Undergraduate Lecture Notes in Physics

George Datseris
Ulrich Parlitz

Nonlinear Dynamics

A Concise Introduction Interlaced with Code

Springer

Contents

1 Dynamical Systems 1
1.1 What Is a Dynamical System? 1
1.1.1 Some Example Dynamical Systems 2
1.1.2 Trajectories, Flows, Uniqueness and Invariance 4
1.1.3 Notation 5
1.1.4 Nonlinearity 6
1.2 Poor Man's Definition of Deterministic Chaos 6
1.3 Computer Code for Nonlinear Dynamics 7
1.3.1 Associated Repository: Tutorials, Exercise Data, Apps 8
1.3.2 A Notorious Trap 9
1.4 The Jacobian, Linearized Dynamics and Stability 9
1.5 Dissipation, Attractors, and Conservative Systems 12
1.5.1 Conserved Quantities 14
1.6 Poincaré Surface of Section and Poincaré Map 14

2 Non-chaotic Continuous Dynamics 21
2.1 Continuous Dynamics in 1D 21
2.1.1 A Simple Model for Earth's Energy Balance 21
2.1.2 Preparing the Equations 22
2.1.3 Graphical Inspection of 1D Systems 23
2.2 Continuous Dynamics in 2D 24
2.2.1 Fixed Points in 2D 24
2.2.2 Self-sustained Oscillations, Limit Cycles and Phases 24
2.2.3 Finding a Stable Limit Cycle 26
2.2.4 Nullclines and Excitable Systems 26
2.3 Poincaré-Bendixon Theorem 29
2.4 Quasiperiodic Motion 30

3 Defining and Measuring Chaos 37
3.1 Sensitive Dependence on Initial Conditions 37
3.1.1 Largest Lyapunov Exponent 38
3.1.2 Predictability Horizon 39
3.2 Fate of State Space Volumes 40
3.2.1 Evolution of an Infinitesimal Uncertainty Volume 40
3.2.2 Lyapunov Spectrum 41
3.2.3 Properties of the Lyapunov Exponents 43
3.2.4 Essence of Chaos: Stretching and Folding 44
3.2.5 Distinguishing Chaotic and Regular Evolution 45
3.3 Localizing Initial Conditions Using Chaos 46

4 Bifurcations and Routes to Chaos 53
4.1 Bifurcations 53
4.1.1 Hysteresis 54
4.1.2 Local Bifurcations in Continuous Dynamics 55
4.1.3 Local Bifurcations in Discrete Dynamics 56
4.1.4 Global Bifurcations 57
4.2 Numerically Identifying Bifurcations 58
4.2.1 Orbit Diagrams 58
4.2.2 Bifurcation Diagrams 61
4.2.3 Continuation of Bifurcation Curves 61
4.3 Some Universal Routes to Chaos 63
4.3.1 Period Doubling 63
4.3.2 Intermittency 65

5 Entropy and Fractal Dimension 71
5.1 Information and Entropy 71
5.1.1 Information Is Amount of Surprise 71
5.1.2 Formal Definition of Information and Entropy 72
5.1.3 Generalized Entropy 73
5.2 Entropy in the Context of Dynamical Systems 73
5.2.1 Amplitude Binning (Histogram) 73
5.2.2 Nearest Neighbor Kernel Estimation 75
5.3 Fractal Sets in the State Space 75
5.3.1 Fractals and Fractal Dimension 76
5.3.2 Chaotic Attractors and Self-similarity 77
5.3.3 Fractal Basin Boundaries 78
5.4 Estimating the Fractal Dimension 79
5.4.1 Why Care About the Fractal Dimension? 81
5.4.2 Practical Remarks on Estimating the Dimension 82
5.4.3 Impact of Noise 82
5.4.4 Lyapunov (Kaplan–Yorke) Dimension 83

6 Delay Coordinates 89
6.1 Getting More Out of a Timeseries 89
6.1.1 Delay Coordinates Embedding 90
6.1.2 Theory of State Space Reconstruction 91
6.2 Finding Optimal Delay Reconstruction Parameters 92
6.2.1 Choosing the Delay Time 93
6.2.2 Choosing the Embedding Dimension 94
6.3 Advanced Delay Embedding Techniques 96
6.3.1 Spike Trains and Other Event-Like Timeseries 96
6.3.2 Generalized Delay Embedding 96
6.3.3 Unified Optimal Embedding 97
6.4 Some Nonlinear Timeseries Analysis Methods 98
6.4.1 Nearest Neighbor Predictions (Forecasting) 98
6.4.2 Largest Lyapunov Exponent from a Sampled Trajectory 99
6.4.3 Permutation Entropy 99

7 Information Across Timeseries 105
7.1 Mutual Information 105
7.2 Transfer Entropy 107
7.2.1 Practically Computing the Transfer Entropy 108
7.2.2 Excluding Common Driver 109
7.3 Dynamic Influence and Causality 110
7.3.1 Convergent Cross Mapping 111
7.4 Surrogate Timeseries 113
7.4.1 A Surrogate Example 114

8 Billiards, Conservative Systems and Ergodicity 121
8.1 Dynamical Billiards 121
8.1.1 Boundary Map 122
8.1.2 Mean Collision Time 123
8.1.3 The Circle Billiard (Circle Map) 124
8.2 Chaotic Conservative Systems 124
8.2.1 Chaotic Billiards 124
8.2.2 Mixed State Space 125
8.2.3 Conservative Route to Chaos: The Condensed Version 126
8.2.4 Chaotic Scattering 127
8.3 Ergodicity and Invariant Density 127
8.3.1 Some Practical Comments on Ergodicity 129
8.4 Recurrences 130
8.4.1 Poincaré Recurrence Theorem 130
8.4.2 Kac's Lemma 130
8.4.3 Recurrence Quantification Analysis 131

9 Periodically Forced Oscillators and Synchronization 137
9.1 Periodically Driven Passive Oscillators 137
9.1.1 Stroboscopic Maps and Orbit Diagrams 138
9.2 Synchronization of Periodic Oscillations 140
9.2.1 Periodically Driven Self-Sustained Oscillators 140
9.2.2 The Adler Equation for Phase Differences 144
9.2.3 Coupled Phase Oscillators 145
9.3 Synchronization of Chaotic Systems 146
9.3.1 Chaotic Phase Synchronization 146
9.3.2 Generalized Synchronization of Uni-Directionally Coupled Systems 149

