1-1-2011

Numerical Calculation of Lyapunov Exponents for Three-Dimensional Systems of Ordinary Differential Equations

Clyde-Emmanuel Estorninho Meador
cemeador@gmail.com

NUMERICAL CALCULATION OF LYAPUNOV EXPONENTS FOR THREE-DIMENSIONAL SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS

A Thesis submitted to
the Graduate College of
Marshall University

In partial fulfillment of
the requirements for the degree of
MASTER OF ARTS

MATHEMATICS

by

CLYDE-EMMANUEL ESTORNINHO MEADOR

Approved by

Dr. Anna Mummert, Committee Chairperson
Dr. Scott Sarra
Dr. Judith Silver

Marshall University
2011

ACKNOWLEDGMENTS

Foremost I would like to express my gratitude to my thesis mentor, Dr. Anna Mummert. Her continual advice, guidance and support were a tremendous help, as was her ability to clearly explain the most obtuse of concepts. I could not imagine a better mentor or advisor for a mathematician.

I am also grateful to the other members of my committee, Dr. Scott Sarra and Dr. Judith Silver. Their advice and feedback were wonderful. Dr. Sarra was always ready to share his knowledge of numerical analysis and was always available for questions about minute details of Runge-Kutta methods.

The leadership of the department by Dr. Ralph Oberste-Vorth, Dr. Alfred Akinsete, and Dr. Bonita Lawrence was impeccable and everything a graduate student could ask for. Stacy Good kept the department running smoothly and assisted me in many different ways. Dr. Carl Mummert and Dr. Clayton Brooks helped me to keep my teaching and other duties on track while working on this thesis, and I am indebted to them for their advice and encouragement.

This thesis would not have been completed without the moral support provided by family and friends, and I am very blessed to have them in my life.

TABLE OF CONTENTS

PAGE

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Numerical calculation of Lyapunov exponents for three-dimensional systems of ordinary differential equations

Clyde-Emmanuel Estorninho Meador

We consider two algorithms for the computation of Lyapunov exponents for systems of ordinary differential equations: orbit separation and continuous Gram-Schmidt orthonormalization. We also consider two Runge-Kutta methods for the solution of ordinary differential equations. These algorithms and methods are applied to four three-dimensional systems of ordinary differential equations, and the results are discussed.

1. CHAOS

Chaos as a branch of mathematics is widely recognized today, thanks to the difficulties of weather prediction and to the popularization of the concept in films such as *The Butterfly Effect* [2] and books such as *Chaos* [9]. One characteristic of chaotic equations is sensitive dependence on initial conditions.

Definition 1.1. *Sensitive dependence on initial conditions*: tiny differences in the initial conditions of the system will lead to large differences in the solutions.

This concept means that initially tiny errors will grow over time and eventually make numerical solutions worthless. Inevitable errors in the floating point representation of numbers will grow over time and the simulation will become inaccurate.

There is no universally accepted definition for chaos although most definitions include sensitive dependence on initial conditions. Some authors [1] only require sensitive dependence on initial conditions, whereas others [8], [23] require several additional characteristics. This thesis studies the *Lyapunov exponents* of a system, which measure the separation of solutions based on tiny differences in initial conditions. Lyapunov exponents are a way to numerically study whether a system has sensitive dependence on initial conditions. Computing Lyapunov exponents allows us to determine whether a system is chaotic using the definition of Alligood *et al.* [1], which is presented in this section as definition 1.4.

Before we discuss Lyapunov exponents and different methods for their computation, there are some basic definitions to be familiar with. A *dynamical system* is a system of equations which changes over time. It may consist of either discrete difference equations or continuous differential equations. Dynamical systems model real-world phenomena such as disease spread and weather patterns. Interested readers can go to [19], [24]. Difference equations are called *maps*, and differential equations are called *flows*. *Trajectory* and *orbit*

describe the evolution of these dynamical systems. The trajectory of a flow is the path the flow takes as time progresses. An orbit is a set of points that a map moves through under iteration. Based on a vector of initial conditions v_0 and a dynamical system F with numerical solution $f_t(v_i)$ at time t_i, $t_0 \leq t_i \leq t_{Final}$, the orbit O_F is a set given by

$$O_F = \{f_t(v_0) : t \in [t_0, t_{Final}]\} \tag{1.1}$$

Trajectories work just as well with this notation.

The dynamical systems we will be studying are all *nonlinear*; they contain variables to higher powers than one or a product of variables (examples: x^2, xy).

As time progresses, orbits and trajectories may approach a single point, remain in a specific bounded region, or approach infinity. For example, the difference equation

$$x_{n+1} = x_n^2 \tag{1.2}$$

grows without bound for $x_0 = 2$; but with initial condition $x_0 = 0.5$ it approaches 0, remaining between 0 and 0.5 at all times. A *fixed point* is a point x_i that maps to itself under a particular function f. In notation, this may be succinctly stated as $f(x_i) = x_i$. The point $x = 0$ is an example of a fixed point for equation 1.2. *Periodic cycles* are a similar concept. Rather than a single point, a periodic cycle is a set of points that repeat themselves. In notation, x_0, x_1, ... ,x_n is a periodic cycle for F if $F(x_0) = x_1$, $F(x_1) = x_2$, ..., $F(x_n) = x_0$.

Strogatz [23] loosely defines an *attractor* as a set to which all neighboring trajectories (or orbits) converge, an intuitively appealing definition. The point $x = 0$ is an attractor for equation 1.2. A *strange attractor* is a set that differs from the simpler fixed points or periodic cycles. Strange attractors often show as visually compelling sets when trajectories or orbits are plotted. The Lorenz attractor, seen in figure 1.1, is the classic example of a strange attractor. The Lorenz attractor demonstrates long-term *aperiodic* behavior: the

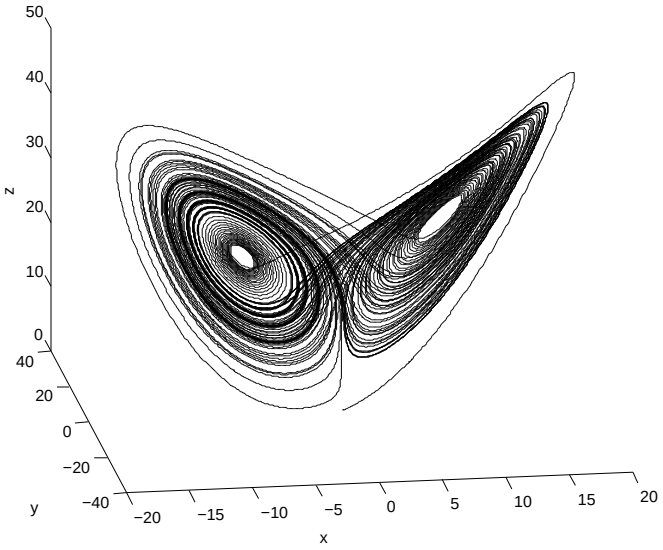

Figure 1.1. The Lorenz attractor with system parameters $\sigma = 10$, $\beta = \frac{8}{3}$, $\rho = 28$, initial conditions $[0.9, 0.9, 0]$. This system is chaotic.

orbit is bounded and does not settle down to a fixed point or periodic cycle. We present further details in section 2.1.

Strogatz [23] provides a more rigorous definition with three parts. A strange attractor is *invariant*, attracts an open set of initial conditions, and is *minimal*. An invariant set A is one in which any trajectory or orbit that begins in the set stays in the set. In notation, $f(A) \subset A$. By attracting an open set, we mean that there is an open set B containing the attractor A such that if an orbit or trajectory O_F begins in B, the distance from successive points in O_F to the set A approaches zero as time progresses. A minimal set is the largest set that satisfies the other conditions.

The concept of chaos itself receives varying definitions on a continuum of rigor. Strogatz's [23] definition of chaos for a system requires no additional terminology:

Definition 1.2. A dynamical system is *chaotic* if it displays long-term aperiodicity and sensitive dependence on initial conditions.

A system with a strange attractor and sensitive dependence on initial conditions is chaotic by Strogatz's definition.

Devaney's [8] definition of chaos for mappings is more rigorous, but requires a few more definitions. The *closure* of a set A is the union of A and the set of limit points of A. A subset B of set C is *dense in* C if C is the closure of B. A function g is *topologically transitive* if any pair of open sets A and B have a nonempty intersection after a finite number of iterations. This idea is written in notation as $g^k(A) \cap B$ for some $k \in \mathbb{N}$. Sensitive dependence on initial conditions for a mapping is defined by Devaney as the existence of a positive number δ so that for any point in the domain and any neighborhood about that point, there exists a point in that neighborhood so that after a finite number of iterations, the points will be separated by more than δ. This idea is seen again in figure 3.1 and is the basis for "orbit separation," the subject of section 6.1. Devaney's definition of chaos follows.

Definition 1.3. A mapping f on a set A is *chaotic* if it is topologically transitive, displays sensitive dependence on initial conditions, and periodic points of f are dense in A.

These rigorous definitions have the advantage of certainty (in knowing if the conditions are verified); but they are of limited use in potential applications where it is often not possible to analytically verify properties such as long-term aperiodicity.

For this thesis, we shall use the definition of Alligood *et al.* [1]:

Definition 1.4. A dynamical system is *chaotic* if it has a positive Lyapunov exponent.

Lyapunov exponents will be defined and discussed in section 3.

2. SYSTEMS OF DIFFERENTIAL EQUATIONS

Here we introduce the systems of equations studied in this thesis. Historical background is provided where available. Equations were chosen based on historical significance and availability, or dearth, of research in the literature. Some are relatively famous, such as the Lorenz and Rössler systems of equations. Chua's circuit is an application from physics, and the Rabinovich-Fabrikant system is a relatively obscure problem that is the subject of only a handful of papers.

2.1. THE LORENZ EQUATIONS

Edward Lorenz was a meteorologist who made several contributions to chaos theory. While manually entering data values to rerun a weather simulation, Lorenz rounded to three decimal places (rather than six) and observed sensitive dependence on initial conditions [22], [15]. He later simplified his model to the system of ordinary differential equations

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(\rho-z)-y \qquad \sigma,\rho,\beta \in \mathbb{R} \\ \dot{z} &= xy-\beta z \end{aligned} \tag{2.1}$$

with constant parameters σ, ρ, and β. A commonly used set of parameters is $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$, because these parameters present the famous "Lorenz attractor." (See figure 1.1).

These equations have been presumed chaotic for decades. In 1999, Tucker [25] proved the existence of a strange attractor for the system, indicating that the system is aperiodic. His results demonstrated that the famous phase plots of the system did reveal an attractor rather than an artifact. Along with sensitive dependence on initial conditions, the aperiodic

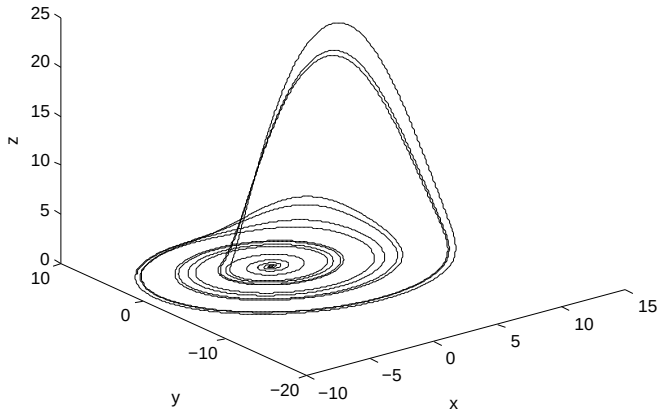

Figure 2.1. The Rössler attractor, for constant parameters $a = 0.2$, $b = 0.2$, $c = 5.7$, and initial conditions $[0.1, -0.1, 0.1]$. The system is chaotic.

nature of the Lorenz equations satisfies definition 1.2 of chaos. More details about the Lorenz equations are available in [23].

