

Chaos from Zero

How mathematics describes the world,
and where it stops predicting it

Konrad Cinkusz

Every number checked by code run on Python 3.14.7, [numpy 2.5.3](#) and [matplotlib 3.11.2](#),
September 2026.

Every listing is a file under `code/` in `konradcinkusz/chaos-theory-for-dummies`.

Prose: CC BY-NC-SA 4.0 · Code: MIT

Edition 0.1

Chaos from Zero: how mathematics describes the world, and where it stops predicting it. Edition 0.1.

Copyright © 2026 Konrad Cinkusz. All rights reserved except as licensed here.

The prose is licensed CC BY-NC-SA 4.0. The code — every file under `code/`, and every listing this book prints from one — is licensed MIT.

Self-published by the author. There is no imprint, no distributor and no ISBN.

Published in two languages, English and Polish, from one repository. Also available in Polish as *Chaos od zera*.

Versions pinned September 2026. Every number in this book that you could not work out in your head was produced by a program in the repository, and the repository's build runs every one of those programs again on every change and refuses to build if a single printed digit no longer matches.

This book is not affiliated with any series of books for beginners, whatever the repository it lives in happens to be called.

Contents

How to use this book	viii
Introduction	xi
I The clockwork: how mathematics describes the world	1
1 A model is a machine for predicting	2
1.1 Three parts of every model	2
1.2 Your laboratory	3
1.3 A model that steps	4
1.4 Wrong, and useful	7
1.5 What happens to a small mistake?	7
1.6 The clockwork	9
Can you?	10
Checkpoint	11
2 The rule of the next step	12
2.1 Today, tomorrow, the day after	12
2.2 One straight line, three fates	14
2.3 Slow explosions, and how many steps until	16
2.4 Where the rule stands still	18
2.5 Staying put, and running away	20
2.6 The slope: how much a nudge is multiplied	21
2.7 Seeing it: the cobweb diagram	22
Can you?	24
Checkpoint	25
3 Feedback: why nothing grows forever	26
3.1 Doubling, and why it cannot last	26
3.2 Feedback	27
3.3 Growth slows as room runs out	28
3.4 Two pushes that do not add up	30
3.5 Where it settles, and the slope	31
3.6 Overshoot	33
3.7 When the settling stops	36
Can you?	38
Checkpoint	38

4	Motion: rules of change and the map of every state	40
4.1	A rule that says how fast	40
4.2	Small steps	41
4.3	Look ahead, and average	43
4.4	The map of every state	45
4.5	Settle, cycle or run off	47
4.6	Two bodies: the clockwork at its best	50
	Can you?	51
	Checkpoint	52
II	Where prediction breaks	53
5	One line of algebra, every kind of behaviour	54
5.1	The rule and its one knob	54
5.2	Turning the knob	55
5.3	The steady state, and when it lets go	56
5.4	The cobweb on a curve	59
5.5	Counting the cycle, and failing to	60
5.6	Random-looking, and not random	63
	Can you?	65
	Checkpoint	65
6	The butterfly effect, measured	66
6.1	Two starts, one part in ten billion	66
6.2	The gap doubles	67
6.3	Where the doubling comes from	70
6.4	What a digit is worth	72
6.5	Lorenz's printout	73
6.6	Deterministic, and still unpredictable	75
	Can you?	76
	Checkpoint	77
7	A number for chaos: the Lyapunov exponent	78
7.1	One step at a time	78
7.2	A logarithm turns multiplying into adding	80
7.3	The Lyapunov exponent	81
7.4	How far ahead: the prediction horizon	84
7.5	Chaos coming and going	87
	Can you?	89
	Checkpoint	89
8	The road to chaos, and a number nobody expected	91
8.1	One picture for every setting of the knob	91
8.2	Forks that crowd together	93
8.3	The top of the hump	95
8.4	A number that belongs to the road	97
8.5	Order inside the chaos	100

Can you?	102
Checkpoint	103

III The shapes of chaos 104

9 Three equations for the weather: the Lorenz attractor 105

9.1 Three numbers for a pan of water	105
9.2 Following the flow	107
9.3 Two starts a hair apart	110
9.4 Unpredictable, but not anywhere	112
9.5 What you can still predict	113
Can you?	116
Checkpoint	116

10 Stretch and fold: strange attractors 118

10.1 Stretch, then fold	118
10.2 Hénon's two-line map	120
10.3 One shape for every start	122
10.4 Squeezed, not stopped	123
10.5 Layers within layers	126
10.6 Cutting a flow into a map	128
Can you?	129
Checkpoint	130

11 Fractals: measuring what has no length 131

11.1 A coastline and a ruler	131
11.2 A curve that never settles	133
11.3 Counting copies	135
11.4 Counting boxes	137
11.5 The shape chaos leaves behind	139
11.6 Fractals in the world	141
Can you?	143
Checkpoint	143

12 Iterating in the plane: Julia and Mandelbrot 144

12.1 Numbers that are points	144
12.2 Square, add, and ask whether it escapes	146
12.3 The Mandelbrot set	149
12.4 Julia sets: fix c , move the start	152
12.5 The logistic map in disguise	153
Can you?	156
Checkpoint	157

IV	Chaos in the world	158
13	Pendulums and planets	159
13.1	A pendulum on a pendulum	159
13.2	The number that must not move	161
13.3	A millionth of a radian	162
13.4	Gentle swings and wild ones	165
13.5	Two bodies, three bodies	167
13.6	Is the solar system a clock?	167
	Can you?	169
	Checkpoint	169
14	Weather, climate and the ensemble forecast	170
14.1	A toy atmosphere	170
14.2	Twenty forecasts instead of one	171
14.3	A chance of rain	175
14.4	How far ahead?	177
14.5	Weather and climate	179
	Can you?	182
	Checkpoint	182
15	Life, markets and noise: telling chaos from chance	183
15.1	Two series, one set of numbers	183
15.2	Plot each value against the next	186
15.3	Forecasting from the past	188
15.4	Real data is a mix	190
15.5	Populations, epidemics, hearts	191
15.6	Markets, and what a claim of chaos needs	193
	Can you?	195
	Checkpoint	195
16	Chaos inside the computer	196
16.1	What a computer keeps	196
16.2	A map the computer cannot run	198
16.3	One rule, two programs	200
16.4	Why more digits cannot save you	202
16.5	Wrong path, right picture	203
16.6	Machines that must repeat	205
	Can you?	207
	Checkpoint	207
V	Living with chaos	208
17	Taming chaos: control and synchronisation	209
17.1	A point the orbit cannot keep	209
17.2	A nudge at the right moment	210
17.3	Small change, not force	212

17.4 Two chaotic systems that agree	215
17.5 What has been done with it, and what has not	217
Can you?	219
Checkpoint	219
18 Why you need chaos: a field guide to prediction	220
18.1 Question one: does a small change grow?	220
18.2 Question two: how far is the horizon?	221
18.3 Question three: what survives the horizon?	224
18.4 Question four: chaos, noise, or a small cause?	226
18.5 Question five: can it be steered?	227
18.6 The field guide on one page	227
18.7 Why everyone needs it	228
18.8 Laplace's demon, one last time	229
Can you?	230
Checkpoint	231
 VI Appendices	 232
A Answers	233
B Misreadings of chaos	239
C The mathematics you need, on four pages	245
C.1 Very large and very small numbers	245
C.2 Rules, steps and orbits	245
C.3 Powers, growth and halving	245
C.4 Logarithms	245
C.5 The slope of a curve	246
C.6 Rates of change	246
C.7 Sums and averages	246
C.8 Remainders	247
C.9 Complex numbers	247
C.10 The Greek letters	247
D Further reading	248
D.1 Where the ideas first appeared	248
D.2 Books to read next	249
E Manifest	250
Lab manifest	250
Plot manifest	251
Diagram manifest	252

How to use this book

This book assumes you can add, multiply, work out a percentage and read a graph. It does not assume you remember any algebra, and it never uses a symbol it has not introduced first. Every formula is said in words as well, and where a formula would only get in the way, it is left out. If you stopped doing mathematics at school and have always wanted to know what people mean when they say ‘chaos’, you are the reader it was written for.

What it does ask of you is to take part. The book is built as a sequence of small steps, and at the end of many of them you are asked a question.

The loop

Every chapter is read the same way:

1. **Read a short stretch.** Each section is a few paragraphs.
2. **Stop and think.** A violet box asks you a question. Answer it — on paper, aloud, or as a guess — before you read on. The answer is directly underneath, so cover it with your hand.
3. **Run the code,** if you have a computer beside you. Every listing is a small file, and running it takes a single command.
4. **Do the lab,** if you want to make the idea your own: a lab is a half-written program and a test that tells you when it works.
5. **Take the checkpoint** at the end of the chapter, and check your answers in Appendix A.

Why the book keeps asking you questions

Being asked a question before you are told the answer is uncomfortable. It is meant to be. An answer you have reached for, even a wrong one, stays with you far longer than one you have only read, and a wrong guess that is then corrected is the fastest way to find out which of your intuitions are going to mislead you. This subject is full of those. Most people’s first guess about how fast a small error grows, or about what a simple rule can do, is wrong, and the book is arranged so that you make that guess before you are shown why.

So when a box asks you to answer before you read on, please do. Nobody is marking it.

The laboratory

Everything in the book that a computer can check has been checked by one, and you can check it again. The programs are in the book’s repository, under code/. They are written

in Python and need one tool, `uv`, which installs the right version of Python and every library by itself. Once you have it, from the repository's code/ directory:

```
uv sync
uv run python ch01/falling_stone.py
uv run pytest -k l01_01
```

The first line sets up the laboratory, once. The second runs a listing from Chapter 1. The third runs the test of that chapter's first lab. You do not need to know Python to begin: the first chapters explain every line they print, and the programs stay short throughout.

The boxes

Besides the questions, the text is broken up by boxes, each with one job:

The trap. A mistake almost everybody makes, set out after you have had the chance to make it yourself.

Where you meet this. A real place the idea turns up: a field, a device, a practice.

What we are not proving. What the book states without proving it, and where the proof lives.

Note, Watch out. An aside, and a caution.

chaoslab. A piece of the book's own small library of chaos, which the listings share.

Lab. Something to do at the keyboard, with a test that tells you when you have done it.

Numbers you can trust

Any number in this book that you could not work out in your head was produced by a program, and the book prints the program's output rather than a number somebody typed. The build checks every one of them again on every change. That matters more than usual here, for a reason the book spends a chapter on: in a chaotic system a computation can go wrong in the last digit and be completely wrong a few dozen steps later. Chapter 16 explains how the book protects itself against that, and why some of its numbers are deliberately rounded.

Reading without a computer

You can. Every listing's output is printed beside it, every figure is on the page, and the questions work on paper. The labs are the only part you will miss, and they are there for readers who want to go further, not to gate the argument.

Two languages, one book

The book is published in English and in Polish from one source. The two editions contain the same chapters, sections, figures, numbers and questions in the same order, and the build compares them to make sure they do.

Introduction

The total eclipse of the Sun that crossed Spain on 12 August 2026 was in the astronomers' tables long before anybody who watched it was born. Its path, its date and the minute it began had been worked out from the motions of the Earth and the Moon, and the tables of eclipses run on, calculated the same way, for thousands of years to come.

Nobody could have told the people waiting under that path, a month earlier, whether their sky would be clear enough to see it.

Both forecasts are made by the same kind of mathematics. In each case a computer takes the state of a piece of the world now, applies rules about how it changes, and steps forward in time. The rules for the air are no secret: they are the same laws of motion and heat that describe the Moon. And yet one forecast is good for millennia and the other for about a week. The difference is not that meteorologists are less careful than astronomers, and it is not that the air is too complicated to write down. The difference has a name, a measurement and a precise mathematical description. It is called chaos, and this book is about it.

What chaos is

In everyday speech chaos means a mess. In mathematics it means something much more interesting, and almost the opposite: a system that follows an exact rule, with no chance involved anywhere, and still behaves in a way that cannot be predicted far ahead. Three things have to be true together:

- **the rule is exact** — the same starting state always leads to the same future, with nothing random added;
- **small differences grow** — two starting states that differ by less than anything you could measure end up, after a while, completely different;
- **it stays in bounds** — the motion does not fly off to infinity, so the growing differences keep being folded back into the same region and never settle down.

Each of those is simple. What surprised the scientists who found them together is how common the combination is. It needs no complicated rule: the first chaotic system in this book is one line of school algebra, and it fits on a pocket calculator. It needs no randomness, no crowd and no mystery. It turns up in dripping taps, swinging pendulums, animal populations, heart rhythms, the orbits of planets and, most famously, the weather.

Why it is worth your time

You might reasonably ask why anyone who is not a scientist should learn this. Here are the reasons that persuaded me.

You live on forecasts. Weather apps, pension projections, epidemic curves, economic outlooks, traffic estimates and the confident answers of computer programs are all predictions made by models. Chaos theory tells you which of them can in principle be trusted far ahead, which only for a short time, and what the honest form of a forecast looks like in each case.

It measures the limit, and the limit is strange. In a chaotic system an error in the starting state grows by roughly the same factor every step. That means improving your measurements a thousandfold does not buy you a thousand times more forecast; it buys you a fixed extra stretch of time, and then the fog closes in exactly as before. Once you have seen this worked out with numbers — and you will, in Part II — you will never again hear ‘we just need better data’ in quite the same way.

It explains why good forecasts come with odds. If the far future cannot be computed as a single answer, what can be computed is a spread of answers and how likely each is. A ‘70 per cent chance of rain’ is not a forecaster hedging. It is the correct answer to a question chaos has made impossible to answer with a yes or no.

It separates things people usually lump together. ‘We cannot predict it’ can mean that nobody knows the rules, that the rules involve chance, or that the rules are known exactly and the future is still out of reach. These are different situations that call for different responses, and chaos is the third one.

It is beautiful, and it is universal. The shapes chaos draws are fractals, structures with detail at every magnification, and some of its numbers turn out to be the same in systems that have nothing in common. A fluid, an electronic circuit and a line of algebra can share a constant that nobody put there.

And it is useful. The same sensitivity that makes chaos unpredictable makes it steerable with very small nudges, and chaotic systems can be made to keep in step with each other. Part V shows both.

What this book is, and is not

It is not a textbook, and it will not make you a mathematician. It is also not a book of pictures with the mathematics taken out. Every idea in it is the real idea, developed from arithmetic upwards, and every claim a computer can check has been checked by one; the programs are printed in the book and you can run them yourself. Where the book states something it does not prove, it says so and says where the proof is.

It starts from nothing. Part I (Chapters 1 to 4) shows how mathematics describes the world at all: what a model is, how a rule is stepped forward in time, why feedback stops things growing for ever, and how motion is described. Part II (Chapters 5 to 8) finds chaos in a single line of algebra, measures how fast small errors grow, turns that into a single number and a forecasting horizon, and follows the road from order to chaos. Part III (Chapters 9 to 12) looks at the shapes chaos makes: the Lorenz attractor, strange attractors, fractals and the Mandelbrot set. Part IV (Chapters 13 to 16) takes it into the world: pendulums and planets, the weather forecast, living systems and markets, and the computer itself. Part V (Chapters 17 and 18) shows how chaos can be controlled, and ends with a field guide to judging any prediction you are given.

The question

In 1814 the mathematician Pierre-Simon Laplace imagined an intelligence that knew every force in nature and the exact state of everything at one instant. For such an intelligence, he wrote, nothing would be uncertain: the future would be as plain as the past. It is the most confident sentence ever written about prediction, and for a century it looked like the natural conclusion of physics.

This book asks whether he was right, and answers with numbers you can compute yourself. Chapter 1 begins with a stone and a cup of tea.

Part I

The clockwork: how mathematics describes the world

A model is a machine for predicting

You use mathematical models every day without calling them that. The app that says your train arrives in seven minutes is running one. So is the oven timer, the tide table, the dosage on a medicine packet and the little voice that tells you to leave at ten past eight if you want to be there by nine. Each of them takes a few numbers about now, applies a rule, and hands you a number about later.

This book is about what happens to that promise. Its first four chapters show how well it works — how a line of algebra can tell you where a stone will be, how hot your tea will be, and when a comet will come back. The rest of the book shows where it breaks, why it breaks even when the rule is perfect, and what you can still predict when it does. That last part is the reason to read it: the breaking is not a failure of mathematics, and knowing its shape changes what you should believe about every forecast you are given.

What you will be able to do

When you have finished this chapter you will be able to:

- name the state, the rule and the prediction of any model put in front of you
- run a model written in Python and read what it prints
- say what it means for a model to be wrong, and why a wrong model can still be useful
- say what happens to a small error in the start for the two models in this chapter
- state the clockwork claim that Laplace made in 1814, and the question the rest of the book asks of it

1.1 Three parts of every model

Drop a stone from a bridge. How far has it fallen after three seconds?

A physicist answers with a formula you may remember from school: a stone falling freely near the Earth's surface covers a distance

$$d = \frac{1}{2} g t^2$$

in t seconds, where g is about 9.81 metres per second per second. That one line is a complete model, and it has three parts you will find in every model in this book.

- **The state:** the numbers that say where things are now. Here it is a single number, the time t since the stone was let go.
- **The rule:** how those numbers turn into the thing you want to know. Here it is the formula.

- **The prediction:** what the rule says. Here, a distance.

Stop and think 1.1

Use the formula for $t = 3$. Roughly how far has the stone fallen: about 10 metres, about 30, or about 45?

Answer it before you read on.

Answer. About 44.1 metres: half of 9.81, times $3 \times 3 = 9$.

If you guessed ten or thirty, you are in good company: most people underestimate, because the stone does not fall at a steady speed. It keeps speeding up, so the distance grows with the *square* of the time. Three times as long a fall is nine times as far.

That is the first thing a model buys you: it catches the intuition that was going to mislead you. Nobody has a feeling for t^2 . The formula does not need one.

The trap

‘A model is a copy of reality.’ It is not, and nothing about the formula above resembles a stone. A model is a machine for answering questions: you put numbers in, it gives numbers out, and the only fair test of it is whether those numbers agree with what you then measure. A map is not the territory, and a model is not even a map — it is a map that also tells you the time you will arrive.

1.2 Your laboratory

Every number in this book that you could not work out in your head was computed by a small program, and every one of those programs is in the book’s repository under `code/`. You do not need to be a programmer to read them. They are short, they are written to be read, and each is explained the first time it appears. Here is the stone.

`code/ch01/falling_stone.py`

```

9 G = 9.81 # metres per second, per second, near the Earth's surface
10
11
12 def distance(t: float) -> float:
13     """How far a dropped stone has fallen after t seconds, air ignored."""
14     return 0.5 * G * t * t

```

Listing 1.1: The falling stone as a model: one constant, one rule

Read it the way you read the formula. The line that starts `G =` sets a constant. The lines that start `def distance` define the rule: give it a time `t` and it gives back `0.5 * G * t * t`, which is the formula with the multiplications written out. The text in triple quotes is a note to the reader, and everything after a `#` is a comment. The program then asks the rule the question for every whole second from 0 to 5:

```
code/ch01/falling_stone.py
```

```
20 print(" t (s)    fallen (m)")
21 for t in range(6):
22     print(f"{t:5d}    {distance(t):10.2f}")
```

Listing 1.2: Asking the model six questions

for t in range(6) means ‘do the next line with t set to 0, then 1, and so on up to 5’. Run it from the code/ directory with `uv run python ch01/falling_stone.py` and it prints:

t (s)	fallen (m)
0	0.00
1	4.91
2	19.62
3	44.14
4	78.48
5	122.63

Look at the right-hand column. The stone falls about five metres in the first second and 122.6 metres in five seconds. If you have ever dropped something from a height and counted, that second number feels too large, and it is not.

Note

The front matter says how to install the laboratory: one tool, `uv`, which installs the exact version of Python the book was checked on (3.14.7) and every package at the exact version printed on the copyright page. After that, `uv run python` followed by a file name runs any listing in the book. If you would rather not run anything, the output of every listing is printed beside it; running them is how you find out that the book is not asking you to take its word for anything.

Lab 1.1 — The stone, both ways round

A model answers questions in more than one direction. Given the time, it gives the distance; given the distance, it gives the time; and given a measured drop, it tells you what gravity must be. The starter has the three functions with their descriptions and no bodies. Fill them in until the test passes. The second one needs a square root, which Python spells `math.sqrt`.

Starter: `code/labs/ch01/l01_01_stone.py`

Test: `code/labs/ch01/test_l01_01_stone.py`

Run: `cd code && uv run pytest -k l01_01_stone`

1.3 A model that steps

The stone’s formula lets you jump straight to any time you like. Most models do not. Pour a cup of tea at 90 °C in a room at 20 °C. Isaac Newton described in 1701 how it cools, and

in the form this book uses his rule says: *each minute, the tea loses a fixed fraction of the gap between itself and the room*. Take the fraction to be 0.1, a tenth.

Stop and think 1.2

The gap at the start is $90 - 20 = 70$ degrees. What is the temperature one minute later?

Answer it before you read on.

Answer. 83 °C. A tenth of the gap is 7, and $90 - 7 = 83$.

To know the temperature after two minutes you do the same thing again, to 83: the gap is now 63, a tenth of it is 6.3, and the tea is at 76.7. There is no shortcut you are expected to know. The rule tells you how to get from one minute to the next, and to get anywhere else you take one step at a time.

code/ch01/cooling_cup.py

```
9 ROOM = 20.0    # degrees Celsius
10 K = 0.1        # the fraction of the gap to the room lost each minute
11
12
13 def next_minute(temp: float) -> float:
14     """The temperature one minute later."""
15     return temp - K * (temp - ROOM)
```

Listing 1.3: The rule of the next minute

The function `next_minute` is the whole model. Given the temperature now, it returns the temperature a minute later. To see the future you call it again and again, feeding each answer back in:

code/ch01/cooling_cup.py

```
21 temp = 90.0
22 for minute in range(31):
23     if minute % 5 == 0:
24         print(f"minute {minute:2d}: {temp:5.1f} C")
25     temp = next_minute(temp)
```

Listing 1.4: Feeding each answer back in, thirty times

```
minute 0: 90.0 C
minute 5: 61.3 C
minute 10: 44.4 C
minute 15: 34.4 C
minute 20: 28.5 C
minute 25: 25.0 C
minute 30: 23.0 C
```

The line `temp = next_minute(temp)` is the most important line in this book, and you will

see it in every chapter under a different name. It replaces the state with the next state. Doing that over and over is called *iteration*, and Chapter 2 is about nothing else.

Stop and think 1.3

From the printout, the tea is above 60 °C at minute 5. Is it below 60 at minute 6? What would you have to do to be sure?

Answer it before you read on.

Answer. Yes: it first drops below 60 at minute 6. To be sure you take one more step with the rule, or ask the program for every minute rather than every fifth.

That is how you ask a stepping model a question: you step until the answer appears. It is also why stepping models were rare before computers and are everywhere now. A machine does not mind taking a million steps.

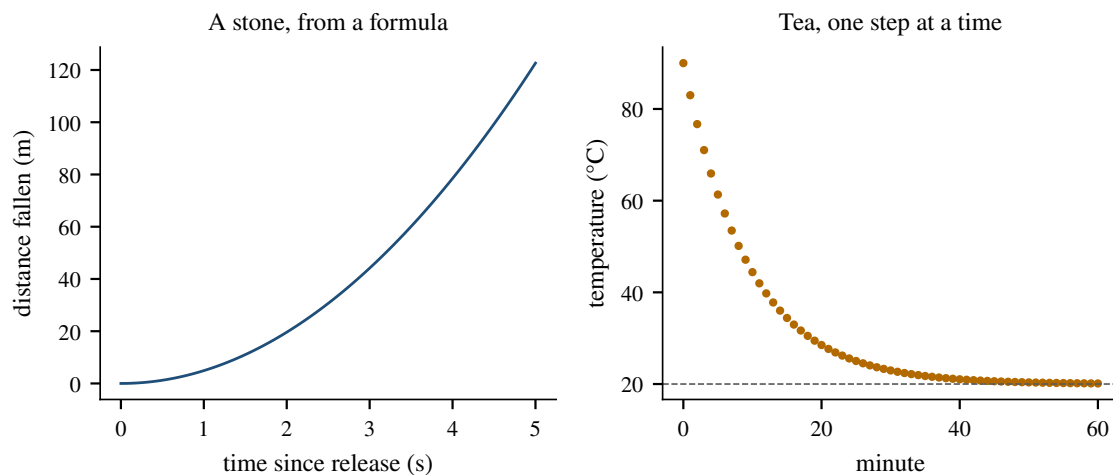


Figure 1.1: Two models of the same kind of question. The stone's formula gives any time directly; the tea's rule gives the next minute, and the curve is its steps joined up.

Lab 1.2 — How long until I can drink it?

Write `cool`, which runs the rule for a number of minutes and keeps every temperature, and `minutes_until`, which answers the question a person actually asks. Your answer for a cup at 90 °C, a room at 20 and a fraction of 0.1 should agree with the one above.

Starter: `code/labs/ch01/l01_02_cooling.py`

Test: `code/labs/ch01/test_l01_02_cooling.py`

Run: `cd code && uv run pytest -k l01_02_cooling`

1.4 Wrong, and useful

Neither model is true. The stone's formula ignores the air, and a feather dropped beside the stone shows you how much that can matter. The tea's rule ignores the cup, the saucer, the steam and your spoon. Measure carefully and you will find both are wrong.

Stop and think 1.4

Since the stone's formula ignores the air, and air obviously matters, is the formula useless?

Answer it before you read on.

Answer. No. For a stone falling a few seconds it is right to within the width of a hand; for a feather it is badly wrong. A model is useful for the questions it answers well enough, and it is your job to know which those are.

The trap

'A model that is wrong is useless.' Every model is wrong, because every model leaves something out — that is what makes it small enough to use. The statistician George Box's often-quoted line says the same thing more briefly: all models are wrong, but some are useful. The useful question is never 'is it true?' but 'is it good enough, for this question, over this range?' Keep that question. Part II of this book is about a range over which a model with no error at all in its rule stops being good enough, and the reason is not what you would guess.

You can check a model against a measurement, and the check can run in either direction. Drop a stone from a ten-metre wall and the formula says it lands after 1.43 seconds. Time a real one and get a slightly different number, and you can turn the formula round and ask what g would have to be. That is the third function in the first lab above, and it is how a model earns trust: by making predictions sharp enough to be wrong, and then not being.

1.5 What happens to a small mistake?

Here is the question the whole book turns on, asked of the gentlest model in it. You never know the start exactly. Your thermometer says 90 °C; the tea is really at 91. How wrong is your prediction half an hour later?

Stop and think 1.5

Two cups start one degree apart and cool by the same rule. After thirty minutes, is the gap between them bigger than one degree, still one degree, or smaller?

Answer it before you read on.

Answer. Smaller: about 0.042 °C. Each minute the gap is multiplied by 0.9, because each cup loses a tenth of its own distance from the room.

code/ch01/two_cups.py

```
13 a, b = 90.0, 91.0
14 for minute in range(31):
15     if minute % 10 == 0:
16         print(f"minute {minute:2d}: gap {b - a:.4f} C")
17     a, b = next_minute(a), next_minute(b)
```

Listing 1.5: Two cups, one degree apart

```
minute 0: gap 1.0000 C
minute 10: gap 0.3487 C
minute 20: gap 0.1216 C
minute 30: gap 0.0424 C
```

The error does not merely stay small; it disappears. Every minute it is multiplied by 0.9, so after ten minutes it is about a third of a degree and after 44 minutes it is less than a hundredth of one. On a logarithmic scale, where each step down is a factor of ten, the gap falls along a straight line:

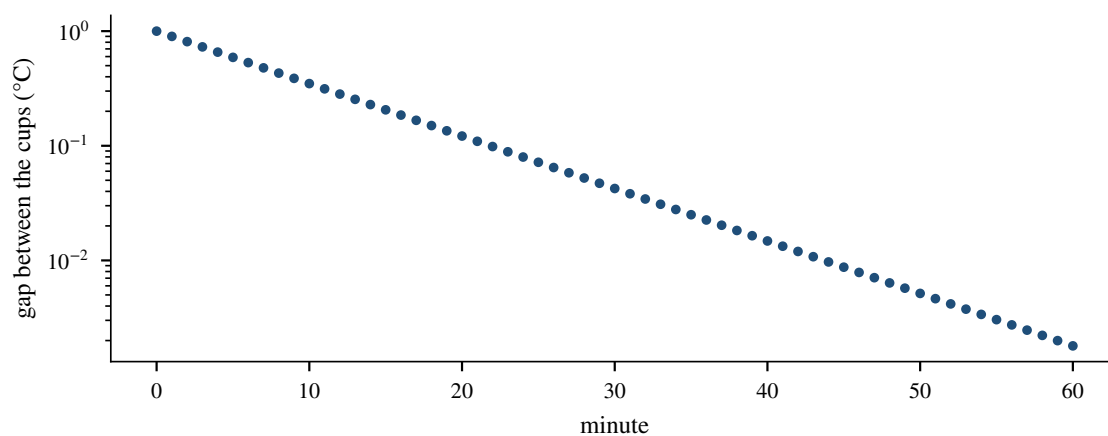


Figure 1.2: The gap between two cups that started one degree apart. On a logarithmic scale it falls along a straight line: the same fraction is lost every minute.

Keep that straight line in your head. A model whose errors shrink forgives you for not knowing the start. The stone's model is nearly as kind: get the moment of release wrong by a tenth of a second and your prediction is off by a fixed amount of time for ever, never by more. Most of the mathematics you met at school describes systems like these, and it is why the idea that nature is predictable feels like common sense.

Where you meet this

The dosing instructions on a medicine packet rest on a model of this kind. Most drugs are cleared from the blood by a roughly fixed fraction every hour, exactly as the tea loses a fixed fraction of its gap (alcohol is the famous exception), so a dose taken a little late or a little large leaves an error that shrinks rather than grows. That forgiveness is why a schedule can be printed on a packet at all.

1.6 The clockwork

Models like these did something astonishing in the eighteenth century. Edmond Halley, using Newton's laws of motion and gravity, argued that the comets seen in 1531, 1607 and 1682 were one object on one orbit, and predicted that it would return around 1758. He died in 1742. The comet was seen again on Christmas night 1758, and it has carried his name ever since.

A rule on paper had reached seventy-six years into the future and been right. Pierre-Simon Laplace drew the conclusion in 1814, in a passage that reads, in its classic English translation:

Given for one instant an intelligence which could comprehend all the forces by which nature is animated and the respective situation of the beings who compose it — an intelligence sufficiently vast to submit these data to analysis — it would embrace in the same formula the movements of the greatest bodies of the universe and those of the lightest atom; for it, nothing would be uncertain and the future, as the past, would be present to its eyes.

The imagined intelligence is known as *Laplace's demon*, and the picture behind it as the *clockwork universe*: know the rule and the present, and the future follows, as surely as the hands of a clock. In 1960 the physicist Eugene Wigner wrote of 'the unreasonable effectiveness of mathematics in the natural sciences', and he meant the same wonder: rules written by people at desks keep turning out to be right about the world.



Figure 1.3: Every model in this book has this shape. A state goes in, a rule turns it into a prediction, and a measurement decides whether to trust it.

Stop and think 1.6

Suppose the rule is exact — not approximately right, but perfectly right — and you know it. Can you then predict the future as far ahead as you like?

Answer it before you read on.

Answer. Only if you also know the start exactly, and you never do. For the tea and the stone that hardly matters, because the error in the start shrinks or stays the same size. Whether it always behaves that kindly is the question the rest of this book answers.

Most people answer yes, and so did most scientists for two hundred years after Laplace. Hold on to your answer. In Chapter 5 you will meet a rule one line long, simpler than the tea's, whose errors do not shrink at all; in Chapter 6 you will measure how fast they grow; and by the end of the book you will be able to say, with numbers you computed, why some things can be predicted for a century and others not for a fortnight. That is chaos theory, and it is not a theory of disorder. It is the mathematics of exactly this question.

Summary

- Every model has three parts: a **state** (the numbers that say where things are now), a **rule** (how they change, or how they turn into an answer), and a **prediction** (what the rule says).
- Some models are formulas you can evaluate at any time you like. Most are **stepping rules**: they tell you how to get from now to the next moment, and you reach the future by applying them over and over. That is iteration.
- Every model is wrong, because every model leaves something out. The useful question is whether it is good enough for this question over this range.
- For the cooling tea, an error in the start **shrinks** by the same factor every minute; for the stone it stays the same size. Models like these forgive you for not knowing the start exactly.
- Laplace's demon is the clockwork claim: know the rule and the present exactly, and the future follows. The rest of the book tests it.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

name the state, the rule and the prediction of any model put in front of you	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
run a model written in Python and read what it prints	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what it means for a model to be wrong, and why a wrong model can still be useful	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what happens to a small error in the start for the two models in this chapter	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
state the clockwork claim that Laplace made in 1814, and the question the rest of the book asks of it	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Name the state, the rule and the prediction of a bus timetable.
2. How far does a stone fall in the first two seconds, air ignored? And in four?
3. Tea at 80°C cools in a room at 20°C , losing a tenth of the gap each minute. What is its temperature after one minute, and after two?
4. Two cups start two degrees apart in the same room and cool by that rule. How far apart are they after one minute?
5. Why can a model that is known to be wrong still be useful?
6. What would Laplace's demon need to know to predict the future, and which of those does the rest of this book say can never be had exactly?

The rule of the next step

Your doctor prescribes one tablet every morning. Each tablet puts 100 milligrams of the drug into your blood, and by the next morning your body has cleared half of whatever was there. Take one a day for a month, a year, ten years: does the amount in your blood keep climbing for as long as you take them?

Hold on to your answer. The question is about a rule applied over and over — halve, add a tablet; halve, add a tablet — and rules of the same kind run savings accounts, cooling tea, populations, loans and a hotel shower. Chapter 1 called `temp = next_minute(temp)` the most important line in the book. This chapter is about what that line does when you run it for ever, and it ends with you able to say what a whole family of rules will do in the long run without running them at all.

What you will be able to do

When you have finished this chapter you will be able to:

- write a rule of the next step in the book's notation and work out its orbit, by hand and in Python
- recognise the straight-line rule behind savings, medicine and cooling, and read its long-run behaviour from one number
- answer a 'how many steps until' question with a logarithm
- find where a rule stands still, and decide whether it stays there when nudged
- measure the slope of a rule at a point, and read a cobweb diagram

2.1 Today, tomorrow, the day after

Put 1000 pounds into an account that pays 5 per cent a year, and leave it alone. After one year the bank adds 50, which is 5 per cent of 1000, and you have 1050. After the second it adds 5 per cent of 1050 and you have 1102.5; after the third, 1157.625. Every year the same thing happens to whatever is there.

Call the balance after n years x_n , read ' x sub n ': the small n says which year and is not multiplied by anything. So $x_0 = 1000$ is where you start and $x_1 = 1050$ is a year later, and the bank's rule is

$$x_{n+1} = 1.05 x_n,$$

next year's balance is 1.05 times this year's, every year.

Every stepping rule in this book has this shape. There is a *state*, a number x saying where things are now; a *rule* f , which stands for 'whatever you do to the state to get the

next one'; and the next state, which is f applied to this one:

$$x_{n+1} = f(x_n).$$

When the step number does not matter the rule is written with an arrow, $x \mapsto 1.05x$, read 'x goes to 1.05 times x'. The list $x_0, x_1, x_2, \dots$ is the *orbit* of x_0 , and producing it, by feeding each answer back in as the next question, is *iteration*.

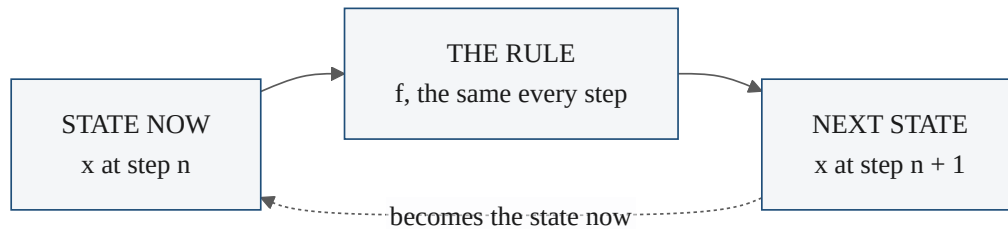


Figure 2.1: Iteration. The rule turns the state now into the next state, which becomes the state now; the same rule is used at every step.

Stop and think 2.1

Take the rule $x \mapsto 2x + 1$: double, then add one. Starting from $x_0 = 1$, what are x_1 , x_2 and x_3 ?

Answer it before you read on.

Answer. 3, 7 and 15. Twice 1 plus one is 3; twice 3 plus one is 7; twice 7 plus one is 15.

There is no way to x_3 except through x_1 and x_2 : a hundred steps ahead takes a hundred steps, tedious by hand and effortless for a machine.

chaoslab

The book's library, `chaoslab`, takes the steps for you. Its orbit is the loop of Chapter 1 with the bookkeeping done once:

code/src/chaoslab/iterate.py

```

0 def orbit(f: Rule, x0: float, n: int) -> list[float]:
1     """Apply the rule `f` n times to x0 and keep every state.
2
3     The list has n + 1 entries: the start, then each step after it.
4     """
5     xs = [x0]
6     for _ in range(n):
7         xs.append(f(xs[-1]))
8     return xs

```

Listing 2.1: Apply a rule n times and keep every state

Its partner `linear(a, b)` hands back the rule $x \mapsto ax + b$, ready to give to `orbit`.

2.2 One straight line, three fates

The savings rule only multiplies. Many rules multiply and then add:

- **The tea** of Chapter 1 loses a tenth of its gap to a room at 20 degrees each minute: from x it goes to $x - 0.1(x - 20)$, which tidies up to $0.9x + 2$.
- **The tablets**: halve, then add 100, or $x \mapsto 0.5x + 100$.
- **A population** in which births outnumber deaths by a fifth each year: $x \mapsto 1.2x$.
- **A rumour** that everyone who knows it passes to one new person a day: $x \mapsto 2x$.

All of them have one shape, $x \mapsto ax + b$: multiply by a fixed number a , then add a fixed number b . Its graph, with the state now along the bottom and the next state up the side, is a straight line, so it is called a *linear* rule — although, as you are about to see, what it does over time is mostly curved. Start four of them from the same place:

code/ch02/fates.py

```

13 RULES = {
14     "2x": linear(2.0, 0.0),          # multiply by 2
15     "0.5x": linear(0.5, 0.0),        # multiply by 0.5
16     "0.5x+5": linear(0.5, 5.0),      # multiply by 0.5, add 5
17     "-0.5x+15": linear(-0.5, 15.0),  # multiply by -0.5, add 15
18 }
19
20
21 def main() -> None:
22     runs = [orbit(rule, 1.0, 8) for rule in RULES.values()]
23     print(" n" + "".join(f"{name:>11}" for name in RULES))
24     for n in range(9):
25         print(f"{n:2d}" + "".join(f"{run[n]:11.4f}" for run in runs))

```

Listing 2.2: Four linear rules, iterated side by side from the same start

n	$2x$	$0.5x$	$0.5x+5$	$-0.5x+15$
0	1.0000	1.0000	1.0000	1.0000
1	2.0000	0.5000	5.5000	14.5000
2	4.0000	0.2500	7.7500	7.7500
3	8.0000	0.1250	8.8750	11.1250
4	16.0000	0.0625	9.4375	9.4375
5	32.0000	0.0312	9.7188	10.2812
6	64.0000	0.0156	9.8594	9.8594
7	128.0000	0.0078	9.9297	10.0703
8	256.0000	0.0039	9.9648	9.9648

The first column doubles for ever and the second halves towards nothing. The third creeps up on 10 from below. The fourth settles on 10 too, but by jumping from side to side, each jump smaller than the last: it is the third column mirrored. Those are the three fates of a linear rule — grow without bound, shrink to nothing, or settle — and a negative multiplier adds alternation, but not a fourth fate.

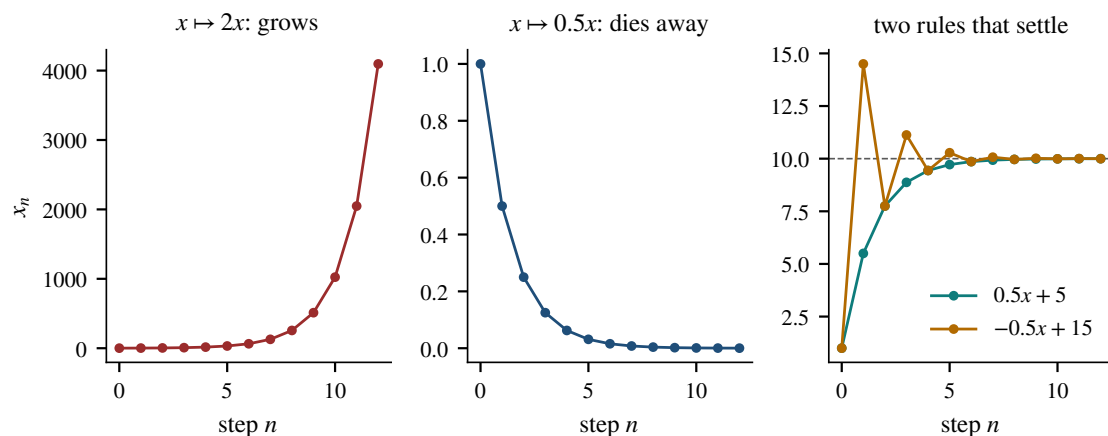


Figure 2.2: The three fates of a linear rule, from the same start. One grows without bound, one dies away, and two settle on the same number, one creeping up on it and one jumping from side to side.

Stop and think 2.2

Back to the savings account. You want the balance after ten years without taking ten steps. What single calculation gives it?

Answer it before you read on.

Answer. Multiply 1000 by 1.05 ten times over: $x_{10} = 1.05^{10} \times 1000$, which is 1628.89.

The small raised 10 is a *power*: 1.05^{10} is 1.05 multiplied by itself ten times. When a rule only multiplies, $x \mapsto ax$, its orbit has a formula, $x_n = a^n x_0$, and you can jump to any step. Growth in which every step multiplies by the same number is called *exponential*; its shrinking twin, with a between 0 and 1, is *exponential decay*.

2.3 Slow explosions, and how many steps until

‘Exponential’ has become a word for ‘fast’. Here is a test of that.

Stop and think 2.3

Two savings accounts both start with 1000. Account A adds 50 every year. Account B grows by 2 per cent a year, which is exponential growth. Which is ahead after thirty years?

Answer it before you read on.

Answer. Account A, with $1000 + 30 \times 50 = 2500$ against B’s 1811. B does not overtake until year 83.

If you chose B, you reasoned from the word rather than the rule: 2 per cent of 1000 is 20, less than A’s 50, and B needs decades of growing on its own growth to catch up.

The trap

‘**Exponential growth means fast growth.**’ It means growth in proportion to what is already there, and the multiplier can be very close to one. At 1 per cent a year a sum takes about 70 years to double. What exponential growth guarantees is not speed but inevitability: however small the rate, it overtakes any growth that adds a fixed amount — eventually, and eventually can be a lifetime.

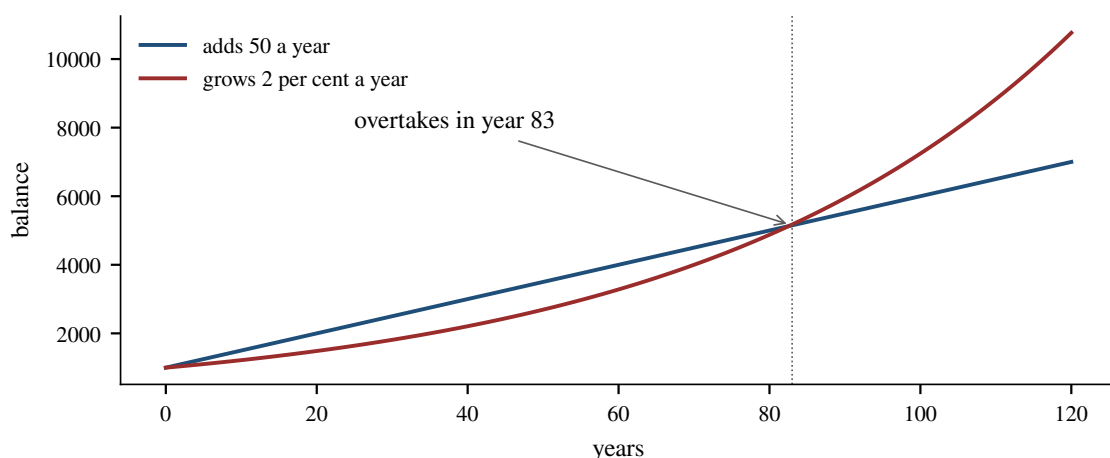


Figure 2.3: Adding a fixed amount against growing by a fixed fraction. The exponential account is behind for most of a century, and then it is not.

The useful question about exponential growth is ‘how many steps until it doubles?’, and the answer does not depend on the start.

Stop and think 2.4

At 7 per cent a year, roughly how many years does a sum take to double: about 5, about 10, or about 14?

Answer it before you read on.

Answer. About 10. Counting whole years, 1000 first passes 2000 in year 11; the precise answer is 10.24 years.

There is a button for this. The question is: for which n is $1.07^n = 2$? A *logarithm* answers exactly that kind of question. Your calculator has one marked $\ln$, the natural logarithm, and the one thing you need to know about it is what it does to a power:

$$\ln(a^n) = n \ln a.$$

It turns ‘multiply by a , n times’ into ‘add $\ln a$, n times’. Take the logarithm of both sides of $a^n = 2$ and you have $n \ln a = \ln 2$, so

$$n = \frac{\ln 2}{\ln a} :$$

the number of steps is the logarithm of how far you want to go, divided by the logarithm of one step. With $\ln 2 = 0.693$ and $\ln 1.07 = 0.0677$, that is 10.24.

code/ch02/doubling.py

```

14 def steps_to_double(rate: float) -> int:
15     """Whole steps of x -> (1 + rate) * x until x has at least doubled."""
16     x, steps = 1.0, 0
17     while x < 2.0:
18         x = (1.0 + rate) * x
19         steps += 1
20     return steps
21
22
23 def doubling_time(rate: float) -> float:
24     """The same question asked of the logarithm: ln 2 / ln(1 + rate)."""
25     return math.log(2.0) / math.log(1.0 + rate)
26
27
28 def main() -> None:
29     print("rate    counted    logarithm    rule of 70")
30     for percent in (1, 2, 3, 5, 7, 10):
31         rate = percent / 100
32         print(f"{percent:3d}%    {steps_to_double(rate):7d}"
33               f"    {doubling_time(rate):9.2f}    {70 / percent:10.1f}")

```

Listing 2.3: How long until a balance doubles, counted and by the logarithm

rate	counted	logarithm	rule of 70
1%	70	69.66	70.0
2%	36	35.00	35.0
3%	24	23.45	23.3
5%	15	14.21	14.0
7%	11	10.24	10.0
10%	8	7.27	7.0

The count is the logarithm rounded up. At 2 per cent that is 35.00 against 36: after 35 years the balance is 1.9999 times what went in, a hair short of double. The last column, the *rule of 70*, divides 70 by the percentage; it works because the logarithm of one step is close to a small rate itself ($\ln 1.07$ is nearly 0.07) and $100 \times \ln 2$ is about 70.

Shrinking works the same way. A drug of which the body clears a fifth every hour keeps 0.8 each hour, and the time until half is left, its *half-life*, is $\ln 0.5 / \ln 0.8$, about 3.1 hours.

Lab 2.1 — How many steps until?

Write `steps_until`, which counts the steps of $x \mapsto ax$ from 1 until x reaches a target, and `time_until`, which asks the logarithm the same question, for growth and for decay. Then write `half_life`. Python's natural logarithm is `math.log`.

Starter: `code/labs/ch02/l02_01_until.py`

Test: `code/labs/ch02/test_l02_01_until.py`

Run: `cd code && uv run pytest -k l02_01_until`

2.4 Where the rule stands still

Now the opening question. The state is the drug in your blood right after the morning tablet, in milligrams: $x \mapsto 0.5x + 100$, starting from $x_0 = 100$.

Stop and think 2.5

Work out the first few mornings: 100, then half of that plus 100, and so on. Does the amount keep climbing for as long as you take the tablets?

Answer it before you read on.

Answer. No. It goes 100, 150, 175, 187.5, 193.75, and on, closing in on 200 without ever reaching it.

Why 200? Because the rule leaves 200 alone: half of 200 is 100, and the tablet puts the 100 straight back. A state the rule does not change is a *fixed point*, and you find it by asking which x satisfies

$$x = f(x).$$

For the tablets, $x = 0.5x + 100$; take $0.5x$ from both sides and $0.5x = 100$, so $x = 200$.

For any linear rule the same steps give

$$x^* = \frac{b}{1-a},$$

the star marking the fixed point. (If a is exactly 1 the rule only keeps adding, and stands still nowhere.) The tea's rule has its fixed point at $2/0.1 = 20$ degrees: the temperature of the room.

code/ch02/drug.py

```

12 KEPT = 0.5      # the fraction of yesterday's drug still in the blood
13 DOSE = 100.0    # milligrams in one tablet
14 tablets = linear(KEPT, DOSE)
15 steady = DOSE / (1.0 - KEPT)    # the fixed point: solve x = 0.5x + 100
16
17
18 def main() -> None:
19     print("day    in blood    short of the fixed point")
20     for day, x in enumerate(orbit(tablets, DOSE, 8)):
21         print(f"{day:3d}    {x:8.2f}    {steady - x:8.2f}")

```

Listing 2.4: One tablet a morning, and the gap to the fixed point

day	in blood	short of the fixed point
0	100.00	100.00
1	150.00	50.00
2	175.00	25.00
3	187.50	12.50
4	193.75	6.25
5	196.88	3.12
6	198.44	1.56
7	199.22	0.78
8	199.61	0.39

The shortfall in the right-hand column halves every morning, because the body halves whatever is there and the tablet adds the same 100 regardless. After 7 days you are within one milligram of 200, and you will never be exactly there. The amount does not climb for ever; it levels off at a fixed point.

Where you meet this

Pharmacologists call this fixed point the *steady state*, and the rule of thumb taught to doctors and pharmacists is that a medicine taken at regular intervals reaches it after four or five half-lives. That is the table above: after four half-lives the shortfall has halved four times, leaving the level at 93.8 per cent of the steady state, and after five at 96.9. When a medicine is needed at full strength at once, a doctor may give a larger first dose, a *loading dose*, which starts the orbit on the fixed point instead of below it.

2.5 Staying put, and running away

Here is a fixed point of a different character. You owe 10 000 on a loan charging 1 per cent a month, and you repay 100 a month, so the balance goes to $1.01x - 100$ each month.

Stop and think 2.6

What happens to the balance over the years? And what if you owe 11 000 instead, with the same interest and the same repayment?

Answer it before you read on.

Answer. At 10 000, nothing, ever: the interest is exactly 100 and your 100 cancels it. At 11 000 the interest is 110, your 100 does not cover it, and after ten years you owe 13 300.

The rule's fixed point is 10 000, but it does not pull you there. Start a thousand above and the debt runs away upwards; start a thousand below, at 9000, and it runs away downwards, to zero, paid off after 232 months, about 19 years.

The trap

'A fixed point is where the system ends up.' It is where the system stays if it is exactly there. Whether a system near it moves towards it or away is a separate question, and the loan's fixed point pushes every other balance away. A fixed point that draws nearby states in is *stable*; one that pushes them out is *unstable*. Finding a fixed point tells you where to look; only its stability tells you whether anything will be there.

The difference is one number. Follow the gap between a state x and the fixed point x^* through a step:

$$f(x) - x^* = (ax + b) - (ax^* + b) = a(x - x^*).$$

The first equals sign uses the fact that x^* is fixed, so $ax^* + b$ is x^* itself. The b cancels, and what is left says: **each step multiplies the gap by a .** The tablets halve it; the loan grows it by 1 per cent a month; the tea, with $a = 0.9$, is why the two cups of Chapter 1 drew closer by a tenth every minute.

What matters is the size of a , written $|a|$, which ignores the sign: $|-0.5| = 0.5$. Below 1 the gap shrinks and the fixed point is stable; above 1 it grows and the fixed point is unstable. The sign says only whether the gap keeps its side or flips.

You have felt a negative a in a hotel shower. You want 38 degrees, the water is at 30, and you overdo it: each turn of the tap moves the temperature by one and a half times the error, the other way. The rule is $x \mapsto x - 1.5(x - 38)$, which is $x \mapsto -0.5x + 57$. From 30 you go to 42, 36, 39, 37.5: too hot, too cold, too hot, each miss half the last. Jerk the tap by two and a half times the error and $a = -1.5$: 30, 50, 20, 65, worse every time.

the multiplier a	a nudge away from x^*	the orbit
above 1	grows	runs away, on one side
between 0 and 1	shrinks	creeps in, from one side
between -1 and 0	shrinks, flipping side	settles, overshooting
below -1	grows, flipping side	runs away, swinging wider

Lab 2.2 — Where does it stop, and does it stay?

For the rule $x \mapsto ax + b$, write `fixed_point`; `is_stable`, which decides whether a nudge away from the fixed point shrinks (try it: start a little off, take one step, compare the gaps); and `steps_to_settle`.

Starter: `code/labs/ch02/l02_02_fixed.py`

Test: `code/labs/ch02/test_l02_02_fixed.py`

Run: `cd code && uv run pytest -k l02_02_fixed`

2.6 The slope: how much a nudge is multiplied

The multiplier a has a second name, which the rest of the book uses. Draw the tablets' rule as a graph, the state now along the bottom and the next state up the side: the straight line $y = 0.5x + 100$. Move along the bottom from 150 to 151, one unit, and the line rises from 175 to 175.5, half a unit. The *slope* of a graph at a point is exactly this: **how far the output moves for each unit the input moves**. A straight line has the same slope everywhere, and for $x \mapsto ax + b$ that slope is a . So the last section's finding, in the words the rest of the book uses, is that *a small error is multiplied by the slope at every step*.

A curve has a slope too, but it differs from point to point, as a hill is steep in one place and nearly flat in another. You measure it at a point by nudging: move the input a tiny amount h either side of x , see how far the output moves, and divide,

$$\text{slope at } x \approx \frac{f(x+h) - f(x-h)}{2h},$$

where $\approx$ means 'very nearly equal'. In `chaoslab`'s slope the nudge h is one millionth: rise over run, for a run too short for the curve to bend.

