ from below whereas those in the lower half plane pass from above. However, when the value is increased above 13.92, these roles are reversed. This can be seen in figures 13 and 14. This change corresponds to the trajectory changing from clockwise to anti-clockwise and the trajectory switching from being attracted from one fixed point to another. The trajectory passes through the plane in a circular motion, beginning from the outer reaches of the plane and slowly winds inwards.

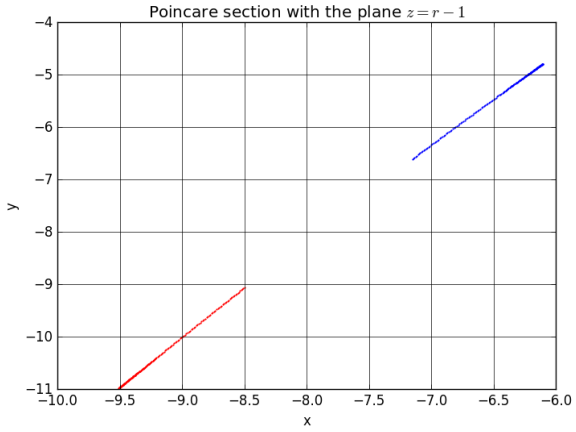


Figure 15: Plot for $r = 24$

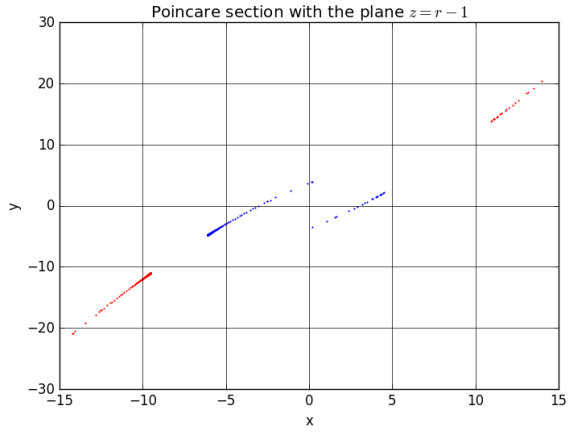


Figure 16: Plot for $r = 24.1$

The next notable change occurs at approximately $r = 24$, as seen in figures 15 and 16. We can see that the two straight lines of the section separate out into two 'wings' which curve toward one another. Here we notice there is a correlation between the closeness to the centre of the xy plane and the way in which the trajectory travels for values of $r > 24.1$. Points closer to the origin tend to pass through the plane from above whereas those further from the origin pass from below. In the case of $r = 24$, the trajectory begins on the outside of the section and slowly moves inward, alternating between passing through the plane from above and below. For $r = 24.1$, the trajectory passes through the plane along one of the two curves, undergoes some cyclic motion before switching over to the other curved line.

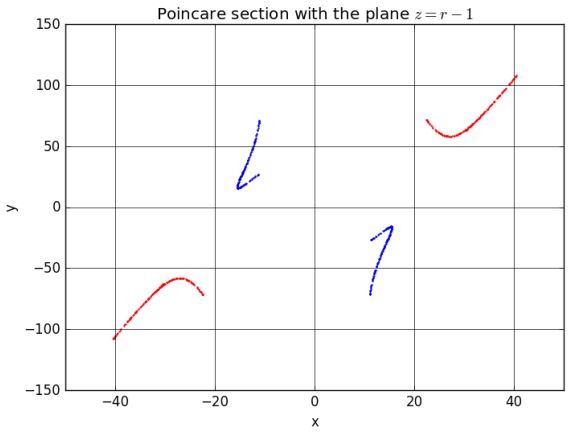


Figure 17: Poincare Section for $r = 200$

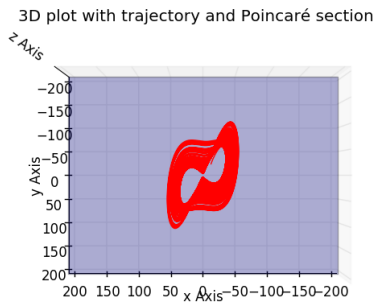


Figure 18: Plot of xy plane for $r = 200$

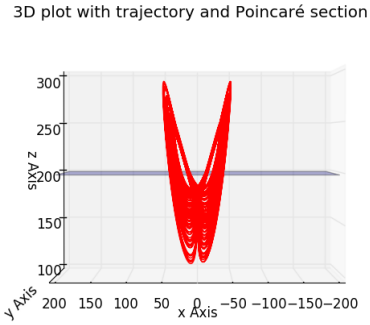


Figure 19: Plot of xz plane for $r = 200$

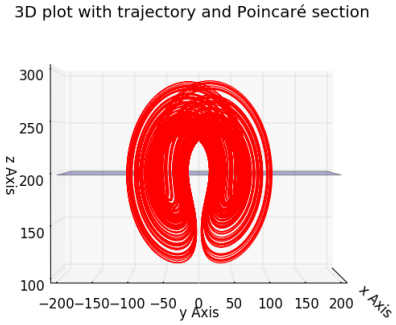


Figure 20: Plot of yz plane for $r = 200$

In figures 18 to 20 we plot the different planes from which the trajectory and the Poincaré plane at $z = 199$.

Increasing the value of r to 200, we see from figure 16 that the wings of the section rapidly curve inwards/outwards along a curve, which contrasts the linearity seen in previous plots. However, the trajectory which is followed seems to follow a fairly regular pattern contrasting the chaotic solutions as seen for $r = 24$ for example. Again we notice the pattern of trajectories which pass through the plane closer to the origin pass from above and ones further away from below. There is also a symmetric pattern between the shape of the wings seen from the trajectory. This can be seen much clearer in 17, with the red and blue patterns showing almost identical shapes. This implies the idea of structure within the system for large values of r. This idea will be revisited in section **6.4**.

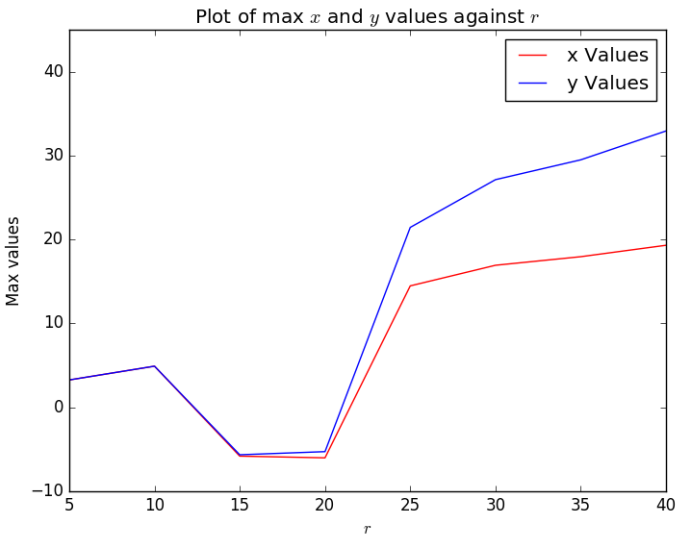


Figure 21: Plot of x and y against r

It may be of interest to find the maximal x and y values for differing values of r. In figure 21 we plot the maximal values of x and y versus values of r incrementing from 5 to 40 in increments of 5. Here we see maximal values of x and y remain close to each other for $5 < r < 15$, with a rapid divergence between the two for $r > 15$. The difference between maximal x and y values for each value of r can be seen in table 5.

r	$\lvert x_{max} - y_{max} \rvert$
5	0.0000001936
10	0.0032672013
15	0.1637843238
20	0.7373810363
25	6.9664260194
30	10.2005335513
35	11.5420254593
40	13.6189039576

Table 5: Distance between maximal x and y values

This behaviour is similar regardless of the initial condition used. For the majority of cases, the x and y max values diverge away from each other at a value of r between 15 and 20, with a significant difference between the maximal values for $r = 40$.

6.2 Lorenz Map

In this section, we will recreate an investigation first posed by Lorenz in his paper 'Deterministic Nonperiodic Flow'. [8]

We begin by considering a chaotic trajectory of the Lorenz equations. Here, we choose the usual parameters of $\sigma = 10, b = \frac{8}{3}$ and $r = 28$ and initial condition $(1, 1, 1)$ which we have seen produces a chaotic solution. From this, we find the maxima of the z points along this trajectory labelling the nth of these as z_n. Here we plot the first 100 points, rather than all 300 time stepping points used when plotting figure 23.

