SOLVING THE LORENZ SYSTEM: A SURVEY OF NUMERICAL METHODS

SARAH SEVERINGHAUS

Abstract. Because the Lorenz System cannot be solved analytically, we must turn to numerical methods to find solutions. When dealing with systems whose behavior is highly sensitive to small perturbations in initial conditions, such as the Lorenz System, the choices of method and step size are highly important because methods with different levels of precision yield variably meaningful results. To demonstrate the applicability of various finite different methods to chaotic systems of ordinary differential equations, this paper examines the expected error from each method's derivation and the validity of solutions found computationally for each method.

1. Introduction

In the 1963 paper "Deterministic Nonperiodic Flow", Lorenz discussed the results of numerical solutions to a simplified, two-dimensional model for cellular convection that is now known as the Lorenz System [4]. Lorenz noticed that tiny variations in the system's initial conditions drastically changed the behavior of the numerical solutions, giving rise to the classification of the system as chaotic. The Lorenz System has no analytic solutions, motivating the use of numerical methods. In this paper, I will demonstrate the application of finite difference methods, a family of numerical methods useful for solving differential equations, to solving the Lorenz System for the same parameters as Lorenz in [4]. Computational experiments will expound on the important considerations and common difficulties in achieving meaningful numerical solutions we may expect from analytic treatment of such methods. Finally, I will discuss a few promising alternatives to overcome the challenges associated with using finite difference methods to numerically solve chaotic systems.

The Lorenz System is given by three autonomous, nonlinear, coupled ordinary differential equations:

$$\begin{aligned} \dot{x} &= \sigma y - \sigma x \\ \dot{y} &= rx - xz - y \\ \dot{z} &= xy - bz, \end{aligned} \tag{1}$$

where the dots denote time derivatives. The positive, dimensionless parameters are σ, the Prandtl number, r, which is related to the Rayleigh number, and b, which is related to the Reynolds number [6, p. 5]. In Lorenz's 1963 paper the parameters studied were $\sigma = 10$, $r = 28$, and $b = \frac{8}{3}$. Lorenz derived the system by simplifying a previously developed convection model describing the behavior of an incompressible fluid heated from below and cooled from above in a closed chamber under the force of gravity [6, p. 5].

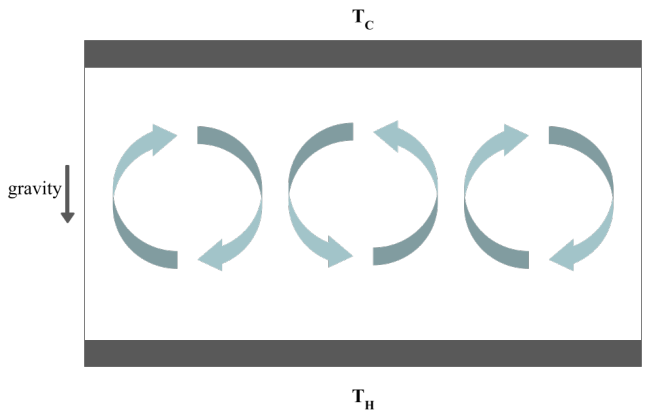

FIGURE 1. Diagram of the convection cell system studied by Lorenz, with boundary temperatures T_C and T_H where $T_H > T_C$. Inspired by [6, Fig. 1.4].

2. Euler's Method

Consider the initial value problem (IVP) given by

$$\frac{df}{dt} = g(t, f(t)), \ \ f(0) = x_0. \tag{2}$$

For small Δt, the Taylor series expansion of $f(t)$ around t gives the approximation

$$f(t + \Delta t) \approx f(t) + \Delta t g(t, f(t)) + \frac{\Delta t^2}{2} f''(t). \tag{3}$$

We can then write

$$f(t + \Delta t) = f(t) + \Delta t g(t, f(t)) + \mathcal{O}(\Delta t^2), \tag{4}$$

where $\mathcal{O}(\Delta t^2)$ denotes that all successive terms in the Taylor series are on the order of Δt^2; that is, the sum of the remaining terms is at most linear with respect to Δt^2.