10 Dynamics on Networks, Power Grids, and Epidemics 157
10.1 Networks 157
10.1.1 Basics of Networks and Graph Theory 157
10.1.2 Typical Network Architectures 158
10.1.3 Robustness of Networks 160
10.2 Synchronization in Networks of Oscillators 161
10.2.1 Networks of Identical Oscillators 161
10.2.2 Chimera States 163
10.2.3 Power Grids 164
10.3 Epidemics on Networks 166
10.3.1 Compartmental Models for Well-Mixed Populations 166
10.3.2 Agent Based Modelling of an Epidemic on a Network 168

11 Pattern Formation and Spatiotemporal Chaos 175
11.1 Spatiotemporal Systems and Pattern Formation 175
11.1.1 Reaction Diffusion Systems 175
11.1.2 Linear Stability Analysis of Spatiotemporal Systems 177
11.1.3 Pattern Formation in the Brusselator 179
11.1.4 Numerical Solution of PDEs Using Finite Differences 180
11.2 Excitable Media and Spiral Waves 181
11.2.1 The Spatiotemporal Fitzhugh-Nagumo Model 182
11.2.2 Phase Singularities and Contour Lines 182
11.3 Spatiotemporal Chaos 185
11.3.1 Extensive Chaos and Fractal Dimension 185
11.3.2 Numerical Solution of PDEs in Spectral Space 186
11.3.3 Chaotic Spiral Waves and Cardiac Arrhythmias 188

12 Nonlinear Dynamics in Weather and Climate 193
12.1 Complex Systems, Chaos and Prediction 193
12.2 Tipping Points in Dynamical Systems 196
12.2.1 Tipping Mechanisms 196
12.2.2 Basin Stability and Resilience 199
12.2.3 Tipping Probabilities 201
12.3 Nonlinear Dynamics Applications in Climate 202
12.3.1 Excitable Carbon Cycle and Extinction Events 202
12.3.2 Climate Attractors 203
12.3.3 Glaciation Cycles as a Driven Oscillator Problem 204

Appendix A: Computing Lyapunov Exponents 211

Appendix B: Deriving the Master Stability Function 217

References 219

Index 233

Chapter 1
Dynamical Systems

Abstract This introductory chapter defines dynamical systems, stresses their wide spread applications, and introduces the concept of "deterministic chaos". Since computer simulations are valuable and even necessary for studying nonlinear dynamics we also show in this chapter examples of runnable code snippets that will be used throughout the book. We then look at fixed points, and when they are stable. To do so we employ linear stability analysis and subsequently discuss how volumes in the state space grow or shrink. We close by reviewing the Poincaré surface of section, a technique to convert a continuous system into discrete one, useful for visualizing higher-dimensional dynamics.

1.1 What Is a Dynamical System?

The term "dynamical system" can describe a wide range of processes and can be applied to seemingly all areas of science. Simply put, a dynamical system is a set of variables, or quantities, whose values change with time according to some predefined rules. These rules can have stochastic components, but in this book we will be considering systems without random parts, i.e., *deterministic dynamical systems*.

The variables that define the system are formally called the *state variables* of the system, and constitute the *state* or *state vector* $\mathbf{x} = (x_1, x_2, \ldots, x_D)$. For example, state variables can be the positions and velocities of planets moving in a gravitational field, the electric current and voltage of an electronic circuit, or temperature and humidity of a weather model, to name a few. The space that $\mathbf{x}$ occupies is called the *state space* or *phase space* $\mathcal{S}$ and has dimension[1] D, with D the amount of variables that compose $\mathbf{x}$.

[1] Typically the state space $\mathcal{S}$ is an Euclidean space $\mathbb{R}^D$. But in general, it can be any arbitrary D-dimensional manifold. For example if some variables are by nature periodic, like the angle of a pendulum, the state space becomes toroidal.

G. Datseris and U. Parlitz, *Nonlinear Dynamics*, Undergraduate Lecture Notes in Physics,
https://doi.org/10.1007/978-3-030-91032-7_1

Dynamical systems can be classified according to the nature of time. In this book we will be mainly considering two classes of dynamical systems, both having a continuous state space. This means that each component of $\mathbf{x}$ is a real number.[2] The first class is called *discrete* dynamical system

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n) \tag{1.1}$$

with f being the *dynamic rule* or *equations of motion* specifying the temporal evolution of the system. Here time is a discrete quantity, like steps, iterations, generations or other similar concepts. At each time step f takes the current state and maps it to the next state: $x_1 = f(x_0)$, $x_2 = f(x_1)$, etc. Discrete systems are usually given by iterated maps as in (1.1), where the state $\mathbf{x}_n$ is plugged into f to yield the state at the next step, $\mathbf{x}_{n+1}$. In our discussion here and throughout this book all elements of the vector-valued function f are real. Any scenario where f inputs and outputs complex values can be simply split into more variables containing the real and imaginary parts, respectively.

In *continuous* dynamical systems

$$\dot{\mathbf{x}} \equiv \frac{d\mathbf{x}}{dt} = f(\mathbf{x}) \tag{1.2}$$

time is a continuous quantity and f takes the current state and returns the rate of change of the state. Continuous systems are defined by a set of coupled ordinary differential equations (ODEs). f is also sometimes called *vector field* in this context. Equation (1.2) is a set of *autonomous* ordinary differential equations, i.e., f does not explicitly depend on time t. This is the standard form of continuous dynamical systems. In general, non-autonomous systems can be written as autonomous systems with an additional state variable corresponding to time (see Chap. 9). Furthermore, (1.2) contains only first order time derivatives. All expressions with higher order derivatives (e.g., $\ddot{x} = ...$, obtained in Newtonian mechanics) can be re-written as first-order systems by introducing new variables as derivatives, e.g., $y = \dot{x}$. Notice the fundamental difference between continuous and discrete systems: in the latter f provides the direct change, while in the former f provides the rate of change.

1.1.1 Some Example Dynamical Systems

To put the abstract definition of dynamical systems into context, and also motivate them as tools for studying real world scenarios, let's look at some of the dynamical

[2] In Chap. 11, we will look at spatiotemporal systems, where each variable is a spatial field that is evolved in time.

