n-Dimensional LCE code

Dale L. Peterson
Department of Mechanical and Aeronautical Engineering
University of California, Davis
dlpeterson@ucdavis.edu

June 10, 2007

Abstract

The Lyapunov characteristic exponents (henceforth LCE's) provide a quantitative numerical indicator of chaos. A tool was developed in Python that calculates all n LCE's for an n-dimensional system of continuous first order ODE's, and additionally calculates the Kaplan-Yorke dimension of the system. This tool is released under the GNU General Public License. It is then applied to various systems, including the Lorenz system, the Rössler system, and the frictionless pendulum. The results were found to be consistent with published results found for the Lorenz and Rössler systems. The pendulum is not a chaotic system, and the simulation results agreed with this. The algorithms in the code are well commented, and only the general mathematics are discussed in this paper.

Introduction

The original aim of this project was to compute the Lyapunov spectrum of the three body problem, and to search for different mass ratios that would result in chaotic behavior. The three body problem is a model of three particles that move in response to the forces that they exert upon each other due to gravitational attraction. The formulation of the equations of motion for the three body problem is relatively simple. Technical issues regarding collisions or near collisions of the particles and the numerical accuracy of such solutions led me to focus on applying my code to simpler, more easily verifiable systems.

Why are we interested in the LCE's, and what exactly do they tell us? A rigorous definition of chaos has historically proven to be a bit slippery, but the numerical (as opposed to analytical) computation of the LCE's provides an approach that is commonly accepted as a way to determine if chaos is present in a system. Knowing whether or not chaos is present in a system, and to what degree it is present, is important if you have any desire to make long term predictions of the behavior of a system. What the LCE's tell you is exactly this – whether or not your system is chaotic. If a system has one or more positive LCE's, it means that arbitrarily close initial conditions will grow exponentially far apart, thus reducing the ability to predict the long term behavior. The magnitude of the positive LCE's indicate how quickly these initial conditions separate, thereby giving one an idea of how long it might be feasible to predict the behavior of a system before predictions become meaningless.

Technical Development

The theoretical development of the calculation of the LCE's is rather lengthy, see [1] and [2] for a rigorous development of the theory and the numerical application. The basic idea is as follows. Given a dynamical system of the form:

$$\begin{aligned} \dot{x} &= f(x,t) \qquad x \in \mathbb{R}^n, \qquad t \in \mathbb{R} \\ f : \mathbb{R}^n \times \mathbb{R} &\to \mathbb{R}^n \end{aligned}$$

This system can be linearized about any point in $\mathbb{R}^n$, and this linearized mapping governs the local dynamics near the point of linearization. It is this linearized flow that allows us to compute the LCE's, simply by time averaging the amount of stretch that a spanning set of basis vectors undergo as they flow along the trajectory. This will be discussed in more detail later. The linearized flow at an arbitrary point $x \in \mathbb{R}^n$ is then simply:

$$\delta\dot{x} = \left.\frac{\partial f}{\partial x}\right|_{(x,t)} \delta x = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}_{(x,t)} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \end{bmatrix}$$

Where $\frac{\partial f}{\partial x}$ is the Jacobian matrix. The above equation is used to evaluate how the basis vectors are changed along trajectories of the system. So, given an n-dimensional system, we need to integrate n first order ODE's to find the trajectory of the system. Additionally, we need to integrate the linearized flow for each of the n basis vectors in $\mathbb{R}^n$. This amounts to $n(n+1)$ first order ODE's.

From an implementation point of view, one can see that for small systems ($n \leq 3$), it wouldn't be too difficult to hard code the above equations, but since the number of equations grows with n^2, at some point it becomes more efficient to generalize the above procedure. To do so, first, define the composite state vector as:

$$\bar{x}^T = [x \quad e_1 \quad \ldots \quad e_n] \in \mathbb{R}^{n(n+1)}$$

Where each $e_i \in \mathbb{R}^n$ is a basis vector at the point of linearization along the trajectory. Then we can write the composite differential equations as:

$$\begin{bmatrix} \dot{x} \\ \dot{e}_1 \\ \vdots \\ \dot{e}_n \end{bmatrix} = \begin{bmatrix} f(x,t) \\ \frac{\partial f}{\partial x} e_1 \\ \vdots \\ \frac{\partial f}{\partial x} e_n \end{bmatrix}$$

Where $\frac{\partial f}{\partial x}$ denotes the Jacobian matrix evaluated at the current point (and time) along the trajectory. By inspection, one can see that this is indeed a system of coupled ordinary differential equations. The right hand side of this composite system of differential equations can conveniently be generated given x, t, $f(x,t)$ and $\frac{\partial f}{\partial x}(x,t)$. The tool developed in Python takes these function definitions and generates the function for the composite system, which is then easily numerically integrated. For the initial condition of the state, one should simply choose an initial condition that isn't in a basin of attraction for a stable fixed point or limit cycle. As for the initial conditions of the basis vectors, the easiest choice is the standard basis in $\mathbb{R}^n$. In fact, any linearly independent basis will suffice, it has been shown in [1] that the LCE's are independent of the basis vectors that are chosen.

