AN ABSTRACT OF THE THESIS OF

Solveig T. Mueller for the degree of Honors Baccalaureate of Science in Mathematics
presented on May 19, 2011.

Title: Numerical Method and Step Size Variation in the Lorenz Equations.

Abstract approved:

Robert Higdon

The relative accuracy of solutions to chaotic systems was examined using the Lorenz system as a case study. The classical fourth order Runge-Kutta method was employed to generate solutions using various step sizes, and the resultant solutions were compared. A second set of solutions was then computed using one of the second order Runge-Kutta methods and the same range of step sizes as before. The resultant data suggest that short-term accuracy can be significantly increased by decreasing the step size of the numerical method, and a more accurate method similarly increases short-term accuracy. As expected, the long-term solutions eventually diverge; the qualitative behavior of these long-term solutions is still evident regardless of the step size or the method used.

Key Words: Chaos, Differential Equations, Numerical Methods, Step Size Variation, MATLAB

Corresponding e-mail address: muellers@onid.orst.edu

Numerical Method and Step Size Variation in the Lorenz Equations

by

Solveig T. Mueller

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics

(Honors Scholar)

Presented May 19, 2011

Commencement June 2011

Honors Baccalaureate of Science in Mathematics project of Solveig T. Mueller presented on May 19, 2011.

APPROVED:

Mentor, representing Mathematics

Committee Member, representing Mathematics

Committee Member, representing Mathematics

Chair, Department of Mathematics

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Solveig T. Mueller, Author

TABLE OF CONTENTS

LIST OF FIGURES

Numerical Method and Step Size Variation in the Lorenz Equations

1. Introduction

Mathematical models play a vital role in virtually every branch of science. Among the vast array of tools available for this purpose, differential equations are a particularly useful modeling tool. From models of quantum particle oscillations to models of atmospheric dynamics, differential equations lay the groundwork for dozens of systems. However, while convenient and useful, differential equations are not always easy to analyze, and one of the prominent examples of this is chaos.

If a system is chaotic, an infinitesimal variation in the initial conditions of the system can cause a drastic difference in the exhibited behavior of that system after a sufficiently long elapsed time. Thus, the only way to determine the exact solution for all time is to start with perfectly precise initial conditions. Unfortunately, it is absolutely impossible to have perfectly accurate initial conditions to a real world problem. The scientific community is continually striving to increase the accuracy of their measurements, but we can never completely banish experimental error; empirical data are inherently imprecise. Thus, it is impossible to attain a perfect solution to a natural system modeled with a chaotic system; there will always be some level of imprecision in the modeling of such a system.

The method used to compute the solution numerically could potentially contribute as much error to the system as the initial conditions can. Error is an intrinsic part of numerical computation, and each iteration builds upon the error of the

preceding iteration. Thus, in a chaotic system, the computation has the power to blow the error completely out of proportion. In this context, the choice of numerical method is very important.

Chaos and imprecise empirically derived initial conditions destroy our long-term ability to definitively determine values for a given system at a specific time. However, there is still a lot of information which can be gained from analyzing the general trends of a chaotic system. Weather and climate modeling are a great example of this. Although the first few days of a weather forecast will be fairly accurate, forecasts which "predict" the weather more than a week in advance are unreliable. The longer the elapsed time, the more unreliable such predictions become. In contrast, climate models provide long-term information on general weather trends such as average temperature and average precipitation. Fine details become obscure within a few days, but big picture trends can be estimated far in advance.

To investigate these behaviors, we have chosen to examine the phase space behavior of the system defined by the Lorenz equations. Named after the man who discovered chaos, the Lorenz equations are a relatively simple chaotic system which elegantly illustrates the behavior discussed above. Although these equations are far simpler than those which model oceanic and atmospheric dynamics, they exhibit chaotic effects analogous to such climate models. Consequently, these equations are very interesting from a theoretical perspective. In this paper, we examine how different step sizes affect the accuracy of the solutions, and we compare the solutions generated by the second order and the fourth order Runge-Kutta methods. Lastly, we discuss the

extent to which numerical computations can capture the defining characteristics of a chaotic system's solutions.

2. Systems in Phase Space

2.1. Second Order System

Many systems can be modeled using first order differential equations, while others systems are represented best by higher order equations. Conveniently, every higher order system can be rewritten as a set of first order equations. This fact has some very interesting benefits.

For example, consider a simple pendulum of unit length without damping. Using Newton's Second Law, we know this system can be represented by a second order equation of the form

$$m\ddot{\theta} = -mg\sin\theta \tag{1}$$

or more simply,

$$\ddot{\theta} = -g\sin\theta, \tag{2}$$

where *g* is the gravitational acceleration, *m* is the mass of the pendulum, θ is the displacement angle from a perfectly downward position, and the double-dot notation indicates the second derivative with respect to time. By transforming Equation 2 into a system of *two* first order equations, we can glean quite a bit of information out of this system without directly solving it. Thus, we make the substitution

$$y_1 = \theta \tag{3}$$

$$y_2 = \dot{\theta} = \frac{d\theta}{dt}.$$

The system becomes

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ -g\sin y_1 \end{bmatrix}, \tag{4}$$

and then we can use phase plane analysis to examine the behavior of y_1 and y_2 over time.

Our new set of variables, y_1 and y_2, correspond to displacement angle and velocity, respectively. The phase plane represents the magnitude of each variable on a different axis. Thus, an ordered pair representing a point in the phase plane for this system represents a potential combination of position and velocity at a fixed time t. Whether it is possible for our pendulum to attain a *particular* combination of position and velocity is entirely dependent on its initial conditions. Therefore, a complete appreciation of the system must consider all combinations of initial conditions and their resulting solutions. To consider all these possibilities at once, we can examine the system's phase portrait.

A phase portrait is like a map of all the potential paths a particle could trace out in phase space as time progresses. Each path, called a flow, represents a unique solution. Because this system is autonomous, every point on a given flow could act as a set of initial conditions to generate that particular solution. Like a gradient field, not every point must have its flow explicitly displayed in the phase portrait; instead, we merely need to see enough flow lines to get a general feel for what the rest of the solutions ought to look like.

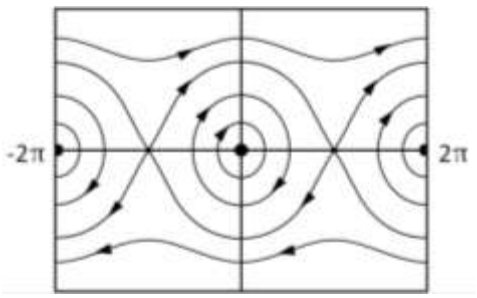


Figure 1: The phase portrait of a simple undamped pendulum. The position, y_1, is shown on the horizontal axis, and the velocity, y_2, is shown on the vertical axis.

Because phase portraits are not animated, the solutions are shown without respect to temporal progression; it is as if the particle had already traversed the solution and left a trial of slug slime behind it. To convey the missing temporal progression, arrows are often included indicating the direction of motion.