2.2. THE RÖSSLER EQUATIONS

Medical doctor Otto Rössler designed the following system in the 1970s.

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + ay \qquad\qquad a, b, c \in \mathbb{R} \\ \dot{z} &= b + z(x - c) \end{aligned} \tag{2.2}$$

The Rössler equations are simpler than the Lorenz equations. Though both are three-dimensional and share the same number of monomials in total, the Rössler equations have only one nonlinear term: the zx term in the $\dot{z}$ portion of equation 2.2. A set of parameters which presents chaos (in the sense of definition 1.4) is $a = 0.2$, $b = 0.2$, $c = 5.7$. Figure 2.1 shows the attractor based on these values.

2.3. THE RABINOVICH-FABRIKANT EQUATIONS

The Rabinovich-Fabrikant equations are

$$\begin{aligned} \dot{x} &= y(z - 1 + x^2) + bx \\ \dot{y} &= x(3z + 1 - x^2) + by \qquad\qquad a, b \in \mathbb{R} \\ \dot{z} &= -2z(a + xy) \end{aligned} \tag{2.3}$$

These equations were used by Rabinovich and Fabrikant to model waves in nonequilibrium substances. The equations are less well known than the Rössler and Lorenz equations, but they are an area of active research [16], [7], [6]. Mathematicians have used them to test numerical methods for ordinary differential equations [6], and different parameter values lead to very different phase portraits. These resemble strange attractors although not all are chaotic by definition 1.4. The differences in Lyapunov exponents for some of these sets of parameter values (and the attendant chaotic/nonchaotic nature of the system) are discussed in section 7.3.

We have included phase plots corresponding to various values of a and b; these can be found in figures 2.2, 2.3, 2.4, and 2.5. Figure 2.2 shows a phase plot for $a = 0.1$, $b = 0.98$. For these values, the system has a positive Lyapunov exponent and the attractor is saddle-shaped. With $a = 0.1$, $b = 0.2715$, the system is not chaotic. The phase plot in figure 2.3 shows a set that appears to be an attractor. Because the largest Lyapunov exponent is negative, this plot is more probably an artifact, a phantom caused by numerical errors. Another nonchaotic version of the system is presented by $a = 0.1$, $b = 0.5$, and figure 2.4 shows what appears to be an attracting fixed point at approximately $(x, y, z) = (1.07, -0.4, 0.07)$. Finally, $a = -1, b = -0.1$ also leads to chaos in the system. Figure 2.5 shows the attractor resulting from these settings.

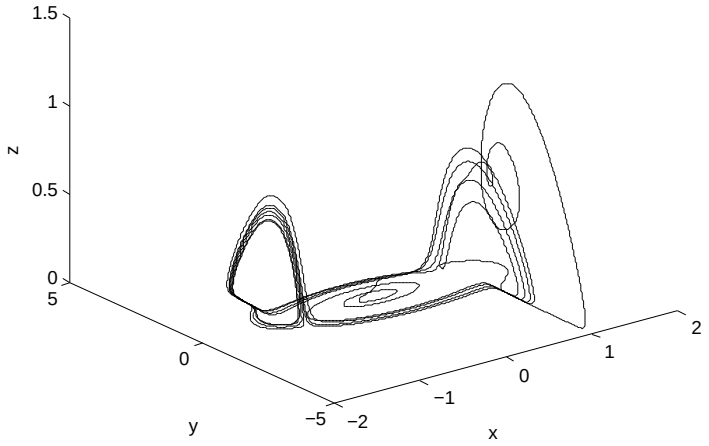

Figure 2.2. Phase plot of the Rabinovich-Fabrikant equations with $a = 0.1$, $b = 0.98$ and initial values $[0.1, 0.1, 0.1]$. The system is chaotic

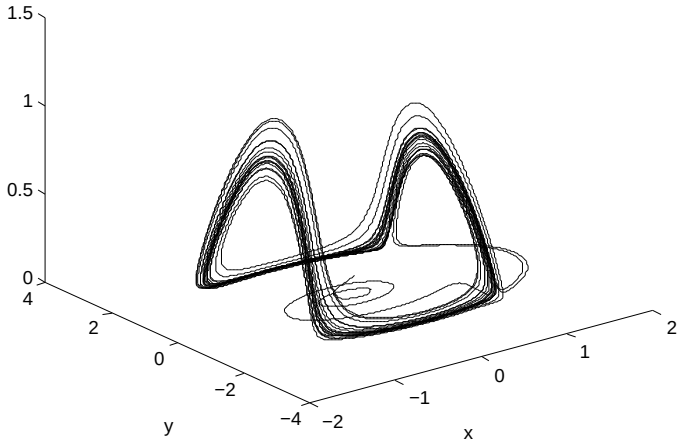

Figure 2.3. Phase plot of the Rabinovich-Fabrikant equations with $a = 0.1$, $b = 0.2715$ and initial values $[0.1, 0.1, 0.1]$. The system is not chaotic.

2.4. CHUA'S CIRCUIT

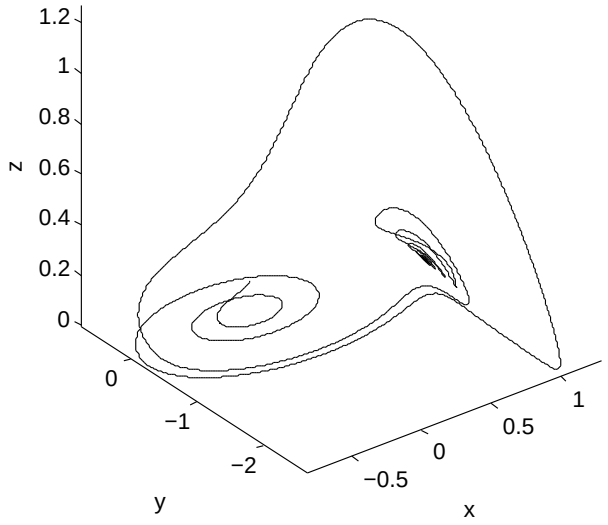

Figure 2.4. Phase plot of the Rabinovich-Fabrikant equations with $a = 0.1$, $b = 0.5$ and initial values $[0.1, 0.1, 0.1]$. The system is not chaotic.

Chua's circuit is given by the following equations.

$$\begin{aligned} \dot{x} &= \alpha(y - x - f(x)) \\ \dot{y} &= x - y + z \qquad\qquad \alpha, \beta \in \mathbb{R} \\ \dot{z} &= -\beta y \end{aligned} \tag{2.4}$$

The function $f(x)$ is defined by:

$$f(x) = \begin{cases} bx + a - b & \text{for } x \geq 1 \\ ax & \text{for } -1 \leq x \leq 1 \\ bx - a + b & \text{for } x \leq -1 \end{cases} \qquad a, b \in \mathbb{R} \tag{2.5}$$

This is a system of three ordinary differential equations that models an electrical circuit. Equation 2.5 is piecewise linear and continuous. For constant parameters $\alpha = 9$, $\beta = \frac{100}{7}$,

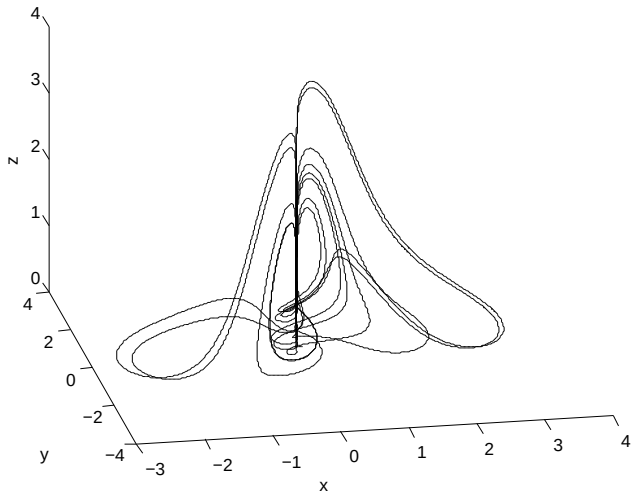

Figure 2.5. Phase plot of the Rabinovich-Fabrikant equations with $a = -1$, $b = -0.1$ and initial values $[0.1, 0.1, 0.1]$. The system is chaotic.

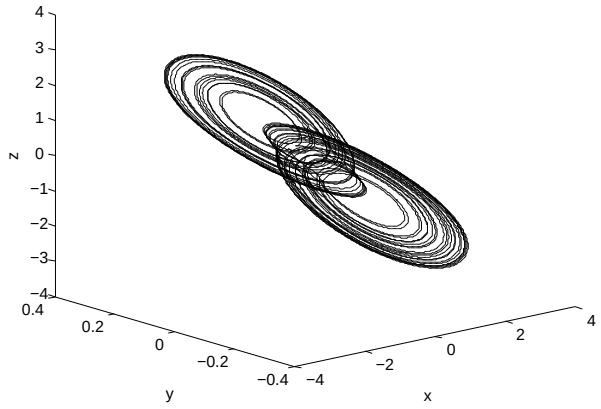

Figure 2.6. Chua's circuit attractor with constant parameters $\alpha = 9$,$\beta = \frac{100}{7}$,$a = \frac{-8}{7}$,$b = \frac{-5}{7}$, and initial values [0.1,0.1,0.1]. The system is chaotic.

$a = -\frac{8}{7}$, $b = -\frac{5}{7}$ there is an attractor that Matsumoto *et al.* refer to as a "double scroll" [17].

Figure 2.6 is a phase plot of the attractor. More details can be found in [17] and [22].

3. LYAPUNOV EXPONENTS

Lyapunov exponents of a system quantify the common component of all definitions of chaos mentioned here, sensitive dependence on initial conditions. They provide a measure of how two orbits that start from close initial conditions differ as time progresses. Two initially close orbits in a system with positive Lyapunov exponent will separate very quickly. After separation, the two numerical solutions grow more dissimilar until they are completely different. Figure 3.1 provides a visual example of divergent orbits. An orbit y is perturbed by ϵ. After one iteration, y_1 and the perturbed y_1^* are d_1 apart. A positive Lyapunov exponent will cause this separation to increase over further iterations. A system with all negative Lyapunov exponents will have an attracting fixed point or periodic cycle and will not present chaotic behavior. The phase plot in figure 2.4 appears to contain an attracting fixed point, and thus the corresponding system has only negative Lyapunov exponents.

A function that is infinitely differentiable is called *smooth*. (All of the differential equations in section 2 are smooth). Alligood *et al.* [1] present Lyapunov exponents of a smooth map f on $\mathbb{R}^m$ with the following definition:

Definition 3.1. For $k = 1, ..., m$, r_k^n is the length of the kth longest orthogonal axis of $J_n U$, where J_n is the Jacobian evaluated at the nth iteration of f and U is an orthogonal basis for $\mathbb{R}^m$.

The various r_k^n measure the change in a sphere of initial conditions near an orbit with specified initial condition vector x_0. The derivatives in the Jacobian measure infinitesimal change (separation), and this sphere is the $\mathbb{R}^m$ analog to the vertical distance example in figure 3.1. An example of the change in a sphere of initial conditions is given in figure 3.2.

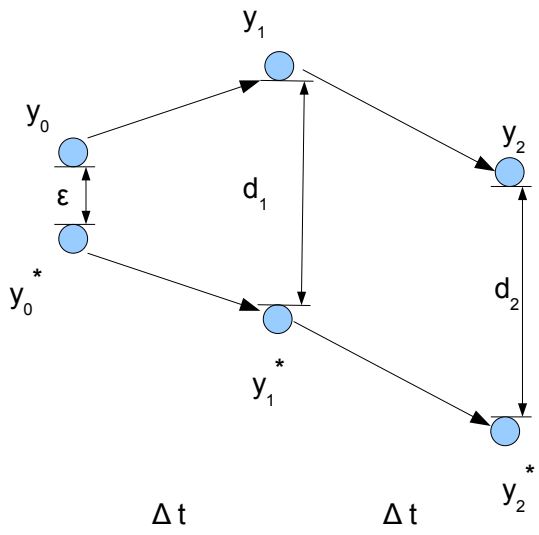

Figure 3.1. An illustration of the divergence of orbits. Although the orbits are initially separated by ϵ, as time progresses their separation grows. After two iterations, the orbits are d_1 apart. With additional iterations, the two orbits will diverge further.