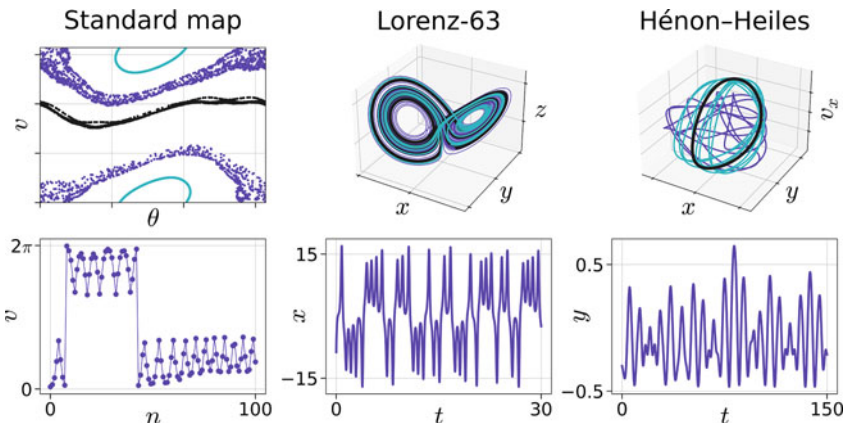


Fig. 1.1 Top row: state space of example systems (for Hénon-Heiles only 3 of the 4 dimensions are shown). Three different initial conditions (different colors) are evolved and populate the state space. Bottom row: example timeseries of one of the variables of each system (only one initial condition is used). Each column is a different system, evolved using Code 1.1. Animations and interactive applications for such plots can be found online at animations/1/trajectory_evolution.

systems that we will be using in the subsequent chapters of this book. We visualize some of them in Fig. 1.1.

The simplest dynamical system, and arguably the most famous one, is the logistic map, which is a discrete system of only one variable defined as

$$x_{n+1} = r x_n (1 - x_n), \quad r \in [0, 4], \quad x \in [0, 1]. \tag{1.3}$$

This dynamical equation has been used in different contexts as an example or a prototype of a simple system that displays very rich dynamics nevertheless, with a prominent role in population dynamics. There it describes how a (normalized) population x of, let's say, rabbits changes from generation to generation. At each new generation n the number of rabbits increases because of rx, with r a growth factor (which is of course the available amount of carrots). But there is a catch; if the rabbit population becomes too large, there aren't enough carrots to feed all of them! This is what the factor $(1 - x)$ represents. The higher the population x_n at generation n, the more penalty to the growth rate.

Another well-studied prototypical two dimensional discrete system is the standard map, that is given by

$$\begin{aligned} \theta_{n+1} &= \theta_n + v_n + k \sin(\theta_n) \\ v_{n+1} &= v_n + k \sin(\theta_n) \end{aligned} \tag{1.4}$$

with θ the angle and v the velocity of an oscillating pendulum on a tabletop (gravity-free). The pendulum is periodically kicked at every time unit by a force of strength k ($k = 1$ unless noted otherwise), which has a constant direction (thus the term sin). You may wonder why this is a discrete system when clearly an oscillating pendulum

is operating in continuous time. The reason is simple: since the kick is applied only at each step n, and the motion between kicks is free motion, we only need to record the momentum and angle values at the time of the kick.

Yet another famous dynamical system, continuous in this case, is the Lorenz-63 system, given by

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= -xz + \rho x - y \\ \dot{z} &= xy - \beta z. \end{aligned} \tag{1.5}$$

It is a simplified model for atmospheric convection, representing a two-dimensional fluid layer. x is the rate of convection and y and z are the temperature variations in the horizontal and vertical direction of the fluid, respectively. The parameters ρ, β, σ are related to the properties of the fluid like its viscosity (we use $\rho = 28$, $\beta = 8/3$, $\sigma = 10$ unless noted otherwise).

The last system that we list here is the Hénon-Heiles system

$$\begin{aligned} \dot{x} &= v_x, \quad & \dot{v}_x &= -x - 2xy \\ \dot{y} &= v_y, \quad & \dot{v}_y &= -y - (x^2 - y^2). \end{aligned} \tag{1.6}$$

This is a simplification of the motion of a star (represented by a point particle with positions x, y and velocities v_x, v_y) around a galactic center (positioned at $(0, 0)$). Although we will discuss how to simulate dynamical systems in detail in Sect. 1.3, let's see how these systems "look like" when evolved in time. In Fig. 1.1 we show both the state space of some systems (populated by evolving three different initial conditions, each with a different color) as well as an example timeseries of one of the variables.

It may seem that these examples were handpicked to be as diverse as possible. This is not at all the case and we were honest with you in the introduction. The framework of dynamical systems applies in seemingly arbitrary parts of reality.

1.1.2 Trajectories, Flows, Uniqueness and Invariance

A *trajectory* (commonly also called an *orbit*) represents the evolution from an initial condition $\mathbf{x}_0 = \mathbf{x}(t = 0)$ in the state space. *Fixed points*, defined by $f(\mathbf{x}^*) = \mathbf{x}^*$ (discrete dynamics) or $f(\mathbf{x}^*) = 0$ (continuous dynamics), are technically trajectories that consist of a single point, i.e., a state that never changes in time. Another special kind of trajectories are periodic ones, which for discrete systems are a finite number of N points satisfying $\mathbf{x}_{n+N} = \mathbf{x}_n$, and for continuous systems are closed curves in the state space satisfying $\mathbf{x}(t + T) = \mathbf{x}(t)$. Putting these two cases aside, for discrete systems a trajectory consists of an infinite, but ordered, number of points and for continuous systems trajectories are curves in the state space that do not close.

In most cases (and for all systems we will consider in this book), the Picard-Lindelöf theorem states that for each initial condition $\mathbf{x}_0$ a unique solution $\mathbf{x}(t)$

of the ODE(s) (1.2) exists for $t \in (-\infty, \infty)$ if the vector field f fulfills a mild smoothness condition, called Lipschitz continuity[3] which is met by most systems of interest. Only in cases where the solution diverges to infinity in finite time (that will not be considered in this book) $\|\mathbf{x}(t \to T_\pm)\| \to \infty$, the existence limits are finite, $t \in (T_-, T_+)$. The uniqueness property implies that any trajectory $\{\mathbf{x}(t) \in \mathcal{S} : -\infty < t < \infty\}$ never intersects itself (except for fixed points or periodic orbits) or trajectories resulting from other initial conditions, i.e., the time evolution is unique.

For discrete systems forward-time uniqueness is always guaranteed for any f, regardless of continuity, simply from the definition of what a function is. To have the same feature backward in time f has to be invertible, in contrast to continuous systems where this is not required.

In simulations (or with measured data) trajectories of continuous systems are sampled at discrete time points which creates a set of states $A = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N\}$. If only a single component of the state vectors $\mathbf{x}_n$ or a scalar function of the states is considered this provides a *uni-variate timeseries* or just *timeseries*. Measuring several observables simultaneously yields a *multivariate timeseries*.