`code/ch02/slope.py`

```

12 tablets = linear(0.5, 100.0)    # a straight line
13
14
15 def square(x: float) -> float:
16     """A curved rule: x times itself."""
17     return x * x
18
19
20 def main() -> None:
21     for x in (0.0, 150.0, 400.0):
22         print(f"line at x = {x:5.1f}: slope {slope(tablets, x):.4f}")
23     for x in (0.5, 1.5, 3.0):
24         print(f"square at x = {x:5.1f}: slope {slope(square, x):.4f}")

```

Listing 2.5: The slope measured by nudging, on a line and on a curve

```

line   at x =  0.0: slope 0.5000
line   at x = 150.0: slope 0.5000
line   at x = 400.0: slope 0.5000
square at x =  0.5: slope 1.0000
square at x =  1.5: slope 3.0000
square at x =  3.0: slope 6.0000

```

The line's slope is 0.5 wherever you measure it; the curved rule $x \mapsto x \times x$ gets steeper the further right you go.

Stop and think 2.7

The square rule has slope 3 at 1.5. You meant to feed it 1.5 but fed it 1.501, an error of 0.001. Roughly how wrong is the output?

Answer it before you read on.

Answer. By about 0.003: the error is multiplied by the slope. Exactly,
 $1.501 \times 1.501 - 2.25 = 0.003001$.

That is the fact behind every gap in this chapter, now on a curve: an error is multiplied by the slope where it happens to be. On a line the factor never changes, which is why this chapter can foresee everything. On the square rule it does: between 0 and 0.5 the slope is below 1 and errors shrink, beyond 0.5 it is above 1 and they grow. So here is a question this chapter cannot answer. What does an orbit do if its rule is curved, so that it keeps passing from places that shrink errors to places that stretch them? Chapter 3 bends the line, and Chapter 5 follows an orbit that does exactly this.

2.7 Seeing it: the cobweb diagram

You can watch iteration happen on the rule's own graph, with a pencil and no arithmetic. Draw the rule's line and the diagonal $y = x$, whose height always equals its position along the bottom. Then:

1. From x_0 on the bottom axis, go up or down to the line. The height you reach is x_1 .
2. Go across to the diagonal, where height equals position: you now stand above x_1 , ready to use it as the next input.
3. Up or down to the line gives x_2 ; across, and so on.

The corners of the path are the orbit, drawn. [chaoslab](#)'s cobweb lists them; for the tablets, starting from an empty bloodstream:

code/ch02/cobweb_steps.py

```

13 tablets = linear(0.5, 100.0)
14 corners = cobweb(tablets, 0.0, 3)
15 print(f"start      ({corners[0][0]:5.1f}, {corners[0][1]:5.1f})")
16 for k, (x, y) in enumerate(corners[1:]):
17     move = "to the line" if k % 2 == 0 else "to the diagonal"
18     print(f"{move:15} ({x:5.1f}, {y:5.1f})")

```

Listing 2.6: The corners of the cobweb for the tablets

```

start      ( 0.0,  0.0)
to the line ( 0.0, 100.0)
to the diagonal (100.0, 100.0)
to the line  (100.0, 150.0)
to the diagonal (150.0, 150.0)
to the line  (150.0, 175.0)
to the diagonal (175.0, 175.0)

```

The corners on the diagonal carry the orbit, each number twice: 0, 100, 150, 175. The path can rest only where the line crosses the diagonal, because there $x = f(x)$: the crossing is the fixed point, seen.

Stop and think 2.8

The loan's line, $y = 1.01x - 100$, is steeper than the diagonal. Starting a little to one side of the crossing, does the path climb into the crossing or away from it?

Answer it before you read on.

Answer. Away. Each step multiplies the gap to the crossing by 1.01, so the path lands a little further out every time, on whichever side it started.

When the slope is negative the path winds round the crossing, and that web gives the diagram its name. The three panels are the table of Section 2.5, drawn.

So you have something rare in this book: complete foresight. Given $x \mapsto ax + b$ and a start, you can say without taking a step what it will do for ever — settle at $b/(1 - a)$ if $|a| < 1$, run away if $|a| > 1$, alternate if a is negative — and how long it will take. That certainty is what the rest of the book takes away: bend the line and a rule one line long can do things no formula here foresees.

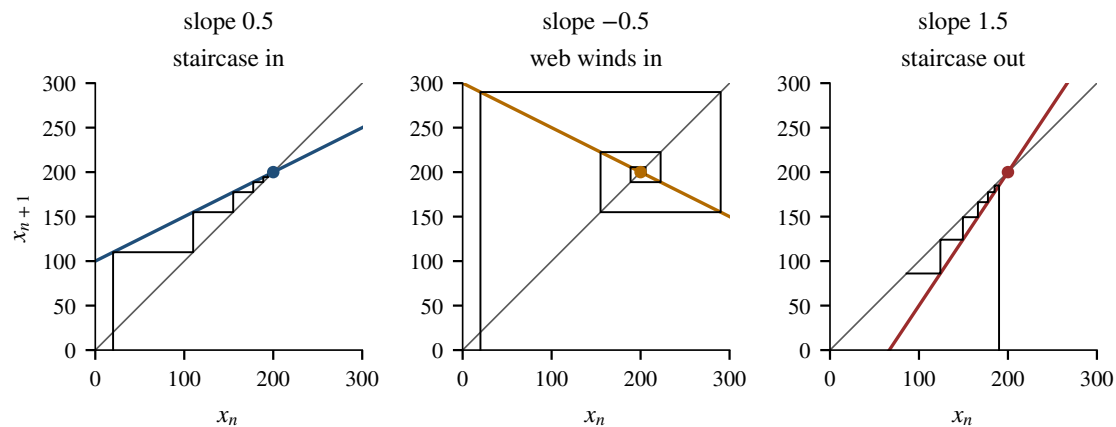


Figure 2.4: Cobwebs on three lines that cross the diagonal at 200. A slope below one in size draws the path into the crossing — a staircase if the slope is positive, a web if it is negative — and a slope above one draws it out.

Summary

- A **rule of the next step** turns the state now into the next, $x_{n+1} = f(x_n)$. Repeating it is **iteration**; the list of states is the **orbit**.
- Savings, cooling and dosing follow a **linear rule**, $x \mapsto ax + b$: it grows without bound, shrinks to nothing, or settles; a negative a makes it **alternate**.
- **Exponential** growth is proportional, not necessarily fast. The **logarithm** says how many steps a change by a factor r takes, $\ln r / \ln a$: the doubling time and the half-life.
- A **fixed point** is a state the rule leaves unchanged: solve $x = f(x)$, which for a linear rule gives $b/(1 - a)$.
- Each step multiplies the gap to the fixed point by a : it is **stable** if $|a| < 1$ and **unstable** if $|a| > 1$, and nothing ends up at an unstable one.
- The **slope**, how far the output moves per unit the input moves, is the factor a small error is multiplied by in one step. A curve's slope differs from point to point.
- The **cobweb diagram** draws an orbit on the rule's graph; on a line it closes in on the crossing with the diagonal when the slope is below one in size.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

write a rule of the next step in the book's notation and work out its orbit, by hand and in Python	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
recognise the straight-line rule behind savings, medicine and cooling, and read its long-run behaviour from one number	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
answer a 'how many steps until' question with a logarithm	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
find where a rule stands still, and decide whether it stays there when nudged	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
measure the slope of a rule at a point, and read a cobweb diagram	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. For the rule $x \mapsto 3x - 2$, starting from $x_0 = 2$, what are x_1 , x_2 and x_3 ?
2. Tea loses a fifth of its gap to a room at 20 degrees every minute. Write its rule in the form $x \mapsto ax + b$.
3. A town grows by 3 per cent a year. Estimate its doubling time with the rule of 70, then say what counting whole years gives.
4. Find the fixed point of $x \mapsto 0.8x + 4$. Is it stable?
5. The rule $x \mapsto 3x - 2$ has a fixed point at 1. Is it stable? What does the first question show?
6. The shower rule $x \mapsto -0.5x + 57$ has slope -0.5 . You are 2 degrees too cold. Where are you after one turn of the tap?

Feedback: why nothing grows forever

In a warm dish with plenty of food, the bacterium *E. coli* divides in two about every twenty minutes. Start with one; each new cell does the same, and so does each of theirs. If nothing got in the way, how long would it take for the colony to weigh more than the Earth?

Nothing like that happens, and this chapter is about why not. Something always pushes back: the food runs short, the space fills, the waste builds up. When the size of a thing changes how fast it grows, that is feedback, and feedback bends the straight-line rules of Chapter 2 into curves. A curve is harder to predict than a line, and the rest of this book is about how much harder. Here you meet the first surprise: a population that runs out of room does not always come to a gentle stop.

What you will be able to do

When you have finished this chapter you will be able to:

- tell positive feedback from negative feedback in a system described in words
- turn ‘growth slows as room runs out’ into a rule, and run it
- test whether the effects of two pushes on a rule add up
- find the fixed points of a curved rule and judge each by its measured slope
- predict from that slope what a population near its settled level will do next

3.1 Doubling, and why it cannot last

Chapter 2 called growth that multiplies by the same number at every step exponential, and counted with a logarithm the steps it takes to double. The bacterium is that kind of growth, with a multiplier of two every twenty minutes.

Stop and think 3.1

Roughly how long until its colony weighs more than the Earth: a century, a year, a month, or two days?

Answer it before you read on.

Answer. Under two days: 133 doublings, which is 44.3 hours. One cell weighs about a million-millionth of a gram.

Most people guess a year or more. The logarithm of Chapter 2 would give the count; the

listing counts in whole numbers instead, and the line `cells = 2 * cells` is the whole of the growth rule.

code/ch03/doubling.py

```

10 CELL = 1e-12      # grams: one E. coli bacterium weighs about a picogram
11 EARTH = 5.97e27   # grams: the mass of the Earth
12 MINUTES = 20      # one doubling, in a warm dish with food to spare
13
14
15 def doublings_to_outweigh(target: float) -> int:
16     """How many doublings until a colony grown from a single cell weighs
17     more than `target` grams."""
18     cells, n = 1, 0
19     while cells * CELL <= target:
20         cells = 2 * cells
21         n = n + 1
22     return n

```

Listing 3.1: Counting doublings until the colony outweighs the Earth

The rest of the program prints the colony every eight hours:

hours	cells	grams
0	1.0e+00	1.0e-12
8	1.7e+07	1.7e-05
16	2.8e+14	2.8e+02
24	4.7e+21	4.7e+09
32	7.9e+28	7.9e+16
40	1.3e+36	1.3e+24

heavier than the Earth after 133 doublings, 44.3 hours

In Python's shorthand for very large and very small numbers, `4.7e+09` is 4.7 with the decimal point moved nine places to the right, and `1.0e-12` is the point moved twelve places to the left. After one day the colony would weigh about 4700 tonnes; less than a day later, more than the planet. Doubling fools the eye because each step adds more than all the steps before it together: put one grain of rice on the first square of a chessboard, two on the second and four on the third, and the last square holds more than the other sixty-three together.

Nothing grows like that for long. A flask of broth stops its bacteria at a few billion cells in every millilitre, when the food is gone and the waste has built up. Growth that feeds on something runs into the limit of that something.

3.2 Feedback

Look again at the doubling rule. The more bacteria there are, the more divisions happen in the next twenty minutes: the size of the colony sets its own rate of growth. A quantity whose size changes its own rate of change is under *feedback*: its effect comes back round and acts on itself.

Feedback comes in two kinds. *Positive feedback* pushes further in the direction things are already going. Hold a microphone near the loudspeaker it feeds and a faint sound goes

round and round the loop, louder every time, until the howl is as loud as the amplifier can make it. *Negative feedback* pushes back. A thermostat switches the heating off when the room is too warm and on when it is too cold. Crowding is negative feedback too: the more animals there are, the less food each one gets and the fewer young the next generation has. Predators and prey make a double loop: more rabbits feed more foxes, more foxes eat more rabbits, and fewer rabbits starve the foxes.



Figure 3.1: Crowding as negative feedback. The more animals there are, the fewer young each of them leaves, so the loop pulls the population back.

Stop and think 3.2

A rumour says a bank may fail, and some of its customers take their money out. Does that push the bank towards failing or away from it — and so, is it positive or negative feedback?

Answer it before you read on.

Answer. Positive. Each withdrawal leaves the bank weaker, which makes the next customer more likely to withdraw too. The run feeds on itself.

The run on Northern Rock in September 2007, the first on a British bank for well over a century, ended only when the government guaranteed the deposits: negative feedback, added from outside. Positive feedback ends like that, or at a limit, and the limit is where the interesting mathematics starts.

3.3 Growth slows as room runs out

Here is the simplest way to write crowding into a rule. Measure the population not in animals but as a fraction of the most the place could ever hold. Call it x : $x = 0$ is an empty place and $x = 1$ is one where every scrap of food and space is taken. Then $1 - x$ is the share of the room still free.

With room to spare, suppose each animal leaves r young that live to breed, so the next generation is r times this one: $x \mapsto rx$. With $r = 2$ that is the doubling rule. Now let the crowding slow it down, by multiplying by the share of the room that is still free:

$$x \mapsto rx(1 - x).$$

In words: the next generation is what this one would breed with room to spare, scaled down by the room that is left. While x is tiny, $1 - x$ is nearly 1 and the rule is nearly the doubling rule. As x grows the brake grows with it, and at $x = 1$ the next generation is nothing. The Belgian mathematician Pierre-François Verhulst proposed a law of growth that slows in this way in 1838; the rule above is his idea taken one generation at a time.

code/ch03/crowding.py

```

11 def next_gen(x: float, r: float) -> float:
12     """The next generation, as a fraction of the room:  $r * x$  is what the
13     population would breed with room to spare, and  $(1 - x)$  is the share
14     of the room still free."""
15     return r * x * (1 - x)

```

Listing 3.2: The crowding rule, one generation at a time

Run it beside plain doubling, both from a thousandth of the room, with $r = 2$:

code/ch03/crowding.py

```

21 r = 2.0
22 free = crowded = 0.001
23 print("gen doubling crowded")
24 for gen in range(15):
25     print(f"{gen:3d} {free:8.4f} {crowded:7.4f}")
26     free = r * free
27     crowded = next_gen(crowded, r)

```

Listing 3.3: Doubling, and doubling with the room running out

gen	doubling	crowded
0	0.0010	0.0010
1	0.0020	0.0020
2	0.0040	0.0040
3	0.0080	0.0079
4	0.0160	0.0158
5	0.0320	0.0310
6	0.0640	0.0601
7	0.1280	0.1130
8	0.2560	0.2005
9	0.5120	0.3206
10	1.0240	0.4356
11	2.0480	0.4917
12	4.0960	0.4999
13	8.1920	0.5000
14	16.3840	0.5000

For six generations the columns are hard to tell apart; a place a few per cent full hardly feels crowded. Then they part. The doubling column passes the whole room at generation 10, which is impossible, and carries on regardless. The crowded column slows, settles at 0.5 and stays there.

Stop and think 3.3

Read the crowded column. In which generations does the population add the most: while it is small, when it is about half-way to where it ends up, or when it is nearly there?

Answer it before you read on.

Answer. About half-way. The biggest jump is from generation 8 to generation 9, where it adds 0.12 of the room while passing 0.25, half the level of 0.5 it settles at.

Small, the population has few animals to breed; nearly full, it has no room to breed into. Checking every level from 0 to 1 in steps of a hundred-thousandth confirms that the growth in one generation is largest at exactly half the settled level. Plotted, the crowded column makes the S-shaped curve of growth that meets a limit.

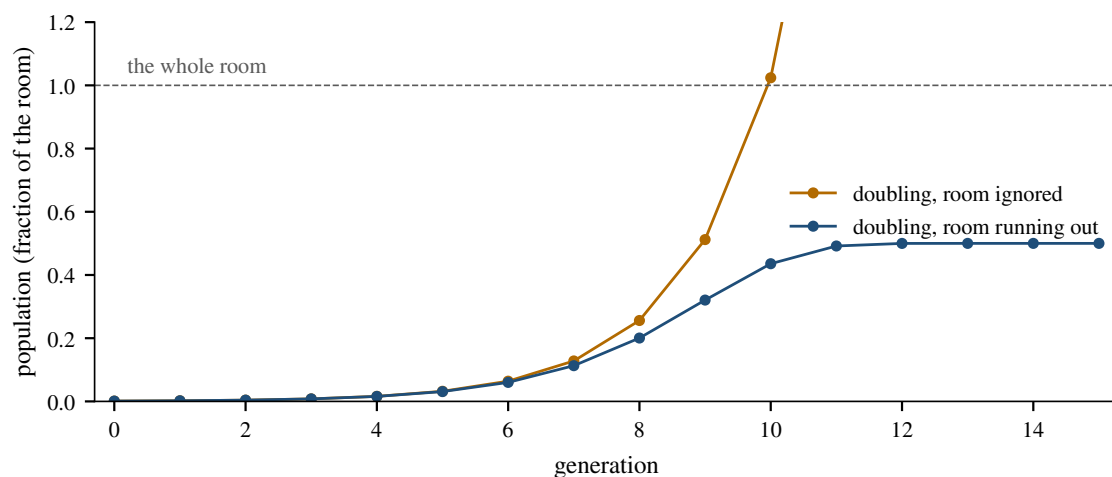


Figure 3.2: Doubling against doubling with the room running out, both from a thousandth of the room. They agree while the place is nearly empty; then one leaves the page and the other bends into an S.

Where you meet this

Fisheries science has used growth curves of this shape since the middle of the twentieth century. A fished-down stock grows back fastest when it is about half its unfished size, so the largest catch that can be taken year after year without shrinking it, the *maximum sustainable yield*, is the one that holds the stock near that half. The idea is written into the 1982 United Nations Convention on the Law of the Sea. It is still a model, and real stocks have collapsed under quotas that looked safe on paper.

3.4 Two pushes that do not add up

Every rule in Chapter 2 was a straight line: draw the next value against this one and you get a straight line. Straight-line rules have a property so natural that you probably

assume it of every rule: the effects of pushes add up. Take the doubling rule with a population of 0.2. Push in an extra 0.1 and the next generation rises by 0.2; push in 0.2 and it rises by 0.4. Two pushes together do what each does separately, added. This is called *superposition*, and it is what makes straight-line rules easy: you can take a problem to pieces, solve each piece and add up the answers.

Now the crowding rule, with $r = 2$ and the same population of 0.2. Unpushed, the next generation is $2 \times 0.2 \times 0.8 = 0.32$. Push in 0.1 and it is $2 \times 0.3 \times 0.7 = 0.42$: the push raised it by 0.1.

Stop and think 3.4

Now push in 0.2, two of those pushes at once. By how much does the next generation rise? Guess first, then work out $2 \times 0.4 \times 0.6$.

Answer it before you read on.

Answer. By 0.16, not 0.2: the next generation is 0.48. The second push was worth only 0.06, because it landed on a more crowded place than the first.

It gets stranger higher up. From a population of 0.6 the next generation is $2 \times 0.6 \times 0.4 = 0.48$; push in 0.1 and it is $2 \times 0.7 \times 0.3 = 0.42$. Adding animals now makes the next generation *smaller*. The same push helps near the bottom and hurts near the top: the effect of a change depends on where you already are. That is what it means for a rule to be *nonlinear*, and it is what feedback does to a rule.

The trap

‘The effect of two pushes is the sum of the effects.’ Only for a straight-line rule. On a curved one the effect of a push depends on where the system already stands and on whatever else is pushing it, so two causes that each do a little can together do much more, much less, or the opposite. Every rule with feedback in it is curved.

3.5 Where it settles, and the slope

The crowded population settled at 0.5. Why there? A population stays put when the next generation is the same size as this one. The empty place is one such level: nothing breeds nothing. The other is where each animal leaves, on average, exactly one young to replace itself. With the share $1 - x$ of the room free each animal leaves $r(1 - x)$, and that is one when $1 - x$ is $1/r$, which is at

$$x = 1 - \frac{1}{r}.$$

For $r = 2$ that is $1 - \frac{1}{2} = \frac{1}{2}$, where the column settled. A level the rule leaves unchanged is a *fixed point*, as in Chapter 2.

Stop and think 3.5

With $r = 1.5$, where does the population settle?

Answer it before you read on.

Answer. At one third: $1/1.5$ is two thirds, and $1 - \frac{2}{3} = \frac{1}{3}$.

A fixed point is only half the answer; the other half is whether a population near it is drawn in or pushed away. Chapter 2 answered that for a straight line: a nudge away from the fixed point is multiplied at every step by the rule's *slope*, and the fixed point holds when the size of the slope is below one. It also found that a curve's slope changes from place to place. That does not spoil the test, because a small enough nudge feels only the curve where it is: zoom in, and near any one point a curve looks straight.

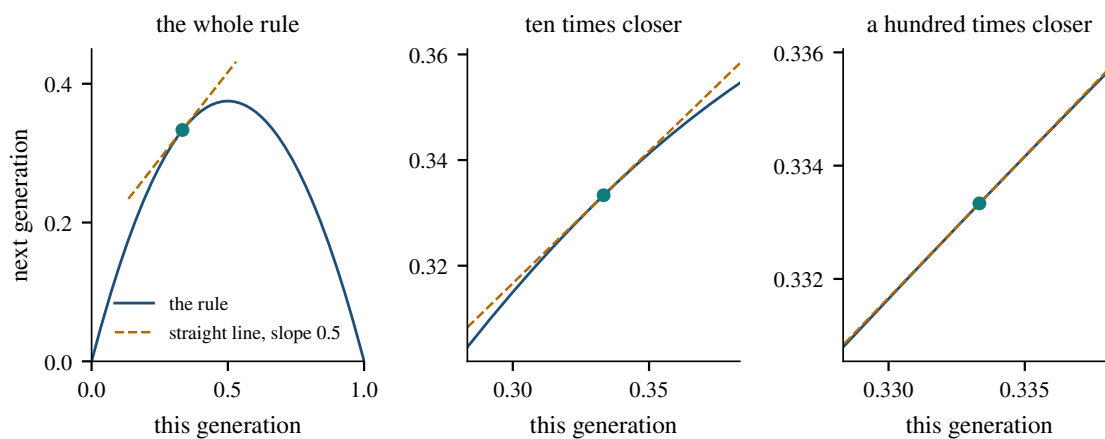


Figure 3.3: The crowding rule with $r = 1.5$, zoomed in on its fixed point at one third. Close enough, the curve is indistinguishable from a straight line, and the steepness of that line decides what happens to a small nudge.

So measure the slope at each fixed point by nudging, with `chaoslab`'s `slope`, as Chapter 2 did:

code/ch03/settle.py

```

16 def slope_at(x: float, r: float) -> float:
17     """The slope of the crowding rule at x, measured by nudging."""
18     return slope(lambda y: next_gen(y, r), x)
19
20
21 def main() -> None:
22     for r in (1.5, 2.0):
23         for x in (0.0, 1 - 1 / r):
24             print(f"r = {r}: at x = {x:.4f} the slope is"
25                   f" {slope_at(x, r):6.3f}")

```

Listing 3.4: The slope of the crowding rule at both fixed points, for two growth rates

```

r = 1.5: at x = 0.0000 the slope is  1.500
r = 1.5: at x = 0.3333 the slope is  0.500
r = 2.0: at x = 0.0000 the slope is  2.000
r = 2.0: at x = 0.5000 the slope is  0.000

```

At the fixed point of one third the slope is 0.500: a small nudge is halved every generation, so the population is drawn back in and the fixed point is *stable*. At the empty place the slope is 1.500: a few animals in an empty place multiply, which is growth, and that fixed point is *unstable*. With $r = 2$ the slope at one half is 0.000. That fixed point sits on the flat top of the hump, where a nudge is all but wiped out in a single generation, which is why the crowded column snapped onto 0.5 so fast.

This is the idea the rest of the book leans on: **at a fixed point of a curve, the slope does the job the multiplier did for a line**, and decides whether a small nudge dies away or grows.

What we are not proving

Taken on trust here: that the slope decides for a curve what the multiplier decided for a line. It holds for nudges small enough that the curve looks straight, which the zoom shows but does not prove; the proof is in any first course on dynamical systems. The measurement estimates what calculus calls the *derivative*, exactly here to every digit printed.

chaoslab

`chaoslab` has the crowding rule as `logistic`: hand it r and it hands back the rule, ready for `slope` and `orbit`. The name is Chapter 5's, where the rule's stranger habits begin.

Lab 3.1 — Read a fixed point's fate off its slope

Write `fixed_points`, which gives both fixed points of the crowding rule; `level_slope`, which measures the slope at the settled level; and `verdict`, which reads from that slope whether a population nudged off the level creeps back, swings back or is pushed away. The tests run the rule for two thousand generations and check your verdict against what happens.

Starter: `code/labs/ch03/l03_01_verdict.py`

Test: `code/labs/ch03/test_l03_01_verdict.py`

Run: `cd code && uv run pytest -k l03_01_verdict`

3.6 Overshoot

Turn the growth rate up to $r = 2.8$. The settled level is $1 - 1/2.8$, about 0.643.

Stop and think 3.6

Start from a small population, as before. Does it creep up to 0.643 and stop, the way the populations with $r = 1.5$ and $r = 2$ did?

Answer it before you read on.

Answer. No. It rises past the level to 0.690 in generation 5, falls below it to 0.599, and swings back and forth, each swing smaller than the last, before it settles.

The trap

‘A limit to growth means growth stops smoothly.’ It can, and with a gentle growth rate it does. With a strong one the population overshoots the level its surroundings can sustain, crashes below it, and oscillates. The limit is still there, but the approach to it is a series of swings, and turned up further the swings need not die away at all.

The slope says why. At the settled level with $r = 2.8$ it is -0.800 . Its size is below one, so a nudge shrinks and the level is stable. But it is negative, and Chapter 2 showed on a straight line what a negative multiplier does: a population a little above the level is sent a little below it, then a little above again. Settling with swings that shrink is called a *damped oscillation*.

What puts the overshoot there is delay. The rule works out a whole generation’s change from how crowded the place is now, and looks again only a generation later. At 0.44, below its level, the population sees plenty of room and breeds a big next generation; by the time those young arrive the room has gone, and the population is past the level. The hotel shower of Chapter 2 overshot for the same reason: the water you feel left the tap a few seconds ago, so you turn the tap further than you need — and then it scalds you.

To test that explanation, take the delay away. Keep the same law of change, but let the population look at its crowding ten times a generation: split each generation’s change into ten instalments, each worked out from the population as it stands after the one before.

code/ch03/overshoot.py

```

14 def in_instalments(x: float, r: float, k: int = 10) -> float:
15     """One generation of the crowding rule, paid in k instalments, each
16     worked out from the population as it stands after the one before."""
17     for _ in range(k):
18         x = x + (next_gen(x, r) - x) / k
19     return x

```

Listing 3.5: The same law of change, paid in ten instalments

code/ch03/overshoot.py

```
27 once = paid = 0.01
28 print(f"r = {r}, settled level {1 - 1 / r:.4f}")
29 print("gen    once    in ten")
30 for gen in range(generations.stop):
31     if gen in generations:
32         print(f"{gen:3d} {once:.4f} {paid:.4f}")
33     once = next_gen(once, r)
34     paid = in_instalments(paid, r)
```

Listing 3.6: Once a generation against ten times a generation

```
r = 2.8, settled level 0.6429
gen    once    in ten
0  0.0100  0.0100
1  0.0277  0.0496
2  0.0755  0.2018
3  0.1954  0.4732
4  0.4401  0.6110
5  0.6900  0.6382
6  0.5990  0.6422
7  0.6726  0.6428
8  0.6166  0.6428
9  0.6619  0.6429
10 0.6266  0.6429
11 0.6551  0.6429
12 0.6326  0.6429
13 0.6508  0.6429
14 0.6364  0.6429
```

Both head for the same level, because the law of change is the same and is zero at the same place, but only the once-a-generation population overshoots. The one paid in instalments even grows faster at first, as its young start breeding within the generation, and still slides into its level without passing it. What makes the overshoot is not the speed of the growth but how late the brakes act. Verhulst's own law is what the instalments become as they are made ever smaller, and it never overshoots.

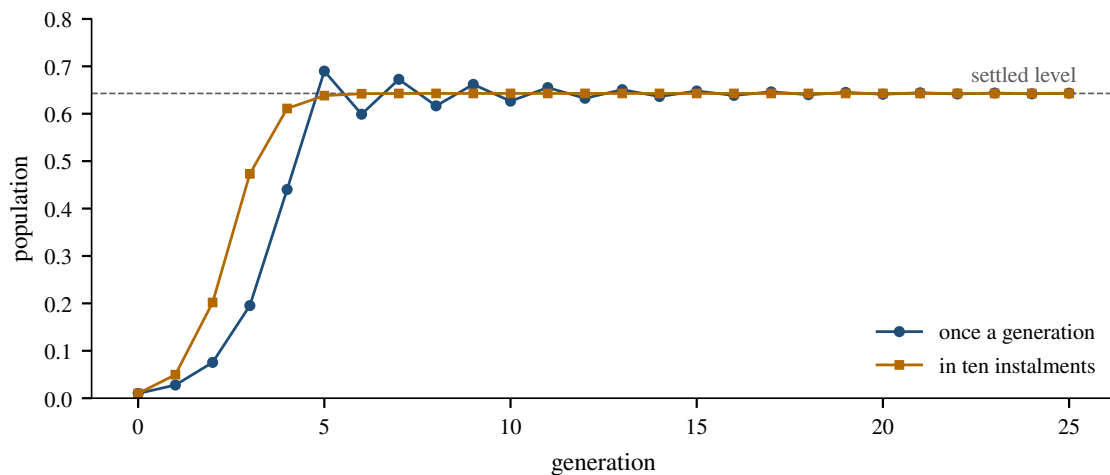


Figure 3.4: The same law of growth with $r = 2.8$. Worked out once a generation it overshoots its level and swings; worked out ten times a generation it slides into the same level.

Lab 3.2 — Late brakes and early brakes

Write the crowding rule, the same rule paid in k instalments, a function that runs either for many generations, and one that says whether a population ever passes its settled level. Then try growth rates of your own.

Starter: `code/labs/ch03/l03_02_instalments.py`

Test: `code/labs/ch03/test_l03_02_instalments.py`

Run: `cd code && uv run pytest -k l03_02_instalments`

3.7 When the settling stops

Turn the growth rate up once more, to $r = 3.1$. The settled level is still there: $1 - 1/3.1$ is about 0.677, and a population placed exactly on it would stay. But the slope there, measured as before, is -1.100 .

Stop and think 3.7

What does a slope of -1.100 do to a small nudge away from the settled level, generation after generation?

Answer it before you read on.

Answer. It flips the nudge to the other side and makes it a tenth bigger, every generation. The nudge grows, so the settled level can no longer hold the population.

code/ch03/two_cycle.py

```

15     r = 3.1
16     level = 1 - 1 / r
17     x = level + 0.001
18     for gen in range(6):
19         print(f"gen {gen}: {x - level:+.5f} from the settled level")
20         x = next_gen(x, r)

```

Listing 3.7: A nudge of a thousandth, generation by generation

code/ch03/two_cycle.py

```

24     compare(3.1, range(60, 66))

```

Listing 3.8: Where the population ends up

```

gen 0: +0.00100 from the settled level
gen 1: -0.00110 from the settled level
gen 2: +0.00121 from the settled level
gen 3: -0.00134 from the settled level
gen 4: +0.00146 from the settled level
gen 5: -0.00162 from the settled level

```

```

r = 3.1, settled level 0.6774

```

```

gen    once  in ten
60  0.5580  0.6774
61  0.7646  0.6774
62  0.5580  0.6774
63  0.7646  0.6774
64  0.5580  0.6774
65  0.7646  0.6774

```

The nudge grows, changing sides every generation, and the population ends up nowhere near the settled level. It swings between two values, 0.558 and 0.765, low one generation and high the next, for ever. Nothing random has come in; the rule is the same single line. The settling has stopped, and in its place is a cycle that repeats every two generations. The population paid in instalments still settles at 0.677: take the delay away and the swinging goes with it.

Somewhere between $r = 2.8$ and $r = 3.1$ the settled level lost its grip. Where exactly, and what the rule does as the growth rate is turned up further still, is the subject of Chapter 5. The rule is one line long, a child could work it with a pencil, and it has not finished surprising you.

Summary

- **Exponential growth**, met in Chapter 2, adds more at each doubling than at all the doublings before, and nothing real grows that way for long.
- **Feedback** is a quantity changing its own rate of change. **Positive feedback**

pushes on (a howl, a bank run) until it meets a limit; **negative feedback** pushes back (a thermostat, crowding).

- Crowding written into doubling gives $x \mapsto rx(1 - x)$, with x the fraction of the room in use: an **S-curve**, growing fastest at half its settled level.
- For a straight-line rule the effects of pushes add up (**superposition**); for a curved, **nonlinear** rule they do not, and a push can help or hurt depending on where the system stands.
- The rule settles where each animal is exactly replaced, at the **fixed point** $1 - 1/r$. The **slope** there, measured by nudging as in Chapter 2, multiplies a small nudge; a size below one makes it stable.
- A negative slope means **overshoot** and a **damped oscillation**. The cause is delay: the same law in small instalments does not overshoot.
- Turned up further, the slope's size passes one and the population swings between two values for ever — the doorway to Chapter 5.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

tell positive feedback from negative feedback in a system described in words	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
turn 'growth slows as room runs out' into a rule, and run it	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
test whether the effects of two pushes on a rule add up	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
find the fixed points of a curved rule and judge each by its measured slope	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
predict from that slope what a population near its settled level will do next	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. On the chessboard, how many grains lie on the eleventh square, and how many on the first ten squares together?
2. Positive or negative feedback: a thermostat; applause that grows louder as more people join in; a price that rises when a product is scarce, so that fewer people buy it?
3. For the crowding rule with $r = 2$, starting at 0.5, what do pushes of 0.1 and of 0.2 do to the next generation?

4. Where does the population settle with $r = 2.5$? And with $r = 1.25$?
5. The slope at a fixed point is -0.5 , and a population starts 0.01 above it. Where is it after one generation, and after two?
6. Why does a population that looks at its crowding ten times a generation not overshoot, when one that looks once a generation does?

Motion: rules of change and the map of every state

The tea of Chapter 1 lived in whole minutes: its rule jumped from one minute to the next. A swinging pendulum does not jump; at every instant it is somewhere and going somewhere. Newton's laws of motion, published in 1687, do not say where a planet will be next Tuesday. They say how fast its velocity is changing right now. How do you get a future out of a rule like that?

This chapter answers with a trick from Chapter 2: take small steps. It then draws the picture the rest of the book is drawn in, a map on which every possible state is a point and every future is a path, and ends with a theorem that says why the weather model of Chapter 9 needs three numbers.

What you will be able to do

When you have finished this chapter you will be able to:

- read a rule of change as a statement about how fast each number of the state is changing right now
- turn a rule of change into a path in code, with Euler's method and with RK4
- use energy to check whether a simulation is telling the truth
- read a phase portrait: its fixed points, its loops and which way the motion goes
- say what motion in a plane can and cannot do, and how many numbers chaos in continuous time needs

4.1 A rule that says how fast

A car's speedometer is a rule of change. It does not tell you where you are; it tells you how fast where you are is changing.

Stop and think 4.1

The speedometer reads 60 kilometres an hour. Roughly where will the car be in one minute? And in one hour?

Answer it before you read on.

Answer. About a kilometre further on in a minute: 60 kilometres an hour is a kilometre a minute. In an hour you cannot say, because the speed will have changed many times.

Over a short time, how fast something is changing tells you how far it goes; over a long time it tells you little. So you go a short way, read the speedometer again, and go on.

Here is the first worked system. A weight hangs on a spring; you pull it one metre from where it rests and let go. Its state is two numbers: its *position* x , how far it is from rest, and its *velocity* v , how fast x is changing. The rule says how fast each of them changes:

- the position changes at the rate v , which is what a velocity is;
- the velocity changes at the rate $-k x$: the spring pulls back towards the middle, harder the further it is stretched. The number k is how stiff the spring is; take $k = 1$. The minus sign says the pull is backwards.

Other books write the same rule as

$$\frac{dx}{dt} = v, \quad \frac{dv}{dt} = -k x,$$

reading $\frac{dx}{dt}$ as ‘how fast x is changing’. A rule of this kind is a *differential equation*. This book never solves one by hand. It follows them in small steps, which is how a computer forecasting the weather does it too.

4.2 Small steps

The simplest way to follow a rule of change is named after Leonhard Euler, who described it in the eighteenth century. Pick a short length of time, the *step*, and move each number for one step at the rate it has now:

$$\text{new } x = x + \text{step} \times v, \quad \text{new } v = v + \text{step} \times (-k x).$$

Then ask the rule again at the new state, and repeat. A rule of change has become a rule of the next step, and iteration takes over.

code/ch04/spring.py

```

11 K = 1.0    # how stiff the spring is, for each kilogram of mass
12
13
14 def rates(x: float, v: float) -> tuple[float, float]:
15     """How fast position and velocity are changing right now."""
16     return v, -K * x    # moves at v; the spring pulls back towards 0
17
18
19 def euler_step(x: float, v: float, dt: float) -> tuple[float, float]:
20     """Pretend the rates stay the same for dt, and move."""
21     dx, dv = rates(x, v)
22     return x + dt * dx, v + dt * dv
23
24

```

```

25 def energy(x: float, v: float) -> float:
26     """Energy of motion plus energy stored in the stretched spring."""
27     return 0.5 * v * v + 0.5 * K * x * x
28
29
30 def main() -> None:
31     x, v, dt = 1.0, 0.0, 0.1          # pulled out 1 metre, let go
32     for step in range(101):           # 100 steps of 0.1 s: ten seconds
33         if step % 20 == 0:
34             print(f"t = {step * dt:4.1f} s    x = {x:6.3f}    "
35                   f"v = {v:6.3f}    energy = {energy(x, v):.4f}")
36             x, v = euler_step(x, v, dt)

```

Listing 4.1: A spring, followed with Euler's method

rates is the rule and `euler_step` is Euler's method, two lines. `energy` adds the energy of motion, half of $v \times v$, to the energy stored in the spring, half of $k \times x \times x$. The weight starts at $x = 1$, at rest, and the steps are 0.1 seconds long.

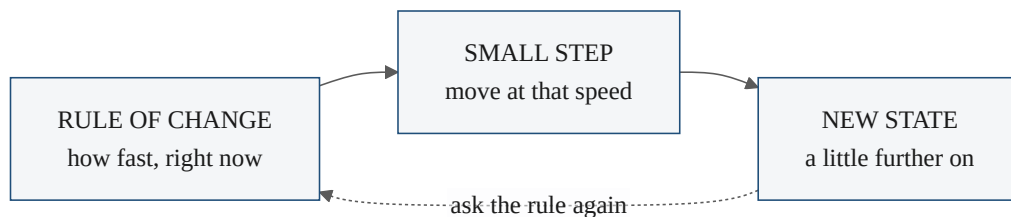


Figure 4.1: Euler's method. The rule gives the rates at this instant; a small step moves as if they stayed the same; at the new state the rule is asked again.

Stop and think 4.2

A real spring with no friction swings for ever, reaching the same distance each time: nothing takes energy out and nothing puts it in. What should the energy column show, line after line?

Answer it before you read on.

Answer. The same number every time: 0.5, half of 1×1 .

t = 0.0 s	x = 1.000	v = 0.000	energy = 0.5000
t = 2.0 s	x = -0.453	v = -1.007	energy = 0.6101
t = 4.0 s	x = -0.810	v = 0.913	energy = 0.7444
t = 6.0 s	x = 1.286	v = 0.402	energy = 0.9083
t = 8.0 s	x = -0.178	v = -1.478	energy = 1.1084
t = 10.0 s	x = -1.409	v = 0.849	energy = 1.3524

It does not. The swings grow and the energy climbs from 0.5 to 1.35 in ten seconds. Nothing in the rule put that energy in. The simulation did.

The reason is a picture. Plot the state as a point, position across and velocity up. With $k = 1$ the energy is half the square of the point's distance from the centre, so the real spring goes round a circle. An Euler step moves along the straight line the rule points along, and a straight line that touches a circle leaves it on the outside. Every step lands a little outside, and by Pythagoras exactly so: each step multiplies the energy by $1 + \text{step} \times \text{step}$, which is 1.01 for a step of 0.1. One per cent a step, 100 times in ten seconds.

Stop and think 4.3

Take steps ten times smaller, 0.01 seconds. Is the simulation now right?

Answer it before you read on.

Answer. Right for longer, not right. Over ten seconds the energy gains 10.5 per cent instead of 170; after a hundred seconds it has gained 172 per cent, about what the larger step gained in ten.

Each small step gains a hundred times less, and there are ten times as many of them each second. Ten times smaller steps bought ten times as long, for ten times the work.

The trap

‘A smaller step makes the simulation right.’ It makes it right for longer. The error of a stepping method is made fresh at every step and piles up, so a smaller step buys time before the pile grows too large. The honest question to ask of a simulation is not how small its step was but how long it has been checked for. Part II meets a harsher horizon, one that no step size can move.

4.3 Look ahead, and average

Euler asks the rule once, at the start of the step, and trusts the answer for the whole step. The rates change during the step, and Euler never finds out. The stepper this book uses, worked out by Carl Runge and Martin Kutta at the turn of the twentieth century and known as RK4, asks four times:

1. the rates at the start;
2. the rates half a step ahead, going at the first rates;
3. the rates half a step ahead, going at the second rates;
4. the rates a whole step ahead, going at the third.

It then takes the real step at an average of the four, counting the middle two twice.

code/src/chaoslab/flows.py

```

34 def rk4_step(field: Field, state: State, dt: float) -> State:
35     """One Runge-Kutta step: sample the velocity four times, average them.
36
37     k1 is the velocity at the start; k2 and k3 are velocities half a step
38     ahead; k4 is a full step ahead. The weights 1, 2, 2, 1 are what make the
39     error in the position shrink sixteen-fold when dt is halved, where
40     Euler's only halves. (The slow drift in a spring's energy shrinks faster
41     still, about thirty-two-fold; Chapter 4 measures it.)
42     """
43     k1 = field(state)
44     k2 = field(_add(state, k1, dt / 2))
45     k3 = field(_add(state, k2, dt / 2))
46     k4 = field(_add(state, k3, dt))
47     return tuple(
48         s + dt * (a + 2 * b + 2 * c + d) / 6
49         for s, a, b, c, d in zip(state, k1, k2, k3, k4, strict=True)
50     )

```

Listing 4.2: The RK4 recipe, from the book's library

field is the rule of change, k1 to k4 are the four samples, and _add works out where you would be after moving for a while at given rates. The weights 1, 2, 2, 1 are chosen so that the errors of the samples cancel as far as they can; the algebra is skipped here. The same function, with `euler_step` and `integrate` beside it, carries the weather model of Chapter 9.

Lab 4.1 — One step of RK4

Write `rk4_step` from the recipe. The test checks it against the library on a spring and a pendulum, and on the rule ‘ x grows at a rate equal to x ’: one step of length 1 from 1 must land on $1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24}$, close to the true answer $e \approx 2.718$.

Starter: code/labs/ch04/l04_01_rk4.py

Test: code/labs/ch04/test_l04_01_rk4.py

Run: `cd code && uv run pytest -k l04_01_rk4`

Stop and think 4.4

RK4 asks the rule four times a step, so for the same work Euler can take steps four times smaller. Which would you rather have: Euler with steps of 0.025 seconds, or RK4 with steps of 0.1?

Answer it before you read on.

Answer. RK4, by a distance. Over ten seconds Euler gains 28 per cent of its energy; RK4 loses 1.4 parts in a million. For the same work its error is about 200 000 times smaller.

code/ch04/energy_drift.py

```

12 FIELD = spring(1.0)          # the rule of change from the chapter
13 START = (1.0, 0.0)          # pulled out 1 metre, at rest
14
15
16 def energy(state: tuple[float, ...]) -> float:
17     x, v = state
18     return 0.5 * v * v + 0.5 * x * x
19
20
21 def drift(stepper, dt: float, seconds: float) -> float:
22     """Fractional change in energy after following the rule for seconds."""
23     state = START
24     for _ in range(round(seconds / dt)):
25         state = stepper(FIELD, state, dt)
26     return energy(state) / energy(START) - 1.0

```

Listing 4.3: Measuring how far a simulated spring's energy drifts

method	step	seconds	rule calls	energy change
Euler	0.100	10	100	+170 %
Euler	0.010	10	1000	+10.5 %
Euler	0.010	100	10000	+172 %
Euler	0.025	10	400	+28.4 %
RK4	0.100	10	400	-0.000139 %
RK4	0.100	1000	40000	-0.0139 %

rule calls counts the work. The last line says that RK4 is not exact either: over a thousand seconds it loses 139 parts in a million. It too is right for longer rather than right, only very much longer.

Lab 4.2 — Measure the drift

Write the spring's energy, and a function that follows the spring with a given stepper for a given time and reports how much its energy has changed. Then answer a question the chapter leaves open: halve the step, and how many times smaller does each method's drift in energy become? (The comment in the RK4 listing is about the error in position, a different measure.) Predict before you run.

Starter: code/labs/ch04/l04_02_drift.py

Test: code/labs/ch04/test_l04_02_drift.py

Run: cd code && uv run pytest -k l04_02_drift

4.4 The map of every state

The left half of Figure 4.2 plots nothing against time. Each point in it is a whole state, a position and a velocity together, and the motion is a path from point to point. A plane of every possible state is a *phase space*, and a drawing of the paths in it is a *phase portrait*. Every chaotic system in this book is drawn in one.

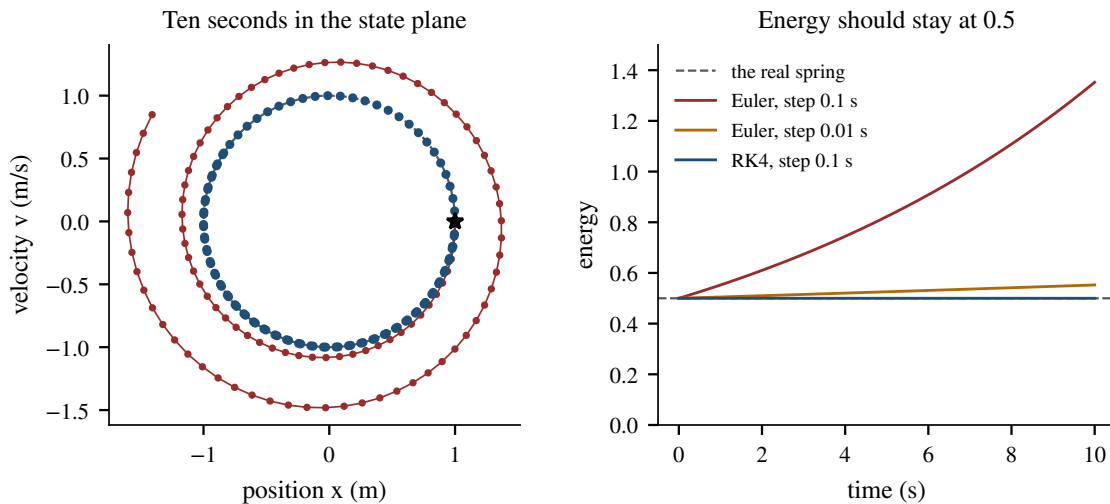


Figure 4.2: The same spring followed for ten seconds by two steppers with the same step. Left: in the plane of states, Euler (red) spirals outwards off the real circle while RK4 (blue) stays on it. Right: the energy each one reports.

A pendulum is the richer example. Its state is the angle from hanging straight down and how fast that angle is changing. Its rule is the spring's with one difference: the pull back is largest when the pendulum sticks out sideways and falls to nothing when it points straight down or straight up. (That is the sine, which chaoslab's pendulum rule calls.)

Stop and think 4.5

Released further out, a pendulum has further to go but moves faster. Does one released at 90 degrees take longer to swing out and back than one released at 10°?

Answer it before you read on.

Answer. Longer: 2.368 seconds against 2.010, about 18 per cent more.
Released at 170 degrees it takes 4.894 seconds, 2.4 times as long.

code/ch04/pendulum.py

```

15 G_OVER_L = 9.81          # gravity over length: a one-metre pendulum
16 FIELD = pendulum(G_OVER_L) # (angle, angular velocity) -> their rates
17
18
19 def period(swing_degrees: float, dt: float = 0.001) -> float:
20     """Seconds for one full swing, out and back, released at rest."""
21     state = (math.radians(swing_degrees), 0.0)
22     steps = 0
23     while True:
24         new = rk4_step(FIELD, state, dt)
25         if new[1] > 0.0: # it has stopped at the far side, turned round
26             share = -state[1] / (new[1] - state[1])
27             return 2 * (steps + share) * dt # out and back is twice this
28     state, steps = new, steps + 1

```

Listing 4.4: Timing a pendulum's swing

released at	one swing out and back
5 deg	2.007 s
10 deg	2.010 s
45 deg	2.086 s
90 deg	2.368 s
170 deg	4.894 s

The school rule that the swing time does not depend on the swing is right for small swings: the times settle on 2.006 seconds as the swing shrinks. That is why a precision pendulum clock swings through only a few degrees.

Now draw the rule as a map. At every point of the plane the rule gives the rates, and so the direction the state is about to move in. Put a small arrow there. Motion is following the arrows.

Stop and think 4.6

Somewhere on the map the arrow has no length: nothing is changing. What are those states, as positions of a real pendulum? What happens if you nudge each?

Answer it before you read on.

Answer. Hanging straight down at rest, and balanced exactly upside down at rest. Nudged, the first swings gently around; the second falls away. They are the two fixed points, one stable and one unstable, as in Chapter 2.

Near the middle the paths are closed loops: the pendulum swings out and back, round the same loop for ever. A path that closes on itself is a *cycle*, and a cycle is perfectly predictable: once round, and you know every later lap. Far above and below, the paths run the whole width: the pendulum goes over the top and keeps turning.

4.5 Settle, cycle or run off

A real pendulum has friction, which takes a little energy out of every swing.

Stop and think 4.7

On the map, what happens to the path of a pendulum with friction released at 150 degrees, and where does it end?

Answer it before you read on.

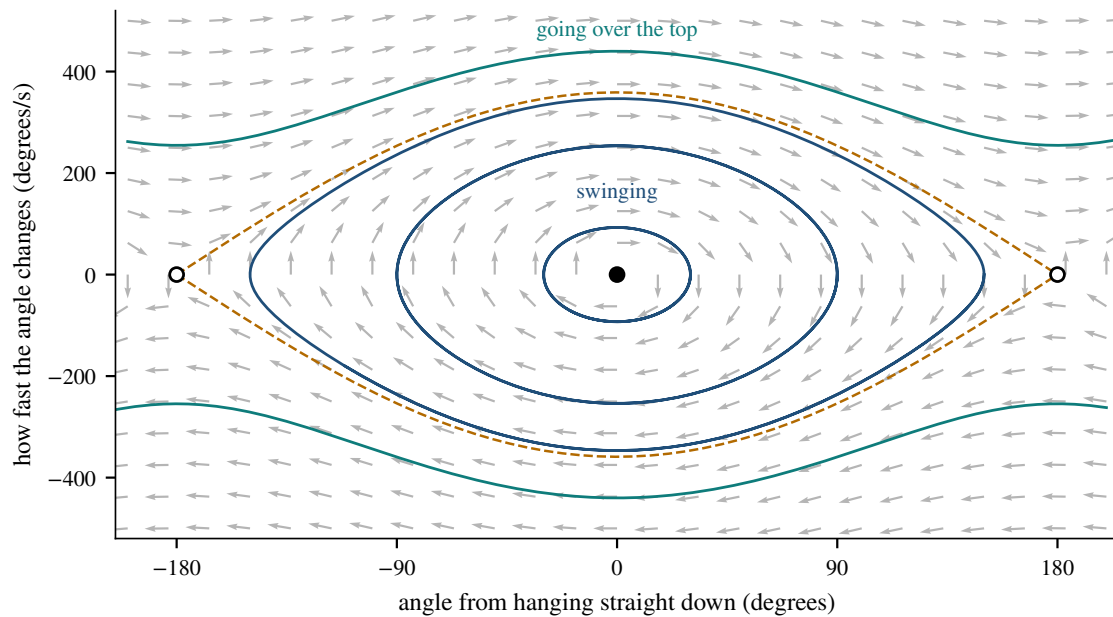


Figure 4.3: The map of every state of a pendulum one metre long. Grey arrows show which way the state moves. Blue loops are swings released at 30, 90 and 150 degrees; green paths go over the top; the dashed curve separates the two. The filled dot is hanging at rest; the two open circles are the same upside-down position.

Answer. Each swing is smaller than the last, so the path cannot close. It spirals inwards and ends on the filled dot: hanging straight down, at rest.

A state that nearby paths all run into is an *attractor*, and the damped pendulum's resting point is the simplest kind. A loop can attract too. The Dutch engineer Balthasar van der Pol wrote down the classic example about a century ago, for circuits built with radio valves:

$$x \text{ changes at the rate } v, \quad v \text{ changes at the rate } -x + (1 - x \times x) v.$$

While x lies between -1 and 1 the new term speeds the motion up; beyond them it brakes. Start near the middle, at 0.1 , and the swing grows; start far out, at 4 , and it shrinks. After 60 seconds both swing out to 2.01 and 2.01 : one loop, reached from inside and from outside. It is a *limit cycle*, and a pendulum clock is one: a small kick every swing, friction taking energy away, and a swing that settles to one size however you started it.

Stop and think 4.8

Why can two different paths on one of these maps never cross?

Answer it before you read on.

Answer. At a crossing the state would have two futures, but the rule gives one arrow at every point: one state, one future.

