Modeling Dynamical Systems for 3D Printing

Stephen K. Lucas, Evelyn Sander, and Laura Taalman

1. Motivation

As 3D printers become increasingly common in science and engineering, they are also making their way into mathematics departments. One potential use is in visualizing dynamical systems. These mathematical structures illustrate some of the important progress in roughly the last sixty years regarding approaches to understanding the role of dynamical models in scientific research. In addition to the illustration of principles, we are in no small part motivated by the beautiful three-dimensional structures of strange chaotic attractors, such as can be seen in [Atm]. See for example the Langford attractor in Figure 1. The aim of this paper is to describe how to practically 3D print actual physical models of such dynamical structures.

Stephen K. Lucas is a professor of mathematics at James Madison University. His email address is lucassk@jmu.edu.

Evelyn Sander is a professor of mathematics at George Mason University. Her email address is esander@gmu.edu.

Laura Taalman is a professor of mathematics at James Madison University. Her email address is taalmala@jmu.edu.

DOI: https://doi.org/10.1090/noti2187

In many cases, it is possible to use a black box differential equations solver to create printable objects. However, in some cases, a black box method is inadequate. We have therefore developed a mixed curvature method to make printing possible in these cases. We mostly restrict our discussion to solutions of differential equations, but we end with a description of how 3D printing can be applied in the context of more general dynamical structures for iterated maps and ordinary and partial differential equations.

Our paper proceeds as follows: In Section 2, we give a straightforward method for generating 3D printable chaotic attractors using built in commands in Mathematica. In Section 3, we describe a new method for creating more visually accurate 3D printable chaotic attractors. This mixed curvature method uses a combination of Matlab and the cost-free software OpenSCAD [Kin], designed specifically for 3D print design. In Section 4 we describe our future directions in creating printable chaotic and dynamical objects. In Appendix A conditions of the objects shown in the figures. In Appendix B, we give a brief introduction to the nuts and bolts of 3D printing so that a reader will be able to use the methods and codes provided in this article to create their own printed attractors.

Figure 1. The Langford chaotic attractor in equation (8) (often erroneously referred to as the Aizawa attractor), modeled in Mathematica and 3D printed in FDM PLA.

Figure 2. The Rössler chaotic attractor in equation (5), modeled in Matlab and OpenSCAD and 3D printed in SLS nylon.

Appendix C contains a set of detailed annotated descriptions of all of the code needed to create the chaotic attractors in this article. The full version of all code is available for download at [LST20].

2. Straightforward Approach

Strange chaotic attractors have interesting dynamical properties, such as the plethora of periodic orbits (in fact infinitely many) and sensitive dependence on initial conditions, meaning that no matter how close solutions start, they will diverge over time [Rue06]. These properties are not visible when looking at the printed attractor. Rather, one sees the strangeness, or fractal, structure of the attractor. Any cross-section of the attractors shown here consists of a Cantor set: the fractal dimension of the attractors is between 1 and 2. This can be seen particularly well in the Rössler attractor in Figure 2, the Lorenz attractor in Figure 3, the Rucklidge attractor in Figure 4, and the Anishchenko attractor in Figure 5.

From the practical point of view of generating chaotic attractors, the most useful properties are the existence of a dense orbit, and the fact that the set is attracting. The first property implies that we can create an arbitrarily good approximation of the chaotic attractor with a single solution starting within the attractor. The second property implies that we do not even need to start with a solution within the attractor. In particular, as long as we start our solution close enough to the attractor, the solution will converge to the attractor as time increases. Based on these principles, in this section we describe how to generate a 3D printable

Figure 3. The Lorenz chaotic attractor in equation (4), modeled in Mathematica and 3D printed in FDM PLA.

solution to a given differential equation starting at a specified initial point.

Mathematical software packages such as Mathematica and Matlab have built in routines for solving systems of first order ordinary differential equations. They allow for

Figure 4. The Rucklidge chaotic attractor in equation (10), modeled in Matlab and OpenSCAD and 3D printed in SLS nylon.

Figure 5. The Anishchenko chaotic attractor in equation (11), modeled in Matlab and OpenSCAD and 3D printed in SLS nylon.

the creation of solutions to differential equations without prior knowledge of numerical methods, and relatively minimal knowledge of the theory of differential equations. Here we present a method using the `NDSolve` command in Mathematica. The choice of Mathematica over Matlab

Figure 6. The Langford chaotic attractor in equation (8), modeled in Matlab and OpenSCAD and 3D printed in SLA resin.

is based on the fact that the package has better developed methods for creating 3D printable objects, meaning that it is possible to generate data for and create a printable (STL format) file within a single software package. This has been successfully used in a class consisting of undergraduate students who had only seen elementary differential equations and had never seen any numerical methods for solving differential equations. They were equipped with the code in Appendix C.1, and were able to adapt it to generate and print many of the examples listed in Appendix A.

The Mathematica command `NDSolve` is proprietary and to some extent a black box, though it is possible to change some of the default settings. In particular, the user inputs an equation of the form

$$\dot{\mathbf{x}} = f(t, \mathbf{x}),\ \mathbf{x}(t_0) = \mathbf{x}_0,$$

where $\mathbf{x}(t), \mathbf{x}_0(t) \in R^n$ and $f : R \times R^n \to R^n$. Here and in all subsequent equations, we use the notation $\dot{\mathbf{x}}$ to represent the derivative of $\mathbf{x}$. In most of the cases we consider, $n = 3$ since that is the dimension of our printing space. The input of the method includes the form of the equation, the initial condition, and the minimum $t_0 = 0$ and maximum T time values for which a solution should be calculated. The method is an adaptive Runge-Kutta one that makes an approximation of the error and correspondingly adjusts the size of the time step. It outputs an interpolation for $\mathbf{x}(t)$ on a discrete but unevenly spaced set of $t \in [0, T]$. The method determines the spacing of the t values such that the approximate solution achieves a desired accuracy. That is, at each t if we denote by $\mathbf{x}_{exact}(t)$ and $\mathbf{x}_{approx}(t)$ the exact and approximate solutions, resp., then the method computes approximations with prescribed small values of the absolute error $|\mathbf{x}_{exact}(t) - \mathbf{x}_{approx}(t)|$ and relative error

$|\mathbf{x}_{exact}(t) - \mathbf{x}_{approx}(t)|/|\mathbf{x}_{exact}(t)|$. Note that $\mathbf{x}_{exact}$ is unknown, but the method still is able to say with some confidence that the approximate solution is sufficiently close to the exact one in terms of both of these measures of error. When these desired accuracy requirements are not set by the user, they remain at default values internal within the software package.

After generating an approximation in the form of an interpolation for $\mathbf{x}(t)$ for $t \in [0, T]$, we can then use the Mathematica command `ParametricPlot3D` to create a piecewise smooth curve approximating the exact solution. By using the `PlotPoints` option with a large value such as 100, we are able to control the level of smoothness of the curve. At this stage, the curve is not a solid printable object since it is infinitely thin. We use the `Tube` graphing option to create a tube of a specified thickness around the curve, thereby making it into a solid and therefore printable object. This solid object can be saved in printable STL format. See Appendix B for more information on how to proceed from an STL file to a physical object.