Another concept which is useful for describing the temporal evolution is the *flow* $\Phi^t(\mathbf{x})$, a function mapping any state $\mathbf{x}$ to its future ($t > 0$) or past ($t < 0$) image $\mathbf{x}(t) = \Phi^t(\mathbf{x})$ (here t could also be n, i.e., the same concept holds for both continuous or discrete systems). Note that $\Phi^0(\mathbf{x}) = \mathbf{x}$ and $\Phi^{t+s}(\mathbf{x}) = \Phi^t(\Phi^s(\mathbf{x})) = \Phi^s(\Phi^t(\mathbf{x}))$. An *invariant set* is by definition a set A which satisfies $\Phi^t(A) = A,\ \forall t$. This means that the dynamics maps the set A to itself, i.e., it is "invariant" under the flow. Invariant sets play a crucial role in the theoretical foundation of dynamical systems.

1.1.3 Notation

In the remainder of this book we will typically use capital letters A, X, Y, Z, Ω to denote sets (sampled trajectories are ordered sets) and matrices, bold lowercase upright letters $\mathbf{a}, \mathbf{x}, \mathbf{y}, \mathbf{z}, \boldsymbol{\omega}$ to notate vectors, and lowercase letters a, x, y, z, ω to notate timeseries, variables or constants. A few exceptions exist, but we believe there will be no ambiguities due to the context. The symbols f, D, p, P are used exclusively to represent: the dynamic rule (also called equations of motion or vector field), the dimensionality of the state space or set at hand, an unnamed parameter of the system, and a probability, respectively. Accessing sets or timeseries at a specific *index* is done either with subscripts x_n or with brackets $x[n]$ (n is always integer here). The syntax $x(t)$ means "the value of x at time t" in the case where t is continuous time, or it means that x is a function of t. The symbol $\|\mathbf{x}\|$ means the norm of $\mathbf{x}$, which depends on the metric of $\mathcal{S}$, but most often is the Euclidean norm (or metric). Set elements are enclosed in brackets {}.

[3] A function $f : \mathcal{S} \to \mathcal{S}$ is called Lipschitz continuous if there exists a real constant $k \geq 0$ so that $\|f(\mathbf{x}) - f(\mathbf{y})\| \leq k\|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{S}$. Every continuously differentiable function is Lipschitz continuous.

1.1.4 Nonlinearity

A linear equation (or system) f by definition satisfies the superposition principle $f(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y})$, $\alpha, \beta \in \mathbb{R}$ and thus any two valid solutions can be added to generate a new solution. Furthermore, the norm of the state $\mathbf{x}$ is irrelevant, as a solution can be given for any scale by changing α, and thus only the orientation of $\mathbf{x}$ matters. Because of these two properties linear dynamical systems can be solved in closed form and do not exhibit any complex dynamics.

You may have noticed that the rules f that we have described in Sect. 1.1.1 are *nonlinear* functions of the state $\mathbf{x}$. In contrast to linear systems, nonlinear systems must be *considered as a whole* instead of being able to be split into smaller and simpler parts, and on a *scale-by-scale basis*, as different scales are dominated by different dynamics within the same system. Therefore, nonlinear systems display a plethora of amazing features and dominate in almost all natural processes around us.

1.2 Poor Man's Definition of Deterministic Chaos

The systems we are considering in this book are nonlinear and *deterministic*. Given an initial condition and the rule f we are in theory able to tell everything about the future of the system by evolving it forward in time. But one must not confuse deterministic systems with simple or easy-to-predict behaviour. While sometimes indeed deterministic dynamics will lead to simple, periodic behavior, this is not always the case as illustrated in the bottom row of Fig. 1.1.

Interestingly, the timeseries shown *never repeat themselves* in a periodic manner,[4] even though the systems are deterministic! On the other hand, the timeseries aren't random either, as they seem to be composed of easily distinguishable patterns. For example, in the case of the standard map, one sees that sequences of points form triangles, and they are either near 2π pointing down or near 0 pointing up. Similar patterns exist in the Lorenz-63 and Hénon-Heiles model, where there are oscillations present, but they are non-periodic. It feels intuitive (and is also precisely true) that these patterns have a dynamic origin in the rule of the system f. Their exact sequence (i.e., when will the triangles in the standard map switch from up to down) may appear random, however, and sometimes may indeed be statistically equivalent to random processes.

This is called *deterministic chaos*: when the evolution of a system in time is non-periodic, apparently irregular, difficult to predict, but still deterministic and full of patterns. These patterns result in *structure in the state space*, as is seen clearly in the top row of Fig. 1.1. The sets that are the result of evolving a chaotic system (or more precisely, an initial condition that leads to chaotic dynamics in that system) will be called *chaotic sets* in this book. The fact that these sets have structure in the state space is the central property that distinguishes deterministic chaos from pure randomness: even if the sequence (timeseries) appears random, in the state space

[4] This might not be obvious from the small time window shown in the plot, so for now you'll have to trust us or simulate the systems yourself based on the tools provided in Sect. 1.3.1.

there is structure (of various forms of complexity). We will discuss deterministic chaos in more detail in Chap. 3.

1.3 Computer Code for Nonlinear Dynamics

As nonlinear dynamical systems are rarely treatable fully analytically, one always needs a computer to study them in detail or to analyze measured timeseries. Just a computer though is not enough, but one also needs code to run! This is the reason why we decided to write this book so that its pages are interlaced with real, runnable computer code. Having a "ready-to-go" piece of code also enables instant experimentation on the side of the reader, which we believe to be of utmost importance.

To write real code, we chose the Julia programming language because we believe it is highly suitable language for scientific code and even more so for the context of this book. Its simple syntax allows us to write code that corresponds line-by-line to the algorithms that we describe in text. Furthermore, Julia contains an entire software organization for dynamical systems, JuliaDynamics whose software we use throughout this book. The main software we'll be using is DynamicalSystems.jl, a software library of algorithms for dynamical systems and nonlinear dynamics, and InteractiveDynamics.jl, which provides interactive graphical applications suitable for the classroom. To demonstrate, let's introduce an example code snippet in Code 1.1, similar to the code that we will be showing in the rest of the book. We think that the code is intuitive even for readers unfamiliar with Julia.