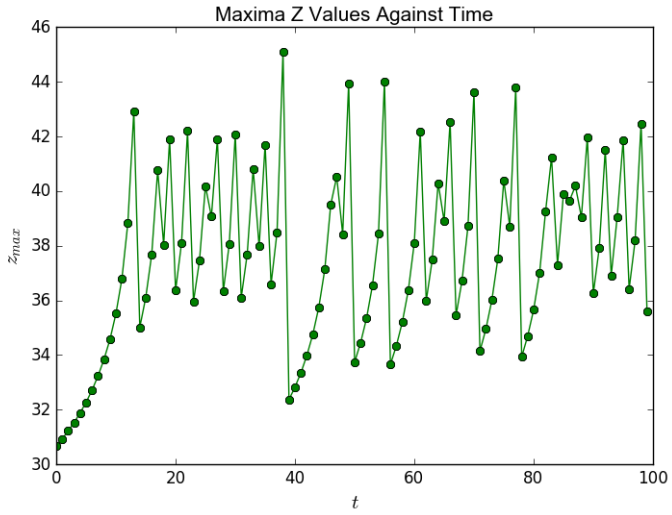


Figure 22: Maxima Z Values

Upon finding all maximum points, we plot z_{n+1} against z_n. The graph in Figure 23 is the outcome of this procedure.

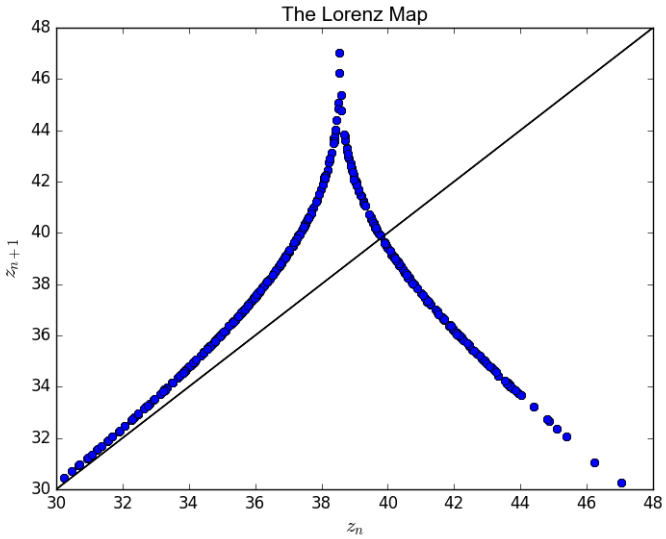


Figure 23: Plot of the Lorenz Map

To improve the accuracy of this figure, we have ensured the use of a high order time stepping algorithm, namely RK4 as it is more efficient compared to higher order schemes. We have also removed

the first 500 points calculated to ensure we are travelling along the attractor at all times.

Figure 23 shows a clear correlation between successive maxima of z values. This is an example of creating order within chaos. If we know the value of a local maximal z value, we are able to accurately predict the value of the next maximal z value. Thus, these values can be called short-term predictable. There is a striking resemblance between the graph seen in Figure 23 and a map called the Tent map in Figure 24. As both of these maps are similar when graphed, we may analyse the simpler Tent map rather than the more complex Lorenz map.

6.2.1 Tent Map

The tent map is defined as:

$$x_{n+1} = \begin{cases} 2x_n & \text{if } x_n < \frac{1}{2} \\ 2(1 - x_n) & \text{if } \frac{1}{2} \leq x_n. \end{cases} \tag{14}$$

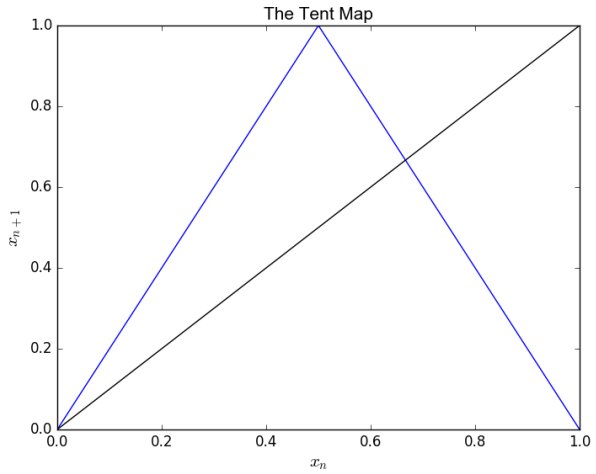


Figure 24: Plot of the Tent Map

This tent map can be shown to be chaotic on the interval $[0, 1]$. We begin with a definition from Devaney [9].

Definition: Let V be a set. $f : V \to V$ is said to be chaotic on V if:

- f has sensitive dependence on initial conditions
- f is topologically transitive
- Periodic points are dense in V

We will prove this by closely following an argument posed by Justin Guo in his paper 'Analysis of Chaotic Systems' [10].

The tent map as defined above has sensitive dependence on initial conditions. This requires a "$\beta > 0$ such that for any $x_0 \in V$ and an open interval U about x_0, there is some $y_0 \in U$ and $n > 0$ such that $|f^n(x_0) - f^n(y_0)| \geq \beta$. We take an $x_0 \in [0, 1]$ and $\beta = \frac{1}{2}$. Any open interval around x_0 will be mapped to $[0, 1]$ after n iterations of the map where n is sufficiently large. Thus there exists y_0 in the open interval such that $|f^n(x_0) - f^n(y_0)| \geq \frac{1}{2}$".

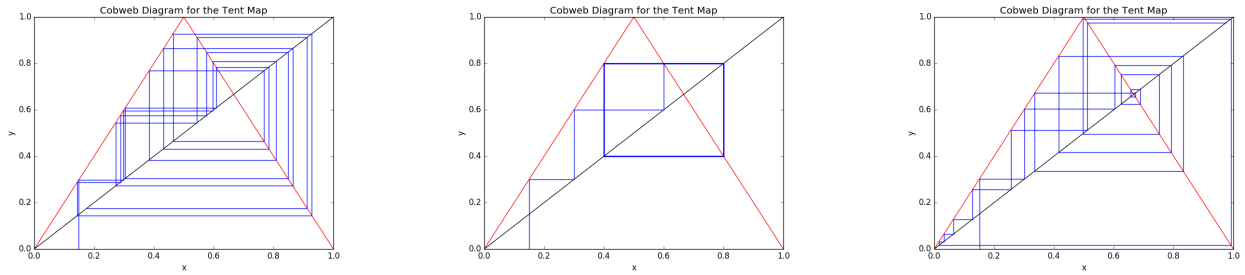


Figure 25: Cobweb diagrams with initial condition $x = 0.149$, $x = 0.15$ and $x = 0.151$

This sensitive dependence can be seen via cobweb diagrams with slightly altered initial conditions. As seen in figure 25, changing the initial value by 0.01 drastically changes the behaviour of the trajectories.

A map is transitive if "given any two open subintervals U_1 and U_2 in V, there is a point $x_0 \in U_1$ and $n > 0$ such that $f^n(x_0) \in U_2$. We take two open subintervals as above. For large enough n, U_1 contains an interval of the form $\left[\frac{k}{2^n}, \frac{k+1}{2^n}\right]$. Thus T^n maps U_1 onto [0,1] which contains U_2".

For example, we can take equal subintervals $U_1 = (0, \frac{1}{2})$, $U_2 = (\frac{1}{2}, 1)$ and $x_0 = 0.3$. Then $T(0.3) = 0.6 \in U_2$. Repeating this argument for all possible subintervals of $[0, 1]$ with suitable values of n proves transitivity.

"The nth iterate of the tent map maps each interval $\left[\frac{k}{2^n}, \frac{k+1}{2^n}\right]$ to $[0, 1]$ for $k = 0, 1, 2..., 2^n - 1$. As seen in figure 24, the first iterate of the map intersects the line $y = x$ once in each interval". In the case of the first iterate, these are found to be $x_1 = 0$ and $x_2 = \frac{2}{3}$. "These are each fixed points and thus a periodic point of period n. For large iterates of n, the periodic points are dense in the interval V".

As all three criteria has been met, we can say the tent map is chaotic.

6.2.2 Alternate Proof of Chaos

We may prove the same result using the main theorem found in "Period Three Implies Chaos" by Yorke and Li [11]. In this paper, it is stated that "if there is a periodic point of period 3" then chaos can be found. We have

$$T\left(\frac{2}{7}\right) = \frac{4}{7}, \quad T\left(\frac{4}{7}\right) = \frac{6}{7}, \quad T\left(\frac{6}{7}\right) = \frac{2}{7}. \tag{15}$$

Therefore, as $\frac{2}{7}$ is mapped back to $\frac{2}{7}$ after 3 iterations, the point is of minimal period 3 and thus the system is chaotic.