To make approximations computationally we must discretize the domain [2, p. 35]. Introduce a grid over t: let $t_j = x_0 + j\Delta t$ for $j = 0 \to n$. Noting that $t_{j+1} = t_j + \Delta t$, using the approximation given by (4) we define $f(t_{j+1})$ recursively,

$$f(t_{j+1}) = f(t_j) + g(t_j, f(t_j))\Delta t. \tag{5}$$

From (4), we expect associated error $\mathcal{O}(\Delta t^2)$ for each step of this recursion[2, p. 35]. Letting $f_j = f(t_j)$ and $f_{j+1} = f(t_{j+1})$,

$$f_{j+1} = f_j + g(t_j, f_j) j\Delta. \tag{6}$$

With the initial condition $f(0) = x_0$, we have the complete algorithm:

$$\begin{aligned} f_0 &= x_0 \\ f_{j+1} &= f_j + g(t_j, f_j)\Delta t, \end{aligned} \tag{7}$$

with known local truncation error $\mathcal{O}(\Delta t^2)$.

Vectorized, the numerical scheme is

$$\begin{aligned} \mathbf{f}_0 &= \mathbf{x}_0 \\ \mathbf{f}_{j+1} &= \mathbf{f}_j + \mathbf{g}(t, \mathbf{f}_j)\Delta t, \end{aligned} \tag{8}$$

again with local truncation error at each step $\mathcal{O}(\Delta t^2)$.

3. Midpoint Runge-Kutta Method (Second Order)

Morally, the midpoint Runge-Kutta method can be though of as improving the Euler's method approximation by evaluating the derivative at $t_j + \frac{\Delta t}{2}$, the midpoint of the step instead of the beginning of the step t_j [3, p. 352]. The midpoint is approximated via Euler's method before that approximation is plugged into $g(t, f(t))$, the derivative of our function f, which is used to determine the total step that should be made over Δt. To derive the midpoint Runge-Kutta method analytically, I follow the outline of the derivation in [2, p. 37-41] supplemented with the description from [3, p. 351-352].

We begin the derivation by once again considering the IVP

$$\frac{df}{dt} = g(t, f(t)),\ f(0) = x_0. \tag{9}$$

To improve upon Euler's method we can include another term in the Taylor expansion of $f(t + \Delta t)$,

$$f(t + \Delta t) = f(t) + \Delta t g(t, f(t)) + \frac{\Delta t^2}{2}\frac{dg(t, f(t))}{dt} + \mathcal{O}(\Delta t^3). \tag{10}$$

Taylor expanding $g(t, f(t))$ around $t + \frac{\Delta t}{2}$,

$$g\left(t + \frac{\Delta t}{2}, f(t + \frac{\Delta t}{2})\right) = g\left(t, f\right) + \frac{\Delta t}{2}\frac{dg(t, f(t))}{dt} + \mathcal{O}(\Delta x^2). \tag{11}$$

So (10) can be written as

$$f(t + \Delta t) = f(t) + \Delta t\left[g\left(t + \frac{\Delta t}{2}, f(t + \frac{\Delta t}{2})\right) + \mathcal{O}(\Delta t^2)\right] + \mathcal{O}(\Delta t^3), \tag{12}$$

which is equivalent to

$$f(t + \Delta t) = f(t) + g\left(t + \frac{\Delta t}{2}, f(t + \frac{\Delta t}{2})\right)\Delta t + \mathcal{O}(\Delta t^3). \tag{13}$$

Using Euler's method to approximate $f(t + \frac{\Delta t}{2})$,

$$f(t + \Delta t) = f(t) + g\left(t + \frac{\Delta t}{2}, f(t) + \frac{\Delta t}{2}g(t, f(t))\right)\Delta t + \mathcal{O}(\Delta t^3). \tag{14}$$

Discretizing the relation,

$$f_{j+1} = f_j + g\left(t_j + \frac{\Delta t}{2}, f_j + \frac{\Delta t}{2}g(t_j, f_j)\right)\Delta t. \tag{15}$$

Because the rightmost term is unwieldy, we define coefficients k_1 and k_2

$$\begin{aligned} k_1 &= g(t_j, f_j)\Delta t \\ k_2 &= g(t_j + \frac{\Delta t}{2}, f_j + \frac{k1}{2}), \end{aligned} \tag{16}$$

so that we have the complete method given by

$$\begin{aligned} k_1 &= g(t_j, f_j)\Delta t \\ k_2 &= g(t_j + \frac{\Delta t}{2}, f_j + \frac{k_1}{2}) \\ f_{j+1} &= f_j + k_2 \\ f_0 &= x_0. \end{aligned} \tag{17}$$

