

B.A. Project CO34 – Chaos

Candidate number: 04666

Trinity Term 2002

Abstract

This project aims to investigate the Lorenz Equations, and properties thereof, using the Runge-Kutta method to produce solutions to the Lorenz Equations, and to then display the solutions graphically. It aims to reproduce some of the work Edward Lorenz produced when he was investigating the properties of these equations, and also attempt to explore the equations by varying their parameters

1 Introduction and mathematical principles

1.1 The Definition of Chaos

The generally accepted definition of chaos is

“Chaos is aperiodic long-term behaviour in a deterministic system that exhibits sensitive dependence on initial conditions”

1. **Aperiodic long-term behaviour** means that the trajectories (i.e. paths that describe the system) do not settle down to fixed points or periodic orbits as $t \rightarrow \infty$.
2. **Deterministic system** means that the system has no random or noisy inputs or parameters. The irregularity arises solely from the system's non-linearity, not from driving forces. [9]
3. **Sensitive dependence on initial conditions** means that nearby trajectories separate exponentially fast, i.e. the system has a positive Lyapunov exponent (see section 3.2). [10]

1.2 Lorenz Equations

The Lorenz equations, named after Ed Lorenz who first derived them[5], are

$$\frac{dy_1}{dt} = a(y_2 - y_1), \quad (1)$$

$$\frac{dy_2}{dt} = ry_1 - y_2 - y_1y_3, \quad (2)$$

$$\frac{dy_3}{dt} = y_1y_2 - by_3, \quad (3)$$

where a, r and b are positive parameters, were derived as a truncation of a partial differential equation for fluid convection [6]. They were used to attempt to model the Earth's atmosphere, where a flat fluid layer (or the atmosphere in this case) is heated from below and cooled from above. The heating from below is due to the Earth being warmed by sunlight, and the cooling from above comes from the atmosphere losing heat to space. In the resultant convective motion, y_1 represents the convective motion, y_2 the horizontal temperature variation, and y_3 the vertical temperature variation. The parameters a , r and b are proportional respectively to the Prandtl number¹, the Rayleigh number² and the size of the region whose behaviour is being approximated by the Lorenz equation system. These equations tend to exhibit chaotic behaviour, and for certain values of a , r and b , a strange attractor (defined in the next couple of paragraphs).

The Lorenz equations have stationary points where $\frac{dy_n}{dt} = 0$ at the points

$$y_1 = y_2 = \pm\sqrt{b(r-1)} \quad , \quad y_3 = r-1, \quad \text{and} \\ y_1 = y_2 = y_3 = 0.[11]$$

The approach to these solutions from an arbitrary starting point depends on the value of r ; if $r < 1$, the only real stationary point is the origin, as shown in figures 1 and 2. [7] [1]. The figures were obtained using the program I wrote, described in section 2.2.

¹The Prandtl number is a dimensionless number relating the ratio of a fluid's capacity to diffuse momentum to its capacity to diffuse heat. Explicitly, the Prandtl number is $Pr = \frac{C_p\mu}{\kappa}$ in which C_p is the heat capacity, μ is the viscosity and κ is the thermal diffusivity.

²The Rayleigh number is defined as $Ra = \frac{g\alpha H^3 \Delta T}{\nu\kappa}$, where $\nu = C_p\mu$ (as defined above), g is gravitational acceleration, α is the coefficient of thermal expansion, ν is the kinematic viscosity, κ is the thermal conductivity, ΔT is a temperature difference, and H is the depth of the system being considered

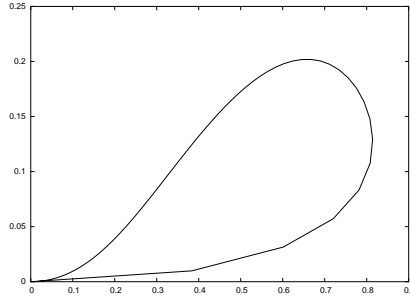


Figure 1: Plot of y_3 against y_2 , with $a = 10$, $b = \frac{8}{3}$, $r = 0.5$, and a starting position of $(0, 1, 0)$. The path ends up on the origin, the only stable point.

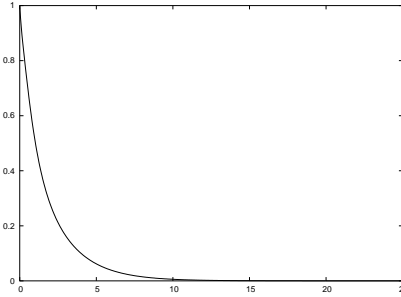


Figure 2: Plot of y_2 against t (where t is time), with $a = 10$, $b = \frac{8}{3}$, $r = 0.5$, and a starting position of $(0, 1, 0)$. As can be seen, y_2 tends exponentially to the origin.