6.2.3 Varying r

We now wish to vary the value of r and see where the Lorenz map first appears similar to the tent map. This may be used as a test of chaos and allows us to see how the value of r affects chaos within the Lorenz system. We begin with the usual parameters of $\sigma = 10, b = \frac{8}{3}$ and initial condition of $(1, 1, 1)$.

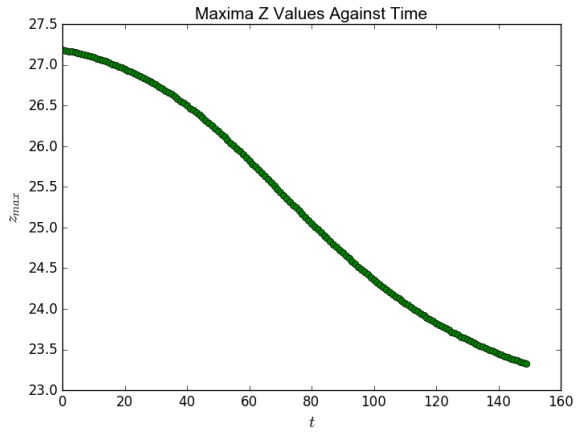


Figure 26: Plot of z_{max} for $r = 23.7$

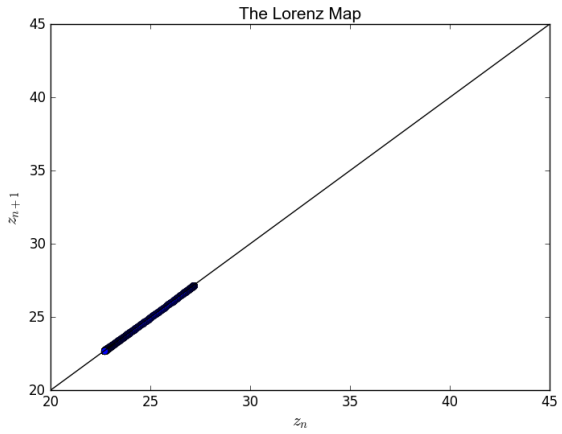


Figure 27: Lorenz Map for $r = 23.7$

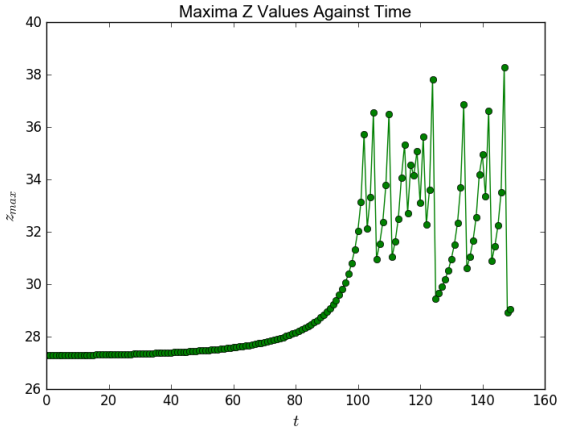


Figure 28: Plot of z_{max} for $r = 23.8$

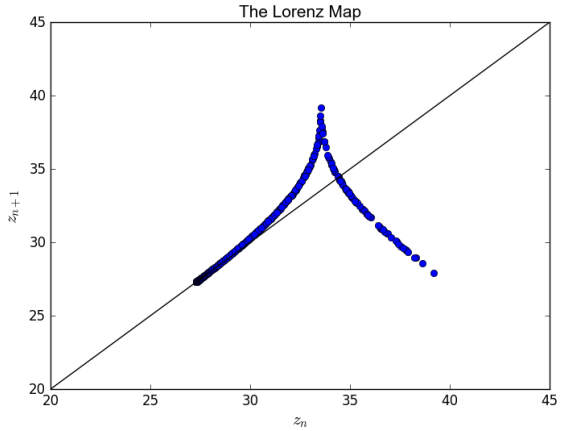


Figure 29: Lorenz Map for $r = 23.8$

The point at which the tent-like Lorenz map first appears is between $r = 23.7$ and 23.8. This can be seen from figures 27 and 29. We can make a link between the plots of z_{max} versus time and the Lorenz Map. From our plots of figures 26-29 it can be seen that an initially decreasing map with a fairly smooth path produces a Lorenz map which is linear whereas an initially increasing plot with eventually 'chaotic' points produces a tent-like Lorenz map.

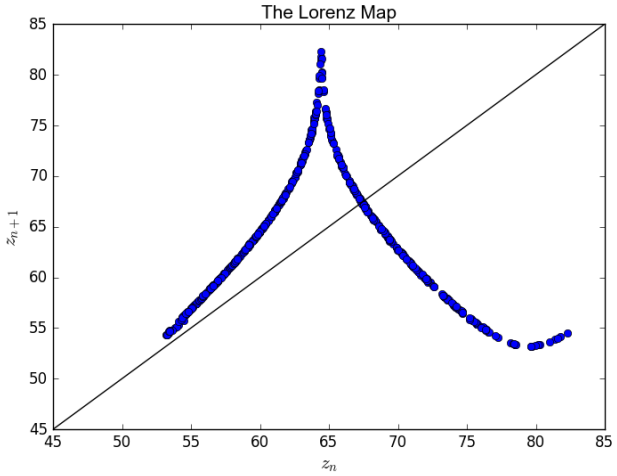


Figure 30: Lorenz Map for $r = 50$

Increasing the value of r to 50, we notice that the right wing of the points plotted begins to curve away from the linear shaped wing as seen in figure 29. However, this map can still be compared to the Tent map and may still have the same properties relating to chaos.

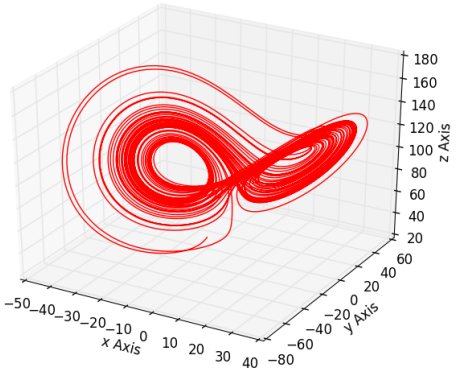


Figure 31: Trajectory for $r = 100$

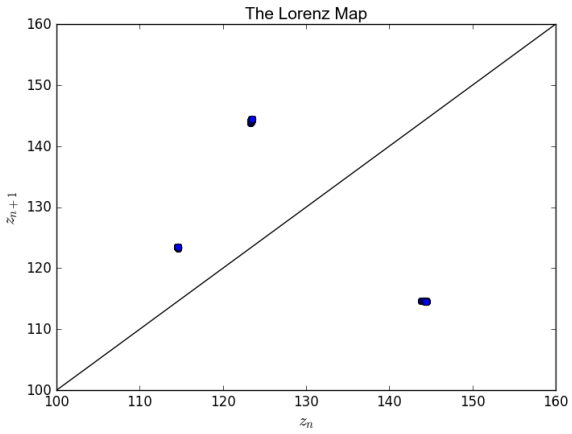


Figure 32: Lorenz Map for $r = 100$

When $r = 100$, the trajectory seems to settle down into a cyclic motion and becomes less chaotic. This is highlighted in figure 32, where adjacent maximal z values are plotted onto one of three points. The maxima of z move in a cyclic pattern between the three.