Code 1.1 Example code defining the Lorenz-63 system, Eq. (1.5) in Julia, and obtaining a trajectory for it using DynamicalSystems.jl.

```
using DynamicalSystems # load the library

function lorenz_rule(u, p, t) # the dynamics as a function
    σ, ρ, β = p
    x, y, z = u
    dx = σ*(y - x)
    dy = x*(ρ - z) - y
    dz = x*y - β*z
    return SVector(dx, dy, dz) # Static Vector
end

p  = [10.0, 28.0, 8/3] # parameters: σ, ρ, β
u₀ = [0.0, 10.0, 0.0]  # initial condition
# create an instance of a `DynamicalSystem`
lorenz = ContinuousDynamicalSystem(lorenz_rule, u₀, p)

T  = 100.0 # total time
Δt = 0.01  # sampling time
A  = trajectory(lorenz, T; Δt)
```

Code is presented in mono-spaced font with a light purple background, e.g., `example`. Code 1.1 is an important example, because it shows how one defines a "dynamical system" in Julia using DynamicalSystems.jl.[5] Once such a dynamical system is defined in code, several things become possible through DynamicalSystems.jl. For example, in Code 1.1 we use the function `trajectory` to evolve the initial condition of a `DynamicalSystem` in time, by solving the ODEs through DifferentialEquations.jl.

1.3.1 Associated Repository: Tutorials, Exercise Data, Apps

Julia, DynamicalSystems.jl, and in fact anything else we will use in the code snippets are very well documented online. As this is a textbook about nonlinear dynamics, and not programming, we will not be teaching Julia here. Besides, even though we use Julia here, everything we will be presenting in the code snippets could in principle be written in other languages as well.

Notice however that Julia has an in-built help system and thus most snippets can be understood without consulting online documentations. Simply put, in the Julia console you can type question mark and then the name of the function that you want to know more about, e.g., `?trajectory`. Julia will then display the documentation of this function, which contains detailed information about exactly what the function does and how you can use it. Another reason not to explain code line-by-line here is because programming languages and packages get updated much, much faster than books do, which runs the risk of the explanations in a book becoming obsolete.

There is an online repository associated with this book, https://github.com/JuliaDynamics/NonlinearDynamicsTextbook. There we have collected a number of tutorials and workshops that teach core Julia as well as major packages. This repository also contains all code snippets and all code that produces the figures that make up this book, and there we can update code more regularly than in the book. It also contains the exact package versions used to create the figures, establishing reproducibility. The same repository contains the datasets that are used in the exercises of every chapter, as well as multiple choice questions that can be used during lectures (e.g., via an online polling service), but also in exams. Perhaps the most important thing in this repository are scripts that launch interactive, GUI-based applications (apps) that elucidate core concepts. These are present in the folder animations and are linked throughout the book. For convenience, a recorded video is also provided for each app.

[5] In the following chapters we will not be defining any new dynamical systems in code, but rather use predefined ones from the `Systems` submodule.

1.3.2 A Notorious Trap

Almost every algorithm that we will be presenting in this book has a high quality performant implementation in DynamicalSystems.jl. In addition, we strongly believe that code should always be shared and embraced in scientific work in nonlinear dynamics. New papers should be published with the code used to create them, establishing reproducibility, while new algorithms should be published with a code implementation (hopefully directly in DynamicalSystems.jl). However, one must be aware that having code does not replace having knowledge. Having a pre-made implementation of an algorithm can lead into the trap of "just using it", without really knowing what it does or what it means. Thus, one must always be vigilant, and only use the code once there is understanding. We recommend students to attempt to write their own versions of the algorithms we describe here, and later compare with DynamicalSystems.jl implementations. Since DynamicalSystems.jl is open source, it allows one to look inside every nook and cranny of an algorithm implementation. Another benefit of using Julia for this book is how trivial it is to access the source code of any function. Simply preface any function call with `@edit`, e.g. `@edit trajectory(...)`, and this will bring you to the source code of that function.

1.4 The Jacobian, Linearized Dynamics and Stability

Previously we've mentioned *fixed points*, that satisfy $f(\mathbf{x}^*) = 0$ for continuous systems and $f(\mathbf{x}^*) = \mathbf{x}^*$ for discrete systems. In the rest of this section we care to answer one simple, but important question: when is a fixed point of a dynamical system "stable"? We will answer this question by coming up with an intuitive definition of stability, in terms of what happens infinitesimally close around a state space point $\mathbf{x}$ as time progresses.

Let's take a fixed point $\mathbf{x}^*$ and perturb it by an infinitesimal amount $\mathbf{y}$. We are interested in the dynamics of $\mathbf{x} = \mathbf{x}^* + \mathbf{y}$ and whether $\mathbf{x}$ will go further away or towards $\mathbf{x}^*$ as time progresses. For simplicity let's see what is happening in the continuous time case, and because $\mathbf{y}$ is "very small", we can linearize the dynamics with respect to $\mathbf{x}^*$, using a Taylor expansion

$$\dot{\mathbf{x}} = \dot{\mathbf{y}} = f(\mathbf{x}^* + \mathbf{y}) = f(\mathbf{x}^*) + J_f(\mathbf{x}^*) \cdot \mathbf{y} + \cdots \tag{1.7}$$

where $J_f(\mathbf{x}^*)$ stands for the Jacobian matrix of f at $\mathbf{x}^*$ with elements

Chapter 3
Defining and Measuring Chaos

Abstract Deterministic chaos is difficult to define in precise mathematical terms, but one does not need advanced mathematics to understand its basic ingredients. Here we define chaos through the intuitive concepts of sensitive dependence on initial conditions and stretching and folding. We discuss how volumes grow in the state space using Lyapunov exponents. We then focus on practical ways to compute these exponents and use them to distinguish chaotic from periodic trajectories.

3.1 Sensitive Dependence on Initial Conditions

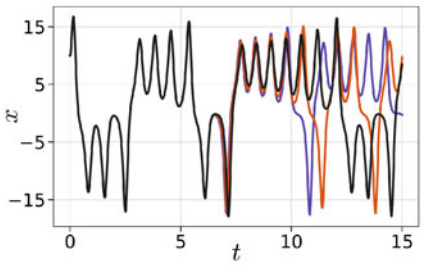
> For want of a nail the shoe was lost.
> For want of a shoe the horse was lost.
> For want of a horse the rider was lost.
> For want of a rider the battle was lost.
> For want of a battle the kingdom was lost.
> And all for the want of a horseshoe nail. [32]

This proverb has been in use over the centuries to illustrate how seemingly minuscule events can have unforeseen and grave consequences. In recent pop culture the term *butterfly effect* is used more often, inspired by the idea that the flap of the wings of a butterfly could lead to the creation of a tornado later on.[1] The butterfly effect is set in the spirit of exaggeration, but as we will see in this chapter, it has a solid scientific basis. It means that tiny differences in the initial condition of a system can lead to large differences as the system evolves. All systems that are chaotic display this effect (and don't worry, we will define chaos concretely in Sect. 3.2.2).

To demonstrate the effect, we consider the Lorenz-63 model from (1.5). In Fig. 3.1 we create three initial conditions with small differences between them and evolve all three of them forward in time, plotting them in purple, orange and

[1] A flap of butterfly wings can never have such consequences. The statement is more poetic than scientific.