If $r > 1$, then all three stationary points exist, but the origin is unstable, so it will be never reached, unless the system is started at the stationary points. An example is shown in figures 3 and 4 (again, these figures were obtained using the program I wrote, described in section 2.2). The manner of approach to the other two solutions depends, again, on the value of r . The system becomes less and less certain of which one it has chosen as r is increased, until for $r \gtrsim 24$ it will never converge on either, but will wander for ever between their neighbourhoods. This is the behaviour that is referred to as a *strange attractor* – it is bounded, but non-periodic (see, for example, figures 5 and 6, which were, again generated by my program, described in section 2.2).

Lorenz considered the equations with the values of $a = 10$, $b = \frac{8}{3}$, and $r = 28$. [12]

The Lorenz equations can be used to illustrate why weather forecasting is a difficult problem. The equations of motion of the atmo-

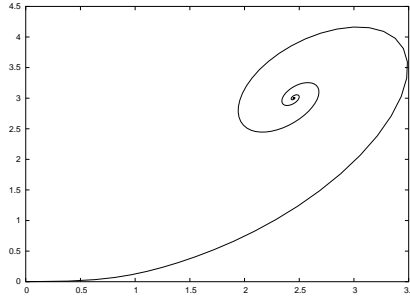


Figure 3: Plot of y_3 against y_1 , with $a = 10$, $b = \frac{8}{3}$, $r = 4$, and a starting position of $(0, 1, 0)$. As can be seen, the trajectory tends to a stationary point.

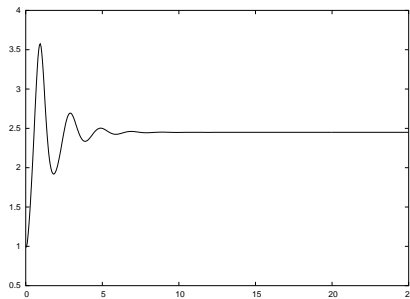


Figure 4: Plot of y_2 against t (time), with $a = 10$, $b = \frac{8}{3}$, $r = 4$ and a starting value of $(0, 1, 0)$. It can be seen that y_2 oscillates for a few cycles (with the amplitude decaying exponentially), eventually settling down to a constant value, around 2.5.

sphere are far more complex[13], but they show a qualitatively similar behaviour, and meteorologists are interested in the details of the system; for example, the variation of temperature and pressure with time. The chaotic nature of such equations can be demonstrated with the following example.

Figure 7 shows a plot of y_1 against t for $r = 28$ from an initial point $(4, 5, 6)$. Figure 8 shows the same variables for the same r , but with a ‘measurement error’ added, by moving the initial point to $(4.01, 5.01, 6.01)$. Both figures were obtained using my program, described in section 2.2. The figures have very similar shapes, up to a point $t \approx 11$, when they diverge a lot. Likewise, when plotting y_2 against y_3 , for the initial point $(4, 5, 6)$ (figure 9), and comparing it to the same plot for an initial point $(4.01, 5.01, 6.01)$ (figure 10), noticeable differences occur, even though the plots both have the same

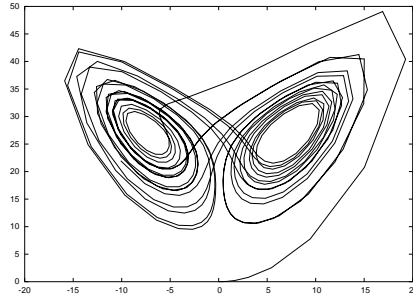


Figure 5: Plot of y_3 against y_1 , with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$. As can be seen, a ‘butterfly’ pattern emerges – the path is in either one loop or the other, but does not tend to a fixed point.

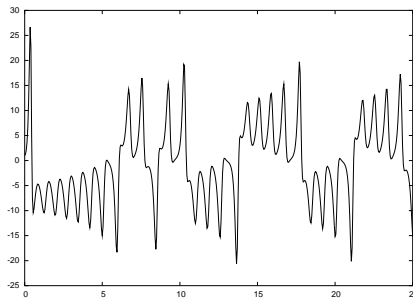


Figure 6: Plot of y_2 against t (time), with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$. The value of y_2 varies seemingly randomly, not tending to any particular value.

basic shape.

For the purposes of the rest of this project, all initial points (unless specified otherwise) are located at $(0, 1, 0)$.

2 Numerical method and computational solution

The numerical method employed in the solution to the Lorenz equations is the Runge-Kutta method[2], for which I have written a program in Fortran (F77). The program outputs sets of numbers that are suitable for use by GNUPlot, which I have used to plot the data.

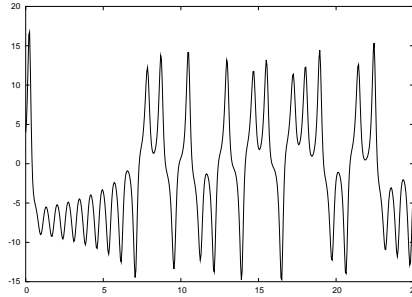


Figure 7: Plot of y_1 against time t for $r = 28$ ($a = 10$, $b = \frac{8}{3}$) with a starting value of $(4, 5, 6)$. It can be seen that y_1 varies seemingly randomly, without tending to any particular value.