$$z_n \approx 145 \to 115 \to 125 \to 145 \cdots$$

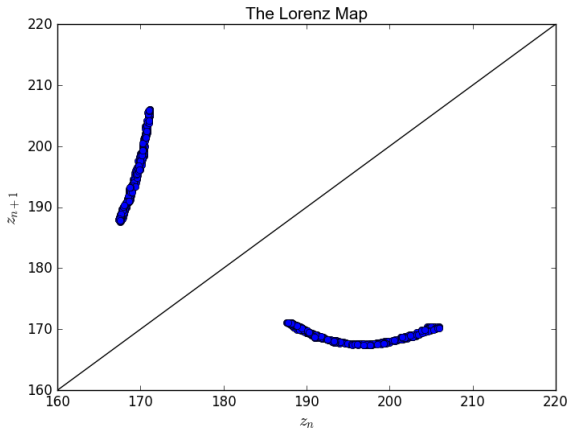


Figure 33: Lorenz Map for $r = 143$

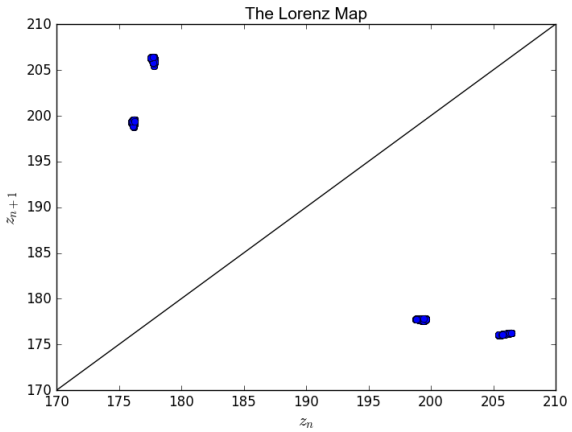


Figure 34: Lorenz Map for $r = 150$

For the case of $r = 143$, we notice the Lorenz map tent shape falls away and gives rise to two separate wings. The upper section of the standard map as seen in figure 23 has been condensed into individual wings with map points clumped closer together. Increasing r further to 150, these wings collapse further into four individual condensed regions of points. These points again fall into a cyclic motion.

$$z_n \approx 199 \to 177 \to 206 \to 176 \to 199 \cdots$$

We now consider another initial condition to see if this affects the point at which the tent structure appears. Here we take $(-10, 0, 27)$ as our initial condition.

We see the tent structure begins to emerge at around $r = 20$. This can be seen in figures 35 and 36.

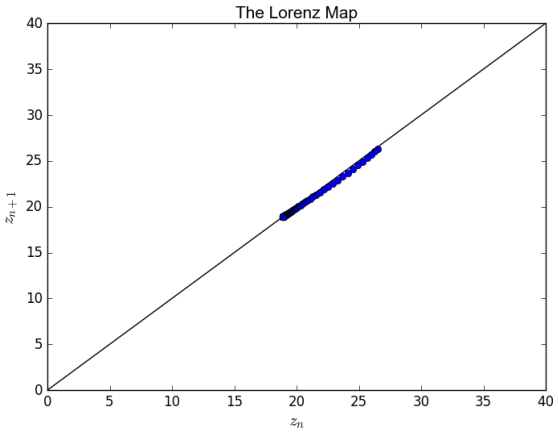


Figure 35: Lorenz Map for $r = 19.9$

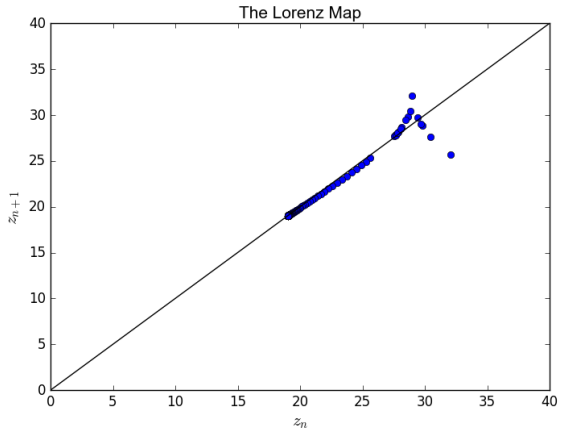


Figure 36: Lorenz Map for $r = 20$

Continuing again up to $r = 50$, we see the right wing once again curving away from the tent map structure. This is the same value of r as for the other initial condition $(1, 1, 1)$. Continuing this analysis for a variety of initial conditions results in similar values for r of around 50.

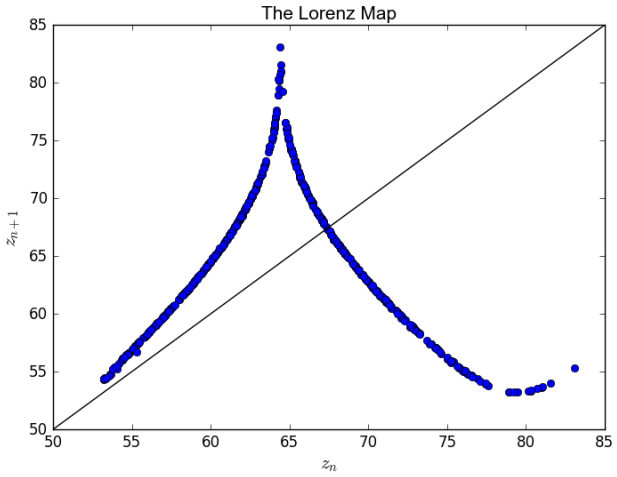


Figure 37: Lorenz Map for $r = 50$

6.3 Limitations of Time Stepping and Predictability

In this section, we will discuss the limitations we may find when using our RK4 time stepping procedure. We wish to analyse the convergence of trajectories toward the final values upon decreasing the time stepping width. We will keep the parameter values constant, with $\sigma = 10, b = \frac{8}{3}$ and, initially, $r = 5$. We shall also keep the initial value of the trajectory the same, with $\mathbf{x} = (1, 0, 0)$. We shall take a tolerance of 10^{-3} for our trajectory to converge to a point.

We complete a number of computer simulations to calculate the end points of the trajectories. These will begin at time $t = 1$ and will also be computed at $t = 10$ and $t = 100$. Upon varying the time step width, we should expect patterns of convergence towards a unique point, regardless of the time width chosen. All values in the tables below are to 10 decimal places.

Δt	x	y	z
0.01	3.4391956687	4.0961105441	2.0023265023
0.005	3.4391956207	4.0961105361	2.0023265026
0.002	3.4391956176	4.0961105356	2.0023265027
0.001	3.4391956176	4.0961105356	2.0023265027

Table 6: Final Trajectory Values for time $t = 1$ and $r = 5$

Δt	x	y	z
0.01	3.2659538735	3.2661303877	3.9994879062
0.005	3.2659538738	3.2661303878	3.9994879068
0.002	3.2659538738	3.2661303878	3.9994879069
0.001	3.2659538738	3.2661303878	3.9994879069

Table 7: Final Trajectory Values for time $t = 10$ and $r = 5$

Δt	x	y	z
0.01	3.2659863237	3.2659863237	4.0000000000
0.005	3.2659863237	3.2659863237	4.0000000000
0.002	3.2659863237	3.2659863237	4.0000000000
0.001	3.2659863237	3.2659863237	4.0000000000

Table 8: Final Trajectory Values for time $t = 100$ and $r = 5$

From the tables above, we can clearly see each of the trajectories converge to a unique point as the value of $\Delta t \to 0$, upon varying t.

Now we consider the case where we increase the value of r to 28.

Δt	x	y	z
0.01	-9.4084966328	-9.0962390229	28.5816945968
0.005	-9.4084511264	-9.0962003927	28.5816275665
0.002	-9.4084505456	-9.0961990866	28.5816275577
0.001	-9.4084505650	-9.0961990718	28.5816276189

Table 9: Final Trajectory Values for time $t = 1$ and $r = 28$

Δt	x	y	z
0.01	-5.8575641373	-5.8306244009	23.9325346464
0.005	-5.8576854531	-5.8310709511	23.9321516861
0.002	-5.8576855021	-5.8310824553	23.9321333659
0.001	-5.8576853924	-5.8310824900	23.9321330086

Table 10: Final Trajectory Values for time $t = 10$ and $r = 28$

Δt	x	y	z
0.01	-5.8467410933	-9.1621184385	16.7614887659
0.005	-4.4292724035	-6.6012621167	21.758219648
0.002	1.0885981333	-0.7735770283	22.829563254
0.001	4.4592913791	8.8741586923	6.5428568381

Table 11: Final Trajectory Values for time $t = 100$ and $r = 28$

Using our tolerance of 10^{-3}, we can analyse for what value of Δt the trajectory converges. At time $t = 1$, the trajectory converges to a unique point for values of $\Delta t \leq 0.01$. For $t = 10$, the trajectory converges to a point for values of $\Delta t \leq 0.005$ but a slightly differing value for $\Delta t = 0.01$. For $t = 100$, changing Δt does not affect the convergence of the trajectory.

Upon performing further computer simulations, we are able to determine the value where t_{max} first fails to converge to a common point. Using our leniency of 10^{-3}, this point occurs at approximately $t = 22$ which is highlighted in Table 12.