In the phase portrait of our undamped pendulum, shown in Figure 1 above, the curves which cross the horizontal axis at $\pm\pi$ are of particular interest; they act as a boundary between two different types of solutions. Any flow which has a point inside this eye-shaped enclosure will have a closed curved structure. This suggests that the pendulum is continually swinging back and forth across the same stretch of space. Any flow which has a point outside the enclosure is not a closed loop. This seems to suggest it never traverses the same stretch of space twice, but such an interpretation is only partially true. From a physical perspective, we know that an angle of zero and an angle of 2π indicate the same position, so the pendulum must be returning to the same physical point as time progresses. From a theoretical perspective, the perpetual increase in displacement angle *actually* indicates the pendulum never changes direction. At $\pm\pi$, the pendulum reaches its highest point and then travels over the top. Thus, while the physical space is still the same, the flow lines differentiate between one rotation and the next by the angular displacement increase. Identifying the distinction between these two fundamentally different solution types is one of the many useful aspects of phase plane diagrams.

We were able to determine all the information in the preceding discussion without ever solving the system. Clearly, phase plane analysis can be a very helpful tool when analyzing differential systems.

2.2. Equilibria and Closed Orbits

Notice that in Figure 1 above, the closed orbits within the eye-shaped enclosure are concentric. The centers of these orbits correspond to where the pendulum has zero velocity, and they are equilibrium points for this system. An equilibrium point is a very special point: if the particle starts at equilibrium, and is not displaced by some external force, it will stay at that point forever. For this reason, the points at $(\pm\pi,0)$ are also equilibrium points, and they correspond to where the pendulum has zero velocity and is balanced pointing straight up. For every other combination of initial conditions, the pendulum's state will change as time progresses.

While the aforementioned points are all equilibria, their respective stability states set them apart. It is physically impossible to maintain ideal conditions without any margin of error, so we must consider what happens when a tiny displacement is added to the system. The reaction to such error determines an equilibrium point's stability. For example, in the straight up position, even the tiniest displacement to either side would cause the pendulum to begin oscillating wildly around the origin (or one of its periodic equivalents). Consequently, this is called an unstable equilibrium. In contrast, the equilibrium at the origin, and its periodic equivalents, are called stable equilibria because a tiny displacement from the origin would create correspondingly tiny

oscillations. Solutions which start close to a stable equilibrium must remain close to that equilibrium. That is why the origin has orbits, whereas ($\pm\pi$,0) do not have orbits.

2.3. Other Behaviors

While the system above was only a relatively simple example, many of the concepts used to analyze it extend to other systems. Two dimensional systems can exhibit many other types of behaviors, such as sources or sinks, which can be identified using phase plane analysis. For example, if the pendulum had been damped, the phase plane would have exhibited flows spiraling in toward the origin; in such a case, the origin would have been called a sink. Conversely, if the pendulum had been driven, the flows would have been spiraling out from the center; in this case, the origin would have been called a source. In a system which is both damped and driven, the phase portrait could have shown a combination of these two behaviors. The flows originating close to the origin could have spiraled out while the flows at infinity spiraled in—until they met at some circular limit cycle at which the forces completely balance each other.

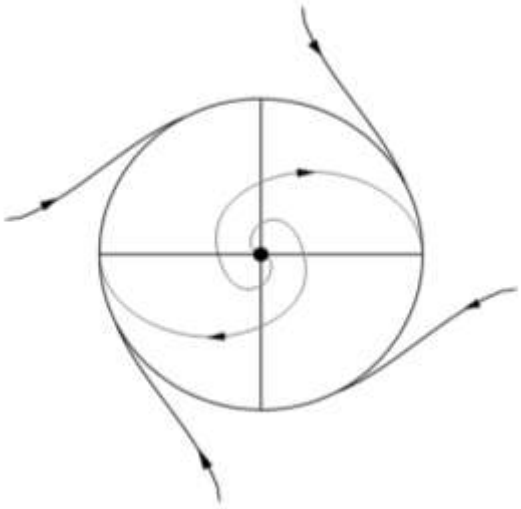


Figure 2: A stable limit cycle.

Unlike the closed cycles from the isolated pendulum above, which did not attract other flow lines, a limit cycle is a closed cycle which attracts nearby flows whose initial conditions are not on the cycle. In this sense, a limit cycle acts similar to a sink. If flow lines are attracted from both the inside and outside, the limit cycle is stable; because it is a cycle rather than a point, such a limit cycle is called orbitally stable.

2.4. Higher Order Systems

These concepts can all be extended to three dimensional systems, so we will use this approach to examine the Lorenz equations. It should be noted, however, that three dimensional systems, including the Lorenz system, are more complex for a couple of different reasons. Solutions in three dimensions can have more complicated flow patterns than solutions of two dimensional systems, and they are harder to represent visually in a two dimensional figure. Secondly, solutions to three dimensional systems exhibit chaotic behavior, whereas two dimensional systems do not. Thus, we must keep these factors in mind while examining the solutions.

3. Chaos and the Lorenz System

3.1. Discovery

Edward N. Lorenz first stumbled upon the effects of chaos in 1961. Lorenz was a meteorologist, and he was developing a complicated weather model using many coupled differential equations. One day, he decided to reexamine some of the simulations he had been running the night before. Instead of starting the entire computation from the beginning, he examined the data he had printed off during the previous session and selected a set of output values from the middle of that data set to serve as initial conditions for his newest computation. Thus, the first few output values from his new data set should have corresponded to the last few entries in his previous data set. The data points from this overlapping time interval should have matched perfectly. His new data, however, appeared to be completely different from his previous run.

The cause was a simple round off error. Much like modern computers, Lorenz's computer displayed fewer digits than the computer was actually using; his computer internally stored and used six digits for its computations, but it only printed out three. Still, conventional wisdom of the time dictated that this minor variation in initial conditions should only create a minor variation in the solutions. This was clearly not the case. This phenomenon, where a slight variation in the initial conditions causes an unintuitively large variation in the long-term solution, became known as sensitivity to initial conditions, and the resulting "random" behavior is called chaos. Lorenz's accident instigated an investigation which ultimately proved that sensitivity to initial conditions is

inherent to certain types of systems, and these systems exhibit mathematical chaos as a result.

