CO24: Chaos

A. Vitarana (pemb5671)
Pembroke College

1 Abstract

We calculate solutions of the Lorentz system using the fourth-order Runga-Kutta algorithm, given initial conditions and parameters for the Lorenz equations. Plotting the trajectory of the variables, it is clear the system is sensitive to changes in its initial conditions in certain regimes of the parameters; the system exhibits chaotic behaviour with parameters set close to a=10, b=8/3 and r=28.

2 Introduction

The Lorenz equations are three ordinary differential equations that form the Lorentz model. The Lorenz equations of the time-dependent variables y_1, y_2 and y_3 are:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = a(y_2 - y_1) \tag{1}$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = ry_1 - y_2 - y_1y_3 \tag{2}$$

$$\frac{\mathrm{d}y_3}{\mathrm{d}t} = y_2y_1 - by_3, \tag{3}$$

where a, r, and b are constants.

These equations can be used as a simplified model of atmospheric conduction, with each variable describing properties of the fluid in the Rayleigh-Bénard experiment, and they can be derived from the Navier-Stokes equations, the equation for heat conduction and the continuity equation [1]. Physically, y_1 is proportional to the rate of convection, y_2 to the temperature difference between currents and y_3 is the temperature difference between the top and the bottom of the fluid, where a is the Prandtl number, r is the Rayleigh number and b is a geometric factor [2].

The set of equations has three stationary solutions:

$$y_1 = y_2 = \pm\sqrt{b(r-1)}, y_3 = (r-1) \tag{4}$$

$$y_1 = y_2 = y_3 = 0, \tag{5}$$

The approach to these solutions from an arbitrary starting point are dependent on the regime of r [3].

1. If $r < 1$ the only real stationary point is the origin
2. If $r > 1$ all three stationary points exist. However, the origin is unstable and so you will never reach it unless you start exactly on it. As r increases, there is a higher degree of uncertainty in the system.
   (a) If $r < 1$ is less than about 24 the system will converge to one of the two stationary points not found at the origin.
   (b) If $r > 1$ is greater than about 24, the system will not converge to either solution and will move between the two fixed points in space. The behaviour is bounded but non-periodic.

3 Methods

These equations are coupled first order differential equation and numerical solutions are much easier to obtain than attempts to solve analytically through decouple the solutions or particular integrals and complementary functions. Runge-Kutta methods are iterative methods of approximately solving ordinary equations using small time steps. The Runge-Kutta method implemented in this program was the fourth-order method (RK4). The equations given in 1-3 are in the form,

$$\frac{\mathrm{d}y_i}{\mathrm{d}t} = f(y_1, y_2, y_3). \tag{6}$$

Defining a small time interval δt, the value of the y_i can be iteratively calculated at time steps of δt. Referring to the mth intermediate point in our calculation as $y_{i,m}$ and $f_{i,m}$, the RK4 iteration steps are:

1. Calculate $f_{i,0}$ using $y_{i,0}$
2. Calculate $f_{i,1}$ using $y_{i,1} = y_{i,0} + \frac{y_{i,0}\delta t}{2}$
3. Calculate $f_{i,2}$ using $y_{i,2} = y_{i,1} + \frac{y_{i,1}\delta t}{2}$
4. Calculate $f_{i,3}$ using $y_{i,3} = y_{i,2} + y_{i,2}\delta t$
5. $y_{i,4} = y_{i,0} + \frac{(f_{i,0}+2f_{i,1}+2f_{i,2}+f_{i,3})\delta t}{6}$

The values $y_{i,4}$ then become $y_{i,0}$ in the next step and the value of $y_{i,n}$, where $n = 4m$, corresponds to the value of y_i at time, $t = m\delta t$. The initial conditions of the system $y_{i,0}$ and the parameters of the system, a, b and r can be varied.

Although faster methods exist, Runge-Kutta methods nearly always work and hence are often used to produce a numerical solution [4]. Runge-Kutta algorithms are derived from Simpson's rule and use varying subintervals depending on their order; the fourth order algorithm gives the best balance between accuracy and computational effort [5]. These algorithms are better approximations of solutions than Euler's method due to the use of higher order terms which reduces truncation error, but they require more computational power.