While our primary goal is to create chaotic attractors for ordinary differential equations, with minimal extra effort the same commands in Mathematica can be used to generate solutions to delay differential equations. Figure 8 depicts a chaotic attractor for the Mackey-Glass delay differential equation. In a delay differential equation, the derivative $\dot{\mathbf{x}}(t)$ depends not only on t and $\mathbf{x}(t)$, but also on the value of $\mathbf{x}$ at a previous time $t - \tau$ for $\tau > 0$. In addition to chaotic attractors, the method works equally well for generating other types of solutions to differential equations. For example, Figure 7 shows a quasiperiodic torus solution to the Hénon-Heiles equation.

Figure 7. The Hénon-Heiles quasiperiodic set in equation (9), projected to three dimensions, modeled in Mathematica and 3D printed in FDM PLA.

Figure 8. The Mackey-Glass attractor in equation (12), projected to three dimensions, modeled in Mathematica and 3D printed in FDM PLA.

Figures 1, 3, 7, 8, and 9 were 3D printed using this straightforward approach.

3. Mixed Curvature Method

In this section, we present a new mixed curvature method for preparing a 3D printable differential equation solution. The method produces a file with a smaller number of data points, but still with the same or in some cases greater visual accuracy. In many cases, the straightforward approach

described in Section 2 is completely sufficient for generating a visibly excellent approximation of a chaotic attractor. For example, Figures 9 and 10 are both equally acceptable 3D printed representations of the Arneodo attractor, where Figure 9 was created using the straightforward method and Figure 10 used the mixed curvature approach. The same applies to the Langford attractor shown in Figures 1 and 6.

There are, however, cases where the straightforward approach does not lead to a visually accurate object, usually when the curvature of the solution curve varies substantially along its length. The straightforward approach tends to produce curves that lack the required smoothness in the high curvature regions, since the data distribution is not considering the shape of the object that we wish to eventually print. Attempts to fix this by increasing the requested density of points leads to a vast data set that is difficult for printers to work with, and contains superfluous data in the low curvature areas of the curve. An example where this behavior is particularly noticeable is when attempting to visualize the solution to the autocatalytic chemical reaction system [GS90]

$$\begin{aligned} \dot{w} &= -0.002w, \\ \dot{x} &= 0.002w - 0.08x - xy^2, \\ \dot{y} &= 0.08x + xy^2 - y, \\ \dot{z} &= y, \end{aligned} \tag{1}$$

with $w(0) = 500$, $x(0) = 0$, $y(0) = 0$, and $z(0) = 0$. This system does not have an analytic solution (apart from $w = 500e^{-0.002t}$), but it can be approximated to prescribed accuracy using a standard numerical solver, such as `NDSolve` in Mathematica, or `ode45` in Matlab. Figure 11 shows the solutions produced using `ode45` to the autocatalytic system with the default settings on the interval $t \in [0, 1000]$.

Figure 9. The Arneodo chaotic attractor in equation (7), modeled using the straighforward method in Mathematica and 3D printed in FDM PLA.

Figure 10. The Arneodo chaotic attractor in equation (7), modeled using the mixed curvature method in Matlab and OpenSCAD, and 3D printed in SLS nylon.

Two plots are shown due to the very different size scales involved in the solutions, with the `ode45` approximation containing 2093 data points.

However, a more intuitive visualization is to plot a projection in three-dimensional space of the position $(x(t), y(t), z(t))$. Dividing z by one hundred and making the axes of equal length, we get the picture in Figure 14. Note that if we didn't scale the z variable, then the curve would be extremely long and thin, and inappropriate for 3D printing. While not as compact as the chaotic attractors, this is still an interesting object for 3D printing, as shown in Figure 15. In all that follows, we scale the z axis for the autocatalytic system.

3.1. Default point placement.

Unfortunately, the data that produced Figures 11 and 14 is not particularly

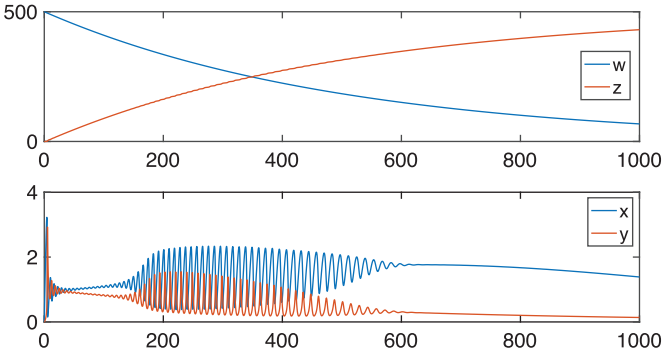

Figure 11. Solutions to the autocatalytic system in equation (1) using ode45 as a function of time.

useful when developing a structure for 3D printing. If one is not using the built-in Mathematica routines, the standard approach is to take a set of data points along the curve, join them by straight line segments, then expand them into tubes with some given cross-section of a pleasing radius. This step is identical to the use of the Mathematica `Tube` command used in the straightforward approach. Having the points placed so that the joins between the tubes don't form obvious corners leads to a structure that is more pleasing to the eye. Unfortunately, the typical data output from a numerical solver is not of this form. An adaptive solver like `ode45` will try and use the minimum number of points in time to satisfy some requested error bound over the interval of interest. Even though solutions between data points can be interpolated, the velocity along a curve will usually vary, and straight line tubes will be of wildly varying lengths. In this particular example, the default `ode45` output looks quite poor in the middle parts of the solution (see Figure 13), where the x and y components are oscillating.

3.2. **Equally spaced points in space.** As a first attempt beyond default output data points, we require the data points that define the tube endpoints to be equally spaced along the curve in the three-dimensional phase space. This can be done most efficiently by adding an additional differential equation to the original set that gives the length of the curve so far as a function of time. Given that the arc length of the curve defined by $(x(t), y(t), z(t))$, $0 \le t \le T$, is

$$s(t) = \int_0^t \sqrt{\dot{x}(u) + \dot{y}(u) + \dot{z}(u)}\, du,$$

taking the derivative gives us

$$\dot{s}(t) = \sqrt{\dot{x}(t) + \dot{y}(t) + \dot{z}(t)} \quad \text{with} \quad s(0) = 0.$$

Formulas for $\dot{x}$, $\dot{y}$, and $\dot{z}$ are given in the original system of differential equations. For example, Figure 12 shows the length of the curve for the autocatalytic system, with z divided by 100, as a function of time. Clearly, the velocity is not constant along this curve, which has total length 163.45, and the default output produces line segments that vary in length from 5.02×10^{-5} to 4.53.

