Computational Modelling of Nonlinear Dynamics: chaos in the Lorenz equations

September 28, 2018

Contents

1 Introduction

Ordinary differential equations (ODEs) govern the behaviour of many systems in the real world, describing behaviour ranging from the evolution of planetary orbits in the solar system to fluctuations in financial markets. An arbitrary n-th order system of ODEs can be written as a set of coupled first-order ODEs for n variables $\{x_1(t), x_2(t), \cdots, x_n(t)\}$, in the form

$$\dot{x}_i = f_i(t, x_1, x_2, \cdots, x_n), \quad 1 \le i \le n,$$

where the dot denotes differentiation w.r.t. time.

When n is small and the functions f_i are simple, it may be possible to find an explicit solution of the ODEs. For example, when $n = 2$, $f_1 = x_2$ and $f_2 = -x_1$, we obtain

$$\dot{x}_1 = x_2, \qquad \dot{x}_2 = -x_1,$$

which are the equations governing the small amplitude oscillations of a pendulum, with angular displacement x_1 and angular velocity x_2. An explicit solutions is possible, of the form $x_1 = a\cos t + b\sin t$, describing simple-harmonic motion with oscillations of period 2π.

However, it is not possible to find explicit solutions of most systems of ODEs, typically because either n is large or the functions f_i are nonlinear. This project is mainly concerned with three issues that then arise:

1. calculating (approximate) numerical solutions of the ODEs;
2. investigating the rich and highly complex behaviour that can arise in nonlinear ODEs;
3. understanding some aspects of this behaviour, using appropriate mathematical analysis.

It is suggested that you start (and maybe even finish) by studying so-called *Lorenz equations* (Lorenz, 1963), which are written here in terms of three variables $x(t)$, $y(t)$ and $z(t)$:

$$\dot{x} = \sigma(y - x), \qquad \dot{y} = rx - y - xz, \qquad \dot{z} = xy - bz, \tag{1}$$

where σ, r and b are positive real numbers. Note the nonlinear terms $-xz$ and xy on the right-hand sides of the second and third equations. The system (1) was derived as a highly simplified model of the strength of atmospheric convection, which is a process whereby warm air rises upwards buoyantly and displaces cold air, which moves downwards leading to a closed circulation. Despite the simplicity of the system, it describes remarkably complex behaviour, which has been the subject of many research papers over the last 50 years.

In the following sections, various tasks based around the Lorenz equations are identified. These tasks fall into three main categories:

1. **computational** – requiring computer programs to be written, or used;
2. **analytical** – requiring mathematical calculations, on paper;
3. **research** – requiring you to read parts of books and research papers, and summarise the results.

To complete all the tasks in detail would probably constitute a Masters thesis worth 60 credits (or more). So, you need not complete all the tasks, and those that you do complete could be investigated to different degrees. There is even the possibility of setting your own tasks in the later stages of the project. All that is required is that you make a coherent set of investigations, that are written up in an interesting project report (see section 8).

2 Derivation of the Lorenz equations

An obvious starting point is to consider how the Lorenz equations are derived. There are two possibilities:

1. follow the instructions in the third paragraph on p.135 of Lorenz (1963); that is, starting from his equations (17), (18), (23) and (24), derive his (25), (26), (27). Unless you are versatile, this will only be possible if if you have studied fluid dynamics (e.g., MATH 2620 or MATH 3620).
2. Alternatively, the Lorenz equations can be thought of as a model of a particular kind of water wheel, as described in Appendix B of Sparrow (1982), or chapter 9 of Strogatz (1994).

The first option is quite hard, whilst the second option is somewhat artificial. So, whether or not you invest time in the derivation is up to you.