In a plane that one fact is strong. A path that may cross neither itself nor any other path,

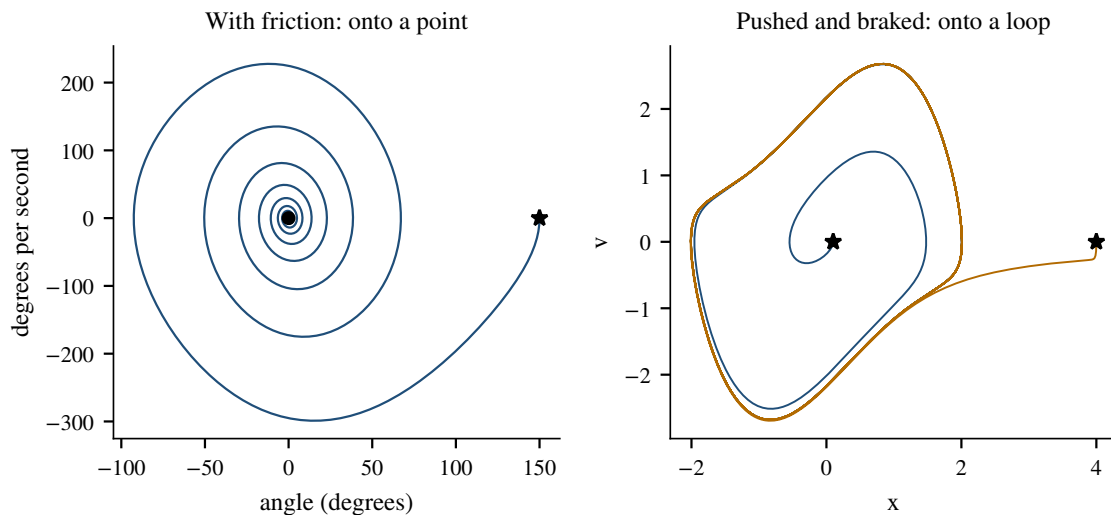


Figure 4.4: Two ways for motion in a plane to settle. Left: a pendulum with friction, released at 150 degrees, spirals onto the point where it hangs at rest. Right: van der Pol's rule from a start inside and a start outside, winding onto the same loop.

and stays in a bounded region, is fenced in by its own past. It can wind towards a fixed point or towards a loop, or it can run off to infinity. It cannot wander for ever without settling.

What we are not proving

This is the *Poincaré–Bendixson theorem*, after Henri Poincaré and the Swedish mathematician Ivar Bendixson. It needs a smooth rule, and its exact statement allows one more ending, a chain of fixed points joined by paths. Its proof takes several pages of a university course and is in any textbook of dynamical systems. It is about motion in continuous time: a rule that jumps in steps can hop over its own path, and Chapter 5 finds wild behaviour in a stepping rule with one number. And it needs a plane: in three dimensions a path can pass over and under itself.

So chaos in continuous time needs at least three numbers of state. That is why the weather model of Chapter 9, which Edward Lorenz cut down in 1963 from a larger model of convection, has three numbers and not two.

Stop and think 4.9

Hang a second pendulum from the bottom of the first. How many numbers is the state now, and does the theorem still promise simple motion?

Answer it before you read on.

Answer. Four: an angle and a speed for each pendulum. The theorem says nothing about four numbers, and Chapter 13 shows that the double pendulum's motion is chaotic.

The trap

‘A pendulum is a simple system, so its motion must be simple.’ For one pendulum it is: two numbers of state, a plane, and a theorem that allows only settling, cycling or running off. Hang a second from it and the state has four numbers, the fence is gone, and the motion can become chaotic. How simple a system looks says little about how simple its motion is. Counting the numbers in its state says more.

4.6 Two bodies: the clockwork at its best

Four numbers of state make chaos possible; they do not make it happen. The best evidence is the system that built the clockwork view: one planet going round the sun. Newton’s gravity is a rule of change. The planet’s position changes at its velocity, and its velocity changes towards the sun by an amount that falls with the square of the distance: twice as far, a quarter of the pull. The sun is so much heavier that it can sit still.

code/ch04/two_bodies.py

```

17 def gravity(s: tuple[float, ...]) -> tuple[float, ...]:
18     """Position changes at the velocity; velocity changes towards the sun,
19     by one over the distance squared."""
20     x, y, vx, vy = s
21     r = math.sqrt(x * x + y * y)
22     pull = 1.0 / (r * r * r)          # 1/r^2, and x/r, y/r point home
23     return (vx, vy, -pull * x, -pull * y)

```

Listing 4.5: Newton’s gravity as a rule of change

The state is four numbers: where the planet is, across and up, and how fast it is going, across and up. The listing launches it sideways at 3 different speeds, follows each orbit ten times round with RK4, and times every return to the starting line. An orbit’s size is half its longest width.

launch	first lap	average of 10	size	lap^2/size^3
1.0	6.283185	6.283185	1.0000	39.4784
1.2	14.993321	14.993321	1.7857	39.4784
1.3	36.403013	36.403013	3.2258	39.4784

The average of ten laps is the first lap, to every digit printed: the planet comes back round its ellipse on schedule, again and again. The last column is the same for all three orbits, 39.48: the square of the lap time divided by the cube of the size. Johannes Kepler found that rule in the planets’ positions in 1619 and Newton showed in 1687 that it follows from gravity; your laptop has found it again from the rule of change alone.

Why no chaos with four numbers? Two quantities never change: the energy, and the rate at which the line from sun to planet sweeps out area. Each pins the state down further, and with two pinned the motion has no room left to be anything but regular. Add a third body and that stops being true, which is where Chapter 13 begins.

Where you meet this

The tables of where the planets, large moons and many asteroids will be, which spacecraft navigators steer by, are published by NASA's Jet Propulsion Laboratory. They are made the way this listing made its orbits: Newton's rule of change for the whole solar system, followed forward in small steps and checked against radar and spacecraft tracking.

That is the clockwork at its best: a rule of change, small steps, and a prediction that comes back on schedule to every digit it prints. Part II asks what happens when the rule is every bit as exact and the future is not.

Summary

- A **rule of change**, or **differential equation**, says how fast each number of the state is changing right now, not what it will be next.
- **Euler's method** moves each number for one short step at the rate it has now. On a spring it gains energy every step, and a smaller step makes it **right for longer**, not right.
- **RK4** samples the rates four times a step and averages them; for the same work its error is far smaller, but it drifts too. Energy is the check.
- A **phase space** is a plane of every possible state; the rule is an arrow at every point. A pendulum draws **cycles** around its stable **fixed point**, and big swings take longer than small ones.
- Friction spirals a path into an **attractor**; a loop can attract too. Paths never cross, so in a plane motion can only settle, cycle or run off (**Poincaré–Bendixson**): chaos in continuous time needs at least three numbers.
- Two bodies have four numbers of state and still trace ellipses on schedule, as Kepler's third law says: three numbers are needed for chaos, not enough.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

read a rule of change as a statement about how fast each number of the state is changing right now	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
turn a rule of change into a path in code, with Euler's method and with RK4	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
use energy to check whether a simulation is telling the truth	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
read a phase portrait: its fixed points, its loops and which way the motion goes	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what motion in a plane can and cannot do, and how many numbers chaos in continuous time needs	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. A speedometer reads 90 kilometres an hour. Roughly how far does the car go in the next 20 seconds?
2. A spring with $k = 1$ is at $x = 0$ moving at $v = 1$, so its energy is 0.5. Take one Euler step of 0.1. What are the new x , v and energy?
3. Euler with a step of 0.1 gains 170 per cent of the spring's energy in ten seconds. Roughly how long can a run with a step of 0.001 go before it has gained as much?
4. Describe the four samples RK4 takes in one step, and how it combines them.
5. On a pendulum's phase portrait, what do a closed loop, a path across the whole width and a spiral each mean for the real pendulum?
6. Why can a smooth rule of change with two numbers of state never be chaotic, and how many numbers does chaos in continuous time need?

Part II

Where prediction breaks

One line of algebra, every kind of behaviour

Here is a rule you could teach a child. Take a number between 0 and 1. Multiply it by one minus itself, then multiply the answer by a fixed number r . That is the next number. Do the same to it, and again, and again. No dice, no measurement, no outside world: each step is fixed exactly by the one before.

Make a guess before you read on. If you turn the rule's one knob, r , slowly upwards, how many different kinds of behaviour do you expect: one, two, a handful? This chapter turns the knob and watches. By the end you will have seen a list of numbers that never settles and never repeats, looks as if it came off a roulette wheel, and was produced by nothing but that line.

What you will be able to do

When you have finished this chapter you will be able to:

- iterate the logistic map by hand for a step or two, and in Python for as many steps as you like
- find the map's steady state for a given r , and say from the slope there whether it holds
- draw a cobweb diagram on a curved rule and read from it what an orbit will do
- tell a steady state, a cycle and a never-repeating orbit apart, and say what a program that looks for cycles can and cannot conclude
- say what *deterministic* and *chaotic* mean, and why looking random is not the same as being random

5.1 The rule and its one knob

Chapter 3 ended with a population whose growth slows near its limit: at a gentle rate it settled, and at a faster one it overshot and swung. This chapter writes a rule of that kind in the form that made it famous. Let x be the population as a fraction of the largest it can be, a number between 0 and 1. Write x_n , read ' x sub n ', for its value after n steps, so that x_0 is the start. The rule is

$$x_{n+1} = r x_n (1 - x_n).$$

In words: the next value is r times the present value times one minus the present value. The factor x_n says that more parents make more young; the factor $1 - x_n$ is the feedback, the room left, which shrinks as the population grows. The number r is the growth knob. A rule that takes a number to the next number is called a *map*, and this one is the *logistic*

map. Beside the rules of Chapter 2, $x_{n+1} = a x_n + b$, it has one extra multiplication, by $1 - x_n$, and everything in this chapter comes from that one factor.

The product $x(1 - x)$ is largest at $x = \frac{1}{2}$, where it is $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$, so the next value is never more than $r/4$. For r up to 4, then, x never leaves the range from 0 to 1.

Stop and think 5.1

Take $r = 2.8$ and start at $x_0 = 0.5$. What is x_1 ? And x_2 ? The first you can do in your head; the second needs a pencil at most.

Answer it before you read on.

Answer. $x_1 = 2.8 \times 0.5 \times 0.5 = 0.7$, and $x_2 = 2.8 \times 0.7 \times 0.3 = 0.588$.

That is all there is to it, and a computer does millions of steps a second. Here is the rule as the book's code writes it.

code/ch05/four_rates.py

```
11 def step(r: float, x: float) -> float:
12     """One step of the logistic map: the next value from this one."""
13     return r * x * (1.0 - x)
```

Listing 5.1: The logistic map: one line of arithmetic

Multiplication and subtraction are rounded in exactly the same way by every computer, so the numbers in this chapter come out the same to the last digit on any machine. For a rule whose whole interest, as you are about to see, lies in its digits, that matters.

5.2 Turning the knob

Start every run at $x_0 = 0.2$. Let the rule take 200 steps without looking, so that the *orbit* — the list of values a rule produces from a start — has time to settle into whatever it will do, then write down the next eight values. Do that at four settings of the knob: $r = 2.8$, 3.2, 3.5 and 3.9.

Stop and think 5.2

Chapter 3 found that a gentle growth rate settles and a faster one overshoots and swings. These four settings run from moderate to fast. After 200 steps, what do you expect at $r = 3.9$: one value repeated, a swing between a few values, or something else?

Answer it before you read on.

Answer. Something else. At $r = 3.9$ the values never settle, and never fall into a pattern.

code/ch05/four_rates.py

```

18 def after(r: float, x0: float, skip: int, keep: int) -> list[float]:
19     """Take `skip` steps from x0 without looking, then keep `keep` values."""
20     x = x0
21     for _ in range(skip):
22         x = step(r, x)
23     values = []
24     for _ in range(keep):
25         values.append(x)
26         x = step(r, x)
27     return values
28
29
30 def main() -> None:
31     print("after 200 steps from 0.2, the next eight values:")
32     for r in (2.8, 3.2, 3.5, 3.9):
33         values = after(r, 0.2, 200, 8)
34         print(f"r = {r}: " + " ".join(f"{v:.3f}" for v in values))

```

Listing 5.2: Two hundred steps without looking, then eight more, at four settings

```

after 200 steps from 0.2, the next eight values:
r = 2.8: 0.643 0.643 0.643 0.643 0.643 0.643 0.643 0.643
r = 3.2: 0.799 0.513 0.799 0.513 0.799 0.513 0.799 0.513
r = 3.5: 0.827 0.501 0.875 0.383 0.827 0.501 0.875 0.383
r = 3.9: 0.400 0.936 0.233 0.697 0.824 0.566 0.958 0.157

```

Each row is a different kind of behaviour. At $r = 2.8$ the population has come to rest at 0.643: a *steady state*. At $r = 3.2$ it alternates for ever between 0.513 and 0.799, high year, low year: a *cycle of period two*. At $r = 3.5$ it goes round four values, from 0.383 to 0.875, and round the same four again: a cycle of period four. At $r = 3.9$ the numbers jump about with no pattern you can see, and, as you will check below, none turns up however long you wait.

In Figure 5.1 every panel starts with a scramble, and three of the four soon find their rhythm. The fourth never does.

The trap

‘A simple rule gives simple behaviour.’ All four panels come from the same line of arithmetic; only r changed, from 2.8 to 3.9. The behaviour went from one resting value to numbers that never repeat. How complicated a system’s behaviour looks tells you very little about how complicated its rule is.

5.3 The steady state, and when it lets go

A *fixed point* is a value the rule leaves unchanged, and a steady state is a fixed point the orbit has settled on. The logistic map always has one at $x = 0$: no population stays no population. For the other, ask when $r x (1 - x)$ equals x . Divide both sides by x to get

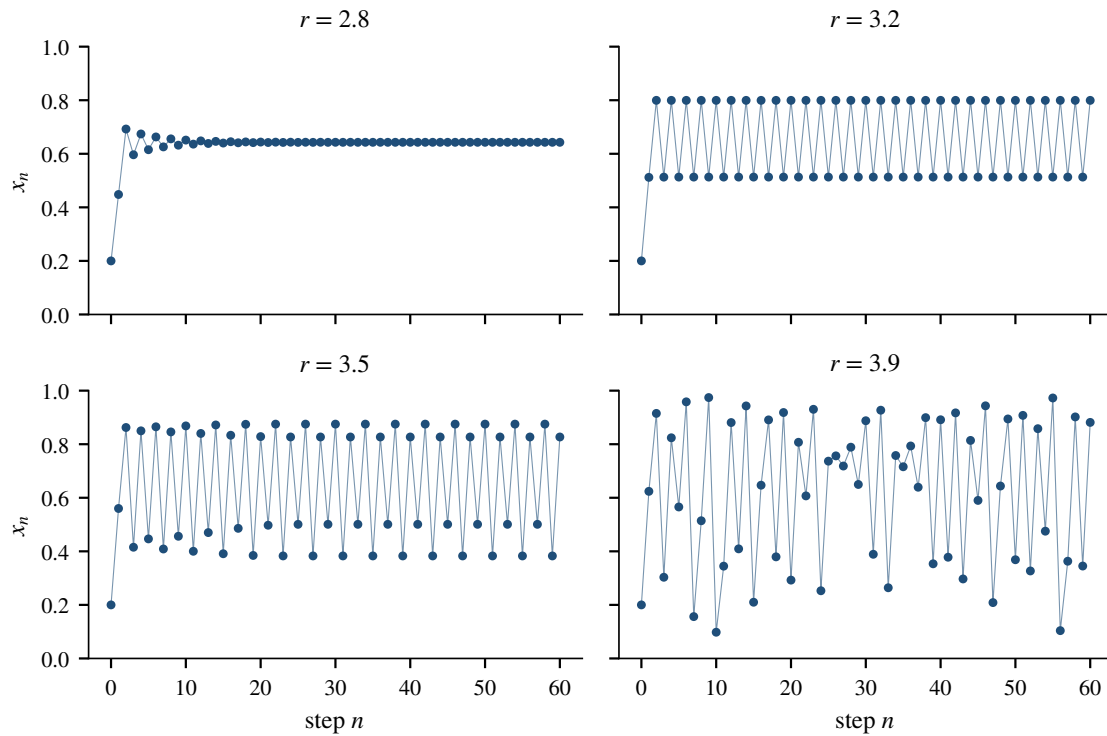


Figure 5.1: The four settings over the first sixty steps, all from $x_0 = 0.2$. Look at how each panel ends: resting, alternating, going round four values, and still jumping.

$r(1 - x) = 1$, so $1 - x = 1/r$, and

$$x^* = 1 - \frac{1}{r}.$$

The star is the usual mark for a fixed point.

Stop and think 5.3

For $r = 2$ the formula gives $x^* = \frac{1}{2}$. Check that the rule really leaves $\frac{1}{2}$ unchanged. What does the formula give for $r = 4$, and does that check out too?

Answer it before you read on.

Answer. $2 \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{2}$. For $r = 4$ it gives $\frac{3}{4}$, and $4 \times \frac{3}{4} \times \frac{1}{4} = \frac{3}{4}$.

At $r = 2.8$ the formula gives 0.643, exactly where the printout came to rest. At $r = 3.2$ it gives 0.688 — and the orbit swings round it instead, from 0.513 to 0.799. The fixed point exists; the orbit will not stay on it.

Chapter 3 explained why with a measurement. Nudge the state a tiny amount off the fixed point and take one step: the nudge comes back multiplied by the *slope* of the rule there. Between -1 and 1 , nudges shrink and the fixed point holds; beyond, they grow and the orbit is pushed away. A negative slope also flips the nudge to the other side at every step. The library's `slope` measures it by nudging x a millionth each way.

code/ch05/fixed_point.py

```

10 from chaoslab import logistic, slope
11
12
13 def fixed_point(r: float) -> float:
14     """The value, other than zero, that the rule leaves unchanged."""
15     return 1.0 - 1.0 / r
16
17
18 def main() -> None:
19     print(" r   fixed point   slope there   a nudge")
20     for r in (2.8, 3.2, 3.5, 3.9):
21         x = fixed_point(r)
22         s = slope(logistic(r), x)      # nudge x both ways, measure the move
23         fate = "shrinks" if abs(s) < 1 else "grows"
24         print(f"{r:.1f}   {x:11.3f}   {s:11.2f}   {fate}")

```

Listing 5.3: The fixed point, and the slope of the rule there

r	fixed point	slope there	a nudge
2.8	0.643	-0.80	shrinks
3.2	0.688	-1.20	grows
3.5	0.714	-1.50	grows
3.9	0.744	-1.90	grows

Each slope is $2 - r$: $2 - 2.8 = -0.8$, $2 - 3.2 = -1.2$, and so on. (With calculus, the slope of $r x (1 - x)$ is $r (1 - 2x)$, and at $x = 1 - 1/r$ that is exactly $2 - r$.) At $r = 2.8$ a nudge flips and shrinks at every step, and the orbit closes in. At 3.2 it flips and grows, and the orbit is thrown off.

Stop and think 5.4

As you turn r up from 2.8, at what value does the slope $2 - r$ reach -1 , so that the fixed point stops holding?

Answer it before you read on.

Answer. At $r = 3$, where $2 - 3 = -1$. Below 3 the steady state holds; above 3 it lets go.

That is the first place where the knob changes the kind of behaviour, and you found it by hand. The change from two values to four comes further up; Chapter 8 finds where, and a pattern in where.

Lab 5.1 — Where the steady state lets go

Write `fixed_point`; `slope_at`, which measures the slope at a point by nudging it each way; and `holds`, which says whether a nudge at the fixed point shrinks. Then write `lets_go`, which finds where `holds` turns false between 2.5 and 3.5 by halving the

interval again and again. It should agree with the answer above.

Starter: code/labs/ch05/l05_01_stability.py

Test: code/labs/ch05/test_l05_01_stability.py

Run: `cd code && uv run pytest -k l05_01_stability`

5.4 The cobweb on a curve

Chapter 2 drew iteration as a cobweb on a straight line. The same drawing works for any rule. Draw the rule's graph, the height $r x (1 - x)$ for every x from 0 to 1: a hump, highest at $x = \frac{1}{2}$. Draw the diagonal, where the height equals x . Start at x_0 on the bottom edge and go up to the hump: the height you reach is x_1 . Go across to the diagonal, which puts you above x_1 ; up or down to the hump for x_2 ; across again; and so on. The hump crosses the diagonal at the fixed point, where the next value equals the present one. The library's `cobweb` lists the corners of the path.

code/ch05/cobweb_data.py

```
10 from chaoslab import cobweb, logistic
11
12
13 def main() -> None:
14     corners = cobweb(logistic(2.8), 0.2, 3)          # three steps from 0.2
15     moves = ["start"] + ["up", "across"] * 3
16     for move, (x, y) in zip(moves, corners, strict=True):
17         print(f"{move:<7} x = {x:.3f} height = {y:.3f}")
```

Listing 5.4: The corners of a cobweb, three steps long

start	x = 0.200	height = 0.000
up	x = 0.200	height = 0.448
across	x = 0.448	height = 0.448
up	x = 0.448	height = 0.692
across	x = 0.692	height = 0.692
up	x = 0.692	height = 0.596
across	x = 0.596	height = 0.596

Each up lands on the hump at the next value of the orbit, $2.8 \times 0.2 \times 0.8 = 0.448$ for the first; each across lands on the diagonal at the same height.

Stop and think 5.5

At $r = 2.8$ the cobweb spirals in to the crossing. At $r = 3.2$ the orbit ends up alternating between two values for ever. What shape does its cobweb settle into?

Answer it before you read on.

Answer. A closed rectangle with two corners on the hump: up to one value, across, down to the other, across, and back to the start.

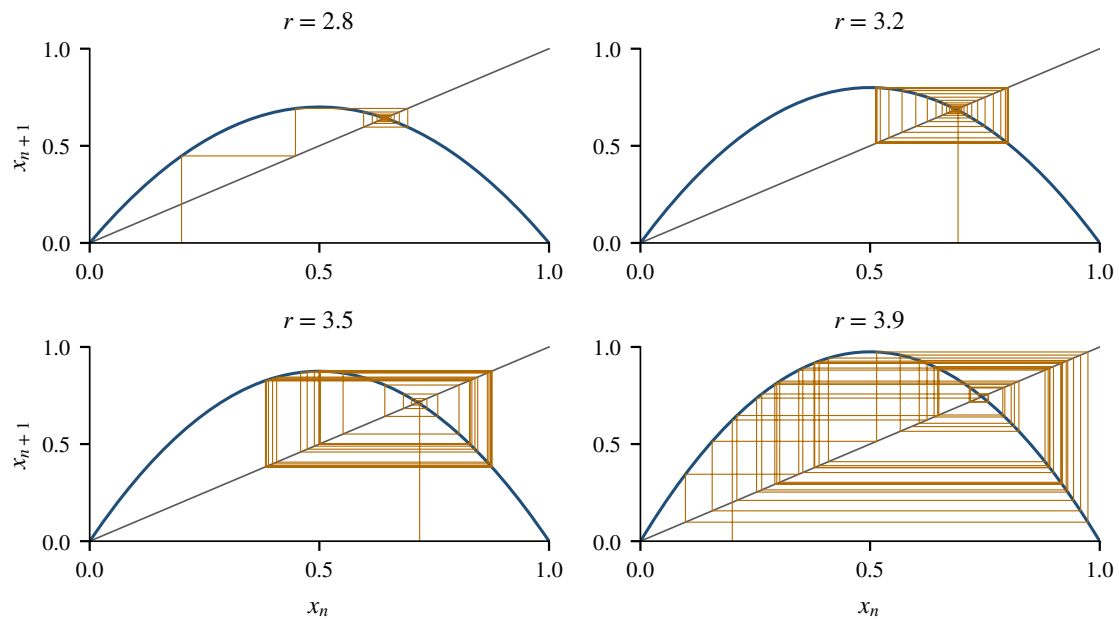


Figure 5.2: Cobwebs for the four settings, each on its own hump with the diagonal. At $r = 2.8$ the path spirals in; at 3.2 and 3.5 it starts right beside the crossing, is pushed away and settles on a closed loop; at 3.9 it never closes.

Look at where the hump crosses the diagonal in Figure 5.2. At $r = 2.8$ it falls through the crossing less steeply than the diagonal climbs, and the spiral tightens. At 3.2 it falls more steeply, and a path started beside the crossing spirals outwards until it is caught on the rectangle. A curve falling exactly as steeply as the diagonal climbs has slope -1 : the drawing and the measurement agree. At 3.5 the loop has four corners on the hump. At 3.9 the path never closes, and slowly fills a whole band.

5.5 Counting the cycle, and failing to

For thousands of numbers, let the computer read the cycle. The library's `period` looks at the last 128 values of an orbit for the smallest whole number p such that every value equals the one p steps later, to within a billionth. It returns that p , or Python's word for nothing, `None`, when no p up to 64 fits.

code/ch05/periods.py

```

11 from chaoslab import logistic, orbit, period
12
13
14 def main() -> None:
15     for r in (2.8, 3.2, 3.5, 3.9, 4.0):
16         xs = orbit(logistic(r), 0.2, 2000) # the start and 2000 steps
17         print(f"r = {r}: period {period(xs)}")

```

Listing 5.5: Asking for the period after two thousand steps

```

r = 2.8: period 1
r = 3.2: period 2

```

```

r = 3.5:  period 4
r = 3.9:  period None
r = 4.0:  period None

```

The detector agrees with the printout: 1, then 2 and 4. At $r = 3.9$, and at $r = 4$, the top of the range, it finds nothing.

Stop and think 5.6

At $r = 3.9$ the detector found no cycle up to 64 steps long at the end of two thousand steps. Suppose you ran the rule for a hundred thousand steps instead. Would a repeat turn up eventually?

Answer it before you read on.

Answer. No. In a hundred thousand steps not a single value comes back.

code/ch05/never_repeats.py

```

11 from chaoslab import logistic, orbit
12
13
14 def main() -> None:
15     for r in (3.9, 4.0):
16         xs = orbit(logistic(r), 0.2, 100_000)
17         again = orbit(logistic(r), 0.2, 100_000)
18         print(f"r = {r}: {len(xs)} values, {len(set(xs))} different")
19         print(f"         same start again, same list: {xs == again}")

```

Listing 5.6: A hundred thousand steps, looking for any value seen before

```

r = 3.9: 100001 values, 100001 different
         same start again, same list: True
r = 4.0: 100001 values, 100001 different
         same start again, same list: True

```

A Python set keeps one copy of each different value, so a value that came back would make the second count smaller. It does not: all 100 001 values are different. That settles the question for these 100 000 steps, because if a value ever did come back, the one after it would too — the same number always produces the same next number — and so would everything after that: a cycle. No value returning means no cycle has begun. The second line of each pair matters later: the same start, run again, gives the same numbers exactly.

The trap

‘If it has not repeated after a thousand steps, it is about to.’ A long wait makes a repeat feel overdue, as a coin that has shown heads five times seems to owe you tails. The orbit owes you nothing. No cycle is pulling it in, and running longer only produces more numbers that have never appeared before.

What we are not proving

A run is evidence, not proof. For $r = 4$ it can be proved that from almost every start the orbit never repeats; for $r = 3.9$ the evidence is computation like this. And a computer holds only finitely many numbers between 0 and 1, so a computed orbit must eventually meet a value it has had before, and cycle — after more steps than this run took. Chapter 16 says what that means for a simulation.

Lab 5.2 — A cycle detector of your own

Write `settle`, which runs the rule without looking and then keeps the next few values, and `period`, which finds the smallest p for which every value in a list equals the one p places on, and returns `None` when no p up to a limit fits. The tests use made-up lists and this section's orbits.

Starter: `code/labs/ch05/l05_02_period.py`
 Test: `code/labs/ch05/test_l05_02_period.py`
 Run: `cd code && uv run pytest -k l05_02_period`

Stop and think 5.7

The detector found a cycle of four at $r = 3.5$ and nothing at 3.9. What do you expect it to find at $r = 3.83$, a little below 3.9?

Answer it before you read on.

Answer. A cycle of period three. After a short scramble the orbit goes round 0.156, 0.505 and 0.957 for ever.

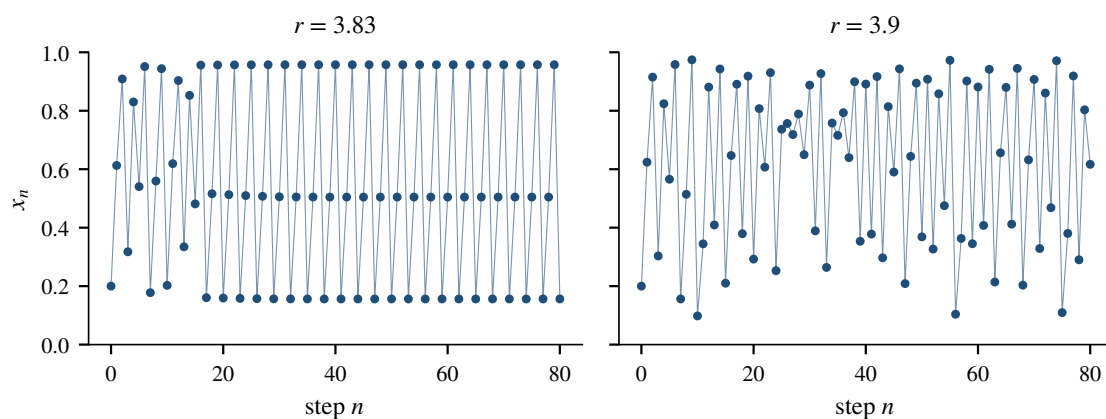


Figure 5.3: Order inside the jumping. At $r = 3.83$ the orbit scrambles for a while, then locks onto a cycle of three; at $r = 3.9$, close by, it never settles.

This surprises almost everyone, and it should. Turning the knob up does not make the behaviour steadily wilder: among the settings where orbits jump without pattern there are narrow windows where order comes back. Where they sit, and what a cycle of three

implies, belong to the whole picture that Chapter 8 draws.

5.6 Random-looking, and not random

Now the names. A rule is *deterministic* when the present fixes the next state completely, with nothing left to chance, so the same start always gives the same orbit. You watched the logistic map be so: run twice from 0.2, it gave the same hundred thousand numbers to the last digit. An orbit is *chaotic* when it comes from a deterministic rule, stays within bounds, never settles into rest or a cycle — and, the part not yet measured, is so sensitive to its start that any difference there, however small, eventually grows until the two futures have nothing in common.

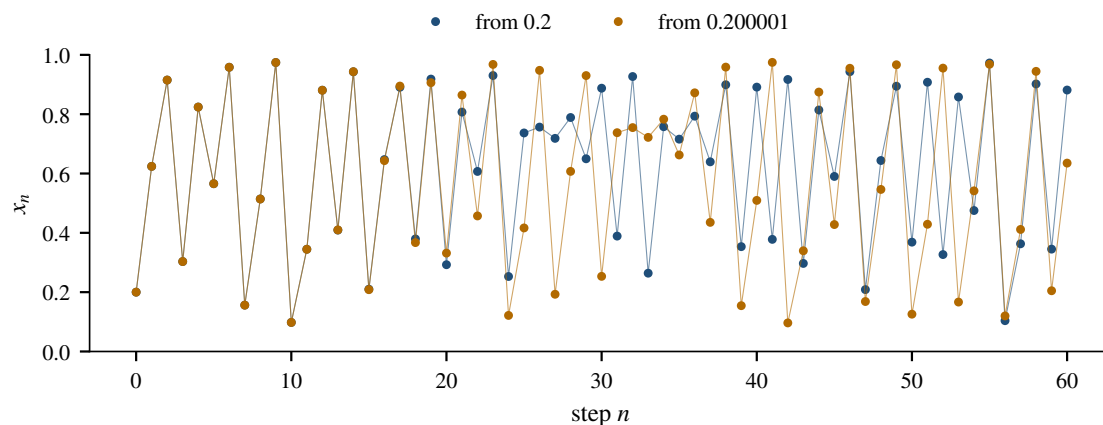


Figure 5.4: Two runs at $r = 3.9$, from 0.2 and from 0.200001, a millionth higher. For a while the lines lie on top of each other; then they part for good.

That last part is Chapter 6's measurement; Figure 5.4 is only a preview. Together they make the strangest fact in this book so far. The orbit at $r = 3.9$ looks random, and is not: every number in it was fixed exactly by the one before. Yet you cannot foretell it far ahead without knowing its start more precisely than anyone knows anything.

In 1976 Robert May, a physicist who had moved into ecology, published a review in *Nature* whose title says what this chapter has shown: *Simple mathematical models with very complicated dynamics*. He wrote it for biologists.

Stop and think 5.8

An ecologist counts the insects in a field every summer for twenty years. The counts jump up and down with no pattern anyone can find. What does that tell you about the cause?

Answer it before you read on.

Answer. Less than it seems. The weather might do it, or errors in counting — or nothing but the population's own rule of growth.

That third possibility was May's point. Irregular numbers from the field were usually read

as the mark of a noisy environment or of imperfect counting; the logistic map shows that one generation a year and one simple rule of growth can produce the same irregularity with no disturbance at all. May ended with a plea that reaches beyond ecology: that students should meet this equation early, because an intuition trained only on straight-line rules leaves people unprepared for what curved ones do. You have now met it.

The trap

‘Irregular output needs a random or complicated cause.’ The $r = 3.9$ row in Section 5.2 came from the same line of arithmetic as the rows above it, with no randomness anywhere. Irregular output can come from chance, from a complicated mechanism, or from a simple deterministic rule, and the output alone does not say which. Chapter 15 teaches how to tell them apart.

Where you meet this

Fisheries science meets the same behaviour in a different formula. William Ricker’s stock-and-recruitment model of 1954, which relates the fish spawning in one year to the young they produce, is a one-line rule of the same kind: growth that slows as numbers rise, a generation at a time. Turn up its growth rate and it passes, in the same order as the logistic map, from a steady stock through alternating booms and busts to irregular swings. A catch that swings from year to year does not, by itself, prove that the weather did it.

One line, one knob: a steady state, cycles of two and four, orbits that never repeat, and islands of order among them. What is still missing is a measure of how unpredictable the chaos is. Chapter 6 counts the steps until two nearby starts part, Chapter 7 turns that into one number, and Chapter 8 draws every setting of the knob in one picture.

Summary

- The **logistic map**, $x_{n+1} = r x_n (1 - x_n)$, is a straight-line rule with one extra multiplication, the feedback $1 - x_n$. For r up to 4, x stays between 0 and 1.
- Turning r gives a **steady state** at 2.8, a **cycle of period two** at 3.2, one of period four at 3.5, and at 3.9 an orbit that never settles or repeats.
- The **fixed point** is $x^* = 1 - 1/r$ and the **slope** there is $2 - r$, so the steady state holds for r below 3.
- A **cobweb** spirals in to a fixed point that holds, is pushed off one that does not, settles on a closed loop for a cycle, and at 3.9 never closes.
- A **cycle detector** reports 1, 2, 4, then None, which means not found, never proved chaotic. No value comes back in a hundred thousand steps; at 3.83 a **window** holds a cycle of three.

- **Deterministic:** same start, same orbit, every time. **Chaotic:** deterministic, bounded, never settling, sensitive to the start. It looks random and is not — May's 1976 warning.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

iterate the logistic map by hand for a step or two, and in Python for as many steps as you like	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
find the map's steady state for a given r , and say from the slope there whether it holds	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
draw a cobweb diagram on a curved rule and read from it what an orbit will do	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
tell a steady state, a cycle and a never-repeating orbit apart, and say what a program that looks for cycles can and cannot conclude	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what <i>deterministic</i> and <i>chaotic</i> mean, and why looking random is not the same as being random	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. With $r = 3$ and $x_0 = 0.5$, what are x_1 and x_2 ?
2. What is the fixed point other than zero when $r = 2.5$, and does it hold?
3. At $r = 3.2$ the fixed point is 0.688, yet the orbit does not stay there. Why not?
4. A cobweb settles on a closed loop with four corners on the hump. What is the orbit doing?
5. A cycle detector returns None. What can you conclude, and what can you not?
6. Yearly counts of an animal population jump about with no pattern. Name three possible causes.
7. What does it mean that the logistic map is deterministic, and how can its output still look random?

The butterfly effect, measured

Chapter 1 ended on a question it answered only for gentle models. If the rule is exact, is the future known? For the cooling tea the answer was nearly yes: two cups that started a degree apart grew closer every minute, so a thermometer that was slightly wrong at the start hardly mattered later.

This chapter asks the same question of the one-line rule from Chapter 5, and measures the answer. Start the rule twice, from two numbers that differ in the tenth decimal place — closer than any thermometer, ruler or kitchen scale could ever tell apart — and let both run. How long do they stay together? The answer is a count of steps, it is the same on every computer, and it is short.

What you will be able to do

When you have finished this chapter you will be able to:

- run one rule from two nearby starts and read the runs side by side
- measure the gap between two runs step by step, and read it on a logarithmic scale
- say what sensitive dependence on initial conditions means, in terms of something you can measure
- use the doubling map to explain why a forecast runs out
- say, for a given error in the start, roughly how many steps a prediction stays useful
- tell determinism from predictability, and retell Lorenz's discovery accurately

6.1 Two starts, one part in ten billion

Chapter 5's rule is $x_{n+1} = r x_n (1 - x_n)$. In words: take the number x_n you have after n steps, multiply it by $1 - x_n$ and by the knob r , and the result is the next number, x_{n+1} . In this chapter the knob is turned to $r = 4$, the largest value that keeps every x between 0 and 1.

Take two starts: 0.2, and 0.2000000001. They differ by one ten-billionth, 10^{-10} : a decimal point, nine zeros and a one. Call the two runs that follow from them the twins.

Stop and think 6.1

Run the rule from both starts. For roughly how many steps will the two runs stay within 0.1 of each other: about ten, about thirty, about three thousand, or for ever?

Answer it before you read on.

Answer. About thirty. The two runs first differ by more than 0.1 at step 30.

Most people guess thousands, or for ever, because a difference in the tenth decimal place feels like no difference at all. For the tea it would have been none. Here is the program that settles it.

code/ch06/twins.py

```

12 START = 0.2
13 NUDGE = 1e-10      # one part in ten billion
14 rule = logistic(4.0) # x -> 4 * x * (1 - x)
15
16
17 def twin_runs(steps: int) -> tuple[list[float], list[float]]:
18     """Run the rule from START and from START + NUDGE; keep every step."""
19     a, b = [START], [START + NUDGE]
20     for _ in range(steps):
21         a.append(rule(a[-1]))
22         b.append(rule(b[-1]))
23     return a, b

```

Listing 6.1: Two runs of one rule from two nearby starts

`logistic(4.0)` is Chapter 5's rule with the knob at 4, taken from `chaoslab`, the small library every chapter's programs share. `twin_runs` keeps two lists, `a` and `b`, and at every step feeds the last number of each back into the same rule. The program then prints both lists every fifth step:

step 0:	0.2000000000	0.2000000001
step 5:	0.5854205387	0.5854205348
step 10:	0.1478365599	0.1478366508
step 15:	0.0039360251	0.0039365381
step 20:	0.8200138734	0.8199131530
step 25:	0.9863789723	0.9873341848
step 30:	0.3203424285	0.2030211838
step 35:	0.1535471506	0.4653612556
step 40:	0.0978744041	0.8017408721

Read down the two columns. Until step 16 the runs differ by less than a millionth, and anyone who stopped reading there would call them the same run. By step 20 they differ by about a ten-thousandth. By step 30 they do not agree even in the first decimal place, and from then on they are two unrelated lists of numbers.

6.2 The gap doubles

In Chapter 1 you watched the gap between two cups. Do the same here. The program below runs the two starts, records the distance between the runs after every step, and prints beside it what the gap would be if it doubled exactly every step. First a question you can answer with Chapter 2's doubling arithmetic.

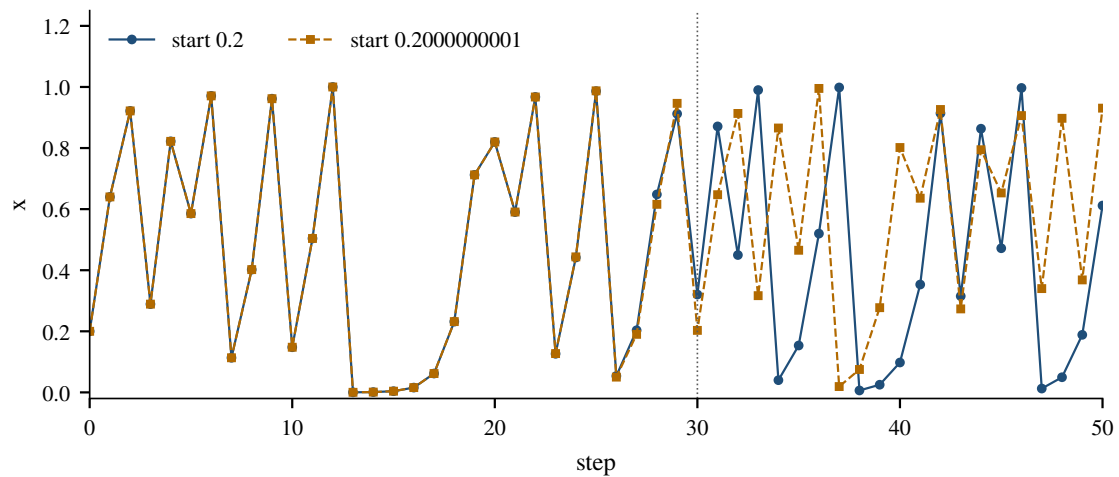


Figure 6.1: The two runs of the listing, step by step. They trace one path until about step 30 (the dotted line) and two unrelated paths after it.

Stop and think 6.2

If a gap doubles every step, how many steps does it take to grow a thousandfold?
And a billionfold?

Answer it before you read on.

Answer. Ten steps for a thousandfold, since $2^{10} = 1024$. Thirty for a billionfold, since a billion is a thousand times a thousand times a thousand.

From one ten-billionth to a tenth is exactly a billionfold. So if the gap doubles each step, the twins should part at about step thirty, which is where they did. Check it step by step:

code/ch06/gap.py

```

14 gaps = separation(rule, START, NUDGE, 45)
15
16 print("step      gap      if it doubled")
17 for n in range(0, 46, 5):
18     print(f"{n:4d}    {gaps[n]:9.2e}    {NUDGE * 2**n:14.2e}")

```

Listing 6.2: The gap between the twins, beside a gap that doubles exactly

step	gap	if it doubled
0	1.00e-10	1.00e-10
5	3.94e-09	3.20e-09
10	9.09e-08	1.02e-07
15	5.13e-07	3.28e-06
20	1.01e-04	1.05e-04
25	9.55e-04	3.36e-03
30	1.17e-01	1.07e-01
35	3.12e-01	3.44e+00
40	7.04e-01	1.10e+02

45 1.81e-01 3.52e+03

Up to step 30 the two columns keep pace. Step by step the measured gap is ragged — some steps multiply it by nearly four, a few shrink it — but over many steps it doubles, on average. Growth by the same factor every step is the exponential growth of Chapter 2, the law of compound interest. There it was your savings that grew. Here it is your error.

This is the property the chapter is named after. A rule has *sensitive dependence on initial conditions* when it pushes nearby starts apart, the gap growing by roughly the same factor at each step until the two runs have nothing to do with each other. The closer the starts, the longer that takes, but never for ever. Chapter 1's tea multiplied its gap by 0.9 every minute; the logistic map at $r = 4$ multiplies it, on average, by two. One factor smaller than one, one larger: that is the whole difference between a model that forgives you and one that does not.

Stop and think 6.3

The doubling column goes on growing after step 35, past one and past a hundred. Why does the measured gap stop following it?

Answer it before you read on.

Answer. Every x lies between 0 and 1, so two of them can never be more than 1 apart. Once the gap is the size of the whole range it can grow no further: it wanders, averaging about 0.4.

On a logarithmic scale, where each step up is a factor of ten, the story is a straight line and a ceiling. Set it beside the tea's picture in Chapter 1: there the straight line went down, here it goes up.

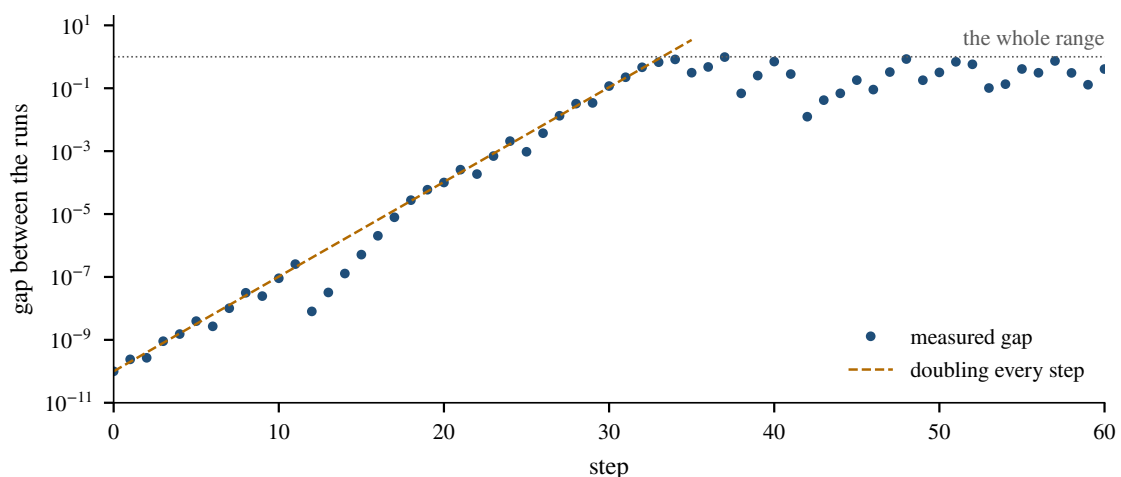


Figure 6.2: The gap between the two runs on a logarithmic scale. It climbs, raggedly, along the dashed line of doubling until it reaches the size of the whole range and can climb no further.

chaoslab

`chaoslab` carries two functions for this chapter. `separation` runs a rule from two starts and returns their distance after every step, which is what the listing above printed; `steps_until_apart` returns the first step at which the distance exceeds a tolerance you choose. The lab below asks you to write both yourself before you trust them.

Lab 6.1 — When do two runs part company?

Write `gaps`, which runs a rule from two starts and keeps the distance between them at every step, and `first_apart`, which returns the first step at which that distance exceeds a tolerance, or `None` if it never does. Both take the rule as a function, so the test tries them on Chapter 1's tea as well: its two cups never part company. For the twins, with a tolerance of 0.1, your answer must be step 30.

Starter: `code/labs/ch06/l06_01_apart.py`

Test: `code/labs/ch06/test_l06_01_apart.py`

Run: `cd code && uv run pytest -k l06_01_apart`

6.3 Where the doubling comes from

Why should a gap double? For the logistic map the answer takes some algebra, and Chapter 7 gives it. There is a simpler rule where the answer can be seen at a glance: the *doubling map*. Double the number, and if the result is 1 or more, take 1 away. It is written $x_{n+1} = 2x_n \bmod 1$, where $\bmod 1$ means 'keep only the part after the point'.

Stop and think 6.4

Start at 0.625. Write down 1 if the number is in the upper half (0.5 or more) or 0 if it is in the lower half, then apply the doubling map, and repeat until you have written three digits.

Answer it before you read on.

Answer.

0.625 is in the upper half: 1. Doubled it is 1.25, and taking 1 away leaves 0.25, in the lower half: 0. Doubled again it is 0.5, in the upper half: 1. You wrote 1, 0, 1.

Now write 0.625 in *binary*. In the decimals you learnt at school the digits after the point count tenths, hundredths and thousandths. In binary they count halves, quarters and eighths, and each digit is 0 or 1. Since 0.625 is $\frac{1}{2} + \frac{1}{8}$, in binary it is written 0.101: the three digits you have just written down.

That is no coincidence. Doubling a binary number moves every digit one place to the left, as multiplying by ten does to decimal digits, and taking 1 away drops the digit that has crossed the point. The first digit after the point says which half the number is in. So the doubling map reads out the binary digits of its start, one per step: its whole future is

written, in advance, in ever finer digits of the present. The program below checks this for the chapter's two starts.

code/ch06/digits.py

```

16 def binary_digits(x: float, n: int) -> str:
17     """The first n binary digits of x after the point.
18
19     x * 2**n moves the point n places right, int() drops what is left
20     over, and format(..., "b") writes the whole number in binary.
21     """
22     return format(int(x * 2**n), f"0{n}b")
23
24
25 def sides(x: float, n: int) -> str:
26     """Run the doubling map n steps: 1 in the upper half, 0 in the lower."""
27     out = ""
28     for _ in range(n):
29         out += "1" if x >= 0.5 else "0"
30         x = doubling(x)
31     return out

```

Listing 6.3: Binary digits worked out directly, and read out by the doubling map

```

digits of 0.2000000000: 001100110011001100110011001100110011
doubling map sides: 001100110011001100110011001100110011
digits of 0.2000000001: 001100110011001100110011001110100001
doubling map sides: 001100110011001100110011001110100001

```

For each start the digits and the doubling map's record of upper and lower halves are the same string. The two starts share their first 32 binary digits, so the map keeps them on the same side for exactly 32 steps, and after that the digits that tell them apart start to be read.

Here is sensitive dependence with nothing hidden. A measurement tells you the first so many digits of the state; the rule uses up one digit per step; once the measured digits are used up, the forecast depends on digits nobody measured. Ten is a little more than $2 \times 2 \times 2 = 8$, so one decimal digit carries a little more than three binary ones, and ten decimal places carry a little over thirty. A computer, too, stores only so many binary digits of each number; what that does to a chaotic calculation is Chapter 16's subject.

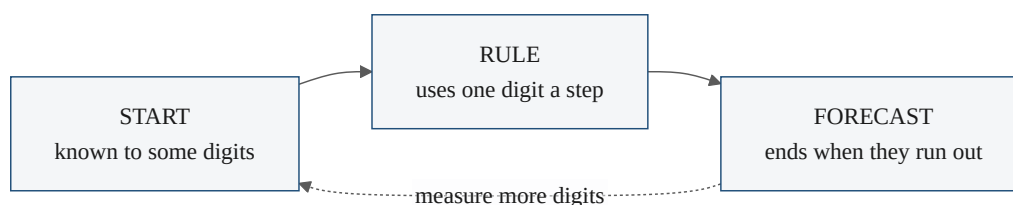


Figure 6.3: Why a chaotic forecast runs out. The start is known to some digits and the rule consumes them one step at a time; once they are used up, the forecast rests on digits nobody measured.

Lab 6.2 — Digits known, steps predicted

Write `shared_digits`, which counts how many leading binary digits two numbers have in common, and `steps_together`, which runs the doubling map on both and counts the steps before they land on different sides of one half. The test checks that the two counts agree for many pairs, and that for the chapter's two starts both are 32.

Starter: `code/labs/ch06/l06_02_digits.py`
 Test: `code/labs/ch06/test_l06_02_digits.py`
 Run: `cd code && uv run pytest -k l06_02_digits`

6.4 What a digit is worth

The doubling map says that each correct binary digit of the start buys one more step of forecast. The logistic map at $r = 4$ is not the doubling map, but you have seen that its gap doubles on average, so the same bargain should hold for it, roughly. That can be measured.

Stop and think 6.5

The twins, whose starts agreed to ten decimal places, stayed within 0.1 of each other for about thirty steps. An instrument a hundred times more precise would give you twelve decimal places. About how many steps then: about thirty-seven, or about sixty?

Answer it before you read on.

Answer. About thirty-seven. A hundred is about $2^7 = 128$, so an error a hundred times smaller needs about seven more doublings to catch up.

Twice the digits does not buy twice the forecast. The program below checks the arithmetic on a thousand starts spread evenly between 0 and 1, since any one start can be lucky or unlucky: for each size of error it counts the steps before the forecast is more than 0.1 out, and averages.

`code/ch06/worth.py`

```

15 TOL = 0.1                                     # "no longer useful"
16 STARTS = [(i + 0.5) / 1000 for i in range(1000)]
17 ERRORS = {2: 1e-2, 4: 1e-4, 6: 1e-6, 8: 1e-8, 10: 1e-10, 12: 1e-12}
18
19
20 def average_steps(error: float) -> float:
21     """Steps a forecast stays within TOL, averaged over every start."""
22     total = 0
23     for x0 in STARTS:
24         total += steps_until_apart(rule, x0, error, TOL)
25     return total / len(STARTS)

```

Listing 6.4: Steps a forecast survives, averaged over a thousand starts

```

start known to 2 decimal places: 4.1 steps
start known to 4 decimal places: 10.8 steps
start known to 6 decimal places: 17.4 steps
start known to 8 decimal places: 24.1 steps
start known to 10 decimal places: 30.6 steps
start known to 12 decimal places: 37.3 steps

```

Each pair of decimal places adds about seven steps, so each one adds about 3.3, which is the doubling map's arithmetic: a decimal digit is worth a little more than three binary ones. Read the table the other way round and it answers the question a forecaster asks. Measure the start to two decimal places and the forecast lasts about 4.1 steps; measure it to ten and it lasts about 30.6; heroic precision, twelve places, buys 37.3. Chapter 7 turns this table into one number for any chaotic system and a formula for how far ahead it can be predicted. Here it is enough that you can measure it.

6.5 Lorenz's printout

In 1961 the meteorologist Edward Lorenz, at the Massachusetts Institute of Technology, was running a small computer model of the weather. Wanting to look at one stretch of a run again, he restarted it from the middle, typing in the numbers from a printout of the earlier run. The printout showed three decimal places, 0.506, where the machine had held six, 0.506127. The new run followed the old one for a while, then drifted away, until the two forecasts had nothing in common. The difference between the starts, about one part in four thousand, was the kind of error he had assumed would not matter.

The logistic map can replay the accident. Run it from the full number, run it again from the full number, and run it once from the printout.

Stop and think 6.6

Suppose Lorenz had typed all six decimal places, 0.506127, the very number the machine had held. Would his second run have matched the first?

Answer it before you read on.

Answer. Yes, digit for digit and step for step. There is no chance in the rule: the same start always gives the same future.

code/ch06/printout.py

```

14 FULL = 0.506127    # what the machine held
15 PRINTED = 0.506    # what the printout showed: three decimal places
16
17 first = orbit(rule, FULL, 20)
18 again = orbit(rule, FULL, 20)    # the same start, typed in full
19 rounded = orbit(rule, PRINTED, 20) # the start copied off the printout

```

Listing 6.5: Lorenz's printout, replayed on the logistic map

step	first	again	rounded
0	0.506127	0.506127	0.506000
2	0.000601	0.000601	0.000576
4	0.009580	0.009580	0.009188
6	0.146050	0.146050	0.140356
8	0.999995	0.999995	0.998792
10	0.000081	0.000081	0.019207
12	0.001290	0.001290	0.278701
14	0.020504	0.020504	0.630075
16	0.295517	0.295517	0.252392
18	0.557118	0.557118	0.740388
20	0.051519	0.051519	0.710870

again == first, every digit of every step: True

The column `again` is the first run repeated, and the program compares every number of it with the first run and prints `True`. The column `rounded` starts 0.000127 away and differs by more than 0.1 at step 12, sooner than the twins, because its error started out far larger.