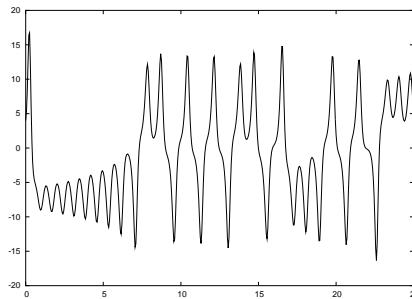


Figure 8: Plot of y_1 against time t for $r = 28$ ($a = 10$, $b = \frac{8}{3}$) with a starting value of $(4.01, 5.01, 6.01)$. It can be seen that y_1 also varies seemingly randomly, but with fairly noticeable differences compared to figure 7

2.1 The Runge-Kutta method

The Runge-Kutta method provides a reliable, if non-optimal, method of obtaining numerical solutions to differential equations. The algorithm that is used to solve the Lorenz equations is now described.

The Lorenz equations (1-3) are expressed as

$$\frac{dy_i}{dt} = f_i(y_1, y_2, y_3). \quad (4)$$

Starting from the values of y_i at some time t , I compute the values of y_i at a slightly later time $t + \delta t$. By repeated application of this process, I obtain solutions at times $t + m\delta t$ (m is an integer). Each step is identical. I let $y_{i,0}$ be the values of y_i at the beginning of the step, at time t , and $y_{i,4}$ be the values at the end of the step, i.e. at time $t + \delta t$.

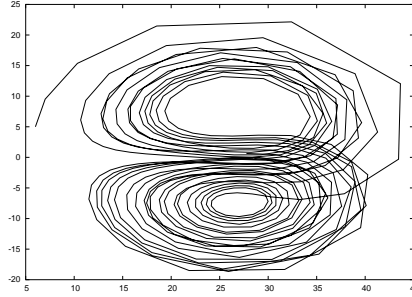


Figure 9: Plot of y_2 against y_3 for $r = 28$ ($a = 10$, $b = \frac{8}{3}$) with a starting value of $(4, 5, 6)$. An ‘butterfly-like’ pattern emerges, the trajectory is in either one loop or the other, but never tends to a fixed value.

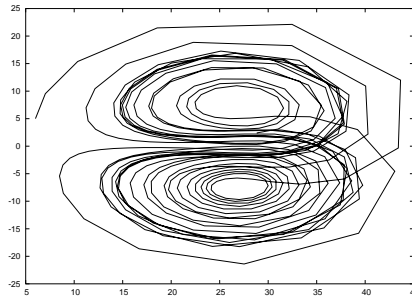


Figure 10: Plot of y_2 against y_3 for $r = 28$ ($a = 10$, $b = \frac{8}{3}$) with a starting value of $(4.01, 5.01, 6.01)$. A similar pattern to the one in figure 9 emerges, but with some quite visible differences.

Several intermediate values of the f_i s between t and $t + \delta t$ are required. All components of y_i and f_i must be calculated at each intermediate point. The steps taken are as follows.

1. Calculate $f_{i,0}$ using the initial values $y_{i,0}$.
2. Calculate $f_{i,1}$ using $y_{i,1} = y_{i,0} + f_{i,0} \frac{\delta t}{2}$.
3. Calculate $f_{i,2}$ using $y_{i,2} = y_{i,0} + f_{i,1} \frac{\delta t}{2}$.
4. Calculate $f_{i,3}$ using $y_{i,3} = y_{i,0} + f_{i,2} \delta t$.
5. Put $y_{i,4} = y_{i,0} + (f_{i,0} + 2f_{i,1} + 2f_{i,2} + f_{i,3}) \frac{\delta t}{6}$.
6. $y_{i,4}$ becomes $y_{i,0}$ in the next step.

To ensure high precision in the calculation, all the variables involved in this algorithm are defined as `double precision`, to use the maximum precision available on whichever computing platform the program is compiled on.

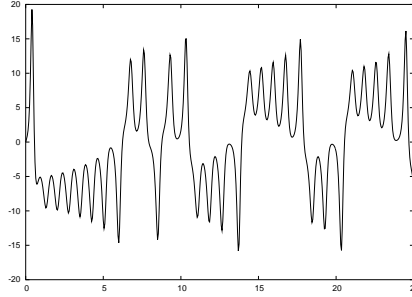


Figure 11: Plot of y_2 against time t , with $r = 28$ ($a = 10$, $b = \frac{8}{3}$), and a starting value of $(0, 1, 0)$. y_2 settles to an irregular aperiodic oscillation as $t \rightarrow \infty$.

2.2 Computational solution

I have written the computational solution to the problem in Fortran. The parameters to the Lorenz equations are set as constant at the beginning of the program to $a = 10$, $b = \frac{8}{3}$ and $\delta t = 0.05$. It asks for a value of n , the number of iterations to be entered, the initial values of y_1, y_2 and y_3 , and the value of r . It also then asks what values are to be plotted along the x and y axes of a graph, respectively (e.g. entering `t y2` would output values suitable for plotting a graph of y_2 against t). The program then outputs the requested values for the x and y axes to a file called `dataset`, which is separately read in by gnuplot, to generate a postscript file containing the graph. I have automated the process slightly by creating a short shell script, and a short gnuplot script, both of which can be seen in Appendix B. The program source code can be seen in Appendix A.