3.2. Birth of the Lorenz Equations

After confirming that his findings were not due to an unrelated computer problem, Lorenz began investigating a simplified system. The set of three equations,

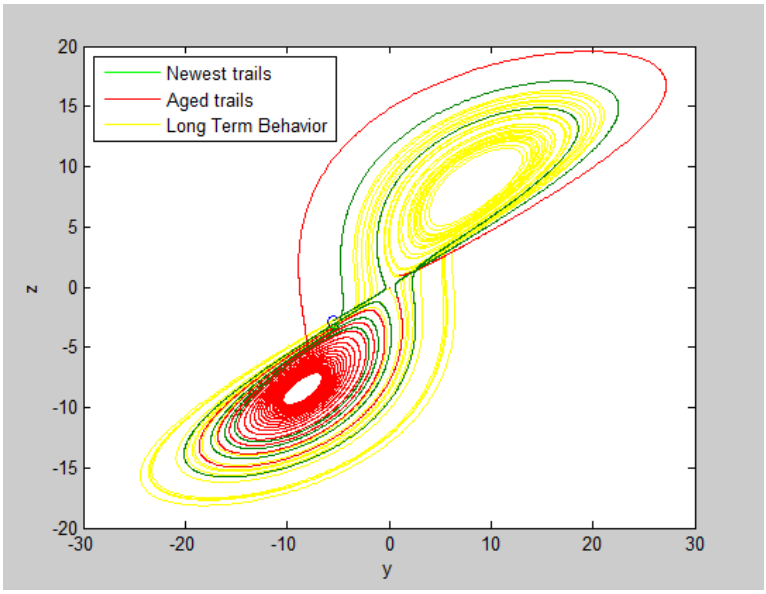
$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz \end{aligned} \tag{5}$$

were a particularly useful research model for him. These three equations, which came to be known as the Lorenz equations, are a simplified representation of the convective motion of a fluid cell which is heated from below and cooled from above. In this system, the variable x is a measure of the convection, and the variables y and z represent the horizontal and vertical temperature variations, respectively. The parameters σ, r, and b are assumed to be positive constants, with $\sigma > b + 1$. Lorenz used the values $\sigma = 10$, $r = 28$, and $b = \frac{8}{3}$. These values have become somewhat conventional since then, and we shall retain them.

In reality, these equations are an oversimplification of a very complex process because they describe a very specialized situation. However, their simplicity is their redeeming factor. Lorenz hoped that by studying chaotic effects in a simple system, it would be easier to identify how chaotic effects would manifest themselves in more complex systems.

3.3. Expected Behavior

With our chosen parameters, the Lorenz equations have three equilibria, located at $(6\sqrt{2}, 6\sqrt{2}, 27)$, at $(-6\sqrt{2}, -6\sqrt{2}, 27)$, and at the origin. The first two of these are stable, but the origin is not.



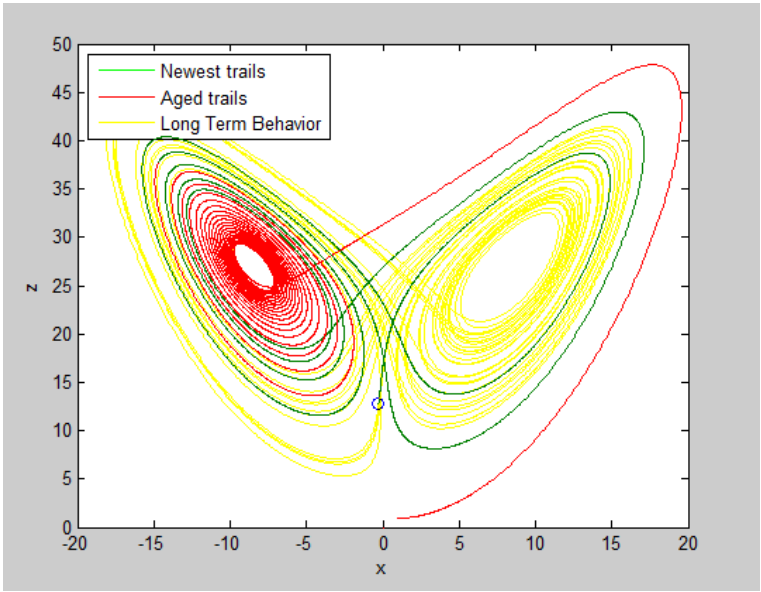


Figure 3: A single trajectory of the Lorenz equations. This solution was computed using the fourth order Runge-Kutta method discussed below.

The expected trajectories are very similar to closed cycles, but they are not closed. As can be seen in Figure 3 above, these trajectories tend to circle around the two stable equilibria but do not ever repeat the same curve twice.

4. Numerical Method

4.1. Approximate Solutions

Consider an initial value problem of the form

$$\dot{y} = f(y), \quad y(t_o) = y_o. \tag{6}$$

Any system of first order equations can be written as an *N* dimensional variation of the scalar case above. This is

$$\dot{\boldsymbol{Y}} = \boldsymbol{F}(\boldsymbol{Y}) \quad \boldsymbol{Y}(t_o) = \boldsymbol{Y}_o, \tag{7}$$

where $\boldsymbol{Y} = \left(Y_{(1)}, Y_{(2)}, \ldots\ Y_{(N)}\right)$ is a vector of *N* unknown functions, $\boldsymbol{Y}_o = \left(y_{(1)o}, y_{(2)o}, \ldots\ y_{(N)o}\right)$ is a vector representing the *N* initial conditions, $\boldsymbol{F} = (f_{(1)}, f_{(2)}, \ldots\ f_{(N)})$ is a vector of *N* functions, and $\dot{\boldsymbol{Y}}$ is the corresponding vector of *N* derivatives. We will maintain this notation throughout the following discussion.

In certain special cases, it is possible to determine the exact solution of the system analytically. In general, however, determining solutions requires the use of numerical approximation; this approach provides us with an approximate solution which simulates an exact solution. The key to computing solutions numerically is the ability to discretize time. Time is continuous, but computers only work with discrete values. Thus, a step size *h* is chosen, and the desired time interval is divided into points which are exactly one time step apart so that

$$t_{n+1} = t_n + h. \tag{8}$$

This effectively divides the entire solution into discrete points because

$$y_1 \approx y(t_1) = y(t_o + h)$$

$$y_2 \approx y(t_2) = y(t_o + 2h) \tag{9}$$

$$\vdots$$

$$y_n \approx y(t_n) = y(t_o + nh).$$

Here, $y(t_n)$ is the exact solution at time t_n, whereas y_n is merely an approximation. Once this has been done, the solution at can be computed one point at a time.

For any two points $y_n$and y_{n+1} which are exactly one time step apart, there must be a slope m between them, defined as the change in position over change in time, which satisfies the relationship

$$y_{n+1} = y_n + mh. \tag{10}$$

Consequently, if we know the initial point, and we know the slope between the two points for a given time step, we can use this relationship to extrapolate where the second point must lie. We know that the exact slope between two points on a smooth curve is the average of the first derivative over the time interval between them. Our defining equations can tell us the derivative at any point; the problem is that we cannot possibly know the derivative at every time on that interval because it would require calculating an infinite number of values. Fortunately, just as we already discretized time, we can approximate a continuous average using a discrete average. Thus, we can compute an approximate slope between a known point and unknown point on our solution, and use the slope to determine the location of the second point.