The trap

‘If I run a chaotic model again, I will get something different.’ Not from the same start. A chaotic rule is a machine like any other: the same input gives the same output, every digit of it, as the `again` column shows. What changes the future is a changed start, however small. If you rerun a computer model and get a different answer, something changed — the start, the rule or the arithmetic — and Chapter 16 is about the last of those.

In 1972 Lorenz gave a talk whose title asked whether the flap of a butterfly’s wings in Brazil could set off a tornado in Texas. The phrase stuck, and sensitive dependence on initial conditions has been called the *butterfly effect* ever since. His own equations, and the shape they draw, are Chapter 9’s.

Stop and think 6.7

A tornado strikes Texas. Suppose you could rerun the atmosphere from two weeks earlier with one butterfly’s flap removed. Would the tornado be gone?

Answer it before you read on.

Answer. Perhaps, perhaps not, and nobody could say in advance. The rerun is a different atmosphere: it might have no tornado, a tornado somewhere else, or one in Texas anyway.

The trap

‘The butterfly causes the tornado.’ Lorenz’s question was about prediction, not about power. The storm’s energy comes from the atmosphere, not from the insect. Remove the flap and the weather takes a different path, which may or may not contain a tornado; every other flap, gust and breath does the same, so no single one of them is the cause. In the replay above, the three dropped digits did not make the rounded run what it became: the rule acting on the whole start did. What the butterfly shows is that weather two weeks away depends on details nobody can measure.

6.6 Deterministic, and still unpredictable

Two words have been doing two different jobs in this chapter, and they need names. A rule is *deterministic* when the same start always gives the same future: no dice, no chance. A system is *predictable*, over some span, when a start you can actually measure pins down its future over that span. The tea is both. The logistic map at $r = 4$ is deterministic — checked above, digit for digit — and still unpredictable beyond a few dozen steps, because no measured start has enough digits.

Stop and think 6.8

The logistic map at $r = 4$ jumps about with no pattern the eye can see, and nobody can forecast it for long. Is its output random?

Answer it before you read on.

Answer. No. Every number follows exactly from the one before, and you can compute the whole list again and get every digit back. What cannot be had is the start, not the rule.

The trap

‘Chaos means random.’ Chaos is the opposite of chance. A chaotic system obeys its rule exactly at every step; what looks like randomness is the rule reading out, one step at a time, digits of the start that nobody knows. A random sequence has no rule at all. A chaotic one has a rule you cannot outrun. How to tell the two apart when all you have is data is Chapter 15’s question.

Where you meet this

A coin toss is used to settle things because nobody can predict it, and yet a spinning coin obeys Newton’s laws like a stone. The statistician Persi Diaconis and colleagues had a machine built that launches a coin the same way every time, and it lands the same way up every time. A thumb cannot repeat its launch that closely, so the outcome of a real toss depends on differences too small to control. The coin is

deterministic, and in your hand it is unpredictable.

Laplace's demon, from Chapter 1, needed the rule and the present exactly. This chapter has priced the word *exactly*. For the logistic map every step forward costs one more binary digit of the present, so seeing for ever needs infinitely many, and an instrument that is merely excellent sees a few dozen steps and then nothing. The clockwork is intact. It is the clock face that cannot be read finely enough.

Summary

- Two runs of the logistic map at $r = 4$ that start one ten-billionth apart agree to a millionth for 16 steps and differ by more than 0.1 from step 30: counted exactly, the same on every computer.
- Their gap roughly **doubles each step**, which is exponential growth of the error, until it reaches the size of the whole range. That is **sensitive dependence on initial conditions**: a straight line up on a logarithmic scale, then a ceiling.
- The **doubling map** reads out the binary digits of its start, one per step, so its future is written in digits of the present that no measurement has.
- Each correct decimal digit of the start buys only about 3.3 more steps of forecast: twice the digits is not twice the forecast.
- Lorenz's rounded restart, replayed on the logistic map: the full start reruns identically, the rounded one parts company. The **butterfly effect** is about prediction, not about causes.
- A rule can be **deterministic** and still **unpredictable**: chaos is not chance, and rerunning the same start gives the same future.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

run one rule from two nearby starts and read the runs side by side	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
measure the gap between two runs step by step, and read it on a logarithmic scale	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what sensitive dependence on initial conditions means, in terms of something you can measure	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
use the doubling map to explain why a forecast runs out	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say, for a given error in the start, roughly how many steps a prediction stays useful	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
tell determinism from predictability, and retell Lorenz's discovery accurately	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Two runs of the logistic map at $r = 4$ start at 0.2 and 0.2000000001. Roughly when do they first differ by more than 0.1?
2. A gap of one millionth doubles every step. After how many steps is it larger than 1?
3. Why does the gap between two runs of the logistic map stop growing?
4. Write 0.75 in binary, and write down which half it is in for three steps of the doubling map.
5. A forecast from a start known to ten decimal places lasts about 30.6 steps. About how long for fourteen decimal places?
6. Lorenz restarted from 0.506 instead of 0.506127. What would he have seen had he typed all six digits, and why?
7. Is the logistic map at $r = 4$ random? What is the difference between deterministic and predictable?

A number for chaos: the Lyapunov exponent

Chapter 6 counted two things about the logistic map at $r = 4$: the gap between two nearby runs roughly doubles at every step, and each correct decimal digit of the start buys only a few more steps of forecast. Both were found by counting, for one map. The question behind them is one you will ask of every system in this book: how fast do small errors grow, and how far ahead does that let you see?

This chapter turns the counting into one number, the Lyapunov exponent, which can be measured for any system, and the number into a horizon: how many steps a prediction is good for. Hold on to this question while you read: if you measured the start of a chaotic system a thousand times more precisely, how much further ahead could you predict it?

What you will be able to do

When you have finished this chapter you will be able to:

- find how much one step of a rule stretches a small error, and check it by running two copies of a program
- use the natural logarithm to turn a chain of multiplications into a sum, and back again
- compute the Lyapunov exponent of the logistic map from an orbit, and read its sign
- check a computed exponent against a value known exactly
- turn an exponent, a measurement error and a tolerance into a prediction horizon
- say what a better measurement buys, counted in steps

7.1 One step at a time

Chapter 6 noticed that the gap between two runs grows raggedly: some steps nearly quadruple it, a few shrink it, and only on average does it double. Start with that raggedness. Run the logistic map at $r = 4$ from $x = 0.3$ and from 10^{-10} higher, and watch what each single step does to the gap: divide the gap after the step by the gap before. Call that ratio the step's *stretch*.

code/ch07/stretch.py

```

11 def step(r: float, x: float) -> float:
12     """One step of the logistic map."""
13     return r * x * (1.0 - x)
14
15
16 def stretch(r: float, x: float) -> float:
17     """How much one step multiplies a tiny nudge at x: the slope there."""
18     return r * (1.0 - 2.0 * x)

```

Listing 7.1: The logistic map and its stretch

The function `step` is the logistic map of Chapter 5. The function `stretch` is its slope, $r(1 - 2x)$. Chapter 3 measured a slope by nudging x and seeing how far the output moved; for this rule it can be written down exactly, and the first lab checks the formula by nudging. The rest of the program measures the stretch of the first eight steps from the two runs, beside the slope where the first run is:

code/ch07/stretch.py

```

36 r, a, b = 4.0, 0.3, 0.3 + 1e-10
37 print("step      x      gap grew by      slope there")
38 for n in range(1, 9):
39     grew = abs(step(r, b) - step(r, a)) / abs(b - a)
40     print(f"{n:4d}{a:9.3f}{grew:15.3f}{abs(stretch(r, a)):15.3f}")
41     a, b = step(r, a), step(r, b)

```

Listing 7.2: Measuring the stretch of eight steps

step	x	gap grew by	slope there
1	0.300	1.600	1.600
2	0.840	2.720	2.720
3	0.538	0.301	0.301
4	0.994	3.955	3.955
5	0.022	3.820	3.820
6	0.088	3.296	3.296
7	0.321	1.433	1.433
8	0.872	2.973	2.973

The two right-hand columns agree. Over one step a tiny gap is multiplied by the slope of the rule where you are, so the stretch can be read off the rule rather than measured with a second run. A negative slope also flips the gap to the other side, which does not change its size; the table prints sizes.

And the stretches are nothing like a steady 2. Step 3 shrinks the gap to less than a third of itself, and steps 4 and 5 nearly quadruple it. At step 3 the run is near 0.5, where the parabola is flat at the top; at steps 4 and 5 it is near 1 and near 0, where the parabola is steep. Chaos does not stretch evenly: hard in one place, gently in another, and sometimes it squeezes.

Stop and think 7.1

Suppose one step multiplies a gap by 4 and the next multiplies it by $\frac{1}{4}$. What has happened to the gap over the two steps? And what single factor, used at both steps, would have done the same?

Answer it before you read on.

Answer. Nothing has happened: $4 \times \frac{1}{4} = 1$, and the gap is back where it started. The single factor is 1, not the 2.125 you get by averaging 4 and $\frac{1}{4}$.

Growth compounds: each step multiplies whatever the step before left behind. So the typical stretch is not the ordinary average of the stretches. It is the factor that, used at every step, gives the same total.

The trap

‘To find the typical stretch per step, average the stretches.’ Try it on the table. The eight stretches average 2.51, and a gap multiplied by that eight times would grow about 1587-fold. Multiply the eight actual stretches together instead and you find that the gap grew 278-fold. When things multiply, adding them up answers a question nobody asked; the average you want must multiply too.

7.2 A logarithm turns multiplying into adding

Chapter 2 used a logarithm to answer ‘how many steps until?’. This chapter needs one particular logarithm, the *natural* logarithm, written $\ln$, and one thing it does. Here are a few of its values:

number	$\frac{1}{2}$	1	2	4	10	e
its natural logarithm	-0.69	0	0.69	1.39	2.30	1

Look at 2 and 4. Since $4 = 2 \times 2$, its logarithm is twice that of 2. That is the whole of what this chapter needs:

$$\ln(a \times b) = \ln a + \ln b.$$

In words: the logarithm of a product is the sum of the logarithms, so multiplying two numbers becomes adding their logarithms. Numbers above 1 have positive logarithms and numbers between 0 and 1 negative ones: $\frac{1}{2} \times 2 = 1$, so $\ln \frac{1}{2}$ and $\ln 2$ add up to $\ln 1$, which is 0. The last column holds e , about 2.718, the number whose natural logarithm is exactly 1; it plays the part that 10 plays for the logarithm to base ten. In Python the natural logarithm is `math.log`, and `math.log(2)` starts 0.6931.

Stop and think 7.2

Using only $\ln 2 \approx 0.69$, what is $\ln 8$? And what is $\ln \frac{1}{4}$?

Answer it before you read on.

Answer. $8 = 2 \times 2 \times 2$, so $\ln 8 = \ln 2 + \ln 2 + \ln 2$, about 2.08. And $\frac{1}{4} \times 4 = 1$, so $\ln \frac{1}{4}$ is -1.39 .

If you wrote 2.07, you did it right: the rounding of $\ln 2$ was counted three times.

This is how the logarithm fixes the average. If a gap is multiplied by g at a step, its logarithm goes up by $\ln g$: a *factor* per step becomes a *rate* per step, and rates add, so they can be averaged in the ordinary way. For the two steps of the question, the rates $\ln 4$ and $\ln \frac{1}{4}$ add to 0, the average rate is 0, and the factor whose logarithm is 0 is 1: the right answer, found by adding.

To go back from a rate to a factor you need the number whose logarithm is that rate, written e^λ , ‘ e to the power λ ’; λ is the Greek letter lambda and stands for the rate. So $e^0 = 1$, $e^1 = e$, and $e^{\ln 2} = 2$. Now the eight steps of the table. The logarithms of their stretches average 0.70, and $e^{0.70}$ is about 2.02: on average the gap grew by a factor of about 2 per step, and 2.02 multiplied by itself eight times is the 278 the table produced. ‘Roughly doubles’ has become a measurement.

7.3 The Lyapunov exponent

Eight steps are a sample. Follow the orbit for a hundred thousand steps instead, take the natural logarithm of the size of the stretch at every point it visits, and average. The result is the *Lyapunov exponent*, named after the Russian mathematician Aleksandr Lyapunov, whose work on the stability of motion appeared in 1892. If $s_1, s_2, \dots, s_N$ are the stretches along the orbit,

$$\lambda = \frac{\ln |s_1| + \ln |s_2| + \dots + \ln |s_N|}{N}.$$

In words: λ is the average of the logarithms of the stretches. The bars mean ‘the size of’, because a flip does not make a gap bigger or smaller.

Read λ as a rate. Over N steps a small error is multiplied by about $e^{\lambda N}$, so its sign says what kind of system you have:

- **λ above zero:** errors grow exponentially, and the system is chaotic;
- **λ below zero:** errors shrink, and the system forgets its disturbances;
- **λ equal to zero:** the borderline, where errors neither grow nor shrink on average.

Stop and think 7.3

At $r = 4$ the gap roughly doubles at every step. What value of λ would you predict?

Answer it before you read on.

Answer. About $\ln 2$, which is 0.69: a gap multiplied by 2 at every step grows at $\ln 2$ per step.

Stop and think 7.4

At $r = 2.8$ the orbit settles on the fixed point $1 - 1/r$ of Chapter 5. There the slope $r(1 - 2x)$ works out to $2 - r = -0.8$, and $\ln 0.8$ is about -0.22 . What is λ ?

Answer it before you read on.

Answer. About -0.22 . Once the orbit sits on the fixed point, every term of the average is $\ln 0.8$, so the average is too. It is negative: a nudge shrinks by a fifth at every step.

code/ch07/exponent.py

```

17 def exponent(r: float, x0: float = 0.3, steps: int = 100_000,
18             burn: int = 1000) -> float:
19     """The average of  $\ln|\text{stretch}|$  along the orbit of the logistic map."""
20     x = x0
21     for _ in range(burn):           # let the orbit settle first
22         x = step(r, x)
23     total = 0.0
24     for _ in range(steps):
25         total += math.log(abs(stretch(r, x)))
26         x = step(r, x)
27     return total / steps

```

Listing 7.3: The Lyapunov exponent, by averaging

The first loop runs the map for a thousand steps and throws them away, so the orbit has settled before anything is averaged. The second adds up the logarithm of the size of the stretch at each point, then steps; the last line divides. That is the formula above, and nothing else. Run it at four settings of the knob r :

code/ch07/exponent.py

```

33 for r in (2.8, 3.2, 3.5, 4.0):
34     lam = exponent(r)
35     verdict = "errors grow" if lam > 0 else "errors shrink"
36     print(f"r = {r}: exponent {lam:+.2f} {verdict}")
37     print(f"exact value at r = 4:  $\ln 2 = \{{\text{math.log(2):.4f}}\}")$ 
```

Listing 7.4: The exponent at four settings of r

```

r = 2.8: exponent -0.22 errors shrink
r = 3.2: exponent -0.92 errors shrink
r = 3.5: exponent -0.87 errors shrink
r = 4.0: exponent +0.69 errors grow
exact value at r = 4:  $\ln 2 = 0.6931$ 

```

Both predictions hold. At $r = 2.8$ the program finds -0.22 . At $r = 4$ it finds 0.69 , and the exact value is $\ln 2 = 0.6931$: an average of a hundred thousand wildly different logarithms agrees with it to within 0.001 . At $r = 3.2$, where the orbit alternates between two values, and at $r = 3.5$, where it cycles through four, the exponent is negative too, -0.92 and -0.87 . A cycle, like a fixed point, forgets its disturbances.

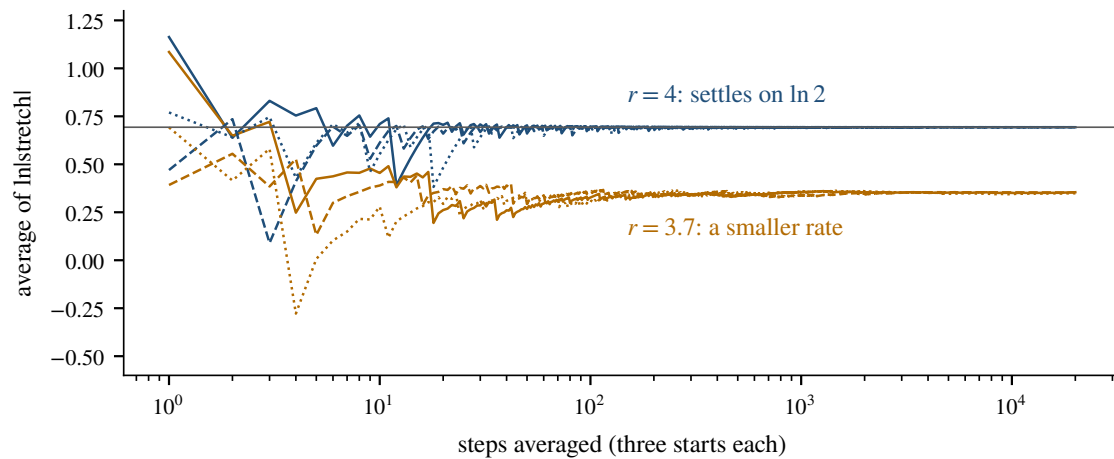


Figure 7.1: The running average of the logarithm of the stretch along six orbits: three starts at $r = 4$ and three at $r = 3.7$. Each wobbles for a few hundred steps and then settles, and the three starts at each setting settle on the same number.

What we are not proving

Two things are used here and not proved. The first is that the average settles on the same number from almost every start. That is a theorem, and ‘almost every’ matters: a start of exactly 0 stays at 0 for ever and averages $\ln 4$. A computer run can illustrate the theorem, as the figure does, but not prove it. The second is why $r = 4$ gives exactly $\ln 2$. Write x as $\sin^2 \theta$ for an angle θ , and one line of school trigonometry turns $4x(1 - x)$ into $\sin^2(2\theta)$: at $r = 4$ the map is doubling an angle in disguise, which is the doubling map of Chapter 6 with the angle counted in half-turns. Doubling stretches every error in the angle by exactly 2, and changing the disguise does not change the average rate.

Lab 7.1 — The exponent, from an orbit

Write `stretch`, the slope $r(1 - 2x)$, and `exponent`, which lets an orbit settle and then averages the logarithm of the size of the stretch along it. The tests check the slope by nudging, as Chapter 3 did, and check your exponent against three values known exactly: $\ln 2$ at $r = 4$, $\ln 0.8$ at $r = 2.8$, and the two-cycle at $r = 3.2$.

Starter: `code/labs/ch07/l07_01_exponent.py`

Test: `code/labs/ch07/test_l07_01_exponent.py`

Run: `cd code && uv run pytest -k l07_01_exponent`

7.4 How far ahead: the prediction horizon

After n steps a small error of size δ (the Greek letter delta) has grown to about $\delta e^{\lambda n}$. You stop trusting a prediction once its error reaches a tolerance T , the largest error you can live with. So the horizon is the number of steps at which $\delta e^{\lambda n} = T$. Take the logarithm of both sides: the product becomes a sum, and the logarithm of $e^{\lambda n}$ is λn , because e has logarithm 1 and is multiplied in λn times. That leaves $\ln \delta + \lambda n = \ln T$, and so

$$n \approx \frac{1}{\lambda} \ln \frac{T}{\delta}.$$

In words: the horizon is the logarithm of how many times the error is allowed to grow, divided by the rate at which it grows.

The number $1/\lambda$ is called the *Lyapunov time*: the number of steps over which errors grow by the factor e . At $r = 4$ it is $1/\ln 2$, about 1.44 steps, and the formula says that a horizon is the Lyapunov time multiplied by $\ln(T/\delta)$.



Figure 7.2: From a rule to a horizon. The slope gives the stretch at each step, the average of its logarithm gives the exponent, and the exponent with the error and the tolerance gives the horizon.

Stop and think 7.5

Suppose a forecasting model has a Lyapunov time of two days, and its forecasts are good for six. A new network of instruments cuts the error in its start tenfold. For how long are the forecasts good now: about sixty days, about twelve, or less than that?

Answer it before you read on.

Answer. About 10.6 days. Cutting the error tenfold adds $\ln 10$ to $\ln(T/\delta)$, so it adds $\ln 10$ Lyapunov times: 2×2.30 , about 4.6 days.

The trap

‘Ten times better data gives ten times longer forecasts.’ It gives a fixed number of extra steps, $\ln 10/\lambda$, whatever the error was before. At $r = 4$ that is 3.3, the figure Chapter 6 counted for each decimal digit, now derived, and true of every chaotic system with its own λ . The horizon grows with the logarithm of your precision, not with the precision: the twentieth correct digit buys no more than the first.

code/ch07/horizon.py

```

16 def horizon(lam: float, delta: float, tol: float) -> float:
17     """Steps until an error delta grows to tol, growing at lam per step."""
18     return math.log(tol / delta) / lam

```

Listing 7.5: The horizon formula

The program sets that line beside the count Chapter 6 made: its function counts follows two runs δ apart from a thousand starts until they differ by more than T . In the output, $1e-03$ is Python's way of writing 10^{-3} , a thousandth.

code/ch07/horizon.py

```

31 lam, tol = math.log(2), 0.1
32 print(" error formula counted: mean fastest slowest")
33 for delta in (1e-3, 1e-6, 1e-9, 1e-12):
34     c = counts(4.0, delta, tol)
35     mean = sum(c) / len(c)
36     print(f"{delta:7.0e}{horizon(lam, delta, tol):10.1f}")
37     f"{mean:16.1f}{min(c):10d}{max(c):10d}")

```

Listing 7.6: Formula against count, for four errors in the start

error	formula	counted: mean	fastest	slowest
1e-03	6.6	7.4	4	43
1e-06	16.6	17.4	13	24
1e-09	26.6	27.3	24	34
1e-12	36.5	37.3	33	44

The formula and the mean count agree to within a step; the count comes out a shade later, because the gap is checked only at whole steps and grows more slowly once it stops being small. The last two columns are a warning: individual starts differ, some by a lot, because the horizon, like the exponent, is an average. But look down the rows. Each thousandfold improvement adds the same ten steps; measured, the rows are 10.0 steps apart on average, against the formula's $\ln 1000 / \ln 2$, which is 6.91 divided by 0.69: 10.

Stop and think 7.6

A computer stores a number to about sixteen significant figures. Suppose you could measure the start of the map at $r = 4$ that well, so that $\delta = 10^{-16}$. With $T = 0.1$, roughly how far ahead could you predict? And how many decimal places of the start would a thousand steps need?

Answer it before you read on.

Answer. About 50 steps: fifteen factors of ten, at 3.3 steps each. A thousand steps would need the start to 303 decimal places; even a hundred steps needs 32.

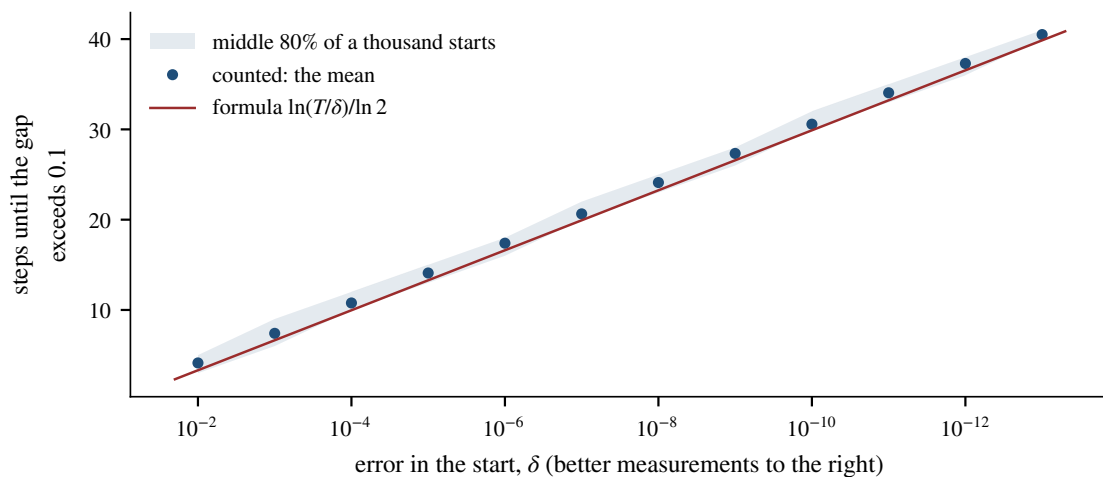


Figure 7.3: The horizon against the error in the start, on a logarithmic scale with better measurements to the right. The count (dots, and a band holding most of the starts) follows the formula's straight line: every factor of ten in precision buys the same number of extra steps.

The trap

'A bigger computer will eventually make any system predictable.' A faster computer does not know the start any better, and the horizon depends on the start only through the logarithm of its error. To double the horizon you must double the number of correct digits, and the most precise measurements ever made, by atomic clocks, reach fewer than twenty. The computer also rounds at every step, which is a fresh error of its own; Chapter 16 measures what that costs.

This is why weather forecasts improve slowly. Satellites, denser observations and faster computers have shrunk the error in the atmosphere's measured state many times over, and each factor bought the same fixed amount of lead time; the improvement usually quoted is about a day of useful forecast per decade. The atmosphere is not the logistic map, and Chapter 14 says what is different about it, but the arithmetic of the logarithm applies all the same.

Where you meet this

The Lyapunov time is how astronomers say how far ahead an orbit can be followed. In 1988 Gerald Sussman and Jack Wisdom tracked Pluto on a computer built for the purpose, found its motion chaotic, and put its Lyapunov time at about twenty million years; the planets' own story is Chapter 13's. On its own the number is not a horizon. The horizon is the Lyapunov time multiplied by $\ln(T/\delta)$, and that logarithm is only about 23 even when the error in the start is ten billion times smaller than the tolerance, so a horizon is a few dozen Lyapunov times at most.

Lab 7.2 — How far ahead, and at what price

Write `horizon`, the formula of this section; `extra_steps`, the steps gained by measuring the start a given factor more precisely; and `digits_needed`, the decimal places a forecast of a given length requires. The tests check that a thousandfold improvement buys the same number of steps whatever error you started from.

Starter: `code/labs/ch07/l07_02_horizon.py`

Test: `code/labs/ch07/test_l07_02_horizon.py`

Run: `cd code && uv run pytest -k l07_02_horizon`

7.5 Chaos coming and going

With one number per setting of the knob, you can turn the knob and watch it. Chapter 5 found a fixed point at $r = 2.8$ and a two-cycle at $r = 3.2$; between them, at $r = 3$, the fixed point gives way.

Stop and think 7.7

At the fixed point the slope is $2 - r$. What is it at $r = 3$, and what does that make λ ?

Answer it before you read on.

Answer. $2 - 3 = -1$, and $\ln 1 = 0$, so $\lambda = 0$: a nudge is neither stretched nor shrunk, only flipped. Run at $r = 3$, the program finds a value within 0.001 of zero.

Zero is the borderline, and it marks where one kind of behaviour hands over to another. To find such places, take the exponent at many settings of r . The listing uses the library's `lyapunov`, which is the loop of `exponent` with the rule and its stretch passed in, so that it works for any map:

`code/ch07/sweep.py`

```

14 print("    r    exponent")
15 for k in range(16):
16     r = (340 + 4 * k) / 100          # 3.40, 3.44, ..., 4.00
17     lam = lyapunov(logistic(r), logistic_slope(r), 0.3, 20_000)
18     verdict = "errors grow" if lam > 0 else "errors shrink"
19     print(f"{r:5.2f}{lam:+11.2f} {verdict}")

```

Listing 7.7: The exponent at sixteen settings of r

r	exponent	
3.40	-0.14	errors shrink
3.44	-0.02	errors shrink
3.48	-0.24	errors shrink
3.52	-0.19	errors shrink
3.56	-0.08	errors shrink
3.60	+0.18	errors grow

3.64	+0.23	errors grow
3.68	+0.34	errors grow
3.72	+0.36	errors grow
3.76	+0.39	errors grow
3.80	+0.43	errors grow
3.84	-0.04	errors shrink
3.88	+0.46	errors grow
3.92	+0.53	errors grow
3.96	+0.54	errors grow
4.00	+0.69	errors grow

Read down the middle column. It is negative up to $r = 3.56$ and positive from $r = 3.60$, so chaos arrives somewhere in between. After that it climbs towards $\ln 2$ at $r = 4$, except at $r = 3.84$, where it is negative again: a narrow range of the knob, inside the chaos, where the orbit has found a cycle and forgets its disturbances once more. A table this coarse steps over most such ranges; the figure does not.

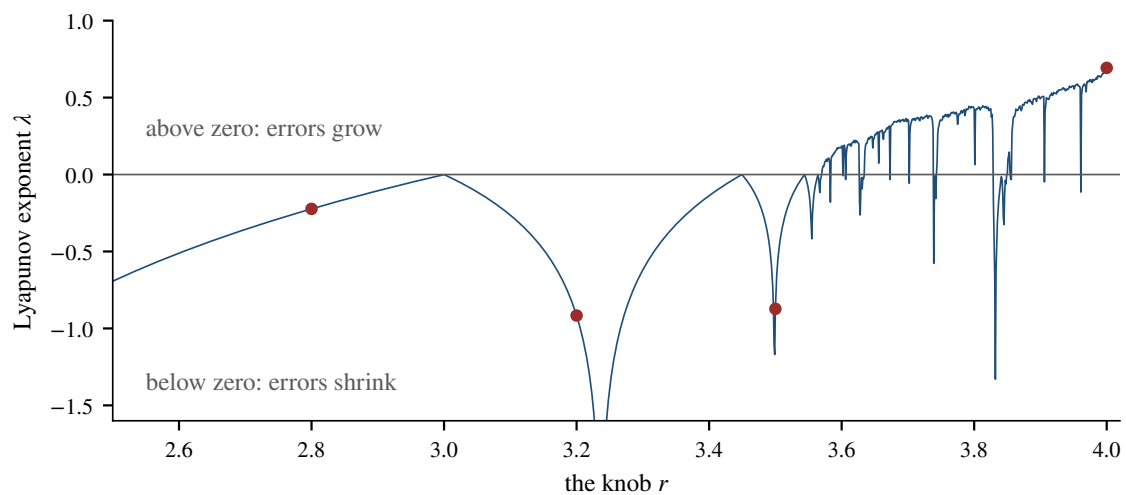


Figure 7.4: The Lyapunov exponent across the knob r . It is negative wherever the orbit settles, touches zero at each hand-over, is positive where the map is chaotic, and dips below zero again in narrow windows. The red dots are the four values of the exponent listing.

Below the start of chaos the curve never rises above zero. It touches zero at $r = 3$ and again a little below $r = 3.5$, each time a cycle hands over to one twice as long. Beyond, it is mostly positive, pierced by downward spikes: windows where order comes back. Where exactly chaos begins, why the hand-overs crowd together and which window is widest are Chapter 8's questions. This chapter's answer is the measurement that tells order from chaos at any setting: one number, and its sign.

Summary

- One step multiplies a tiny error by the **stretch**, the slope of the rule where you are. Along a chaotic orbit the stretches vary wildly, and their ordinary average is the wrong summary.

- The **natural logarithm** turns multiplying into adding: $\ln(a \times b) = \ln a + \ln b$, $\ln 1 = 0$, $\ln 2 \approx 0.69$ and $\ln e = 1$. A growth factor g per step becomes a rate $\ln g$, and rates can be averaged.
- The **Lyapunov exponent** λ is the average of $\ln |\text{stretch}|$ along an orbit: positive for chaos, negative for order, zero at the borderline. For the logistic map at $r = 4$ it is $\ln 2$, and the computation agrees.
- The **prediction horizon** is about $\frac{1}{\lambda} \ln(T/\delta)$, and $1/\lambda$ is the **Lyapunov time**. A start measured a thousand times more precisely buys $\ln 1000/\lambda$ more steps, about ten at $r = 4$: a fixed amount, not a thousand times as many.
- Across r the exponent shows **chaos coming and going**: negative where the orbit settles, zero at each hand-over, positive in chaos, and negative again in narrow windows.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

find how much one step of a rule stretches a small error, and check it by running two copies of a program	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
use the natural logarithm to turn a chain of multiplications into a sum, and back again	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
compute the Lyapunov exponent of the logistic map from an orbit, and read its sign	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
check a computed exponent against a value known exactly	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
turn an exponent, a measurement error and a tolerance into a prediction horizon	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what a better measurement buys, counted in steps	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. A gap is multiplied by 3, then by $\frac{1}{3}$, then by 3, then by $\frac{1}{3}$. By how much has it grown overall, and what is the ordinary average of the four factors?
2. Using $\ln 2 \approx 0.69$ and $\ln 10 \approx 2.30$, find $\ln 20$ and $\ln 0.2$.
3. An orbit settles on a fixed point where the slope is -0.5 . What is the Lyapunov exponent, and what does its sign say?
4. A system has $\lambda = 0.5$ per step. An error of 0.001 must stay below 0.1. About how many steps is the prediction good for?

5. In the same system you measure the start a hundred times more precisely. How many steps do you gain? And if you do it again?
6. Why will a faster computer, on its own, hardly lengthen the forecast of a chaotic system?
7. At $r = 3.84$ the exponent is negative, although it is positive at $r = 3.80$ and $r = 3.88$. What is happening there?

The road to chaos, and a number nobody expected

Chapter 5 turned one knob, r , to four positions and found four behaviours. At $r = 2.8$ the logistic map settled on one value; at 3.2 it alternated between two; at 3.5 it cycled through four; at 3.9 it never repeated at all. Four photographs of one machine. What happens between them? Does order turn into chaos at one setting of the knob, the way water boils at a hundred degrees, or along a road?

The answer is one picture, among the most reproduced in science, and hidden in it is a number nobody put there. It comes out of other equations too, it has been measured in liquid helium, and by the end of the chapter you will have computed it to five figures with nothing but addition, subtraction, multiplication and division.

What you will be able to do

When you have finished this chapter you will be able to:

- read a bifurcation diagram and say what the map does at any value of the knob it shows
- locate a period doubling by watching runs of the map, and say what limits how precisely that can be done
- find a superstable parameter by bisection, using nothing but $+$, $-$, $\times$ and $\div$
- compute the ratio of successive gaps between doublings, and estimate where the doublings run out
- run the same search on a second rule and say what the comparison shows
- find a window of order inside the chaotic range, and state what Li and Yorke proved about period three

8.1 One picture for every setting of the knob

To see every r at once you need a recipe that turns a whole run into a few dots, and then you draw the dots of thousands of runs side by side.

1. Choose a value of r and start the map at $x = 0.5$.
2. Run it for a thousand steps and throw them away. That stretch is the *transient*: the orbit finding its way to wherever it is going.
3. Run it for three hundred more, and draw a dot above r at the height of each value.

code/ch08/bifurcation.py

```

13 BURN = 1000    # steps thrown away while the orbit settles
14 KEEP = 300    # steps kept: where the orbit lives
15
16
17 def long_run(r: float, burn: int = BURN, keep: int = KEEP) -> list[float]:
18     """The values the logistic map visits at r once it has settled."""
19     xs = orbit(logistic(r), 0.5, burn + keep)
20     return xs[burn + 1:]

```

Listing 8.1: The recipe: let the orbit settle, then keep what it visits

orbit is the book's library function that applies a rule over and over and keeps every value; xs[burn + 1:] keeps the ones after the transient.

Stop and think 8.1

At $r = 2.8$ the map settles on the fixed point $1 - 1/r$ that Chapter 5 found. Once the transient has been thrown away, how many different dots does the recipe draw above $r = 2.8$?

Answer it before you read on.

Answer. One. All three hundred values are 0.6429 to four decimals, which is $1 - 1/2.8$, so the dots land on top of each other.

A two-cycle draws two dots, a four-cycle four, and an orbit that never repeats draws three hundred different ones, a short vertical smear. The rest of the listing asks at six values of r :

r	period	where the orbit lives
2.8	1	0.6429
3.2	2	0.5130 0.7995
3.5	4	0.3828 0.5009 0.8269 0.8750
3.56	8	8 values from 0.35 to 0.89
3.567	16	16 values from 0.34 to 0.89
3.9	-	anywhere from 0.10 to 0.97

Two rows are new: at 3.56 the orbit cycles through eight values, and at 3.567 through sixteen. Now run the recipe for thousands of values of r between 2.8 and 4 and draw every dot at once.

This is the *bifurcation diagram*, the whole of Chapter 5 in one frame. Read it from left to right, as a journey along the knob. One line up to $r = 3$; there it splits in two; near 3.45 each branch splits again; then everything dissolves into a smear streaked with pale stripes. A *bifurcation* (from the Latin for a two-pronged fork) is a value of the knob at which the kind of long-run behaviour changes. Every split here is a *period doubling*: each value the orbit visited becomes two, so the cycle takes twice as long to come round.

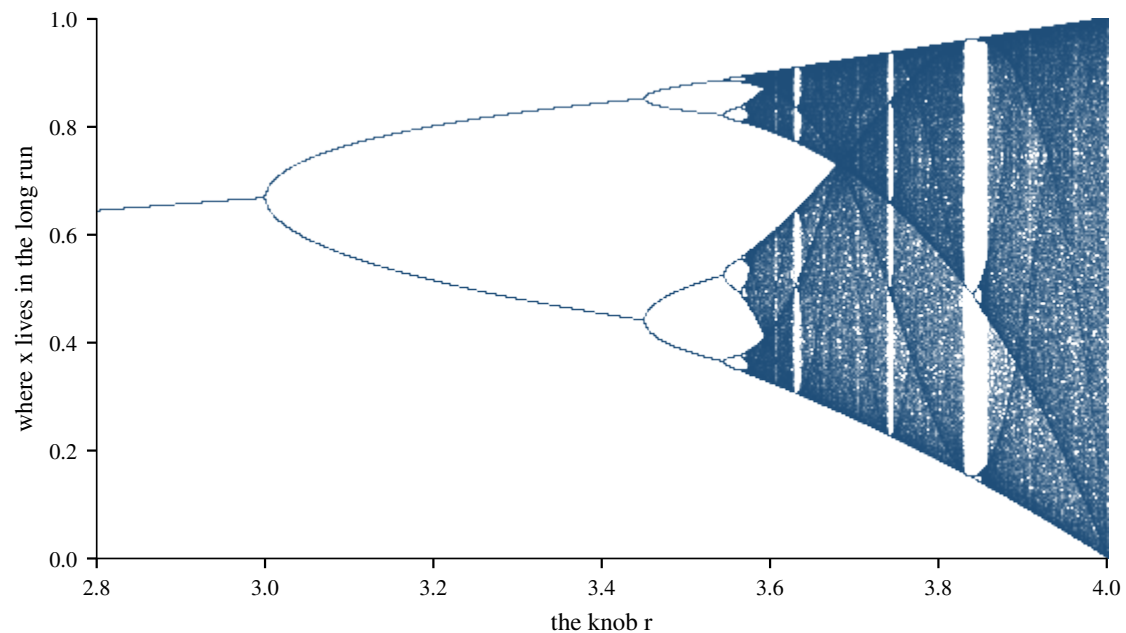


Figure 8.1: The bifurcation diagram of the logistic map. Above each value of r the dots show where x spends its time once the transient is over: one line where it settles, two where it alternates, a smear where it never repeats.

Stop and think 8.2

Look at the picture between $r = 3.5$, where the orbit has period four, and the smear. Does order give way to chaos at one value of r , like a switch?

Answer it before you read on.

Answer. It looks like a switch, and it is not. Magnify the stretch before the smear and each branch splits again, and again: periods 8, 16, 32 and on without end, all fitting in before $r = 3.5699$. The table has already caught two of them.

The trap

‘The change from order to chaos is a sudden switch.’ Not on this road. Order breaks down in an infinite *cascade* of doublings, each arriving sooner than the one before, and chaos begins where the cascade runs out. The smear looks sudden only because its last splits are too close together to draw. Other systems reach chaos by other roads; this one, the *period-doubling cascade*, is the road a single hump takes.

8.2 Forks that crowd together

Where exactly is each split? The table brackets them: period one at 2.8, period two at 3.2, so the first split lies between. Try the middle of the bracket, see which period the map

settles into there, keep the half that still contains the change, and repeat. Twenty rounds shrink the bracket a millionfold. This is *bisection*, and all it needs is a way to ask ‘is it still period p here?’

code/ch08/doublings.py

```

19 def fork(p: int, lo: float, hi: float, rounds: int = 20) -> float:
20     """The r between lo and hi at which period p gives way to 2p."""
21     for _ in range(rounds):
22         mid = (lo + hi) / 2
23         q = settled_period(mid)
24         if q is not None and q <= p:
25             lo = mid          # still period p here: the fork is further on
26         else:
27             hi = mid          # already doubled: the fork is behind us
28     return (lo + hi) / 2

```

Listing 8.2: Pinning down a split by halving the bracket

settled_period runs the map for twenty thousand steps and asks the library’s period function what repeats. The listing runs fork on the four brackets the table provides.

Stop and think 8.3

The first two splits turn out to be 0.4498 apart in r , and the next two 0.0947 apart. How many times smaller is the second gap? If the next gap shrinks by the same factor, how long will it be?

Answer it before you read on.

Answer. About 4.75 times smaller: $0.4498 \div 0.0947$. The same factor again would make the next gap about 0.02.

fork	r	gap	ratio
1 -> 2	2.9995		
2 -> 4	3.4493	0.4498	
4 -> 8	3.5440	0.0947	4.75
8 -> 16	3.5644	0.0204	4.64

The third gap is 0.0204; the second divided by the third is 4.64. So each gap is a little under five times the next, and the gaps still to come (the last one divided by 4.7, then by 4.7 again, and so on) add up to the last gap divided by $4.7 - 1$, a little over a quarter of it. Infinitely many splits fit in a finite stretch of the knob: the cascade ends a short way beyond 3.5644.

Stop and think 8.4

The first split is known exactly. Chapter 5 found that the fixed point loses its grip at $r = 3$, where its slope, $2 - r$, passes -1 . The table says 2.9995. What went wrong?

Answer it before you read on.

Answer. Nothing in the rule: the measurement ran out of patience. At $r = 2.9998$ each step multiplies a small nudge by $2 - r = -0.9998$, so the nudge shrinks by a fiftieth of a per cent per step, and after twenty thousand steps the orbit is still visibly flipping from side to side. `settled_period` calls it a two-cycle.

Every split measured by watching comes out early for the same reason: close to a split, the cycle that is losing its grip settles very slowly.

Note

The second split can be found exactly. Once round the two-cycle, a nudge is multiplied by the slope at each of its two points, and a page of school algebra shows that the product is $4 + 2r - r^2$. That reaches -1 when $r^2 - 2r - 5 = 0$, at $r = 1 + \sqrt{6} = 3.4495$; the table's 3.4493 is early, as expected. Later splits have no formula this simple.

8.3 The top of the hump

In 1975 Mitchell Feigenbaum, a physicist at the Los Alamos laboratory in New Mexico, was working out where these splits happen on a programmable pocket calculator, and noticed the pattern you have just seen: each gap a fixed fraction of the one before. To measure that fraction sharply, use a better-placed landmark than the split, where a cycle hangs on by its fingertips: the point in each stretch where the cycle is held most firmly.

The slope of the logistic map at x is $r(1 - 2x)$, the stretch factor Chapter 3 taught you to measure. It is zero at $x = 0.5$, the top of the hump. A cycle through 0.5 multiplies any nudge by that zero once per trip, wiping it out. Such a cycle is called *superstable*, and for each period 1, 2, 4, 8, ... there is exactly one value of r on the road where it happens: this chapter calls those values *landmarks*. Each is the answer to one equation: start at 0.5, take as many steps as the period, and arrive back at 0.5.

Stop and think 8.5

For period one: for which r does one step take 0.5 straight back to 0.5? The step is $r \times 0.5 \times 0.5$.

Answer it before you read on.

Answer. $r = 2$. Half of a half is a quarter, and $2 \times 0.25 = 0.5$.

For longer periods there is no mental shortcut, and bisection needs none. `from_top` reports how far from 0.5 the orbit ends, which is positive on one side of a landmark and negative on the other:

code/ch08/superstable.py

```

20 def from_top(f: Rule, steps: int) -> float:
21     """Start at the top of the hump; how far from it are you after steps?"""
22     x = 0.5
23     for _ in range(steps):
24         x = f(x)
25     return x - 0.5
26
27
28 def bisect(g: Callable[[float], float], lo: float, hi: float,
29           rounds: int = 60) -> float:
30     """A root of g between lo and hi, where g changes sign: halve and keep
31     the half in which the sign still changes."""
32     below = g(lo) < 0
33     if g(hi) < 0 == below:
34         raise ValueError("g must change sign between lo and hi")
35     for _ in range(rounds):
36         mid = (lo + hi) / 2
37         if g(mid) < 0 == below:
38             lo = mid
39         else:
40             hi = mid
41     return (lo + hi) / 2

```

Listing 8.3: Start at the top, and halve the bracket until the orbit comes back to it

The first two landmarks are bracketed by eye from the picture. After that each gap is smaller than the last, so the next landmark is sought between a tenth and a half of the last gap further on, a generous bracket for gaps shrinking by a factor near five. A *family* is a function that, given r , builds the rule.

code/ch08/superstable.py

```

46 def returns(family: Family, steps: int) -> Callable[[float], float]:
47     """The function of r whose root is the landmark with this period."""
48     return lambda r: from_top(family(r), steps)
49
50
51 def landmarks(family: Family, first: tuple[float, float],
52              second: tuple[float, float], count: int,
53              base: int = 1) -> list[float]:
54     """The landmarks for periods base, 2*base, 4*base, ... `first` and
55     `second` bracket the first two; each later one is sought between a
56     tenth and a half of the last gap beyond the last landmark found."""
57     found = [bisect(returns(family, base), *first),
58             bisect(returns(family, 2 * base), *second)]
59     for n in range(2, count):
60         gap = found[-1] - found[-2]
61         found.append(bisect(returns(family, base * 2 ** n),
62                             found[-1] + gap / 10, found[-1] + gap / 2))
63     return found
64

```

```

65 LOGISTIC = (logistic, (1.5, 2.5), (3.0, 3.4))
66

```

Listing 8.4: The landmarks for periods 1, 2, 4, 8, . . . , each bracketed from the last

The listing finds the first 11 landmarks, for periods 1 to 1024, with nothing but +, −, × and ÷, so every digit is the same on every computer:

n	period	landmark	R_n	gap	ratio
0	1	2.0000000000			
1	2	3.2360679775	1.2360679775		
2	4	3.4985616993	0.2624937218	4.7089	
3	8	3.5546408628	0.0560791634	4.6808	
4	16	3.5666673799	0.0120265171	4.6630	
5	32	3.5692435316	0.0025761518	4.6684	
6	64	3.5697952937	0.0005517621	4.6690	
7	128	3.5699134654	0.0001181717	4.6692	
8	256	3.5699387742	0.0000253088	4.6692	
9	512	3.5699441946	0.0000054204	4.6692	
10	1024	3.5699453555	0.0000011609	4.6692	

Look down the last column, each gap divided by the next. It wobbles for a few rows and then stops moving: 4.6692. That is *Feigenbaum's constant*, written δ (the Greek letter delta). The landmarks give it far more sharply than the splits did, because at a landmark the orbit settles at once instead of barely at all. Splits and landmarks take turns along the road, one of each per period, so both crowd in at the same rate towards the same end, $r = 3.5699$. The period-two landmark, 3.2361, is exactly $1 + \sqrt{5}$, where the note's product $4 + 2r - r^2$ is zero.

Feigenbaum published in 1978; Pierre Couillet and Charles Tresser in France found the same number independently at about the same time. What he did next is why the number is famous.

Lab 8.1 — Find a landmark yourself

Write `from_top` and `bisect`, and put them together in `superstable`. Check your answer for period two against $1 + \sqrt{5}$. Then find the r between 3.82 and 3.84 at which 0.5 comes back to itself after *three* steps. You will meet that number again.

Starter: `code/labs/ch08/l08_01_superstable.py`

Test: `code/labs/ch08/test_l08_01_superstable.py`

Run: `cd code && uv run pytest -k l08_01_superstable`

8.4 A number that belongs to the road

Feigenbaum's next step was to try a different rule, the *sine map*

$$x_{n+1} = s \sin(\pi x_n),$$

which returns s times the sine of πx . You need no trigonometry to follow it: for x between 0 and 1, the sine of πx is a smooth hump rising from 0 to 1 at $x = 0.5$ and falling back to

0. So the sine map shares the logistic map's shape — one hump, its top at 0.5 — and nothing of its formula: no $x(1 - x)$, a different knob s , a curve that is not a parabola.

Stop and think 8.6

Run the same landmark search on the sine map. Do you expect its gaps to shrink by the logistic map's 4.6692 again, or by some other number that belongs to the sine?

Answer it before you read on.

Answer. The same number. The sine map's gaps shrink by 4.05 at first and then close in, landmark by landmark, on 4.67, where the logistic map's ratios settle too.

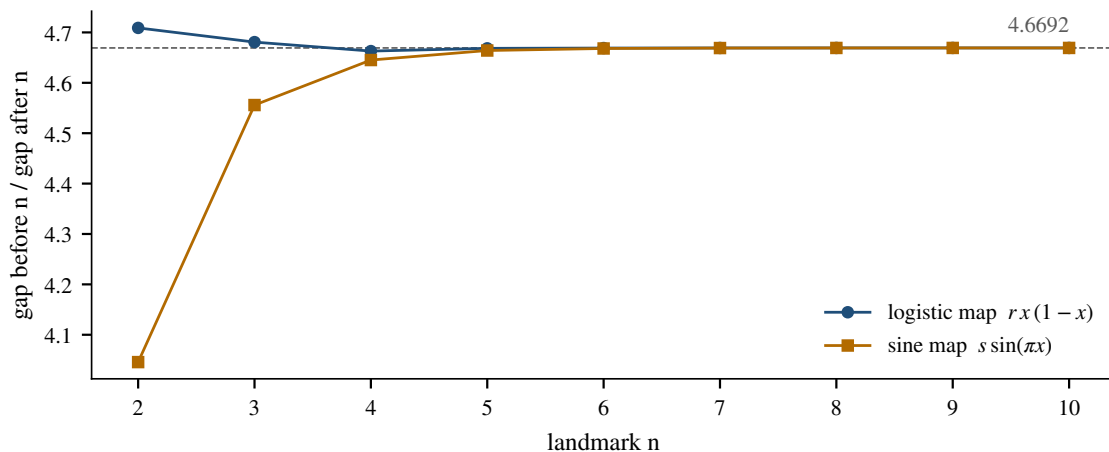


Figure 8.2: Each gap between landmarks divided by the next, for two rules with nothing in common but a single smooth hump. They start apart and end on the same number.

The listing knows nothing about the sine map but its family and two brackets read off its diagram:

code/ch08/two_maps.py

```

17 SINE = (sine_map, (0.4, 0.6), (0.72, 0.84)) # brackets read off its diagram
18
19
20 def main() -> None:
21     a = landmarks(*LOGISTIC, count=11)
22     b = landmarks(*SINE, count=11)
23     print(" n   logistic   sine map")
24     for n, (x, y) in enumerate(zip(ratios(a), ratios(b), strict=True), 2):
25         print(f"{n:2d} {x:9.2f} {y:9.2f}")
26     print(f"last{a[-1]:9.4f} {b[-1]:9.4f}")

```

Listing 8.5: The same search on two unrelated rules

n	logistic	sine map
2	4.71	4.05
3	4.68	4.56
4	4.66	4.65
5	4.67	4.66
6	4.67	4.67
7	4.67	4.67
8	4.67	4.67
9	4.67	4.67
10	4.67	4.67
last	3.5699	0.8656

The two roads end in different places (the last row: 3.5699 and 0.8656) and start differently, yet they settle on the same ratio to every digit printed. The function `ratios` divides each gap by the next; the second lab asks you to write it.

The trap

‘A different equation gives a different route.’ It gives different signposts — the splits sit at different values of the knob — but the road has the same shape, and its splits crowd together at the same rate. That number belongs to the route, not to the equation. This is *universality*, and it is why a constant of nature can come out of a toy model.

Why? Each new split is decided by a smaller and smaller stretch of the map near the top of its hump, seen through a stronger and stronger magnifying glass. Magnify any smooth, rounded hump enough around its top and it looks like a parabola; whatever made the sine map a sine has been magnified out of sight. The constants of school geometry, like π , belong to shapes. This one belongs to a process.

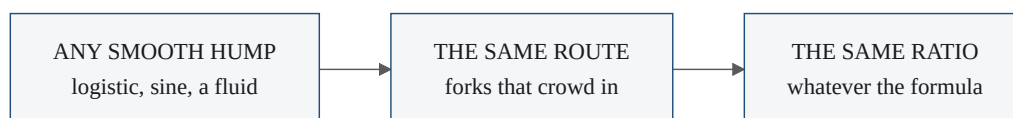


Figure 8.3: Universality in one line: any rule with one smooth hump takes the same road, and the road carries its own ratio.

It was a shock because it predicted something about systems whose equations nobody could write down: if one reaches chaos by period doubling, its splits should crowd together at Feigenbaum’s rate, whatever it is made of.

Where you meet this

Around 1980 Albert Libchaber and Jean Maurer in Paris heated a tiny box of liquid helium from below and recorded the temperature at one point as the fluid churned. As they turned up the heat, the temperature’s rhythm doubled its period, and doubled again. Later experiments on convecting fluids and electronic circuits measured the ratio of successive gaps and found numbers close to Feigenbaum’s: not to four

decimals, since a laboratory can resolve only the first few doublings, but close enough that nobody takes it for a coincidence. In 1986 Feigenbaum and Libchaber shared the Wolf Prize in physics.

What we are not proving

Why every smooth hump gives the same δ is not proved here. Feigenbaum explained it with a method from physics, *renormalisation*, which compares the map after each doubling with a rescaled copy of itself; Oscar Lanford gave a computer-assisted proof in 1982. *Smooth* matters: a hump whose top is flatter than a parabola's gives a different constant.

Lab 8.2 — Feigenbaum's ratio, and where the doublings end

Write `ratios`, which divides each gap by the next, and `where_it_ends`, which adds up every gap still to come, assuming each is the one before divided by the last ratio. The test feeds your functions the landmarks of both maps: one ratio, two different ends.