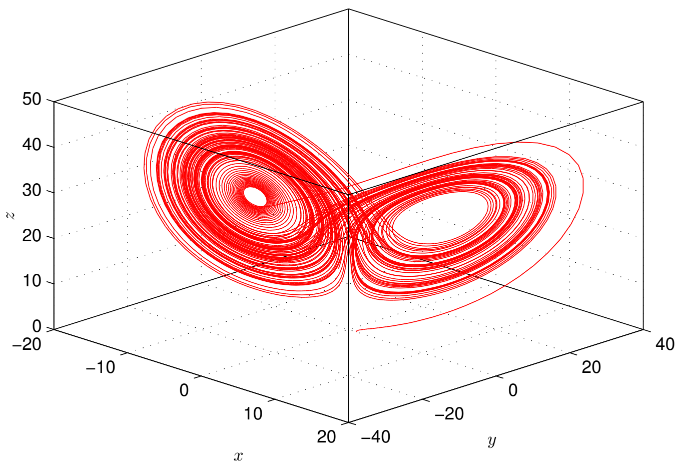

Figure 1: An illustration of the evolution of the Lorenz questions. The parameters are $\sigma = 10$, $r = 28$, and $b = 8/3$, and the equations are run out to $t = 100$ from $x(0) = 1$, $y(0) = 0$ and $z(0) = 0$.

3 Numerical solution of the Lorenz equations: basics

In Appendix A, you are provided with a simple program (in python) that uses a time-stepping scheme from the Runge–Kutta family to find an approximate solution of (1). Make some test runs, and try varying the parameters σ, r and b, along with the initial conditions and final time. You might like to try recreating some of the data given in the paper by Lorenz (1963), or the evolution shown in Figure 1 (although you would need to spend some time figuring out how to make 3D plots in python).

4 Properties of the Lorenz equations

In addition to these numerical solutions, it is worth considering what may be determined analytically. The most important tasks are to determine the fixed points of the Lorenz equations, and then to determine their stability. The key ideas here will have been covered (albeit for systems of 2 variables) in first-year and second-year modules (e.g., MATH 1400, MATH 2391). You should

1. determine the fixed points of the Lorenz equations (1) for arbitrary σ, r and b. You should find that there are two different cases, according as to whether or not $r < 1$. In the latter case, we denote the three fixed points by $\mathbf{x}_-$, $\mathbf{x}_0 = (0, 0, 0)$ and $\mathbf{x}_+$.

2. Determine the stability of the fixed points, which involves finding the eigenvalues λ of a 3×3 matrix. This leads to a cubic equation for λ, which can be analysed in various ways. Quite a lot of progress can be made mathematically: this is not bad for the fixed point at the origin, but it is much harder for $\mathbf{x}_-$ and $\mathbf{x}_+$ – see Sparrow (1982) or Strogatz (1994). However, `python` can be used to find the eigenvalues numerically, so you can say something this way. To do this, it is suggested that you fix $\sigma = 10$ and $b = 8/3$ (these are traditional values, used by Lorenz and others), and then consider how the eigenvalues change as r is increased. You can start with the simple code given in Appendix A.1.

If you have taken MATH 2391 (Nonlinear Differential Equations), you will have learned about other standard tests for determining properties of ODEs. Any of these could be applied to the Lorenz equations, in principle. An interesting result of this sort is to show that all orbits are bounded in phase space (that is, there are no orbits for which $|\mathbf{x}|$ increases without bound). This can be established using the methods of Lorenz (1963) or those in Appendix C of Sparrow (1982).

5 Numerical solution of ODEs: general considerations

At some point, you need to consider the approximations that are inherent in a numerical solution of the Lorenz equations. The issues here are most easily appreciated by considering a reduced set of ODEs, obtained by omitting the nonlinear terms in (1). If we further set $\sigma = 1$, $r = 4$ and $b = 1$ (for convenience – other choices are possible), we obtain

$$\dot{x} = y - x, \qquad \dot{y} = 4x - y, \qquad \dot{z} = -z. \tag{2}$$

This system of equations may be solved analytically: find the general solution, and then the solution subject to the initial conditions

$$x(0) = 1, \;\; y(0) = -1, \;\; z(0) = 2. \tag{3}$$

If we then solve the same equations numerically, we are able to check that our numerical schemes are working correctly by comparing with the exact analytical solution.

You should modify your codes to solve (2) rather than (1), subject to (3).

1. Run your code out to $t = 1$, with $\Delta t = 0.05$. What is the error in the solution, perhaps as measured by
$$E = \sqrt{(x(1) - x_f)^2 + (y(1) - y_f)^2 + (z(1) - z_f)^2},$$
where (x_f, y_f, z_f) is the exact solution at $t = 1$, which you should have determined analytically. You should find that the error is small – about 0.0015. If the error is larger, then there is mistake in either your analysis or the program.