As different aspects of the Lorenz equations were investigated, the program had to be modified slightly to suit the investigation. As the core of the program (the mathematical method) remains the same, I shall only mention the changes that were made to the program, instead of listing the whole program again in the Appendices.

3 Investigation of the Lorenz Equations

3.1 Chaos on a strange attractor

With the starting point of $(0,1,0)$, $a = 10$, $b = \frac{8}{3}$ and $r = 28$, the function $y_2(t)$ looks like figure 11.

As can be seen, after an initial transient, the solution settles into an irregular oscillation that persists as $t \rightarrow \infty$, but never repeats exactly;

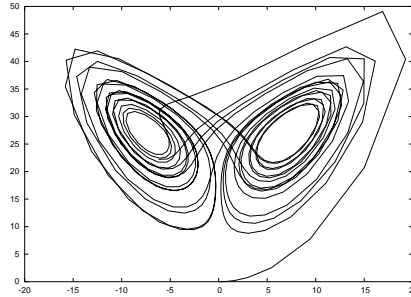


Figure 12: Plot of y_3 against y_2 , with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$. The trajectory lies either on one ‘butterfly wing’, or the other, but does not tend to any value

the motion is aperiodic.

Lorenz found out that if the solution is visualised as a trajectory in phase space, then various interesting structures appear. For example, if y_3 is plotted against y_1 , then a ‘butterfly’-like pattern emerges, as in figure 12.[3]

The trajectory appears to cross itself repeatedly, but that is just an artifact of projecting the 3-dimensional trajectory onto a 2-dimensional plane. In 3 dimensions, no crossings occur (figure 13, plotted using a modification to the program in appendix A which outputs y_1 , y_2 and y_3 values, then fed into `gnuplot` and plotted using `splot`). Moreover, the number of circuits made on either side varies unpredictably from one cycle to the next. The sequence of the number of circuits has many of the characteristics of a random sequence (but the system is chaotic, not random, because it is still governed by a set of equations).[14]

3.2 Exponential Divergence of Nearby Trajectories

The motion of a strange attractor exhibits sensitive dependence on initial conditions, i.e. two trajectories starting very close together will rapidly diverge from each other, and thereafter have totally different futures. The practical implication is that long-term prediction becomes impossible in a system like this, where small uncertainties are amplified very quickly (which is a huge problem in, for example, weather forecasting). [4]

If we take the starting vector $(0,1,0)$, and consider a nearby point $(0+\delta_0, 1,0)$, i.e. start y_1 at a slightly different position, where δ_0 is of

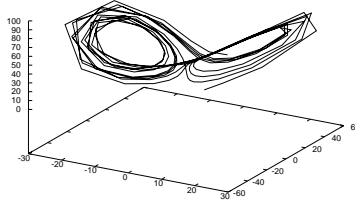


Figure 13: 3d plot of y_1 , y_2 and y_3 , with $a = 10$, $b = \frac{8}{3}$, $r = 60$ and a starting value of $(4, 5, 6)$. The values were chosen because this makes the graph slightly easier to see than using $r = 28$ and a starting value of $(0, 1, 0)$, while still having the properties of a strange attractor. The trajectories do not converge to any point, and also do not cross (which may be slightly difficult to see due to the way the 3D plot is projected onto the page).

the order of 10^{-15} , then, through numerical studies, it is found that

$$|\delta(t)| \sim |\delta_0| \exp^{\lambda t} \quad \text{where } \lambda \text{ is the Lyapunov exponent [8]} \quad (5)$$

That is, neighbouring trajectories separate exponentially fast. Equivalently, if $\ln |\delta(t)|$ against t is plotted, then a curve that is fairly close to a straight line (on average) is obtained, as in figure 14. To obtain the plot, I modified the original program (Appendix A), so that after entering the starting values, δ_0 would have to be entered (in units of 10^{-15}). The program would then, for each iteration, work out the values for $y_{i,n}$ and the values for the slightly offset $y_{i,n}s$, take the difference between $y_{1,n}$ and the offset $y_{1,n}$, and output (to the file `dataset`) values of $\ln |(\text{the difference})|$ and t .

It can be seen from the plot that the curve is not quite straight – this is because the strength of the exponential divergence varies along the attractor.

Using a slightly different plot (1500 iterations, and an initial divergence of 10^{-9} ; figure 15), it can be seen that the divergence stops at some point. This happens when the separation is comparable to the ‘diameter’ of the attractor; the trajectories cannot get further apart than that.[10]

The number λ is often called the *Lyapunov exponent* [8], although this is not technically accurate, as λ depends (slightly) on which trajectory is being looked at. That is, λ is not technically a constant, as it varies by small amounts. To get the true value of λ , an average should be taken over many different points on the same trajectory.