From (14), the last step before discretization, we expect to have local truncation error $\mathcal{O}(\Delta t^3)$ associated with each step. The midpoint Runge-Kutta has a smaller error at each step than Euler's method by a whole order of Δt. While each step is more computationally expensive because of the coefficient computations, we can get better accuracy with the simplest of Runge-Kutta schemes than we can with Euler's method at the same step size Δt.

The vector form of the method, which we use for our system of ODEs, is

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{g}(t_j, \mathbf{f}_j)\Delta t \\ \mathbf{k}_2 &= \mathbf{g}(t_j + \frac{\Delta t}{2}, \mathbf{f}_j + \frac{\mathbf{k}_1}{2}) \\ \mathbf{f}_{j+1} &= \mathbf{f}_j + \mathbf{k}_2 \\ \mathbf{f}_0 &= \mathbf{x}_0. \end{aligned} \tag{18}$$

4. Fourth Order Runge-Kutta Method

The fourth order Runge-Kutta method is the most popular for solving nonlinear systems of ODEs of the methods described here due to its balance of lower truncation error and computational speed. In 1996, Lorenz too implemented the fourth order Runge-Kutta method to solve the convection cell problem instead of the second order method his computations from the 1960's typically relied on [5]. I will not derive the fourth order method in detail here as the derivation follows the same logic as the midpoint method's derivation and there are many possible formulations. Instead, I will explain and provide justification for the choices of coefficients k_1 through k_4 that appear in the fourth order Runge-Kutta method's most popular form.

For the IVP

$$\begin{aligned} \frac{df}{dt} &= G(t, f(t)) \\ f_0 &= x_0, \end{aligned} \tag{19}$$

the fourth-order Runge-Kutta scheme for this initial value problem is

$$\begin{aligned} f_0 &= x_0 \\ k_1 &= g(t_j, f_j)\Delta t \\ k_2 &= g(t_j + \frac{\Delta t}{2}, f_j + \frac{k_1}{2})\Delta t \\ k_3 &= g(t_j + \frac{\Delta t}{2}, f_j + \frac{k_2}{2})\Delta t \\ k_4 &= g(t + \Delta t, f_j + k_3)\Delta t \\ f_{j+1} &= f_j + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \end{aligned} \tag{20}$$

with local truncation error $O(x^5)$.

The k_2 and k_3 terms estimate the midpoint using the k_1 and k_2 terms, respectively. With that estimate the k_2 and k_3 terms approximate the slope at the midpoint to approximate the function at Δt. The k_4 term finds the slope at the end of the interval to approximate the function at Δt. Then, the k terms are averaged with higher weighting attributed to the midpoint slope estimates (k_2 and k_3). Therefore any linear combination of k coefficients that sums to one is valid, although some choices of linear combinations are particularly well motivated. By weighting k_1 and k_4 equally less than the middle terms, the approximation is less susceptible to inaccuracy caused by a rapid, dramatic change in slope at the beginning or end of the time step interval.

In vector form, the complete fourth order Runge-Kutta method is given by

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{g}(t_j, \mathbf{f}_j)\Delta t \\ \mathbf{k}_2 &= \mathbf{g}(t_j + \frac{\Delta t}{2}, \mathbf{f}_j + \frac{\mathbf{k}_1}{2})\Delta t \\ \mathbf{k}_3 &= \mathbf{g}(t_j + \frac{\Delta t}{2}, \mathbf{f}_j + \frac{\mathbf{k}_2}{2})\Delta t \\ \mathbf{k}_4 &= \mathbf{g}(t + \Delta t, \mathbf{f}_j + \mathbf{k}_3)\Delta t \\ \mathbf{f}_0 &= \mathbf{x}_0 \\ \mathbf{f}_{j+1} &= \mathbf{f}_j + \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}. \end{aligned} \tag{21}$$