2. Calculate E for other values of Δt, including 0.001, 0.005, 0.01 and 0.1; you may include other values if you prefer. Tabulate your results, make a plot of $\log E$ versus $\log \Delta t$, and fit a straight line to your data points using linear regression (i.e., least squares – you could write your own program to do this, or use `numpy.polyfit`). By postulating a relationship of the form $E = c\Delta t^n$ for some constant c, what can be deduced about the value of n for this scheme?

3. Higher values of n imply smallers errors, and thus higher accuracy. Implement a higher order scheme of your choice (possibilities include the third-order and fourth-order Runge–Kutta schemes), and repeat the experiment from step 1, again tabulating and graphing your results and calculating n. There are many excellent references for time-stepping schemes for ODEs, such as Press et al. (2007).

Note that it is not necessary to run your code several times and tabulate the results by hand. By introducing appropriate loops, you could write a second program to run the evolution for several different values of Δt, calculate and store the errors, draw a graph, perform the least squares analysis, and return a value for the order of the scheme, n.

In your report, you should include appropriate tables of results (i.e., E versus Δt) and graphs. You should also choose a numerical scheme to use for later investigations, and explain why this is a good choice.

6 Numerical solutions of the Lorenz equations: analysis

6.1 Verification of the fixed point analysis

An obvious first consideration is to use your code to verify the fixed point analysis of Section 4.

It is easiest to start by considering the fixed point at the origin $\mathbf{x}_0 = (0, 0, 0)$. Take an initial condition $\mathbf{x}(0)$ close to $\mathbf{x}_0$ (say within 10^{-6}), integrate the ODEs and plot $D(t) = |\mathbf{x} - \mathbf{x}_0|$. For parameters leading to stability (instability), verify that the rate of exponential decay (growth) $\lambda_{\max}$ of D with t matches that predicted from your analysis. Include a few plots in your report, and discuss how you determine $\lambda_{\max}$ rate in each case. For example, how long an evolution is required? You should also check that $\lambda_{\max}$ is not influenced by your choice of Δt.

Now impose $r > 1$ and consider one of the other fixed points, perhaps $\mathbf{x}_+$. Again take an initial condition $\mathbf{x}(0)$ close to $\mathbf{x}_+$, integrate the ODEs and plot $D(t) = |\mathbf{x} - \mathbf{x}_+|$. What do you notice about the behaviour of D? Again estimate the rate of exponential decay or growth, and compare with that predicted by the fixed-point analysis (i.e., by solving a cubic equation, perhaps using a `python` function). You will probably find that the estimates of $\lambda_{\max}$ are poorer than those obtained for the fixed-point at the origin. Why is this? To get better estimates, you may need to run a trajectory out to larger times. Specifically, you could investigate how $\lambda_{\max}$ changes with $t_{\max}$ when $\sigma = 10$, $b = 8/3$, and $r = 20$; you could try using $t_{\max} = 10$, 20, 30, 40 or 50. If you are feeling brave, you could also try to estimate the imaginary part of λ.

Again taking $\sigma = 10$ and $b = 8/3$, the fixed-point analysis predicts that $\mathbf{x}_\pm$ are unstable for $r > r_c$, where r_c can be found in terms of σ and b. You may have already found an expression for r_c theoretically, but can you now estimate r_c numerically? To do so, estimate $\lambda_{\max}$ for a few values of r between 20 and 30, and then think how the critical value of r_c can be extracted from the data. For example, it might be useful to make a plot of $\lambda_{\max}$ against r.

6.2 Global behaviour

When making further runs, you will find that the long-time behaviour is rather sensitive to the choice of the parameter r. Make plots of trajectories with $\sigma = 10$ and $b = 8/3$ (as in Figure 1), but with different values of r – perhaps 0.5, 1, 2, 5, 10, 15, 20, 25, 30, 50, 100, 200 and 400. Just using a single initial condition may give a misleading picture of the dynamics, so in each case try using two (or more) different values: $(1, 0, 0)$ and $(0, -1, r - 1)$ seem to work well. Ideally, you should calculate the resulting trajectories simultaneously, and then show both trajectories in the same plot.