We define the kth *Lyapunov number* of f to be

$$L_k = \lim_{n\to\infty} (r_k^n)^{\frac{1}{n}} \tag{3.1}$$

The kth *Lyapunov exponent* of f is

$$\lambda_k = \ln L_k \tag{3.2}$$

The *Lyapunov numbers* measure the average stretching or compressing per iteration along the axes. If a Lyapunov number for a system is greater than 1, then on that axis orbits separate from one another, and the system displays sensitive dependence on initial conditions. If a Lyapunov number is greater than 1, the corresponding Lyapunov exponent is positive.This leads us back to definition 1.4; a system is chaotic if it has a positive Lyapunov exponent.

To numerically compute Lyapunov exponents for a continuous dynamical system, first a numerical solution is calculated from an initial time to an final time using a fixed timestep.

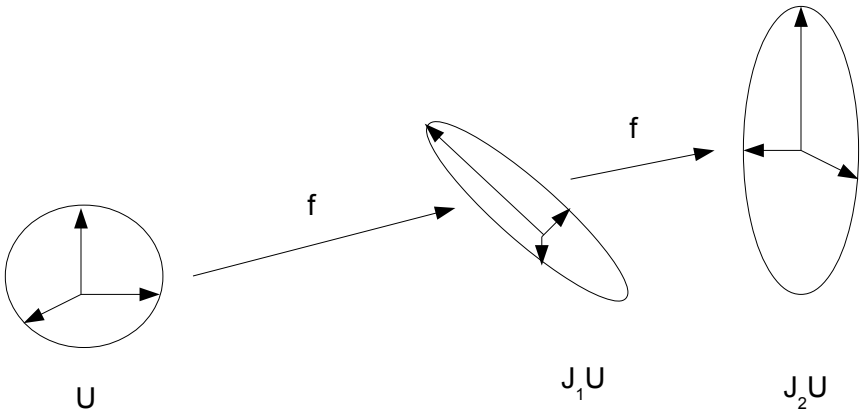

Figure 3.2. This figure illustrates change in a sphere of initial conditions under a map. Orthogonal axes corresponding to the r_k are shown.

Ideally a small timestep is used - the smaller the timestep, the greater the (theoretical) accuracy. (See section 4.3). The discretized numerical solution is the orbit of a map that displays the same behavior as the flow, so by computing its Lyapunov exponents, we may approximate the Lyapunov exponents of the flow by a variety of algorithms.

Numerical methods with changing timesteps exist, but we did not consider them suitable for several reasons. The algorithms used to compute Lyapunov exponents require values uniformly separated in time. Although it is possible to cherry-pick uniformly separated solution values from a changing timestep method, this process adds computational complexity. The change in timestep over time will vary from problem to problem, so it would be necessary to either filter the data for uniformly separated values or force the method to solve at particular time-values. The second and more pressing reason we avoided adaptive timestep methods is that it has been demonstrated that the Rabinovich-Fabrikant equations can present qualitatively different phase plots for adaptive timestep methods [18], even with identical initial values and constant parameters. Time-step independent solutions (at least, independent to

a finite precision) can be computed; but they require extended precision arithmetic [21], which is undesirable for this project due to the large increase in computing time. For these reasons, we have restricted ourselves to fixed timestep methods.

Algorithm	System	Lyapunov Exponents	Source
OS	Lorenz	0.9056 0 -14.5723	Sprott [22]
OS	Rössler	0.0714 0 -5.3943	Sprott [22]
OS	Chua	0.3271 0 -2.5197	Sprott [22]
Other	Chua	0.23 0 -1.78	Matsumoto *et al.* [17]

Table 3.1. Computed Lyapunov exponents for the Lorenz, Rössler, and Chua equations.

4. NUMERICAL METHODS

The numerical calculation of Lyapunov exponents requires the numerical solution of systems of ordinary differential equations (abbreviated as ODEs). This topic is an interesting and complicated subject in its own right, so we present in this section a discussion of some of the properties of these methods. The properties we will discuss affect the accuracy, reliability, and efficiency of implementation of numerical methods for the solution of ODEs; so they are used to distinguish between methods.

Many systems of ordinary differential equations do not have a closed-form solution. So, researchers use numerical methods for the solution of ODEs to find very accurate approximate solutions. There are many different methods that can provide a numerical solution for a system of ODEs, starting from a set of initial conditions. These methods vary in accuracy (represented by order), stability (resistance to explosive error propagation), and simplicity (ease and efficiency of implementation). The selection of a method is made by balancing desirable method properties with the computing resources at hand. Effective methods may be implemented on personal computers, although they sometimes require significant computing time.

For a discussion of the reliability of a numerical method (stability) and the difficulty of a problem (stiffness) we follow the presentation in Iserles [13] and recommend it for further details and references.

4.1. STABILITY

The stability of a numerical method is essentially its resistance to explosive error propagation. When a method becomes unstable, tiny computational errors compound and the numerical solution quickly approaches infinity. The stability properties of a numerical method

depend on the method rather than the particular ODE being studied; so they are always measured in the same way. The stability of a numerical method is described by a region in the complex plane called the stability region. The stability region of a numerical method is determined by the initial value problem

$$\dot{x} = zx \qquad z \in \mathbb{C}, \quad t \geq 0, \quad x_0 = 1 \tag{4.1}$$

that has the analytical solution

$$x(z,t) = e^{zt} \tag{4.2}$$

Definition 4.1. The *stability region* is the set of z-values for which bounded end-behavior in the actual solution is also present in the numerical solution. This set causes the numerical solution $\hat{x}_n(z,t)$ of equation 4.1 to satisfy

$$\lim_{n\to\infty} \hat{x}_n(z,t) = 0 \tag{4.3}$$

For $z \in \mathbb{C}$ with negative real part, the limit of equation 4.2 is zero as $t \to \infty$. For z with positive real part, the limit is infinity. Complex numbers z with negative real part that preserve this bounded behavior lie in the stability region.

A desirable property for a numerical method is to contain the entire left half of the complex plane in its stability region because the limit of the analytical solution $x(z,t)$ of equation 4.1 is zero exactly when the real part of z is negative [13]. This property is called *absolute stability*, and only occurs with implicit methods, a type of numerical method that is discussed in section 5.

Definition 4.2. Let λ_i be the eigenvalues of the Jacobian matrix for a system of ODEs. The numerical solution is *stable* if $\lambda_i h$ lie in the stability region of the numerical method where h is the step size.

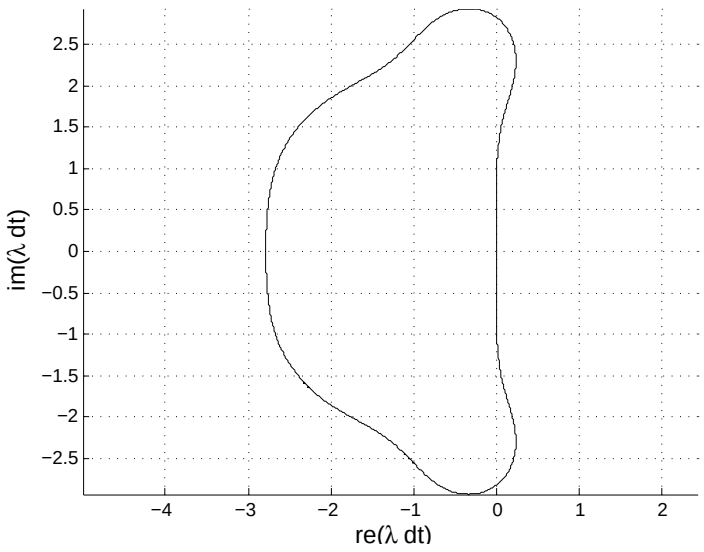

Figure 4.1. Stability region for RK4. For initial values in this region, bounded behavior is preserved in the numerical solution of equation 4.1.

The stability of a numerical solution depends on both the numerical method and on the system of ODEs because the stability region depends on the numerical method and the eigenvalues of the Jacobian depend on the system of ODEs. In practice, the solution is considered stable as long as the solution remains bounded as instability manifests as an unbounded solution.

It is important to note that this idea of stability is based on a linear ODE, and the translation from a linear problem with an exact solution to a nonlinear problem without an exact solution is not perfect [13]. Despite this fact, linear stability is the criterion generally used to describe the reliability of a given numerical method. Figure 4.1 presents the stability region for the fourth-order Runge-Kutta method used in this thesis. (This method is abbreviated throughout as "RK4"). Note that RK4 does not have absolute stability.

4.2. STIFFNESS

System	Parameters	t_0	h	t_{End}	Trials	Stiffness Ratio
Lorenz	$\sigma = 10$, $\beta = \frac{8}{3}$, $\rho = 28$	0	0.01	1000	1000	837530
Rössler	$a = 0.1$ $b = 0.2$, $c = 5.7$	0	0.01	1000	1000	2094804
RF	$a = 0.1$ $b = 0.5$	0	0.01	1000	1000	220912
Chua	$\alpha = 9$ $\beta = \frac{100}{7}$ $a = -\frac{8}{7}$ $b = -\frac{5}{7}$					2.4861

Table 4.1. Average stiffness ratios for the Lorenz, Rössler, and Rabinovich-Fabrikant equations based on 1000 sets of random initial conditions and for the Chua equations based on its constant Jacobian matrix.

Stiffness is a characteristic of systems of equations rather than of numerical methods. However, because it has a powerful effect on numerical results it is taken into consideration when selecting numerical methods for a particular problem. Stiffness is a property that describes the difficulty of numerically solving that system. A commonly used informal definiton is that, when using a numerical method, a stable numerical solution of a *stiff* system requires a smaller step-size h. By a stable solution, we mean one that has remained bounded.

Stiffness is often measured by the *stiffness ratio*, a fraction composed of the largest and smallest eigenvalues of the Jacobian matrix. Computations involving both very large and very small numbers are more error-prone in floating point arithmetic.

Table 4.1 provides calculated stiffness ratios for the systems of equations studied in this thesis. For the Lorenz, Rössler, and Rabinovich-Fabrikant systems, we used the 4th-order Runge-Kutta method from section 5.1 with uniform stepsize $h = 0.01$, a final time value $t_{End} = 1000$, and averaged the results from 1000 random sets of initial conditions. Of particular interest in this table is the high stiffness ratio of the Rössler system of equations, which is a simpler system than the Lorenz equations (as described in section 2). Despite

this, its stiffness ratio is a full order of magnitude higher than that of the Lorenz equations. Chua's circuit has two possible Jacobian matrices that are both constant, leading to the result of 2.4861 for its stiffness ratio.

Stiffness is a useful concept, but currently it is only a qualititative one. Formalizations such as stiffness ratios are an attempt to quantify the relative difficulty of computing a numerical solution for a given system of equations. The easiest way to tell if you are working with a stiff system is just to carry out the numerical computations; if your numerical solution becomes unstable, the ordinary differential equations you are working with are probably stiff.

4.3. ACCURACY

The accuracy of a numerical method is described by its *order*. A numerical ODE solver is of order ρ if the error of its numerical solution $\hat{x}$ is proportional to $(h)^\rho$. In other words, the difference $x(t_{n+1}) - \hat{x}(x(t_n))$ satisfies

$$x(t_{n+1}) - \hat{x}(x(t_n)) = \mathbf{O}(h^\rho) \tag{4.4}$$

where $\mathbf{O}(h^\rho)$ indicates that the difference is proportional to (h^ρ). What this means is that the error of a method of order ρ is proportional to the stepsize to the power ρ, which will be a very small number if $h < 1$.