Starter: `code/labs/ch08/l08_02_ratio.py`

Test: `code/labs/ch08/test_l08_02_ratio.py`

Run: `cd code && uv run pytest -k l08_02_ratio`

8.5 Order inside the chaos

Past $r = 3.5699$ the doublings are used up, and the diagram turns to smear.

Stop and think 8.7

From $r = 3.5699$ to $r = 4$, is the logistic map chaotic at every setting of the knob, the way it is at 3.9?

Answer it before you read on.

Answer. No. Look at the picture again: the smear is streaked with pale stripes, and in each one the orbit is back on a short cycle. The widest, near 3.83, is a window of period three.

`code/ch08/windows.py` walks r across that stripe with the same `settled_period`, then runs the landmark search inside it:

r	period
3.82	–
3.8284	–
3.8285	3
3.83	3
3.84	3
3.845	6

```

3.848      12
3.85       -
the window opens at  $1 + \sqrt{8} = 3.82843$ 
periods 3, 6, 12, ...: last ratio 4.669

```

At 3.8284 the orbit never repeats; at 3.8285 it has settled into a three-cycle. The window opens between the two, at exactly $r = 1 + \sqrt{8} = 3.82843$. The three-cycle then doubles to six and twelve, and by 3.85 it has dissolved into smear again. The whole road repeats in miniature. The window's first landmark is the r you found in the first lab, 3.8319, and its landmarks crowd together at a ratio of 4.669: Feigenbaum's number once more, now from periods 3, 6, 12, ... instead of 1, 2, 4, ...

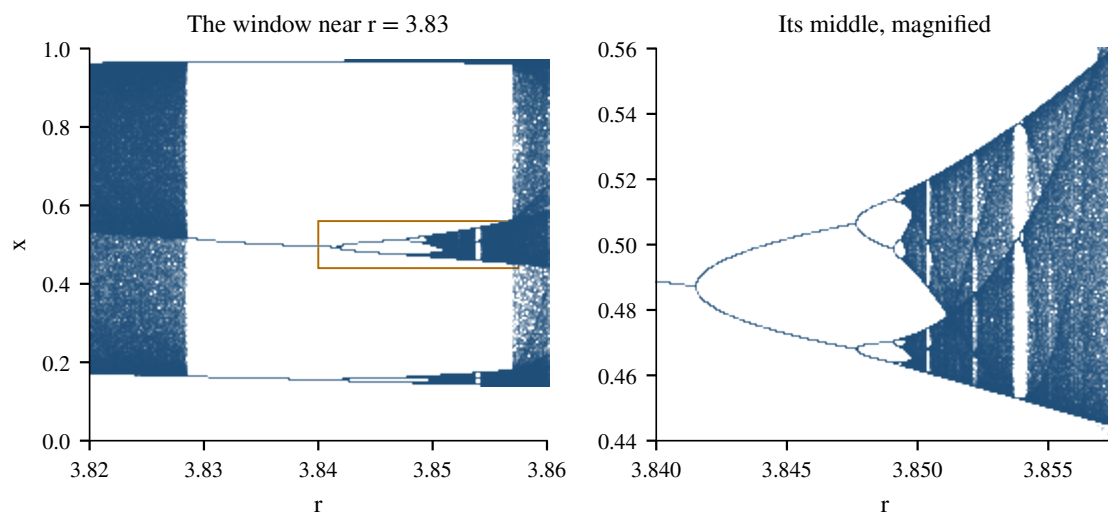


Figure 8.4: Left: the period-three window. Right: the window's middle branch, magnified: a small copy of the whole bifurcation diagram, with its own cascade, its own smear and its own windows.

The trap

‘Once a system has reached chaos, it is chaotic everywhere beyond that point.’

Between $r = 3.5699$ and 4 the logistic map returns to order infinitely many times, in windows of every period. Turn the knob a little and a chaotic system can become perfectly periodic; turn it a little further and it is chaotic again. Chapter 7's exponent dips below zero in exactly these stripes.

Note

In 1975 Tien-Yien Li and James Yorke published a paper whose title was its result: *Period three implies chaos*. Paraphrased: if a continuous rule takes an interval into itself, as the logistic map does from 0 to 1, and some start comes back to itself after exactly three steps, then the rule has cycles of every length, and starts whose orbits never settle into any cycle. The paper gave the word *chaos* its mathematical meaning; a stronger result had been proved in 1964 by Oleksandr Sharkovsky in Kyiv, little known in the West until later. In the window almost every start ends on

the three-cycle, but the theorem still holds: the other cycles are there, unstable, and an orbit may wander among them for a long time before settling.

The diagram has one more surprise, and it belongs to Chapter 12: this whole picture lies along a line through the most famous image in mathematics, with the splits and the window exactly where a one-line conversion puts them.

Summary

- The **bifurcation diagram** shows, for each value of the knob, where the orbit lives once the transient is over. A **bifurcation** is a value at which the long-run behaviour changes kind; on this road each is a **period doubling**.
- The splits crowd together, each gap a little under five times the next, so infinitely many fit before $r = 3.5699$. Measured by watching, a split comes out early, because near it the orbit settles very slowly.
- At a **superstable** landmark the cycle passes through the top of the hump, where the slope is zero. Each is one equation, solved by **bisection** with $+$, $-$, $\times$ and $\div$, and the ratio of their gaps settles on **Feigenbaum's constant**, 4.6692.
- The sine map gives the same ratio: the number belongs to the route every smooth one-hump rule takes, not to a formula. That is **universality**, and it has been measured in fluids and circuits.
- Beyond the cascade, chaos is interrupted by **windows** of order. The widest, of period three, opens at $1 + \sqrt{8}$ and holds a miniature of the whole road; Li and Yorke proved that period three implies chaos.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

read a bifurcation diagram and say what the map does at any value of the knob it shows	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
locate a period doubling by watching runs of the map, and say what limits how precisely that can be done	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
find a superstable parameter by bisection, using nothing but $+$, $-$, $\times$ and $\div$	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
compute the ratio of successive gaps between doublings, and estimate where the doublings run out	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
run the same search on a second rule and say what the comparison shows	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
find a window of order inside the chaotic range, and state what Li and Yorke proved about period three	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. What does the column of dots above one value of r in a bifurcation diagram show, and what does it look like for a fixed point, a two-cycle and chaos?
2. Two successive gaps between splits are 0.08 and 0.017. Roughly what is their ratio, and how long should the next gap be?
3. Why does a split located by watching the orbit come out slightly early?
4. What is a superstable landmark, and why can it be found to many digits with $+$, $-$, $\times$ and $\div$ alone?
5. The sine map's doublings run out at about 0.8656, the logistic map's at about 3.5699. Does that contradict universality?
6. Is the logistic map chaotic at $r = 3.83$? What did Li and Yorke prove about period three?

Part III

The shapes of chaos

Three equations for the weather: the Lorenz attractor

Put a wide pan of water on a gentle flame. The warm water at the bottom wants to rise and the cool water at the top wants to sink, and past a certain heat the water starts to roll: up on one side, down on the other, like a slow wheel. Air does the same over a sunny field, and on a grander scale that rolling is one of the engines of the weather.

Describe that roll with three numbers and three short rules, exact, with nothing random in them. Will it turn steadily, like a wheel? Or can three numbers be as irregular as the sky? Part II found chaos in rules that take one step at a time. This chapter finds it in something that flows — and then finds, inside the chaos, something nobody expected: a shape.

What you will be able to do

When you have finished this chapter you will be able to:

- read Lorenz's three rules of change term by term, and say what each term does to the roll
- follow the rules with RK4 and read the result from two views: one number against another, and one number against time
- measure how fast two nearly identical runs separate, and turn that rate into a prediction horizon
- say what an attractor is, and why a chaotic system cannot go anywhere it likes
- compute a statistic of a run and check that it does not depend on where the run began

9.1 Three numbers for a pan of water

In 1963 Edward Lorenz, a meteorologist at the Massachusetts Institute of Technology, published a paper called 'Deterministic Nonperiodic Flow'. Its title is this chapter in miniature. *Deterministic*: the rules contain no chance. *Nonperiodic*: the motion never repeats. *Flow*: time runs smoothly, so the rule gives the speed at which the state changes rather than its next value, as in Chapter 4. His model is a drastic cut-down of a fluid heated from below, and it keeps three numbers:

- x : how fast the roll turns, and which way (positive one way, negative the other);
- y : how much warmer the rising side is than the sinking side;

- z : how far the temperature, from bottom to top, has been bent away from an even slope by the roll's mixing.

Write dx/dt , read 'the rate of x ', for how fast x is changing right now; it is a name for a speed, not a fraction to simplify. Lorenz's rules are

$$\frac{dx}{dt} = \sigma (y - x), \quad \frac{dy}{dt} = x (\rho - z) - y, \quad \frac{dz}{dt} = x y - \beta z,$$

with three fixed numbers named by Greek letters: σ (sigma) is 10, ρ (rho) is 28 and β (beta) is $\frac{8}{3}$. In words:

- **The roll chases the temperature difference.** When y is bigger than x the roll speeds up, and when it is smaller the roll slows; σ says how briskly.
- **The roll builds the difference, and the difference leaks.** Multiplied out, the second rule is $\rho x - xz - y$. A turning roll carries warm water up one side and cool water down the other, building y at the rate ρx , where ρ is how hard the pan is heated ($\rho = 1$ is the least heating that starts a roll at all). The $-y$ is the difference leaking away; the $-xz$ is the mixing fighting back.
- **The roll stirs the layer, and the layer settles.** A roll carrying warm water upwards (x and y of the same sign) bends the temperature profile at the rate xy ; left alone it relaxes at the rate βz , where β comes from the shape of the rolls.

Only two terms, xz and xy , multiply two of the numbers together; every other term is a fixed number times one of them. Those two products are the whole of the model's nonlinearity, in Chapter 3's sense, and they will turn out to be enough.

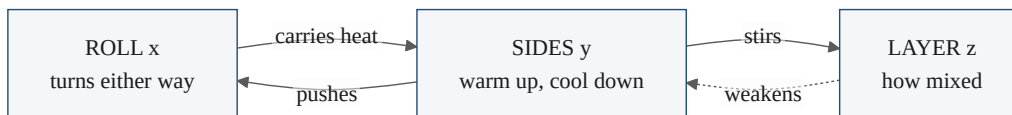


Figure 9.1: The three numbers push one another round a loop. The roll builds the temperature difference, the difference drives the roll, and together they stir the layer, which weakens the difference.

Stop and think 9.1

Suppose the water is perfectly still: $x = 0$, $y = 0$ and $z = 0$. What do the three rules say?

Answer it before you read on.

Answer. Every rate is zero: $10 \times (0 - 0) = 0$, then $0 - 0 = 0$, then $0 - 0 = 0$. Nothing changes.

That is the pan passing its heat quietly upwards with no roll at all. A state where every rate is zero is a *fixed point* of the flow, in Chapter 2's words: start there and you stay there. It is not the only one.

Stop and think 9.2

Try $x = y = \sqrt{72}$, which is about 8.49, and $z = 27$. Are all three rates zero? You need only $\frac{8}{3} \times 27 = 72$.

Answer it before you read on.

Answer. Yes. The first rate is $10 \times 0 = 0$. The second is $x \times (28 - 27) - y = x - y = 0$. The third is $x \times x - 72 = 72 - 72 = 0$.

This is a *steady roll*, turning at a constant speed with a constant temperature difference. With minus signs, $x = y = -\sqrt{72}$ and $z = 27$, it is the same roll turning the other way. For any heating ρ above 1 the two steady rolls sit at $x = y = \pm\sqrt{\beta(\rho - 1)}$ and $z = \rho - 1$, where $\pm$ means ‘plus or minus’.

Lab 9.1 — Lorenz’s rules, in your own hand

Write `rates`, which takes the state (x, y, z) and returns the three rates, and `fixed_points`, which returns every state where all three are zero. The test checks your rates against the book’s library, and checks that they are zero at every point you return.

Starter: `code/labs/ch09/l09_01_rates.py`

Test: `code/labs/ch09/test_l09_01_rates.py`

Run: `cd code && uv run pytest -k l09_01_rates`

9.2 Following the flow

Here are the same rules as the book’s library writes them. The last three lines are the three rules; `lorenz` hands back a function that takes a state and returns its three rates, which is what your lab asked for.

`code/src/chaoslab/systems.py`

```

25 def lorenz(sigma: float = 10.0, rho: float = 28.0,
26         beta: float = 8.0 / 3.0) -> Field:
27     """Lorenz's 1963 convection model: three numbers, three rules."""
28
29     def field(s: State) -> State:
30         x, y, z = s
31         return (sigma * (y - x),
32                 x * (rho - z) - y,
33                 x * y - beta * z)
34
35     return field

```

Listing 9.1: Lorenz’s 1963 rules in chaoslab

To follow them, use Chapter 4’s RK4: `integrate` takes steps of length `DT`, a hundredth of a time unit, and keeps every state. The time unit is the model’s own, not a second or an hour.

code/ch09/lorenz_run.py

```
13 DT = 0.01 # one step is a hundredth of a time unit
14 field = lorenz(sigma=10.0, rho=28.0, beta=8.0 / 3.0)
15 path = integrate(field, (1.0, 1.0, 1.0), DT, 4000) # forty time units
```

Listing 9.2: Forty time units of Lorenz’s flow, from the start $(1, 1, 1)$

Stop and think 9.3

A pendulum with friction spirals into its resting point. Lorenz's pan has three fixed points. Started at $(1, 1, 1)$, close to the still water, will the run settle onto one of them?

Answer it before you read on.

Answer. No. It is pushed away from the still water, circles one steady roll in widening loops, is flung across to circle the other, and never settles.

At this heating neither steady roll can hold the state; they can only while ρ is below $\frac{470}{19}$, about 24.74, a number that comes out of the rules by algebra this book does not do. Turn the heating down to 20 and the same start does settle: the last line of the printout is a steady roll at $x = y = -7.12$, $z = 19$.

time	x	y	z	wing
0	1.00	1.00	1.00	R
2	-8.17	-9.56	24.62	L
4	-10.09	-10.93	27.97	L
6	-9.74	-7.71	30.89	L
8	-7.01	-4.59	28.39	L
10	-4.90	-3.74	24.69	L
12	-3.34	-1.98	23.60	L
14	8.75	16.25	12.39	R
16	-6.58	-10.81	15.86	L
18	0.84	-0.85	22.29	R
20	13.47	12.69	34.39	R

loops: LLLLLLLLLLLLLLLLLLLLLLRLLLRLRRRLRLRLLRLRRRRLLRLRLLLLL

rho = 20, after 100 time units: -7.12, -7.12, 19.00

The line marked loops prints one letter per loop: L for a loop on the side where x is negative, R for the other. The run makes 19 loops on the left before its first on the right, and after that the letters come in runs of anything from one to five, with no pattern. There is none to find: that is the *nonperiodic* of Lorenz's title.

The left half of Figure 9.2 has earned the name *Lorenz butterfly*. Each wing is the roll turning one way; crossing between them is the roll reversing.

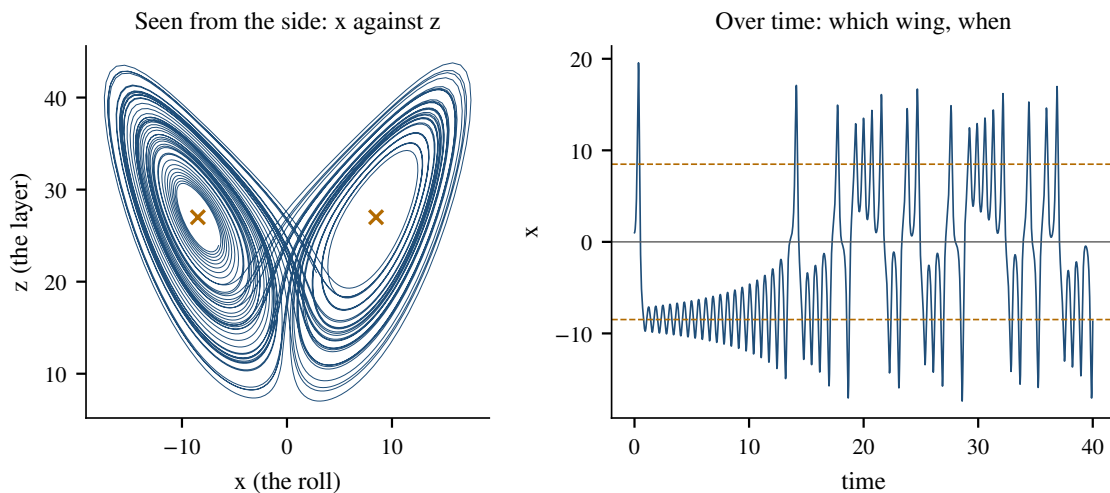


Figure 9.2: The run from $(1, 1, 1)$. Left: x against z from time 5 to time 60, looping round the two steady rolls (crosses) and tracing two wings. Right: x against time for the first forty time units; the dashed lines are the steady rolls.

Stop and think 9.4

Weather services today run models carrying many millions of numbers. Lorenz's carries three. Is three enough for real chaos, or only for a sketch of it?

Answer it before you read on.

Answer. Three is enough, and three is the fewest a flow can do it with.

Chapter 4 stated the lower limit; here is the reason. In a plane a path can never cross itself, because two paths through one point would have the same future; so a flow with two numbers can only settle, spiral or go round a fixed loop. With a third number the path can pass over and under itself, and that is room enough.

The trap

'Chaos needs many variables.' It needs three numbers and two multiplications. The weather is not hard to forecast only because the atmosphere is big: a pan of water cut down to three numbers is already unpredictable. What chaos does need is Chapter 10's subject.

Where you meet this

The *chaotic waterwheel* is Lorenz's pan built in wood and water: leaky cups hang round the rim of a wheel and water pours into the cups at the top. Willem Malkus and Louis Howard built one at MIT, and its equations, after renaming the quantities, are Lorenz's: the wheel turns, slows and reverses at moments nobody can call in advance. The equations of a certain kind of laser are Lorenz's too, as Hermann Haken showed in 1975.

9.3 Two starts a hair apart

Chapter 6 measured sensitivity on a map. Here is the same measurement on a flow. The run first goes fifty time units, to leave its start behind; then it is copied, and the copy is nudged by one part in a hundred million in x .

code/ch09/two_runs.py

```

14 DT = 0.01
15 field = lorenz()
16 a = integrate(field, (1.0, 1.0, 1.0), DT, 5000)[-1] # fifty time units
17 b = (a[0] + 1e-8, a[1], a[2]) # the copy, 0.00000001 further in x

```

Listing 9.3: Two runs, one part in a hundred million apart

Both are followed with the same rule, and the program prints the x of each and their distance apart every two time units:

time	x of A	x of B	apart
0	-3.66	-3.66	1.0e-08
2	-2.90	-2.90	6.9e-08
4	-5.28	-5.28	1.1e-06
6	-0.62	-0.62	1.4e-06
8	-15.34	-15.34	4.8e-05
10	2.80	2.80	2.0e-04
12	2.39	2.39	4.6e-04
14	-11.84	-11.84	5.8e-03
16	-2.13	-2.11	3.1e-02
18	3.74	3.62	2.5e-01
20	9.63	9.01	1.2e+00
22	-1.57	-0.08	4.2e+00
24	-12.07	8.35	2.6e+01

first more than 1 apart at time 19.2

Up to time 14 the two runs agree to two decimals; by time 24 they are on different wings. The distance grows about tenfold every two or three time units, though not smoothly: it swells and shrinks as the pair goes round a loop. It first passes 1 at time 19.2.

Chapter 7 named that rate the Lyapunov exponent, λ (lambda): the average of the logarithm of how much a small gap is stretched. For a flow the average is per unit of time. The library measures it so:

code/src/chaoslab/measure.py

```

77 def lyapunov_flow(field: Field, state: State, dt: float, steps: int,
78                  delta: float = 1e-8, burn: int = 1000,
79                  stepper: Stepper = rk4_step) -> float:
80     """The largest Lyapunov exponent of a flow, per unit of time.
81
82     Follow two runs delta apart; after every step measure how far apart
83     they have drifted, add the log of the growth, and pull the second run
84     back to distance delta along the same direction. Pulling back is what
85     keeps the gap small enough to measure the stretching rather than the
86     size of the attractor.
87     """

```

```

88 for _ in range(burn):
89     state = stepper(field, state, dt)
90 other = (state[0] + delta,) + tuple(state[1:])
91 total = 0.0
92 for _ in range(steps):
93     state = stepper(field, state, dt)
94     other = stepper(field, other, dt)
95     d = _distance(state, other)
96     total += math.log(d / delta)
97     other = tuple(s + (o - s) * delta / d
98                  for s, o in zip(state, other, strict=True))
99 return total / (steps * dt)

```

Listing 9.4: The Lyapunov exponent of a flow

Follow two runs a hair apart. After each step, add up the logarithm of how much their gap has grown, then pull the second run back to its first small distance, in the same direction, so that the gap never grows to the size of the butterfly, where it would stop measuring stretching. The total, divided by the time elapsed, is λ . Here it is from three starts:

code/ch09/exponent.py

```

16 field = lorenz()
17 for start in STARTS:
18     lam = lyapunov_flow(field, start, 0.01, 50_000) # 500 time units
19     print(f"from {start}: {lam:.2f} per time unit")

```

Listing 9.5: The exponent of Lorenz's flow, from three starts

```

from (1.0, 1.0, 1.0): 0.92 per time unit
from (-5.0, 3.0, 20.0): 0.91 per time unit
from (10.0, -10.0, 30.0): 0.91 per time unit

```

Between 0.91 and 0.92 per time unit, from three quite different starts: call it 0.9. On average the gap is multiplied by e to the power 0.9, about 2.5, every time unit.

Stop and think 9.5

Chapter 7's horizon: an error grows to a tolerance in about $\ln(\text{tolerance}/\text{error})/\lambda$. With an error of 10^{-8} , a tolerance of 1 and $\lambda = 0.9$, how long should two runs stay within 1 of each other? You need $\ln(10^8)$, which is about 18.4.

Answer it before you read on.

Answer. About $18.4/0.9$, which is roughly 20 time units. The pair above took 19.2.

One pair is one sample. Split the run at 50 different places along the butterfly and the first passage past 1 comes anywhere from time 16 to time 25, averaging 20. The formula predicts the average; the scatter is there because the stretching is not the same all round the wings.

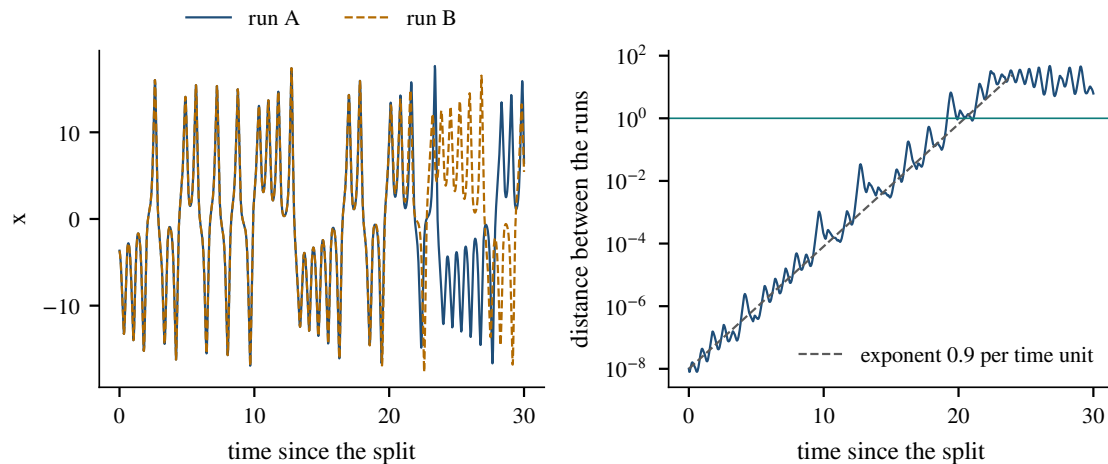


Figure 9.3: Two runs that start one part in a hundred million apart. Left: their x agree, then part company. Right: their distance, on a logarithmic scale, climbs along the dashed line of the exponent until it reaches the size of the butterfly.

Nothing in the rule is uncertain, and yet a forecast from a start known to eight decimal places is good for about twenty time units. (Two runs that have drifted apart can be made to agree again by letting one feel the other: that is Chapter 17's story.)

9.4 Unpredictable, but not anywhere

If a run cannot be forecast beyond twenty time units, what can be said about it at all?

Stop and think 9.6

By time 24 the two runs above have nothing to do with each other: one has x near -12 , the other near 8 . Let them run a thousand time units more. Could one of them turn up at $x = 100$, or at $z = -50$?

Answer it before you read on.

Answer. No. Over 20 000 time units, x never gets further than 20 from zero either way, and z stays between 1 and 48. That holds from wherever you start.

Test the 'from wherever' with eight starts at the corners of a box eighty wide, far outside anything in the pictures above:

code/ch09/cloud.py

```
12 field = lorenz()
13 corners = [(x, y, z) for x in (-40.0, 40.0) for y in (-40.0, 40.0)
14             for z in (-40.0, 80.0)]
```

Listing 9.6: Eight starts far outside the butterfly

The program follows each for fifty time units, ignores the first five, and prints the room each run lived in:

start	largest $ x $	lowest z	highest z
(-40, -40, -40)	18.7	3.8	46.0
(-40, -40, 80)	18.2	5.2	45.0
(-40, 40, -40)	18.4	4.7	45.4
(-40, 40, 80)	17.7	6.5	44.1
(40, -40, -40)	18.4	4.7	45.4
(40, -40, 80)	17.7	6.5	44.1
(40, 40, -40)	18.7	3.8	46.0
(40, 40, 80)	18.2	5.2	45.0

Every run is pulled in from its corner and then lives in the same room as the long run. Notice, too, that the rows come in identical pairs: $(-40, -40, -40)$ and $(40, 40, -40)$ give the same numbers. The next section says why.

The trap

‘A chaotic system can go anywhere.’ Unpredictable is not unbounded. You cannot say where on the butterfly a Lorenz run will be in a hundred time units, but you can say it will be on the butterfly, and that is a real prediction: a thin set of states out of all the ones the three numbers could take.

A set of states that runs are drawn onto, and then never leave, is an *attractor*. A pendulum with friction has a single point for its attractor; a pendulum clock, kept swinging, has a loop. Lorenz’s has the butterfly, the same one from almost every start, and a run on it never repeats and never leaves. How a path can loop for ever in a bounded region without crossing or retracing itself is a puzzle about the shape, which is why it is called a *strange attractor*. Chapter 10 opens the shape up, and Chapter 11 measures shapes like it.

What we are not proving

Two things are shown here by running, not proved. That no run can escape can be proved with a page of algebra about a quantity that shrinks whenever the state is far out. That the butterfly the computer draws really exists, and really is chaotic, rather than being an artefact of stepping and rounding, stayed open for more than thirty years, until Warwick Tucker settled it with a proof in which a computer did the arithmetic with every rounding error accounted for.

9.5 What you can still predict

The butterfly says where a run lives. The next question is how it spends its time there.

Stop and think 9.7

You cannot say which wing a run will be on at time 50. Can you say what fraction of a long run it spends on the right-hand wing, where x is positive?

Answer it before you read on.

Answer. Yes: a half. Change the signs of x and y together: the first two rates change sign with them, and the third, $xy - \beta z$, does not change at all, because $(-x)(-y) = xy$.

So the rules do not care which way the roll turns. Every run on one wing has a mirror-image run on the other, which is why the cloud's rows came in pairs, and a long run shares its time between the wings equally.

The trap

'If the path is unpredictable, nothing about it is predictable.' The path is unpredictable beyond about twenty time units. The shape it lives on is not, and neither is how much of its time it spends in each part of the shape. Those are *statistics of the attractor*, and they come out the same from every start.

Lab 9.2 — Two numbers the start does not decide

Write `fraction_right`, the share of a run's states with x above zero, and `mean_height`, the average of its z . The test runs Lorenz's rules from $(1, 1, 1)$ and from a far corner of the cloud, and checks that your two summaries agree between the runs.

Starter: `code/labs/ch09/l09_02_wings.py`

Test: `code/labs/ch09/test_l09_02_wings.py`

Run: `cd code && uv run pytest -k l09_02_wings`

Here are the book's versions:

`code/ch09/averages.py`

```

17 def fraction_right(path: list[tuple[float, ...]]) -> float:
18     """The share of the states with x positive: time on the right wing."""
19     return sum(1 for x, _, _ in path if x > 0) / len(path)
20
21
22 def mean_height(path: list[tuple[float, ...]]) -> float:
23     """The average of z over the run: how high the state lives."""
24     return sum(z for _, _, z in path) / len(path)

```

Listing 9.7: Two statistics of a run

The listing runs five starts for 500 time units each, drops the first ten so that each start is forgotten, and prints for each run which wing it is on at time 50, its fraction of time on the right wing, and its average height:

start	wing at 50	time on right	mean height
(1, 1, 1)	L	0.50	23.48
(-5, 3, 20)	L	0.49	23.52
(10, -10, 30)	R	0.50	23.50
(0, -1, 50)	L	0.49	23.51
(30, -30, 5)	R	0.50	23.57

The middle column is the unpredictable part: left, left, right, left, right, and nothing you could know in advance picks it. The last two columns are the predictable part. The fraction on the right is between 0.49 and 0.50 from every start, as the mirror said. The average height is between 23.48 and 23.57, and no symmetry told you that number: it is simply the same from every start.

A statistic of a finite run wobbles a little. Chop one run of 20 000 time units into pieces: in pieces 100 time units long, the fraction on the right typically strays from a half by about 0.05; in pieces 2000 long, by about 0.01; over the whole run it is 0.50. The longer you watch, the sharper the statistic — the opposite of the path, whose forecast gets worse the further ahead you look.

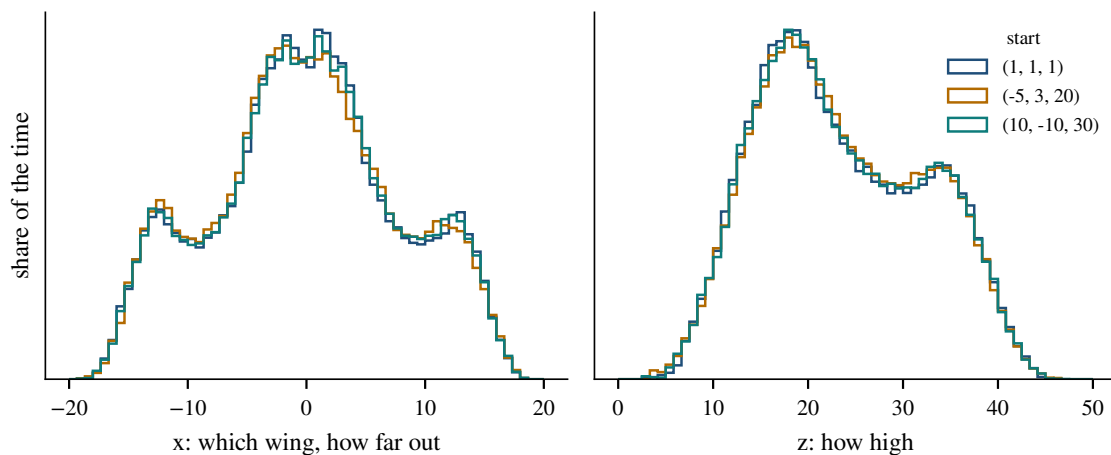


Figure 9.4: How much of its time the run spends at each value of x and of z , from three starts, 500 time units each. The three outlines lie on top of one another: the histogram does not care where the run began.

The whole histogram of where the run spends its time comes out the same from three starts. That is the chapter's hinge and the second big idea of this book: **the individual future of a chaotic system is unpredictable, but where the system lives, and how often it visits each part, is predictable.** Even λ is a number of that kind.

Chapter 14 builds on exactly this to separate a question about next week's weather from a question about climate, and shows how forecasters run many copies at once. Chapter 16 asks why a statistic can be trusted even though the computer's own rounding soon makes its path a different one from the true path.

Summary

- Lorenz's 1963 model of a heated pan keeps three numbers: the roll's speed x , the temperature difference y and the mixing z . Only two terms of its three **rules of change**, xy and xz , are **nonlinear**. Its **fixed points** are the still pan and two **steady rolls**.
- At $\rho = 28$ a run settles on no fixed point: it loops round one steady roll, then the other, switching irregularly and never repeating. Three numbers are the fewest a **flow** needs for chaos, and enough.
- Two runs one part in a hundred million apart separate in about twenty time units. The **Lyapunov exponent** of the flow is about 0.9 per time unit, and Chapter 7's horizon formula predicts the average separation time.
- Every run is drawn onto the same bounded, butterfly-shaped set: an **attractor**. A chaotic system cannot go anywhere it likes.
- **Statistics of the attractor**, such as the fraction of time on each wing and the average height, are the same from every start. The path is unpredictable; where the system lives is not.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

read Lorenz's three rules of change term by term, and say what each term does to the roll	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
follow the rules with RK4 and read the result from two views: one number against another, and one number against time	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
measure how fast two nearly identical runs separate, and turn that rate into a prediction horizon	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what an attractor is, and why a chaotic system cannot go anywhere it likes	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
compute a statistic of a run and check that it does not depend on where the run began	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. In words, what does the rule for the rate of x , $\sigma(y - x)$, make the roll do when y is larger than x ?

2. With $\rho = 28$, where are the three fixed points of Lorenz's rules?
3. Why can a flow with only two numbers not be chaotic, and how many does Lorenz's model use?
4. The exponent is about 0.9 per time unit. If the start were known a thousand times more precisely, how much longer would two runs stay within 1 of each other?
5. Someone says a Lorenz run, being chaotic, might one day reach $z = 200$. What do you tell them?
6. What fraction of a long run is spent on the right-hand wing, and how can you know without running anything?
7. Name a number about a Lorenz run that no symmetry predicts but that comes out the same from every start.

Stretch and fold: strange attractors

In Chapter 9 two runs of Lorenz's equations that began a hair apart drifted until they had nothing to do with each other, and yet neither ever left the butterfly-shaped region it started in. Put those two facts side by side and they look impossible. If nearby states are pushed apart at every step, why do they not end up at opposite ends of the universe? And if everything stays inside a small region, where does all that separation go?

The answer is something every baker does with their hands. This chapter shows it on a strip of dough, then on a two-line rule the astronomer Michel Hénon wrote down in 1976, and then follows it to a shape with no area, no end to its length, and layers inside its layers however closely you look.

What you will be able to do

When you have finished this chapter you will be able to:

- explain how motion can stay inside a region for ever and still be sensitive to its start, using the words stretch and fold
- iterate the tent map and Hénon's map in Python and read what they print
- work out how much one step of Hénon's map multiplies an area, and check it by measuring
- say what an attractor, its basin and a strange attractor are
- describe what zooming in shows on a strange attractor, and how that differs from zooming in on a smooth curve
- describe how a Poincaré section turns a flow into a map

10.1 Stretch, then fold

Put a lump of dough on a board and press two raisins into it side by side. Now knead: roll the dough out to twice its length, then fold the far half back on top of the near half. Do it again, and again. After a minute the raisins are nowhere near each other, and neither has left the board.

A knead is two moves, and they are the whole of this chapter. The *stretch* pulls every pair of neighbours further apart. The *fold* puts the stretched dough back into the space it came from. Stretching alone would carry the raisins off the board; folding alone would do nothing to them. Together they do what neither can.

To turn that into arithmetic, flatten the dough into a strip running from 0 to 1 and let x be a raisin's position along it. Stretching sends x to $2x$. Anything now past 1 is folded back, so a point that stuck out to $2x$ lands at $2 - 2x$, as far short of the end as it had gone

past it:

$$x_{n+1} = 2x_n \text{ if } 2x_n \leq 1, \quad x_{n+1} = 2 - 2x_n \text{ otherwise,}$$

where x_n is the raisin's position after n kneads. Its graph climbs in a straight line from 0 to 1 and comes straight back down, like a tent seen from the side, so it is called the *tent map*.

Stop and think 10.1

Two raisins start at 0.2345 and 0.2355, a thousandth of the strip apart. While both stay on the same side of the fold, every knead doubles the gap between them. If it went on doubling, after how many kneads would the gap be longer than the whole strip? What must happen instead?

Answer it before you read on.

Answer. After ten: $0.001 \times 2^{10} = 1.024$, longer than the strip. It cannot happen: two raisins on a strip of length one are never more than one apart, so before then the fold must catch one raisin and not the other.

code/ch10/kneading.py

```

12 def knead(x: float) -> float:
13     """One knead: stretch the strip to twice its length, then fold."""
14     stretched = 2.0 * x
15     if stretched <= 1.0:
16         return stretched          # still on the strip: nothing to fold
17     return 2.0 - stretched        # past the end: fold it back
18
19
20 def main() -> None:
21     a, b = 0.2345, 0.2355          # two raisins a thousandth apart
22     print("knead  raisin A  raisin B    gap")
23     for n in range(14):
24         print(f"{n:5d}  {a:8.4f}  {b:8.4f}  {abs(b - a):6.4f}")
25         a, b = knead(a), knead(b)

```

Listing 10.1: Kneading a strip of dough: stretch, then fold

knead	raisin A	raisin B	gap
0	0.2345	0.2355	0.0010
1	0.4690	0.4710	0.0020
2	0.9380	0.9420	0.0040
3	0.1240	0.1160	0.0080
4	0.2480	0.2320	0.0160
5	0.4960	0.4640	0.0320
6	0.9920	0.9280	0.0640
7	0.0160	0.1440	0.1280
8	0.0320	0.2880	0.2560
9	0.0640	0.5760	0.5120
10	0.1280	0.8480	0.7200
11	0.2560	0.3040	0.0480
12	0.5120	0.6080	0.0960
13	0.9760	0.7840	0.1920

The gap doubles nine times in a row, to 0.512. At the tenth knead one raisin is folded and the other is not, and the gap is 0.72. One knead later they are only 0.048 apart, by chance: scattered raisins can be any distance apart, and from now on knowing where one is tells you nothing about the other. Every knead multiplies every short gap by exactly 2, so this rule's Lyapunov exponent, in the sense of Chapter 7, is $\ln 2 \approx 0.69$ per knead.

The trap

'If a system can never leave a small region, it cannot be very sensitive to its start.'

The region limits how *big* an error can get, not how fast it grows. A tiny error doubles and doubles until it is as large as the region, and from then on the two runs are as unrelated as two raisins dropped on the board at random. Staying in the bowl and being predictable are different promises, and the fold is what lets a system keep the first without the second.

Lab 10.1 — A strip of dough

Write `knead`, one stretch and one fold. Then write `kneads_until_apart`, which counts the kneads before two raisins that start a small gap apart are more than a tolerance apart. For raisins a thousandth apart and a tolerance of 0.1 the answer is seven. Start them ten times closer and see how few extra kneads that buys.

Starter: `code/labs/ch10/l10_01_knead.py`

Test: `code/labs/ch10/test_l10_01_knead.py`

Run: `cd code && uv run pytest -k l10_01_knead`

10.2 Hénon's two-line map

The strip is one number. The Lorenz system of Chapter 9 has three, and it stretches and folds as a trajectory swings round a lobe, is pulled away from its neighbours and is thrown back into the middle. In 1976 Michel Hénon, an astronomer at the observatory in Nice, published a rule, about as simple as a rule can be, that does the same thing to a plane —

so simple that a computer can run it for millions of steps. It takes a point (x, y) , a state of two numbers, to a new point:

$$x_{n+1} = 1 - ax_n^2 + y_n, \quad y_{n+1} = bx_n,$$

with $a = 1.4$ and $b = 0.3$. In words: the new x is one, minus a times the square of the old x , plus the old y ; the new y is the old x shrunk to three tenths of its size.

Hénon built it from three moves. A *bend* moves every point up or down by $1 - ax^2$, a hump that sends x and $-x$ to the same height, so a straight line across the plane is creased into an arch: that is the fold. A *squeeze* presses the plane sideways to three tenths of its width. A *swap* exchanges the two numbers, so the next bend works across the last one.

Stop and think 10.2

Start at the origin, $(0, 0)$. Where is the point after one step of Hénon's map, and after two?

Answer it before you read on.

Answer. After one step, $(1, 0)$. After two, $x = 1 - 1.4 \times 1 + 0 = -0.4$ and $y = 0.3 \times 1$, so the point is $(-0.4, 0.3)$.

code/ch10/henon.py

```

11 A = 1.4    # how hard each step bends
12 B = 0.3    # how hard each step squeezes
13
14
15 def henon(x: float, y: float) -> tuple[float, float]:
16     """One step of Henon's map."""
17     return 1.0 - A * x * x + y, B * x

```

Listing 10.2: Hénon's map: two numbers, one line of arithmetic each

step	x	y
0	0.0000	0.0000
1	1.0000	0.0000
2	-0.4000	0.3000
3	1.0760	-0.1200
4	-0.7409	0.3228
5	0.5543	-0.2223
6	0.3476	0.1663
7	0.9972	0.1043
8	-0.2879	0.2992

The listing carries on where your hand stopped. The point jumps about without settling, repeating or running away: however long you let it go, x stays between -1.28 and 1.27 .

10.3 One shape for every start

One start is one story. Take many: a grid of forty by forty, 1600 starting points spread evenly over the square in which both x and y run from -0.5 to 0.5 .

Stop and think 10.3

Run every one of these points through Hénon's map for thirty steps. Where do you expect the cloud to be: still spread over its square, all gathered at a single point, or somewhere else?

Answer it before you read on.

Answer. Somewhere else. A few points leave for good, and all the rest lie on one thin, bent set — the set a single start draws if you run it long enough. They do not gather at a point, because the stretching keeps them apart along that set.

code/ch10/cloud.py

```

24 def marked_squares() -> set[tuple[int, int]]:
25     """Every little square that one long run of the map visits."""
26     x, y = 0.0, 0.0
27     for _ in range(1000):                # let it settle first
28         x, y = henon(x, y)
29     marked = set()
30     for _ in range(REFERENCE):
31         x, y = henon(x, y)
32         marked.add(square_of(x, y))
33     return marked
34
35
36 def main() -> None:
37     marked = marked_squares()
38     cloud = [(-0.5 + (i + 0.5) / 40, -0.5 + (j + 0.5) / 40)
39             for i in range(40) for j in range(40)]
40     start = len(cloud)
41     print("step left for good on the attractor")
42     for step in range(31):
43         if step in (0, 1, 2, 3, 5, 10, 20, 30):
44             on = sum(square_of(x, y) in marked for x, y in cloud)
45             print(f"{step:4d} {start - len(cloud):13d} "
46                   f"{on:6d} of {len(cloud)}")
47         cloud = [henon(x, y) for x, y in cloud]
48         cloud = [(x, y) for x, y in cloud if abs(x) + abs(y) < 10]

```

Listing 10.3: A cloud of starts, and whether each lands on the attractor

step	left for good	on the attractor
0	0	104 of 1600
1	0	237 of 1600
2	0	811 of 1600
3	0	1297 of 1600
5	139	1409 of 1461
10	172	1428 of 1428
20	172	1428 of 1428
30	172	1428 of 1428

To decide whether a point is on that set, the program first runs one start for 500 000 steps and marks every little square of side 0.01 that it visits. At the start only 104 of the 1600 points sit in a marked square. From step 10 on, all 1428 that remain do, and 172 have left for good, running off past ten and on towards infinity.

That is what the word *attractor* means: a set of states that the motion settles onto and never leaves, and that pulls in the states around it. The resting point of Chapter 4's damped pendulum is the simplest attractor; the butterfly of Chapter 9 is another. Every start that ends up on an attractor belongs to its *basin*, as every raindrop that reaches a lake fell in the lake's basin. Hénon's attractor has a basin with an edge, and starts outside it run off to infinity.

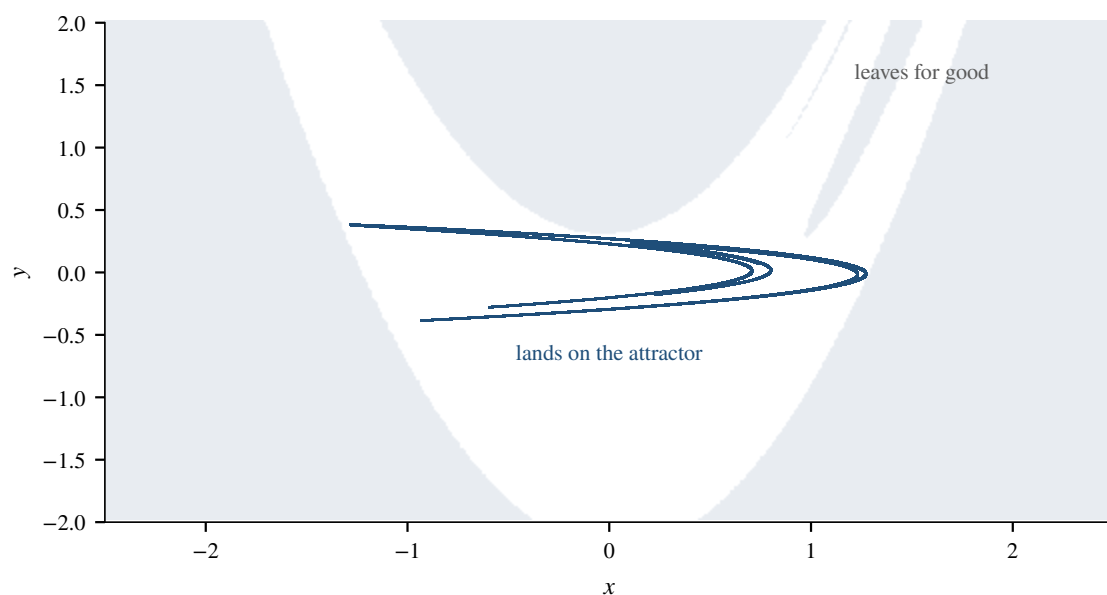


Figure 10.1: Hénon's attractor (blue) on its basin. Every start in the white region ends up on the attractor; every start in the grey region leaves for good. The top edge of the listing's square reaches into the grey, and that is where the starts that left began.

10.4 Squeezed, not stopped

The cloud began as a square with an area and ended on a set as thin as a line. Hénon's rule says exactly how fast the area goes. Take a tiny square of side h with a corner at (x, y) . One step moves its two sides, the arrows $(h, 0)$ and $(0, h)$, to two new arrows. Nudging x

by h changes the new x by about $-2ax h$, because $(x + h)^2 = x^2 + 2xh + h^2$ and h^2 is tiny next to h , and it changes the new y by bh . Nudging y by h changes the new x by h and the new y not at all. So the square becomes a slanted parallelogram with sides

$$(p, q) = (-2ax h, b h), \quad (r, s) = (h, 0).$$

The area of a parallelogram with sides (p, q) and (r, s) is the size of $ps - qr$, a number called the *determinant*; for a rectangle with sides $(2, 0)$ and $(0, 3)$ it gives $2 \times 3 - 0 \times 0 = 6$. Here it is $(-2ax h) \times 0 - (b h) \times h = -b h^2$, so the area h^2 becomes $b h^2$. Every step multiplies every area by $b = 0.3$, wherever the square was: the position x , which decides how much the square is stretched, is multiplied by zero and drops out.

Stop and think 10.4

A patch of starting points has an area of 1. What is its area after ten steps? Does that mean its points slow down and come to rest, as the damped pendulum of Chapter 4 did?

Answer it before you read on.

Answer. About 0.0000059, which is 0.3^{10} : six millionths of what it was. And no, the points do not slow down at all. The area is squeezed out of the patch; the motion is not.

The trap

‘If the area keeps shrinking, the motion must be dying out.’ A damped pendulum loses area because everything is pulled towards one resting point. Hénon’s map squeezes hard in one direction while it stretches in another, so a patch becomes a longer, thinner ribbon whose points go on moving for ever. Shrinking area says that the motion ends up on something thin, not that it stops.

Now measure it. A square a tenth on a side becomes sixteen thousand points round its edge; the map moves each one, and the surveyor’s shoelace formula measures the area inside the new outline.

code/ch10/area.py

```

45     edge = outline(0.1, 4000)                # a square a tenth on a side
46     area0, length0 = area(edge), length(edge)
47     print("step   area / start   0.3 ** step   outline / start")
48     for step in range(1, 7):
49         edge = [henon(x, y) for x, y in edge]
50         print(f"{step:4d}   {area(edge) / area0:13.5f}   "
51               f"B ** step:12.5f}   {length(edge) / length0:16.2f}")

```

Listing 10.4: Measuring what the map does to an area

step	area / start	$0.3 ** \text{step}$	outline / start
1	0.30000	0.30000	0.66
2	0.09000	0.09000	1.56
3	0.02700	0.02700	1.63
4	0.00810	0.00810	4.13
5	0.00243	0.00243	7.58
6	0.00073	0.00073	9.81

The area falls by the factor the determinant promised, to five decimals, down to 0.00073 of the start after six steps. The outline does the opposite: after first getting shorter, as the squeeze flattens the square, it keeps growing, and after six steps it is 9.8 times as long, wrapped round less than a thousandth of the area.

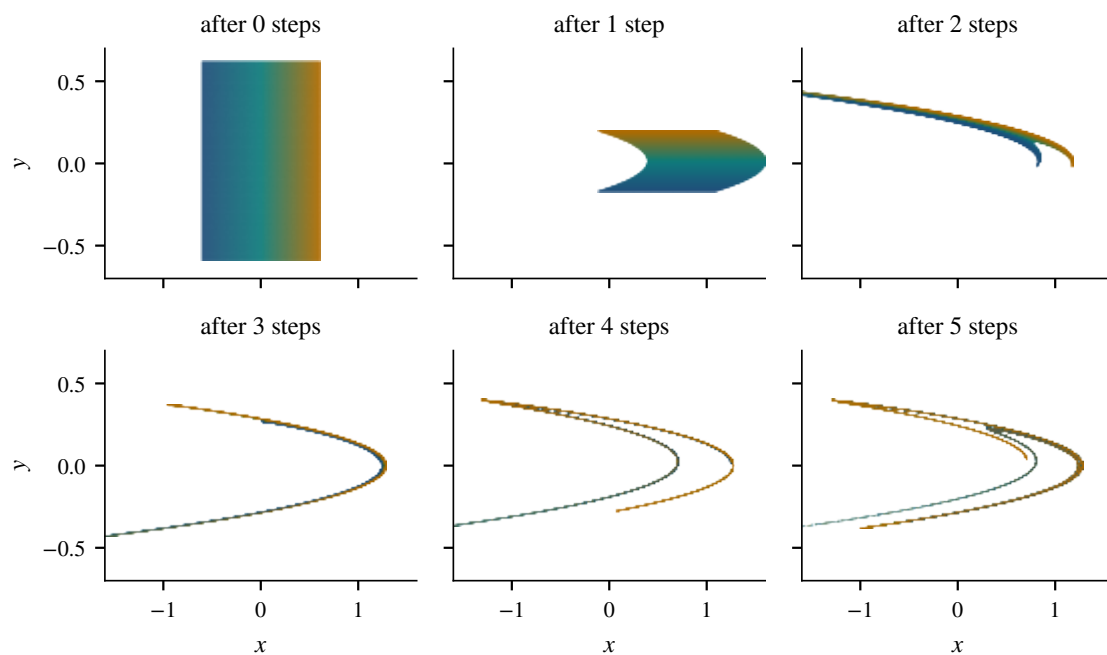


Figure 10.2: A square of starting points, coloured by where along x each point began, after up to five steps. One step bends it into a crescent, the next stretches it into a thin strip, and by the third the strip is folded back on itself and its colours lie in layers.

How hard is the stretching? Carry a tiny arrow along one long run, moving it at each step as the sides of the square were moved. Measure how much longer it got, add the logarithm of that to a running total, and shrink it back to length one. The average of the logarithms is the Lyapunov exponent of Chapter 7, now for a map of the plane.

code/ch10/stretch.py

```

38 def exponent(steps: int = 200_000) -> float:
39     """The average of log(stretch) of a tiny arrow along one long run."""
40     x, y = settle()
41     dx, dy = 1.0, 0.0 # an arrow of length one
42     total = 0.0
43     for _ in range(steps):
44         dx, dy = -2.0 * A * x * dx + dy, B * dx # one step moves it
45         x, y = henon(x, y)

```

```

46     grow = math.sqrt(dx * dx + dy * dy)           # how much it stretched
47     total += math.log(grow)
48     dx, dy = dx / grow, dy / grow                 # back to length one
49     return total / steps

```

Listing 10.5: The average stretch of a tiny arrow

```

step 0: gap 1.0e-10
step 10: gap 3.0e-08
step 20: gap 1.5e-07
step 30: gap 4.0e-06
step 40: gap 3.5e-05
step 50: gap 3.6e-04
step 60: gap 7.4e-02
step 70: gap 2.1e+00
step 80: gap 3.1e-01
Lyapunov exponent: 0.42 per step
stretch per step: 1.52
squeeze per step: 0.20

```

The first lines come from the other half of the file: two starts 10^{-10} apart, which Python writes as $1.0\text{e-}10$. Their gap grows until by step 70 it is about 2, and after that it never exceeds 2.6, roughly the width of the attractor. Bounded, and sensitive, like the dough. The exponent is 0.42 per step: along the attractor a tiny arrow grows on average by a factor of 1.52 a step, while across it the map squeezes by 0.20. Multiply the two and you get back about 0.3, the area factor. Stretch one way, squeeze the other, fold to fit: that is the whole recipe.

Lab 10.2 — One step, and its area

Write `step`, one step of Hénon's map, and `area_factor`, which moves three corners of a tiny square one step, reads off the two new sides as arrows (p, q) and (r, s) , and returns the size of $ps - qr$ divided by the old area. Check that it gives 0.3 wherever you put the square, and that changing b changes it while changing a does not.

Starter: `code/labs/ch10/l10_02_henon.py`

Test: `code/labs/ch10/test_l10_02_henon.py`

Run: `cd code && uv run pytest -k l10_02_henon`

10.5 Layers within layers

Drawn at the size of a page, Hénon's attractor looks like a few curved lines. Look closer.

Stop and think 10.5

Draw any smooth curve with a pen and zoom in on a small piece of it, again and again. What does the piece look like in the end? Now guess what you will see if you zoom in on one of the attractor's lines.