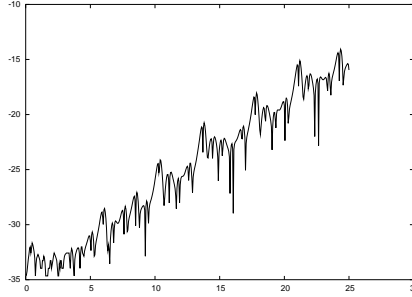


Figure 14: Plot of $\ln |\delta|$ against time t . $a = 10$, $b = \frac{8}{3}$, $r = 28$, and a starting value of $(0, 1, 0)$ on one trajectory, and $(1 \times 10^{-15}, 1, 0)$ on the other. It can be seen that $\ln |\delta|$ against t follows an approximately straight line.

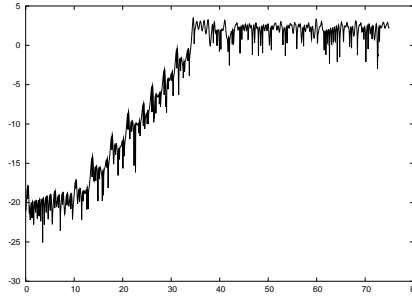


Figure 15: Plot of $\ln |\delta|$ against time t with $a = 10$, $b = \frac{8}{3}$, $r = 28$, and a starting value of $(0, 1, 0)$ on one trajectory, and $(1 \times 10^{-9}, 1, 0)$ on the other. It can be seen that the plot follows a curve up to a point $t \approx 35$, when $\ln |\delta|$ becomes constant.

Moreover, there are actually n different Lyapunov exponents for an n -dimensional system; the λ obtained is actually the largest one of these.

When a system has a positive Lyapunov exponent, then there is a time horizon beyond which prediction breaks down. This is because a prediction is made assuming $|\delta_0| = 0$, whereas in an actual measurement, there $|\delta_0|$ has a finite value. Suppose the initial conditions of an experimental system are measured as accurately as possible. There is still some error in the measurement, so there will always be some error δ_0 between the measurement and the ‘true’ initial state. After a time t , the discrepancy grows to $|\delta(t)| \sim |\delta_0| \exp(\lambda t)$. If, then, we specify a tolerance a before a measurement divergence becomes intolerable – i.e. $|\delta(t)| \geq a$ is intolerable, then the tolerance limit comes (through

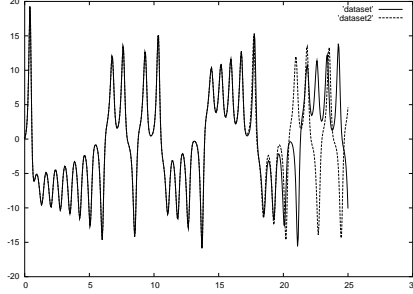


Figure 16: Plot of y_1 against t for two trajectories with a starting value differing by 5×10^{-4}

rearranging equation 5 at

$$t_{\text{max tolerance}} \sim O\left(\frac{1}{\lambda} \ln \frac{a}{|\delta_0|}\right). \quad (6)$$

A good way of illustrating this is in figure 16. Here, the solid line (marked **dataset**) represents the trajectory of y_1 , from a given initial value. The dashed line (marked **dataset2**) represents the trajectory of y_1 , with a slightly offset starting value (in this case, 5×10^{-4}). The graph starts to diverge at $t \approx 17$, with significant divergence starting at $t \approx 19$, so any predictions made on a measurement break down at this point. By the time $t \approx 21$, the two functions are so divergent, that any predictions made will be completely wrong.

As a comparison, I re-created the graph, but using an initial divergence of 1×10^{-4} – i.e. equivalent to a measurement that was five times more accurate. As can be seen from figure 17, the trajectories start very significantly diverging at $t \approx 21$, with very slight divergence starting to occur at $t \approx 19$. This shows that even if the accuracy of measurement increases considerably, the time before the prediction breaks down only increases slightly.

Applying equation 6 to the two situations, taking the tolerance level to be 0.5 (which is fairly noticeable on the graph),

$$\begin{aligned} & \text{For } |\delta_0| = 5 \times 10^{-4} : \\ t_{\text{max tolerance}} & \approx \frac{1}{\lambda} \ln \frac{0.5}{5 \times 10^{-4}} = \frac{3 \ln 10}{\lambda} \end{aligned} \quad (7)$$

$$\begin{aligned} & \text{For } |\delta_0| = 1 \times 10^{-4} : \\ t_{\text{max tolerance}} & \approx \frac{1}{\lambda} \ln \frac{0.5}{1 \times 10^{-4}} = \frac{\ln 5 \times 10^3}{\lambda} = \frac{\ln 5}{\lambda} + \frac{3 \ln 10}{\lambda} \end{aligned} \quad (8)$$

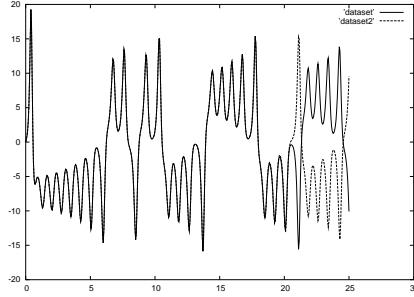


Figure 17: Plot of y_1 against t for two trajectories with a starting value differing by 1×10^{-4}