The procedure for calculating the LCE's is conceptually straightforward. For a good understanding of it, one must review the Gram-Schmidt orthonormalization procedure that takes a basis and generates an orthonormal basis. The Gram-Schmidt procedure is an iterative procedure that begins with the first vector in the list, normalizes it and uses this as the first normal basis vector. Each sebsequent basis vector is then generated by taking the next vector in the original list, subtracting from it any components in the direction of the new orthonormal basis vectors, and then normalizing. In order to determine the LCE's, one needs to take the time average of the stretching of each of the basis vectors *in each of the orthogonal directions*. This amounts to accumulating the Euclidean 2-norm of each of the orthogonalized basis vectors, which can be efficiently implemented as an intermediate step in the Gram-Schmidt procedure. It is essential to understand that the norms to be accumulated are *not* just the norms of each of the basis vectors after they have evolved for some period of time – one must compute the norm of the vector that is orthogonal to the previous orthonormal vectors in the list.

Given a basis $\{e_1 \quad \ldots \quad e_n\}$, the procedure is as follows:

$$\begin{aligned}
d_1 &= ||e_1||_2 \\
\tilde{e}_1 &= \frac{e_1}{d_1} \\
d_2 &= ||e_2 - < e_2, \tilde{e}_1 > \tilde{e}_1||_2 \\
\tilde{e}_2 &= \frac{e_2 - < e_2, \tilde{e}_1 > \tilde{e}_1}{d_2} \\
\ldots &= \ldots \\
d_i &= \left|\left| e_i - \sum_{j=1}^{i-1} < e_i, \tilde{e}_j > \tilde{e}_j \right|\right|_2 \\
\tilde{e}_i &= \frac{e_i - \sum_{j=1}^{i-1} < e_i, \tilde{e}_j > \tilde{e}_j}{d_i}
\end{aligned}$$

Where $< *, * >$ is the standard inner product on $\mathbb{R}^n$, and $||*||_2$ is the standard Euclidean 2-norm. The LCE's are then simply:

$$\lambda_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \ln\left(d_i(t)\right)$$

In practice, we cannot integrate indefinitely, and we can only compute each d_i at some multiple of the integration time step, so this becomes:

$$\lambda_i \approx \frac{1}{N_{lce} T_{pb}} \sum_{k=1}^{N_{lce}} \ln\left(d_i(kT_{pb})\right)$$

Where T_{pb} is the period of time between each 'pull-back' or orthonormalization procedure, and N_{lce} is the number of 'pull-back' iterations to perform during the computation of the LCE's.

This procedure of normalizing the basis vectors can be done at varying time intervals. Conceptually, it would makes sense to do it continuously, but due to the discrete nature of the numerical implementation, one can perform it at most once per integration time step. In practice, this procedure can be performed even less frequently and still achieve reliable results. However, if one doesn't perform the normalization process frequently enough, the basis vectors will tend to align in the direction of the largest LCE, and numerical errors will arise due to the vectors becoming very closely aligned, as well as some becoming very long and others becoming very short, both of which will degrade the accuracy of the Gram-Schmidt procedure. In all of the studies in this project, the normalization procedure was performed once every 10 integration time steps, although this is a parameter that can be changed in the function call of `ComputeLCE()`.

Once the LCE's have been computed, the Kaplan - Yorke dimension of the system can be easily computed. Ordering the LCE's from largest to smallest, $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$, the Kaplan - York dimension is then:

$$D_{KY} \quad = \quad j + \frac{\sum_{i=1}^{j} \lambda_i}{|\lambda_{j+1}|}$$

Where j is the largest integer for which $\lambda_1 + \lambda_2 + \ldots + \lambda_j \geq 0$. The Kaplan - Yorke dimension is an upper bound on the information dimension of the system [4]. This is implemented as `KaplanYorkeDimension()` in the code.

Dynamical Systems

The code was applied to three dynamical systems, two of which are known to be chaotic, and one of which is not chaotic. Outlined here are the equations of motion for each system. Also required for the computation is each system's Jacobian matrix, which is easily determined from the equations of motion.

Lorenz

The Lorenz system has three positive parameters: $\sigma, r, b \geq 0$. These parameters were taken to be $\sigma = 10.0$, $r = 28.0$, and $b = \frac{8}{3}$.