5. Mathematica Implementation

To apply this method to the Lorenz System with initial conditions given by $\mathbf{x}_0$, define $\mathbf{g}$ such that

$$\mathbf{g}(t, \mathbf{f}(t)) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \sigma y - \sigma x \\ rx - xz - y \\ xy - bz \end{pmatrix}. \tag{22}$$

For a system of autonomous ODEs, such as the Lorenz system, the Mathematica implementations for each of the numerical methods discussed in this paper are given below. The code assumes that the input function `g` and initial conditions `x0` are vectors of the same size and that `t_List` is a list of all time steps ($j\Delta t$ for $j : 0 \to n$ for n steps). `t_List` can be created using `Range[0, t_max, dt]` where `dt` is the desired step size and `t_max` is the desired end of the time interval.

5.1. Euler's Method.

```
    Euler[g_, x0_, t_List] :=
 Module[{y, dt, a}, y = ConstantArray[0., {Length[x0], Length[t]}];
y[[All, 1]] = x0;
  For[a = 1, a < Length[t], a++, dt = t[[a + 1]] - t[[a]];
   y[[All, a + 1]] = y[[All, a]] + dt g[y[[All, a]]];];
  y]
```

5.2. **Midpoint Runge-Kutta.**

```
  RungeKuttaMidpoint[g_, x0_, t_List] :=
Module[{y, dt, k1, k2, a},
 y = ConstantArray[0., {Length[x0], Length[t]}];
 y[[All, 1]] = x0;
 For[a = 1, a < Length[t], a++, dt = t[[a + 1]] - t[[a]];
  k1 = dt g[y[[All, a]]];
  k2 = dt g[y[[All, a]] + 0.5 k1];
  y[[All, a + 1]] = y[[All, a]] + k2;];
 y]
```

5.3. **Fourth Order Runge-Kutta.**

```
  RungeKutta4[g_, x0_, t_List] :=
Module[{y, dt, k1, k2, k3, k4, a},
 y = ConstantArray[0., {Length[x0], Length[t]}];
 y[[All, 1]] = x0;
 dt = Abs[t[[2]] - t[[1]]];
 For[a = 1, a < Length[t], a++, k1 = g[y[[All, a]]];
  k2 = g[y[[All, a]] + 0.5*k1*dt];
  k3 = g[y[[All, a]] + 0.5*k2*dt];
  k4 = g[y[[All, a]] + k3*dt];
  y[[All, a + 1]] = y[[All, a]] + (k1 + 2*k2 + 2*k3 + k4)*dt/6;];
 y]
```

6. Numerical Solutions

The following solutions for each numerical method are evaluated for $t \in [0, 30]$, $\Delta t = 0.001$, and $\mathbf{x}_0 = (1, 1, 1)$ with parameters $\sigma = 10$, $r = 28$, $b = \frac{8}{3}$, and $\mathbf{x}_0 = (1, 0, 0)$.

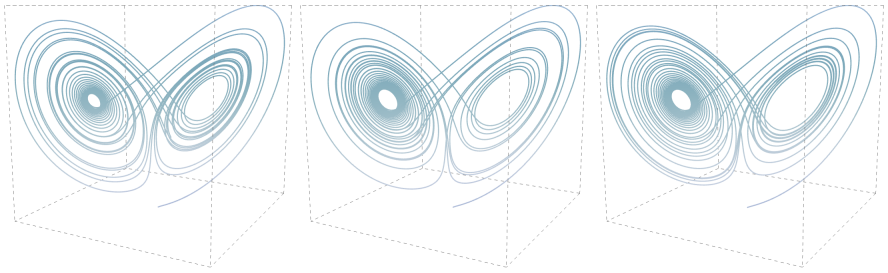

Figure 2. From left to right: the Euler's method, Midpoint Runge-Kutta, and Fourth-Order Runge-Kutta solutions.

As shown in Fig. 2, the different methods give similar yet not identical results. Fig. 3 gives a clearer picture of the differences between the three methods; as expected, the Euler's method solution, which has local error $\mathcal{O}(\Delta t^2)$ at each step, diverges from the fourth order Runge-Kutta solution faster than the midpoint Runge-Kutta solution, which has local error $\mathcal{O}(\Delta t^3)$ at each step.