4 Analysis and Results

4.1 Different Regimes of r

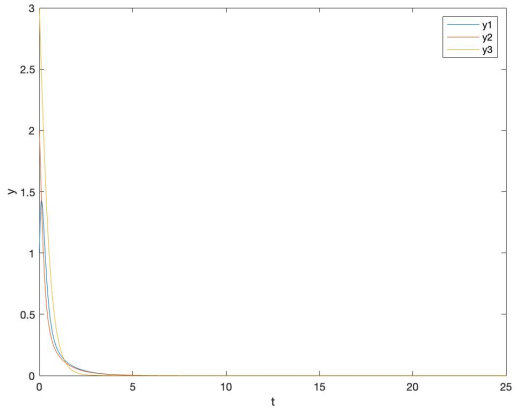
We fix the values of the parameters $a = 10$ and $b = 8/3$. Also, to compare different regimes we fix the initial conditions $y_{1,0} = 1$, $y_{2,0} = 2$ and $y_{3,0} = 3$ and the solutions are evaluated over $n = 500$ time steps, $\delta t = 0.05$.

4.1.1 $r < 1$

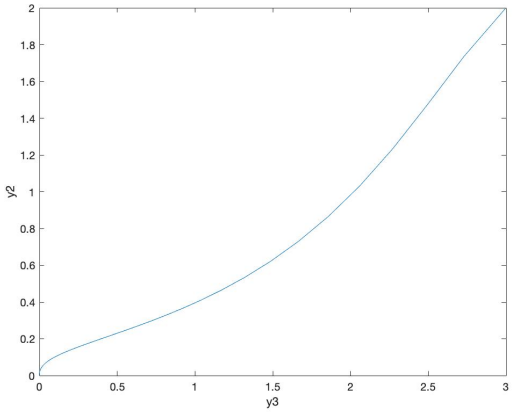
Setting $r = 0$, from figure 1(a), we observe that the values of y_1, y_2 and y_3 converge to zero, which we expect since the only real stationary point in this regime is the origin. Figure 1(b) shows the relation between variables y_2 and y_3; their behaviour towards converging at the origin is non-chaotic.

4.1.2 $1 < r < 24$

Setting $r = 15$, from figure 2(a), we observe that the values of y_1, y_2 and y_3 converge to one of the solutions, not including the origin, unless the initial conditions start at the origin since it is unstable. Figure 2(b) shows this convergence is a reached by spiralling around the solution.