Investigate any transitions by using smaller increments of r. For larger values of r, you may need to reduce Δt, perhaps to 0.001.

How do your results relate to the fixed-point analysis?

6.3 Poincaré sections

At $\sigma = 10$ and $b = 8/3$, and for sufficiently large r, you should find that the trajectories appear to settle down to a cyclic motion on a two-dimensional surface within three-dimensional space. The surface appears to be made up of two approximately planar sections, and can be clearly seen in Figure 1. However, although such pictures are pretty, it is hard to be precise about the dynamics due to the limitations of a three-dimensional visualisation on a two-dimensional page! A more precise way to visualise the dynamics is in two dimensions, using a *Poincaré section*, which is simply a planar slice through a three-dimensional trajectory such as that of Figure 1.

Write a program to make a Poincaré section for a given trajectory, with $r = 28$. You will first need to choose a plane for the section; $z = r - 1$ is a good choice, which is the level of the two fixed points. Then, at each time step, you will need to perform a test with the updated z-coordinate (z_{new}, say) and the previous z-coordinate (z_{old}). More precisely, you will need to add an (x, y) point to your Poincaré section if $z - (r - 1)$ changes sign.

You will need to take care to get a neat Poincaré section. First, you will need to run your code for a long time to get enough points – perhaps a thousand. Second, you may wish to consider whether or not you wish to use all the points obtained, or simply those over a certain portion of the evolution. Third, you may need to interpolate between $\mathbf{x}_{\text{new}}$ and $\mathbf{x}_{\text{old}}$ to find more precisely the values of x and

y for your plot. Finally, make sure that you plot 'points' rather than 'blobs'; use a command like `plt.plot(xp,yp,'r.',markersize=1)`.

Once you have a neat Poincaré section, generate further sections for different values of r. Start by varying r slowly, and if nothing much seems to be changing then use larger increments. In your report, it might be nice to include Poincaré sections for the same values of r that you used in Section 6.2.

Various interesting things can be done with the Poincaré sections. For example, you might wish to calculate the maximum values of x and y that are obtained as a function of r. You may also wish to generate Poincaré sections in another plane (e.g., $y = 0$).

6.4 Limitations of time-stepping, and predictability

In this section it is suggested that you make some further runs at $\sigma = 10$ and $b = 8/3$, with $\mathbf{x}(0) = (1, 0, 0)$.

Start at $r = 5$. Calculate the evolution out to $t = 1$ using $\Delta t = 0.01$, 0.005, 0.002 and 0.001 (and smaller values if you wish). Note down the final values of (x, y, z) in each case. Now calculate the evolution out to $t = 10$ and $t = 100$, and again note down the final values in each case. Do the solutions appear to converge (as $\Delta t \to 0$), as in Section 5? You may even be able to again verify the order of your time-stepping scheme, perhaps by regarding one run (with the smallest value of Δt) as being 'truth', and calculating errors for the remaining values of Δt.

Now repeat these runs with $r = 28$. Do the solutions still converge (as $\Delta t \to 0$) for all values of t_{max}? At roughly what value of t_{max} does convergence fail to appear? You will need to make additional runs with intermediate values of t_{max} between 10 and 100 to answer this. For values of t_{max} where you don't find convergence, does making Δt even smaller fix the problem?

Can this behaviour be interpreted? You will find it helpful to use plots of a trajectory (for a single Δt) for each t_{max}. At $r = 28$, you may also find it helpful to make runs starting from a different initial condition $\mathbf{x}(0) = (-10, 0, 27)$, and seeing if this shortens the interval over which the solutions converge.

6.5 Further tasks

In your report, it is important for you to discuss and interpret your results, by making links between the different numerical results that you have, for example. You should also be reading related works on the Lorenz equations, and seeing how your results link with what is known about the properties of the equations. For example, your report should contain some discussion of the 'strange attractor' and 'chaos'.

Feel free to experiment with anything that you find to be interesting, or be inspired by experiments that you read about elsewhere. If you are lucky, you might find something that nobody else has ever noticed or studied!