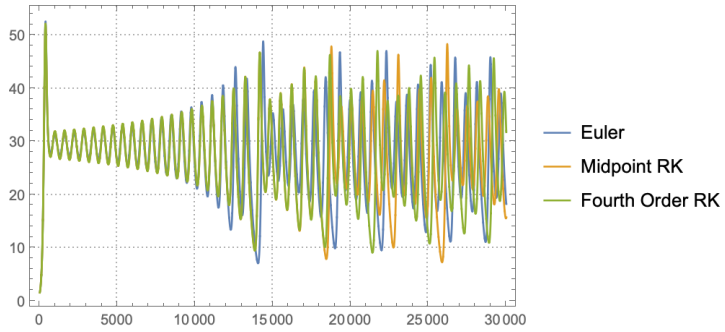

FIGURE 3. Normed magnitude of each solution versus step number.

Generally when evaluating the validity of solutions yielded by numerical methods, one can adjust the step size until further decreases in step size do not change the solution (at least to some desired precision). To get the Euler's method solutions to stop diverging from one another the step size has to be extraordinarily small, as opposed to the Runge-Kutta method in which the solutions converge at reasonable choices in step size. Below are the results of numerical experiments to determine the step sizes required to get meaningful results with the methods discussed, if possible.

6.1. **Euler's Method.**

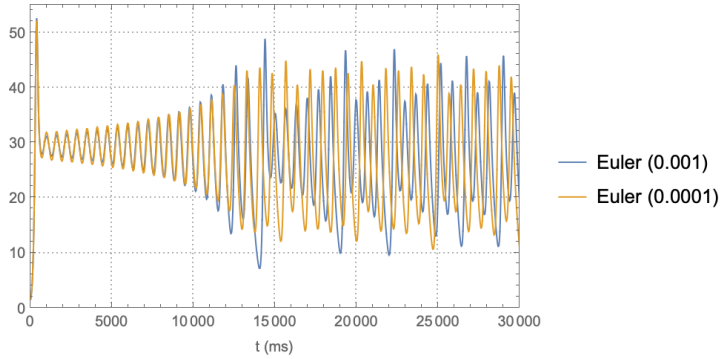

FIGURE 4. Euler's method solutions with $\Delta t = 0.001$ and $\Delta t = 0.0001$. Because the step sizes are different the horizontal axis has units of time (milliseconds) instead of step number.

In [3, p. 61-62], Giordano claims that the standard Euler method is suitable for solving the Lorenz System when step size is made small enough because "the energy lost (or added) through the error terms associated with the Euler method can be made much smaller than the energy lost to the effective damping, or added by the effective driving force." Despite the Lorenz System's damping-like behavior, I have not succeeded in getting Euler's method solutions to agree in the time interval unlike both Runge-Kutta methods. Computation time at $\Delta t = 0.000001$ for even small time intervals gets quite large, and Mathematica becomes much more liable to crashing or quitting the kernel.

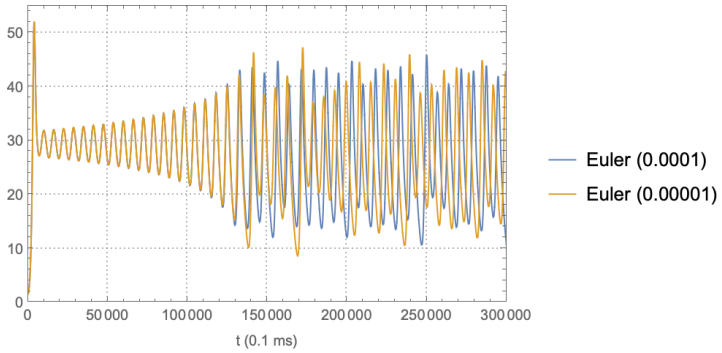

FIGURE 5. Euler's method solutions with $\Delta t = 0.0001$ and $\Delta t = 0.00001$. Horizontal axis has units of 0.1 milliseconds. Even with these very small step sizes, the solutions do not agree within this short interval.