Answer it before you read on.

Answer. A smooth curve looks more and more like a straight line. The attractor's line does not: under the magnifier it splits into several parallel lines, and each of those, magnified again, splits again.

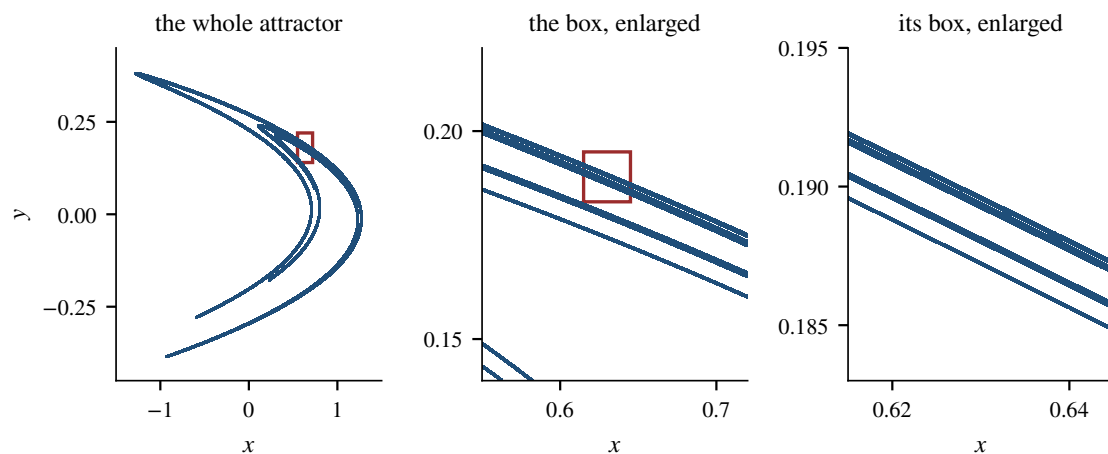


Figure 10.3: Zooming into Hénon's attractor. Each red box is shown enlarged in the next panel. What looked like one line is a bundle of lines, and each line of the bundle is itself a bundle.

The layers come straight from the recipe. Every step stretches the set, folds it over and squeezes it to fit, so each fold lays a thinner copy of everything beside what was there, and the copy carries its own layers inside it. Two consequences sound absurd until you have watched them happen. The attractor has no area, since every area is multiplied by 0.3 at every step, for ever. And its length has no end, since it is traced by a curve stretched by about 1.52 at every step, for ever.

The trap

'Zoom in far enough on any curve and it turns into a straight line.' That is true of the smooth curves you draw with a pen and of the graphs of the formulas you met at school. It is not true here. However far you zoom into Hénon's attractor, each line splits into a bundle: the structure has no smallest scale.

An attractor with this layered structure, on which the motion is chaotic, is called a *strange attractor*, a name given by David Ruelle and Floris Takens in 1971. Lorenz's butterfly is one, and so is Hénon's. Such a set is more than a line, which would have a finite length, and less than an area, since it has none: its dimension lies between one and two, and Chapter 11 measures where.

Where you meet this

Chemical engineers mix thick fluids — resins, sealants, food pastes — with a *static mixer*, a pipe fitted with a row of short twisted blades and no moving parts. Each blade splits the stream in two, turns the halves and stacks them: a stretch and a fold, done by the shape of the pipe. Every blade doubles the number of layers, so twenty blades make roughly a million, because ten doublings make about a thousand.

What we are not proving

Stephen Smale turned the baker's knead into mathematics about a decade before Hénon's map. His *horseshoe* stretches a square into a long strip, bends it into a horseshoe and lays it back across the square. The points that stay in the square for ever form a set with gaps inside gaps, and on it Smale proved that the motion is as unpredictable as a sequence of coin tosses. What this chapter shows about Hénon's map at $a = 1.4$ and $b = 0.3$ is measured, not proved: nobody has proved that the set the computer draws is not, say, an extremely long cycle. Michael Benedicks and Lennart Carleson proved in 1991 that maps of Hénon's form do have strange attractors for many other choices of the two numbers.

10.6 Cutting a flow into a map

Why should a rule with two numbers tell you anything about the Lorenz system, a flow with three? Picture the butterfly with a pane of glass pushed through one of its wings. The trajectory pierces the pane again and again, leaving a dot each time. Forget the path between the crossings and keep only the dots. Given one dot, the flow decides where the next will be, so the dots obey a rule of their own: a map from the pane to itself, with two numbers for a state. This is a *Poincaré section*, after Henri Poincaré, who used it in his work on the three bodies of Chapter 13.



Figure 10.4: A Poincaré section. The flow's path crosses a pane again and again; the rule that takes one crossing to the next is a map of the pane.

Hénon's map is not the Lorenz flow's own section, but it was built to behave like one: between two crossings a flow like Lorenz's stretches a patch of the pane, squeezes it and folds it, which is what the bend, the squeeze and the swap do.

Stop and think 10.6

Chapter 4 said that a smooth flow needs at least three numbers in its state to be chaotic. Hénon's map is chaotic with two, and the logistic map of Chapter 5 with one. Is that a contradiction?

Answer it before you read on.

Answer. No. That rule is about flows, whose paths move continuously and in a plane can never cross one another. A map jumps from one state to the next, so nothing stops it folding. Cutting a three-number flow with a plane leaves a two-number map, and the chaos survives the cut.

Summary

- Chaos in a bounded system is two moves repeated: a **stretch**, which pulls neighbours apart, and a **fold**, which puts everything back. The **tent map** kneads a strip and doubles every small gap.
- **Hénon's map**, $x_{n+1} = 1 - ax_n^2 + y_n$ and $y_{n+1} = bx_n$, bends, squeezes and swaps a plane.
- A cloud of starts ends up on one set, the **attractor**; the starts that end there form its **basin**.
- Every step multiplies every area by b , the **determinant** of the step, yet the motion never stops: the map **squeezes** one way while it **stretches** the other.
- Zoomed in, each line of the attractor splits into a bundle, again and again. An attractor like that, with chaotic motion on it, is a **strange attractor**: no area, no end to its length.
- A **Poincaré section** turns a flow into a map, which is why a two-number map can be chaotic although a two-number flow cannot.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

explain how motion can stay inside a region for ever and still be sensitive to its start, using the words stretch and fold	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
iterate the tent map and Hénon's map in Python and read what they print	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
work out how much one step of Hénon's map multiplies an area, and check it by measuring	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what an attractor, its basin and a strange attractor are	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
describe what zooming in shows on a strange attractor, and how that differs from zooming in on a smooth curve	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
describe how a Poincaré section turns a flow into a map	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. A raisin sits at 0.3 on the strip. Where is it after one knead, and after two?
2. Two raisins start 0.01 apart and stay on the same side of the fold. How far apart are they after three kneads, and why can the doubling not go on for ever?
3. Take one step of Hénon's map from the point $(0.5, 0)$.
4. A patch of starting points has an area of 2. What is its area after three steps of Hénon's map, and have its points stopped moving?
5. What is the difference between an attractor and its basin?
6. What do you see when you zoom in on a smooth curve, and when you zoom in on Hénon's attractor?
7. What is a Poincaré section, and why does it let a map with a two-number state be chaotic?

Fractals: measuring what has no length

How long is the coast of Britain? It sounds like a question for an atlas. Look it up in two reference books, though, and you may well find two figures too far apart for either to be a rounding of the other, and neither of them a misprint. Why?

This chapter answers that, and then replaces length with a number that does describe shapes like a coast. You will build two such shapes by repeating one rule, measure the new number by counting boxes, and then measure the shape chaos left behind in Chapter 10 the same way.

What you will be able to do

When you have finished this chapter you will be able to:

- measure a curve with rulers of different lengths and say whether its length settles
- build the Koch curve and the Cantor set by repeating a rule, and work out what each round does to their length
- work out a dimension from how many copies a shape is made of and how much each copy is shrunk
- compute a box-counting dimension from a list of points, read it off a log-log plot, and say how far to trust it
- say what a dimension between one and two claims about a shape, and over what range of sizes a real object can earn one

11.1 A coastline and a ruler

The first person to take the question seriously was Lewis Fry Richardson, an English physicist. Studying wars late in his life, he wanted to know whether countries with a longer shared border fought more often, and found that neighbours could quote quite different lengths for the same border. His measurements were published in 1961, after his death.

You measure a curve on a map with dividers: two compass points set a fixed distance apart. Walk them along the coast, count the steps and multiply by the step. That fixed distance is your ruler.

Stop and think 11.1

You measure a stretch of coast twice on the same detailed map: once with the dividers set to 10 kilometres, once set to 1 kilometre. Which measurement is longer, or do they agree?

Answer it before you read on.

Answer. The 1-kilometre measurement. The long ruler steps straight across every bay narrower than itself; the short one goes into the bay and out again, and each detour adds length.

In 1967 Benoit Mandelbrot built a paper in the journal *Science* on Richardson's measurements, with the title 'How long is the coast of Britain?'. His answer, which this chapter works towards, was that the question has none, and that a better question does.

First, see what the experiment does to a curve with no hidden detail: half a circle of radius one, whose length is π , about 3.14159 (half of the 2π round a whole circle). Put points on it, walk from each to the next in a straight line, and add up the steps. That is the whole ruler:

code/ch11/_ruler.py

```

17 def length(points: list[Point]) -> float:
18     """Add up the straight steps from each point to the next."""
19     total = 0.0
20     for (x0, y0), (x1, y1) in pairwise(points):
21         dx, dy = x1 - x0, y1 - y0
22         total += math.sqrt(dx * dx + dy * dy) # Pythagoras
23     return total

```

Listing 11.1: The ruler: add up the straight steps

`pairwise(points)` hands the loop each point with the next one, and the line marked Pythagoras is the distance between them: across squared plus up squared, square-rooted. The next listing starts with three points on the half circle and, round after round, adds a point on the circle halfway between every pair of neighbours, which halves the ruler:

code/ch11/semicircle.py

```

40 print(" steps    ruler    measured length")
41 for pts in rounds(8):
42     steps = len(pts) - 1
43     ruler = length(pts[:2]) # one step, from the first point
44     print(f"{steps:6d} {ruler:.4f} {length(pts):.5f}")

```

Listing 11.2: Half a circle, measured with shorter and shorter rulers

steps	ruler	measured length
2	1.4142	2.82843
4	0.7654	3.06147
8	0.3902	3.12145
16	0.1960	3.13655
32	0.0981	3.14033
64	0.0491	3.14128
128	0.0245	3.14151
256	0.0123	3.14157
512	0.0061	3.14159

Every shorter ruler measures more, as on the coast. But each round adds about 0.25 of what the round before added, so the additions die away and the measurements settle on π . That settling is what a length *is*: the number your measurements approach as the ruler shrinks. A smooth curve has one. Richardson's coasts did not: for every coast and border he tried, the measured length kept climbing as the ruler shrank.

11.2 A curve that never settles

You cannot reason about detail you have not yet mapped, so make a coast by a rule, where every scale of detail is known in advance. Helge von Koch described this one in 1904. Start with a straight line of length one. Cut it into three equal pieces and replace the middle one with the two upper sides of a triangle, each as long as the piece it replaces, like a tent pitched on the middle third. That is round one. Round two does the same to each of the four pieces, and so on, for ever.

Stop and think 11.2

Each round replaces every straight piece with four pieces, each a third as long. By what number does the total length get multiplied each round?

Answer it before you read on.

Answer. By $\frac{4}{3}$: four pieces, each $\frac{1}{3}$ as long as the piece they replace, add up to $4 \times \frac{1}{3}$ of it.

Look at round 5: the curve is made of four smaller copies of itself, each made of four smaller copies again. That is *self-similarity*, the rule showing through at every scale. The listing builds rounds 0 to 6 and measures each with the ruler of Listing 11.1:

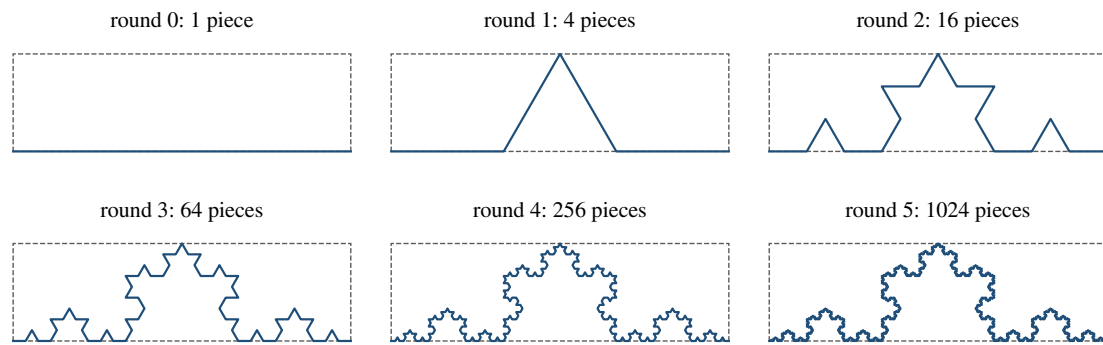


Figure 11.1: The Koch curve after rounds 0 to 5. Each round pitches a tent on the middle third of every straight piece; the dashed box, as tall as the first tent, holds every round.

code/ch11/koch_length.py

```

35 print("round  pieces  ruler  length  height  area")
36 for n in range(7):
37     pts = koch(n)
38     pieces = len(pts) - 1
39     ruler = 1 / 3**n
40     height = max(y for x, y in pts)
41     print(f"{n:5d} {pieces:6d} {ruler:7.5f} {length(pts):7.3f}"
42           f" {height:7.4f} {area(pts):7.4f}")

```

Listing 11.3: The Koch curve, measured round by round

round	pieces	ruler	length	height	area
0	1	1.00000	1.000	0.0000	0.0000
1	4	0.33333	1.333	0.2887	0.0481
2	16	0.11111	1.778	0.2887	0.0695
3	64	0.03704	2.370	0.2887	0.0790
4	256	0.01235	3.160	0.2887	0.0832
5	1024	0.00412	4.214	0.2887	0.0851
6	4096	0.00137	5.619	0.2887	0.0859

After round n every piece is $\frac{1}{3^n}$ long, so measuring that round is measuring the Koch curve with a ruler of that length. The length column is $\frac{4}{3}$ times the row above, every time: by round 6 it is 5.619, and by round 20 it would be 315. The height stays at 0.2887, the first tent's. The area between the curve and its base (a function higher in the same file) creeps towards 0.0866 and no further: each round pitches four times as many tents, each a ninth of the area, so it adds $\frac{4}{9}$ of what the round before added, and those additions die away like the half circle's.

The rule itself is `koch` in `chaoslab`. The one unfamiliar line is the tent's tip: half a piece along the middle third, then sideways by h times the piece, the height of an equilateral triangle.

code/src/chaoslab/fractals.py

```

31 def koch(level: int) -> list[Point]:
32     """The corners of the Koch curve from (0, 0) to (1, 0) after `level`
33     rounds of replacing every segment's middle third with a tent."""
34     h = math.sqrt(3.0) / 2.0 # height of a unit equilateral triangle
35     pts = [(0.0, 0.0), (1.0, 0.0)]
36     for _ in range(level):
37         nxt = [pts[0]]
38         for (x0, y0), (x1, y1) in zip(pts, pts[1:], strict=False):
39             dx, dy = (x1 - x0) / 3, (y1 - y0) / 3
40             a = (x0 + dx, y0 + dy)
41             b = (x0 + 2 * dx, y0 + 2 * dy)
42             # the tip: the middle third turned 60 degrees to the left
43             tip = (a[0] + dx / 2 - h * dy, a[1] + dy / 2 + h * dx)
44             nxt += [a, tip, b, (x1, y1)]
45         pts = nxt
46     return pts

```

Listing 11.4: The Koch rule, in chaoslab

Stop and think 11.3

Suppose you could draw every round, for ever, and measure the finished curve with a ruler as short as you like. What length would you get?

Answer it before you read on.

Answer. No length at all. Each time the ruler shrinks by three the measurement grows by $\frac{4}{3}$, and it never settles. The finished curve is longer than any number you care to name, and it still fits in the dashed box.

The trap

‘A coastline has a length; it is only hard to measure.’ A length is what your measurements settle on as the ruler shrinks, and the measurements of a coast do not settle: every finer map finds bays inside the bays. A coastline’s length is a fact about the coast *and the ruler* together, so a figure quoted without its ruler, or the scale of its map, is not a fact about the coast at all.

11.3 Counting copies

Georg Cantor described a second shape in 1883, and it goes the other way. Take the line from 0 to 1, cut it into thirds and throw away the middle third, keeping its ends. Do the same to each piece that is left, for ever.

Stop and think 11.4

Throw away middle thirds for ever. How much of the line's length is left at the end? And is anything left at all?

Answer it before you read on.

Answer.

No length: each round keeps $\frac{2}{3}$ of what was there, so after 20 rounds 1 048 576 pieces hold 0.0003 of it, and the total heads for zero. But points are left. The ends of every piece, such as $\frac{1}{3}$ and $\frac{2}{3}$, are never thrown away. Nor is $\frac{1}{4}$, which is never an end: it lies in the left third; stretch that third three times and $\frac{1}{4}$ lands on $\frac{3}{4}$, in a right third; stretch that one and it is back at $\frac{1}{4}$. It dodges every middle third for ever.

What is left is the *Cantor set*: a dust of infinitely many points, more than you could ever write down in a list, holding no length at all. Length is the wrong question for both shapes: one has too much of it, the other too little. Here is a better one.

Shrink a line to a third of its size and three copies rebuild the original. Shrink a square to a third of its width and it takes nine; a cube takes 27. Those are 3^1 , 3^2 and 3^3 , and the power is the dimension. So a shape made of copies of itself, each shrunk by the same factor, has the dimension D with

$$\text{shrink}^D = \text{copies.}$$

Stop and think 11.5

The Koch curve is four copies of itself, each shrunk by three. If $3^D = 4$, is D equal to 1, to 2, or somewhere between?

Answer it before you read on.

Answer. Between: $3^1 = 3$ copies is too few and $3^2 = 9$ is too many. The power of 3 that gives 4 is about 1.26.

Finding that power is the job of the logarithm, which answered 'how many steps until' in Chapter 2 and here answers 'how many factors of three make four':

$$D = \frac{\ln 4}{\ln 3} \approx 1.26,$$

where $\ln$ is a calculator's natural logarithm. The Cantor set is two copies of itself, each shrunk by three, so its dimension is $\ln 2 / \ln 3$, about 0.63: more than separate points, of dimension 0, and less than a line.

The trap

‘A dimension must be a whole number.’ It is a whole number for the shapes you met at school because they are smooth: zoom in on a smooth curve and it looks like a line. Count copies instead of directions and dimension becomes a measurement. The Koch curve’s 1.26 says something precise: too long for a length, too thin for an area. A shape whose dimension is not a whole number is what Mandelbrot, in 1975, named a *fractal*, from the Latin *fractus*, broken.

Dimension also explains Richardson’s rulers. Shrink the ruler by three and the measured length is multiplied by 3^{D-1} : by $\frac{4}{3}$ for the Koch curve, by 1 for anything smooth, by $\frac{2}{3}$ for the Cantor set. On axes where every step is a factor of ten each is a straight line (Figure 11.2). Richardson’s coasts gave such lines, and Mandelbrot read dimensions off their slopes: about 1.25 for the west coast of Britain.

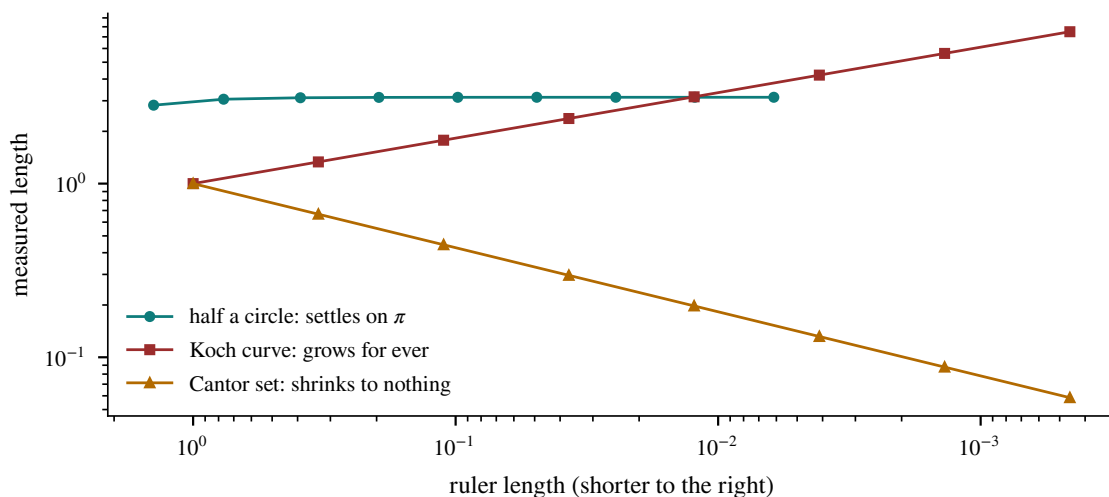


Figure 11.2: What happens to a measured length as the ruler shrinks (rightwards). A smooth curve settles; the Koch curve climbs for ever and the Cantor set shrinks to nothing, each along a straight line whose slope is its dimension minus one.

Lab 11.1 — The Cantor set by rule

Write `cantor`, which returns the pieces left after a number of rounds, and `length_left`, which adds them up. Then write `similarity_dimension`, which turns ‘this many copies, each shrunk by this factor’ into a dimension with `math.log`. The test also checks that $\frac{1}{4}$ survives every round.

Starter: `code/labs/ch11/l11_01_cantor.py`
 Test: `code/labs/ch11/test_l11_01_cantor.py`
 Run: `cd code && uv run pytest -k l11_01_cantor`

11.4 Counting boxes

Counting copies needs a rule, and no rule built a coast. What works on anything you can draw is *box counting*: lay a grid of squares of side s over the shape, count the squares it

touches, call that N , halve s and count again.

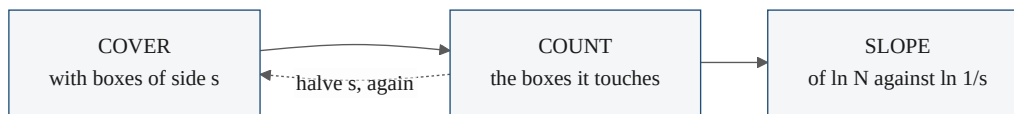


Figure 11.3: Box counting in three steps. The recipe needs a list of points and nothing else, so it works on shapes that no rule built.

For a line, halving the box doubles the count; for a filled square it multiplies the count by four. In general, halving the box multiplies N by 2^D , and that is how D is read off.

Stop and think 11.6

If the Koch curve's dimension really is 1.26, about how many times as many boxes should it need each time the box size is halved?

Answer it before you read on.

Answer. About $2^{1.26}$, which is 2.4: more than a line's 2 and fewer than a square's 4.

The next listing covers the 65 537 corners of the round-8 curve with boxes from $\frac{1}{2}$ down to $\frac{1}{1024}$, in halves on purpose: the curve is built in thirds, and a measurement that knew that would be less of a test.

code/ch11/koch_boxes.py

```

35 ns = counts(POINTS, SIZES)
36 print(" box side   boxes   times as many as the row above")
37 for k, (s, n) in enumerate(zip(SIZES, ns, strict=True)):
38     line = f" 1/{round(1 / s):<6d} {n:8d}"
39     if k:
40         line += f"   {n / ns[k - 1]:.2f}"
41     print(line)
42 print(f"slope of ln N against ln(1/s): {dimension(SIZES, ns):.3f}")
43 print(f"ln 4 / ln 3:                      {math.log(4) / math.log(3):.3f}")
  
```

Listing 11.5: Box counting on the Koch curve

box side	boxes	times as many as the row above
1/2	3	
1/4	7	2.33
1/8	15	2.14
1/16	33	2.20
1/32	91	2.76
1/64	201	2.21
1/128	539	2.68
1/256	1255	2.33
1/512	3169	2.53
1/1024	7325	2.31
slope of $\ln N$ against $\ln(1/s)$: 1.262		
$\ln 4 / \ln 3$: 1.262		

The last column wobbles around 2.4, because thirds and halves do not line up. To average the wobble away, plot the logarithm of each count against the logarithm of each $1/s$. If N grows like $(1/s)^D$, the logarithm turns that power into a straight line of slope D , and `fit_slope` draws the line that passes as close as it can to all ten points. Its slope is 1.26; $\ln 4 / \ln 3$ is 1.26.

Do not trust that agreement further than it deserves. Fit through any five neighbouring sizes instead of all ten and the slope lands anywhere from 1.21 to 1.31. That spread is the honest size of the error, and it is why a box-counting dimension is fitted over as wide a range of sizes as the data allow.

Lab 11.2 — Count the boxes yourself

Write `count_boxes`, which says how many squares of a given side hold at least one point, and `box_dimension`, which fits the slope over a list of sizes. The test tries a straight line, a filled square, the Koch curve and the Cantor set, whose box-counting dimension should agree with the one you got from copies in the first lab.

Starter: `code/labs/ch11/l11_02_boxes.py`

Test: `code/labs/ch11/test_l11_02_boxes.py`

Run: `cd code && uv run pytest -k l11_02_boxes`

11.5 The shape chaos leaves behind

Chapter 10 ran Hénon's map, which takes a point (x, y) of the plane to $(1 - 1.4x^2 + y, 0.3x)$, and found that its points settle onto an attractor with no area at all, whose stretching gives it no finite length either. Nobody built that shape copy by copy. It is exactly the kind of object box counting is for.

Stop and think 11.7

No area and no finite length. Before any box is counted: would you expect the Hénon attractor's dimension to be 1, 2, or something between?

Answer it before you read on.

Answer. Something between 1 and 2: no finite length makes it more than a curve, and no area makes it less than a surface. The listing measures about 1.23.

code/ch11/henon_boxes.py

```

37 print("boxes of side 1/4, 1/8, ... 1/1024 touched by the attractor")
38 for n in (100_000, 1_000_000):
39     ns = counts(attractor(n), SIZES)
40     fives = [dimension(SIZES[i:i + 5], ns[i:i + 5]) for i in range(5)]
41     print(f"{n:9d} points:" + "".join(f"{c:6d}" for c in ns))
42     print(f"    slope through all nine sizes:    "
43           f"{dimension(SIZES, ns):.3f}")
44     print(f"    through any five in a row:          "
45           f"{min(fives):.3f} to {max(fives):.3f}")

```

Listing 11.6: Box counting on the Hénon attractor

```

boxes of side 1/4, 1/8, ... 1/1024 touched by the attractor
100000 points:    28    63   148   349   830  1917  4455 10081 22362
slope through all nine sizes:    1.214
through any five in a row:        1.190 to 1.234
1000000 points:   28    63   148   349   833  1930  4548 10551 25457
slope through all nine sizes:    1.230
through any five in a row:        1.226 to 1.237

```

The function `attractor` runs the map from $(0,0)$ and keeps every point after the first thousand steps. With 1 000 000 points the slope through all nine box sizes is 1.23, and through any five neighbouring sizes it lies between 1.23 and 1.24. With 100 000 points it drops to 1.21: the smallest boxes need many points before every thin layer of the attractor has been visited, so too few points leave boxes uncounted and flatten the line.

Careful published measurements, with far more points and smaller boxes, put the dimension at about 1.26. Ours is a little under, has a spread, and has a known reason to be low: that is what an honest measurement of a dimension looks like. That the published figure matches the Koch curve's is a coincidence: the two shapes know nothing of each other.

This is the sense in which the attractors of this part of the book are strange. Stretching makes them long, folding keeps them in a box, and a shape too long for a length and too thin for an area is a fractal.

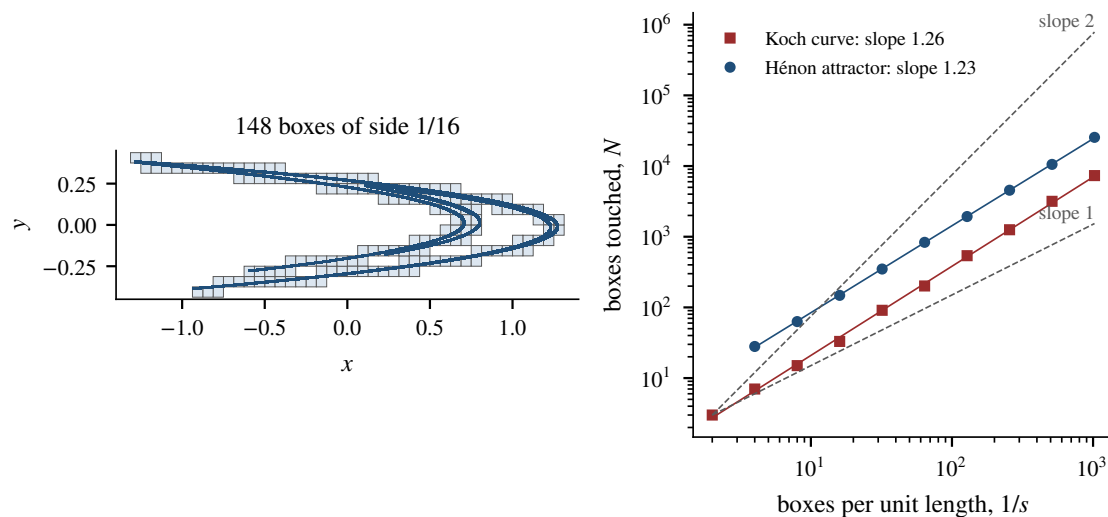


Figure 11.4: Left: the Hénon attractor covered by boxes of side $\frac{1}{16}$. Right: boxes touched against boxes per unit length, on axes where every step is a factor of ten. Both shapes lie on straight lines between the dashed lines of slope 1 and 2.

11.6 Fractals in the world

Once you know the shape you see it everywhere. A fern frond is made of smaller fronds. A river gathers streams that gather smaller streams. A romanesco broccoli is a spiral of cones, each a spiral of smaller cones. Blood vessels branch, and branch again.

Stop and think 11.8

Your lungs must pass oxygen into the blood across a thin surface, and all of that surface must fit inside your chest. Why might a lung be built like a fractal, branching and branching again?

Answer it before you read on.

Answer. Because a fractal packs an enormous surface into a small space, as the Koch curve packs an unlimited length into its dashed box. Every round of branching multiplies the surface while the chest stays the same size.

The trap

‘Fractals are decoration: pretty pictures, no use.’ Branching in lungs, blood vessels and river networks does work: it reaches every point of a volume or a landscape and packs a large surface or a long edge into a bounded space. The shape is the solution to a problem, and its dimension is a number you can measure and compare.

Where you meet this

In your lungs the airways divide in two about twenty-three times between the windpipe and the tiny air sacs where gas crosses into the blood, folding a surface of several tens of square metres into your chest. Physiologists describe the airway tree with this chapter’s ideas: each generation of branches is a smaller copy of the one before.

Every one of these is fractal only over a range of sizes. The Koch curve has tents inside tents all the way down, because a rule does not get tired; a coast stops showing new detail around its pebbles and grains of sand, a fern at its smallest leaflets, a lung at air sacs a fraction of a millimetre across. So when somebody calls a real object a fractal, ask: over what range of sizes, and measured how? A box count spanning a factor of a thousand is a finding; one spanning a factor of three is a guess.

The most famous fractal of all comes from iterating a rule on the points of a plane rather than of a line, and it is the subject of Chapter 12.

Summary

- A curve’s **length** is what its measurements settle on as the ruler shrinks. A smooth curve settles; a coastline keeps growing, as Richardson found and Mandelbrot asked about in 1967.
- The **Koch curve** multiplies its length by $\frac{4}{3}$ every round, so it has no finite length, while its height and the area it bounds stay fixed. It is **self-similar**: made of smaller copies of itself.
- The **Cantor set** keeps $\frac{2}{3}$ of its length every round, so no length is left, yet infinitely many points are. A shape made of copies of itself, each shrunk by the same factor, has **dimension** D with $\text{shrink}^D = \text{copies}$: about 1.26 for Koch and 0.63 for Cantor. A shape whose dimension is not a whole number is a **fractal**.
- **Box counting** measures a dimension from points alone: the slope of $\ln N$ against $\ln(1/s)$, where N boxes of side s touch the shape. The spread over different ranges of sizes is the honest error.
- The **Hénon attractor** measures about 1.23 (published: about 1.26): strange attractors are fractals.

- Real objects **are fractal over a range of sizes**, never all the way down, and fractal branching packs a large surface into a small space.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

measure a curve with rulers of different lengths and say whether its length settles	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
build the Koch curve and the Cantor set by repeating a rule, and work out what each round does to their length	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
work out a dimension from how many copies a shape is made of and how much each copy is shrunk	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
compute a box-counting dimension from a list of points, read it off a log-log plot, and say how far to trust it	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what a dimension between one and two claims about a shape, and over what range of sizes a real object can earn one	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Two atlases give the same coastline as 800 and 1100 kilometres long. Can both be right, and what should each have told you?
2. A Koch curve starts as a line of length 1. How long is it after three rounds?
3. A different rule cuts every piece of a line into five equal parts and throws away the second and the fourth. How much length is left after one round, and what is the dimension of what is left for ever?
4. A shape is made of 8 copies of itself, each shrunk by 2. What is its dimension, and what familiar shape has it?
5. Halving the box size takes a box count from 100 to 283. Roughly what is the dimension?
6. Why did the Hénon attractor's measured dimension rise when the number of points went up?
7. Somebody tells you a head of broccoli is a fractal. What should you ask?

Iterating in the plane: Julia and Mandelbrot

Take a number, square it and add 1. Do the same to the answer, and again. From 0 you get 0, 1, 2, 5, 26, and the numbers are gone, off towards infinity. Add -1 instead of 1 and the same recipe goes 0, -1 , 0, -1 for ever. So here is a question with a yes-or-no answer for every number c you might add: *started at 0, does the recipe run away?*

On the number line the answer is dull: the c that do not run away form one unbroken stretch, from -2 to $\frac{1}{4}$. This chapter asks the same question of the points of a flat plane, with nothing else changed, and the answer is the most famous picture in mathematics. It is also Chapter 8's road to chaos seen from the side, which is the last thing Part III has to show you about the shapes chaos makes.

What you will be able to do

When you have finished this chapter you will be able to:

- add and multiply two points of the plane, by hand and with Python's complex numbers
- run the rule $z \rightarrow z^2 + c$ from a starting point and find the step at which its orbit escapes
- say what a Julia set and the Mandelbrot set each hold fixed and each let vary
- convert between the logistic map's r and the Mandelbrot set's c , and find Chapter 8's cascade on the real axis
- tell the parts of the Mandelbrot set where the orbit settles from the parts where it never does

12.1 Numbers that are points

Every number in this book so far has lived on a line. A point of a flat plane needs two numbers: how far across, and how far up. Write the point three across and four up as $3 + 4i$. For now the i is only a label meaning 'up': the number stuck to it is the up part, and the other is the across part. Numbers written like this are called *complex numbers*; the across part is called *real* and the up part *imaginary*, names you will have reason to distrust by the end of this section.

Adding two of them is sliding. Add the across parts and the up parts separately, $(3 + 4i) + (1 + 2i) = 4 + 6i$, exactly as two moves on a map add up to one.

Multiplying needs a rule, and here it is in words. Every point has a *length*, its distance from 0, and an *angle*, how far it is turned anticlockwise from the positive half of the

across line. For $3 + 4i$ the length is 5, by Pythagoras, and the angle is 53.13 degrees. The rule: **to multiply two points, multiply their lengths and add their angles**. Multiplying by 2, which has length 2 and angle 0, doubles a point's distance from 0 and leaves its direction alone. Multiplying by i , the point one step up, keeps the length and turns the point a quarter turn anticlockwise.

Stop and think 12.1

The point i has length 1 and angle 90 degrees. What is $i \times i$?

Answer it before you read on.

Answer. -1 . The lengths multiply to $1 \times 1 = 1$ and the angles add to 180 degrees: half a turn from 1 lands on -1 .

School tells you that no number squares to a negative, and on the line that is true. Off the line it is not, and nothing mysterious has happened: two quarter turns make a half turn. With $i \times i = -1$ in hand you can multiply out brackets as you always have and tidy up at the end:

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i.$$

In words: the across part of the product is the two across parts multiplied, minus the two up parts multiplied; the up part is each across part times the other's up part, added. That is the rule a computer uses, and it agrees with the lengths-and-angles rule for every pair of points. Python has it built in: it calls the type `complex` and writes i as `1j`.

code/ch12/complex_numbers.py

```
13 def times(a: float, b: float, c: float, d: float) -> tuple[float, float]:
14     """(a + b i) times (c + d i): multiply out, then use i * i = -1."""
15     return a * c - b * d, a * d + b * c
```

Listing 12.1: The multiplication rule, by hand

code/ch12/complex_numbers.py

```
26 print("i times i, by hand: ", times(0, 1, 0, 1))
27 print("i times i, by Python:", 1j * 1j)
28 z = 3 + 4j
29 print("(3+4i)^2, by hand: ", times(3, 4, 3, 4))
30 print("(3+4i)^2, by Python: ", z * z)
31 print(f"lengths: {abs(z)} and {abs(z * z)}")
32 print(f"angles: {degrees(z):.2f} and {degrees(z * z):.2f} degrees")
```

Listing 12.2: By hand, by Python, and by lengths and angles

```
i times i, by hand: (-1, 0)
i times i, by Python: (-1+0j)
(3+4i)^2, by hand: (-7, 24)
(3+4i)^2, by Python: (-7+24j)
lengths: 5.0 and 25.0
```

angles: 53.13 and 106.26 degrees

Square $3 + 4i$ both ways. The algebra gives $9 - 16 = -7$ across and $12 + 12 = 24$ up. The picture gives a length of $5 \times 5 = 25$ and an angle of 106.26 degrees, twice 53.13, and the point $-7 + 24i$ has exactly that length and that angle. Two descriptions, one answer.

Stop and think 12.2

Numbers with an i in them are called imaginary. Does anything in the world actually behave like i ?

Answer it before you read on.

Answer. Yes: anything that has a size and a direction and can be turned. Multiplying by i is a quarter turn, and turning is as real as anything you can measure.

The trap

‘Complex numbers are imaginary, so they describe nothing real.’ The name is history, not a description. A complex number is a point of the plane with a rule for multiplying, and the rule says how to stretch and how to turn. Anything that stretches and turns — a wave, a rotation, an alternating current — is described by complex numbers because that is what they are made for, not because anyone is pretending.

Where you meet this

Electrical engineers describe an alternating voltage as a point going round a circle: its length is the size of the voltage and its angle is where in its cycle it is. A circuit multiplies that point by a fixed complex number, its *impedance*, which scales the size and shifts the timing — lengths multiplying, angles adding. Engineers write j for i , because i was already taken for current, and Python follows them: that is why it writes $1j$.

12.2 Square, add, and ask whether it escapes

Here is the chapter’s whole rule. Pick a complex number c . Start at $z = 0$ and step by

$$z \rightarrow z^2 + c,$$

where the arrow means *becomes*: square the point, then slide it by c . It is the iteration of Chapter 2, with a plane in place of a line. When c sits on the across line this is the opening’s arithmetic: a run to infinity for $c = 1$, a cycle of two for $c = -1$.

Stop and think 12.3

Start at 0 with $c = -2$ and write down the first few steps. Does the orbit run away?

Answer it before you read on.

Answer. No. It goes 0, -2 , 2, and then $2 \times 2 - 2 = 2$ for ever: it sits exactly on 2 and never gets further out.

code/ch12/orbits.py

```

10 def orbit_of_zero(c: complex, steps: int) -> list[complex]:
11     """0, then z*z + c applied again and again: steps + 1 values."""
12     zs = [0j]
13     for _ in range(steps):
14         z = zs[-1]
15         zs.append(z * z + c)
16     return zs

```

Listing 12.3: The orbit of 0 under the rule

c = 1:	0	1	2	5	26	677
c = -1:	0	-1	0	-1	0	-1
c = 0.25:	0	0.25	0.3125	0.3477	0.3709	0.3875
c = -2:	0	-2	2	2	2	2
c = i:	0	1i	-1+1i	-1i	-1+1i	-1i

The last row is the plane at work. With $c = i$ the orbit leaves the line at once, runs round a small loop and then swaps between $-1 + i$ and $-i$ for ever. The row for $\frac{1}{4}$ creeps upwards and never gets past $\frac{1}{2}$.

You cannot watch for ever, so you need a test that settles the question in finitely many steps, and there is one: **once the orbit's length passes 2, it never comes back.** The number of steps it takes to pass 2 is the point's *escape time*, and the book's library computes it in five lines.

What we are not proving

Why 2 is enough, in words rather than as a proof. If c is longer than 2, the first step already passes 2, so suppose it is not, and that z is longer than 2. Squaring makes the length of z its own square, and adding c pulls a point back by at most the length of c , which is shorter than z . So the new length is at least the old length times (the old length minus one), which is more than the old length: the point moves out, and the further out it is, the faster it goes. 'Adding c pulls back by at most its length' is the triangle inequality, proved in any first course on complex numbers.

code/src/chaoslab/fractals.py

```

51 def escape_time(c: complex, limit: int = 100, z: complex = 0j) -> int:
52     """Iterate z -> z*z + c from z; return the step at which |z| exceeds 2,
53     or `limit` if it never does. Once |z| > 2 it is gone for good."""
54     for n in range(limit):
55         if z.real * z.real + z.imag * z.imag > 4.0:
56             return n
57         z = z * z + c
58     return limit

```

Listing 12.4: Escape time, from the book's library

The test adds the squares of the two parts and compares them with 4, which is Pythagoras without the square root: a length passes 2 exactly when its square passes 4. For $c = 1$ the answer is 3, the step at which the orbit reaches 5. A c whose orbit never escapes runs until `limit` and returns it.

Stop and think 12.4

$c = \frac{1}{4}$ never escapes: it is the creeping row of the table. The point $c = 0.26$ is a hundredth further right. Roughly how many steps does its orbit take to pass 2: about 5, about 30, or about 300?

Answer it before you read on.

Answer. About 30: it passes 2 at step 30. It creeps up towards $\frac{1}{2}$ like its neighbour, lingers there, and only then slips past.

code/ch12/slow_escape.py

```

15 for c in (0.3, 0.26, 0.251, 0.2501, 0.25):
16     n = escape_time(c, LIMIT)
17     if n < LIMIT:
18         print(f"c = {c:<6} passes 2 at step {n}")
19     else:
20         print(f"c = {c:<6} still inside after {LIMIT} steps")

```

Listing 12.5: Points close to the edge linger

```

c = 0.3      passes 2 at step 12
c = 0.26     passes 2 at step 30
c = 0.251    passes 2 at step 97
c = 0.2501   passes 2 at step 312
c = 0.25     still inside after 10000 steps

```

A little further out an orbit leaves quickly: $c = 0.3$ is gone by step 12. But each time c comes ten times closer to $\frac{1}{4}$, the orbit lingers about three times as long: 30, 97, 312 steps. So there is no step count after which you can be sure: a point that has not escaped after a thousand steps may still escape after ten thousand. Every picture that follows is drawn with a limit, and close to the edge it is an honest approximation, not a verdict.

Lab 12.1 — Escape time without complex numbers

Store a point as a pair of ordinary numbers and write one step of the rule with the multiplication rule of Section 12.1; then the escape time on top of it; then two yes-or-no questions: for this c , does the orbit of 0 stay, and for this fixed c , does the orbit of this start stay? The test checks your escape times against the library's at 480 points of the plane.

Starter: `code/labs/ch12/l12_01_escape.py`

Test: `code/labs/ch12/test_l12_01_escape.py`

Run: `cd code && uv run pytest -k l12_01_escape`

12.3 The Mandelbrot set

Now colour every c in the plane: dark if the orbit of 0 never escapes, lighter the sooner it does. The dark points are the *Mandelbrot set*. The program below draws it in characters, one for each c , and counts a point as in the set if it is still inside after 50 steps.

`code/ch12/ascii_mandelbrot.py`

```

13 LIMIT = 50
14 MARKS = " .,:--==+*##" # the later it escapes, the denser the mark
15
16
17 def mark(c: complex) -> str:
18     n = escape_time(c, LIMIT)
19     return "@" if n == LIMIT else MARKS[min(n, len(MARKS) - 1)]
20
21
22 def main() -> None:
23     for row in range(19): # top to bottom
24         y = (9 - row) / 7.5 # from 1.2 down to -1.2
25         line = "".join(mark(complex(-2.1 + col * 0.042, y))
26                         for col in range(64))
27         print(line.rstrip())

```

Listing 12.6: The Mandelbrot set in characters

```

.....
.....:~+:.....
.....:~*~:.....
.....:~*@@@##~:.....
.....:~*#+~##+@@@##+~---#:...
.....:~+@@@@@@@@@@@@@@@@@@@@@@~:..
.....:~*~-----~*@@@@@@@@@@@@@@@@@@@@@@@@@@~+~:..
.....:~---+##@###+##+@@@@@@@@@@@@@@@@@@@@@@@@@@@@~:..
.....:~---+##@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@~:..
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*~:..
.....:~---+##@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@~:..
.....:~---+##@###+##+@@@@@@@@@@@@@@@@@@@@@@@@@@@@~:..
.....:~*~-----~*@@@@@@@@@@@@@@@@@@@@@@@@@@@@~+~:..
.....:~+@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@~:..
.....:~*#+~##+@@@##+~---#:...
.....:~*@@@##~:.....
.....:~*~:.....
.....:~+:.....
.....

```

The middle row is the across line, and it is solid @ from -2 to $\frac{1}{4}$, as the opening promised. Off the line the set is a heart-shaped body with a round bulb to its left, smaller bulbs on both, and a thin needle running out to -2 . Figure 12.1 draws it properly, with two squares on its edge magnified.

Pierre Fatou and Gaston Julia studied rules like this one around 1918 to 1920, with pen and paper and no pictures at all. Benoit Mandelbrot's computer pictures, published in 1980, made the set famous, and the rule he iterated in that paper was the logistic rule itself, written with complex numbers — which the last section of this chapter explains.

Stop and think 12.5

Magnify the edge a thousand times, a million times, and there is still new detail: spirals, hairs, tiny copies. Where is all of that detail stored?

Answer it before you read on.

Answer. Nowhere. Every point is worked out afresh from the rule and the escape test. The program that drew the characters above is 11 lines long.

The trap

‘Infinite detail means infinite information.’ The edge has detail at every scale you can reach, and all of it follows from one line of algebra and one question. The detail is *generated*, not stored, which is Chapter 5's lesson again: there, one line produced every kind of behaviour. To tell someone the whole set exactly you need only tell them the rule and the question; a short description can have an endless consequence.

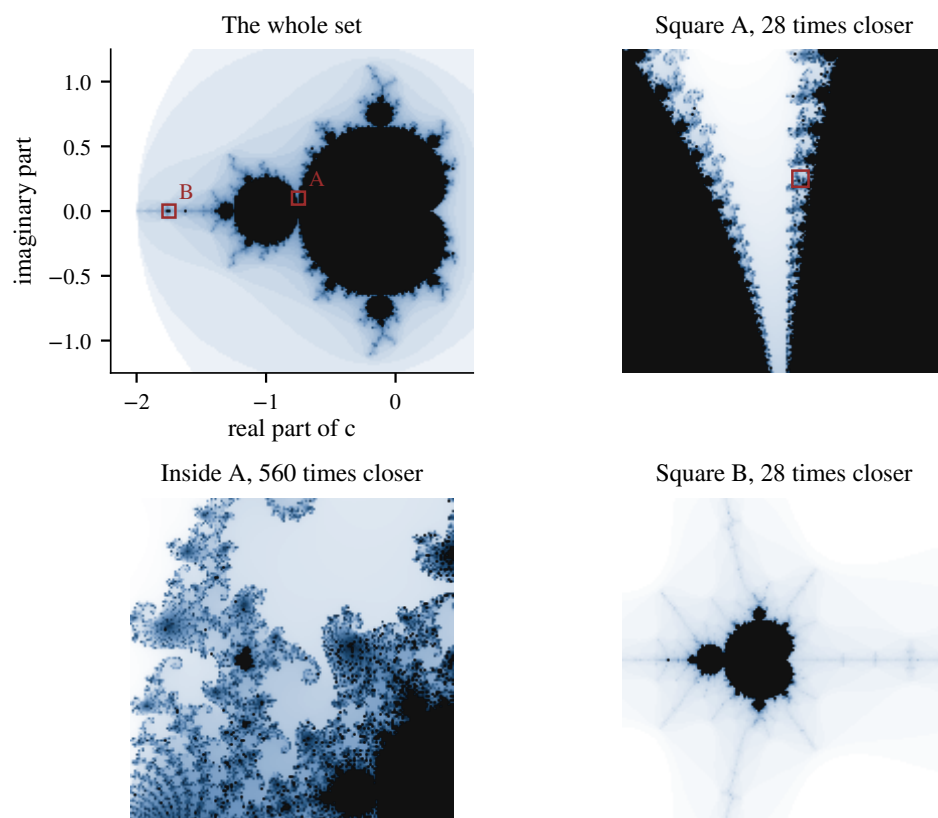


Figure 12.1: The Mandelbrot set: black where the orbit of 0 never escapes, darker blue the longer it lingers before escaping. Square A sits on the edge where the heart meets its biggest bulb and is shown at two magnifications; square B sits on the needle.

Some plain facts about the set are still unknown, such as its area. Count the grid squares whose centre is still inside after 50 steps and you get about 1.59; after 200 steps, 1.53; after 1000, 1.51. The count keeps falling as slow escapers near the edge are found out, and nobody has a formula for where it stops. The edge itself is so crumpled that its dimension, in the sense of Chapter 11, is 2, the most that any shape drawn in a plane can have: Mitsuhiro Shishikura proved it.

12.4 Julia sets: fix c , move the start

The same rule answers a second question. Hold c fixed and let the *start* vary: for each point z of the plane, does the orbit that begins there run away? The starts that stay fill in the *Julia set* of that c , named after Gaston Julia. (Strictly, the Julia set is the edge of that region.)

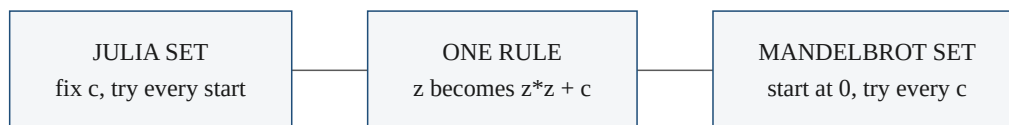


Figure 12.2: One rule, two questions. Fix c and ask about every start, and you draw a Julia set; start at 0 and ask about every c , and you draw the Mandelbrot set.

Stop and think 12.6

Take $c = 0$, so the rule is only ‘square the point’. Which starting points run away?

Answer it before you read on.

Answer. Those more than 1 from the centre. Squaring squares the length: a length above 1 grows without end, one below 1 shrinks towards 0, and a length of exactly 1 stays 1. The starts that stay fill a disc of radius 1, and its edge is a circle.

Change c and the circle crumples. Figure 12.3 shows two. For $c = -1$ the set is in one piece, a chain of bulbs pinched together. For $c = -0.8 + 0.2i$ it has fallen apart into dust: infinitely many specks with no two joined, like the Cantor set of Chapter 11.

What decides which? The orbit of 0, which is the Mandelbrot question. For $c = -1$ it swaps between 0 and -1 and stays; for $c = -0.8 + 0.2i$ it passes 2 at step 15. A theorem, not proved here but in any book on complex dynamics, says that this always decides it: the Julia set is in one piece exactly when c is in the Mandelbrot set, and is dust exactly when it is not. So the Mandelbrot set is a map of every Julia set at once, and each black point of it is a Julia set that holds together.

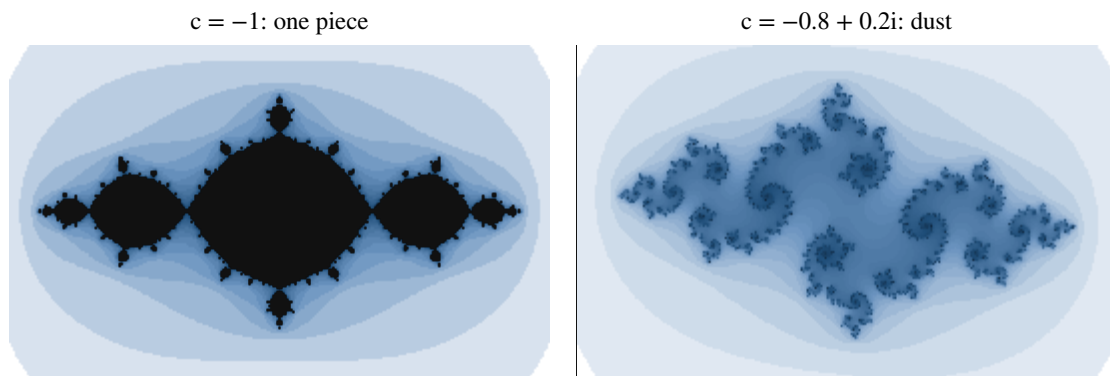


Figure 12.3: Two Julia sets, black where a start never escapes. For $c = -1$ the set is one piece; for $c = -0.8 + 0.2i$ only dust is left, too fine to show as black, marked by the dark spirals where starts linger longest.

12.5 The logistic map in disguise

The chapter's rule and Chapter 5's logistic map $x \rightarrow rx(1 - x)$ look different, and they are the same rule. Relabel the line: measure each x from $\frac{1}{2}$ instead of from 0, flip it, and stretch it by r , which is the change

$$z = r \left(\frac{1}{2} - x \right).$$

Then $x(1 - x) = \frac{1}{4} - z^2/r^2$, and the next z , which is $\frac{r}{2} - r^2x(1 - x)$, comes out as $z^2 + c$ with

$$c = \frac{r}{2} - \frac{r^2}{4}.$$

In words: halve r , then take away a quarter of its square. Nothing is lost in the translation. Every orbit of one rule is an orbit of the other on a relabelled line, and $x = \frac{1}{2}$ becomes $z = 0$: the orbit of 0 that the Mandelbrot set asks about is the logistic orbit from $\frac{1}{2}$.

Stop and think 12.7

Chapter 5's fixed point loses its grip at $r = 3$, and a cycle of two takes over. Which c is that?