To produce data points equally spaced in terms of distance along a curve on some time interval $[0, T]$, since we have the distance function $s(t)$, we want data at times $\{t_i\}_{i=0}^m$ that satisfy

$$s(t_i) - \frac{is(T)}{m} = 0 \quad \text{for} \quad i = 0, 1, \ldots, m,$$

where $t_0 = 0$ and $t_m = T$. An easy way to solve for the t_i's assumes we know $s(t)$ for $0 \le t \le T$ and then use the secant method with the initial guesses t_i and $t_i + 10^{-3}$ when trying to find t_{i+1}. While Matlab's `ode45` outputs the solutions at particular points, it allows for interpolation between these data points, so we can find the length of the curve at any time. Since s is an increasing function, the secant method is a robust way of finding these data points.

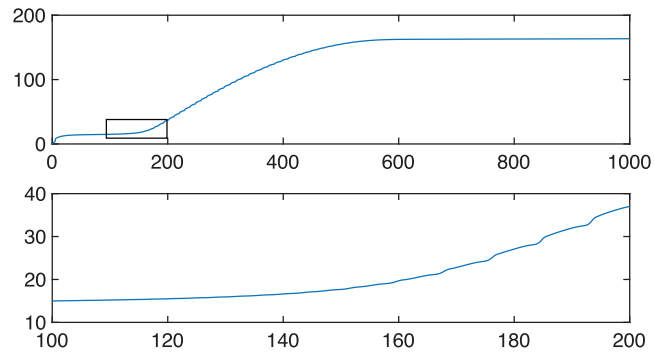

Figure 12. Length of the autocatalytic curve as a function of time (scaled in z direction). The lower graph shows a closer view of the boxed region outlined in the upper graph.

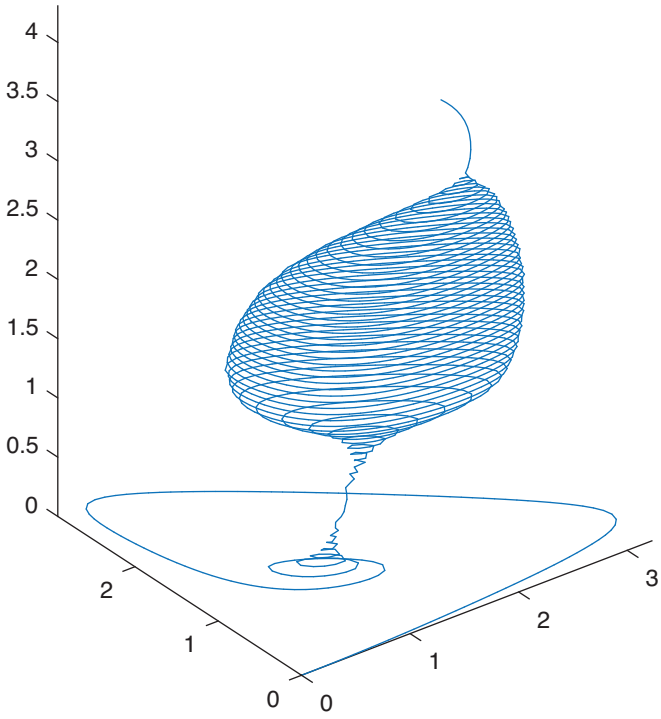

Figure 13. Straight line segments for the autocatalytic system in equation (1) scaled in the z direction with 3000 equally spaced (in terms of arc length) points.

For example, Figure 13 shows the three-dimensional curve for the autocatalytic problem with 3000 equally spaced points in terms of arc length. It is immediately obvious that this is not a particularly good representation. Even with more data points than output from `ode45`, there are places along the curve where it is clearly not smooth, particularly in the lower section after the initial spiral inwards.

3.3. **Bounding curvature.** The problem with the equally spaced approach is that regions of the curve with high curvature will lead to straight line segments with a noticeable corner between them. An alternative approach, then, is to choose data points so that the maximum distance from the actual curve to the approximating straight line segment has an upper bound. The radius of curvature of the curve could be calculated along the curve with some additional analysis, and keeping the curve close to the line segments is equivalent to bounding the angle of the sector of the osculating circle that the curve moves through between data points.

While it is theoretically possible to calculate the radius of curvature along the curve and choose an interval length associated with the smallest radius of curvature on the interval, it turns out to be a challenging numerical exercise not worth detailing here. Even worse, we found that simply using the radius of curvature was also a poor choice. Choosing interval lengths to place a sufficient number of points in regions of high curvature means far too many points were placed in regions where the curvature is small.

3.4. **Mixed curvature method.** Ideally we wish to have a combination approach, with some upper bound on the arc length between points when the curvature is low, then placing more points where the curvature is high. While sophisticated analysis is possible, we suggest the following simple approach that works well in every situation we have considered. Start with the equally spaced in space approach, giving some initial length of line segments that initially appears reasonable when the curvature is not high. Then, if either of the angles at the joins of the current line segment with its neighbors is too small, less than some angle $(180 - d)^\circ$, bisect the line segment by adding a data point at the middle of the arc length curve, splitting the original line segment into two equal length pieces, thus increasing the density of data points in regions of high curvature. If any line segment has been split, repeat a full pass over the current set of line segments, splitting as necessary, until eventually all of the angles between line segments are sufficiently close to 180°.

For the autocatalytic example with time interval $[0, 1000]$, we chose to start with 1000 equally spaced points and angle bound $d = 10$. The choice of 1000 points made the relatively straight parts of the curve look good: even small angles are more noticeable between straight line segments when they are long. After six passes subdividing problematic intervals, we ended up with 4012 data points and the picture in Figure 14.

Once data points for a curve have been established by the mixed curvature method, we need to transform from line segments to tubes and form an STL format file for 3D printing. Our experience thus far is that this process is not particularly straightforward or successful within

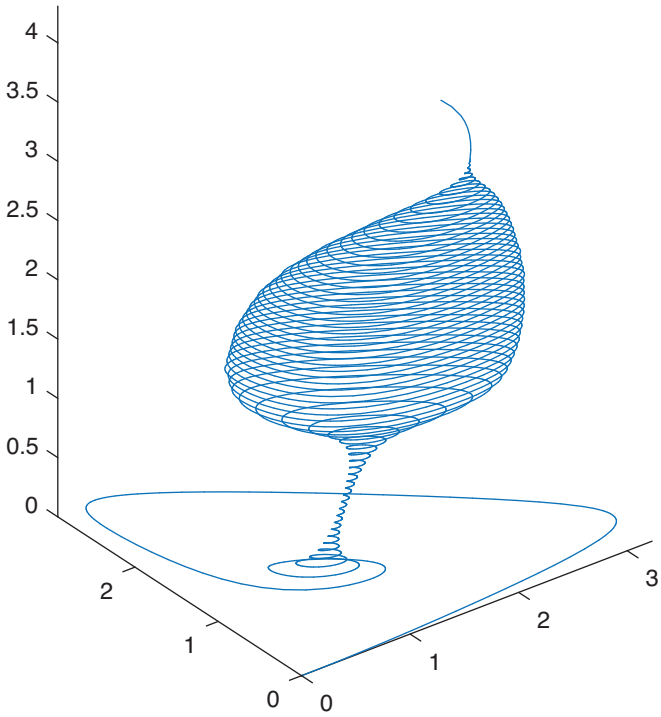

Figure 14. Straight line segments for the autocatalytic system scaled in the z direction with initially 1000 points and minimum angle 170°.