The order of a method is generally derived alongside the method itself. The difference in accuracy between a 4th-order method like RK4 (section 5.1) and a 1st-order method like Euler's method makes the higher-order method very desirable. For example, with a stepsize of $h = 10^{-2}$, the error in a solution found with Euler's method is proportional to 10^{-2} whereas the error in a solution found with a 4th-order Runge-Kutta method is proportional to 10^{-8}. The same comparison can be made between the 4th-order method (RK4) in section 5.1 and the 8th order method (IRK8) in section 5.3 used in this thesis.

However, the computational cost of numerical methods is another important consideration. While the 8th order method will give a more accurate result than the 4th order method, it is also more difficult to code and requires significantly more computational time. When implementing the two algorithms for Lyapunov exponent computation (section 6), we selected our methods based on computational cost. We used both methods with the faster algorithm in section 6.1, but when implementing the more computationally intensive algorithm in section 6.2, we restricted ourselves to the faster 4th-order method.

A *convergence plot* is a figure that can be used to provide experimental verification of order. This plots the difference between the numerical solution of $\dot{x} = -x$ and the actual solution e^{-t} as a function of the stepsize d.

$$x(t_{n+1}) - \hat{x}(x(t_n)) \approx Ch^{\rho}. \tag{4.5}$$

If we take the logarithm of both sides, we get

$$\log\left(x(t_{n+1}) - \hat{x}(x(t_n))\right) \approx \rho \log h + \log C \tag{4.6}$$

On a plot with both axes on a logarithmic scale, this is a straight line. An example is presented in figure 4.2, using the following method. Beginning with an exponential set of inputs, n =10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, place n evenly spaced points between 0 and 1. Then define the timestep h as $\frac{1}{n-1}$. Calculate the difference between the numerical and analytic solutions at $t = 1$ for each n, and plot the difference versus the timesteps on logarithmic axes. The slope of the line connecting the points on graph will be the order of the method. In figure 4.2, the slopes of the 1st-order and 4th-order lines are shown as expected.

A final concern is that numerical solutions of chaotic ODEs become less accurate as time progresses due to sensitive dependence on initial conditions. Floating-point numbers are stored in binary, and small rounding errors at the limits of machine precision will occur.

Once a rounding error has been made, sensitive dependence on initial conditions guarantees that the difference between the solution based on the rounding and the solution with no rounding will grow.

The reason to accept numerical solutions for the approximation of properties like Lyapunov exponents comes from developments in shadowing. A *shadow* is an exact trajectory for an ODE that remains very close to a numerical solution for a long time. Previous work [11], [12], guarantees that numerical solutions have shadows, so the mapping that a numerical method gives us is assumed to be a close-enough approximation of the true (but impossible to calculate) solution, especially with regard to the shape and behavior of the attractor. The Lyapunov exponent for the numerical solution is then considered a close approximate value to that of the true solution.

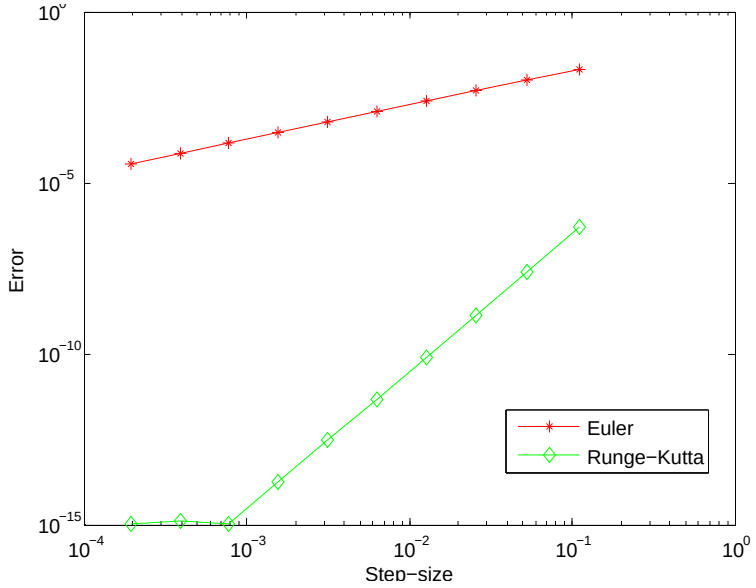

Figure 4.2. Convergence plots of Euler's method and RK4.

5. RUNGE-KUTTA METHODS

Runge-Kutta methods for the solution of ODEs are generalizations of Euler's method

$$y_{n+1} = y_n + hf(y_n, t_n).$$

They give more accurate results by taking more function evaluations within one timestep, although this does require more computational time to complete a computation. The timestep is split into v stages and the ODE is evaluated at each one. The stages are weighted and added to the previous value to get the numerical solution at the end of a timestep. A general statement of Runge-Kutta methods is given in [3], and may be summarized as follows.

An approximate solution at time $n+1$ given by a v-stage method is

$$\hat{y}_{n+1} = y_n + h\sum_{i=1}^{v} b_i k_i \tag{5.1}$$

where y_n is the numerical solution at the previous timestep, the b_i are constant coefficients, and the k_i are the "stages" where

$$k_i = f\left(y_n + h\sum_{j=1}^{v} a_{ij}k_i\right). \tag{5.2}$$

The a_{ij} are coefficients that place the subfunction evaluations throughout the timestep, and $i, j = 1, 2, ...v$. The b_i are weight coefficients on the stages k_i.

If a method's stage values k are given by explicit equations, it is an *explicit method*. If the stage values are given by implicit equations, it is an *implicit* method. For explicit methods, the stages k_i given by equation 5.2 can be computed immediately, and equation 5.1

c_1	a_{11}	$\cdots$	a_{1v}
c_2	a_{21}	$\cdots$	a_{2v}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_v	a_{v1}	$\cdots$	a_{vv}
	b_1	$\cdots$	b_v

Table 5.1. A general example of a Butcher table.

0	0
	1

Table 5.2. The Butcher table for Euler's method

can be solved with no further work. Implicit methods require the use of Newton's method,

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)},$$

to solve the nonlinear systems of equations for the stages k_i (a simple example of this is presented in section 5.2).

The customary way to list the coefficients of a Runge-Kutta method is via a *Butcher table* [4]. The general form of a Butcher table is given by table 5.1. Presenting the coefficients in this manner is both efficient and readable, so all numerical methods in this thesis will be presented this way. The c_i weight the time t within the stages k. As all of the ODEs in this thesis are autonomous, the c_i are not used here. Note that for any explicit method, the entries on and above the main diagonal of the a_{ij} are zero.

The simplest Runge-Kutta method, Euler's method has the Butcher table given in table 5.2.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table 5.3. The Butcher table for RK4

5.1. 4TH-ORDER EXPLICIT RUNGE-KUTTA METHOD

The popular 4th-order explict Runge-Kutta method (RK4) is

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= f(y_n) \\ k_2 &= f\left(y_n + \frac{1}{2}hk_1\right) \\ k_3 &= f\left(y_n + \frac{1}{2}hk_2\right) \\ k_4 &= f(y_n + hk_3) \end{aligned} \tag{5.3}$$

The Butcher table for this method is presented as table 5.3. RK4's popularity stems from the fact that it gives 4th-order accuracy in 4 stages while remaining easy to describe and implement. As stated in [4], explicit Runge-Kutta methods of order $o > 4$ cannot be performed in o stages. The derivation of this fact requires the use of rooted trees, and so it is omitted here.

5.2. EXAMPLE OF A 2-STAGE IMPLICIT RUNGE-KUTTA METHOD

If the coefficients a_{ij} for a Runge-Kutta method are not uniformly zero for $i \leq j$, the stages k_i will be implicit equations. If this is the case, the Runge-Kutta method is an implicit

method. Implementation of implicit methods is more difficult than that of explicit methods because the solution of the stages k_i requires the solution of a nonlinear system of implicit equations. This nonlinear system will be $v \times d$-dimensional, where v is the number of stages and d is the dimension of the system of differential equations. Implicit Runge-Kutta methods of order $o > 4$ do not necessarily suffer from the same stage limitation as explicit methods and may achieve order greater than the number of stages. Many examples of $v - stage$ implicit Runge-Kutta methods with order greater than v are given in [4].

We will demonstrate how to set up Newton's method for use with an implicit Runge-Kutta method with two stages. We apply a 2-stage implicit Runge-Kutta method

$$\begin{array}{c|cc} c_1 & a_{11} & a_{12} \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array}$$

to the ODE

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} x - \frac{1}{3}xy \\ \frac{1}{2}x + \frac{2}{3}xy \end{pmatrix}. \tag{5.4}$$

Because the method has 2 stages, to evaluate from t_n to t_{n+1}, the stage evaluations

$$k_i = \begin{bmatrix} k_i^{(1)} \\ k_i^{(2)} \end{bmatrix} \tag{5.5}$$

needed to go from $(\hat{x}_n, \hat{y}_n)$ to $(\hat{x}_{n+1}, \hat{y}_{n+1})$ are given by the equation

$$\begin{pmatrix} k_1^{(1)} \\ k_1^{(2)} \\ k_2^{(1)} \\ k_2^{(2)} \end{pmatrix} = \begin{pmatrix} \hat{x}_n + ha_{11}(k_1^{(1)} - \frac{1}{3}k_1^{(1)}k_1^{(2)}) + ha_{12}(k_2^{(1)} - \frac{1}{3}k_2^{(1)}k_2^{(2)}) \\ \hat{y}_n + ha_{11}(\frac{1}{2}k_1^{(2)} + \frac{2}{3}k_1^{(1)}k_1^{(2)}) + ha_{12}(\frac{1}{2}k_2^{(2)} + \frac{2}{3}k_2^{(1)}k_2^{(2)}) \\ \hat{x}_n + ha_{21}(k_1^{(1)} - \frac{1}{3}k_1^{(1)}k_1^{(2)}) + ha_{22}(k_2^{(1)} - \frac{1}{3}k_2^{(1)}k_2^{(2)}) \\ \hat{y}_n + ha_{21}(\frac{1}{2}k_1^{(2)} + \frac{2}{3}k_1^{(1)}k_1^{(2)}) + ha_{22}(\frac{1}{2}k_2^{(2)} + \frac{2}{3}k_2^{(1)}k_2^{(2)}) \end{pmatrix}.$$

The notation $k_i^{(j)}$ indicates the jth element of k_i. We subtract the right side to have an equation equal to zero and use Newton's method to find the k values. To do this, we must find the first derivative matrix M of the resulting equation. Because equation 5.4 has Jacobian matrix

$$J = \begin{bmatrix} 1 - \frac{1}{3}y & -\frac{1}{3}x \\ \frac{1}{2} + \frac{2}{3}y & \frac{2}{3}x \end{bmatrix}, \tag{5.6}$$

M is

$$\begin{bmatrix} 1 - ha_{11}J_{11}(k_1) & -ha_{11}J_{12}(k_1) & -ha_{12}J_{11}(k_2) & -ha_{12}J_{12}(k_2) \\ -ha_{11}J_{22}(k_1) & 1 - ha_{11}J_{21}(k_1) & -ha_{12}J_{22}(k_2) & -ha_{12}J_{21}(k_2) \\ -ha_{21}J_{11}(k_1) & -ha_{21}J_{12}(k_1) & 1 - ha_{22}J_{11}(k_2) & -ha_{22}J_{12}(k_2) \\ -ha_{21}J_{22}(k_1) & -ha_{21}J_{21}(k_1) & -ha_{22}J_{22}(k_2) & 1 - ha_{22}J_{21}(k_2) \end{bmatrix}.$$

The notation $J_{rc}(k_i)$ indicates the entry in row r, column c of the Jacobian matrix evaluated at k_i.