The procedure used to calculate the approximated slope from the differential equations is the defining characteristic of a particular numerical method of this form. For example, the Forward Euler method is the simplest method; as shown in Figure 4 below, it assumes that the derivative evaluated at an initial point is equal to the slope

between that initial point and the unknown point one time step in the future. In other words, given our initial value problem in Equation 6, this method would set m equal to $f(y_n)$ and use Equation 10 to calculate y_{n+1}. Thus, the numerical solution is the solution which passes through (t_{n+1}, y_{n+1}), as shown by the red line in Figure 4.

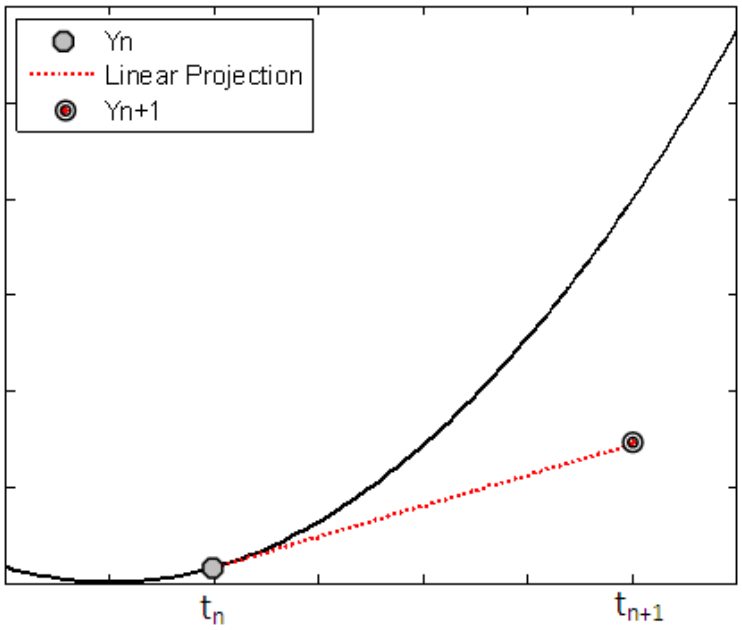


Figure 4: Illustration of the Forward Euler method. Time is represented on the horizontal axis, and Y is represented on the vertical axis. The black curve shown is the exact solution which passes through (t_n, y_n).

Notice that this approach also works for systems of equations. Using the Forward Euler method on the initial value problem in Equation 7, we would set m_i equal to $f_i(y_n)$ for each $i = 1,2, \ldots N$, and then calculate the *N* values of y_{n+1}using Equation 10. In that case, our second point will be at

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_n + h\, \boldsymbol{F}(\boldsymbol{Y}_n). \tag{11}$$

This second point becomes part of the numerical solution, and it is then used as the 'initial point' for the next iteration.

This method works reasonably well for systems in which the derivatives experience little variation, or very gradual variation. However, if the derivatives change

significantly between discrete time points, the Forward Euler method has no way to detect it or account for it. Thus, the Forward Euler method's error can be very high, and it is not reliable enough for our purposes. Instead, we decided to use the fourth order Runge-Kutta method and the second order Runge-Kutta method; both of these are somewhat similar to the Forward Euler method, but more advanced.

4.2. Fourth Order Runge-Kutta Method

While we are still unable to compute a continuous average, we can improve our slope approximation by increasing the number of values used to compute the discrete average. Unlike the simplistic Forward Euler method, which used only one value, the classical fourth order Runge-Kutta method uses a weighted average of four values—one from the initial point and three from distinct intermediate points. These three intermediate points help to account for variations in the function *F* and significantly improve the accuracy of the solution.

Consider again the initial value problem in Equation 7. Using this method, $\boldsymbol{Y}_{n+1}$ is calculated based on the previous point, $\boldsymbol{Y}_n$, according to the formula

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_n + h\left(\frac{\boldsymbol{k_{n1}}+2\boldsymbol{k_{n2}}+2\boldsymbol{k_{n3}}+\boldsymbol{k_{n4}}}{6}\right), \tag{12}$$

where the $\boldsymbol{k}_{ni}$ are vectors of derivatives given by

$$\begin{aligned} \boldsymbol{k}_{n1} &= \boldsymbol{F}(t_n, \boldsymbol{Y}_n) \\ \boldsymbol{k}_{n2} &= \boldsymbol{F}(t_n + \tfrac{1}{2}h, \boldsymbol{Y}_n + \tfrac{1}{2}h\boldsymbol{k}_{n1}) \\ \boldsymbol{k}_{n3} &= \boldsymbol{F}(t_n + \tfrac{1}{2}h, \boldsymbol{Y}_n + \tfrac{1}{2}h\boldsymbol{k}_{n2}) \\ \boldsymbol{k}_{n4} &= \boldsymbol{F}(t_n + h, \boldsymbol{Y}_n + h\boldsymbol{k}_{n3}). \end{aligned} \tag{13}$$

Although the Lorenz equations define an autonomous system, we have presented the relationships in Equation 12 in the nonautonomous form in order to explicitly illustrate how time acts within these relationships.

The phase space coordinates of the first intermediate point are calculated only half a time step ahead of the known point, and they are found using the Forward Euler method. This is process is shown in Figure 5 below for the N=1 case.

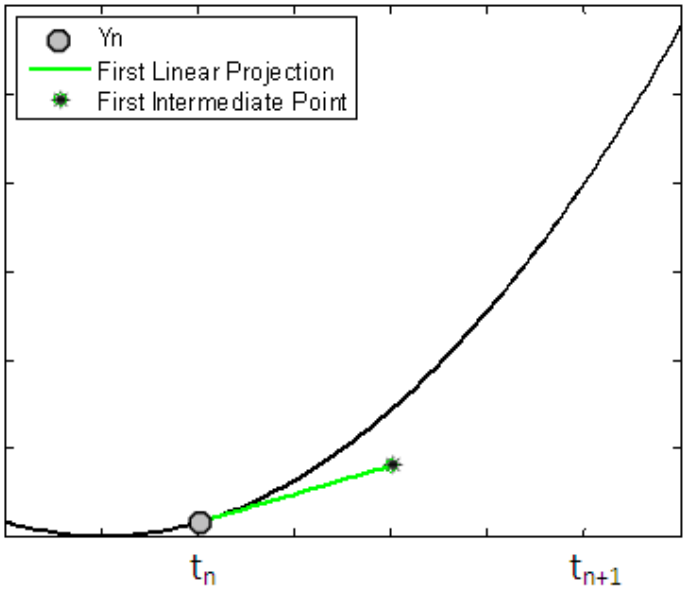


Figure 5: The first step of the Runge-Kutta method for a one dimensional system. This step is effectively identical to the Forward Euler method shown in Figure 4 except here the step size is h/2 instead of h. Again, the black curve shown is the exact solution which passes through (t_n, y_n).

The value of y on the line at time $t_n + \frac{1}{2}h$ is used to evaluate the derivative, and this derivative is labeled k_{n2}. Next, a line with slope equal to k_{n2} is projected from (t_n, y_n); the value of y at time $t_n + \frac{1}{2}h$ is again used to evaluate the derivative, and this derivative is labeled k_{n3}. A third line is projected from (t_n, y_n), this time with slope equal to k_{n3}. The value of y at time $t_n + h$ is used to evaluate the derivative, and this derivative is labeled k_{n4}. Finally, the values of k_{n1}, k_{n2}, k_{n3} and k_{n4} are used in Equation 12 to calculate $\boldsymbol{Y}_{n+1}$.