Δt	x	y	z
0.01	-1.7449120751	-2.6958098529	14.4755360817
0.005	-2.0761192880	-3.2547780267	14.434139995
0.002	-2.0756398637	-3.2539790572	14.4341160619
0.001	-2.0753115737	-3.2534265517	14.4341325182

Table 12: Final Trajectory Values for time $t = 22$ and $r = 28$

We note that using $\Delta t = 0.002$ converges to $(-2.076, -3.254, 14.434)$ whereas using $\Delta t = 0.001$ converges to $(-2.075, -3.253, 14.434)$.

However, upon decreasing the time step width even further, we can see that convergence is possible for some values. For example, as an extension to table 12 with a smaller time step of 0.001, we notice a clear pattern of convergence.

Δt	x	y	z
0.0005	-2.0752842875	-3.2533806253	14.4341339134

Table 13: Final Trajectory Values for time $t = 22$ and $r = 28$

This does not hold for all values of t_{max}. An example of this can be seen in Table 14.

Δt	x	y	z
0.01	-4.6407378081	2.7218009780	31.5206286711
0.005	5.5238501852	4.5112009643	25.0866479526
0.002	0.7745183247	1.4269345194	14.9852717395
0.001	-3.9694320227	-1.0402944043	26.3983472704
0.0005	-6.2027620256	-8.9553653010	18.9981208925

Table 14: Final Trajectory Values for time $t = 70$ and $r = 28$

We may also alter the initial condition of the trajectory to see if this affects the interval over which the trajectories converge. When we use the initial condition of $(-10, 0, 27)$ we find the trajectory fails to converge when $t_{max} = 12$ as seen in table 16.

Δt	x	y	z
0.01	-8.6459642711	-2.2345950070	33.8593301035
0.005	-8.8170182464	-2.2917257736	34.1196833809
0.002	-8.8164050162	-2.2914657806	34.1187951937
0.001	-8.8162178949	-2.2913951110	34.1185168116

Table 15: Final Trajectory Values for time $t = 11$ and $r = 28$

Δt	x	y	z
0.01	-0.1943781979	0.2006368660	17.3444445406
0.005	-0.0300049233	0.5137200941	17.7495499732
0.002	-0.0304762096	0.5127866098	17.7481886229
0.001	-0.0306372047	0.5124716821	17.7477445114

Table 16: Final Trajectory Values for time $t = 12$ and $r = 28$

From table 15 we see that shortening the time width allows the trajectory to converge to a uniform point. However, when increasing to $t = 12$, we note that the trajectory when using $\Delta t = 0.002$ ends at the point $(-0.030, 0.513, 17.748)$ whereas using $\Delta t = 0.001$ ends at the point $(-0.031, 0.512, 17.748)$.

6.4 Lorenz Equations for Large Values of r

6.4.1 Reduced Equations

We now consider the behaviour of solutions of the Lorenz equations in the limit as $r \to \infty$. This is motivated by Strogatz [4] exercise 9.5.5 and will follow similar work laid out by Robbins [12].

Consider the change of variables below:

$$X = \epsilon x \quad Y = \epsilon^2 \sigma y \quad Z = \sigma(\epsilon^2 z - 1) \quad \tau = \tfrac{t}{\epsilon} \quad \epsilon = \tfrac{1}{\sqrt{r}}.$$

The choice of epsilon allows $\epsilon \to 0$ as $r \to \infty$. This will become important as it removes unwanted terms in the analysis.

Taking time derivatives of x, y and z

$$\begin{aligned}
\dot{x} &= \frac{1}{\epsilon^2}\frac{dX}{d\tau} = \sigma(y - x) = \sigma(\frac{Y}{\epsilon^2\sigma} - \frac{X}{\epsilon}) \\
\dot{y} &= \frac{1}{\epsilon^3\sigma}\frac{dY}{d\tau} = rx - xz - y = \frac{rX}{\epsilon} - \frac{X}{\epsilon^3}(\frac{Z}{\sigma} + 1) - \frac{Y}{\epsilon^2\sigma} \\
\dot{z} &= \frac{1}{\epsilon^2}(\frac{1}{\sigma\epsilon} + 1)\frac{dZ}{d\tau} = xy - bz = \frac{XY}{\epsilon^3\sigma} - \frac{b}{\epsilon^2}(\frac{Z}{\sigma} + 1).
\end{aligned} \tag{16}$$

From these equations, we find that

$$\begin{aligned}
\frac{dX}{d\tau} &= Y - \sigma\epsilon X \\
\frac{dY}{d\tau} &= -XZ - \epsilon Y \\
(1 + \epsilon\sigma)\frac{dZ}{d\tau} &= XY - b\epsilon\sigma(\frac{Z}{\sigma} + 1).
\end{aligned} \tag{17}$$

Taking the limit $\epsilon \to 0$ in each equation

$$\begin{aligned}
\lim_{\epsilon\to 0}\frac{dX}{d\tau} &= Y \\
\lim_{\epsilon\to 0}\frac{dY}{d\tau} &= -XZ \\
\lim_{\epsilon\to 0}(1 + \epsilon\sigma)\frac{dZ}{d\tau} &= XY.
\end{aligned}$$

Thus the equations become

$$\begin{aligned}
X' &= Y \\
Y' &= -XZ \\
Z' &= XY.
\end{aligned} \tag{18}$$

6.4.2 Constants of Motion

We can now find the constants of motion for the system with $r \to \infty$. These are quantities which are conserved throughout the motion along the Lorenz curve for the reduced set of equations.

$$\begin{aligned}
(X^2 - 2Z)' &= 2XX' - 2Z' = 2XY - 2XY = 0 \\
(Y^2 + Z^2)' &= 2YY' - 2ZZ' = 2Y \cdot -XZ - 2Z \cdot XY = 0.
\end{aligned}$$

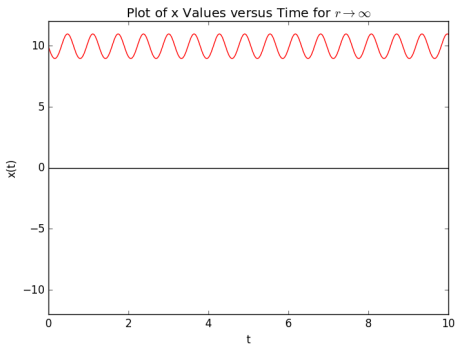
6.4.3 Volume Preservation

It can be shown that the new system is volume-preserving in a similar way to that of section 3.2. That is, the system of equations ensures volumes are kept constant even though the shape of the object in space may change. Taking $\mathbf{F'} = (X', Y', Z')$

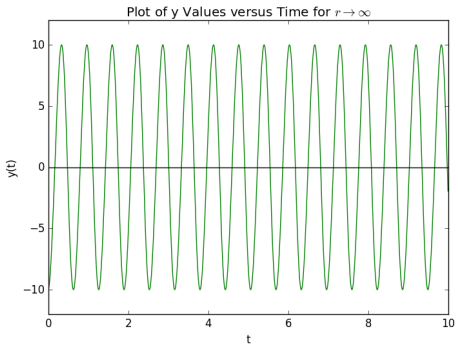
$$\dot{V} = \int_V \nabla \cdot \mathbf{F}' \, dV = \int_V \nabla \cdot (X', Y', Z') \, dV = \int_V \nabla \cdot (Y, -XZ, XY) \, dV = \int_V 0 \, dV = 0.$$

Thus, the rate of change of volume with respect to time is 0 and is therefore constant.

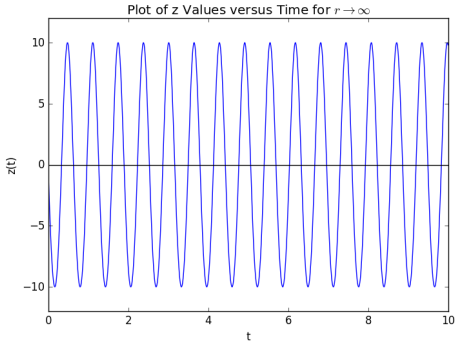
6.4.4 Long Term Behaviour



(a)



(b)



(c)

Figure 38: (a) Plot of x values in the case of $r \to \infty$, (b) Plot of y values in the case of $r \to \infty$, (c) Plot of z values in the case of $r \to \infty$

As seen in figure 38, the x, y and z values settle into periodic motion as $r \to \infty$. We also notice that the change in x values is much less significant compared to those of y and z. As this is also the case for the normal system, we can say our simpler system is a good approximate for the Lorenz equations for large values of r.

7 Conclusion

I hope that this paper provides a detailed overview of the Lorenz Equations and the interesting properties it beholds.