G. Datseris and U. Parlitz, *Nonlinear Dynamics*, Undergraduate Lecture Notes in Physics,
https://doi.org/10.1007/978-3-030-91032-7_3




```
using DynamicalSystems, PyPlot
u0 = [10.0,10,10]
lo = Systems.lorenz(u0)
for i in 1:3
    u = u0 .+ i*1e-3
    tr = trajectory(lo, 15, u)
    plot(0:0.01:15, tr[:, 1])
end
```

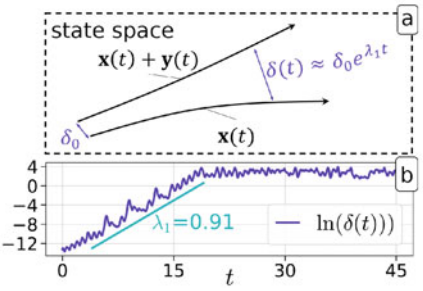
Fig. 3.1 Sensitive dependence on the initial condition illustrated for the Lorenz-63 system simulated with three different, but very similar, initial conditions

black color in Fig. 3.1. Initially there is no visual difference, and only the black curve (plotted last) is seen. But after some time $t \approx 8$ the three different trajectories become completely separated. A 3D animation of this is available online at animations/3/trajectory_divergence. It is also important to note that decreasing the initial distance between the trajectories would not eliminate the effect. It would take longer, but eventually the trajectories would still separate (see Sect. 3.1.2). This is the simplest demonstration of *sensitive dependence on the initial condition*, the mathematically precise term to describe the butterfly effect. It means that the *exact* evolution of a dynamical system depends sensitively on the *exact* initial condition. It should be noted however that if we are interested in averaged properties of a dynamical system instead of the exact evolution of a single trajectory starting at some particular initial value, then sensitive dependence isn't much of a problem due to ergodicity, see Chap. 8.

3.1.1 Largest Lyapunov Exponent

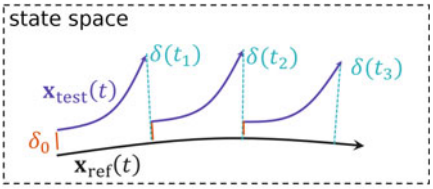
Let's imagine a state $\mathbf{x}(t)$ of a chaotic dynamical system evolving in time. At time $t = 0$ we create another trajectory by perturbing the original one with a small perturbation $\mathbf{y}$, with $\delta = ||\mathbf{y}||$, as shown in Fig. 3.2. If the dynamics is chaotic, the original and the perturbed trajectories separate more and more in time as they evolve. For small perturbations, this separation is happening approximately exponentially fast. After some time t the two trajectories will have a distance $\delta(t) \approx \delta_0 \exp(\lambda_1 t)$. The quantity λ_1 is called the *largest Lyapunov exponent* and quantifies the *exponential divergence* of nearby trajectories.

After some time of exponential divergence, the distance between the two trajectories will saturate and stop increasing exponentially, as is also visible in Fig. 3.2. Precisely why this happens will be explained in more detail in Sect. 3.2.4, but it stems from the fact that f is nonlinear and the asymptotic dynamics is bounded in state space. The important point here is that *exponential* divergence of nearby trajectories occurs *only for small distances between trajectories*. This fact is the basis of




```
using DynamicalSystems
using LinearAlgebra, PyPlot
lo = Systems.lorenz([20,20,20.0])
X₁ = trajectory(lo, 45)
u₂ = get_state(lo) .+ 1e-6
X₂ = trajectory(lo, 45, u₂)
δ  = norm.(X₂.data .- X₁.data)
plot(0:0.01:45, log.(δ))
λ = lyapunov(lo, 100000)
plot([4, 19], λ.*[0, 15] .- 13)
```

Fig. 3.2 **a** Sketch of exponential divergence of neighbouring trajectories characterized by the largest Lyapunov exponent λ_1. **b** The evolution of $\delta(t)$ for the Lorenz-63 system, produced via the code on the right. In general, the initial condition should be on or very near to the set whose divergence properties are to be characterized. For the Lorenz-63 model this requirement is not so strict because it is strongly dissipative such that any trajectory converges very quickly to the chaotic attractor



$$\lambda_1 \approx \frac{1}{t_N - t_0} \sum_{i=1}^{N} \ln\left(\frac{\delta(t_i)}{\delta_0}\right)$$

Fig. 3.3 Computing the largest Lyapunov exponent λ_1. The initial distance δ_0 of the two trajectories $\mathbf{x}_{\text{ref}}$ and $\mathbf{x}_{\text{test}}$ which are evolved in parallel has to be chosen very small. Their distance increases with time, and at times $t_i = i \cdot \Delta t$ the orbit $\mathbf{x}_{\text{test}}$ is re-scaled to have again distance δ_0 from $\mathbf{x}_{\text{ref}}$ with the transformation $\mathbf{x}_{\text{test}} \to \mathbf{x}_{\text{ref}} + (\mathbf{x}_{\text{test}} - \mathbf{x}_{\text{ref}})(\delta_0/\delta(t_i))$. λ_1 is approximated by time-averaging the logarithm of the ratio of distances

the algorithm described in Fig. 3.3, which estimates λ_1, and is also implemented as `lyapunov` in DynamicalSystems.jl.

3.1.2 *Predictability Horizon*

The value of λ_1 has immediate practical impact. Imagine for a moment that there exists a "real" physical system that we want to predict its future. Let's also assume that we have a mathematical model that perfectly describes the real system. This is never true in reality, and the imperfect mathematical model introduces further uncertainty, but for now let's just pretend.

We now make a measurement $\tilde{\mathbf{x}}$ of the state of the real system $\mathbf{x}$. Our goal is to, e.g., plug $\tilde{\mathbf{x}}$ into our mathematical model and evolve it in time on a computer, much

faster than it would happen in reality. But there is a problem: our measurement of the state $\mathbf{x}$ is not perfect. For various reasons the measurement $\tilde{\mathbf{x}}$ is accompanied by an error $\mathbf{y}$, $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{y}$. Therefore, our measured state $\mathbf{y}$ has an initial distance from the real state $\delta = ||\mathbf{y}||$. If the system is chaotic, the measured state and the real state will exponentially diverge from each other as they evolve in time as we have discussed so far. Let's say that if the orbits separate by some tolerance Δ, our prediction has too much error and becomes useless. This will happen after time $t_\Delta = \ln(\Delta/\delta)/\lambda_1$. It explains why chaotic systems are "unpredictable", even though deterministic, since any attempt at prediction will eventually become unusable.