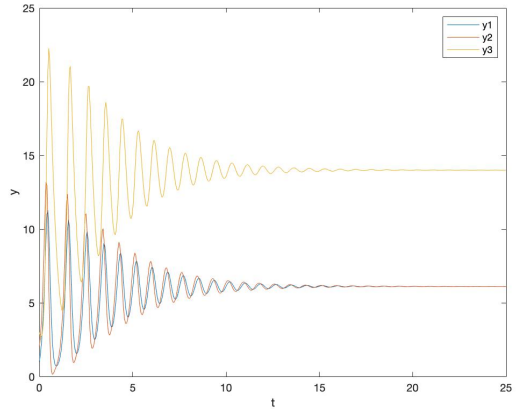


(a) Graph of y_1, y_2 and y_3 against time

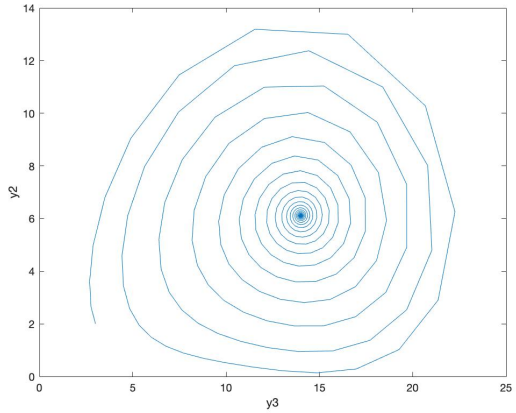


(b) Graph of y_2 against y_3

Figure 1: $r = 0$

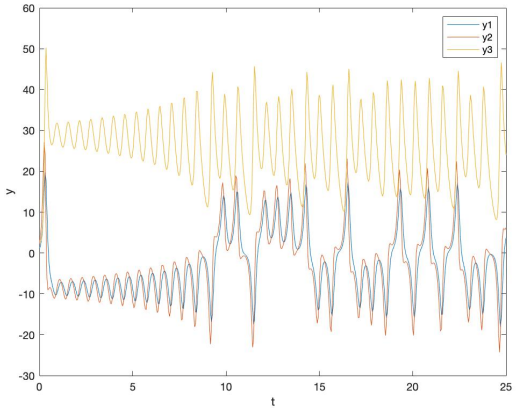


(a) Graph of y_1, y_2 and y_3 against time

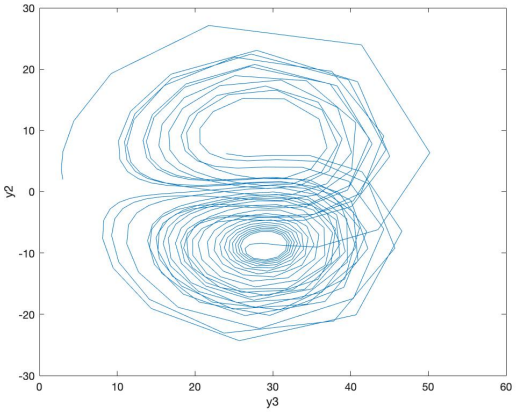


(b) Graph of y_2 against y_3

Figure 2: $r = 15$



(a) Graph of y_1, y_2 and y_3 against time



(b) Graph of y_2 against y_3

Figure 3: $r = 30$

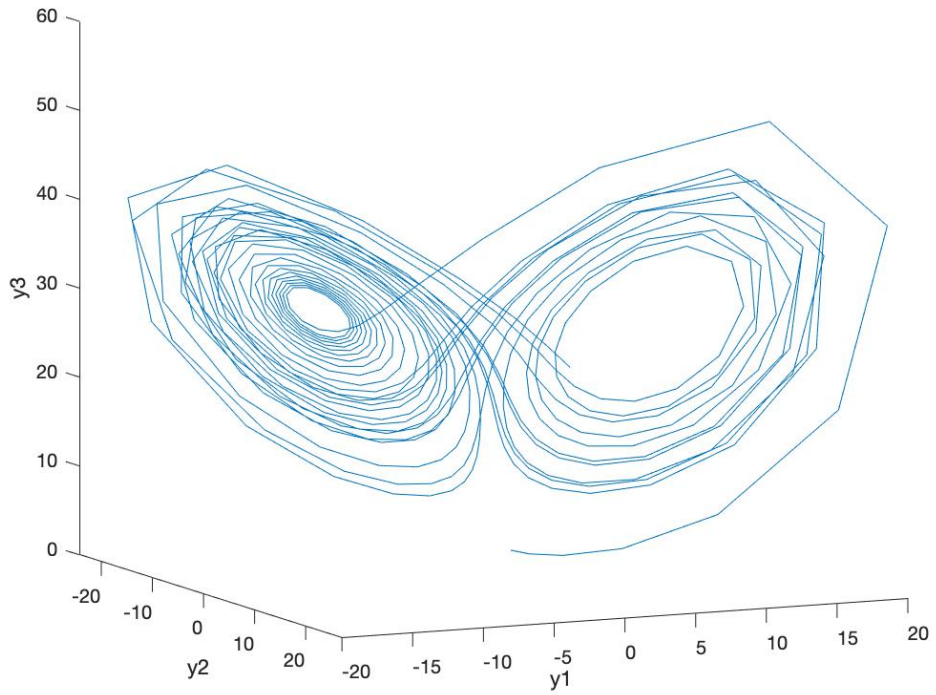


Figure 4: 3D plot of y_1, y_2 and y_3 at $r = 30$

4.1.3 $r > 24$

Setting $r = 30$, from figure 3(a), we observe that the values of y_1, y_2 and y_3 do not converge to one of the solutions, but instead wanders between the two solutions that are not the unstable origin. Figure 3(b) shows this convergence is a reached by spiralling around the solution.

This is a strange attractor, since the solutions are bounded but non-periodic. Plotting all three coordinates against each other, a more intuitive image of the Lorentz attractor can be viewed as in figure 4.

4.2 Chaotic Behaviour: 'the weather forecasting' phenomena

From §4.1.3, we observe what seems to be chaotic behaviour. To analyse this behaviour we fix the values of the parameters $a = 10$, $b = 8/3$ and $r = 28$, but slightly change the initial conditions $y_{1,0}$, $y_{2,0}$ and $y_{3,0}$. As before, the solutions are evaluated over $n = 500$ time steps, $\delta t = 0.05$. Initially, we set the initial conditions $y_{1,0} = 4$, $y_{2,0} = 5$ and $y_{3,0} = 6$, which we call these the real conditions. The time-evolution of the variables are compared to initial conditions $y_{1,0} = 4.01$, $y_{2,0} = 5.01$ and $y_{3,0} = 6.01$, which we call a 'small measurement error'.