Answer it before you read on.

Answer. $c = \frac{3}{2} - \frac{9}{4} = -0.75$: exactly where the heart meets the round bulb to its left.

So the heart is the fixed point and that bulb is the cycle of two. As r runs from 1 to 3, c runs from $\frac{1}{4}$ down to -0.75 along the heart's stretch of the line. The listing runs both rules side by side at the values of r from Chapter 5, from starts that the change matches up, and reports the length of the cycle each settles into.

code/ch12/same_rule.py

```

15 def square_plus(c: float) -> Callable[[float], float]:
16     """The rule  $z \rightarrow z^2 + c$ , on the real line."""
17     return lambda z: z * z + c
18
19
20 def both_periods(r: float, x0: float = 0.3) -> tuple[int | None, ...]:
21     """The cycle length of each rule, after 3000 steps from matching starts.
22     None means no cycle up to 64 steps long was found."""
23     xs = orbit(logistic(r), x0, 3000)
24     zs = orbit(square_plus(r_to_c(r)), r * (0.5 - x0), 3000)
25     return period(xs), period(zs)

```

Listing 12.7: The same rule, run twice

r	c	cycle of x	cycle of z
2.80	-0.5600	1	1
3.20	-0.9600	2	2
3.50	-1.3125	4	4
3.56	-1.3884	8	8
3.83	-1.7522	3	3
3.90	-1.8525	none	none

The cycles agree at every r — one, two, four, eight, three, and none at all at 3.9 — and that is Chapter 8’s bifurcation diagram written along the real axis. Figure 12.4 sets one picture above the other.

The second splitting, at $r = 1 + \sqrt{6}$, converts to $c = -1.25$, where the first bulb meets the next. Each bulb also has a centre, the c at which the orbit of 0 comes back to exactly 0: -1 , then -1.3107 , then -1.3815 , and on. Converted to r they are 3.2361, 3.4986 and 3.5546: the values at which Chapter 8’s cycles pass through $\frac{1}{2}$, its superstable parameters. The gaps between the centres shrink by a factor that settles at 4.669, Feigenbaum’s number again, and the bulbs pile up at $c = -1.401$, which is $r = 3.5699$: the Feigenbaum point. Beyond it lies the needle. Even the period-three window is there. It opens at $c = -1.75$, and square B of Figure 12.1 shows what sits on the needle at that point: a tiny copy of the whole set.

Stop and think 12.8

Chapter 5’s orbits that never settle live at $r = 3.9$. The table above converts it. Is that c in the heart, in a bulb, or on the needle?

Answer it before you read on.

Answer. On the needle: c is about -1.85 , left of the Feigenbaum point, past the last of the bulbs that pile up there.

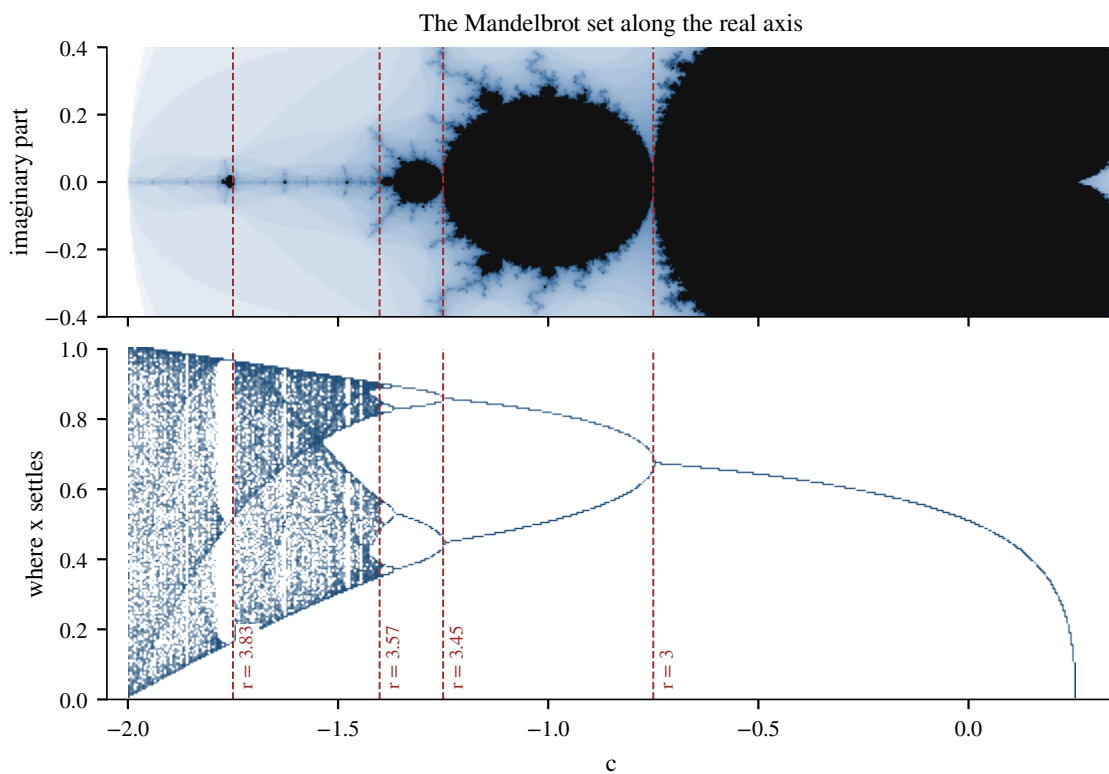


Figure 12.4: Above, the Mandelbrot set along the real axis; below, Chapter 8's bifurcation diagram with each r drawn at its c . Every splitting of the cycle happens where one bulb meets the next. The dashed lines mark $r = 3$, $r = 1 + \sqrt{6}$, the Feigenbaum point and the period-three window.

The trap

‘The Mandelbrot set is a picture of chaos.’ Most of what is black is order. In the heart the orbit of 0 settles on one point, in the big bulb on a cycle of two, and in every bulb on a cycle whose length the bulb fixes. The never-settling orbits of Chapter 5 sit on the thin edge, and on the real axis that means the needle, between its windows. The famous picture is mostly a map of order, with chaos living along its border.

Lab 12.2 — From r to c and back

Write the conversion both ways, r to c and c back to r , and the change $z = r(\frac{1}{2} - x)$ itself. The test runs the logistic map and $z \rightarrow z^2 + c$ side by side and checks that your change keeps them in step. Then send the period-doubling values of r in the starter across and see where on the real axis they land.

Starter: `code/labs/ch12/l12_02_convert.py`

Test: `code/labs/ch12/test_l12_02_convert.py`

Run: `cd code && uv run pytest -k l12_02_convert`

That closes Part III: chaos has a shape, and its last picture contains Chapter 8 whole. Part IV takes the ideas out into pendulums, planets, weather and living things.

Summary

- A **complex number** is a point of the plane, written $a + bi$. Adding is sliding; multiplying multiplies the **lengths** and adds the **angles**, the same as multiplying out with $i \times i = -1$.
- The rule $z \rightarrow z^2 + c$, run from 0, either stays or escapes, and once the length passes 2 it has escaped for good. The **escape time** counts the steps; near the edge it grows without limit.
- The **Mandelbrot set** is every c whose orbit of 0 never escapes: a heart, bulbs, a needle to -2 , and an edge with detail at every scale, generated by the rule. Its area is not known exactly.
- A **Julia set** fixes c and asks about every start. It is in one piece when c is in the Mandelbrot set and dust when it is not.
- The change $z = r(\frac{1}{2} - x)$ turns the **logistic map** into $z \rightarrow z^2 + c$ with $c = \frac{r}{2} - \frac{r^2}{4}$, so the real axis of the set is the bifurcation diagram: bulbs are cycles, they pile up at the Feigenbaum point, and chaos lives on the needle.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again

before you move on.

add and multiply two points of the plane, by hand and with Python's complex numbers	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
run the rule $z \rightarrow z^2 + c$ from a starting point and find the step at which its orbit escapes	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what a Julia set and the Mandelbrot set each hold fixed and each let vary	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
convert between the logistic map's r and the Mandelbrot set's c , and find Chapter 8's cascade on the real axis	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
tell the parts of the Mandelbrot set where the orbit settles from the parts where it never does	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Work out $(2 + 3i) + (1 - i)$ and $(1 + i)(1 - i)$.
2. Use lengths and angles to find $(1 + i)^2$, then check it by multiplying out.
3. Start at 0 with $c = 2$. At which step does the orbit pass 2?
4. Is $c = -1.5$ in the Mandelbrot set?
5. For $c = 0$, does the start $0.6 + 0.9i$ run away?
6. Convert $r = 3.2$ to c . Is it in the heart or in the big bulb, and does that agree with Chapter 5?
7. Why is the Mandelbrot set not a picture of chaos?

Part IV

Chaos in the world

Pendulums and planets

A grandfather clock keeps time because a pendulum is the most regular thing most people will ever own: let it go and it swings at the same beat for as long as the clock is wound. Chapter 4 showed why: a pendulum's state is two numbers, and without friction they go round the same closed loop for ever. The clockwork universe of Chapter 1 was built on this kind of mechanics.

Now hang a second pendulum from the bottom of the first, with a hinge where the two meet, and let go of it from high up. The rule is still Newton's, it is still exact, and nothing in it is random. What does it do? This chapter answers with a measurement, then follows the same question from a pendulum on a desk to the planets round the Sun, where it was first asked.

What you will be able to do

When you have finished this chapter you will be able to:

- step a double pendulum with RK4 and read its state from a printout
- check a simulation of a system without friction by watching its energy, and say what that check does and does not prove
- measure how far apart two nearly identical releases drift, and test whether the answer depends on the size of the step
- tell a gentle release from a wild one, and say what keeping its energy does and does not decide about a system's future
- separate what has been measured about the solar system's chaos from what is folklore

13.1 A pendulum on a pendulum

Take two rods a metre long and hinge the second to the end of the first. Put a one-kilogram bob at the hinge and another at the far end; the bobs carry all the weight and the rods count as weightless. Hold both arms up, 120° from hanging straight down, and let go.

Stop and think 13.1

One pendulum swings with perfect regularity. Do you expect two, one hanging from the other, to settle into some pattern that repeats — more complicated than one pendulum's, but regular all the same?

Answer it before you read on.

Answer. No. Let go from high up, the double pendulum does not settle into any repeating pattern you can find: the lower arm whips round, sometimes right over the top, and the motion goes on changing. Nothing random is involved.

The difference is the number of numbers. One pendulum's state is two: its angle and how fast the angle is changing. A double pendulum's state is four, two angles and two rates of change, and Chapter 4 said that a smooth rule needs at least three numbers to be chaotic.

Newton's laws give a rule of change for the four numbers: given the state now, it says how fast each is changing. It is about fifteen lines of arithmetic with sines and cosines, and you need not read it to use it; it is `double_pendulum` in the book's library, `chaoslab`. Chapter 4's RK4 steps it a thousandth of a second at a time:

code/ch13/release.py

```

16 DT = 0.001                # seconds per step
17 RULE = double_pendulum()  # g = 9.81, arms of 1 m, bobs of 1 kg
18
19
20 def release(upper: float, lower: float) -> tuple[float, ...]:
21     """A pendulum let go from rest, with its arms at these angles (degrees).
22
23     The state is four numbers: the two angles (in radians, measured from
24     straight down) and how fast each angle is changing (radians a second).
25     """
26     return (math.radians(upper), math.radians(lower), 0.0, 0.0)

```

Listing 13.1: The release: four numbers, and the size of one step

`math.radians` turns degrees into radians, the unit Python's sine and cosine expect. A full turn is 2π radians, a little over six, so one radian is about 57° . The listing then steps the state for ten seconds and prints it every two:

code/ch13/release.py

```

32 state = release(120, 120)
33 print(" t (s)   upper arm   lower arm   energy (J)")
34 for step in range(10_001):
35     if step % 2000 == 0:
36         upper = math.degrees(state[0])
37         lower = math.degrees(state[1])
38         energy = double_pendulum_energy(state)
39         print(f"{step * DT:5.1f} {upper:9.1f} deg {lower:7.1f} deg"
40               f" {energy:13.8f}")
41     state = rk4_step(RULE, state, DT)

```

Listing 13.2: Ten seconds of a double pendulum, printed every two

t (s)	upper arm	lower arm	energy (J)
0.0	120.0 deg	120.0 deg	14.71500000
2.0	-109.5 deg	-108.3 deg	14.71500000
4.0	83.2 deg	76.8 deg	14.71500000
6.0	-22.0 deg	-52.7 deg	14.71500000

8.0	-85.5 deg	414.2 deg	14.71499999
10.0	178.5 deg	758.5 deg	14.71499999

Read the arms first. A negative angle is a swing to the other side, and an angle beyond 360° means the arm has gone right round: 758° at ten seconds is two whole turns and 38° more. The lower arm is whirling. Now read the last column.

13.2 The number that must not move

A pendulum with no friction cannot gain or lose energy: nothing pushes it and nothing slows it. Its energy has two parts. Energy of height: each one-kilogram bob contributes $g \times h$, where g is 9.81 and the height h is measured upwards from the pivot. Energy of motion: half of each bob's speed squared. As the pendulum falls, height turns into speed, and back again; the total stays exactly the same.

Stop and think 13.2

Suppose you let go of the pendulum with both arms horizontal, pointing the same way, at 90° . What is its energy, with heights measured from the pivot?

Answer it before you read on.

Answer. Zero. Both bobs are level with the pivot, so both heights are zero, and both are still, so there is no energy of motion either.

Heights below the pivot count as negative, so a pendulum hanging at rest has negative energy; only changes in energy mean anything. For the release in the printout, a little geometry puts the upper bob half a metre above the pivot and the lower bob a whole metre, so the energy is $9.81 \times (0.5 + 1)$, about 14.7 joules, where the energy column starts. Read down that column: over ten seconds of whirling it changes by one unit in its eighth decimal place, a hundred-millionth of a joule.

That is the check, and it is the most useful habit in simulating mechanics. A computer does not solve the rule; it takes small steps, each slightly wrong. The true pendulum keeps its energy exactly, so any change in the simulated energy is the stepper's error, measured.

Lab 13.1 — The number that must not move

Write the double pendulum's energy yourself: heights gives the height of each bob above the pivot, and energy gives the total. The starter explains the one step that needs care: the lower bob is carried by the upper one, so its speed combines two swings. The test compares your energy with the library's, and then watches it along a swing in which it must not move.

Starter: `code/labs/ch13/l13_01_energy.py`

Test: `code/labs/ch13/test_l13_01_energy.py`

Run: `cd code && uv run pytest -k l13_01_energy`

13.3 A millionth of a radian

Now let go of two double pendulums. The second is the first exactly, except that its upper arm starts one millionth of a radian further round: at the end of a one-metre arm, a thousandth of a millimetre, far thinner than a hair. The two obey the same exact rule, with the same step.

Stop and think 13.3

Both pendulums keep their energy, as the check showed, and nothing random touches either. Will the two still swing together at the end of the ten seconds printed above? After a minute? Or will they part, and if so, roughly when?

Answer it before you read on.

Answer. They part. After 10.2 seconds they differ by more than a tenth of a radian, about six degrees, which anyone watching would see. A few seconds later they are doing completely different things.

code/ch13/two_releases.py

```
29 def gap(a: tuple[float, ...], b: tuple[float, ...]) -> float:
21     """How far apart two pendulums are: the larger of the two angle gaps."""
22     return max(abs(a[0] - b[0]), abs(a[1] - b[1]))
```

Listing 13.3: How far apart two pendulums are

code/ch13/two_releases.py

```
28 a = release(120, 120)
29 b = (a[0] + NUDGE, a[1], a[2], a[3])
30 step = 0
31 while gap(a, b) <= APART:
32     if step % 1000 == 0:
33         print(f"t = {step * DT:4.1f} s    gap {gap(a, b):.0e} rad")
34         a, b = rk4_step(RULE, a, DT), rk4_step(RULE, b, DT)
35         step += 1
36 print(f"more than {APART} rad apart after {step * DT:.1f} s")
```

Listing 13.4: Two releases a millionth of a radian apart

```
t = 0.0 s    gap 1e-06 rad
t = 1.0 s    gap 9e-07 rad
t = 2.0 s    gap 7e-06 rad
t = 3.0 s    gap 3e-06 rad
t = 4.0 s    gap 7e-05 rad
t = 5.0 s    gap 3e-05 rad
t = 6.0 s    gap 1e-04 rad
t = 7.0 s    gap 1e-03 rad
t = 8.0 s    gap 8e-03 rad
t = 9.0 s    gap 2e-02 rad
```

t = 10.0 s gap 5e-02 rad
more than 0.1 rad apart after 10.2 s

Python writes $7e-05$ for 7×10^{-5} , seven hundred-thousandths. The gap does not grow smoothly: it rises and falls as the arms swing. But the trend is relentless. From a millionth to a tenth is five factors of ten in about ten seconds, so on average the gap is multiplied by ten every two seconds: Chapter 6's butterfly effect, in a machine you could build on a workbench.

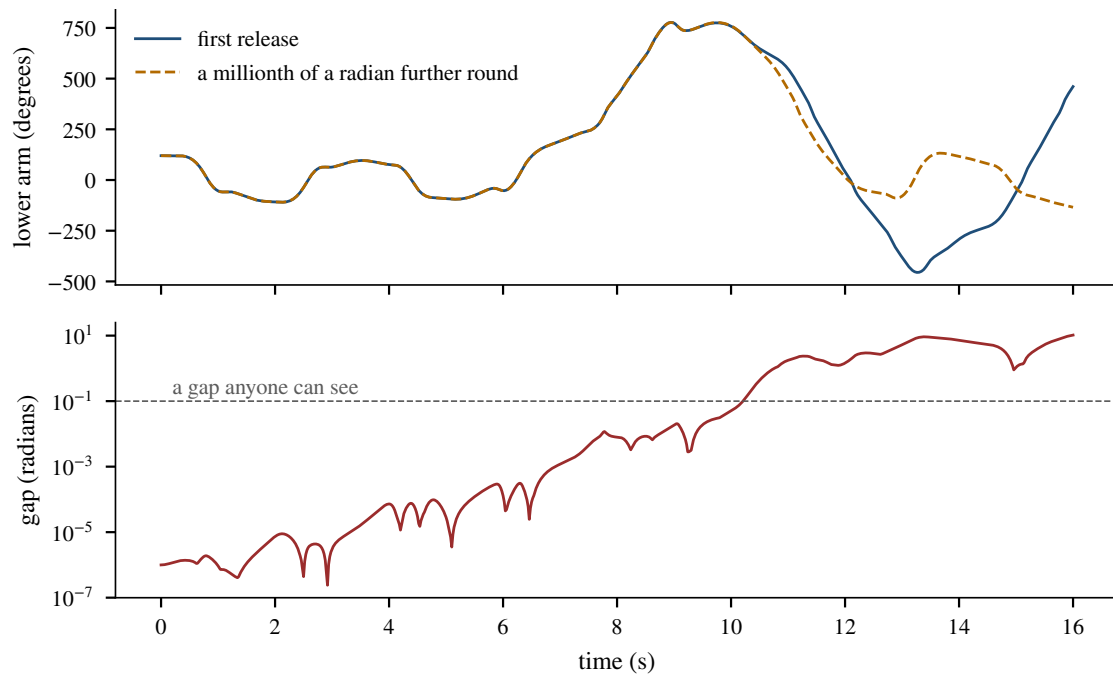


Figure 13.1: Two double pendulums let go a millionth of a radian apart. Above: the lower arm of each, indistinguishable for about ten seconds. Below: the gap between them on a logarithmic scale, climbing unevenly until it is large enough to see.

Here is the doubt that should occur to you. Every step is a little wrong: what if the parting is made by those errors and not by the pendulum? Then a more accurate stepper would move it. So run the same two releases with four step sizes, watching the energy as well:

code/ch13/step_check.py

```

16 def parting(dt: float) -> tuple[float, float]:
17     """Seconds until the pair is APART, and how far the energy moved."""
18     a = release(120, 120)
19     b = (a[0] + NUDGE, a[1], a[2], a[3])
20     start = double_pendulum_energy(a)
21     worst, step = 0.0, 0
22     while gap(a, b) <= APART:
23         a, b = rk4_step(RULE, a, dt), rk4_step(RULE, b, dt)
24         step += 1
25         worst = max(worst, abs(double_pendulum_energy(a) - start))
26     return step * dt, worst

```

Listing 13.5: The same pair, with any step size

step 0.0040 s	apart after 10.21 s	energy moved 2.5e-05 J
step 0.0020 s	apart after 10.21 s	energy moved 1.7e-06 J
step 0.0010 s	apart after 10.21 s	energy moved 1.1e-07 J
step 0.0005 s	apart after 10.21 s	energy moved 6.8e-09 J

Each halving of the step divides the energy error by about 16, as Chapter 4's RK4 recipe promised, and the moment the pair parts does not move by a hundredth of a second. At the book's step of 0.001 seconds the energy has moved by about 10^{-7} joule before the pair parts: about one part in 10^8 . The parting is the pendulum's, not the computer's.

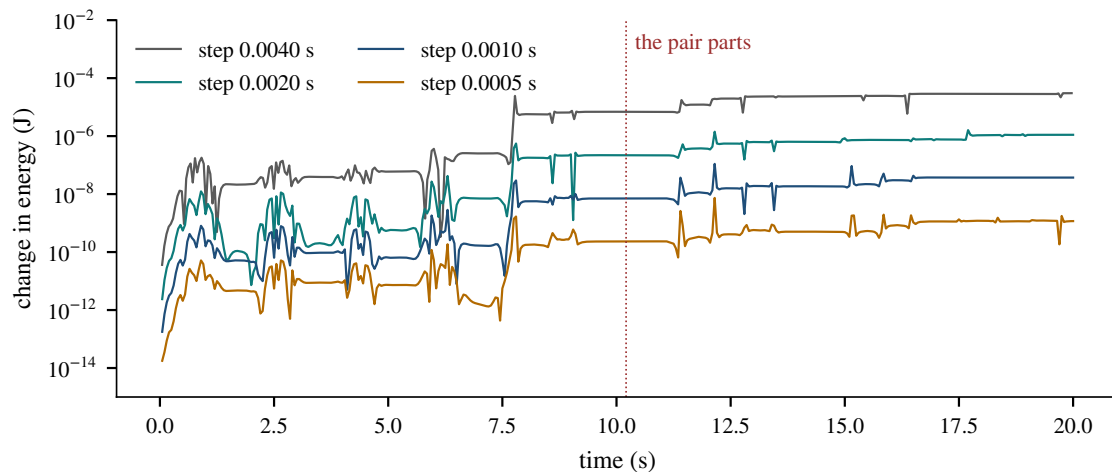


Figure 13.2: How far the simulated energy wanders from its true, constant value, for four step sizes. Each halving of the step lowers the whole curve by about the same factor, and the dotted line marks the moment the two releases part, which is the same for all four.

Note

The check has a limit. Compute one single release twice, with the book's step and with half of it, and the two computations part too, after about 15 seconds. Each step's tiny error is a nudge, and chaos grows it like any other. It starts far smaller than a millionth of a radian, which is why our pair parts first and every step size agrees on when. Chapter 16 asks what can still be trusted after that.

The trap

'A pendulum that keeps its energy, with no friction and nothing random in it, cannot be chaotic.' It needs neither friction nor chance. Keeping its energy pins a pendulum down only partly. One pendulum's state is two numbers; fixing the energy leaves one direction to move in, round one closed loop, which is why it repeats. A double pendulum's state is four numbers; fixing the energy leaves three directions, and three is enough for chaos. The energy check proves the simulation honest. It does not make the motion predictable.

Lab 13.2 — When do two releases part?

Turn the listing into a function. `seconds_until_apart` steps two pendulums side by side and returns the first moment they differ by more than a tolerance, or `None` if they are still together when time runs out. Then use it: how much longer do releases a billionth of a radian apart take to part than releases a millionth apart? Compare your answer with Chapter 7's arithmetic of horizons.

Starter: `code/labs/ch13/l13_02_apart.py`

Test: `code/labs/ch13/test_l13_02_apart.py`

Run: `cd code && uv run pytest -k l13_02_apart`

13.4 Gentle swings and wild ones

Everything so far was a release from high up, with a lot of energy. A double pendulum can also be let go gently.

Stop and think 13.4

Release the same pendulum with both arms at 10° instead of 120° : same rule, same machine, same nudge of a millionth of a radian. Does the gap still grow to something anyone could see within twenty seconds?

Answer it before you read on.

Answer. No. At 10° the two stay within a few millionths of a radian of each other: the nudge barely grows. The same machine is calm at small swings and wild at large ones.

`code/ch13/gentle_wild.py`

```

19 def fate(angle: float) -> tuple[float | None, float]:
20     """When a pair let go at `angle` degrees parts (None if it does not),
21     and the largest gap it reached before then."""
22     a = release(angle, angle)
23     b = (a[0] + NUDGE, a[1], a[2], a[3])
24     largest = gap(a, b)
25     for step in range(1, round(SECONDS / DT) + 1):
26         a, b = rk4_step(RULE, a, DT), rk4_step(RULE, b, DT)
27         if gap(a, b) > APART:
28             return step * DT, largest
29         largest = max(largest, gap(a, b))
30     return None, largest

```

Listing 13.6: What a pair does, released at one angle

10 deg	together; largest gap 1e-06 rad
30 deg	together; largest gap 2e-06 rad
50 deg	together; largest gap 4e-06 rad
70 deg	together; largest gap 8e-06 rad

90 deg	apart after 16.9 s
110 deg	apart after 12.1 s
130 deg	apart after 8.2 s
150 deg	apart after 10.1 s
170 deg	apart after 5.4 s

Up to 70° , every pair is still together after 20 seconds, and even at 70° the largest gap is 8×10^{-6} radians: the nudge has grown several times over, not a hundred thousand times. From 90° on, every pair parts. The plot fills in the angles between.

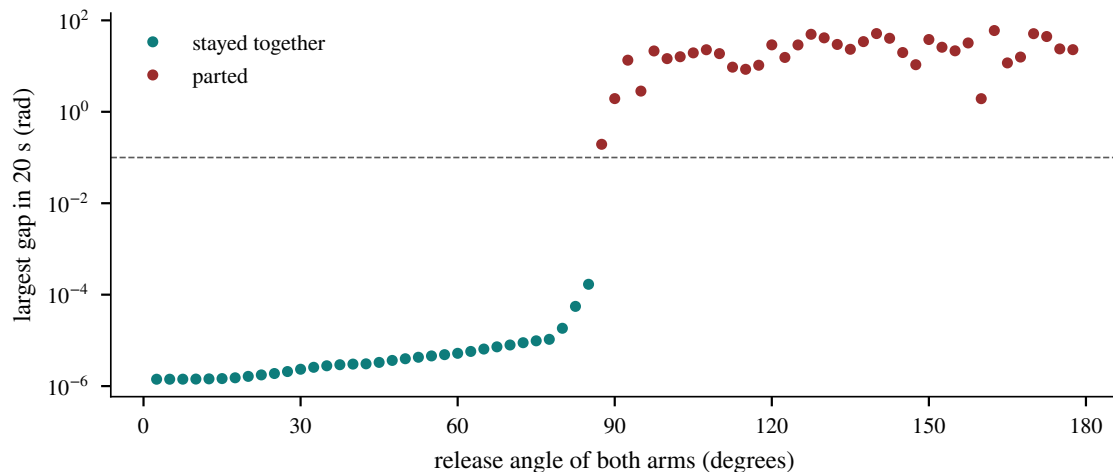


Figure 13.3: Each dot is a release of both arms at one angle, with a copy a millionth of a radian further round. Small releases stay together for twenty seconds; large ones part, and the change comes over a few degrees.

This is how chaos lives in a system without friction. With little energy the double pendulum is nearly two simple swings, the arms moving together or against each other, and its motion is regular. Let it go from higher up and chaos appears; higher still, and chaos takes over, with regular motion surviving only in pockets: a chaotic sea with islands of regular motion in it.

What we are not proving

This book does not prove that the double pendulum is chaotic; it measures sensitivity at a handful of releases. The theorem behind the sea and its islands is called KAM, after Andrey Kolmogorov, who announced it in 1954, and Jürgen Moser and Vladimir Arnold, who proved versions of it in 1962 and 1963. It says that when a regular, friction-free system is disturbed a little, most of its regular motions survive, slightly bent, and chaos is confined to thin layers between them; as the disturbance grows, the layers widen. Its proof is graduate mathematics, found in books on the mechanics of such systems.

One more difference from Chapter 9. Lorenz's equations have friction in them, and friction squeezes every start onto one attractor. The double pendulum has none: every start keeps its own energy for ever, and there is no attractor. There is a fence instead. The energy says which states can ever be reached, and the motion, however wild, stays inside.

13.5 Two bodies, three bodies

The clockwork of Chapter 1 was the solar system, and its triumph was the problem of two bodies: one planet and the Sun. Johannes Kepler found early in the seventeenth century that planets move on ellipses, and Isaac Newton showed in 1687 that his law of gravity makes two bodies do exactly that. Two bodies have a formula: an ellipse, traced for ever. Every question about them has an answer you can write down.

Add a third body and there is no formula of that kind at all. In 1889 Henri Poincaré won a prize offered in the name of King Oscar II of Sweden for work on the problem of several bodies moving under their own gravity. While his prize memoir was being prepared for publication, an error was found in it. Correcting it, Poincaré found that the orbits near a simple, repeating motion of three bodies can wind about it in a tangle so intricate that he later wrote he would not even try to draw it. The corrected memoir appeared in 1890, and what it describes is recognised today as the first glimpse of chaos: a rule as exact as Newton's, followed by motion that no formula follows.

In 1908, writing for the general reader, Poincaré put the lesson in words this book has been measuring: a cause too small to notice can decide an effect we cannot miss, and then prediction becomes impossible.

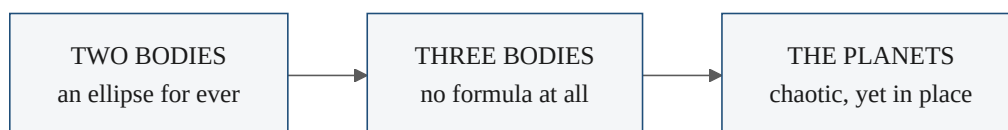


Figure 13.4: Two bodies have a formula; three have none; the planets are chaotic over tens of millions of years and yet stay where they are.

13.6 Is the solar system a clock?

For two centuries the solar system was the model of a perfect clock. In the late eighteenth century Joseph-Louis Lagrange and Pierre-Simon Laplace calculated, to the accuracy their methods allowed, that the sizes of the planets' orbits wobble but do not drift. Laplace's demon in Chapter 1 had the planets as its evidence.

In 1989 Jacques Laskar computed the slow wobbles of the planets' orbits over two hundred million years and found the motion of the inner planets — Mercury, Venus, Earth and Mars — chaotic. The Lyapunov time he found, in the sense of Chapter 7, was about five million years: an error in where the planets are is multiplied by e , about 2.7, every five million years.

Stop and think 13.5

A hundred million years is twenty of those Lyapunov times. By roughly how much is an error multiplied over twenty of them: a hundred times, a million times, or a billion times?

Answer it before you read on.

Answer. About 4.9×10^8 times: nearer a billion than a million. At that rate, an error of one metre in where Earth is today would grow to 490 000 kilometres, more than the distance to the Moon.

The trap

‘The solar system is a perfect clock.’ Over a few thousand years it is as good a clock as anything we have: eclipses are predicted centuries ahead. Over tens of millions of years it cannot be read. No measurement and no computer can say where Earth will be on its orbit a hundred million years from now. The rule is exact; the start is not.

Unpredictable is not the same as unstable. Laskar’s result is not that the planets fly apart, and longer computations since, over billions of years, find them keeping to orbits of much the same size and shape in almost every run. We cannot say where Earth will be on its orbit in a hundred million years; we can say it will very probably be on an orbit much like this one. That is Chapter 9’s lesson in a new place: the path cannot be predicted, and the region it moves in can.

Chaos shows itself faster in some corners of the solar system. Saturn’s moon Hyperion is a lumpy body on a stretched orbit, and calculations predict that its spin cannot settle into a steady rotation but tumbles chaotically, tugged one way and then another by Saturn. Observations from the Voyager and Cassini spacecraft fit that prediction: Hyperion has no steady day.

Where you meet this

Geologists date layers of sediment by the climate rhythms written into them. Slow cycles in the shape of Earth’s orbit and the tilt of its axis, named after Milutin Milanković, change how sunlight falls on the planet, and the climate leaves matching bands in the rock. To turn bands into dates you need Earth’s orbit far into the past, and geologists take it from computations by Laskar’s group. Those computations are trusted, cycle by cycle, back to about fifty million years and not further, because of the chaos in this section.

Summary

- A **double pendulum**’s state is four numbers, two angles and two rates of change. Its rule is Newton’s, exact and without chance, and RK4 steps it.
- Without friction the **energy** cannot change, so a simulation’s energy measures the stepper’s error: at the book’s step, about one part in 10^8 .
- Releases a millionth of a radian apart part after 10.2 seconds at every step size: the **parting is physics**. Keeping energy does not prevent chaos.
- Gentle releases are **regular** and wild ones chaotic: a sea with islands, as the KAM theorem describes. With no friction there is no attractor; the energy fences the motion in.

- Two bodies have a formula and three do not. **Poincaré's** corrected prize memoir of 1890 was the first glimpse of chaos.
- Laskar found the inner planets chaotic, with a **Lyapunov time** of about five million years: their positions cannot be predicted a hundred million years ahead, yet their orbits stay much as they are.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

step a double pendulum with RK4 and read its state from a printout	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
check a simulation of a system without friction by watching its energy, and say what that check does and does not prove	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
measure how far apart two nearly identical releases drift, and test whether the answer depends on the size of the step	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
tell a gentle release from a wild one, and say what keeping its energy does and does not decide about a system's future	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
separate what has been measured about the solar system's chaos from what is folklore	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Why can a double pendulum be chaotic when a single pendulum cannot, though neither loses energy?
2. A double pendulum is let go from rest with both arms pointing straight up. What is its energy, measuring heights from the pivot?
3. Your simulation of a machine without friction loses a tenth of its energy in a minute. What do you conclude?
4. Two releases part after ten seconds. You halve the step and they now part after three. What does that tell you?
5. The double pendulum has no attractor. What keeps its motion from going anywhere at all?
6. With a Lyapunov time of five million years, by roughly what factor is an error multiplied in ten million?
7. Does the chaos of the solar system mean the planets will fly apart?

Weather, climate and the ensemble forecast

The forecast on your phone says there is a 70 per cent chance of rain tomorrow afternoon. Seventy per cent of what? Not of the sky, and not of the afternoon: nobody expects rain on seven tenths of your town for seven tenths of the time. The number is a count, and by the end of this chapter you will have made one yourself, on a small model of an atmosphere, and checked whether it can be trusted.

This is where Part II's bad news becomes a method. The atmosphere is chaotic, so a single forecast has a horizon, and Chapter 7 showed that better measurements move it only a little. Forecasters could change neither fact. What they changed was the question they answered — from one forecast to many — and it is the most successful practical use chaos theory has had.

What you will be able to do

When you have finished this chapter you will be able to:

- run an ensemble forecast on a toy atmosphere and read its spread as a measure of confidence
- turn an ensemble into a probability, and say how a probability forecast is judged
- measure how far ahead a forecast beats quoting the climate average
- use the rule of Chapter 7 to say what a better start buys a forecast
- tell a question about the weather from a question about the climate

14.1 A toy atmosphere

A weather model divides the air into boxes, gives each box a few numbers — temperature, pressure, wind, moisture — and uses the physics of fluids to say how each changes. The first forecast made this way on an electronic computer ran on ENIAC in 1950; today's global models carry many millions of numbers.

In 1996 Edward Lorenz, whose three equations you met in Chapter 9, proposed a model small enough to think about and big enough to behave like weather. Space 40 numbers, x_1 to x_{40} , round a circle of latitude, so that each stands for the weather at one place and the last one's neighbour is the first. Read the left-hand side as Chapter 4 taught you, as how fast x_i is changing:

$$\frac{dx_i}{dt} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F.$$

In words: each number is carried along by its neighbours (the product, which moves weather round the circle as wind does), leaks away (the $-x_i$, which plays friction), and is pushed by a steady drive F , the *forcing*, which plays the sun. Lorenz's standard forcing is $F = 8$, and in `chaoslab` the whole rule is one line:

code/src/chaoslab/systems.py

```

124 def lorenz96(forcing: float = 8.0) -> Field:
125     """Lorenz's 1996 toy atmosphere: n numbers round a circle of latitude.
126
127     Each one is pushed by its neighbours, damped, and driven by `forcing`.
128     The number of variables is the length of the state you pass in.
129     """
130
131     def field(s: State) -> State:
132         n = len(s)
133         return tuple((s[(i + 1) % n] - s[i - 2]) * s[i - 1] - s[i] + forcing
134                     for i in range(n))
135
136     return field

```

Listing 14.1: Lorenz's forty-number toy atmosphere

The circle closes by itself: when i is 0, Python reads $s[i - 2]$ as $s[-2]$, the second number from the end.

Stop and think 14.1

Put every number at 8, the value of the forcing. What does the rule say each of them does next?

Answer it before you read on.

Answer. Nothing. The neighbours cancel, $(8 - 8) \times 8 = 0$, and the leak of 8 cancels the push of 8, so every rate of change is zero.

A perfectly calm atmosphere stays calm. Calm is a fixed point, and as in Chapter 3 the question is whether it is stable. It is not: the listings below nudge one number by a hundredth and wait, and the calm breaks into waves that travel round the circle and never settle.

Lorenz chose his units so that one time unit is roughly five days of real weather, which makes the listings' step of 0.05 about six hours. Measured with `lyapunov_flow` from `chaoslab`, the exponent of Chapter 7 is 1.70 per time unit: a small error doubles every 0.41 units, about two days.

14.2 Twenty forecasts instead of one

A forecast starts from an *analysis*: the best estimate of the state of the atmosphere now, built from measurements that are never exact. Say the analysis is right to within 0.1 on every one of the forty numbers. You know from Chapter 6 what a single forecast from it will do: follow the real weather for a while, and then not.

A weather service cannot know tomorrow's truth while it forecasts. We can, because we play nature ourselves. Run the model once and call that run the *truth*; make an analysis by adding an error to it; forecast from the analysis; score the forecast against the truth. Model and truth obey the same rule, so every error below comes from the start and none from the model: the toy has every advantage a real forecast lacks.

Now the new idea. Instead of one forecast, make 20. Each *member* starts from the analysis nudged by another error of the same size, so the members differ from one another about as much as the analysis could differ from the truth. Together they are an *ensemble*.

code/ch14/ensemble.py

```

20 N = 40          # numbers round the circle of latitude
21 FORCING = 8.0   # the drive: Lorenz's standard value
22 DT = 0.05       # one step; about six hours in Lorenz's units
23 ERROR = 0.1     # the analysis is right to within 0.1 on every number
24 MEMBERS = 20    # copies in the ensemble
25
26
27 def run(state: State, steps: int, forcing: float = FORCING) -> State:
28     """Follow the toy atmosphere for a number of steps."""
29     field = lorenz96(forcing)
30     for _ in range(steps):
31         state = rk4_step(field, state, DT)
32     return state
33
34
35 def spin_up(forcing: float = FORCING, nudge: float = 0.01) -> State:
36     """A calm atmosphere with one number nudged, left to make weather."""
37     calm = (forcing + nudge,) + (forcing,) * (N - 1)
38     return run(calm, 2000, forcing)
39
40
41 def perturb(state: State, size: float, rng: random.Random) -> State:
42     """Add an error no bigger than `size` to every number."""
43     return tuple(x + rng.uniform(-size, size) for x in state)

```

Listing 14.2: The toy atmosphere, a calm start, and an error of a given size

`rng.uniform(-size, size)` picks a number between `-size` and `size`, every value equally likely, so `perturb` adds a different small error to each number. Three scores do the rest. The *ensemble mean* averages each number over the members. The *spread* is how far the members typically sit from that mean: square each difference, average the squares, take the square root. The *error* is the same recipe applied to a forecast and the truth. So spread and error are the same kind of number, measured against each other or against the truth.

Stop and think 14.2

The members start within a few hundredths of one another. As the forecast runs on, does their spread keep growing for ever?

Answer it before you read on.

Answer. No. It grows quickly and then levels off: the members cannot end up further apart than two unrelated days of weather, and the attractor bounds how far apart those can be.

code/ch14/ensemble.py

```

73 def ensemble(truth: State, rng: random.Random) -> tuple[State, list[State]]:
74     """An analysis of `truth`, and MEMBERS copies of it, each nudged."""
75     analysis = perturb(truth, ERROR, rng)
76     return analysis, [perturb(analysis, ERROR, rng) for _ in range(MEMBERS)]
77
78
79 def forecast(truth: State, rng: random.Random,
80             leads: range) -> list[tuple[float, float, float, float]]:
81     """Score an ensemble forecast at each lead (counted in steps): the
82     spread, the error of the ensemble mean, and the error of one run
83     from the analysis alone."""
84     single, members = ensemble(truth, rng)
85     done, rows = 0, []
86     for lead in leads:
87         gap = lead - done
88         truth = run(truth, gap)
89         single = run(single, gap)
90         members = [run(m, gap) for m in members]
91         done = lead
92         rows.append((lead * DT, spread(members),
93                     error(mean(members), truth), error(single, truth)))
94     return rows

```

Listing 14.3: An ensemble forecast, scored at each lead time

lead	spread	error of the mean	error of one run
0.0	0.06	0.06	0.06
1.0	0.42	0.35	0.31
2.0	1.53	1.35	1.03
3.0	2.49	2.38	3.78
4.0	3.10	2.89	3.93
5.0	3.32	3.58	5.01
6.0	3.44	3.69	4.74
7.0	3.54	3.76	5.34
8.0	3.54	3.18	4.49

The *lead* is how far ahead the forecast looks, in time units. For the first unit or two the spread is small and the ensemble mean is close to the truth; by six the spread has stopped

growing, because the members have forgotten their start. The single run from the analysis, in the last column, is as good as the mean at first and much worse later. Here is site 0 for every member, drawn against the truth:

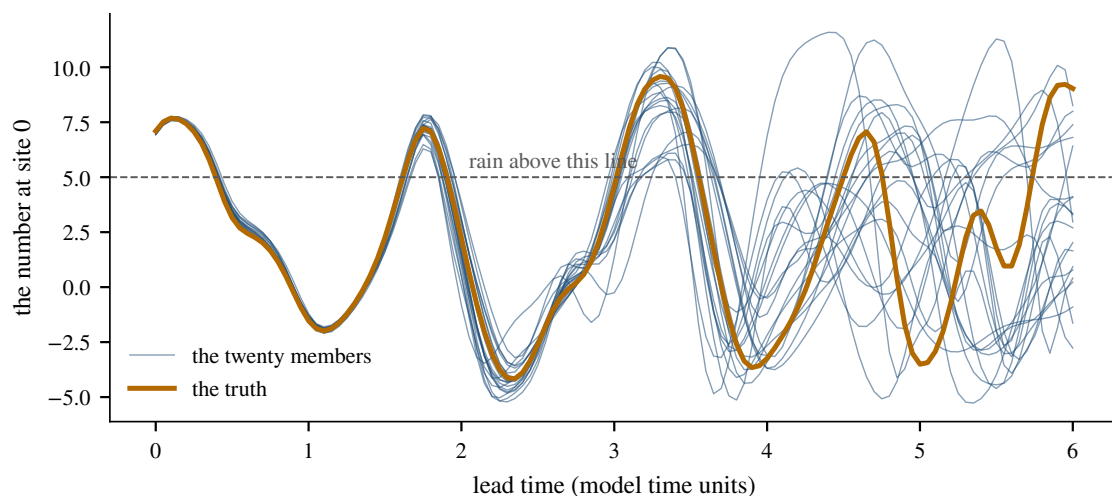


Figure 14.1: The twenty members of the forecast above at site 0, with the truth in amber. They move almost as one for nearly three time units, then fan out; by five they are scattered across the weather's whole range.

Stop and think 14.3

In a real forecast you cannot see the truth, so you cannot compute the two error columns. Look at the table: could the spread, which you can compute, tell you how large the error of the mean is?

Answer it before you read on.

Answer. Yes, roughly. Averaged over 40 forecasts from different days, the spread and the error of the mean are 0.47 and 0.45 at a lead of one time unit, and 2.84 and 2.92 at three.

The truth is, as far as anyone can tell, one more member: the analysis differs from it by an error of the same size as the members' nudges. So it sits about as far from the ensemble mean as they do, and the width of the plume measures the size of the mistake without anyone seeing the truth.

The trap

'A forecast is a single number.' A single number is the least a forecast can tell you. The ensemble gives a number and a warning with it: on a day the members agree, trust the mean; on a day they scatter, the same number deserves a shrug. The width of the plume changes from day to day with the weather itself, and it says how far ahead the forecast is worth reading.

A weather centre does this every day and closes the loop: today's analysis is made by

blending new measurements with yesterday's forecast.

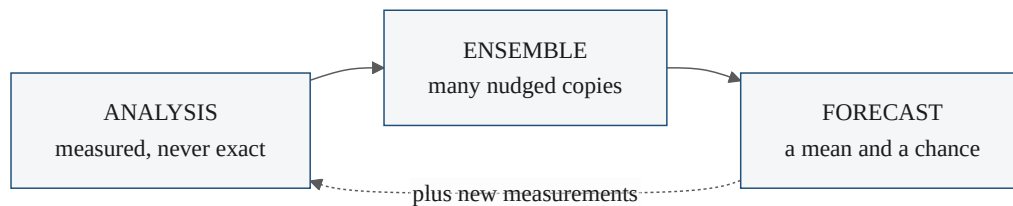


Figure 14.2: The forecast cycle. An inexact analysis becomes many nudged copies; their spread and their counts become the forecast; tomorrow the forecast meets new measurements and becomes the next analysis.

14.3 A chance of rain

Now the seventy per cent. Call site 0 your town, and say it rains there whenever its number is above 5. Lorenz's numbers are not rain; this is a rule for turning a number into weather, and nothing below depends on the choice. In a long run it rains on 25 per cent of occasions: that is the climate of your town, measured in the last section.

Stop and think 14.4

At some lead time, 15 of the 20 members show rain at your town. What is the forecast chance of rain?

Answer it before you read on.

Answer. 75 per cent: 15 out of 20 is three quarters.

That is all a chance of rain is: the fraction of equally plausible futures in which it rains. Here it is for the forecast you have been following.

code/ch14/chance_of_rain.py

```

15 RAIN = 5.0 # it rains at a site whenever its number is above this
16
17
18 def chance(members: list[State], site: int = 0) -> float:
19     """The fraction of the members in which it rains at `site`."""
20     wet = sum(1 for state in members if state[site] > RAIN)
21     return wet / len(members)
  
```

Listing 14.4: A chance of rain is a count of members

lead	members	raining	chance	did it rain?
0.5	0	of 20	0%	no
1.0	0	of 20	0%	no
1.5	0	of 20	0%	no
2.0	0	of 20	0%	no
2.5	0	of 20	0%	no
3.0	3	of 20	15%	no
3.5	15	of 20	75%	yes
4.0	1	of 20	5%	no
4.5	5	of 20	25%	yes
5.0	7	of 20	35%	no
5.5	6	of 20	30%	no
6.0	5	of 20	25%	yes

For the first few time units every member says dry, and it stays dry. Then they split. At one lead three members in four say rain, and it rains; later only one in four does, and it rains then too. By the end the chances hover near the climate of your town: the forecast has forgotten its start.

Lab 14.1 — A chance of rain, and keeping score

Write `chance`, which turns a list of members into a probability, and `hit_rate`, which scores many forecasts at once: of those that gave a chance within a range, what fraction were followed by rain? One test runs the chapter's own forecast and expects your `chance` to agree with the table above.

Starter: `code/labs/ch14/l14_01_chance.py`

Test: `code/labs/ch14/test_l14_01_chance.py`

Run: `cd code && uv run pytest -k l14_01_chance`

Stop and think 14.5

Near the end of the table the ensemble gave rain one chance in four, and it rained. Was the forecast wrong?

Answer it before you read on.

Answer. You cannot tell from one day. A chance of one in four says that on about a quarter of such days it rains, and this may be one of them.

The trap

‘A 30 per cent chance of rain, followed by rain, was a wrong forecast.’ A good forecast of that kind comes true about three times in ten, and this was one of them. What would make it wrong is a record: if, over hundreds of forecasts of 30 per cent, it rained on 10 per cent of them, or 60. Forecasters call the check *reliability*, and it can only be made over many forecasts, never over one.

So keep score. The chapter’s measurement makes 6000 forecasts, at every site on many different days, each two time units ahead. Of the 380 occasions on which the ensemble gave rain a chance between 25 and 35 per cent, it rained on 31 per cent. Of the 243 on which it gave between 65 and 75 per cent, it rained on 68 per cent. The percentages mean what they say.

Where you meet this

Every day the European Centre for Medium-Range Weather Forecasts, based in Reading, runs an ensemble of about fifty members of its global model, each from a slightly different analysis, out to fifteen days ahead. Weather services around the world turn ensembles like it, often after a statistical correction, into the percentages on your phone and into warnings of storms and floods. Ensembles have been run every day for more than thirty years, and the chance of rain on your phone is, behind the scenes, a count much like the one in the listing above.

14.4 How far ahead?

One forecast needs no model at all: always quote the climate. Tomorrow, at any site, the number will be about 2.3, give or take 3.6 — the long-run average and typical swing that the last section measures. That forecast is never good, but its typical error is never worse than the swing. A forecast has *skill* only while it beats it.

Stop and think 14.6

Long after the horizon, the single run from the analysis is one more day of weather with no connection to the truth. Which is then the better forecast: that run, or the climate average?

Answer it before you read on.

Answer. The climate average. Two unrelated days of weather differ by more than either differs from the average: by about $\sqrt{2}$, that is 1.4, times the swing.

Measured, the single run’s error settles at 5.1, and it becomes worse than the climate average at a lead of 2.85 time units — in Lorenz’s scaling, about a fortnight. A forecast that is confidently wrong is worse than one that knows nothing. The ensemble mean never crosses that line: as the members forget their start it drifts towards the climate average

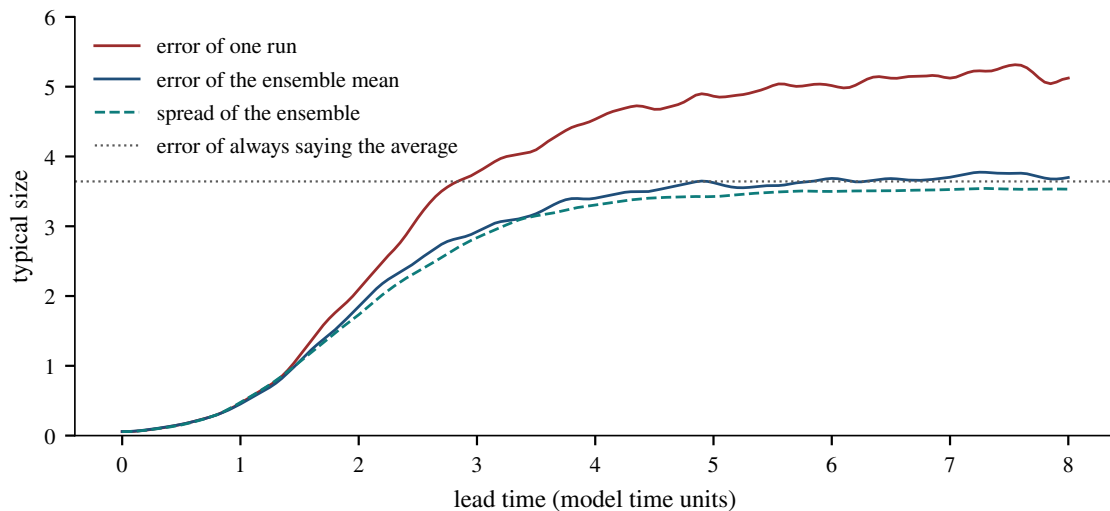


Figure 14.3: Spread and errors against lead time, each averaged over forty forecasts. The spread (dashed) tracks the error of the ensemble mean; the single run ends well above the dotted line, the error of always quoting the average.

itself, and its error settles at 3.7, a shade above the swing because twenty members are not infinitely many.

Lab 14.2 — When does the ensemble stop spreading?

Write `spread` for a whole ensemble, then `saturation_lead`: the first lead at which the spread reaches nine tenths of the level where it settles, which averaged over forty forecasts is 3.5. The test runs the chapter's forecast through your code. One forecast is one roll of the dice, so expect your lead to differ from the forty-forecast average, 3.60 time units.

Starter: `code/labs/ch14/l14_02_saturation.py`

Test: `code/labs/ch14/test_l14_02_saturation.py`

Run: `cd code && uv run pytest -k l14_02_saturation`

Stop and think 14.7

Suppose the analysis were ten times better, right to within a hundredth instead of a tenth. Using the rule of Chapter 7, roughly how much later would the single run lose to the climate?

Answer it before you read on.

Answer. About $\ln 10$ divided by the exponent later, whatever the starting error: $\ln 10 \approx 2.3$, and divided by 1.70 that is 1.35 time units.

Measured, the horizon moves from 2.85 to 4.25, and with an analysis ten times better again, to 5.75: gains of 1.40 and 1.50, close to the rule. Each tenfold improvement buys the same extra stretch, never ten times as much.

The real atmosphere is less generous than the toy, and Lorenz explained why in 1969. It has motions on many scales, from the width of an ocean down to a gust; the small ones go wrong fastest, and their errors spread upwards into the large ones. Better measurements delay the damage but do not stop it. His conclusion, still the commonly quoted figure, is that detailed weather cannot be predicted more than about two weeks ahead, however good the data. Forecasts have improved steadily all the same — better observations, not least from satellites, better models, bigger computers — but the gain is usually put at about one day of useful lead time per decade: the logarithm in the rule of Chapter 7, paid for in decades.

What we are not proving

The two-week figure is not derived in this book, and the toy's horizon, which moves whenever its analysis error does, is not a measurement of it. The figure comes from estimates of how fast errors grow at each scale of the real atmosphere, begun by Lorenz in 1969 and refined since by comparing forecasts with what then happened. The toy shows why there is a horizon.