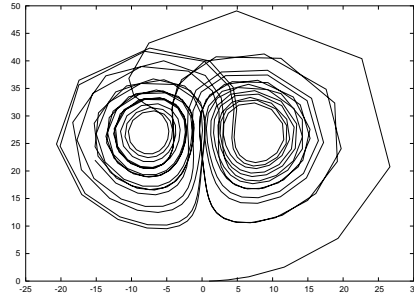


Figure 18: Plot of y_3 against y_2 , with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$

As can be seen, increasing the accuracy by a factor of 5 only increases the time that predictions can be considered accurate by $\frac{\ln 5}{\lambda}$. If we put the value for $t_{\max \text{ tolerance}}$ obtained from the graph in figure 16 ($t \approx 19$) into equation 7, then we get a value of λ as 0.364. Using that in equation 8, we get that the increase in prediction time is 4.43; i.e. only a 23% increase.

3.3 Lorenz Map

Plotting y_3 against y_2 for the Lorenz attractor yields figure 18.

Lorenz wrote

“The trajectory apparently leaves one spiral only after exceeding some critical distance from the centre... It therefore seems that some single feature of a given circuit should predict the same feature of the following circuit.”

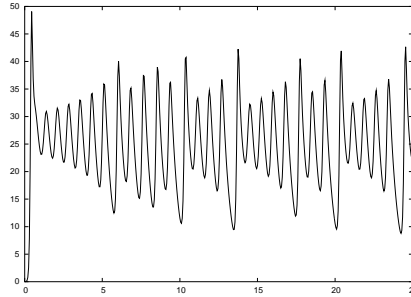


Figure 19: Plot of y_3 against t , with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$. If a maximum is called $y_{3,n}$, then the next maximum along t is $y_{3,n+1}$.

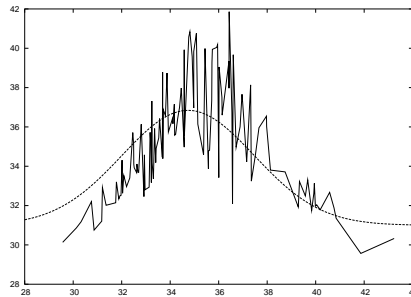


Figure 20: Plot of $y_{3,n+1}$ against $y_{3,n}$, with $a = 10$, $b = \frac{8}{3}$, $r = 28$ and a starting value of $(0, 1, 0)$. The data can be seen following approximately a curve (which is a fitted Gaussian).

The ‘single feature’ that he focused on was $y_{3,n}$, the n ’th local maximum of $y_3(t)$. [15] Plotting y_3 against t gives figure 19.

Lorenz’s idea was that $y_{3,n}$ should predict $y_{3,n+1}$. Indeed, if $y_{3,n+1}$ is plotted against $y_{3,n}$, figure 20 is obtained. The data from the chaotic time series appears to fall fairly neatly on a curve. The function $y_{3,n+1} = f(y_{3,n})$ is called the *Lorenz map*. It should be noted that the curve on the graph has some thickness, so, strictly speaking, $f(y_3)$ is not a well-defined function.

To obtain the data in figure 20, I modified the program in Appendix A to write out all the maxima of y_3 to a file called `zmax`. This was done by comparing the value of y_3 at the end of an iteration to the value of y_3 at the beginning of that iteration – if the new value was smaller, then the old value was a local maximum, and hence written to the file. If this occurred, a flag was set, so the program would then

find the next local minimum value of y_3 (using the same method as finding the maximum, but would stop when the new value of y_3 was larger than the old value). Once the minimum had been found, the program would then proceed to find the next maximum, and so on, until all the iterations were completed. Once all the maximum values for y_3 were found and written to the file `zmax`, the program read the data back from the file, and wrote values of $y_{3,n}$ and $y_{3,n+1}$ to a file called `dataset`. The code to do this is listed below.

```

open (1,FILE='zmax',STATUS='old')
open (2,FILE='dataset',STATUS='unknown')

read (1,*),zn

* Repeat until the file ends
100  read (1,*,END=200),zn1
     write (2,*),zn,zn1
     zn = zn1
     go to 100

200  close(1)
     close(2)

```

`dataset` contains unsorted values, which, if fed directly into `gnuplot` would make a meaningless plot. The easiest way of sorting the data numerically was to use the Unix `sort` command, and put the results into another file (`sort -n dataset > dataset.sorted`), which was to be read by `gnuplot` and used to plot figure 20.

3.4 Exploring Parameter Space

So far, most of this project has concentrated on the parameters used in Lorenz' original work ($a = 10$, $b = \frac{8}{3}$, $r = 28$). [12] Varying these parameters can yield very interesting results; one can find exotic limit cycles tied in knots, pairs of limit cycles linked through each other, intermittent chaos, noisy periodicity, as well as strange attractors.