The use of Newton's method to find the stage values is an iterative process. We begin with an initial guess k_0 for the stage values, and then Newton's method refines these values. We used $k_0 = 0$ in line with Hairer and Wanner [10]. Implemented versions of Newton's method stop when one of two things happens: Either a specified maximum number of iterations is reached or two successive values for k are sufficently close to one another. Proceeding from k_i to k_{i+1} is a two-step process. First, solve the linear system

$$M(\Delta k_i) = -f(k_i) \tag{5.7}$$

for Δk_i. Then

$$k_{i+1} = k_i + \Delta k_i. \tag{5.8}$$

$\frac{1}{2}-\omega_2$	ω_1	$\omega_1'-\omega_3+\omega_4'$	$\omega_1'-\omega_3-\omega_4'$	$\omega_1-\omega_5$
$\frac{1}{2}-\omega_2'$	$\omega_1-\omega_3'+\omega_4$	ω_1'	$\omega_1'-\omega_5'$	$\omega_1-\omega_3'-\omega_4$
$\frac{1}{2}+\omega_2'$	$\omega_1+\omega_3'+\omega_4$	$\omega_1'+\omega_5'$	ω_1'	$\omega_1+\omega_3'-\omega_4$
$\frac{1}{2}-\omega_2$	$\omega_1+\omega_5$	$\omega_1'+\omega_3+\omega_4'$	$\omega_1'+\omega_3-\omega_4'$	ω_1
	$2\omega_1$	$2\omega_1'$	$2\omega_1'$	$2\omega_1$

Table 5.4. The Butcher table for IRK8. The ω coefficients are given in table 5.5. Consult [3] for more information.

$\omega_1=\frac{1}{8}-\frac{\sqrt{30}}{144}$	$\omega_2=\frac{1}{2}\sqrt{\frac{15+2\sqrt{30}}{35}}$	$\omega_3=\omega_2\left(\frac{1}{6}+\frac{\sqrt{30}}{24}\right)$
$\omega_4=\omega_2\left(\frac{1}{21}+\frac{5\sqrt{30}}{168}\right)$	$\omega_5=\omega_2-2\omega_3$	
$\omega_1'=\frac{1}{8}+\frac{\sqrt{30}}{144}$	$\omega_2'=\frac{1}{2}\sqrt{\frac{15-2\sqrt{30}}{35}}$	$\omega_3'=\omega_2'\left(\frac{1}{6}-\frac{\sqrt{30}}{24}\right)$
$\omega_4'=\omega_2'\left(\frac{1}{21}-\frac{5\sqrt{30}}{168}\right)$	$\omega_2'-2\omega_3'$	

Table 5.5. The coefficients ω used in table 5.4

5.3. 8TH-ORDER IMPLICIT RUNGE-KUTTA METHOD

The implicit method used in this thesis is a 4-stage method with order 8, abbreviated IRK8 . Table 5.4 is the Butcher table for this method. The coefficients ω are given in table 5.5.

IRK8 was originally developed by Butcher [3], and was implemented in a script using a modified Newton's method by Sarra [20]. IRK8 is a member of a particular class of methods called "Gauss" methods. Gauss methods are v stage implicit Runge-Kutta methods that achieve order $2v$ [4].

The use of Newton's method rather than fixed-point iteration (another technique for solving implicit nonlinear equations) to solve the implicit stage evaluations maintains stability of the method. Liniger and Willoughby [14] note that fixed-point iteration transforms

an implicit method into an explicit one, ruining the stability properties that make implicit methods desirable. The use of Newton's method avoids this problem.

The modifications to Newton's method save computational cost. The Jacobian is calculated once per timestep rather than being recalculated per iteration in Newton's method. There is also a change of variables that allows the use of vectorized operations and tensor products. These modifications are discussed in [10], but the key results follow. With A (the matrix of a_{ij} previously defined for Runge-Kutta methods) and b (a row vector containing the b_i coefficients) for an v-stage method, we define

$$(d_1, ..., d_v) = (b_1, ..., b_v)A^{-1}. \tag{5.9}$$

The invertibility of A may be well-established because A is a constant matrix.

In order to go from x_n to x_{n+1} our Runge-Kutta process becomes

$$x_{n+1} = x_n + \sum_{i=1}^{v} d_i z_i \tag{5.10}$$

where the z_i are defined by the implicit equation

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_v \end{pmatrix} = A \begin{pmatrix} hf(x_n + z_1) \\ hf(x_n + z_2) \\ \vdots \\ hf(x_n + z_v) \end{pmatrix} \tag{5.11}$$

The application of Newton's method to solve equation 5.11 is done in the same manner as in section 5.2. With the Jacobian approximation J and an appropriately-sized identity matrix I, the modified Newton's method for finding z_{k+1} from z_k is

$$\begin{aligned} (I - hA \otimes J)M &= -z_k + h(A \otimes I)f(z_k) \\ z_{k+1} &= z_k + M \end{aligned} \tag{5.12}$$

For a discussion of this modified Newton's method, consult [14].

6. ALGORITHMS FOR THE CALCULATION OF LYAPUNOV EXPONENTS

In this thesis, we use two different algorithms for the calculation of Lyapunov exponents. *Orbit separation* calculates the largest Lyapunov exponent only, whereas *continuous Gram-Schmidt orthnormalization* calculates all n Lyapunov exponents for an n-dimensional system of ordinary differential equations. MATLAB scripts used to implement these algorithms are included in the appendices.

6.1. ORBIT SEPARATION

Orbit separation (OS) is a method for computing the largest Lyapunov exponent for a system of ordinary differential equations. More details are presented by Sprott [22]. Two numerical orbits y and y^* are initialized so that $y_0 = y_0^*$. Then, y_0^* is perturbed by a small constant ϵ. This process is illustrated in figure 3.1. At this point, a numerical method with fixed step-size begins to compute a numerical solution. The distance separating the two numerical solutions y and y^* is calculated for some timestep and stored. Then y^* is changed so that it is again only ϵ from y but in a manner that preserves direction. At every timestep, the orbits are ϵ apart, so that we measure the effect of perturbing by *epsilon* throughout the orbit. If $c_k = y_k - y_k^*$ and $d_k = |c_k|$ for $\{k \in \mathbb{Z} : k \geq 1\}$, then the equation

$$y_k^* = y_k + \frac{\epsilon}{d_k} c_k \tag{6.1}$$

describes the normalization procedure that takes place every timestep. This process measures the average separation of orbits that are perturbed by a uniform ϵ. An illustration is provided in figure 6.1. The computation is based on a discrete numerical solution, so it is based on a mapping rather than a flow.

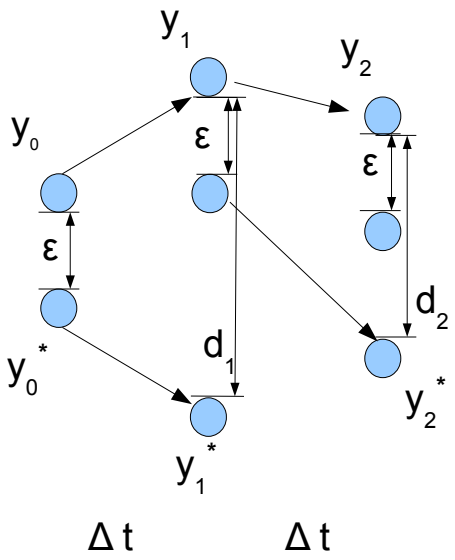

Figure 6.1. A visual example of orbit separation. Two orbits are uniformly perturbed and the changes in separation measured and stored.

As this process of numerical solution and normalization takes place, other calculations are made in order to compute the Lyapunov exponent. The first few points of the numerical orbits are not used in order to eliminate data that are not sufficiently close to the attractor. We wish to to analyze the separation of orbits after they begin exhibiting the aperiodic behavior. Orbits display this behavior after they reach the attractor, so data before an orbit reaches an attractor are not relevant. How many to discard is a judgment call: we discarded the first 2000 points from a set of 100000 total. Then, after every timestep the natural logarithm of the relative separation is computed as follows:

$$L_f = \log\left(\left|\frac{d_1}{\epsilon}\right|\right)$$

The sum of the L_f over time (denoted L_s) is initialized as zero and maintained every timestep as $L_s = L_s + L_f$. After n iterations at timestep h, the largest Lyapunov exponent for the orbit is:

$$\lambda_1 = \frac{L_s/n}{h}$$

In order to have a more representative number, this process is performed for a large number of random initial conditions (in this thesis, 1000), and the λ_1 are averaged to yield the calculated largest Lyapunov exponent for the system. Taking an average in this way gives us a result that should be close to most values.

Here is a summary of orbit separation:

- Initialize two orbits with identical initial conditions
- Perturb the second orbit by ϵ
- Apply a numerical method
- Every timestep, readjust the second orbit as in equation 6.1
- Every timestep, compute the natural logarithm of the relative separation
- Sum the logarithms and divide by the number of iterations
- Divide by the timestep to get the largest Lyapunov exponent

6.2. CONTINUOUS GRAM-SCHMIDT ORTHONORMALIZATION

Continuous Gram-Schmidt orthonormalization (CGSO) as developed by Christiansen and Rugh [5] is a method for computing all of the Lyapunov exponents of systems of equations. A new system of differential equations is developed in the following manner: Given a k-dimensional system $\dot{x} = f(x)$, the first k rows are the original system. Following these are

a set of k^2 rows that represent the change to an orthonormal basis of $\mathbb{R}^k$. The last k rows are the k computed Lyapunov exponents at the corresponding time. A numerical method then solves the system for a final time value.

Christiansen and Rugh's CGSO method depends on the use of orthonormal frames. An *orthonormal k-frame* is a set of of orthonormal vectors $e_1(t), e_2(t), ..., e_k(t)$ that form a basis for $\mathbb{R}^k$ and depend on time t. From here, two matrix definitions are required before we can present the augmented system of equations. The elements of the Jacobian matrix J may be represented in notation as an inner product

$$J_{ik} = \langle e_i, J(e_k) \rangle \tag{6.2}$$

The elements of the *stabilized matrix* L are defined as

$$L_{ii} = J_{ii} + \beta(\langle e_i, e_i \rangle - 1) \tag{6.3}$$

for the diagonal elements and as

$$L_{ik} = J_{ik} + J_{ki} + 2\beta\langle e_i, e_k \rangle \tag{6.4}$$

for the other elements.

At this point we may use equations 6.2, 6.3, 6.4, and $\dot{x} = f(x)$, a system of ordinary differential equations, to define the augmented system of differential equations

$$\begin{aligned} \dot{x} &= f(x) \\ \dot{e}_i &= J(e_i) - \sum_{h \leq i} e_h L_{hi} \qquad \{i \in \mathbb{Z}^+ | 1 \leq i \leq k\} \\ \dot{\Lambda}_i &= J_{ii} \end{aligned} \tag{6.5}$$

For the 3-dimensional ordinary differential equations in this thesis, we used an orthonormal 3-frame given by the traditional unit vectors

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

This 3-frame satisfies the initial condition for the e_i. Consequently, each vector in the frame is a 3×1 column vector, and the augmented system (6.5) is 15 rows. Numerical solution of this system gives approximate values for all three Lyapunov exponents for the systems of ordinary differential equations studied in this thesis.

In summary, orbit separation calculates sensitive dependence on initial conditions by perturbing an orbit y by ϵ and measuring the resulting separation at every timestep. The logarithms of the relative separations are averaged and then divided by the timestep to get the largest Lyapunov exponent. Continuous Gram-Schmidt orthonormalization uses orthonormal frames e_i and the stabilized matrix L to set up a system of ODEs that contain $\dot{\Lambda}_i$, the change in the i^{th}-largest Lyapunov exponents. Computing a numerical solution of this system to time t will give all of the Lyapunov exponents at time t.