6.2. **Midpoint Runge-Kutta.**

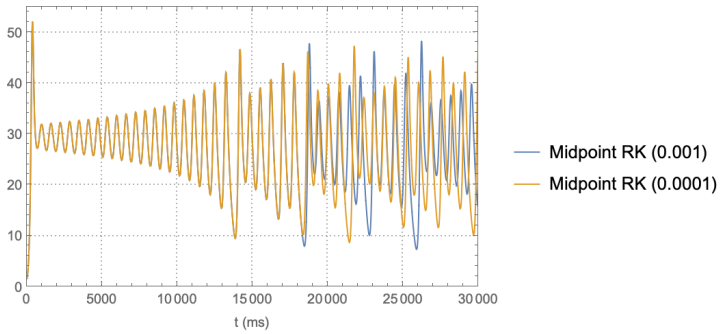

FIGURE 6. Midpoint Runge-Kutta method solutions with $\Delta t = 0.001$ and $\Delta t = 0.0001$. Horizontal axis has units of milliseconds.

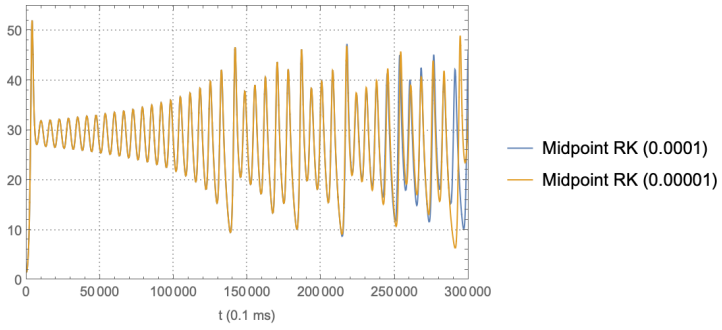

FIGURE 7. Midpoint Runge-Kutta method solutions with $\Delta t = 0.0001$ and $\Delta t = 0.00001$. Horizontal axis has units of 0.1 milliseconds. The solutions begin to agree much better.

6.3. **Fourth Order Runge-Kutta Method.**

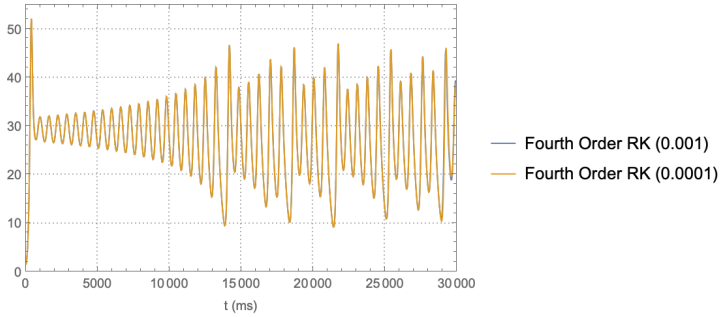

FIGURE 8. Fourth order Runge-Kutta method solutions with $\Delta t = 0.001$ and $\Delta t = 0.0001$. Refining the step size further does not change the solution within this time interval. Because the solutions have already stabilized as step size decreases, further decreasing the step size will not make the results more meaningful and increase computation time.

7. Conclusion

The fourth order Runge-Kutta method indeed produces the best results out of the methods discussed here. Even when $\Delta t = 0.00001$ the Euler method's $\mathcal{O}(\Delta t^2)$ local truncation error at each step results in error propagation too large for solutions to converge. The nature of chaotic systems necessitates that careful attention to error order is made when finding solutions numerically. Additionally, for potentially unstable nonlinear systems of ODEs such as the Lorenz System, alternative numerical methods may be less prone to error amplification and less computationally expensive than finite differences such as those discussed in this paper. Promising methods include the use of neural networks [2, p. 351-364] and finite element methods [1].

References

[1] Matthias Chung, Justin Krueger, and Honghu Liu. Least-squares finite element method for ordinary differential equations. *Journal of Computational and Applied Mathematics*, 418(114660):1–21, 2023.

[2] Joel Franklin. *Computational Methods for Physics*. Cambridge University Press, Cambridge, U.K., 2013.

[3] Nicholas J. Giordano. *Computational Physics*. Prentice Hall, Upper Saddle River, N.J., 1997.

[4] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130–141, 1963.

[5] Edward N Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on predictability*, volume 1, pages 1–18. European Centre for Medium-Range Weather Forecasts, 1996.

[6] Edward Ott. *Chaos in dynamical systems*. Cambridge University Press, Cambridge, U.K., 2nd edition, 2002.