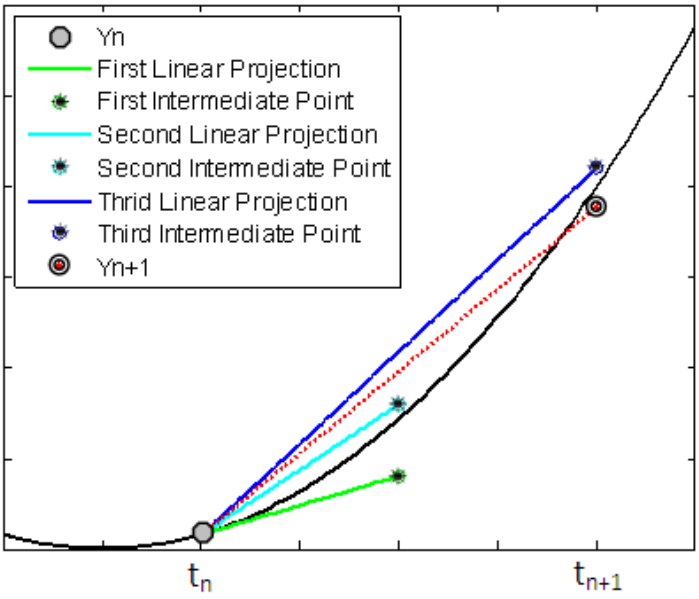


Figure 6: Full Runge-Kutta method. Time is represented on the horizontal axis, and Y is represented on the vertical axis.

Calculating the derivatives at the first and second intermediate points helps to determine if the values of the function *F* changes between the initial point and the midpoint. The third point performs a similar role. If the function *F* increases over the interval, the derivatives evaluated at these intermediate points will increase as well, and the weighted slope in Equation 12 will be increased accordingly. The derivatives at the intermediate points would similarly adjust for function decreases along the interval. This interaction results in significantly higher accuracy than found in the Forward Euler and similar methods.

The calculation of $\boldsymbol{Y}_{n+1}$ concludes one iteration of this method. Notice that $\boldsymbol{Y}_n$ and the $\boldsymbol{Y}_{n+1}$ are the only points in this process which are actually part of the numerical solution; the other three intermediate points are temporary and are discarded once the process is complete. To perform another iteration, $\boldsymbol{Y}_{n+1}$ would become the "initial point" and the entire proceeding process would be repeated.

4.3. Second Order Runge-Kutta Method

The second order Runge-Kutta method we selected, defined as follows, is a drastically simplified version of the fourth order method described above. Using it,

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_n + h(\boldsymbol{k_{n2}}) , \tag{14}$$

where $\boldsymbol{k_{n2}}$ is derived the same as above. Given its simplistic nature, it is more accurate than the Forward Euler method but significantly less accurate than the classical fourth order Runge-Kutta method.

4.4. Error

Regardless of the relative error between methods, it is important to note that numerical computations are inherently imprecise. Every $\boldsymbol{Y}_n$other than $\boldsymbol{Y}_o$is not part of the exact solution. Therefore, the computation of each new point generates a tiny error, and these errors build upon each other. We can attempt to minimize the error by reducing the time step size, but it is impossible to completely eliminate the error while keeping the time interval discrete.

We already know that any two solutions with slightly different initial conditions will have different long-term behavior. However, it is important to note that the iterative computational process treats each point in the numerical solution as if it were the initial value. Thus, the error introduced into a system by the discrete nature of the computation could potentially affect the solution as if the initial conditions had been different. If two different step sizes or two different methods are used, the error

contributed will be different, and it seems reasonable that their long-term solutions will differ from each other accordingly.

5. Numerical Experiments: Variations in Step Size

To test how the size of the time step affects the resultant solution, a MATLAB function was written which solved the Lorenz system using the fourth order Runge-Kutta method, given a specified time step. The code can be seen in Appendix A. The initial point was set to (1,1,1), and a total time was set to 50 for each step size. These parameters allowed the solution to easily settle into its familiar pattern and ensured that the long-term effects had enough time to emerge. Data sets were produced for steps sizes of 0.01, 0.001, 0.0005, 0.0002, and 0.0001. Because of the accuracy of the fourth order Runge-Kutta method, we shall consider the solution with h=0.0001 as the standard with which to compare the other solutions; we will refer to it often in the following.

When the data are plotted, it is clear that any two solutions with different step sizes eventually follow different trajectories. An example can be seen in Figure 7 below.

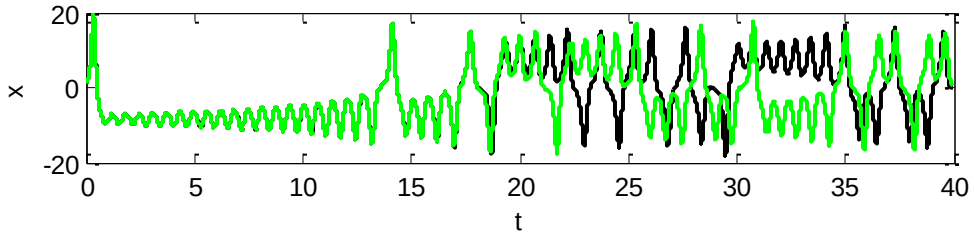


Figure 7: Divergence of solutions with different step sizes.
The black series has h=0.001. The green series has h=0.0001.

Before the solutions diverge, however, there is a certain time frame over which the solutions are very near to each other, despite the step size variation. Consequently, models can have relatively high accuracy during this initial period. When solutions from multiple step sizes are plotted together, as in Figure 8, it also becomes apparent that

there is a connection between the relative step sizes of two solutions and the elapsed time before they diverge.