Here is a summary of continuous Gram-Schmidt orthonormalization

- Create a new system of ordinary differential equations
- Part of this system continuously re-orthonormalizes a basis
- This part tracks the change in a sphere of initial conditions continuously
- The last rows of the new system measure the change in exponential separation of orbits
- To get the Lyapunov exponents, take the last rows at the final time value and divide by time

7. NUMERICAL RESULTS

In this section we present the results of our numerical experiments. Table 7.1 collects our computed Lyapunov exponents for the Lorenz equations organized by algorithm. It includes the values used for t_{End}, t_1, and the number of initial values. Table 7.2 lists the computed Lyapunov exponents for the Rössler equations and also includes the particular values used for t_{End}, t_1, and the number of initial values. Table 7.3 includes the computed results for the Rabinovich-Fabrikant equations along with the corresponding constant parameters a, b, the number of initial values, and the constants t_1 and t_{End}. Finally, table 7.4 presents a computed largest Lyapunov exponent for Chua's circuit as well as the associated number of initial values, t_1, and t_{End}. Computed Lyapunov exponents from the literature are included in table 3.1 in section 3. MATLAB scripts used to find this data are included in the appendices. It is worth mentioning again that both RK4 and IRK8 were used in tandem with OS, but with CGSO we used only RK4 to avoid prohibitively long computing time.

7.1. THE LORENZ EQUATIONS

Our study of the Lorenz equations had several purposes. We wanted to make sure that our newly implemented algorithms were properly written, delivering comparable results both to each other and to published results such as Sprott's [22]. Our orbit separation implementation returned a very close value to Sprott's published $\lambda_1 = .9056$, which is not surprising as Sprott is our source for the orbit separation algorithm.

We also used the Lorenz equations as a benchmark to experiment with settings such as t_{End} and we used a large number of orbits, from random initial conditions, to average the results. We concluded from these tests that 1000 random initial values are sufficient, as are $t_1 = 20$ and $t_{End} = 1000$. Table 7.1 collects the results of these experiments.

Algorithm	Numerical Method	t_1	t_{End}	Number of initial values	Lyapunov exponents
OS	RK4	20	1000	1000	0.905744684705866
OS	RK4	20	1000	2000	0.905761423985076
OS	IRK8	20	1000	1000	0.905758237263004
CGSO	RK4		1000	1000	0.895462506170606 0.002098916817564 -14.564258489126374
CGSO	RK4		500	1000	0.885311976076725 0.004195827397323 -14.556179060698607
CGSO	RK4		1000	500	0.885137292321675 .004284019108387 -14.556134884379532

Table 7.1. Computed Lyapunov exponents for the Lorenz equations with system parameter values $\sigma = 10$, $\beta = \frac{8}{3}$, $\rho = 28$ and method parameters $t_0 = 0$, $h = 0.01$

When using orbit separation (OS), doubling the number of orbits to 2000 had little effect. Computing Lyapunov exponents is a computationally expensive process. In practice, doubling the number of orbits doubles the time to complete the computation, so the close agreement between results for 1000 or 2000 initial values indicates that 1000 are sufficient. IRK8 and RK4 delivered comparable results to each other; so when studying other systems we either used both or chose one based on time constraints.

Our implementation of continuous Gram-Schmidt orthonormalization (CGSO) yielded results for λ_1 and λ_2 that did not correspond closely with our orbit separation results. However, our computed λ_3 was very close to the published value $\lambda_3 \approx -14.5723$ [22]. We also experimented with halving the number of orbits to average and lowering t_{End} to 500. This affected both λ_1 and λ_2: λ_1 was further removed from our orbit separation (and Sprott's) result while λ_2 doubled, growing farther from 0. We maintained at least 1000 orbits and $t_{End} = 1000$ for future trials based on these results.

Algorithm	Numerical Method	t_1	t_{End}	Number of initial values	Lyapunov exponents
OS	RK4	20	1000	1000	0.071015024065389
OS	IRK8	20	1000	1000	0.071005884751426
CGSO	RK4		1000	1000	0.071265371260077 0.004274602121437 -5.398739627693884

Table 7.2. Computed Lyapunov exponents for the Rössler equations with system parameter values $a = 0.2$, $b = 0.2$, $c = 5.7$ and stepsize $h = 0.01$.

7.2. THE RÖSSLER EQUATIONS

Sprott [22] states that a three-dimensional system of ODEs that has $\lambda_1 > 0$ must have $\lambda_2 = 0$ in order to remain bounded. Keeping in mind that Sprott derives λ_2 to be 0 rather than calculating it, it is not surprising that our computed value for λ_2 was nonzero. Despite the difference in λ_2, the calculated λ_1 from both algorithms was very close to Sprott's value of 0.0714. This result is a little surprising given the pronounced difference in computed values of λ_1 for the Lorenz equations. The continuous Gram-Schmidt orthonormalization result for λ_3 was also very close to Sprott's (derived) result of -5.3943. Sprott derives λ_3 by noting that

$$\lambda_1 + \lambda_2 + \lambda_3 = trace(J).$$

After setting $\lambda_2 = 0$ and computing λ_1, λ_3 may be calculated.

Parameters a, b	t_1	t_{End}	Number of initial values	Lyapunov Exponents
$0.1, 0.98$	20	1000	1000	0.023375301581978
$0.1, 0.5$	20	1000	1000	-0.081743950195497
$0.1, 0.2715$	20	1000	1000	-0.035868531917007
$-1, -0.1$	20	1000	1000	0.071203019400215

Table 7.3. Computed Lyapunov exponents for the Rabinovich-Fabrikant equations using OS and IRK8 with stepsize $h = 0.01$

7.3. THE RABINOVICH-FABRIKANT EQUATIONS

The Rabinovich-Fabrikant equations can display very different phenomena in their phase plots depending on the constant system parameters a, b used. Refer to figures 2.2, 2.3, 2.4, and 2.5. Luo *et al.* [16] have generated phase plots for many sets of parameters.

Figure 2.3 appears to have an attractor, but, for those parameter values, the system is not chaotic. Figure 2.4, a plot with what appears to be an attracting fixed point, corresponds to a negative computed λ_1. Figures 2.2 and 2.5 are chaotic, and their attractors are shown. The chaotic nature of the Rabinovich-Fabrikant equations for $a = -1$, $b = -0.1$ is somewhat surprising as Luo *et al.* [16] refer to these settings as a mathematical curiosity only.

7.4. CHUA'S CIRCUIT

The final system of ODEs studied in this thesis is Chua's circuit with constant parameters $\alpha = 9$, $\beta = \frac{100}{7}$, $a = -\frac{8}{7}$, $b = -\frac{5}{7}$. For these values, the largest Lyapunov exponent is positive and therefore the system is chaotic. Figure 2.6 is a phase plot of Chua's circuit for these parameters.

Algorithm	Numerical Method	t_1	t_{End}	Number of initial values	Lyapunov exponents
OS	IRK8	20	1000	1000	0.326602252020003
OS	RK4	20	1000	1000	0.326746202448946

Table 7.4. Computed largest Lyapunov exponent for Chua's circuit using OS and IRK8 for the constant parameter values $\alpha = 9$, $\beta = \frac{100}{7}$, $a = -\frac{8}{7}$, $b = -\frac{5}{7}$ with stepsize $h = 0.01$.

APPENDIX A

STIFFNESS RATIO

The MATLAB script stiffnessCalculator calculates the stiffness ratios of the Lorenz, Rössler, and Rabinovich-Fabrikant equations.

```
function stiffnessCalculator()
clear all
format long
%-------------------------------------------------------------------
%RF system parameters
%{
a = 0.1;
b = 0.5;
%}

%Lorenz system parameters (all > 0)
%
sigma = 10;
beta = 8/3;
rho = 28;
%}

%{
%Rossler system constant parameters
a = 0.2;
b = 0.2;
c = 5.7;
%}
```

```
%-------------------------------------------------------------------
t0 = 0;                                          %initial time
dt = .01;                                        %timestep
tEnd = 1000;                                     %end time
T = t0:dt:tEnd;
%initial conditions
x0 = rand;
y0 = rand;
z0 = rand;

v = zeros(3, length(T));%set up new matrix for solution
v(:,1) = [x0; y0; z0];              %initialize new vector
k = 1;                                  %k counts timesteps
t=t0;                                     %initialize time
maxEval = 0;                               %set up maxEval
minEval = 1000;                            %set up minEval

while t<tEnd
    v(:,k+1) = rk4(v(:,k),t,dt,@F);    %iterate solution
    %eigenvalue calculation
    jCal = jacobianCal(v(:,k));          %jacobian matrix
    evals = abs(eig(jCal));              %calculate evals
    mxE=max(evals);           %find max eval at this step
    mnE=min(evals);           %find min eval at this step
    vMax=[maxEval; mxE];         %set up max evals vector
    vMin=[minEval; mnE];         %set up min evals vector
    maxEval=max(vMax);                          %keep max
```

```
    minEval=min(vMin);                          %keep min
    %end eigenvalue calculation
    t = t+dt;                                   %iterate time
    k = k+1;                                    %iterate index
end

if (minEval ~= 0) %if statement avoids division by zero
    stiffnessRatio= maxEval/minEval;
else stiffnessRatio='undefined';
end;
display(stiffnessRatio);          %print stiffness ratio
display(maxEval);                 %display max&min values
display(minEval);

%The nested functions-------------------------------------
%RF system
%{
  function f = F(x,t)
     f = [x(2)*( x(3) - 1 + x(1)^2 ) + a*x(1);
         x(1)*( 3*x(3) + 1 - x(1)^2 ) + a*x(2);
         -2*x(3)*( b + x(1)*x(2) ) ];
  end

  function j = jacobianCal(x)
    j=[2*x(1)*x(2)+a x(1)^2+x(3)-1 x(2);
       -3*x(1)^2+3*x(3)+1 a 3*x(1);
       -2*x(2)*x(3) -2*x(1)*x(3) -2*(x(1)*x(2)+b)];
```

```
  end
%}
%Lorenz system
%
    function f = F(x,t)
        f = [sigma*(x(2) - x(1));
            x(1)*(rho - x(3)) - x(2);
            x(1)*x(2) - beta*x(3)];
    end

    function j = jacobianCal(x)
    j = [-sigma, sigma, 0;
        rho-x(3), -1, -x(1);
        x(2), x(1), -beta];
    end
%}
%Rossler system
%{
    function f = F(x,t)
        f = [-x(2) - x(3);
            x(1) + a*x(2);
            b + x(3)*(x(1)-c)];
    end

    function j = jacobianCal(x)
        j = [0 -1 -1;
            1 a 0;
```

```
            x(3)  0  x(1)-c];
    end
%}
%-------------------------------------------------------------------------
%Written 2010 by Clyde Meador
end                                                    %end function
```

APPENDIX B

ORBIT SEPARATION This appendix contains three MATLAB scripts. The first two, OS_rk4_driver.m and OS_g8i_driver.m, implement IRK8 and RK4 respectively to calculate the largest Lyapunov exponent for the Lorenz, Rössler, or Rabinovich Fabrikant equations. The third script, OS_g8i_driver_Chua.m, calculates the largest Lyapunov exponent for Chua's circuit using IRK8. Chua's circuit is analyzed separately because it requires some conditional logic to discard orbits that do not approach the attractor.