7 Other things to try

1. You might like to investigate other sets of nonlinear equations. Possibilities include (i) the Stommel model, which describes certain aspects of the large-scale ocean circulation; (ii) the Lorenz 84 equations, which describe certain aspects of the large-scale atmospheric circulation; (iii) the coupled Stommel/Lorenz 84 model, which is a simple model of coupled atmosphere-ocean dynamics (and climate!). References for each model, and details of numerical simulations of the coupled model (iii), can be found in Van Veen et al. (2001).

2. The motion of n masses under mutual gravitation is often referred to as the n-body problem. It is easy to write down the governing differential equations, which, as a special case, will

describe the motion of the planets in the solar system, for example. Even the case with $n = 3$ (the three-body problem) can give remarkably complex behaviour. You could try reproducing some of the known solutions of the n-body problem, or try looking for chaotic behaviour.

3. A related system, from fluid dynamics, is that of point vortex dynamics. Again, it is easy to write down the governing equations for the interaction of n point vortices, and, again, the solutions can be remarkably complex.

8 Writing your report

Your report should include standard elements: a table of contents, an Introduction (describing the background to the project and giving an overview of what is included), individual sections for each part of your results, a conclusion (summarising what has been achieved and what more could be done), and a list of references. Supplementary information (such as computer codes) or long proofs that would disrupt the flow of your report may be included as an appendix. This document has the right kind of structure, although your report will be much longer (about 25–30 pages).

In addition to the references included here, you are expected to give details of additional references that you have used when writing your report. These should be cited at the appropriate points in your report. A good report would have more than 10 cited works (books or research papers).

It is hard to write a good scientific report. All assertions and results need to be supported by either an explicit proof, tables of data, graphs, or a citation. Note that Figures (and Tables) must be referred to in the text by their number – so one would write ‘Figure 2’, rather than ‘the Figure at the top of the next page’, for example. Each Figure (and Table) must also have a caption, which typically starts with a single sentence describing the contents, followed by more detailed information (giving parameters, or explaining line colours, etc.). Even when all the components are in place, you will need to read and rewrite your report several times to obtain a polished product. If this is not done, it is almost inevitable that you will not score well for your report, however interesting is the mathematical content.

8.1 Typesetting using LaTeX

As you may know, LaTeX is a typesetting package designed for writing mathematical reports, and it is recommended that you use it for this project. You can access the software in different ways:

1. use LaTeX on the University machines – the best version is called TeXworks.
2. Download LaTeX freely from the web. Nice versions are TeXworks, or TeXshop (for Mac).
3. Use a web interface such as `www.overleaf.com`. This avoids the hassle of installing software, and you can work on your files from anywhere (including from tablets and smartphones).

There are many LaTeX tutorials along with other documentation on the web. For example, you may find it helpful to consult

```
http://www.maths.leeds.ac.uk/latex/
http://www.maths.leeds.ac.uk/latex/texworks23.pdf
```

The LaTeX files used to generate this report are available at

```
http://www.maths.leeds.ac.uk/~sdg/teaching/math3001/lorenz_equations.tex
http://www.maths.leeds.ac.uk/~sdg/teaching/math3001/lorenz_biblio.bib
http://www.maths.leeds.ac.uk/~sdg/teaching/math3001/lorenz.pdf
```

The second file is a bibliography file, which already contains some references that will be useful. Further references can be added by searching for keywords or authors using Google Scholar. Before doing so, on the main welcome page of Google Scholar, select Settings, and then select 'Show links to import citations into BibTeX'.

A Computer programming

You may write your programs in any modern programming language, such as FORTRAN, C, or MATLAB. However, the language python is the obvious choice, since it is used in other modules in years 1 and 2. It is also a versatile language that is increasingly used in a variety of workplaces, so it is a good choice for employability.