The characteristic time scale $1/\lambda_1$ is often called the *Lyapunov time* and defines the *predictability horizon* t_Δ. One more difficulty for predicting chaotic systems is that because of the definition of t_Δ, in order to increase this horizon linearly (e.g., double it), you need to increase the measurement accuracy exponentially (e.g., square it). Although the reasoning regarding the predictability horizon is valid so far, we should be careful with how generally we apply it, see Chap. 12 for further discussion.

3.2 Fate of State Space Volumes

3.2.1 Evolution of an Infinitesimal Uncertainty Volume

So far we have represented "uncertainty" of a measurement as a single perturbation of an initial condition. While this is useful for conceptualizing divergence of trajectories, a more realistic scenario is to represent uncertainty as an infinitesimal volume of perturbed initial conditions around the state of interest $\mathbf{x}_0$. Then, we are interested to see how this infinitesimal volume changes size and shape as we evolve the system.

To simplify, we will consider the initial volume as a D-dimensional hypersphere around an initial condition $\mathbf{x}_0$. Once we evolve the infinitesimal hypersphere, we will see it being deformed into an ellipsoid as shown in Fig. 3.4. Movies of this process for continuous and discrete systems in 3D can be seen online in animations/3/volume_growth. The ellipsoid will also rotate over time, but for this discussion we only care about the change in its size. We can quantify the change

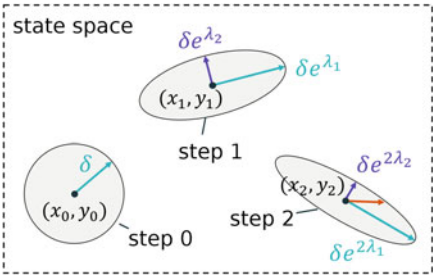


Fig. 3.4 Evolution of an infinitesimal perturbation disk along a trajectory of a two-dimensional discrete system. From step 1 to 2, the purple vector gets mapped to the orange one (because arbitrary perturbations align towards the maximal expansion direction)

Further Reading

Poincaré is without a doubt one of the most important figures for the theory of dynamical systems, nonlinear dynamics and deterministic chaos. Not only he had a plethora of contributions of tremendous impact, including some of the most foundational theorems, but also he was also the first to discover sensitive dependence on initial conditions and deterministic chaos [33] (some sources wrongly attribute this to Lorenz, see below). He first encountered sensitive dependence in the context of the three body problem. Poincaré's goal was to prove that the solar system (represented by three gravitationally attractive bodies, Sun, Moon and Earth) was stable. To progress with his proof, at some point he made the assumption that trajectories that were initially close, would stay close indefinitely. This seemed reasonable at first, given that until that time only regular dynamics had been studied. Thankfully for us, Poincaré realised that this assumption was false, see the article by Green [34] for the interesting story behind these events. What Poincaré understood was that the three body system is chaotic and thus displays sensitive dependence on initial conditions.

The term "butterfly effect" is originating from a title of a talk of Lorenz, "Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?" which is based on his monumental paper on deterministic nonperiodic flow [8]. Lorenz, while not being the first to discover deterministic chaos, is often named the "father" of chaos theory. This is justified, because Lorenz's discoveries and discussions made deterministic chaos be understood as the widespread and almost universal phenomenon that it is, thus establishing the importance of nonlinear dynamics and chaos in the scientific community. Lorenz was also the first person to rigorously discuss and appreciate the impact of chaos and sensitive dependence on initial conditions for the real world, with extra focus on what this means for weather and climate. For more about E. Lorenz, see the discussion by Ott [35].

A relevant textbook that is written in a mathematical tone with proofs and deeper information related to deterministic chaos and its definition is *Chaos: An introduction to Dynamical Systems* by Alligood, Sauer and Yorke. See also *Dynamics: the geometry of behavior* by Abraham and Shaw [36] for several illustrative demonstrations of chaos and stretching and folding.

Lyapunov had a lot of contributions and involvement in the study of stability of dynamical systems [37], hence the Lyapunov exponents bear his name. See also the relevant Scholarpedia page [38] for more historical references. The existence and other properties of Lyapunov exponents are rigorous mathematical results, coming from the Oseledets theorem [39]. The practical method of calculating Lyapunov exponents that we presented here is based on the work of Shimada and Nagashima [40] and Benettin et al. [41]. See [42] for an illustrative description of the algorithm, [43] for further details and alternative algorithms, and [44] for a more recent review on Lyapunov exponents. The most comprehensive treatment of Lyapunov exponents and vectors can be found in book written by Pikovsky and Politi devoted only to this topic [45].

In this chapter we mentioned that at least one positive Lyapunov exponent and bounded evolution of a nonlinear dynamical system means deterministic chaos. Precisely defining deterministic chaos is unfortunately a bit more complicated than that. A nice discussion of the shortcomings of the various previous definitions of chaos is given by Hunt and Ott in [46]. The authors there define an improved definition of chaos based on a quantity called expansion entropy, which is also available in DynamicalSystems.jl as `expansionentropy`.

Besides the Lyapunov exponents there are several other methods that can quantify chaos based on the linearized (or tangent) dynamics. One of them is GALI [47], which is implemented as `gali` in DynamicalSystems.jl. GALI looks at how fast the various perturbation vectors that compose volumes in the linearized space align towards the direction of maximal increase. This alignment will happen exponentially fast for chaotic motion, or polynomially fast (or no alignment at all) for regular motion. Other methods that can be used as chaotic quantifiers are the Orthogonal Fast Lyapunov Indicators (OFLI) [48, 49] or the Mean Exponential Growth of Nearby Orbits (MEGNO) [50]. These methods (and others) are discussed in detail in *Chaos Detection and Predictability* [51] (several authors). Articles that compare the efficiency of methods to quantify chaos are also cited in this book. For example, a comparative study of all methods that look at perturbation growth in the linearized space is [52]. Worth mentioning is a recent different quantitative description of chaos still requiring a known dynamic rule by Wernecke et al. [53]. Instead of using the linearized dynamics here one evolves several nearby trajectories in the real state space and looks at their cross-correlations. It is intuitive (and also can be shown explicitly) that correlation functions of chaotic systems decay exponentially simply because of the exponential separation of nearby orbits. Based on this, this method can claim not only if the trajectory is chaotic or not, but also to what degree it is chaotic (by looking at how the correlations decay). This method is implemented in DynamicalSystems.jl as `predictability`.