B.1. EXPLICIT 4TH-ORDER RUNGE-KUTTA METHOD

```
function OS_rk4_driver()
%-------------------------------------------------------------------
%function name: OS_rk4_driver()
%Inputs: none
%Outputs: none
%Notes on use: This script requires uncommenting in
%tandem of  the system of ODEs (in the ''nested
%functions'' section at the end and also of the
%associated system parameters (at the beginning).
%-------------------------------------------------------------------
clear all                                %clear workspace
format long      %display decimals to default precision
%-------------------------------------------------------------------
%The systems
%RF system parameters
%{
a = 0.1;
```

```
b = 0.05;
%}
%Lorenz system parameters
%
sigma = 10;
beta = 8/3;
rho = 28;
%}
%Rossler system parameters
%{
a = 0.2;
b = 0.2;
c = 5.7;
%}
%-------------------------------------------------------------------
t0 = 0;                                          %initial time
dt = .001;                                           %timestep
tEnd = 100;                                          %end time
t1 = 10;            %time before which to discard values
T = t0:dt:tEnd;                                   % time range
num_It = 1;                              % number of iterations

LLE_vector = zeros(1,num_It);    %vector holds exponents

for co1=1:num_It
x0 = rand;                                  %initial conditions
y0 = rand;
```

```
z0 = rand;

d0 = 10^(-9);                    %separation constant d0
%set up new matrices to handle separate solutions
v = zeros(3, length(T));
w = zeros(3, length(T));
%initialize new vectors
v(:,1) = [x0; y0; z0];
w(:,1) = [x0; y0; z0];
w(1,1) = w(1,1) + d0;                    %separate by d0
k = 1;                    %count for number of timesteps
n =0;                    %count for number of logs taken
L2 = 0;                        %sum of lyapunov exponents
t=t0;                                    %initialize time
while t<tEnd
    %iterate solution 1 step
    v(:,k+1) = rk4(v(:,k),t,dt,@F);
    w(:,k+1) = rk4(w(:,k),t,dt,@F);
    %calculate distance between v and w
    d11 = (v(1,k+1)-w(1,k+1))^2;
    d12 = (v(2,k+1)-w(2,k+1))^2;
    d13 = (v(3,k+1)-w(3,k+1))^2;
    d1 =sqrt(d11+d12+d13);
    if t>t1                          %only perform after t1
        L1 = log(abs(d1/d0));                %evaluate log
        L2 = L1 + L2;                            %sum logs
        n = n+1;                        %iterate log count
```

```
    end
%change w vector so it is epsilon (d0) away from v
diff = [w(:,k+1)-v(:,k+1)];
w(:,k+1) = v(:,k+1) + (d0/d1)*diff;
t = t+dt;                                   %iterate time
k = k+1;                                    %iterate index
end
L3 = (L2/n)/dt;                  %Largest Lyapunov exponent
LLE_vector(co1) = L3;                    %store L3 in vector
end
L4 = (sum(LLE_vector))/num_It;             %average all L3
display(L4);                                %print average

%-------------------------------------------------------------
%The nested functions
%RF system
%{
  function f = F(x,t)
      f = [x(2)*( x(3) - 1 + x(1)^2 ) + a*x(1);
          x(1)*( 3*x(3) + 1 - x(1)^2 ) + a*x(2);
          -2*x(3)*( b + x(1)*x(2) ) ];
  end

%}
%Lorenz system
%
    function f = F(x,t)
```

```
        f = [sigma*(x(2) - x(1));
            x(1)*(rho - x(3)) - x(2);
            x(1)*x(2) - beta*x(3)];
    end
%}
%Rossler system
%{
    function f = F(x,t)
        f = [-x(2) - x(3);
            x(1) + a*x(2);
            b + x(3)*(x(1)-c)];
    end
%}
%-------------------------------------------------------------
%Written 2010 by Clyde-Emmanuel Meador
end                                          %end function
```

B.2. IMPLICIT 8TH-ORDER RUNGE-KUTTA METHOD

```
function OS_g8i_driver()
%-------------------------------------------------------------
%function name: OS_g8i_driver()
%Inputs: none
%Outputs: none
%Notes on use: This script requires uncommenting in
%tandem of  the system of ODEs (in the ''nested
%functions'' section at the end and also of the
```

```
%associated system parameters (at the beginning).
%-------------------------------------------------------------------
clear all                                   %clear workspace
format long      %display decimals to default precision
%-------------------------------------------------------------------
%The systems
%RF system parameters
%{
a = 0.1;
b = 0.05;
%}
%Lorenz system parameters
%
sigma = 10;
beta = 8/3;
rho = 28;
%}
%Rossler system parameters
%{
a = 0.2;
b = 0.2;
c = 5.7;
%}
%-------------------------------------------------------------------
%Initializations
t0 = 0;                                 %initial time value
dt = .01;                                           %timestep
```

```
tEnd = 1000;                                    %end time
t1 = 20;          %time before which to discard values
T = t0:dt:tEnd;              %vector holding time values
%initial conditions
num_IVs = 1000;%number of random initial values to take
LLE_vector = zeros(1,num_IVs);          %holds exponents
%for-loop iterates for each set of IVs
for co1=1:num_IVs
x0 = rand;
y0 = rand;
z0 = rand;
%separation constant d0
d0 = 10^(-9);
%set up new matrices to handle separate solutions
v = zeros(3, length(T));
w = zeros(3, length(T));
%initialize new vectors
v(:,1) = [x0; y0; z0];
w(:,1) = [x0; y0; z0];
%separate by d0
w(1,1) = w(1,1) + d0;
k = 1;                                  %counter for solution
n =0;                  %counter for number of logs taken
L2 = 0;                          %sum of Lyapunov constants
t=t0;                                       %initialize time

    while t<tEnd %numerically solve system
```

```
    %iterate solution 1 step
    J = jacobianCal(v(:,k)); %Jacobian matrix
    J2 = jacobianCal(w(:,k));%Jacobian matrix
    v(:,k+1) = gauss8_newton(v(:,k),dt,@F,J);
    w(:,k+1) = gauss8_newton(w(:,k),dt,@F,J2);
    %

    %calculate distance between v, w
    d11 = (v(1,k+1)-w(1,k+1))^2;
    d12 = (v(2,k+1)-w(2,k+1))^2;
    d13 = (v(3,k+1)-w(3,k+1))^2;
    d1 =sqrt(d11+d12+d13);
        if t>t1              %only perform after time t1
        L1 = log(abs(d1/d0));              %evaluate log
        L2 = L1 + L2;                        % sum logs
        n = n+1;                       %iterate log count
        end
    %adjust w so it is epsilon (d0) from v
    diff = [w(:,k+1)-v(:,k+1)];
    w(:,k+1) = v(:,k+1) + (d0/d1)*diff;
    t = t+dt;                                %iterate time
    k = k+1;                                %iterate index
    end

L3 = (L2/n)/dt;          %calculate largest Lyapunov exp.
LLE_vector(co1) = L3;          %enter the LLE into vector
end
```

```
L4 = (sum(LLE_vector))/num_IVs;          %average LLEs
display(L4);                             %display average
%can ungreen plot3 to check solution
%plot3(v(1,:),v(2,:),v(3,:))
%-------------------------------------------------------
%The nested functions
%RF system
%{
  function f = F(x,t)
     f = [x(2)*( x(3) - 1 + x(1)^2 ) + a*x(1);
          x(1)*( 3*x(3) + 1 - x(1)^2 ) + a*x(2);
          -2*x(3)*( b + x(1)*x(2) ) ];
  end

  function j = jacobianCal(x)
    j=[2*x(1)*x(2)+a x(1)^2+x(3)-1 x(2);
       -3*x(1)^2+3*x(3)+1 a 3*x(1);
       -2*x(2)*x(3) -2*x(1)*x(3) -2*(x(1)*x(2)+b)];
  end
%}
%Lorenz system
%
    function f = F(x,t)
        f = [sigma*(x(2) - x(1));
             x(1)*(rho - x(3)) - x(2);
             x(1)*x(2) - beta*x(3)];
    end
```

```
    function j = jacobianCal(x)
    j = [-sigma, sigma, 0;
        rho-x(3), -1, -x(1);
        x(2), x(1), -beta];
    end
%}
%Rossler system
%{
    function f = F(x,t)
        f = [-x(2) - x(3);
            x(1) + a*x(2);
            b + x(3)*(x(1)-c)];
    end

    function j = jacobianCal(x)
        j = [0 -1 -1;
            1 a 0;
            x(3) 0 x(1)-c];
    end
%}
%-------------------------------------------------------------------------
%Written 2010 by Clyde-Emmanuel Meador
end                                                    %end function
```

B.3. IMPLICIT 8TH ORDER RUNGE-KUTTA METHOD: CHUA'S CIRCUIT

```
function OS_g8i_driver_Chua()
%-------------------------------------------------------------------
%function name: OS_g8i_driver_Chua()
%Inputs: none
%Outputs: none
%Notes on use: Not every initial value in the cube
% x,y,z in [0,1] approaches the attractor. If an orbit
% does not approach the attractor in this case, it
% spirals outward and has infinite Lyapunov exponent
%-------------------------------------------------------------------
clear all                                   %clear workspace
format long      %display decimals to default precision
%-------------------------------------------------------------------
%Chua's circuit constant parameters
%
alpha = 9;
beta = 100/7;
a = -8/7;
b = -5/7;
%}
%-------------------------------------------------------------------
%Initializations
t0 = 0;                                  %initial time value
dt = .01;                                          %timestep
tEnd = 1000;                                       %end time
t1 = 20;          %time before which to discard values
T = t0:dt:tEnd;              %vector holding time values
```

```
ActNum = 0;              %orbits which approach attractor
escapeCounter = 0;                    %orbits which don't

%initial conditions
num_IVs = 1000;   %number of initial random IVs to take
LLE_vector = zeros(1,num_IVs);          %holds exponents
%for-loop iterates for each set of IVs
for co1=1:num_IVs
x0 = rand;
y0 = rand;
z0 = rand;
%separation constant d0
d0 = 10^(-9);
%set up new matrices to handle separate solutions
v = zeros(3, length(T));
w = zeros(3, length(T));
%initialize new vectors
v(:,1) = [x0; y0; z0];
w(:,1) = [x0; y0; z0];
%separate by d0
w(1,1) = w(1,1) + d0;
k = 1;                                %counter for solution
n =0;                  %counter for number of logs taken
L2 = 0;                          %sum of lyapunov constants
t=t0;                                    %initialize time

    while t<tEnd %numerically solve system
```

```
    %iterate solution 1 step
    J = jacobianCal(v(:,k));            %Jacobian matrix
    J2 = jacobianCal(w(:,k));           %Jacobian matrix
    v(:,k+1) = gauss8_newton(v(:,k),dt,@F,J);
    w(:,k+1) = gauss8_newton(w(:,k),dt,@F,J2);
    %calculate distance between v, w
    d11 = (v(1,k+1)-w(1,k+1))^2;
    d12 = (v(2,k+1)-w(2,k+1))^2;
    d13 = (v(3,k+1)-w(3,k+1))^2;
    d1 =sqrt(d11+d12+d13);
        if ((t>t1))          %only perform after time t1
        L1 = log(abs(d1/d0));              %evaluate log
        L2 = L1 + L2;                          %sum logs
        n = n+1;                       %iterate log count
        end
    %adjust w so it is epsilon (d0) from v
    diff = [w(:,k+1)-v(:,k+1)];
    w(:,k+1) = v(:,k+1) + (d0/d1)*diff;       %normalize
    t = t+dt;                                %iterate time
    k = k+1;                                %iterate index
    end


L3 = (L2/n)/dt;          %calculate largest Lyapunov exp.
    if(isnan(L3)) %if orbit does not approach attractor
        escapeCounter = escapeCounter+1;
        break
    else              %if orbit does approach attractor
```

```
        LLE_vector(co1) = L3;%enter the LLE into vector
        ActNum = ActNum+1;
    end

end
L4 = (sum(LLE_vector))/ActNum; %average LLEs over #IVs.
display('LLE:')
display(L4);                          %display average LLE
display('Proportion of IVs not in basin:')
display(escapeCounter/num_IVs);
%-------------------------------------------------------------
%The nested functions
%Chua's Circuit
    function f = F(x,t)
        if (x(1) > 1)
            f = [alpha*(x(2)-x(1)-( b*x(1) + a - b ));
                x(1)-x(2)+x(3);
                -beta*x(2)];
        elseif (x(1) < -1)
            f = [alpha*(x(2)-x(1)-( b*x(1) - a + b ));
                x(1)-x(2)+x(3);
                -beta*x(2)];
        else
            f = [alpha*(x(2)-x(1)-( a*x(1) ));
                x(1)-x(2)+x(3);
                -beta*x(2)];
        end
```

```
    end

    function j = jacobianCal(x)

        if (x(1) > 1)
            cc = b;
        elseif (x(1) < -1)
            cc = b;
        else
            cc = a;
        end

        j = [0, alpha, cc;
            1, -1, 1;
            0, -beta, 0];
    end
% ------------------------------------------------------------------
%Written 2011 by Clyde-Emmanuel Meador
end                                             %end function
```

APPENDIX C

NUMERICAL METHODS The two scripts in this appendix, rk4.m and gauss8newton.m were written by Scott Sarra. They implement the numerical methods RK4 and IRK8.