There are various ways to run python on different machines. Sometimes it can be invoked by typing python at a command-line shell; for example, on a Mac, you can start a Terminal window (in Applications), and then just type python. There is also a common interactive environment called IDLE, which is available for most platforms. However, it is easiest to interact with python via one of many nice graphical interfaces, such as `spyder`, which is part of the `anaconda` package that you can download for free (for Windows, Mac, or UNIX) from `http://continuum.io/downloads` (or google `anaconda python`). **It is best to download python3 rather than python2**. There is a really good introduction to downloading and using python via anaconda at

`http://www.southampton.ac.uk/~fangohr/blog/installation-of-python-spyder-numpy-sympy-scipy-pytest-matplotlib-via-anaconda.html`

Since this package is used for teaching python at Leeds, it is also installed on University machines. You can find it by searching from the Start Menu, or by looking in All Programs / Departmental Software, although it is often tucked away in a further subfolder (e.g., All Programs / Departmental Software / Engineering / Computing, or All Programs / Departmental Software / MAPS / Maths). Again, ensure that you choose python3 rather than python2.

Once you have figured out how to run a python code, save this simple code as `test.py`:

```
a=1
b=2
c=a+b
print(c)
```

Run the code; the output should be the number 3, printed somewhere on screen. There is no point going further until you can get this simple code to work!

A.1 Finding eigenvalues and eigenvectors

Here is a simple python code that finds the eigenvalues of the Jacobian matrix of the linearized Lorenz equations around the fixed point at $(0, 0, 0)$:

```
import numpy as np

# set standard parameters
b=8/3
sigma=10
r=1.1

# make a 3x3 matrix
a=np.array([ [-sigma,sigma,0], [r,-1,0], [0,0,-b] ])

print(a)
```

```
# calculate eigenvalues and eigenvectors
evals,evecs=np.linalg.eig(a)

print(evals)
```

A.2 Python implementation of the Lorenz equations

Here is a simple python code that timesteps the simplified Lorenz equations (2) and plots the results:

```
# time steps the Lorenz equations
import numpy as np, matplotlib.pyplot as pt, timestepping as ts

# parameters
b=8/3
r=28
sigma=10

# time-stepping parameters
tmax=1         # run to this time
nsteps=100     # number of time steps
dt=tmax/nsteps # calculate the time step

# initial conditions
t=0
x=np.array([1,-1,20])

n=0  # number of timesteps taken; initialise to 0
while n < nsteps:
    x=ts.step_rk2(x,dt,b,r,sigma)
    t=t+dt
    n=n+1
    # plot the three variables (red, blue, green dots):
    pt.plot(t,x[0],'r.')
    pt.plot(t,x[1],'b.')
    pt.plot(t,x[2],'g.')

print(t,x[0],x[1],x[2]) # print the final time and coordinates

pt.savefig('test.pdf') # save the plot to a file
```

In the second line of the code, three modules are loaded: `numpy` (which is a standard module for numerical calculations in python), `matplotlib.pyplot` (which is a standard module for plotting in python), and `timestepping`, which contains some additional routines for time-stepping the Lorenz equations:

```
def step_rk2(x,dt,b,r,sigma):

    # advances the input array x (at time t)
    # to the returned value x_out (at time t+dt)

    dxdt1=calc_dxdt(x,b,r,sigma)
    xtemp1=x+0.5*dt*dxdt1
    dxdt2=calc_dxdt(xtemp1,b,r,sigma)
    x_out=x+dt*dxdt2

    return x_out

def calc_dxdt(x,b,r,sigma):

    # calculates the vector dx/dt for the Lorenz equations

    dxdt=0*x
    dxdt[0]=sigma*(x[1]-x[0])
    dxdt[1]=r*x[0]-x[1]-x[0]*x[2]
    dxdt[2]=x[0]*x[1]-b*x[2]

    return dxdt
```

This module should be saved as `timestepping.py` in the same directory (folder) as the main program, which should also carry the extension `.py`.

A.3 Plotting

Visualisation of your results is key, so you will need to get familiar with `matplotlib.pyplot`. Documentation and examples can be viewed at

`http://matplotlib.org/index.html`
`http://www.loria.fr/~rougier/teaching/matplotlib/`

References

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes 3rd edition: the art of scientific computing*. Cambridge University Press.

Sparrow, C. (1982). *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*. Springer.

Strogatz, S. (1994). *Nonlinear Dynamics and Chaos*. Perseus Books.

Van Veen, L., Opsteegh, T., and Verhulst, F. (2001). Active and passive ocean regimes in a low-order climate model. *Tellus*, 53 A:616–628.