Selected Exercises

3.1 Write your own function that takes as an input a discrete dynamical system and performs the algorithm described by Fig. 3.3 to calculate the maximum Lyapunov exponent λ_1. Apply it to the standard map (1.4) and calculate its λ_1 for k ranging from 0.6 to 1.2, always starting with initial condition $(0.1, 0.11)$. *Hint: the chosen test state does not really matter, provided that it is chosen close enough to the reference state.*

3.2 Repeat the above exercise, but now for the Lorenz-63 system for ρ ranging from 20 to 100. Since this is a continuous system, and you need to solve ODEs, you can do it as follows: evolve the two initial conditions for Δt, then stop integration, obtain the final states, and then re-normalize the test one as in Fig. 3.3. Then set this as a new initial condition and evolve for Δt more, also evolving the existing reference state without any modification. Calculate their

distance, re-normalize, and repeat until necessary. *Hint: be careful to evolve both initial conditions for exactly the same amount* Δt!

3.3 Compare your results from the previous two exercises with the result of `lyapunov` from DynamicalSystems.jl and confirm their correctness.

3.4 Implement the full algorithm for calculating the Lyapunov spectrum of a discrete dynamical system as discussed in detail in Sect. 3.2.2. Apply it to the standard map for k ranging from 0.6 to 1.2, always starting with initial condition (0.1, 0.11). Plot the resulting exponents. For a given k what should be the sum of the two Lyapunov exponents? Use this answer to confirm the validity of your implementation.

3.5 Repeat the above exercise for a continuous dynamical system, specifically the Lorenz-63 system, (1.5), versus the parameter ρ from 20 to 100, and plot the resulting three exponents. *Hint: since this is a continuous system you must make a "new" dynamic rule that contains* (3.1) *and plug that into your differential equations solver.*

3.6 Calculate analytically what the sum of the Lyapunov exponents of the Lorenz-63 should be for a given set of parameters, and use this fact to validate your implementation of the previous exercise.

3.7 Compare your Lyapunov spectrum results of the discrete and continuous systems of the previous exercises with the `lyapunovspectrum` function from DynamicalSystems.jl (or with Code 3.1, which is almost identical) and confirm again that they are correct.

3.8 Based on the discussed algorithm for calculating the Lyapunov spectrum, analytically show that for a 1D discrete dynamical system, its Lyapunov exponent is calculated by the expression

$$\lambda(x_0) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln \left| \frac{df}{dx}(x_i) \right| .$$

3.9 Calculate the Lyapunov spectrum of the towel map,

$$\begin{aligned} x_{n+1} &= 3.8x_n(1-x_n) - 0.05(y_n + 0.35)(1-2z_n) \\ y_{n+1} &= 0.1\left((y_n + 0.35)\,(1-2z_n) - 1\right)(1-1.9x_n) \\ z_{n+1} &= 3.78z_n(1-z_n) + 0.2y_n \end{aligned}$$

Based on the Lyapunov spectra of the towel map and the Lorenz-63 system, explain the movies in animations/3/volume_growth.

3.10 The maximum Lyapunov exponent is in principle approximated by the formula

$$\lambda_1 = \lim_{t\to\infty} \lim_{\mathbf{y}_0\to 0} \frac{1}{t} \ln \left(\frac{|\mathbf{y}(t)|}{|\mathbf{y}_0|} \right)$$

for a perturbation $\mathbf{y}_0$ of some initial condition $\mathbf{x}_0$. Consider a continuous dynamical system with a fixed point $\mathbf{x}^*$. If λ_1 is the largest Lyapunov exponent

characterizing this "set" (a single point is also a set) and μ_i the eigenvalues of the Jacobian there, show that $\lambda_1 = \mathrm{Re}(\mu_1)$ with μ_1 the eigenvalue with largest real part. *Hint: express the perturbation in the eigenvectors of the Jacobian as in Sect. 1.4 and then extract λ_1 via its definition.*

3.11 Repeat the above exercise for a discrete system to find that $\lambda_1 = \log(|\mu_1|)$ with μ_1 the eigenvalue with largest absolute value.

3.12 Without using code, provide the signs of Lyapunov exponents (choosing between $+, 0, -$) the following kind of orbits should have, in the case of three dimensional discrete, or continuous, systems: stable fixed point, unstable fixed point, stable periodic orbit, stable quasiperiodic orbit, chaotic attractor. *Hint: for example, a stable fixed point of a discrete system is characterized by the triplet* $(-, -, -)$.

3.13 Modify the code you wrote in exercise 3.1/3.2 to provide a convergence time-series of λ_1 instead of the final value. For the Lorenz-63 system, report how the speed of convergence depends on the choice of d_0, Δt.

3.14 Modify the code you wrote in exercise 3.5 so that you can obtain timeseries of convergence for all Lyapunov exponents. How fast do the exponents converge for the Lorenz-63 system? How does convergence depend on the size of Δt?

3.15 Continuing from the above exercise, now apply the same process to the Hénon-Heiles system. Define a convergence measure and initialize several initial conditions all with the same energy. Produce a Poincaré surface of section where each point is color-coded by its convergence time (similarly with Fig. 3.7). What do you observe?

3.16 Consider the following 4D Hamiltonian system, defined by

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\left(\mathbf{q}^2 + \mathbf{p}^2\right) + \frac{B}{\sqrt{2}} q_1 \left(3q_2^2 - q_1^2\right) + \frac{1}{8}\mathbf{q}^4$$

with $\mathbf{q} = (q_1, q_2)$, $\mathbf{p} = (p_1, p_2)$ and $B = 0.5$. It is used in nuclear physics to study the quadrupole vibrations of the nuclear surface [54]. For it, create a figure similar to Fig. 3.7: generate Hamilton's equations of motion, decide on a reasonable energy value, generate initial conditions that all have the same energy, evolve them and obtain their PSOS, and color-code their PSOS plot according to their λ_1. *Hint: if you haven't solved exercise 3.2, you can use* `lyapunov` *from DynamicalSystems.jl.*

3.17 In Sect. 1.5 we discussed conservative and dissipative systems. Some systems are mixed in the sense of having sets in the state space that are either dissipative or conservative. A simple case of this is due to Sprott [55]

$$\dot{x} = y + 2xy + xz, \quad \dot{y} = 1 - 2x^2 + yz, \quad \dot{z} = x - x^2 - y^2. \tag{3.4}$$

For this system the initial condition $(1, 0, 0)$ leads to conservative motion on a 2-torus, while $(2, 0, 0)$ leads to dissipative motion on a chaotic attractor. Confirm that this is true by calculating how the volume of an infinitesimal sphere would change along the orbit in each case.