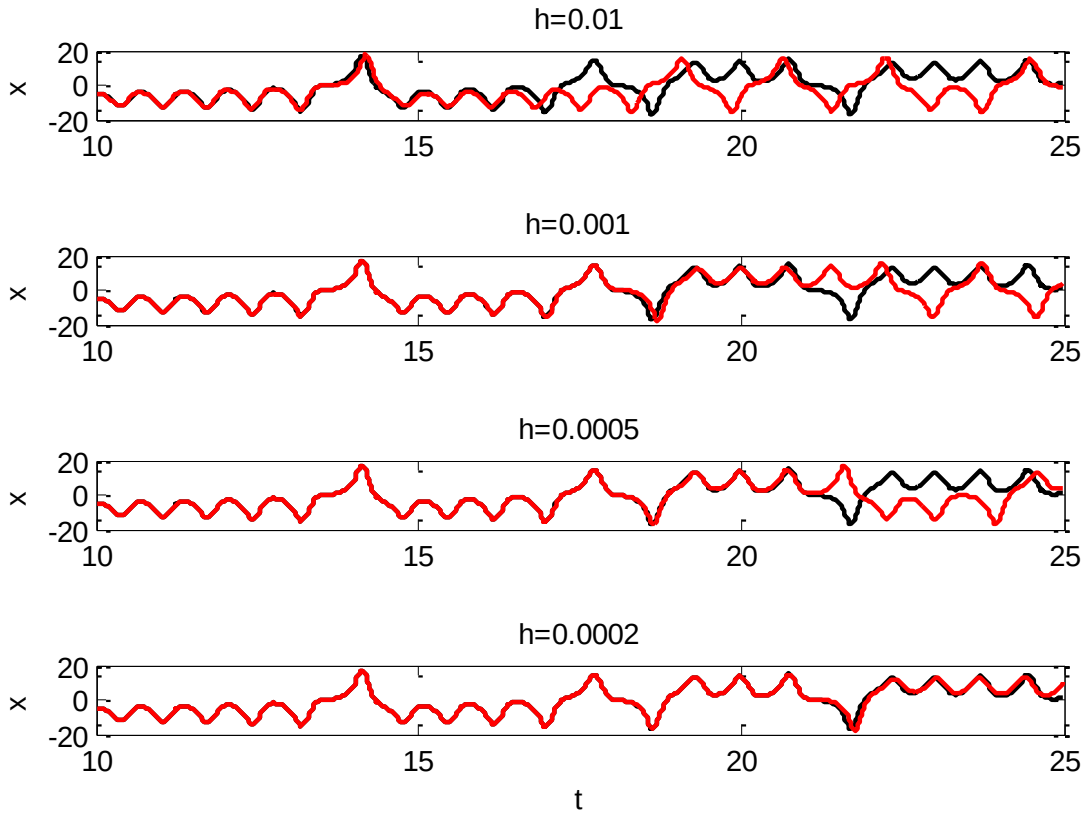


Figure 8: Relative step size.
The red series have the step size indicated; the black series has a step size of 0.0001.

In the figure above, the relative step size decreases from top to bottom. Clearly, data with closer step sizes produce solutions which remain similar for a longer time. Indeed, the solutions with the two smallest size sizes are almost identical on the interval shown. However, even these two solutions diverge shortly after the displayed interval. Reducing the relative time step lengthens the time period during which the solutions are similar, but these solutions are still prone to the long-term chaotic effects.

6. Numerical Experiments: Variations in Method

To determine whether the numerical method significantly affects the resultant solutions, a second MATLAB function was written using the second order Runge-Kutta method to solve the system. This code can be seen in Appendix B. The initial point and total time frame were the same as above, and data was computed for the same set of step sizes.

When plotted, the initial behavior produced by the second order Runge-Kutta method seems very similar to the data produced by the fourth order method, as illustrated by Figures 9 and 10 below. For data sets with the same step size, any deviations were minor, and such deviations do not seem to cause the solutions to diverge sooner or more wildly.

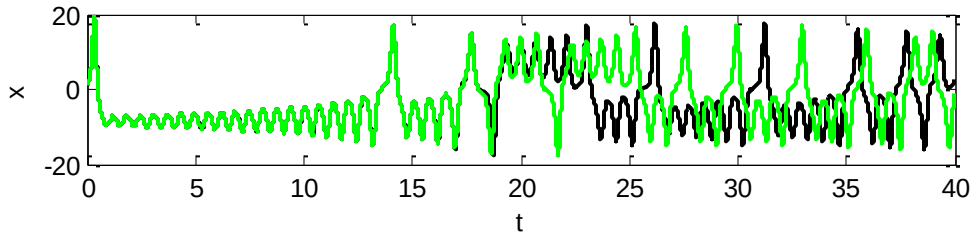


Figure 9: Divergence of solutions using the second order Runge-Kutta method. The black series has h=0.001. The green series has h=0.0001.

Upon further examination of the two closest step sizes, the similarities remain evident. As seen in Figure 11, the pointwise deviations of the second order solution are clearly larger, but they follow a nearly identical pattern to the fourth order solution, and do not seem prone to increased instability.

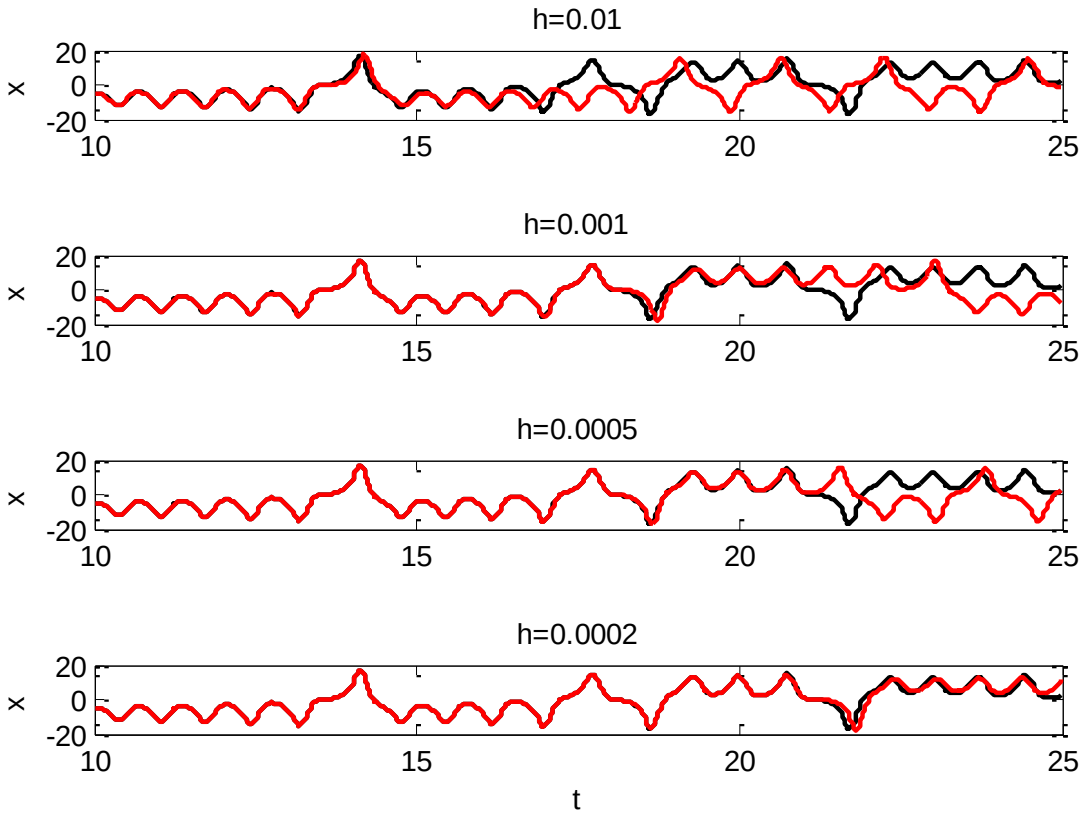
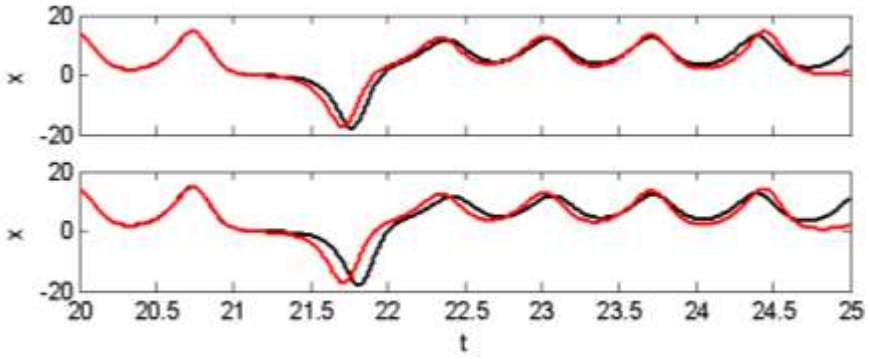