Focusing on the time-evolution of y_1, Figure 5 shows that the slight change in initial conditions leads to a similar initial time evolution. However, after a short period of time the solutions separate and begin to wander about the two solutions following different paths. This demonstrates deterministic chaotic behaviour, since the present determines the future but the approximate present does not approximately determine the future. Since, weather systems can be approximated by the Lorentz model this demonstrates the difficulties of predicting the weather; small measurement errors will lead to vastly different outcomes.

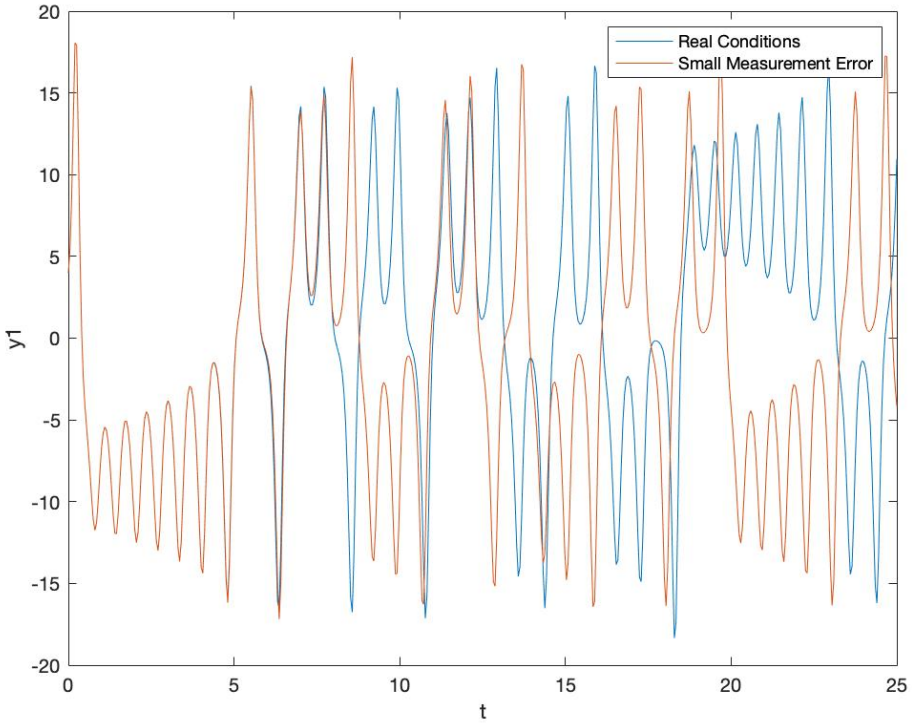


Figure 5: y_1 against time with a 'small measurement error' in initial conditions

5 Conclusions

The Lorentz system demonstrates chaotic behaviour in the regime $r > 24$, since a small change in the initial conditions of the system drastically changes the time-evolution of the system. The Lorentz attractor is observed, since the system is bounded by two solutions, but does not converge to either. In the approximately the regime $1 < r < 24$, the system will converge to one of the two solutions, which are not the origin, depending upon initial conditions. In the regime, $r < 1$, the system will converge to the solution at the origin.

The RK4 method offers a clear visualisation of the chaotic behaviour observed. However, since the RK4 method is an iterative method it propagates error. This is particularly unhelpful when studying chaos as the system is sensitive to small changes and therefore the numerical solution obtained by the RK4 method is likely to be vastly different from the actual solution after a long period of time; a small local error develops into a large global error. Another disadvantage in my ODE solver program was the use of equally spaced time intervals as at turning points the RK4 method is more inaccurate. Combing the RK4 method with intelligent adaptive step-size routine would increase the accuracy of the solution; MatLab has an inbuilt ode45 function that uses RK4 with changing time-steps. For systems with a large numbers of coupled equations, this adaptive step-size is particularly important. The ODE solver is extremely versatile and slight modifications to the program allow the higher order ODEs to be solved, such as a damped oscillator.

References

[1] Schuster H. G. (1988). *Deterministic chaos: an introduction*, 2nd Ed. Darmstadt: betz-druck gmbh, p 223-225.