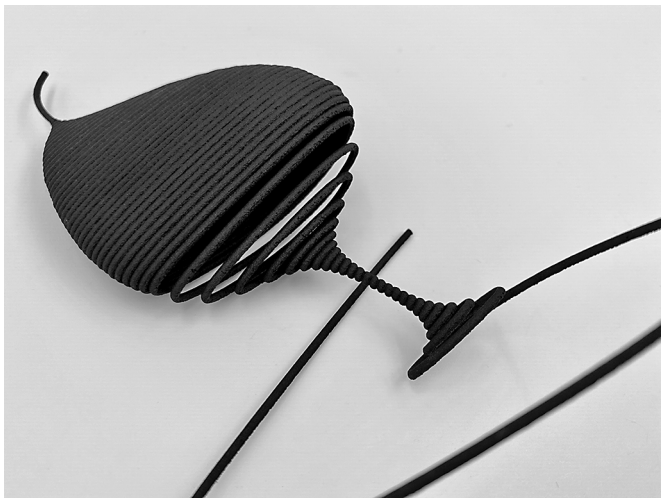

Figure 15. Autocatalytic system in equation (1) with scaled z variable modeled with the mixed curvature method and 3D printed in SLS Nylon.

Matlab, and so we turned to using the application OpenSCAD [Kin] to take as input this data, produce a tubular structure, then form an STL file. Figures 2, 4, 5, 6, 10, 15, and 21 were 3D printed using the mixed curvature method.

It is appropriate at this time to compare the STL files formed by the straightforward and mixed curvature method approaches. As we have stated, for most of the chaotic attractors we have printed and shown previously,

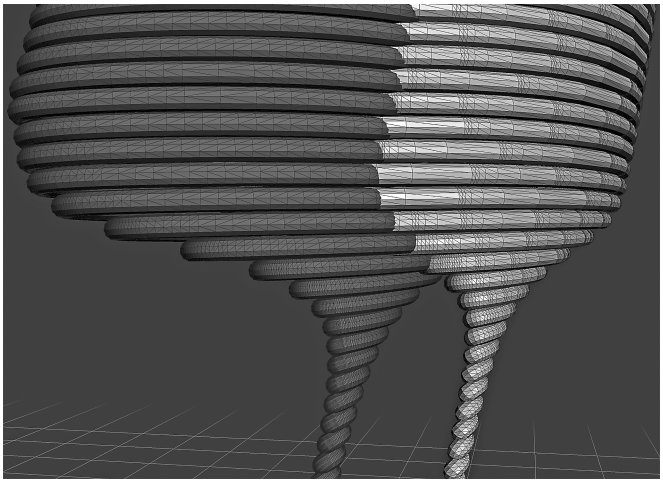

Figure 16. Mesh comparison of straightforward (light model) and mixed curvature (dark model) methods for the scaled autocatalytic system. Note the higher accuracy and more appropriate mesh variation in the mixed curvature model.

it is very difficult to tell which approach has been used. Most of the time, the straightforward approach is by far the simplest and best. But Figure 16 places the mesh formed by the two methods in the same visualization space. It is obvious that the mixed curvature model provides a more pleasing object to print.

4. Future Directions

While it is not hard to find examples of 3D printed objects related to both multivariable calculus and geometry, there is relatively less material available on the topic of 3D printing in dynamical systems. The examples in this article are only one step in filling such a gap, and there is plenty of room for future work in this direction. In particular, there are many dynamical objects beyond solutions to and attractors of differential equations that are well suited for 3D printing. We include a few examples of possible future directions below.

The methods described in this paper can be used to consider differential equations which vary with respect to a parameter, and used to create a series of attractors in three dimensions as a parameter varies. This allows for the visualization of bifurcations in attracting sets. An example of this appeared in [Gag18].

Attractors of three-dimensional iterated maps are equally well suited to 3D printing as those of differential equations. However, since the orbit of an iterated map consists of a set of disconnected points which only form a connected set when combined, it is more difficult to create a printable mesh. The most obvious method of combining such points is to create a small sphere at each iterate, and combine these overlapping spheres to create an object. While this is doable, it results in an object with an extremely large file size. In addition, overlapping objects

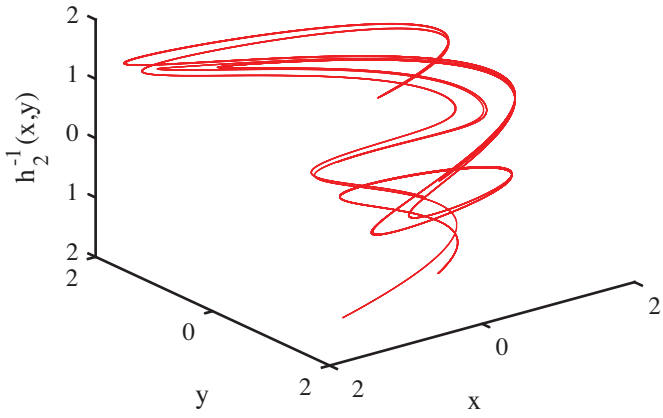

Figure 17. A delay embedded Hénon attractor in equation (2) created using a branch of the unstable manifold.

can cause mesh errors because it is not clear what to interpret as the boundary of the objects, and this is only exacerbated by the fractal nature of a strange attractor. These can often be fixed using a mesh repair program, but this is an extra and not always successful step. A more in-depth mathematical solution to the creation of an attractor of an iterated map is to use the structure of the chaotic attractors.

An attractor of a three-dimensional iterated map almost always falls into one of two categories, a multiply folded one-dimensional curve or a two-dimensional surface, described in [ASY97] as the *spaghetti-lasagna dichotomy*. In the spaghetti case, the attractor is typically the closure of a branch of a one-dimensional unstable manifold of a fixed point or periodic point of the map. The advantages of using this characterization is that the unstable manifold is a connected curve, and therefore there are no longer the same meshing issues as with small spheres. Figure 17 shows $(x, y, h_2^{-1}(x, y))$ for an attractor of the Hénon

$$h(x, y) = (1.4 - x^2 - 0.3y, x) \qquad (2)$$

created using a branch of the unstable manifold. The attractor lies in two dimensions, but we have included a preimage for a delay embedded version, making the graph three dimensional. The creation of attractors which arise as the closure of a curve is rather straightforward, though it involves more mathematical discussion than the differential equations case. However, the creation of a lasagna-type attractor surface is quite involved and better done using previously written software packages. For an idea of the difficulties involved, the creation of a crocheted Lorenz unstable manifold is described in [OK04].