We have given an in depth analysis into various aspects of the nonlinear system. We initially investigated the three fixed points the system beholds and their stability, making use of the Jacobian matrix analysis. We also generalised the value of r_h where the two fixed points x_+ and x_- change in stability from stable to unstable. This is important as it is a first indicator as to where chaotic solutions arise. We then moved onto plotting solutions in three dimensions to allow a more visual investigation to take place.

In section 3, we looked at further properties of the equations such as symmetries and the property of being dissipative. We showed that volumes contract under the flow exponentially quickly with use of the divergence theorem. We also showed, using Lyapunov functions, that the solutions of the equations are located within a bounded region. This ensured that no solutions diverged away to infinity.

In section 4, we gave a brief overview of chaos and how it can be seen from our system. We also discussed how the system may be described as a 'Strange Attractor', referencing Tucker's paper proving this property.

We then moved onto an analysis of different numerical schemes which could be used in further analysis. This was done by instantiating specific parameter values and solving these to give an exact solution. Upon using the time stepping schemes, we could then compare the final values in order to determine the accuracy. From this, we could also find the order of the numerical schemes. As a result of this analysis, we found the numerical scheme of RK4 to be the most beneficial for our analysis purposes.

In section 6, we used a range of mathematical techniques in order to analyse the system in greater detail. We began with Poincaré sections which allowed our 3D trajectories to be reduced into a 2 dimensional image. This allowed for a much more accurate visualisation of the system and its behaviour. We then investigated the Lorenz map which plotted adjacent maximal z values against each other which then formed a tent-like structure for the standard chaos parameter values. We then compared this map to a similar looking map called the Tent map and determined that this was chaotic. We also varied the value of r to see where this pattern emerged and any order which could be formed.

In addition to this, we looked into the limitation of time stepping within our system by varying the width of the time step used and the time at which the point was found to see if this impacted the convergence to a unique point. This was performed for a number of values of r and final time t. We also investigated whether changing the initial condition has an affect of scope of convergence. We used a change of variables to convert the standard equations to a reduced set which can be analysed in the limit of $r \to \infty$. We showed the solutions settle into a cyclic motion and that our reduced system was appropriate for the case of the limit of r.

If the reader would like to explore the topic further, I would advise reading any of the books referenced at the end of the paper, especially Strogatz and Sparrow. It has become clear that chaotic solutions can arise unexpectedly within such a system, with the solution's properties depending drastically on the initial conditions and input parameters. The use of the python programming language proved an excellent tool for computing solutions and analysing the system in depth.

I would like to study the system in more detail. For example, I would investigate the fixed point analysis in further detail, making use of Lyapunov exponents. I would also wish to investigate other nonlinear models such as the Chen attractor or the Lorenz 84 equations. However, this would require a considerable amount of additional time and significantly more computational power. It would also be preferable to develop a derivation for the equations from a chaotic waterwheel model as well as

discussing bifurcation theory in further detail.

As chaos theory is such an important part of both the physical and mathematical world, it is vital we continue further research as this would allow us to model real life situations and predict future outcomes as a result. It is a simple concept which most non-mathematicians are able to grasp and may experience in everyday life. For example: weather systems, data encryption, population dynamics and electrical circuit systems.

The aim of this paper was to provide an insight into chaos theory within a relatively basic system. With our new mathematical and computational skills, we can apply these techniques to other systems to help develop our knowledge of chaotic systems. The development of this paper has also allowed me to improve my programming knowledge of a new programming language extensively.

I would like to thank Dr. Stephen Griffiths for his invaluable guidance throughout the development of this paper and guiding me to investigate the system through a number of differing viewpoints and techniques.

A Code

A.1 Computation of Eigenvalues

Below is a section of code which receives a matrix and a number of parameters to be evaluated within the matrix.

```
import math
import numpy as np

#Fixing values of sigma and b and varying r
sigma = 10
b = 8/3
r = 128

a = np.array([[-sigma,sigma,0],[1,-1,-math.sqrt(b*(r-1))],
[math.sqrt(b*(r-1)),math.sqrt(b*(r-1)),-b]])

evals,evecs = np.linalg.eig(a)
print(evals)
```

A.2 Global Behaviour

Below is code which creates 3-D plots of trajectories for two differing initial conditions.

```
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts
from mpl_toolkits.mplot3d import Axes3D

# parameters
b = 8/3
sigma = 10
r = 28

# time-stepping parameters
tmax = 30 # run to this time
nsteps = 10000 # number of time steps
dt=tmax/ nsteps # calculate the time step

# initial conditions (x Array: 3 position vars)
t=0

x1 = np.array([1,0,0])
x2 = np.array([0,-1,r-1])

# number of timesteps taken; initialise to 0
n=0

xValues1 = [x1[0]]
yValues1 = [x1[1]]
zValues1 = [x1[2]]

xValues2 = [x2[0]]
yValues2 = [x2[1]]
zValues2 = [x2[2]]
```

```
tValues = [t]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x_Axis')
ax.set_ylabel('y_Axis')
ax.set_zlabel('z_Axis')


while n < nsteps:
    x1=ts.step_rk4(x1,dt,b,r,sigma)
    x2=ts.step_rk4(x2,dt,b,r,sigma)

    xValues1 = np.append(xValues1, x1[0])
    yValues1 = np.append(yValues1, x1[1])
    zValues1 = np.append(zValues1, x1[2])

    xValues2 = np.append(xValues2, x2[0])
    yValues2 = np.append(yValues2, x2[1])
    zValues2 = np.append(zValues2, x2[2])

    tValues = np.append(tValues, t)


    t = t+dt
    n = n+1


#print(t, x1[0], x1[1], x1[2])
#print(t, x2[0], x2[1], x2[2])

#3D Plot
plt.figure(1)
ax.plot(xValues1, yValues1, zValues1, c= 'red')
ax.plot(xValues2, yValues2, zValues2, c= 'blue')

plt.show()
plt.savefig('test.pdf')

#plt.figure(2)
#plt.plot(tValues, xValues1, 'r')
#plt.plot(tValues, xValues2, 'b')
```

A.3 Chaos

Below is code which shows how small deviations in initial conditions affect the final path of the trajectories.

```
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts,
matplotlib.lines as mlines



# parameters
b = 8/3
```

```
r = 28
sigma = 10

# time-stepping parameters
tmax = 25 # run to this time
nsteps = 1000 # number of time steps
dt=tmax/ nsteps # calculate the time step

# initial conditions (x Array: 3 position vars)
t=0

x1 = np.array([1,1,1])
x2 = np.array([1.01,1,1])

# number of timesteps taken; initialise to 0
n=0

xValues1 = [x1[0]]
yValues1 = [x1[1]]
zValues1 = [x1[2]]

xValues2 = [x2[0]]
yValues2 = [x2[1]]
zValues2 = [x2[2]]

tValues = [t]


while n < nsteps:
    x1=ts.step_rk4(x1,dt,b,r,sigma)
    x2=ts.step_rk4(x2,dt,b,r,sigma)

    xValues1 = np.append(xValues1, x1[0])
    yValues1 = np.append(yValues1, x1[1])
    zValues1 = np.append(zValues1, x1[2])

    xValues2 = np.append(xValues2, x2[0])
    yValues2 = np.append(yValues2, x2[1])
    zValues2 = np.append(zValues2, x2[2])

    tValues = np.append(tValues, t)

    t = t+dt
    n = n+1


LineXValues = [17.25,17.25]
LineYValues = [25,-25]


plt.figure(1)
plt.plot(tValues, xValues1, c='red')
plt.plot(tValues, xValues2, c='blue')
```

```
plt.title('Sensitivity of Initial Conditions')
plt.xlabel(r'$t$')
plt.ylabel(r'$x$')

plt.plot(LineXValues,LineYValues, 'k-')

plt.ylim(-25,25)


red_line = mlines.Line2D([], [], color='red', marker='.', markersize=0,
label='Trajectory with IC (1,1,1)')
blue_line = mlines.Line2D([], [], color='blue', marker='.', markersize=0,
label='Trajectory with IC (1.01,1,1)')
plt.legend(handles=[red_line, blue_line])


plt.show()
```