In figures 22 - 26, I have varied the parameters and produced various plots. The specific parameters used, and variables plotted are captioned under the appropriate plot. All the starting points are at $(0, 1, 0)$.

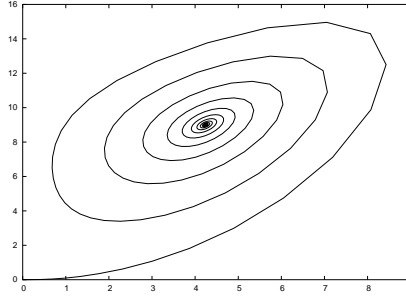


Figure 21: y_3 against y_1 , $a = 10$, $b = \frac{8}{3}$, $r = 10$, and a starting value of $(0, 1, 0)$. It can be seen that the trajectory tends towards a stable point, at around $y_3 = 9$ and $y_1 = 4.5$.

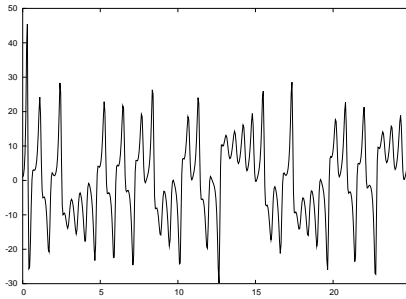


Figure 22: y_2 against time t , $a = 10$, $b = \frac{8}{3}$, $r = 50$, and a starting value of $(0, 1, 0)$. It can be seen that y_2 varies chaotically, not tending to any particular value.

4 Conclusions

The aim of this project was to write a program to solve the Lorenz equations numerically (through use of the Runge-Kutta method), and demonstrate properties of them graphically, thus recreating some of the work that Ed Lorenz did, when investigating the properties of the equations that he devised. As can be seen in the various figures, I have managed to recreate some of his work to a reasonable degree of accuracy. When attempting to solve the equations, using different values of a , b and r , as in Section 3.4, (as a slight extension to Lorenz' original work) I found that whatever values I tried to use, the essential features of the plots always stayed the same, i.e. decaying trajectories looked similar, no matter what parameters were used, and strange attractor trajectories also looked similar, no matter what parameters were used.

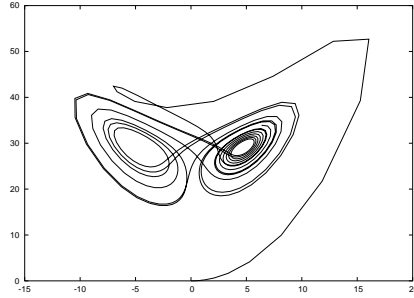


Figure 23: y_3 against y_1 , with $a = 5$, $b = \frac{7}{9}$, $r = 30$, and a starting point of $(0, 1, 0)$. It can be seen that the trajectory looks like a strange attractor, settling, after an initial transient, to orbit in one or the other loop without tending to any particular value.

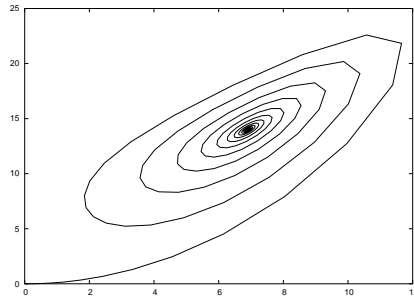


Figure 24: y_3 against y_1 , with $a = 7$, $b = \frac{17}{5}$, $r = 15$, and a starting point of $(0, 1, 0)$. It can be seen that the trajectory tends towards a stationary point, at $y_3 \approx 14$, and $y_1 \approx 7$.

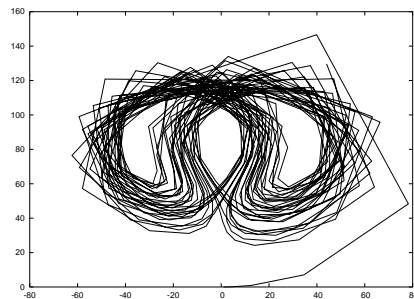


Figure 25: y_3 against y_2 , $a = 7$, $b = \frac{17}{5}$, $r = 87$, with a starting point of $(0, 1, 0)$. This trajectory follows a path that looks like a strange attractor, with the trajectory being in either one loop or the other, but without settling down to any particular point.

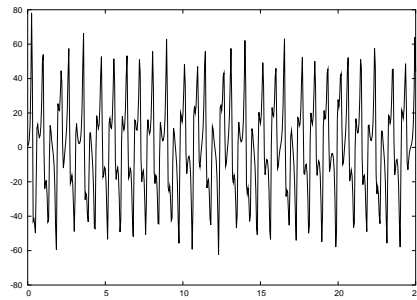


Figure 26: y_2 against time t , $a = 7$, $b = \frac{17}{5}$, $r = 87$, with a starting point of $(0, 1, 0)$. y_2 varies chaotically, not tending to any particular value.