Figure 10: Relative step size using the second order Runge-Kutta method.
The red series have the step size indicated; the black series has a step size of 0.0001.

Figure 11: A closer look at h=0.0002 and h=0.0001.
Above is the solution produced by the fourth order Runge-Kutta method.
Below is the solution produced by the second order Runge-Kutta method.



However, the long-term behavior reveals the differences between the numerical methods. See Figure 12 below.

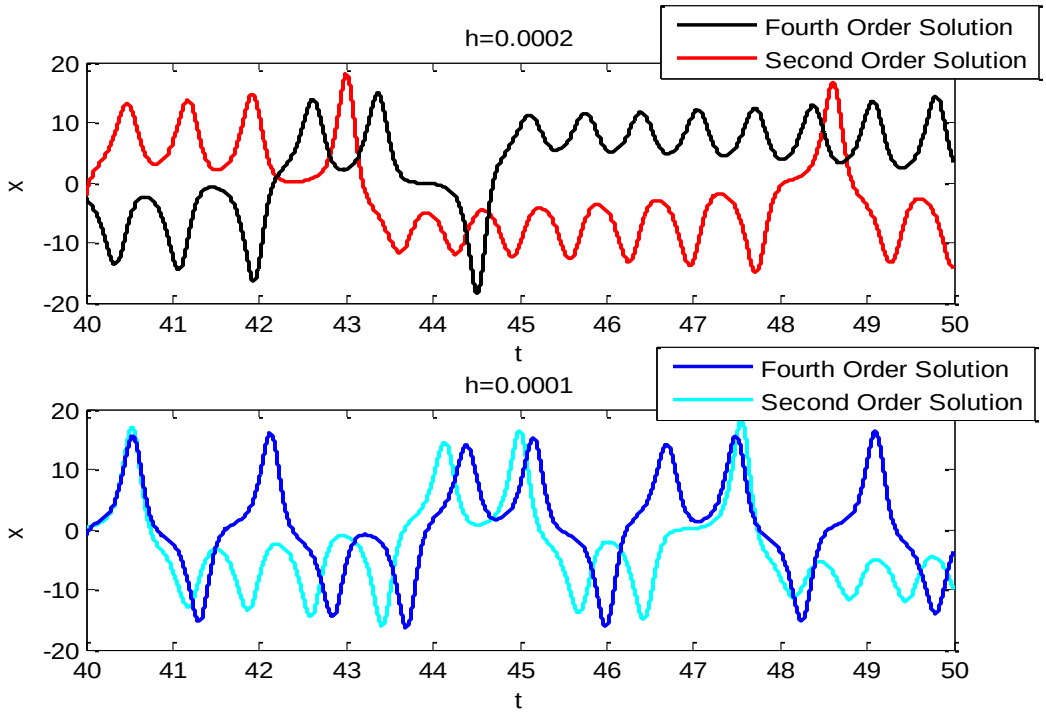


Figure 12: Long-term behavior of solutions.

This suggests that the difference in numerical method is significant. However, the difference between the solutions caused by using two different methods with h=0.0001 was not as significant as the difference caused by using the same method and time steps h=0.0002 and h=0.0001. Thus, in this case, the choice of step size has a greater effect on the solutions than the choice of method.

7. Conclusion

As Lorenz discovered half a century ago, solutions to chaotic systems are sensitive to variations in the system's initial conditions. In reality, however, these systems are sensitive to other variations as well. In this investigation, we maintained constant initial conditions while we varied the numerical step size, and we found that long-term trajectories of solutions with different step sizes had the same qualitative behavior but eventually diverged from each other. Still using the same initial conditions, we then varied the type of numerical method used to solve the system; again, we found that the long-term solutions diverged while maintaining the same qualitative behavior. In both cases, the system exhibited sensitivity despite the fact that the initial conditions had been held constant. Thus, we must conclude that chaotic systems are sensitive to numerical method and step size as well as initial conditions. On the other hand, the long-term qualitative behavior can still be computed with good accuracy despite this sensitivity to step size and method.

If these conclusions hold for all other numerical methods, they would have broad philosophical implications regarding our ability to completely understand classical physical systems. Even if we could determine the exact initial conditions of a particular chaotic system, and thus avoid complications arising from sensitivity to initial conditions, our numerical computation process would still destroy our ability to calculate an exact solution. Under such conditions, we would no longer be able to view chaotic systems as being deterministic; an exact solution would exist, but we would simply not have the means to determine that solution with long-term pointwise

accuracy. From a physical perspective, however, we can still determine the average behavior of such systems regardless of whether the system is deterministic. Consequently, we are able to tease certain information out of a chaotic system despite the effects of chaos.

8. References

American Physical Society. (2003, January). *This Month in Physics History; Circa January 1961: Lorenz and the Butterfly Effect.* Retrieved March 15, 2011, from APS News: http://www.aps.org/publications/apsnews/200301/history.cfm

Boyce, W. & DiPrima, R. (2005). *Elementary Differential Equations and Boundary Value Problems* (8th ed.). Hoboken, New Jersey: John Wiley & Sons.

Hirsch, M. W., Smale, S., & Devaney, R. L. (2004). *Differential Equations, Dynamical Systems, and an Introduction to Chaos.* San Diego, California: Elsevier.

Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences, 20,* 130–141.

Lorenz, E. N. (1995). *The Essence of Chaos*. University of Washington Press.

Wiggins, S. (2003). *Introduction to Applied Nonlinear Dynamical Systems and Chaos.* New York, New York: Springer-Verlag New York, Inc.

9. Acknowledgements

Much of the material in sections 2 and 3 of this paper is a collection of ideas from Boyce & DiPrima, Hirsch, Smale, & Devaney, and Wiggins.

Appendices

Appendix A: MATLAB code for the Fourth Order Runge-Kutta method.