A.4 Numerical Solution of ODEs

This appendix entry shows a piece of code which determines the error between the numerical and analytic solution for the simplified system as discussed in Section 5.

```
# time steps the Lorenz equations
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK2 as ts,
matplotlib.lines as mlines
from mpl_toolkits.mplot3d import Axes3D

# parameters
b = 1
r = 4
sigma = 1


# time-stepping parameters
tmax = 1 # run to this time

dtArray = [0.001, 0.005, 0.01, 0.05, 0.1]
ErrorArray = []


for dt in dtArray:
    x=np.array([1,-1,2])
    t=0
    n=0
    nsteps=tmax/ dt # calculate number of steps

    while n < nsteps:
        x=ts.step_rk2(x,dt,b,r,sigma)

        t = t+dt
        n = n+1
```

```
    Error = np.sqrt((x[0] - (1/4*np.e+3/4*np.e**(-3)))**2 + (x[1] -
    (1/2*np.e-3/2*np.e**(-3)))**2 + (x[2]-2*np.e**(-1))**2)
    ErrorArray.append(Error)

print(dtArray)
print(ErrorArray)

fig=plt.figure()
ax = fig.add_subplot(111)


plt.xlabel(r'$Log(\Delta t)$')
plt.ylabel(r'$Log (E)$')

plt.title('Plot of Error versus Time Increment using RK2')

LogdtArray = np.log10(dtArray)
LogErrorArray = np.log10(ErrorArray)

plt.plot(LogdtArray, LogErrorArray, '.', color = 'r')

LogdtArrayPrime = LogdtArray[:-1]
LogErrorArrayPrime = LogErrorArray[:-1]

Slope, Intercept = np.polyfit(LogdtArrayPrime, LogErrorArrayPrime, 1)

y_fit = np.exp(Slope * LogdtArray + Intercept)

plt.plot(LogdtArray, np.log(y_fit), ''-'')

blue_line = mlines.Line2D([], [], color='blue', marker='.', markersize=0,
label='Linear Regression Line')
plt.legend(handles=[blue_line])

print(Slope)

plt.show()
```

The time stepping procedures of RK2 and RK4 can be seen below.

```
def step_rk2(x,dt,b,r,sigma):
    #Advances the input array x at time t to the returned value x_out
    at time t+dt

    dxdt1 = calc_dxdt(x,b,r,sigma)
    xtemp1 = x+0.5*dt*dxdt1
    dxdt2 = calc_dxdt(xtemp1,b,r,sigma)
    x_out = x+dt*dxdt2

    return x_out

def step_rk4(x,dt,b,r,sigma):
    #Advances the input array x at time t to the returned value x_out
```

```
    at time t+dt

    dxdt1 = calc_dxdt(x,b,r,sigma)
    xtemp1 = x+0.5*dt*dxdt1
    dxdt2 = calc_dxdt(xtemp1,b,r,sigma)
    xtemp2 = x+0.5*dt*dxdt2
    dxdt3 = calc_dxdt(xtemp2,b,r,sigma)
    xtemp3 = x+dt*dxdt3
    dxdt4 = calc_dxdt(xtemp3,b,r,sigma)

    x_out = x+dt*(dxdt1+2*(dxdt2+dxdt3)+dxdt4)/6

    return x_out

def calc_dxdt(x,b,r,sigma):
    #Calculates the vector dx/dt for the Lorenz Equations

    dxdt = 0*x
    dxdt[0] = x[1]-x[0]
    dxdt[1] = 4*x[0]-x[1]
    dxdt[2] = -x[2]

    return dxdt
```

A.5 Poincaré Sections

This code plots the 3D trajectories with the Poincaré plane as well as the Poincaré sections seen in section 6.1.

```
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts
from mpl_toolkits.mplot3d import Axes3D



# parameters
b = 8/3
r = 200
sigma = 10

# time-stepping parameters
tmax = 100 # run to this time
nsteps = 30000 # number of time steps
dt=tmax/ nsteps # calculate the time step

# initial conditions (x Array: 3 position vars)
t=0

x=np.array([1,0,0])

# number of timesteps taken; initialise to 0
n=0

xValues = [x[0]]
yValues = [x[1]]
zValues = [x[2]]
```

```
tValues = [t]

PoincarexUpValues = []
PoincareyUpValues = []

PoincarexDownValues = []
PoincareyDownValues = []

PoincarePlane = r-1


while n < nsteps:
    xOld = x[0]
    yOld = x[1]
    zOld = x[2]

    x=ts.step_rk4(x,dt,b,r,sigma)

    xValues = np.append(xValues, x[0])
    yValues = np.append(yValues, x[1])
    zValues = np.append(zValues, x[2])
    tValues = np.append(tValues, t)

    if (zOld - PoincarePlane)*(x[2]-PoincarePlane) < 0:

        #Interpolate
        a = (PoincarePlane - zOld)/(x[2] - zOld)
        PCxValue = xOld + a*(x[0] - xOld)
        PCyValue = yOld + a*(x[1] - yOld)

        if (zOld - x[2]) < 0:
            PoincarexUpValues.append(PCxValue)
            PoincareyUpValues.append(PCyValue)
        else:
            PoincarexDownValues.append(PCxValue)
            PoincareyDownValues.append(PCyValue)

    t = t+dt
    n = n+1

plt.figure(1)

PoincarexUpValues = PoincarexUpValues[10:]
PoincareyUpValues = PoincareyUpValues[10:]
PoincarexDownValues = PoincarexDownValues[10:]
PoincareyDownValues = PoincareyDownValues[10:]


plt.plot(PoincarexUpValues,PoincareyUpValues,'r.', markersize=2)
plt.plot(PoincarexDownValues,PoincareyDownValues,'b.', markersize=2)

plt.title(r'Poincare section with the plane $z = r - 1$')
plt.xlabel('x')
```

```
plt.ylabel('y')


plt.show()

fig = plt.figure(2)
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x Axis')
ax.set_ylabel('y Axis')
ax.set_zlabel('z Axis')

ax.set_title(r'3D plot with trajectory and Poincar section')

limit = r

xs = np.linspace(-limit, limit, 10)
ys = np.linspace(-limit, limit, 10)
X, Y = np.meshgrid(xs, ys)
Z = np.zeros_like(X)


# Plot the surface
ax.plot(xValues, yValues, zValues, c= 'red')
ax.plot_surface(X,Y,Z+(PoincarePlane), alpha = 0.3)


plt.show()
```

This code calculates and plots the maximal x and y values for varying value of r.

```
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts,
matplotlib.lines as mlines


rVals = [5,10,15,20,25,30,35,40]

xMaxValArray = []
yMaxValArray = []

for i in rVals:


    # parameters
    r = i
    b = 8/3
    sigma = 10

    # time-stepping parameters
    tmax = 100 # run to this time
    nsteps = 30000 # number of time steps
    dt=tmax/ nsteps # calculate the time step

    # initial conditions (x Array: 3 position vars)
    t=0
```

```
x=np.array([-5,16,27])

# number of timesteps taken; initialise to 0
n=0
xValues = [x[0]]
yValues = [x[1]]
zValues = [x[2]]
tValues = [t]

PoincarexUpValues = []
PoincareyUpValues = []

PoincarexDownValues = []
PoincareyDownValues = []

PoincarePlane = r-1


while n < nsteps:
    xOld = x[0]
    yOld = x[1]
    zOld = x[2]
    x=ts.step_rk4(x,dt,b,r,sigma)

    xValues = np.append(xValues, x[0])
    yValues = np.append(yValues, x[1])
    zValues = np.append(zValues, x[2])
    tValues = np.append(tValues, t)

    if (zOld - PoincarePlane)*(x[2]-PoincarePlane) < 0:

        #Interpolate
        a = (PoincarePlane - zOld)/(x[2] - zOld)
        PCxValue = xOld + a*(x[0] - xOld)
        PCyValue = yOld + a*(x[1] - yOld)

        if (zOld - x[2]) < 0:
            PoincarexUpValues.append(PCxValue)
            PoincareyUpValues.append(PCyValue)
        else:
            PoincarexDownValues.append(PCxValue)
            PoincareyDownValues.append(PCyValue)

    t = t+dt
    n = n+1



PoincarexUpValues = PoincarexUpValues[10:]
PoincareyUpValues = PoincareyUpValues[10:]
PoincarexDownValues = PoincarexDownValues[10:]
PoincareyDownValues = PoincareyDownValues[10:]
```

```
    xMaxVal = max(max(PoincarexUpValues), max(PoincarexDownValues))
    yMaxVal = max(max(PoincareyUpValues), max(PoincareyDownValues))

    xMaxValArray.append(xMaxVal)
    yMaxValArray.append(yMaxVal)



plt.figure(1)
plt.plot(rVals, xMaxValArray, color = 'r')
plt.plot(rVals, yMaxValArray, color = 'b')

plt.ylim(-10,43)

plt.title(r'Plot of max $x$ and $y$ values against $r$')
plt.xlabel(r'$r$')
plt.ylabel('Max Values')

red_line = mlines.Line2D([], [], color='red', marker='.', markersize=0,
label='x Max Values')
blue_line = mlines.Line2D([], [], color='blue', marker='.', markersize=0,
label='y Max Values')


plt.legend(handles=[red_line, blue_line])
```