[2] Weisstein, E. (2018). *CO24: chaos* [Accessed 24 January 2020] Available at: http://mathworld.wolfram.com/LorenzAttractor.html

[3] (2018). *Lorentz Attractor* [Accessed 24 January 2020] Available at: https://www-teaching.physics.ox.ac.uk

[4] Press W. H. et al. (2007). *Numerical Recipes: the Art of Scientific Computing*, 3rd Ed. Cambridge: Cambridge University Press, p 908.

[5] O'Hare A. (2005). *Numerical Methods for Physicists*, Oxford Physics, p 69.

Appendix A ODE Solver

```
function [y] = ode_solve_rk(f1, f2, f3 , y0 , t)
% Author: Aaron Vitarana, Date: 22/01/2020
%
% Solve ODE problems using Runge-Kutta algorithm.
%
% Input:
% * f1, f2, f3: functions that receive current states of y1, y2 and y3
% * y0: the initial state of the system, given in a column matrix (3 x
    1).
% * t: vector of equally spaced position/time steps with length N where
    the values of y will be returned.
%
% Output:
% y: (3 x N) matrix that contains the values of y at every position/time
% step and columns correspond to the position/time and rows to the
    element of y.


% setting up initial vector and filling it with the initial conditions
y = zeros(3, length(t))
y(1, 1) = y0(1, 1);
y(2, 1) = y0(2,1);
y(3, 1) = y0(3,1);

% determining time step
dt = abs(t(2) - t(1));


for a = 1:(length(t)-1) %iterate through all values of t except the
    largest value
     k1 = f1(y(1,a), y(2,a), y(3,a));
     l1 = f2(y(1,a), y(2,a), y(3,a));
     m1 = f3(y(1,a), y(2,a), y(3,a));
     k2 = f1(y(1,a) + 0.5*k1*dt, y(2,a) + 0.5*l1*dt, y(3,a) + 0.5*m1*dt);
     l2 = f2(y(1,a) + 0.5*k1*dt, y(2,a) + 0.5*l1*dt, y(3,a) + 0.5*m1*dt);
     m2 = f3(y(1,a) + 0.5*k1*dt, y(2,a) + 0.5*l1*dt, y(3,a) + 0.5*m1*dt);
     k3 = f1(y(1,a) + 0.5*k2*dt, y(2,a) + 0.5*l2*dt, y(3,a) + 0.5*m2*dt);
     l3 = f2(y(1,a) + 0.5*k2*dt, y(2,a) + 0.5*l2*dt, y(3,a) + 0.5*m2*dt);
     m3 = f3(y(1,a) + 0.5*k2*dt, y(2,a) + 0.5*l2*dt, y(3,a) + 0.5*m2*dt);
     k4 = f1(y(1,a) + k3*dt, y(2,a) + l3*dt, y(3,a) + m3*dt);
     l4 = f2(y(1,a) + k3*dt, y(2,a) + l3*dt, y(3,a) + m3*dt);
     m4 = f3(y(1,a) + k3*dt, y(2,a) + l3*dt, y(3,a) + m3*dt);

     y(1, a+1) = y(1, a) + (k1 + 2*k2 + 2*k3 + k4)*dt/6;
     y(2, a+1) = y(2, a) + (l1 + 2*l2 + 2*l3 + l4)*dt/6;
     y(3, a+1) = y(3, a) + (m1 + 2*m2 + 2*m3 + m4)*dt/6;
end

end
```

Appendix B Lorenz Equations

```
function [y] = solve_lorenz(y0 , a , b , r , t)
% Author: Aaron Vitarana, Date: 22/01/2020
% Solve Lorenz equations using the implemented ODE solver.
% Dedicated to Anna
% Input:
% * y0: a column vector of the starting point with size (3 x 1)
% * a, b, r: parameters of the Lorenz equations
% * t: an N?element vector of time/position steps where y will be
   calculated
%
% Output:
% * y: a (3 x N) matrix that contains the values of y for every time
   step

% defining Lorenz functions
f1 = @(y1, y2, y3) (a*(y2-y1))
f2 = @(y1, y2, y3) (r*y1-y2-y1*y3)
f3 = @(y1, y2, y3) (y1*y2-b*y3)


% calls Runga-Kutta function to solve ODEs
% returns y: (M x N) matrix that contains the values of y at every
   position/time
% step and columns correspond to the position/time and rows to the
   element of y.
[y] = ode_solve_rk(f1, f2, f3, y0 , t)

end
```