The data generation process was divided into two different functions. The function `hvardata` specified the initial point and the total time interval; then it called `lorenzdata` to compute the iterations.

```
function [f]= hvardata(h)
%%Description: This function specifies the initial point
%and then uses lorenzdata.m to compute a solution to the
%lorenz system given a particular time step.

%These data need to be compiled by a different program.

%%Data Calculation
%Define initial values
x1=1;
y1=1;
z1=1;
T=50;

%Calculate Data
[xh1,yh1,zh1]=lorenzdata(x1,y1,z1,h,T);
t1=(h:h:T)';
f=[xh1,yh1,zh1,t1];
end

function [x,y,z] = lorenzdata(x1,y1,z1,h,T)
%LORENZDATA Takes the initial values, the desired time
%step, h, and the desired total time, T, and computes data
%for those parameters using the Lorenz equations. Used by
%hvardata.

%The x,y, and z values are stored in matrices with those
%names.  The derivatives at every time step are stored in
%matrices called Fx, Fy, and Fz. Sigma, tau, and beta are
%fixed.

%Set constants
sigma=10;
tau=28;
beta=8/3;

%----%
%Lorenz Equations
fx=inline('sigma*(Y-X)');    %fx(X,Y,sigma) = sigma*(Y-X)
fy=inline('X*(tau-Z)-Y');    %fy(X,Y,Z,tau) = X*(tau-Z)-Y
fz=inline('X*Y-beta*Z');     %fz(X,Y,Z,beta) = X*Y-beta*Z

%Establish future matrices
x(1,:)=x1;
y(1,:)=y1;
z(1,:)=z1;
```

```
Fx=fx(x1,y1,sigma);
Fy=fy(x1,y1,z1,tau);
Fz=fz(x1,y1,z1,beta);

t(1,:)=h;

%Calculate future points

for n=1:T/h-1
    %Calculate time
    t(n+1,:)=t(n,:)+h;

    %Calculate (temporary) intermediate points
    X1=x(n,:);
    Y1=y(n,:);
    Z1=z(n,:);

    X2=x(n,:)+(h/2)*Fx(n,:);
    Y2=y(n,:)+(h/2)*Fy(n,:);
    Z2=z(n,:)+(h/2)*Fz(n,:);

    X3=x(n,:)+(h/2)*fx(X2,Y2,sigma);
    Y3=y(n,:)+(h/2)*fy(X2,Y2,Z2,tau);
    Z3=z(n,:)+(h/2)*fz(X2,Y2,Z2,beta);

    X4=x(n,:)+h*fx(X3,Y3,sigma);
    Y4=y(n,:)+h*fy(X3,Y3,Z3,tau);
    Z4=z(n,:)+h*fz(X3,Y3,Z3,beta);

    XX=[X1;X2;X3;X4];
    YY=[Y1;Y2;Y3;Y4];
    ZZ=[Z1;Z2;Z3;Z4];

    %Calculate next (permanent) point
    x(n+1,:)=x(n,:)+(h/6)*(fx(X1,Y1,sigma)
+2*fx(X2,Y2,sigma)   +2*fx(X3,Y2,sigma)
+fx(X4,Y4,sigma));
    y(n+1,:)=y(n,:)+(h/6)*(fy(X1,Y1,Z1,tau)
+2*fy(X2,Y2,Z2,tau)  +2*fy(X3,Y3,Z3,tau)
+fy(X4,Y4,Z4,tau));
    z(n+1,:)=z(n,:)+(h/6)*(fz(X1,Y1,Z1,beta)
+2*fz(X2,Y2,Z2,beta) +2*fz(X3,Y3,Z3,beta)
+fz(X4,Y4,Z4,beta));

    %Calculate derivative at next point
    Fx(n+1,:)=fx(x(n+1,:),y(n+1,:),sigma);
    Fy(n+1,:)=fy(x(n+1,:),y(n+1,:),z(n+1,:),tau);
    Fz(n+1,:)=fz(x(n+1,:),y(n+1,:),z(n+1,:),beta);

end

end
```

Appendix B: MATLAB code for the Second Order Runge-Kutta method.

As above, the function `hvardataRK2` specified the initial point and the total time interval; then it called `lorenzdataRK2` to compute the iterations.

```
function [f]= hvardataRK2(h)
%%Description: This function specifies the initial point
%and then uses lorenzdataRK2.m to compute a solution to the
%Lorenz system given a particular time step.

%These data need to be compiled by a different program.

%%Data Calculation
%Define initial values
x1=1;
y1=1;
z1=1;
T=50;

%Calculate Data
[xh1,yh1,zh1]=lorenzdataRK2(x1,y1,z1,h,T);
t1=(h:h:T)';
f=[xh1,yh1,zh1,t1];

end


function [x,y,z] = lorenzdataRK2(x1,y1,z1,h,T)
%LORENZDATA Takes the initial values, the desired time
%step, h, and the desired total time, T, and computes data
%for those parameters using the Lorenz equations. Used by
%hvardataRK2.

%The x,y, and z values are stored in matrices with those
%names.  The derivatives at every time step are stored in
%matrices called Fx, Fy, and Fz. Sigma, tau, and beta are
%fixed.

%Set constants
sigma=10;
tau=28;
beta=8/3;

%----%

%Lorenz Equations
fx=inline('sigma*(Y-X)');    %fx(X,Y,sigma) = sigma*(Y-X)
fy=inline('X*(tau-Z)-Y');    %fy(X,Y,Z,tau) = X*(tau-Z)-Y
fz=inline('X*Y-beta*Z');     %fz(X,Y,Z,beta) = X*Y-beta*Z

%Establish future matrices
x(1,:)=x1;
y(1,:)=y1;
```

```
z(1,:)=z1;

Fx=fx(x1,y1,sigma);
Fy=fy(x1,y1,z1,tau);
Fz=fz(x1,y1,z1,beta);

t(1,:)=h;

%Calculate future points

for n=1:T/h-1
    %Calculate time
    t(n+1,:)=t(n,:)+h;

    %Calculate (temporary) intermediate points
    X1=x(n,:);
    Y1=y(n,:);
    Z1=z(n,:);

    X2=x(n,:)+(h/2)*Fx(n,:);
    Y2=y(n,:)+(h/2)*Fy(n,:);
    Z2=z(n,:)+(h/2)*Fz(n,:);

    %Calculate next (permanent) point
    x(n+1,:)=x(n,:)+(h)*(fx(X2,Y2,sigma));
    y(n+1,:)=y(n,:)+(h)*(fy(X2,Y2,Z2,tau));
    z(n+1,:)=z(n,:)+(h)*(fz(X2,Y2,Z2,beta));

    %Calculate derivative at next point
    Fx(n+1,:)=fx(x(n+1,:),y(n+1,:),sigma);
    Fy(n+1,:)=fy(x(n+1,:),y(n+1,:),z(n+1,:),tau);
    Fz(n+1,:)=fz(x(n+1,:),y(n+1,:),z(n+1,:),beta);

end

end
```