A.6 Lorenz Map

This code plots the Lorenz map as found in section 6.2.

```
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts

title_font = {'fontname':'Arial', 'size':'15'}
axis_font = {'fontname':'Arial', 'size':'15'}

# parameters
b = 8/3
r = 28
sigma = 10

# time-stepping parameters
tmax = 305 # run to this time
nsteps = 30501 # number of time steps
dt=tmax/ nsteps # calculate the time step

# initial conditions (x Array: 3 position vars)
t=0

x=np.array([1,1,1])

# number of timesteps taken; initialise to 0
n=0

xValues = [x[0]]
```

```
yValues = [x[1]]
zValues = [x[2]]
tValues = [t]

maxValuesList = []
plotListXValues = []
plotListYValues = []


while n < nsteps:
    x=ts.step_rk4(x,dt,b,r,sigma)

    xValues = np.append(xValues, x[0])
    yValues = np.append(yValues, x[1])
    zValues = np.append(zValues, x[2])
    tValues = np.append(tValues, t)

    t = t+dt
    n = n+1

xValues = xValues[500:]
yValues = yValues[500:]
zValues = zValues[500:]

#Determines whether the middle point is larger than the two beside it
def maxFinder(left, middle, right):
    if (middle > left) and (middle > right):
        return True
    else:
        return False


#Finds all such max values
for i in range(1, 30000):
    if maxFinder(zValues[i-1], zValues[i], zValues[i+1]) == True:
        maxValuesList.append(zValues[i])


#Creates list of X and Y values
maxValuesListLength = len(maxValuesList)

for i in range(1, maxValuesListLength):

    plotListXValues.append(maxValuesList[i-1])
    plotListYValues.append(maxValuesList[i])


LineXValues = [30,48]
LineYValues = [30,48]


#Plots
plt.figure(1)
```

```
plt.plot(plotListXValues,plotListYValues, 'bo')
plt.plot(LineXValues,LineYValues, 'k-')

plt.xlim(30,48)
plt.ylim(30,48)


plt.title('The Lorenz Map', **title_font)
plt.xlabel(r'$z_{n}$', **axis_font)
plt.ylabel(r'$z_{n+1}$', **axis_font)



numbers = []

for i in range(0,100):
    numbers.append(i)

del maxValuesList[100:]

plt.figure(2)
plt.plot(numbers,maxValuesList, 'go-')

plt.title('Maxima Z Values Against Time', **title_font)
plt.xlabel(r'$t$', **axis_font)
plt.ylabel(r'$z_{max}$', **axis_font)
```

This code plots the Tent map.

```
import numpy as np , matplotlib.pyplot as plt


title_font = {'fontname':'Arial', 'size':'15'}
axis_font = {'fontname':'Arial', 'size':'15'}



MapXValues = [0,0.5,1]
MapYValues = [0,1,0]

LineXValues = [0,1]
LineYValues = [0,1]

#Plots
plt.figure(3)
plt.plot(MapXValues,MapYValues, 'b-')
plt.plot(LineXValues,LineYValues, 'k-')

plt.xlim(0,1)
plt.ylim(0,1)


plt.title('The Tent Map', **title_font)
plt.xlabel(r'$x_{n}$', **axis_font)
plt.ylabel(r'$x_{n+1}$', **axis_font)
```

This code draws the Tent Map and cobweb diagrams as discussed in section 6.2.

```
import numpy as np, matplotlib.pyplot as plt

#Initial Variables
n = 20
xinit = 0.151

#Map
def TentMap(x):
    return min(2*x,2*(1-x))

#Draw Map and line y=x
MapValues = []
xValues = np.linspace(0,1,num = 1000)

for i in xValues:
    MapValues.append(TentMap(i))

plt.figure(1)
plt.plot(xValues,MapValues, c= 'red')
plt.plot([0,1], [0,1] , 'k')

plt.plot([xinit,xinit],[0,xinit],'b')

#Draw cobweb trajectories
x = xinit

for i in range(n):
    y = TentMap(x)
    plt.plot([x,x], [x,y], 'b')
    plt.plot([x,y], [y,y], 'b')
    x = y

plt.xlabel('x')
plt.ylabel('y')
plt.title('Cobweb Diagram for the Tent Map')

plt.show()
```

A.7 Large r

This section plots trajectories for the large values of r in each spacial direction.

```
# time steps the Lorenz equations
import numpy as np , matplotlib.pyplot as plt , TimesteppingRK4 as ts

# parameters
b = 8/3
r = 28
sigma = 10

dtArray = [0.01, 0.005, 0.002, 0.001, 0.0005]

# time-stepping parameters
```

```
tmax = 70 # run to this time


for dt in dtArray:

    # initial conditions (x Array: 3 position vars)
    t=0

    x=np.array([1,0,0])

    # number of timesteps taken; initialise to 0
    n=0

    xValues = [x[0]]
    yValues = [x[1]]
    zValues = [x[2]]
    tValues = [t]


    nsteps = tmax/dt
    while n < nsteps:
        x=ts.step_rk4(x,dt,b,r,sigma)

        xValues = np.append(xValues, x[0])
        yValues = np.append(yValues, x[1])
        zValues = np.append(zValues, x[2])
        tValues = np.append(tValues, t)


        t = t+dt
        n = n+1

    print(t, x[0], x[1], x[2])

#3D Plot
plt.figure(1)
ax.plot(xValues, yValues, zValues, c= 'red')

plt.show()
plt.savefig('test.pdf')



#Axes plots against time
plt.figure(2)
plt.plot(tValues, xValues, 'r')
plt.plot(tValues, 0*tValues, 'black')
plt.ylim(-12,12)
plt.title(r'Plot of x Values versus Time for $r \rightarrow \infty$')
plt.xlabel('t')
plt.ylabel('x(t)')

plt.figure(3)
```

```
plt.plot(tValues, yValues, 'g')
plt.plot(tValues, 0*tValues, 'black')
plt.ylim(-12,12)
plt.title(r'Plot of y Values versus Time for $r \rightarrow \infty$')
plt.xlabel('t')
plt.ylabel('y(t)')

plt.figure(4)
plt.plot(tValues, zValues, 'b')
plt.plot(tValues, 0*tValues, 'black')
plt.ylim(-12,12)
plt.title(r'Plot of z Values versus Time for $r \rightarrow \infty$')
plt.xlabel('t')
plt.ylabel('z(t)')
```

References

[1] Encyclopaedia Britannica. [Online]. s.v. Edward Lorenz, 2009. [Accessed 15 October 2019]. Available from: https://www.britannica.com

[2] Sparrow, C. The Lorenz Equations: Bifurcations, Chaos and Strange Attractors. Applied Mathematical Sciences. 1982, **41**, pp.10-11.

[3] Fuchs, A. Nonlinear Dynamics in Complex Systems. Berlin: Springer, 2013.

[4] Strogatz, S. H., Nonlinear Dynamics and Chaos, Massachusetts, Perseus Books Publishing, 1994.

[5] Tucker, W., A Rigorous ODE Solver and Smale's 14th Problem, Foundations of Computational Mathematics, 2002, **2**(1), pp. 53-117.

[6] Mikhailov, A. Lyapunov Functions [Lecture notes accessed through Minerva]. MATH2391 Nonlinear Differential Equations. University of Leeds, 2018

[7] Press, W. H., Teukolsky, S. A.,Vetterling, W. T., and Flannery, B. P. Numerical Recipes: the art of scientific computing. 3rd ed. Cambridge: Cambridge University Press, 2007.

[8] Lorenz, E. N., Deterministic Nonperiodic Flow, Journal of the Atmospheric Sciences, 1963, **20**(2), pp. 139-140.

[9] Devaney, R. L., An Introduction to Chaotic Dynamical Systems. 2nd ed. Florida: CRC Press, 2018.

[10] Guo, J. Analysis of Chaotic Systems. [Online]. 2014. [Accessed 22 February 2019]. Available from: http://math.uchicago.edu

[11] Yorke, J.A., and Li, T. Period Three Implies Chaos, The American Mathematical Monthly, 1975, **82**(10), pp.985-992.

[12] Robbins, K. A., Periodic Solutions and Bifurcation Structure at High R In the Lorenz Model, SIAM Journal on Applied Mathematics, 1979, **36**(3), pp, 457-458