For a two-dimensional dynamical system which varies with respect to a parameter, each attractor or other dynamical object will be contained in a plane. Therefore we can explore variation with respect to a parameter in the scope of a single 3D print. Namely, each dynamical object occurring at a fixed parameter occurs at a single slice, and together

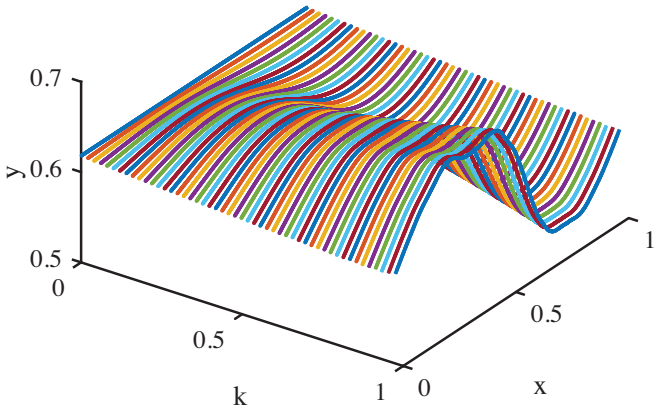

Figure 18. For the Chirikov standard map in equation (13), the quasiperiodic curve with golden mean rotation number varying with respect to parameter k.

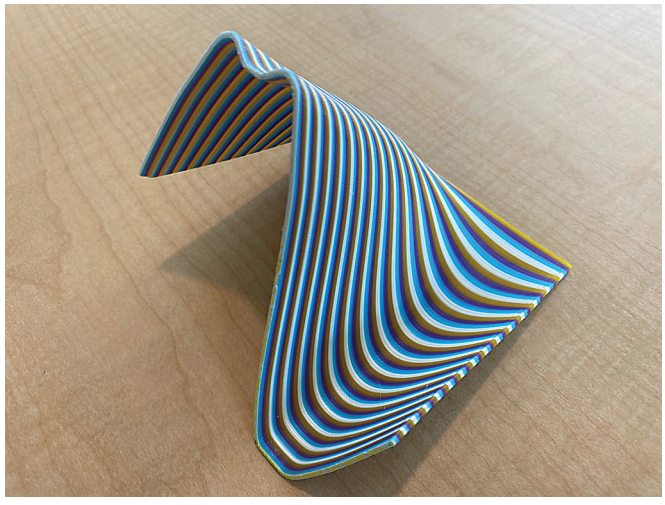

Figure 19. 3D printed model of the curve in Figure 18, in PLA plastic with colored stripes made possible from a Palette filament splicer.

the slices form a 3D object. For example, the Chirikov standard map in equation (13) in Appendix A is an area-preserving map on the unit square which varies with respect to a parameter k. At each fixed k value between $k = 0$ and $k \approx 0.971635406$, there is an invariant curve consisting of the closure of a single periodic orbit with the property that the rotation number (average of $f(x) - x$ along the orbit) is equal to the golden mean $(\sqrt{5} - 1)/2$. Rather than being chaotic, these invariant curves are *quasiperiodic*. Figure 18 shows how the curves vary with respect to k for $0 < k < 0.9716$. At each fixed k, we use a root finding method to find the curve with the given rotation number. Figure 19 is a photo of a 3D printed model of these curves.

We also note that iterated maps can be made higher dimensional by including point density information using a binning approach, where we count how often a map visits regions in some mesh. For example, Figure 20 shows the

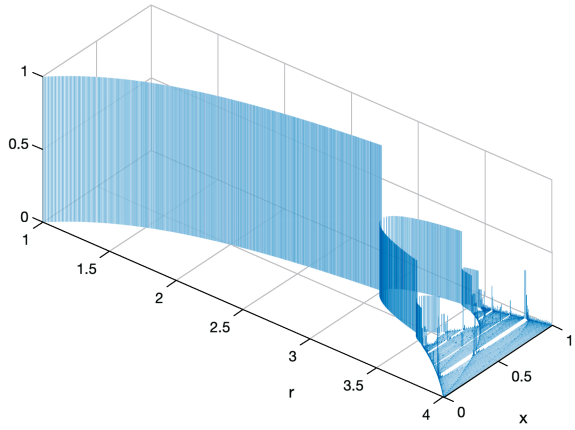

Figure 20. The logistic map in equation (3) where the height is the probability of visiting a particular location as r varies.

iterated logistic map

$$x_{n+1} = rx_n(1 - x_n), \tag{3}$$

where the height is a measure of how likely it is for the map to visit a particular region as we vary r.

Finally, parabolic partial differential equations can also be viewed as dynamical systems for solutions of the form $u(t, x)$, and at fixed t values, the solution u can have rich structure. These provide a rich set of examples for creation of 3D prints. An example of a 3D print of a spinodal decomposition in the Cahn-Hilliard equation is in [San15].

A. List of Dynamical Systems

We have tested the standard algorithm on the following examples, chosen both for beauty and for importance. Many of these examples can be found in [ASY97] and [Mei17].

- Lorenz attractor [Lor63]:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= x(\rho - z) - y, \\ \dot{z} &= xy - \beta z, \end{aligned} \tag{4}$$

  with $\sigma = 3$, $\beta = 1$, and $\rho = 28$, and initial conditions $x_0 = 0$, $y_0 = 1$, $z_0 = 0$. The Lorenz system is the first chaotic attractor within the scientific community. It is an atmospheric model created to understand unpredictability of linear models in weather prediction. See Chapter 9 of [ASY97] for a detailed discussion of the history of the model.
- Rössler attractor [Rös76]:

$$\begin{aligned} \dot{x} &= -y - z, \\ \dot{y} &= x + ay, \\ \dot{z} &= b + z(x - c), \end{aligned} \tag{5}$$

  with $a = 0.1$, $b = 0.1$, $c = 18$, and initial conditions $x_0 = 0$, $y_0 = 1$, $z_0 = 0$. This system was created to show that chaos could occur in systems that were even simpler than the Lorenz equations.

- Chen double scroll attractor [CU99]:

$$\begin{aligned} \dot{x} &= a(y-x), \\ \dot{y} &= (c-a)x - xz + cy, \\ \dot{z} &= xy - bz, \end{aligned} \tag{6}$$

with $a = 40$, $b = 3$, $c = 28$, and initial conditions $x_0 = -0.1$, $y_0 = 0.5$, $z_0 = -0.6$. This system was created to exhibit properties of both Lorenz and Rössler attractors.

- Arneodo attractor [ACST85]:

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= z, \\ \dot{z} &= -\alpha x - \beta y - z + \delta x^3, \end{aligned} \tag{7}$$

with $\alpha = -5.5$, $\beta = 3.5$, $\delta = -1$, and initial conditions $x_0 = 0.2$, $y_0 = 0.2$, $z_0 = -0.75$. This attractor was developed to illustrate chaos in a physical system near a triple instability.

- Langford attractor [Lan84]:

$$\begin{aligned} \dot{x} &= (z-b)x - dy, \\ \dot{y} &= dx + (z-b)y, \\ \dot{z} &= c + az - z^3/3 - (x^2+y^2)(1+ez) + fzx^3, \end{aligned} \tag{8}$$

with $a = 0.95$, $b = 0.7$, $c = 0.6$, $d = 3.5$, $e = 0.25$, $f = 0.1$, and initial conditions $x_0 = 0.1$, $y_0 = 1$, $z_0 = 0$. Note that while this attractor is commonly named after Yoji Aizawa, it cannot be found within his published work, and was in fact developed by Langford [Lan84]. He was investigating models of chaotic behavior moving upon a torus.