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= (r - z)x - y \\ \dot{z} &= xy - bz \end{aligned}$$

Rössler

The Rössler system is the simplest known system that exhibits chaos because it has only one nonlinear term. The parameters chosen are the traditional parameters as studied by Rössler, $a = b = 0.2$, and $c = 5.7$.

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + ay \\ \dot{z} &= b + z(x - c) \end{aligned}$$

Frictionless Pendulum

The pendulum is a well studied system that is not chaotic. The dynamics of the pendulum are similar to many processes in real life: flying a rocket to outer space (must keep the nose pointed up) and even the dynamics of a vehicle under front or rear wheel braking exhibit similar dynamics to that of a pendulum. The pendulum system has two parameters: g and l. The downward position of the pendulum is taken to be $\theta = 0$. Letting $x := \theta$, and $y := \dot{\theta}$, the equations of motion can be written as:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\tfrac{g}{l} \sin x \end{aligned}$$

This system was studied for the case when $g = l = 1$

Results

All simulations used the following parameters:

- t_0: 0.0 s
- dt: 0.01 s
- T_{pb}: 0.1 s
- N_{trans}: 100
- N_{lce}: 100000

N_{trans} is the number of pullback iterations to integrate the ODE's before starting the calculation of the LCE's. This parameter allows one to let a system settle down to an attractor before beginning the LCE computation. All numerical integration was done with the `scipy.odeint` module, which is simply a wrapper to the LSODA integration routine developed by Linda R. Petzold and Alan C. Hindmarsh at Lawrence Livermore [5]. This code adaptively adjusts the integration method between a stiff backward differentiation technique and a non-stiff Adams linear multi-step technique. More details can be found at: http://www.netlib.org/odepack/opkd-sum

The following results were obtained for the three systems:

1. Lorenz System:

   LCE's: (0.0906509485, 0.000635914405, -14.5738118)

   D_{ky}**:** 2.06220126187

2. Rössler System:

   LCE's: (0.0709792970, 0.0000132794085, -5.39406910)

   D_{ky}**:** 2.01315876674

3. Pendulum System:

   LCE's: (0.00060355, -0.00060395)

   D_{ky}**:** 1.0

The positive LCE's seen in the Lorenz and Rössler systems indicate the presence of chaos. Additionally, the Kaplan Yorke dimension of the Lorenz is slight higher than that of the Rössler. The pendulum is a two dimensional system, and thus cannot exhibit chaos. The numerical results indicate two very small LCE's, one positive and one negative, both of nearly the same magnitude. This is an indicator that the system actually has two zero LCE's, and if the simulation was run for longer periods of time, these two LCE's would continue to shrink.

The results of the Lorenz system are identical to the results obtained by Professor Crutchfield's code developed for the course PHYS 250 at University of California, Davis. Additionally, other authors have published the same results for the Lorenz and Rössler systems when using the same parameters [6].

Conclusions and Future Work

A general tool for numerically estimating the LCE's of an n-dimensional system was developed. It was validated on two well studied chaotic systems in order to verify its correctness. Python was found to be a great asset in the development of a flexible general code that was readable, reliable, and relatively fast.

Further work to be done in this area would be to apply this code to other systems to test for chaotic behavior. Integrating this tool with PyDSTool [3] is another project to be undertaken in the near future, and I have been in correspondence with the authors at Cornell University regarding collaboration on this front. Finally, I plan on using this tool to test for the presence of chaos in a benchmark model of a bicycle. The bicycle is a challenging nonholonomic system that has very interesting dynamics, and up to this point, the presence of chaos in the bicycle is unknown.

References

[1] Giancarlo Benettin, Luigi Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15:9–20, 1980.

[2] Giancarlo Benettin, Luigi Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 2: Application. *Meccanica*, 15:21–30, 1980.

[3] Rob Clewley. Python dynamical systems tool. http://www.cam.cornell.edu/ rclewley/cgi-bin/moin.cgi.

[4] I. Grassberger, P. Procaccia. Measuring the strangeness of strange attractors. *Physica D*, 9:189–208, 1983.

[5] Alan C. Hindmarsh. ODEPACK, A systematized collection of ODE solvers. In R. S. Stepleman, M. Carver, R. Peskin, W. F. Ames, and R. Vichnevetsky, editors, *Scientific Computing*, pages 55–64. North-Holland, Amsterdam, 1993. Volume 1 of IMACS Transactions on Scientific Computation, report also available as Lawrence Livermore National Laboratories Report UCRL-88007, August 1982.

[6] Julien Clinton Sprott. Common chaotic systems. http://sprott.physics.wisc.edu/chaos/comchaos.htm.