C.1. RK4

```
function v = rk4(V,t,k,F)

    s1 = feval(F,V,t);
    s2 = feval(F,V + k*s1/2,t+k/2);
    s3 = feval(F,V + k*s2/2,t+k/2);
    s4 = feval(F,V + k*s3,t+k);

    v = V + k*(s1 + 2*s2 + 2*s3 + s4)/6;
    %written by Scott Sarra
```

C.2. IRK8

```
%  8th order implicit Runge-Kutta method
%
%  Gauss Method, stages=4, order=8
%
%  modified Newton's method to solve
%  the nonlinear systems
```

```
function v = gauss8_newton(V,dt,F,J,TOL,MAXIT)

   %use default max iterations
    if nargin<6, MAXIT=250; end
    if nargin<5, TOL=1e-14; end%use default tolerance

     s = 4;                                    %four stages
     N = length(V);                  %number of variables

     %RK coefficients
      w1 = 1/8 - sqrt(30)/144;
     w1p = 1/8 + sqrt(30)/144;
      w2 = 0.5*sqrt((15 + 2*sqrt(30))/35);
     w2p = 0.5*sqrt((15 - 2*sqrt(30))/35);
      w3 = w2*(1./6 + sqrt(30)/24);
     w3p = w2p*(1/6 - sqrt(30)/24);
      w4 = w2*(1/21 + 5*sqrt(30)/168);
     w4p = w2p*(1/21 - 5*sqrt(30)/168);
      w5 = w2 - 2*w3;
     w5p = w2p - 2*w3p;
      %A8 = table of a_ij stage coefficients
      A8 = [w1 w1p-w3+w4 w1p-w3-w4 w1-w5;
           w1-w3p+w4 w1p w1p-w5p w1-w3p-w4;
           w1+w3p+w4 w1p+w5p w1p w1+w3p-w4;
           w1+w5 w1p+w3+w4p w1p+w3-w4p w1];
      %inverse of A8, used to get d vector
```

```
A8i = inv(A8);
 %b vector holds weights
b = [2*w1 2*w1p 2*w1p 2*w1];
d = b*A8i;                      %x_n+1 = x_n + sum d, z
c = sum(A8');                              %time weights
    %I  - kronecker product of A8 and J
    A = (eye(s*N) - dt*kron(A8,J));
    %kronecker product of A8 and I
    B = kron(A8,eye(3));
    %LU factorization of A
[L,U] = lu(A);
    %zero vector of appropriate size
    z = zeros(s*N,1);
    %orig. f(x) in matrix form
    Fx = repmat(feval(F,V),s,1);%initial z is zero

   df = 1;                  %counter: convergence rate
 iter = 0;                      %counter: # iterations
%while-loop performs Newton's method
 while (abs(df)>TOL & iter<MAXIT)

   Fx([1 2 3]) = feval(F, V + z([1 2 3]));
   Fx([4 5 6]) = feval(F, V + z([4 5 6]));
   Fx([7 8 9]) = feval(F, V + z([7 8 9]));
   Fx([10 11 12]) = feval(F, V + z([10 11 12]));

   rt = -z + dt*B*Fx;                    %calculate rt
```

```
        dz = U\(L\rt);                          %calculate dz
        z = z + dz;                               %iterate z

        df = norm(dz,inf);  %update convergence counter
        iter = iter + 1;       %update iteration counter
      end  % while

 %new vector is the next timestep value.
 %if prev. is x_n, this is x_{n+1}
    v(1) = V(1) + dot(d,z([1 4 7 10]));
    v(2) = V(2) + dot(d,z([2 5 8 11]));
    v(3) = V(3) + dot(d,z([3 6 9 12]));
%written by Scott Sarra
```

REFERENCES

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

[2] Bress, E., Gruber, J. M., and Directors. *The Butterfly Effect*.

[3] Butcher, J. Implicit Runge - Kutta processes. *Math. Comp. 18*, 85 (1964), 50 – 64.

[4] Butcher, J. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2008.

[5] Christiansen, F., and Rugh, H. Computing lyapunov spectra with continuous gram-schmidt orthonormalization. *Nonlinearity 10* (1997), 1063–1072.

[6] Danca, M.-F. A multistep algorithm for odes. *Dyn. Cont. Disc. Imp. Sys. 13* (2006), 803 – 821.

[7] Danca, M.-F., and Chen, G. Bifurcation and chaos in a complex model of dissipative medium. 3409 – 3447.

[8] Devaney, R. L. *An Introduction to Chaotic Dynamical Systems*, second ed. Westview, 2003.

[9] Gleick, J. *Chaos*. Penguin, 1988.

[10] Hairer, E., and Wanner, G. *Solving Ordinary Differential Equations II*. Springer-Verlag, 1996.

[11] Hayes, W. B. *Rigorous Shadowing of Ordinary Differential Equations by Containment*. PhD thesis, University of Toronto, 2001.

[12] Hayes, W. B., Jackson, K. R., and Young, C. Rigorous high-dimensional shadowing using containment: The general case. *Discrete Contin. Dyn. S. 14*, 2 (2006), 329–342.

[13] Iserles, A. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 2004.

[14] Liniger, W., and Willoughby, R. A. Efficient integration methods for stiff systems of ordinary differential equations. *SIAM J. Numer. Anal. 7* (1970), 47–66.

[15] Lorenz, E. N. Deterministic nonperiodic flow. *J. Atmos. Sci. 20* (1963), 130–141.

[16] Luo, X., Small, M., Danca, M.-F., and Chen, G. On a dynamical system with multiple chaotic attractors. *Int. J. Bif. Chaos 17* (2007), 3235 – 3251.

[17] Matsumoto, T., Chua, L. O., and Komuro, M. The double scroll. *IEEE Trans. Circuits Syst. CAS-32*, 8 (1985), 798 – 818.

[18] Meador, C. Numerical methods, phase plots, and the rabinovich-fabrikant system, May 2009. Senior Thesis, Marshall University.

[19] Murray, J. *Mathematical Biology: I. An Introduction*, third ed. Springer, 2002.

[20] Sarra, S. A. Personal communication.

[21] Sarra, S. A., and Meador, C. On the solution of chaotic dynamical systems using extend precision floating point arithmetic and very high order numerical methods. under review, 2011.

[22] Sprott, J. C. *Chaos and Time-Series Analysis*. Oxford University Press, 2003.

[23] Strogatz, S. H. *Nonlinear Dynamics and Chaos*. Westview Press, 2000.

[24] Thompson, P. *Numerical Weather Analysis and Prediction*. The Macmillan Company, 1961.

[25] Tucker, W. The lorenz attractor exists. *C. R. Acad. Sci. Paris 328* (1999), 1197 – 1202.

VITA

Clyde-Emmanuel Estorninho Meador

Marshall University

Department of Mathematics

Smith Hall

Huntington, WV 25755

Phone: (304) 237-4926

Email: meador16@marshall.edu

Personal

- Born on December 14, 1986.
- United States Citizen.

Education

- B.S. Mathematics, Marshall University, 2009
- M.A. Mathematics, Marshall University, 2011

Employment

- Adjunct Instructor, Department of Mathematics, Marshall University, Huntington WV (May-June 2010). I taught an expanded section of the college algebra course during a 4-week summer term. I wrote the syllabus and course examinations.
- Teaching Assistant, Department of Mathematics, Marshall University (August 2009 May 2011). I taught a selection of math courses in addition to tutoring duties. I designed the course structure, wrote syllabi, and was the only lecturer for 6 courses. Classes taught: Algebra, Trigonometry, Precalculus.

Research Experience

- Numerical computation of Lyapunov exponents. Department of Mathematics and Applied Sciences, College of Science, Marshall University. My thesis research involves methods for the computation of Lyapunov exponents, their uses, and relative merits. Supervising professor: Dr. Anna Mummert.

- Numerical methods for differential equations. Department of Mathematics and Applied Sciences, College of Science. My ongoing research with Dr. Scott Sarra has involved testing of numerical integration methods as well as extended precision floating-point arithmetic experiments. An article was submitted for publication during the Spring 2010 term, and another was submitted in the Fall 2010 term.

- Numerical Methods and the Rabinovich-Fabrikant Equations. Department of Mathematics and Applied Sciences, College of Science, Marshall University. My undergraduate research involved MATLAB implementation of numerical methods of integration such as Runge-Kutta methods and Local Iterative Linearization. These methods are used on the chaotic Rabinovich-Fabrikant system in order to compare phase plots and the differing results between methods for a variety of constant parameters. An article was submitted for publication during the Spring 2009 term. Supervising professor: Scott Sarra.

Talks

- April 13, 2011. Numerical calculation of Lyapunov exponents for three-dimensional systems of ordinary differential equations. Master's thesis defense.

- April 24, 2010. Numerical methods for ordinary differential equations. Overview of the motivation for numerical solvers with a derivation of Euler's method, discussion of important characteristics of methods such as stability and order. Given to an audience

of faculty, undergraduates, graduate students, and alumni for the Minds of Marshall Academic Festival.

- February 11, 2010. Numerical methods for ODEs: Implicit and Explicit Methods. Discussion of the use of explicit and implicit numerical methods for ordinary differential equations with a focus on a specific predictor-corrector method that was misrepresented in literature. Given to an audience of faculty members, graduate students, and undergraduate members of Pi Mu Epsilon (Beta chapter).

- December 8, 2008. Numerical methods and the Rabinovich-Fabrikant Equations. Overview of implementation of numerical integration methods given to an undergraduate numerical analysis class. Marshall University.

- December 5, 2008. Numerical methods and the Rabinovich-Fabrikant Equations. Presentation and defense of senior research to faculty members, graduate students, and seniors. Marshall University.

- November 14, 2008. Numerical methods and the Rabinovich-Fabrikant Equations. Accessible summary of a process using MATLAB to study the Rabinovich-Fabrikant equations numerically with emphasis on qualitative phase plot analysis given to undergraduate and graduate members of Pi Mu Epsilon (Beta chapter) as well as interested faculty members.

Software Proficiency

- MATLAB scripting
- Python scripting
- Microsoft Office
- HTML

Language Proficiency

Spanish minor, undergraduate (completed)

References

- Dr. Anna Mummert, Assistant Professor, Department of Mathematics and Applied Sciences, Marshall University, Huntington WV 25755 Tel: (304) 696-3041, email: mummerta@marshall.edu
- Dr. Carl Mummert, Assistant Professor, Department of Mathematics and Applied Sciences, Marshall University, Huntington WV 25755 Tel: (304) 696-4646 email: mummertc@marshall.edu
- Dr. Scott Sarra, Professor, Department of Mathematics and Applied Sciences, Marshall University, Huntington WV 25755 Tel: (304) 208-3365, email: sarra@marshall.edu
- Dr. Ralph Oberste-Vorth, Professor and Chair, Department of Mathematics and Applied Sciences, Marshall University, Huntington WV 25755 Tel: (304) 696-6010, email: oberstevorth@marshall.edu