- Hénon-Heiles invariant torus [HH64]:

$$\begin{aligned} \dot{x} &= z, \\ \dot{y} &= w, \\ \dot{z} &= -x - 2xy, \\ \dot{w} &= -y - x^2 + y^2, \end{aligned} \tag{9}$$

with initial conditions $x_0 = 0$, $y_0 = 0$, $z_0 = 0.35$, $w_0 = -0.3$. The total time length is 200. This system models the motion of individual stars as affected by the rest of a galaxy. Unlike the other models in this paper, this is a four-dimensional example. We project and only plot (x, y, z). The system is Hamiltonian, so the chaotic solutions are not strange attractors and thus do not make very pretty prints. We have chosen an initial condition corresponding to a quasiperiodic solution (it has energy approximately 0.10625), which is a topological torus in four dimensions.

- Rucklidge attractor [Ruc92]:

$$\begin{aligned} \dot{x} &= \kappa x - \lambda y - yz, \\ \dot{y} &= x, \\ \dot{z} &= -z + y^2, \end{aligned} \tag{10}$$

with $\kappa = -2$, $\lambda = -6.7$, and initial conditions $x_0 = 1$, $y_0 = 0$, $z_0 = 4.5$. This system came about when modelling two-dimensional convection in a fluid layer rotating uniformly about a vertical axis.

- Anishchenko-Ashtakhov attractor [AA83]:

$$\begin{aligned} \dot{x} &= \mu x + y - xz, \\ \dot{y} &= -x, \\ \dot{z} &= -\eta z + \eta H(x) x^2, \end{aligned} \tag{11}$$

with $\mu = 1.2$, $\eta = 0.5$, $H(x)$ is the Heaviside function, and with initial conditions $x_0 = -0.1$, $y_0 = 0.5$, $z_0 = -0.6$. This system was proposed in the study of nonlinear oscillators. The original paper is in Russian, but a description appears in English in [AS98].

- Mackey-Glass attractor [MG77]:

$$\dot{x}(t) = \beta x(t-\tau)/(1 + x(t-\tau)^n) - \gamma x(t), \tag{12}$$

with $\gamma = 1$, $\beta = 2$, $\tau = 2$, and $n = 9.65$, and initial condition $x(t) = t^2$ for $-\tau < t < 0$. This equation models dynamical diseases including respiratory disorders such as irregular breathing apnea and hematologic disorders such as chronic myelogenous leukemia, in which blood cell counts oscillate rather than staying constant. Unlike other examples this produces a strange chaotic attractor for a delay differential equation. We plot the solution in three dimensions by projecting to $(x(t), x(t-\tau), \dot{x}(t-\tau))$. Both Mathematica and Matlab have built in delay equations solvers, making it possible to use a black box code for solving this equation.

- Chirikov standard map [Chi79]:

$$\begin{aligned} x_{t+1} &= x_t + y_{t+1} \bmod 1, \\ y_{t+1} &= y_t + -\frac{k}{2\pi} \sin(2\pi x_t). \end{aligned} \tag{13}$$

This iterated map is a well-known example in the study of area-preserving maps and Kolmogorov-Arnold-Moser theory. When $k = 0$, all points lie on "rotational" invariant circles, which are graphs of x as a function of y. For each fixed rotation number $\omega \in (0, 1)$, rotational invariant circles exist and vary smoothly for a parameter interval $(0, k_\omega)$. The largest value of k_ω occurs when ω is equal to the golden mean (or its inverse).

B. 3D Printing Notes

The physical models photographed in this paper were 3D printed using a variety of methods, which we outline here so that others can use these techniques to print their own models.

The 3D models shown in Figures 1, 3, 7, 8, and 9 were 3D printed with desktop filament-deposition (FDM) machines in PLA plastic. When printing with FDM, one layer of plastic is drawn out at a time, each supporting the next. If your model has overhangs (as will certainly be the case for these attractors), then you will also have to include extensive support material as part of your print. This support material can be removed after printing, but leaves marks and can damage or break the model during cleanup. For this reason models printed with FDM can have a rough surface and in addition need to have thicker path diameter for strength, so less detail is possible.

The model in Figure 6 was printed on a desktop resin 3D printer using stereolithography (SLA) technology. This method uses a laser to harden liquid resin one thin layer at a time, with the model developing upside-down while attached from a build plate that dips into a pool of resin. Models printed with SLA can in general be very delicate, but the sweeping curves of attractors do tend to break during support removal and cleanup, so a thick path diameter is recommended with this method as well. SLA printing also requires washing with isopropyl alcohol and curing with sunlight or a UV light. The final printed models have a very high-quality finish.

The models shown in Figures 2, 4, 5, 10, 15, and 21 were printed by the service bureau Shapeways in Nylon plastic, using Selective Laser Sintering (SLS). With this technology, the model is created by depositing very thin layers of powder which are solidified by a laser in the spots that intersect the design. At the end of printing the model is completely encased and supported by loose powder, so there are no supports to remove. Very thin and detailed models can be printed successfully with this method. Compare the detail in the SLS model of the Langford attractor shown in Figure 21 with the coarser models of about the same overall size printed in FDM (Figure 1) and SLA (Figure 6). SLS printing is a particularly good option for art/display-quality models and for those without their own 3D printers in house.

Finally, the model in Figure 19 was printed on an FDM machine with an additional Palette attachment that allowed for splicing filament colors together mid-print, resulting in a striped multicolor pattern that highlights the levels of the surface.

C. Code Resources

The authors were inspired by Segerman's seminal article [Seg12] that provided executable code that others could use to get started on related projects, and in the same spirit we include this Appendix. In this section we provide excerpts of fully executable, commented code for creating 3D-printable files with Mathematica, Matlab, and OpenSCAD. Full code files can be found at [LST20].

Figure 21. The Langford chaotic attractor (8). Modeled in Matlab and OpenSCAD and 3D printed in SLS nylon.