A Computational Solution to the Runge-Kutta algorithm

```
program chaos
*   Solution to a set of coupled differential equations using the
*   Runge-Kutta method

*   Define variables to be used n = Number of steps y(1-3)(0-3) =
*   Variables; y(1-3)(0) are the starting points r is an arbitrary
*   parameter that affects the behaviour of the chaotic system

integer i,n,k,l
double precision r,t,y(3,0:3),f(3,0:3),nplot(2)
character*2 plot(2)

*   Define the constants here
parameter (a = 10)
parameter (b = 8/3)
parameter (dt = 0.05)

print *, 'Enter the number of iterations to perform:'
read *, n
print *, 'Enter the value of r to use:'
read *, r
print *, 'Enter the initial values of y1, y2 and y3:'
read *, y(1,0), y(2,0), y(3,0)

print *, 'Which values do you want to use as x & y when being'
print *, 'plotted (give two of y1,y2,y3,t in the form of \'x y\')'
read *, plot(1), plot(2)

t = -dt ! Set this to -dt so on the first pass, t=0

open (1, FILE='dataset', STATUS='unknown')

*   'unknown' will overwrite the file if it exists, when data is
*   written to it, or create the file if it doesn't already exist

do 10 i=0,n

    t = t + dt

*   nplot(1) is the value for the 'x' data, nplot(2) is the value
```

```

*      for the 'y' data

      do 50 l=1,2
        if (plot(l) .eq. 'y1') then
          nplot(l)=y(1,0)
        elseif (plot(l) .eq. 'y2') then
          nplot(l)=y(2,0)
        elseif (plot(l) .eq. 'y3') then
          nplot(l)=y(3,0)
        elseif (plot(l) .eq. 't') then
          nplot(l)=t
        endif

50      continue

      write (1,100),nplot(1),nplot(2) !Gnuplot expects data in x,y pairs

100     format (F12.5,F12.5)

*      Calculate f(j,0) for the initial values of y(j,0)
      f(1,0) = a*(y(2,0) - y(1,0))
      f(2,0) = r*y(1,0) - y(2,0) - y(1,0)*y(3,0)
      f(3,0) = y(1,0)*y(2,0) - b*y(3,0)

      do 30 k=1,3
*      Repeat this 3 times to get f(j,1) f(j,2) and f(j,3)
        do 20 j=1,3
*      Work out the y values that are then used to calculate the f's as
*      needed for the Runge-Kutta method calculations
          y(j,k) = y(j,0) + f(j,k-1)*(dt/2)
          if (k .eq. 3) then
*      Compensate for the fact that y(i,3) = y(i,0) + f(i,2)*dt, rather
*      than dt/2
            y(j,k) = y(j,k) + f(j,k-1)*(dt/2)
          end if
20      continue

*      Work out the intermediate values for f
      f(1,k) = a*(y(2,k) - y(1,k))
      f(2,k) = r*y(1,k) - y(2,k) - y(1,k)*y(3 ,k)
      f(3,k) = y(1,k)*y(2,k) - b*y(3,k)

30      continue

```

```
*      Now complete the step (see writeup) and make this value y(j,0),
*      and then start the whole iteration again
      do 40 j=1,3
        y(j,0) = y(j,0) + (f(j,0) + 2*f(j,1) + 2*f(j,2) + f(j,3))
1       *(dt/6)
40      continue

10     continue

      close(1)

      end
```

B Shell Script and gnuplot script

B.1 Shell script

```
#!/bin/sh

if [ -z $1 ] ; # If no parameter is given
then
    echo "You need to give a filename" ;
else
    ./co24-chaos # This is the name of the compiled Fortran program
    cat gnuscript | gnuplot # Run gnuplot, scripted from the file gnuscript
    mv gpoutput $1 ;
fi
```

B.2 Gnuplot script

```
set data style lines # Join the points together with lines
set terminal postscript # Create a postscript file as output
set output 'gpoutput' # Name of the PS file
set nokey # Don't print 'dataset' on the graph
plot 'dataset' # Plot points from the file 'dataset'
```

References

- [1] David Acheson. *From Calculus to Chaos: An Introduction to Dynamics*. OUP, 1998. p.158.
- [2] David Acheson. *From Calculus to Chaos: An Introduction to Dynamics*. OUP, 1998. p.51.
- [3] David Acheson. *From Calculus to Chaos: An Introduction to Dynamics*. OUP, 1998. p.159.
- [4] David Acheson. *From Calculus to Chaos: An Introduction to Dynamics*. OUP, 1998. p.159.
- [5] Arun V. Holden, editor. *Chaos*. Manchester University Press, 1986. p. 111.
- [6] Arun V. Holden, editor. *Chaos*. Manchester University Press, 1986. p. 19.
- [7] Arun V. Holden, editor. *Chaos*. Manchester University Press, 1986. p. 113.
- [8] Arun V. Holden, editor. *Chaos*. Manchester University Press, 1986. p. 275.
- [9] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 34.
- [10] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 148.
- [11] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 207.
- [12] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 208.
- [13] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 204.
- [14] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. p. 208.
- [15] Michael Tabor. *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989. pp. 209-210.

Word count: \approx 4400 words.