C.1. Mathematica. The following Mathematica code, available in the Mathematica `.nb` form from [LST20], can be used to create 3D-printable solutions to differential equations using the straightforward method, and is particularly adapted for chaotic attractors. This particular code creates the Chen double-scroll attractor [CU99], but can be easily adapted to create all the other solutions described here.

```
(* Set the parameters *)
a = 40; b = 3; c = 28;

(* Set the differential equation *)
F1[x_, y_, z_] := a (y - x)
F2[x_, y_, z_] := (c - a) x - x z + c y
F3[x_, y_, z_] := x  y - b z

(* Set the initial conditions *)
x0 = -0.1; y0 = 0.5; z0 = -0.6; ta = 2;

(* Solve, keeping the final value because
   we are removing transient behavior
   and waiting for the solution to
   converge to the attractor *)
ap = NDSolve[
  {xa'[t] == F1[xa[t], ya[t], za[t]],
   ya'[t] == F2[xa[t], ya[t], za[t]],
   za'[t] == F3[xa[t], ya[t], za[t]],
   xa[0]==x0 , ya[0]==y0, za[0]==z0},
```

```
  {xa, ya, za}, {t, 0, ta},
  MaxSteps -> Infinity]

(* Start a new computation at the place
   where the last solution ended. *)
x1 = xa[ta] /. ap;
y1 = ya[ta] /. ap;
z1 = za[ta] /. ap;
tb = 50;

(* Solve again, except this time
   starting at the place the solution
   ended the last time *)
bp = NDSolve[
  {xb'[t] == F1[xb[t], yb[t], zb[t]],
   yb'[t] == F2[xb[t], yb[t], zb[t]],
   zb'[t] == F3[xb[t], yb[t], zb[t]],
   xb[0]==x1, yb[0]==y1, zb[0]==z1},
  {xb, yb, zb}, {t, 0, tb},
  MaxSteps -> Infinity]

(* Plot the attractor for viewing
   on screen *)
ParametricPlot3D[
  {xb[t], yb[t], zb[t]} /. bp,
  {t, 0, tb}, PlotRange -> All]

(* Plot again in 3D printable form *)
cp = ParametricPlot3D[
  {xb[t], yb[t], zb[t]} /. bp,
  {t, 0, tb}, PlotStyle -> Tube[.5],
  PlotPoints -> 100,  PlotRange -> All]

(* Export a 3D printable STL file *)
Export["chenattractor.stl", cp]
```

C.2. Matlab. The following Matlab code can be used to produce data points that can be joined by tubes in OpenSCAD for 3D plots. See [LST20] for a Matlab `.m` format copy. Here we are implementing the Rucklidge attractor, but this can be easily adapted for any dynamical system.

The following function defines the Rucklidge attractor:

```
function ret=rucklidge(~,y)
% Function for the Rucklidge attractor
% 1: x'=-2x+6.7y-yz
% 2: y'=x
% 3: z'=-z+y^2
% 4: s'=sqrt(x'^2+y'^2+z'^2)
ret=zeros(4,1); % Column vector
ret(1)=-2*y(1)+6.7*y(2)-y(2)*y(3);
ret(2)=y(1);
ret(3)=-y(3)+y(2)^2;
ret(4)=sqrt(ret(1)^2+ret(2)^2+ret(3)^2);
```

The following function implements the secant method to find a given t such that $s(t) = b$:

```
function newt=secant(sol,s,p0)
% Secant method to find the time when the
% length of a curve is a given value
% Input: sol = ode solution, values can
%              be found at any time
%        s = required curve length
%        p0 = initial guess
N=500; % N is maximum number of steps
tol=1.0e-11; % tol is error bound
i=2; % i is function eval count
y=deval(sol,p0); %  y is sol at time p0
q0=y(4)-s; % first function value
p1=p0+1e-4; y=deval(sol,p1);
q1=y(4)-s; % second function value
while i<=N && abs(p0-p1)>tol
  % secant step and update
  p=p1-q1*(p1-p0)/(q1-q0);
  i=i+1; p0=p1; q0=q1; p1=p;
  y=deval(sol,p1); q1=y(4)-s;
end
if abs(p-p0)>tol, error('no conv'); end
newt=p;
```

The following code constructs the data points using the mixed curvature approach, called runrucklidge.m at [LST20]:

```
% Matlab code for plotting chaotic
% attractors. Output is data that is
% initially equally spaced points along the
% curve, but subdivide if the angle between
% line segments is not large enough
N=500; % N is maximum number of steps
NUM=1000; % NUM is the initial number of
% pieces of equal arc length
T=200; % T is the total time
init=[1,0,4.5]; % initial conditions
% solve the system, initial length=0
sol=ode45(@rucklidge,[0,T],[init,0]);
% y is the solution at a given time
y=deval(sol,T);
len=y(4); % len is the length of the curve
% Initialize data arrays, pos is position
% data, seg is segment length data
segnum=NUM; % current number of segments
% First point
pos=zeros(segnum+1,3); pos(1,:)=init;
% Last point
pos(segnum+1,:)=[y(1),y(2),y(3)];
% Location along arc of points
segment=(0:segnum)*len/segnum;
time=zeros(segnum+1,1); time(1)=0;
time(segnum+1)=T;
% time is time at each position
for i=1:segnum-1
   % Solve for the time for each new
   % constant length piece using the
   % secant method starting at the
   % previous time, and plus a bit
   time(i+1)=secant(sol,segment(i+1), ...
      time(i));
   y=deval(sol,time(i+1));
   pos(i+1,:)=[y(1),y(2),y(3)];
end
% Plot equally-spaced-in-time data points
figure(1)
plot3(pos(:,1),pos(:,2),pos(:,3),'.-')
axis square, axis equal
pause
done=false;
% Keep going until no segments need to be
% split
while ~done
   done=true;
```

```
   % Identify line segment vectors and
   % lengths
   vec=zeros(segnum,3);
   lenvec=zeros(segnum,1);
   for i=1:segnum
      vec(i,:)=pos(i+1,:)-pos(i,:);
      lenvec(i)=norm(vec(i,:));
   end
   % Identify angles between segments and
   % which ones to split
   ang=zeros(segnum-1,1);
   split=zeros(segnum,1);
   for i=1:segnum-1
      ang=acosd(sum(vec(i,:).* ...
         vec(i+1,:))/(lenvec(i)* ...
         lenvec(i+1)));
      if ang>10, split(i)=1;
         split(i+1)=1; done=false;
      end
   end
   disp(sum(split));
   % Split segments at half the distance
   % between endpoints
   newn=segnum+sum(split);
   newpos=zeros(newn+1,3);
   newpos(1,:)=init;
   newseg=zeros(newn+1,1);
   newseg(1)=0; newtime=zeros(newn+1,1);
   newtime(1)=0; newtime(newn+1)=T;
   j=1;
   for i=1:segnum
      if split(i)  % Add a new midpoint
         j=j+1;
         news=(segment(i)+segment(i+1))/2;
         newtime(j)=secant(sol,news, ...
            time(i)); y=deval(sol,newtime(j));
         newpos(j,:)=[y(1),y(2),y(3)];
         newseg(j)=news;
      end
      % Add endpoint
      j=j+1; newpos(j,:)=pos(i+1,:);
      newseg(j)=segment(i+1);
      newtime(j)=time(i+1);
   end
   % Update variable list
   segment=newseg; pos=newpos;
   time=newtime; segnum=newn;
   figure(1)
   plot3(pos(:,1),pos(:,2),pos(:,3),'.-')
   axis square, axis equal
   pause
end
% output data to file
f=fopen('rucklidge.txt','w');
for i=1:segnum+1
   fprintf(f,'[%f, %f, %f],\n',pos(i,:));
end
% Final plot of data
figure(1)
plot3(pos(:,1),pos(:,2),pos(:,3),'.')
axis square, axis equal
figure(2)
plot3(pos(:,1),pos(:,2),pos(:,3))
axis square, axis equal
```

C.3. OpenSCAD. The following OpenSCAD code takes lists of data points as inputs and constructs a 3D-printable tubular polyhedron following those data points. A self-contained version of the code is provided at [LST20].

```
// input files
use <list-comprehension/sweep.scad>
use <scad-utils/shapes.scad>

// choose start and end datapoints
step = 1;
start = 32;
end = 2740;

// style and size parameters
radius = 1;
sides = 8;
function shape() =
   circle(radius, $fn=sides);

// overall scale applied before the
   tubular polyhedron is constructed
scale = 12;

// large list of datapoints
   obtained from Matlab solver
points = [
[2.000000, 3.000000, 1.000000],
[2.147146, 2.904262, 1.173744],
[2.268546, 2.805048, 1.364594],
[2.362653, 2.702993, 1.568898],
...
(thousands of data points omitted)
...
[0.864680, 1.798989, 3.940367],
[0.792858, 1.667819, 3.743417],
[0.747632, 1.539176, 3.538058],
[0.700565, 1.276545, 3.119037]
];

// construct tubular polyhedron
   following curve of datapoints
path =
   [ for (i=[start:step:end])
      scale * points[i] ];
path_transforms =
   construct_transform_path(path,true);
sweep(shape(), path_transforms,false);
```

References

[ASY97] Kathleen T. Alligood, Tim D. Sauer, and James A. Yorke, *Chaos: An introduction to dynamical systems*, Textbooks in Mathematical Sciences, Springer-Verlag, New York, 1997. MR1418166

[AA83] V. S. Anishchenko and V. V. Astakhov, *Experimental study of the mechanism of the appearance and the structure of a strange attractor in an oscillator with inertial nonlinearity*, Radiotekhnika i Elektronika **28** (1983), 1109–1115. In Russian.

[AS98] V. S. Anishchenko and G. Strelkova, *Irregular attractors*, Discrete Dynamics in Nature and Society **2** (1998), no. 1, 53–72.
[ACST85] A. Arneodo, P. H. Coullet, E. A. Spiegel, and C. Tresser, *Asymptotic chaos*, Phys. D **14** (1985), no. 3, 327–347, DOI 10.1016/0167-2789(85)90093-4. MR793709
[Atm] Chaotic Atmospheres, *Math:rules, strange attractors*. https://chaoticatmospheres.com/mathrules-strange-attractors, accessed February 2020.
[CU99] Guanrong Chen and Tetsushi Ueta, *Yet another chaotic attractor*, Internat. J. Bifur. Chaos Appl. Sci. Engrg. **9** (1999), no. 7, 1465–1466, DOI 10.1142/S0218127499001024. MR1729683
[Chi79] Boris V. Chirikov, *A universal instability of many-dimensional oscillator systems*, Phys. Rep. **52** (1979), no. 5, 264–379, DOI 10.1016/0370-1573(79)90023-1. MR536429
[Gag18] Michael S. Gagliardo, *3d printing chaos*, Bridges 2018 Conference Proceedings (2018), 491–494. http://archive.bridgesmathart.org/2018/bridges2018-491.pdf.
[GS90] Peter Gray and Stephen K. Scott, *Chemical oscillations and instabilities*, International Series of Monographs on Chemistry, vol. 21, Oxford University Press, Oxford, 1990.
[HH64] Michel Hénon and Carl Heiles, *The applicability of the third integral of motion: Some numerical experiments*, Astronom. J. **69** (1964), 73–79, DOI 10.1086/109234. MR158746
[Kin] Marius Kintel, *OpenSCAD*: *The programmers solid 3D CAD modeller*.
[Lan84] W. F. Langford, *Numerical studies of torus bifurcations*, Numerical methods for bifurcation problems (Dortmund, 1983), Internat. Schriftenreihe Numer. Math., vol. 70, Birkhäuser, Basel, 1984, pp. 285–295. MR821035
[Lor63] Edward N. Lorenz, *Deterministic nonperiodic flow*, J. Atmospheric Sci. **20** (1963), no. 2, 130–141, DOI 10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2. MR4021434
[LST20] Stephen K. Lucas, Evelyn Sander, and Laura Taalman, *Supplementary materials for modeling dynamical systems for 3d printing* (2020). http://math.gmu.edu/~sander/EvelynSite/supplementary-materials-for.html.
[MG77] M. C. Mackey and L. Glass, *Oscillation and chaos in physiological control systems*, Science **197** (1977), no. 4300, 287–289.
[Mei17] James D. Meiss, *Differential dynamical systems*, Revised edition, Mathematical Modeling and Computation, vol. 22, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2017. MR3614477
[OK04] Hinke M. Osinga and Bernd Krauskopf, *Crocheting the Lorenz manifold*, Math. Intelligencer **26** (2004), no. 4, 25–37, DOI 10.1007/BF02985416. MR2104464
[Rös76] O. E. Rössler, *An equation for continuous chaos*, Phys. Lett. A **57** (1976), 397–398.
[Ruc92] A. M. Rucklidge, *Chaos in models of double convection*, J. Fluid Mech. **237** (1992), 209–229, DOI 10.1017/S0022112092003392. MR1161996
[Rue06] David Ruelle, *What is ... a strange attractor?*, Notices Amer. Math. Soc. **53** (2006), no. 7, 764–765. MR2255038
[San15] Evelyn Sander, *Spinodal decomposition*, GMU Math Makerlab Blog (2015). http://gmumathmaker.blogspot.com/2015/01/spinodal-decomposition.html.
[Seg12] Henry Segerman, *3d printing for mathematical visualization*, Math. Intell. **34** (2012), no. 4, 56–62. https://math.okstate.edu/people/segerman/papers/3d_printed_visualisation.pdf.

ACKNOWLEDGMENTS. We thank the referees for their helpful comments, which improved the quality of this paper.

Thank you to Eric Stauffer, JMU Libraries, for processing and printing the filament-spliced model in Figure 19, and to Patrick Bishop, GMU, who printed the colorful models in Figures 1, 3, 7, and 9. Thanks also to Edmund Harriss, who took beautiful photographs of some of our models.

This material is based upon work supported by the National Science Foundation under Grant No. DMS-1439786 and the Simons Foundation Institute Grant Award ID 507536 while LT & ES were in residence at the Institute for Computational and Experimental Research in Mathematics in Providence, RI, during the Fall 2019 semester. ES's work was supported by a grant from the Simons Foundation/SFARI (636383, ES).


Stephen K. Lucas


Evelyn Sander


Laura Taalman

Credits

Opening image and photo of Laura Taalman are courtesy of Laura Taalman.

Figures 1, 6–9, and 11–20 are courtesy of the authors.

Figures 2, 4, 5, 10, and 21 are courtesy of Edmund O. Harriss.

Photo of Stephen K. Lucas is courtesy of Amira Lucas.

Photo of Evelyn Sander is courtesy of Arnold Feldman.