14.5 Weather and climate

If nobody can say whether it will rain in your town three weeks from now, can anybody say how often it will rain there next year?

Stop and think 14.8

Run the model twice from different starts, each for 500 time units, a few years of Lorenz's weather. After the first few units their weather has nothing in common. Will their long-run averages, their typical swings and how often it rains differ too?

Answer it before you read on.

Answer. No. Both runs give an average of 2.3, a swing of 3.6 and rain on 25 per cent of occasions to the figures printed, while on their last step site 0 reads completely different numbers.

code/ch14/climate.py

```

22 def climate(forcing: float, nudge: float,
23             steps: int = 10000) -> tuple[float, float, float, float]:
24     """Average, typical swing and rain frequency over a long run, and
25     the number at site 0 when the run ends."""
26     state = spin_up(forcing, nudge)
27     field = lorenz96(forcing)
28     total = squares = 0.0
29     wet = count = 0
30     for _ in range(steps):
31         state = rk4_step(field, state, DT)
32         for x in state:
33             total += x
34             squares += x * x

```

```

35     if x > RAIN:
36         wet += 1
37         count += 1
38     average = total / count
39     swing = math.sqrt(squares / count - average * average)
40     return average, swing, wet / count, state[0]

```

Listing 14.5: Boiling a long run down to three statistics

forcing	nudge	average	swing	rain	last x[0]
8	0.01	2.3	3.6	25%	1.44
8	1.00	2.3	3.6	25%	-2.06
10	0.01	2.6	4.4	30%	10.70

This is Chapter 9’s discovery on a bigger model: the path is unpredictable, the attractor is not, and how often the path visits each region belongs to the rule rather than to the start. Weather is the path. Climate is the statistics of the attractor. Forecasting the weather is an *initial-value* problem, whose answer depends on where you start and which chaos gives a horizon. The climate does not depend on the start at all.

The trap

‘If the weather cannot be predicted, the climate cannot be predicted either.’ They answer different questions. Chaos limits questions about the path: will it rain on this day? Questions about the statistics — how often does it rain, how widely does the weather swing — have stable answers, and the two runs above disagree about every day while agreeing about the climate.

What does change the climate is the rule. The third row of the transcript turns the forcing up from 8 to 10, more sunshine in the toy, and the statistics move: the average becomes 2.6, the swing 4.4, and it rains on 30 per cent of occasions instead of 25. The weather on any single day is exactly as unpredictable as before.

That is the shape of a climate projection. A climate model is run not from today’s weather but under a changed forcing — in the real case, a changed composition of the atmosphere — and asked how the statistics move. It is called a *projection* rather than a prediction because it is conditional on the forcing assumed. The forcing is a *boundary condition*: it belongs to the rule, not to today’s state, so the weather’s horizon does not rule the projection out. Whether a particular climate model gets the shift right is a different question, answered by testing it against past climates, which this toy cannot settle. What it shows is the logic: an initial condition decides the weather, a boundary condition the climate.

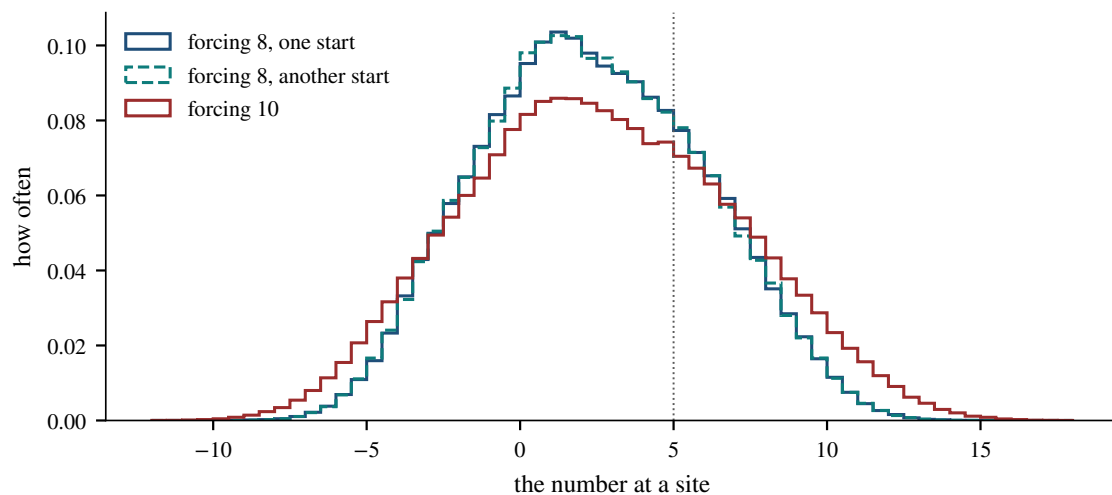


Figure 14.4: How often each value occurs over a long run. The two runs with forcing 8, from different starts, lie on top of each other; with forcing 10 the distribution widens and more of it lies above the rain line at 5.

Summary

- Lorenz's 1996 **toy atmosphere** puts forty numbers round a circle, pushed by their neighbours, damped, and driven by a **forcing**. Its calm state is unstable, and its errors double in about 0.41 time units.
- An **ensemble** is many forecasts from nudged copies of the **analysis**. Its **spread** grows with lead time, levels off, and tracks the error of the ensemble mean without anyone seeing the truth.
- A **chance of rain** is the fraction of members in which it rains. A probability forecast is judged by **reliability**, over many forecasts, never over one day.
- A forecast has **skill** while it beats the climate average. A single run loses to it at the **horizon** and ends about $\sqrt{2}$ times worse; each tenfold better analysis moves the horizon by the same amount, and the commonly quoted limit for real weather is about two weeks.
- **Weather** is the path and depends on the start; **climate** is the statistics of the attractor and does not. A change of forcing, a **boundary condition**, moves the statistics, which is what a **climate projection** asks about.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

run an ensemble forecast on a toy atmosphere and read its spread as a measure of confidence	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
turn an ensemble into a probability, and say how a probability forecast is judged	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
measure how far ahead a forecast beats quoting the climate average	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
use the rule of Chapter 7 to say what a better start buys a forecast	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
tell a question about the weather from a question about the climate	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. What is an analysis, and why does a forecast from it go wrong even when the model is perfect?
2. An ensemble of 50 members has 12 showing rain. What is the chance of rain?
3. Over a year a forecaster said '80 per cent chance of rain' 100 times, and it rained 55 times. Were those forecasts reliable?
4. Long after the horizon, why is a single forecast worse than quoting the climate average, and by roughly how much?
5. In the toy atmosphere the analysis error is made a hundred times smaller. By about how much does the horizon move?
6. Weather or climate: will it snow in Warsaw next Christmas Day? How many snowy December days will Warsaw have around 2050 under a given forcing?

Life, markets and noise: telling chaos from chance

A cardiologist looks at the gaps between heartbeats, an epidemiologist at a century of monthly measles counts, a trader at a price. Each sees a wiggly line that never quite repeats, and wants to know what makes it wiggle: a rule, with a few numbers pushing each other about, or chance, countless small causes no model will ever list. Most often it is some of both.

Chapter 5 showed that a rule one line long can produce output as irregular as a lottery. This chapter asks the question the other way round: handed a wiggly series and no rule, can you tell whether a rule made it? You will build two tests, measure how well they work when the data are noisy and short, and then visit the places where chaos has been claimed. The message is not that chaos is everywhere, nor that it is nowhere. It is that a claim of chaos can be checked.

What you will be able to do

When you have finished this chapter you will be able to:

- make a chaotic series and a random one out of exactly the same numbers, and say why no histogram can tell them apart
- draw a return map, count the squares it lands in, and read the count
- forecast a series from its own past by the method of analogues, and measure how the error grows with the number of steps ahead
- say how measurement noise and a short record weaken both tests
- judge a claim of chaos in populations, epidemics, hearts or markets by the evidence offered for it

15.1 Two series, one set of numbers

Start with a puzzle. Two series each hold 2000 numbers between 0 and 1. One was made by the logistic rule of Chapter 5 with its knob at $r = 4$: if x_n is the value at step n , the next one is

$$x_{n+1} = 4x_n(1 - x_n),$$

four times the value times one minus the value. The other series is random. Both are printed below to four decimal places; the program that made them keeps about sixteen significant figures.

Stop and think 15.1

Series A begins 0.0169, 0.8488, 0.6610, 0.6086. Series B begins 0.2000, 0.6400, 0.9216, 0.2890. Which one did the rule make? Can you tell by looking, and can you tell by a calculation?

Answer it before you read on.

Answer. B. By eye you cannot tell. By calculation you can:
 $4 \times 0.2 \times 0.8 = 0.64$, which is B's second value, and $4 \times 0.64 \times 0.36$ is 0.9216, its third. For A the rule gives about 0.07 from the first value, and the second value is 0.8488.

Look at what the test used. Not the size of the numbers, not how often each size turns up, but their *order*: whether each value follows from the one before. That is the whole of this chapter.

The trap

'A random-looking sequence must come from randomness.' Series B looked every bit as random as A, and it came from one line of arithmetic with no chance in it. Looking random is a fact about the look. It says nothing about the cause until you have tested for the kind of order a rule leaves behind.

Series A was not made by rolling dice. It was made from series B, by shuffling:

code/ch15/twins.py

```

20 def rule_series(n: int = N, x0: float = X0) -> list[float]:
21     """n values of the logistic map x -> 4x(1 - x), starting at x0."""
22     return orbit(logistic(4.0), x0, n - 1)
23
24
25 def shuffled(xs: list[float], seed: int = SEED) -> list[float]:
26     """The same numbers in a random order: a surrogate."""
27     copy = list(xs)
28     random.Random(seed).shuffle(copy)
29     return copy

```

Listing 15.1: Two series from one: a rule, and the same numbers shuffled

The first function iterates the rule; the second shuffles a copy of the list with Python's random number generator, started from a fixed *seed* so that the shuffle is the same on every computer and your run matches the page. A series built from another by scrambling its order is called a *surrogate*: the hardest random series to tell from the original, because it has exactly the same numbers.

Stop and think 15.2

How will the histogram of the shuffled series — how many of its values fall in each fifth of the range from 0 to 1 — compare with that of the rule's series?

Answer it before you read on.

Answer. Identical, bar for bar. A histogram counts values and ignores their order, and a shuffle changes only the order.

```
rule shuffled
0.2000 0.0169
0.6400 0.8488
0.9216 0.6610
0.2890 0.6086
0.8219 0.6066
counts in five slices of width 0.2:
rule [582, 282, 250, 289, 597]
shuffled [582, 282, 250, 289, 597]
```

Note the shape: 582 of the 2000 values sit in the lowest fifth and only 250 in the middle one, because the rule lingers near 0 and 1. The shuffle has the same lopsided histogram, and the same average, spread, largest and smallest value. No summary that ignores order can tell the twins apart.

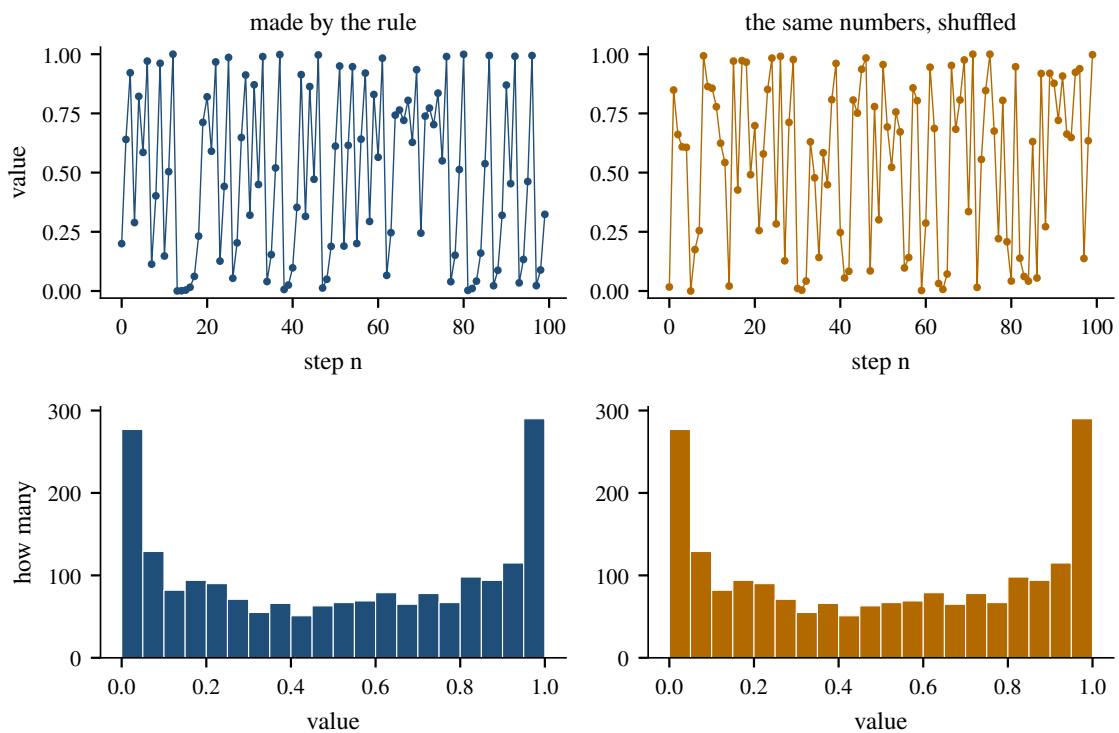


Figure 15.1: The rule's series and its shuffle. Top: the first hundred steps of each, equally ragged. Bottom: their histograms, which are identical.

15.2 Plot each value against the next

In the puzzle you could test each value against the one before because you were told the rule. With real data nobody tells you, so let the data draw the rule for you. Put a point at (x_n, x_{n+1}) for every step n : across by this value, up by the next. The picture is called a *return map*.

Stop and think 15.3

Where will the points of the rule's return map lie? And the shuffle's?

Answer it before you read on.

Answer. The rule's points lie on one curve, the hump of $y = 4x(1 - x)$, because every next value *is* the rule applied to the value before. The shuffle's points spread over the whole square, because in a shuffle the next value has nothing to do with this one.

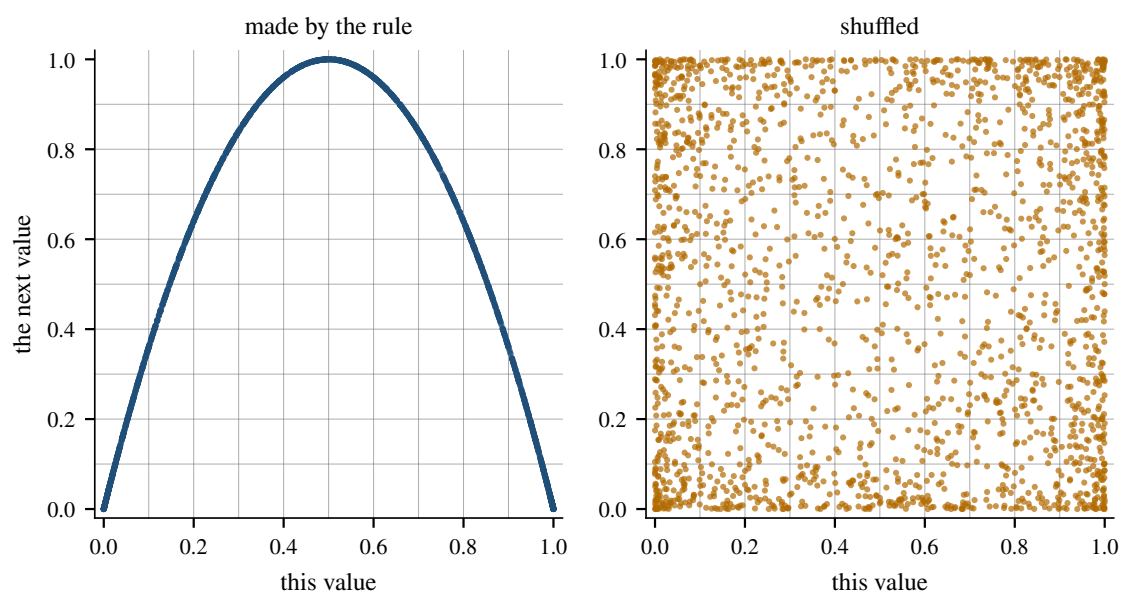


Figure 15.2: Each value plotted against the next. The rule's series draws its own rule; the shuffled series fills the square. The faint lines cut the square into a hundred little squares.

A picture is a judgement; a count is a measurement. Cut the square into a hundred little squares and count how many the return map lands in:

code/ch15/return_map.py

```

14 def return_pairs(xs: list[float]) -> list[tuple[float, float]]:
15     """Each value paired with the one after it."""
16     return list(zip(xs[:-1], xs[1:], strict=True))
17
18
19 def squares_touched(xs: list[float], k: int = 10) -> int:
20     """Cut the unit square into k by k little squares; count how many

```

```

21     of them the return map lands in."""
22
23     def box(v: float) -> int:
24         return min(k - 1, max(0, int(v * k)))
25
26     return len({(box(a), box(b)) for a, b in return_pairs(xs)})

```

Listing 15.2: Pairs of neighbours, and the squares they land in

values	rule	shuffled	(of 100 squares)
100	24	64	
500	28	93	
2000	28	100	

A curve can pass through only so many squares. The rule's 2000 values touch 28, and a longer record would not raise the count, because every new point lands on the same curve. The shuffle keeps finding new squares until it has touched all 100.

Lab 15.1 — Curve or cloud?

Write the return map test yourself: pair each value with the next, count the squares the pairs land in, and return "curve" if fewer than half the squares are touched and "cloud" otherwise. Then try a grid of two by two on a series that cycles through three values, and see how a grid that is too coarse misleads.

Starter: `code/labs/ch15/l15_01_return.py`

Test: `code/labs/ch15/test_l15_01_return.py`

Run: `cd code && uv run pytest -k l15_01_return`

Where you meet this

Robert Shaw, a physicist at the University of California at Santa Cruz, timed the drops falling from a leaking tap. At some rates of flow the drops came at irregular intervals, and when he plotted each interval against the next, the points did not fill a cloud: they lay along a thin curved shape. A kitchen tap became one of the best-known demonstrations of chaos read from a single measured quantity.

Shaw measured one thing, the gap between drops, though the water has far more going on than one number describes. That he saw a shape is not luck. In 1980 Norman Packard, James Crutchfield, Doyne Farmer and Shaw himself showed how to rebuild the shape of a system's motion from one measured quantity, by using delayed copies of it — the reading now, a moment ago, a moment before that — as stand-ins for the numbers nobody measured. In 1981 Floris Takens proved that this works. The return map is the simplest case: one quantity, one delay.

What we are not proving

Takens' theorem in words: for a smooth deterministic system whose state has d numbers, $2d + 1$ delayed copies of almost any single measurement give a faithful copy of the motion, bent and stretched but with the same shape. The proof, in his 1981 paper *Detecting strange attractors in turbulence*, uses differential topology.

15.3 Forecasting from the past

The second test asks what a user of the data cares about: can I predict it? With no rule, you can search the record for the moment most like now and assume that what happened next then will happen next now. This is the *method of analogues*. In 1969 Edward Lorenz tried it on weather maps and found no past day close enough to be useful: with so many numbers in its state, the atmosphere had not come back near where it had been. For a series of single numbers it works:

code/ch15/analogues.py

```

15 def forecast(library: list[float], x: float, h: int) -> float:
16     """Predict h steps after x: find the value in the library most like
17     x, and return what came h steps after it."""
18     best = min(range(len(library) - h),
19               key=lambda j: abs(library[j] - x))
20     return library[best + h]
21
22
23 def mean_error(xs: list[float], h: int, split: int = SPLIT) -> float:
24     """The average size of the forecast's miss, h steps ahead: learn from
25     xs[:split], test on every value after it."""
26     library, test = xs[:split], xs[split:]
27     misses = [abs(forecast(library, test[t], h) - test[t + h])
28              for t in range(len(test) - h)]
29     return sum(misses) / len(misses)

```

Listing 15.3: Forecasting by the nearest past analogue

forecast finds the past value closest to x , its *nearest neighbour*, and returns what came h steps after it. mean_error learns from the first thousand values, forecasts every value after them, and averages the sizes of the misses.

Stop and think 15.4

For the rule's series the average miss one step ahead is about 0.00099, a thousandth. A blind guess — any value picked from the histogram — misses by about 0.42. Chapter 7 measured that at $r = 4$ a small error roughly doubles at every step. About how many steps ahead can the forecast go before its misses reach half of a blind guess's?

Answer it before you read on.

Answer. About nine: a thousandth doubled nine times is about a half, since $2^9 = 512$. The measurement agrees: the misses first pass half of the shuffle's at step 9.

steps ahead	rule	shuffled
1	0.00099	0.42
2	0.0019	0.42
3	0.004	0.41
4	0.0079	0.42
5	0.016	0.38
6	0.031	0.4
7	0.061	0.39
8	0.12	0.4
9	0.2	0.39
10	0.31	0.41
11	0.37	0.41
12	0.38	0.41

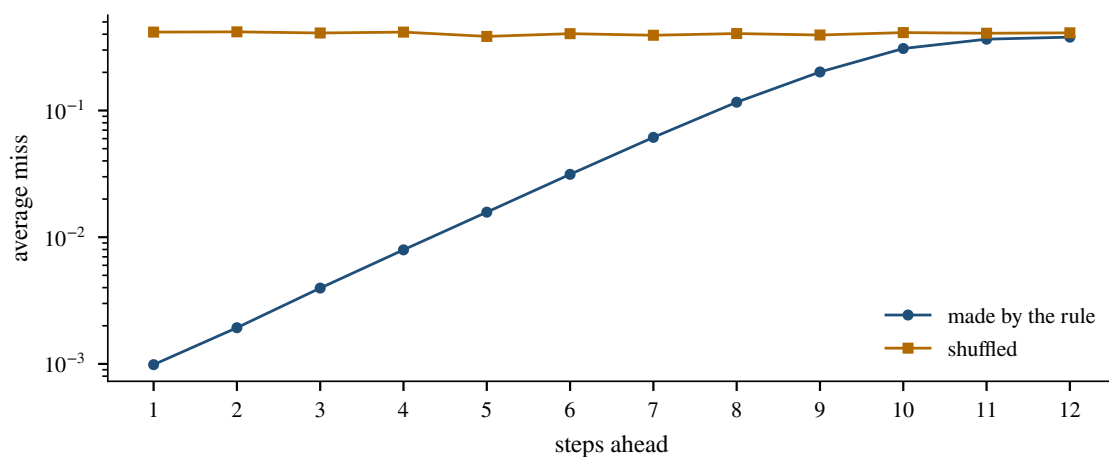


Figure 15.3: The average miss of a forecast by analogues against the number of steps ahead, on a logarithmic scale. The shuffle is a blind guess at every horizon; the rule's series is forecast well, and less well at every step.

For the shuffled series the forecast misses by about 0.42 at every horizon: its past holds no information about its next value. For the rule's series the miss one step ahead is about 420 times smaller, and it grows by a factor of about 2.0 a step — the stretching Chapter 7 measured from the inside, seen from the outside — until after a dozen steps it too is a blind guess.

So chaos and noise differ in both directions. Noise is equally unpredictable at every horizon. Chaos is predictable a few steps ahead, where noise is not, and loses that at a steady, measurable rate: a stronger fingerprint of chaos than irregularity will ever be.

15.4 Real data is a mix

Nothing is measured exactly. A scale, a thermometer or a clerk copying a register moves each value a little, at random. To see what that does, move each value of the rule's series by a random amount no bigger than some size:

code/ch15/noisy.py

```

21 def with_noise(xs: list[float], size: float,
22               seed: int = NOISE_SEED) -> list[float]:
23     """Move each value by a random amount between -size and +size."""
24     rng = random.Random(seed)
25     return [x + rng.uniform(-size, size) for x in xs]
```

Listing 15.4: Measurement noise: each value moved at random

Stop and think 15.5

Add noise of up to 0.01 to the rule's series: a hundredth, thinner than the curve in the return map as printed. What happens to the average miss one step ahead, which was 0.00099?

Answer it before you read on.

Answer. It rises to 0.018, many times worse. The analogue the forecast finds is itself off by up to a hundredth, and so is the value being forecast, before the rule's doubling has even started.

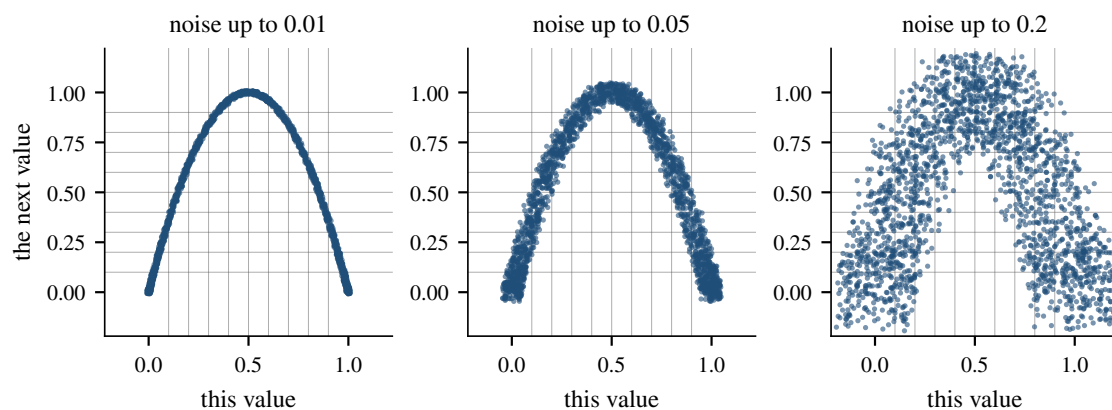


Figure 15.4: Return maps of the rule's series seen through measurement noise of three sizes. The curve thickens into a band, and at the largest size the band nearly fills the square.

noise	squares	1-step error
0.00	28	0.00099
0.01	28	0.018
0.05	44	0.076

0.20	84	0.26
right answers out of 100 short records:		
length	forecast test	squares test
	clean noisy	clean noisy
10	79 74	56 38
20	98 89	95 73
40	100 95	100 92
100	100 99	100 99

In the upper table, the whole record, the curve thickens into a band as the noise grows: with noise of up to 0.2, a fifth of the whole range, it touches 84 squares. Even so, with 2000 values the forecast still beats a blind guess, 0.26 against 0.42.

The lower table is the cruel part. It cuts the record into short pieces and counts, out of a hundred, how often each test says the rule's piece is less random than its own shuffle. With forty noisy values the forecast test is right 95 times. With ten it is right 74 times, and the squares test only 38 — worse than tossing a coin. Short, noisy data can make order look like chance, and chance look like order. And this is the easiest chaos there is, with one number per step; a system with many numbers in its state needs a far longer record.

Lab 15.2 — Ask several analogues

With noise in the record, the closest analogue may be close by accident. Write a forecaster that finds the few closest past values instead and averages what came after them. The test checks it on a small library you can work by hand, and then that averaging five analogues beats trusting one on a noisy series.

Starter: `code/labs/ch15/l15_02_forecast.py`

Test: `code/labs/ch15/test_l15_02_forecast.py`

Run: `cd code && uv run pytest -k l15_02_forecast`

15.5 Populations, epidemics, hearts

Robert May's 1976 paper on the logistic map was written for ecologists. Its lesson was that a population jumping about irregularly from year to year need not be pushed by weather or luck: its own breeding rule could do it.

Stop and think 15.6

Many real insect populations do rise and fall irregularly from one year to the next. Does that make them chaotic?

Answer it before you read on.

Answer. No. Irregular numbers are what chaos produces, and also what a steady population buffeted by weather, disease and chance produces. Irregularity cannot tell the two apart; only tests of order can.

The trap

‘Irregular data proves chaos.’ Every series in this chapter was irregular, and half of them were shuffles. Chaos is the claim that a rule with few numbers in it makes the irregularity, and it needs evidence of that rule: structure in the return map, a forecast that starts good and worsens steadily, and a record long and clean enough for either to mean something.

The strongest evidence has come from the laboratory, where the rule can be changed on purpose. In 1997 Robert Costantino and colleagues reported in *Science* on flour beetles in jars, whose rates of adult death and recruitment the experimenters set by hand. As those rates changed, the colonies moved between steady numbers, cycles and irregular swings where their model predicted. A wild population counted yearly for a few decades gives about as few values as the lower table above.

Epidemics are the other classic. The *SIR model* divides people into the susceptible, who can catch the disease, the infectious, who have it, and the recovered. New infections each day are proportional to how many susceptible and infectious people there are to meet, a case stays infectious for 13 days on average, and births bring in new susceptibles. Take a measles-like disease, where one case among people with no immunity infects about 17 others, and a lifetime of 50 years. Then add a school year: children mix more in term than in the holidays, so let the contact rate rise and fall once a year by a fraction called the *season*.

code/ch15/epidemic.py

```

23 def contact(day: int, season: float) -> float:
24     """How much above or below average the contact rate is on this day:
25     1 + season on New Year's Day, 1 - season at midsummer, in straight
26     lines between."""
27     return 1.0 + season * (4.0 * abs(day / 365.0 - 0.5) - 1.0)
28
29
30 def one_year(s: float, i: float,
31             season: float) -> tuple[float, float, float]:
32     """Step through one year; return the new s and i, and the fraction
33     of people infected during the year."""
34     births, recover, dt = 1.0 / LIFE, 1.0 / DAYS_ILL, 1.0 / STEPS
35     beta0 = R0 * (recover + births)
36     cases = 0.0
37     for day in range(365):
38         beta = beta0 * contact(day, season)
39         for _ in range(STEPS):
40             new = beta * s * i
41             s = s + dt * (births - new - births * s)
42             i = i + dt * (new - (recover + births) * i)
43             cases = cases + dt * new
44     return s, i, cases

```

Listing 15.5: An epidemic with a school year: the SIR rule stepped a quarter-day at a time

season	cases per thousand people, eight years running							
0.0	19	19	19	19	19	19	19	19

0.1	24	13	24	13	24	13	24	13
0.3	33	7	32	4	33	7	32	4
0.4	0	38	44	0	0	0	75	9
the same, starting one part in a billion more susceptible:								
0.1	24	13	24	13	24	13	24	13
0.4	0	39	29	0	0	56	25	0

With no school year the model settles: 19 cases per thousand people, every year the same. With a season of 0.1, big and small years alternate, as measles did in England and Wales before vaccination began in 1968. At 0.3 the pattern repeats every four years, and at 0.4 it never repeats, and a nudge of one part in a billion at the start sends the years a different way. That is the road into chaos of Chapter 8, walked by an epidemic.

It is also where honesty is needed. In the chaotic setting the model's epidemics burn out so completely that at the lowest point fewer than one person in 10^{12} is infectious. No city has a millionth of a person: a real disease would die out. Whether measles was ever chaotic is still argued over. In 1990 George Sugihara and Robert May applied a forecast by analogues to monthly measles counts from New York City and found this chapter's signature, a good forecast that worsened steadily with the horizon, while chickenpox in the same city looked like a yearly cycle with noise on top.

Hearts are harder still. In laboratory experiments reported in 1981, clusters of heart cells driven by regular electrical pulses went through period doubling into irregular beating. A healthy heart does not beat like a metronome either, but the once popular claim that its variability is chaos rests on this chapter's tests applied to short noisy records, and remains contested. Variability is not the same thing as chaos.

15.6 Markets, and what a claim of chaos needs

Prices are the most watched wiggly lines in the world, and for a while many hoped that chaos was the pattern beneath them.

Stop and think 15.7

Suppose someone proved that a share price is made by a chaotic rule with only a few numbers in its state. Could you get rich forecasting it?

Answer it before you read on.

Answer. Not far. Chaos is forecastable only a few steps ahead, only as well as the measurements allow, and not at all beyond its horizon. And every trader who found the same rule would trade on it, changing the prices it describes.

The trap

‘Chaos in markets means you can predict them.’ Chaotic prices would promise a short horizon with a hard limit behind it, not a crystal ball. In fact economists who searched found returns hard to forecast and calm and turbulent periods bunched together, but the evidence for a chaotic rule with few numbers in its state is weak and contested. Irregular, jumpy and hard to predict do not add up to chaotic.

Put the chapter together, and a claim of chaos in real data needs:

- **a long record**, long compared with the number of quantities that matter;
- **little noise**, or tests that allow for it;
- **a fair comparison**: the series must beat its surrogates, including surrogates that keep its slow ups and downs, because a series with memory, like daily temperature, is forecastable a day ahead with no chaos at all;
- **the right shape of forecast error**: small a step ahead and growing at a steady rate;
- and, best of all, **a mechanism**, ideally with a knob that can be turned, as the beetles had.

Few claims about hearts, brains or economies clear all five. That is chaos theory doing its job: telling you what a wiggly line does and does not license you to conclude.

Summary

- A **surrogate** made by shuffling has the same numbers, so the same **histogram**. Only the **order** can tell a rule from chance.
- A **return map** plots each value against the next: a rule draws a curve, noise fills the square. **Delay embedding** (Takens, 1981) rebuilds a system’s shape from one measured quantity.
- The **method of analogues** forecasts from the nearest past value. Chaos is forecast well a step ahead and worse at a steady rate; noise is a blind guess at every horizon.
- **Measurement noise** and **short records** weaken both tests; with ten noisy values the squares test does worse than a coin.
- Chaos has been shown convincingly in laboratory populations; for measles, hearts and markets the evidence is mixed, contested or weak. **Irregular is not chaotic**.
- A claim of chaos needs a long clean record, a fair comparison with surrogates, the right shape of forecast error, and ideally a mechanism.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

make a chaotic series and a random one out of exactly the same numbers, and say why no histogram can tell them apart	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
draw a return map, count the squares it lands in, and read the count	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
forecast a series from its own past by the method of analogues, and measure how the error grows with the number of steps ahead	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say how measurement noise and a short record weaken both tests	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
judge a claim of chaos in populations, epidemics, hearts or markets by the evidence offered for it	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. What does shuffling a series keep, and what does it destroy?
2. The return map of a series lies on one thin curve. What does that tell you about how the series was made?
3. One series touches 30 of the hundred squares with 500 values and still 30 with 5000. Another touches 80 and then 100. Which behaves like a rule's series?
4. A forecast by analogues misses by about 0.001 one step ahead, and its misses double at every step. About how many steps ahead does the miss reach 0.25?
5. Why is a record of ten noisy values poor evidence for chaos or against it?
6. What happens to the model epidemic as the school year is made stronger, and what makes its chaotic setting doubtful as a picture of real measles?
7. Someone shows you that a share price is irregular. Is that evidence of chaos? What would be?

Chaos inside the computer

Every picture of chaos in this book was drawn by a computer, and so was every number. Here is an awkward fact about the machine that drew them. It cannot store the number 0.1. It stores a number very close to it, and every time it adds or multiplies two numbers it rounds the answer to one it can store.

You have spent several chapters learning that chaos turns the smallest error into a large one. So if the computer makes a tiny error at every step, how can any of those pictures be right? This chapter measures the answer, which is better than you might fear and worse than you might hope.

What you will be able to do

When you have finished this chapter you will be able to:

- say what a computer stores when you type a number, and why $0.1 + 0.2$ does not come out as 0.3
- predict the step at which the doubling map dies on a computer
- estimate how many steps a chaotic computation stays right from the number of binary digits it keeps
- say which results of a chaotic simulation you can trust, and why
- describe a random-number generator as a deterministic rule, and say why it must repeat

16.1 What a computer keeps

Write one third as a decimal and you never finish: $0.333 \dots$, the threes running on for ever. Decimals can write exactly only the fractions whose bottom is built from twos and fives, the factors of ten. A computer counts in twos instead. In *binary* the places after the point are worth a half, a quarter, an eighth and so on, so $\frac{3}{4}$ is written 0.11: a half plus a quarter.

Stop and think 16.1

Which of these can binary write exactly, with a fixed number of places: $\frac{1}{2}$, $\frac{3}{8}$, $\frac{1}{10}$?

Answer it before you read on.

Answer. $\frac{1}{2}$ and $\frac{3}{8}$, but not $\frac{1}{10}$. Binary writes exactly the fractions whose bottom is a power of two, and ten has a factor of five.

In binary one tenth is 0.000110011..., with the block 0011 repeating for ever, so a computer has to cut it off. Python, like almost every program you use, keeps a number as a *double*: 53 significant binary digits, about 16 decimal ones, plus a power of two that says where the point goes. For one tenth it keeps the nearest double, which is too large by about 5.55×10^{-18} . Every sum and product is then *rounded* too: the exact answer is replaced by the nearest double.

Stop and think 16.2

Both 0.1 and 0.2 are stored slightly wrong, and their sum is rounded once more. Does the computer agree that $0.1 + 0.2$ is 0.3?

Answer it before you read on.

Answer. No. The sum prints as a number a whisker above 0.3, and the comparison answers False.

code/ch16/floats.py

```

14 print(0.1 + 0.2)
15 print(0.1 + 0.2 == 0.3)
16 print(Decimal(0.1))           # the exact value stored for 0.1
17 print((0.1).as_integer_ratio()) # the same number as a fraction
18 print(2**55)                  # the bottom of that fraction
19 print((0.1 + 0.2) + 0.3, 0.1 + (0.2 + 0.3))
20 big = float(2**53)
21 print(big + 1.0 == big)        # 2**53 + 1 has no room to be stored

```

Listing 16.1: What the computer stores, and what it does with it

```

0.30000000000000004
False
0.1000000000000000055511151231257827021181583404541015625
(3602879701896397, 36028797018963968)
36028797018963968
0.6000000000000001 0.6
True

```

The third line is the exact value kept for 0.1; the fourth writes it as a fraction, whose bottom is 2^{55} . Every double is a whole number divided by a power of two. The sixth line adds three numbers in two orders and gets two answers: once every step rounds, the grouping that school said cannot matter does matter. The last shows the room running out: past 2^{53} a double cannot hold every whole number.

Note

These rounding rules were fixed in 1985 in a standard called IEEE 754, which almost every computer follows. It requires $+$, $-$, $\times$, $\div$ and the square root to give the exact answer rounded to the nearest double. That is why your computer should agree

with every number this book prints: the book's own measurements use only those operations.

16.2 A map the computer cannot run

The simplest chaotic rule there is fits in a line: double the number, and if the answer is 1 or more, take 1 away. In symbols, $x \rightarrow 2x \bmod 1$, read '2x with its whole part thrown away'. This *doubling map* doubles the gap between any two nearby starts at every step. In binary you can watch it work: doubling moves every digit one place left, and taking away 1 drops the digit that crossed the point, so 0.1011 becomes 0.011, then 0.11, then 0.1. The map reads out its start one binary digit at a time.

Stop and think 16.3

Work the doubling map by hand from $\frac{1}{10}$, in fractions, for five steps. What does its orbit do for ever after?

Answer it before you read on.

Answer. $\frac{1}{5}, \frac{2}{5}, \frac{4}{5}, \frac{3}{5}, \frac{1}{5}$: it goes round the same four values for ever, and never reaches 0.

The computer's 0.1, though, is a whole number divided by 2^{55} : it has a last binary digit, and every step pushes one digit out past the point.

Stop and think 16.4

What will the computer's orbit of 0.1 be after sixty steps?

Answer it before you read on.

Answer. Exactly 0. When the last binary digit leaves, at step 55, nothing is left, and 0 doubled is 0 for ever.

code/ch16/doubling.py

```

22 x = 0.1                      # what the computer stores for one tenth
23 t = Fraction(1, 10)         # one tenth itself
24 for n in range(61):
25     if n <= 5 or n in (40, 45, 50, 53, 54, 55, 60):
26         print(f"step {n:2d}:  computer {x:.6f}  truth {float(t):.6f}")
27         x, t = doubling(x), exact_doubling(t)

```

Listing 16.2: The doubling map from one tenth, on the computer and in exact fractions

```

step 0:  computer 0.100000  truth 0.100000
step 1:  computer 0.200000  truth 0.200000

```

```

step 2:  computer 0.400000  truth 0.400000
step 3:  computer 0.800000  truth 0.800000
step 4:  computer 0.600000  truth 0.600000
step 5:  computer 0.200000  truth 0.200000
step 40: computer 0.600006  truth 0.600000
step 45: computer 0.200195  truth 0.200000
step 50: computer 0.406250  truth 0.400000
step 53: computer 0.250000  truth 0.200000
step 54: computer 0.500000  truth 0.400000
step 55: computer 0.000000  truth 0.800000
step 60: computer 0.000000  truth 0.600000

```

For forty steps the two agree to six decimal places; then the computer drifts, and dies. A computed orbit can be wrong in its whole character: a loop has become a fall to nothing.

The strange part is that the computer's arithmetic made no error at all. Doubling a double and taking away 1 never needs rounding, and the chapter's measurement script checks, step by step, that the computer's orbit is exactly the true orbit of the number it stored. What is wrong is the start. A storable number has finitely many binary digits, so every storable start is swallowed by 0, while almost every true start has infinitely many digits and never is. From each of the 996 starts $\frac{1}{997}$, $\frac{2}{997}$, and so on, the computer's orbit died in between 45 and 62 steps, most often in 53.

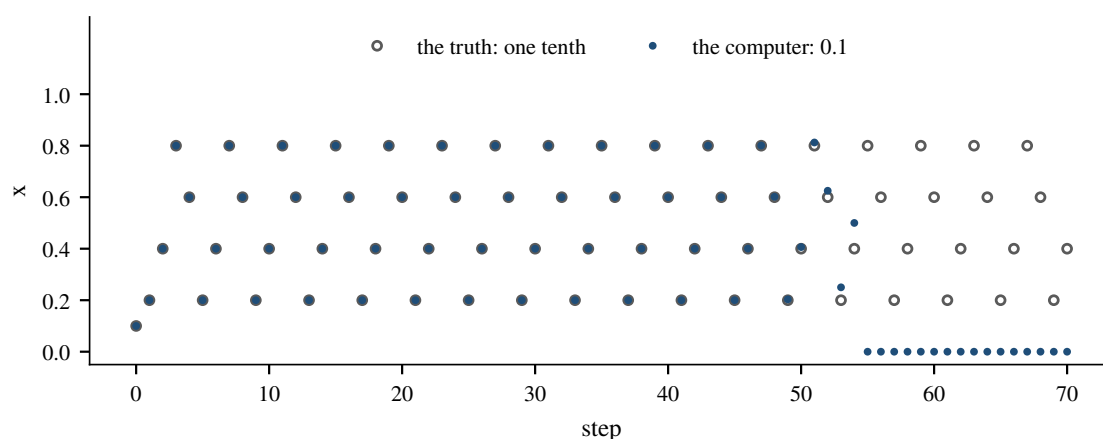


Figure 16.1: The doubling map from one tenth. The truth (open circles) loops for ever; the computer (dots) agrees for about forty steps, drifts, and at step 55 falls to 0 for good.

Lab 16.1 — When does the doubling map die?

Write the doubling map, a function that counts the steps until an orbit is exactly 0, and one that predicts the count without running anything, from `x.as_integer_ratio()`, which gives the stored number as a whole number over a power of two. The test checks that the two always agree.

Starter: `code/labs/ch16/l16_01_collapse.py`

Test: `code/labs/ch16/test_l16_01_collapse.py`

Run: `cd code && uv run pytest -k l16_01_collapse`

16.3 One rule, two programs

The doubling map never rounds. The logistic map of Chapter 5 is more typical: nearly every step rounds. Here it is at $r = 4$, written two ways.

code/ch16/two_formulas.py

```

14 def rule_a(x: float) -> float:
15     return 4.0 * x * (1.0 - x)
16
17
18 def rule_b(x: float) -> float:
19     return 4.0 * x - 4.0 * x * x      # the same, multiplied out

```

Listing 16.3: The logistic map at $r = 4$, written two ways

Multiply out the bracket in the first and you get the second: $4x(1 - x) = 4x - 4x^2$. On paper they are one rule.

Stop and think 16.5

Start both from 0.1 on the same computer and run them for seventy steps. Will the two programs print the same numbers?

Answer it before you read on.

Answer. No. They differ in the last binary digit from step 1, and by step 53 they are more than 0.1 apart.

The first formula rounds $1 - x$ and then a product; the second rounds $4x^2$ and then a difference. Each error is about one part in 10^{16} , and the map doubles it every step. The right-hand column below is the truth: the orbit of exactly one tenth, carried to a hundred significant digits.

step	$4x(1-x)$	$4x-4xx$	truth
0	0.100000	0.100000	0.100000
10	0.147837	0.147837	0.147837
20	0.820014	0.820014	0.820014
30	0.320342	0.320342	0.320342
40	0.097905	0.097917	0.097917
45	0.473588	0.474255	0.474241
50	0.560037	0.538755	0.539206
55	0.879163	0.895376	0.904061
60	0.872399	0.806533	0.368811
70	0.039478	0.972805	0.554893

Which formula is right? Both, for forty steps. By step 53 the first has left the truth, and by step 58 so has the second. Neither formula is wrong; each is right for about fifty steps, and after that it is following a different orbit.

The trap

‘A deterministic program always gives the same answer.’ Run the same program on the same machine and it does, to the last digit. But change the computer, the compiler, a maths library or the order in which a long sum is added, and you change which roundings happen, as the two formulas did. Compilers may rearrange arithmetic for speed, or fuse a multiplication and an addition into one step with one rounding. In most programs the difference stays in the last digit. In a chaotic one it reaches the first.

Fifty is not an accident, and you can predict it.

Stop and think 16.6

At $r = 4$ an error roughly doubles every step, as Chapter 6 measured. A double’s rounding error is about 10^{-16} , and ten doublings multiply an error by about a thousand. Roughly how many steps until an error of 10^{-16} has grown to 0.1?

Answer it before you read on.

Answer. About fifty. From 10^{-16} to 10^{-1} is fifteen factors of ten, and every three factors of ten take about ten doublings. Chapter 7’s horizon, with the rounding error as the error in the start, gives 50.

In one sentence: at $r = 4$ the map uses up one binary digit of accuracy per step, so a double’s 53 digits last about fifty steps. *Single precision*, which graphics cards and much machine-learning software use because it takes half the memory, keeps 24 binary digits, about 7 decimal ones; the same arithmetic predicts 21 steps, and it lasts 23.

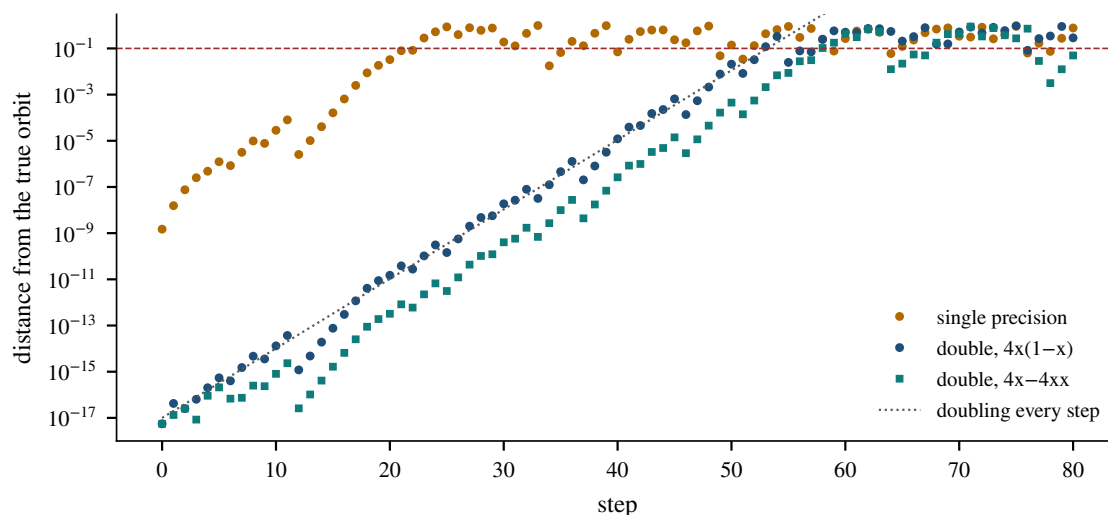


Figure 16.2: Distance from the true orbit of one tenth, on a logarithmic scale, for the logistic map at $r = 4$. Every run’s error doubles each step, along the dotted line, until it fills the whole interval: after about twenty steps in single precision, fifty in double.

Lab 16.2 — Single against double

Round a number to single precision by packing it into four bytes and reading it back (the starter shows how). Write the logistic map at $r = 4$ with every operation rounded that way, run it beside the double version, and find the first step at which they differ by more than 0.1.

Starter: `code/labs/ch16/l16_02_single.py`
 Test: `code/labs/ch16/test_l16_02_single.py`
 Run: `cd code && uv run pytest -k l16_02_single`

Where you meet this

Machine learning meets this daily. A neural network is trained by adding up millions of small numbers on a graphics card, whose parallel threads can finish in a different order each time, so a sum rounds differently from run to run. Two runs of the same program from the same random seed can end with networks whose weights differ throughout — and which usually score about the same. The documentation of PyTorch, a widely used library, says that fully reproducible results are not guaranteed across platforms and releases, and offers a switch that forces a fixed order, at a cost in speed.

16.4 Why more digits cannot save you

The obvious cure is to stop rounding. Python's `fractions` module keeps every number as an exact fraction. One tenth stays one tenth, and $4x(1 - x)$ from it gives $\frac{9}{25}$, then $\frac{576}{625}$, with no error at all.

Stop and think 16.7

From $\frac{9}{25}$ to $\frac{576}{625}$ the bottom went from 25 to $625 = 25 \times 25$. If it is squared at every step, what happens to its number of digits?

Answer it before you read on.

Answer. It roughly doubles at every step: squaring a number about doubles its number of digits.

`code/ch16/more_digits.py`

```

35 x = Fraction(1, 10)
36 for n in range(13):
37     if n % 2 == 0:
38         print(f"step {n:2d}: {len(str(x.denominator)):5d} digits")
39     x = 4 * x * (1 - x)

```

Listing 16.4: Exact fractions: the bottom is squared at every step

```

step 0:      2 digits
step 2:      3 digits
step 4:     12 digits
step 6:     45 digits
step 8:    179 digits
step 10:   716 digits
step 12: 2863 digits
  16 digits kept: right for 54 steps
  32 digits kept: right for 108 steps
  64 digits kept: right for 217 steps
 128 digits kept: right for 430 steps

```

After ten steps the bottom has 716 digits. After fifty it would have about 7.9×10^{14} , and storing it would take some 327 terabytes. Exact arithmetic gets every step right and cannot finish. The last four lines try the practical cure: keep a fixed but larger number of digits, with Python's `decimal` module, and compare with a thousand-digit run. Sixteen digits last 54 steps, like a double; twice the digits last twice as long. Every decimal digit buys about 3.3 steps, because it is worth about 3.3 binary ones.

The trap

'More decimal places make a simulation exact.' They make it right for longer, by a fixed number of steps per digit, never for ever. To follow a chaotic orbit for a thousand steps you must carry about three hundred decimal digits, and know the start that precisely too, which no measurement will give you. It is Chapter 7's cruel arithmetic seen from the computer's side: precision buys a horizon that grows by a constant, not in proportion.

16.5 Wrong path, right picture

So every long chaotic computation in this book followed the wrong orbit. Did it draw the wrong picture? The way to find out rests on one observation.

Stop and think 16.8

Running the logistic map forwards doubles a small error every step, on average. What does running it backwards, from each value to the one before it, do to a small error?

Answer it before you read on.

Answer. Halves it, on average. Whatever a forward step stretches, the backward step squeezes.

So start at the end of a computed orbit and walk back. Two numbers lead to any value in one step, one below $\frac{1}{2}$ and one above; take the one on the same side as the computed value. Walking back shrinks errors, so the walk stays close to the computed orbit all the way to the start, and what it finds, worked to four hundred digits, is a genuine orbit of the rule.

code/ch16/shadow.py

```

20 def shadow(xs: list[float], digits: int = 400) -> list[Decimal]:
21     """A true orbit that stays near the computed orbit xs, found backwards.
22
23     Two numbers lead to y in one step:  $1/2 - s/2$  and  $1/2 + s/2$ , where s is
24     the square root of  $1 - y$ . Take the one on the same side of  $1/2$  as the
25     computed value, and step back again.
26     """
27     with localcontext() as ctx:
28         ctx.prec = digits
29         y = Decimal(xs[-1])
30         ys = [y]
31         for x in reversed(xs[:-1]):
32             s = (1 - y).sqrt()
33             y = HALF - s / 2 if x < 0.5 else HALF + s / 2
34             ys.append(y)
35     return ys[::-1]

```

Listing 16.5: Walking a computed orbit backwards to find the true orbit it follows

```

steps computed with  $4x(1-x)$ :      1000
shadow's start minus  $1/10$ :          $-1.1\text{e-}17$ 
farthest its exact orbit strays:  $8.6\text{e-}15$ 
fraction of a million steps below 0.1:
     $4x(1-x)$  0.205     $4x-4xx$  0.205    exact 0.205

```

The computed orbit of 0.1 left the truth after 53 steps. But a start differing from one tenth by about 1.1×10^{-17} has an exact orbit that stays within 8.6×10^{-15} of every one of the 1000 computed values. The computer did not follow the orbit it was asked for. It followed, faithfully, the orbit of a start so close that no measurement could tell the two apart. That true orbit is its *shadow*.

What we are not proving

The *shadowing lemma*, proved by Dmitri Anosov and Rufus Bowen, covers systems that stretch in some directions and squeeze in others at every point, at rates that never weaken (*uniformly hyperbolic* ones, such as the doubling map). It says that however closely you want a computed orbit to be shadowed, a small enough error per step guarantees it, for ever. The logistic map at $r = 4$ is not quite of that kind: near $x = \frac{1}{2}$ its curve is flat, and there the backward walk stretches. For it, shadowing was confirmed by careful computation over long but finite stretches, notably by Hammel, Yorke and Grebogi in 1987. This book proves neither; the proofs live in graduate texts on dynamical systems.

Now the payoff. A shadow is a true orbit, so it spends its time as true orbits do. The last two lines of the output measure that: over a million steps the first formula spends the fraction 0.205 of its time below 0.1, the second 0.205, and the exact answer, from a formula this book does not derive, is 0.205. Two programs whose paths parted after fifty steps agree about where the orbit lives. That is why the lobe fractions of Chapter 9 and

the climate statistics of Chapter 14 can be trusted when no single computed path can.

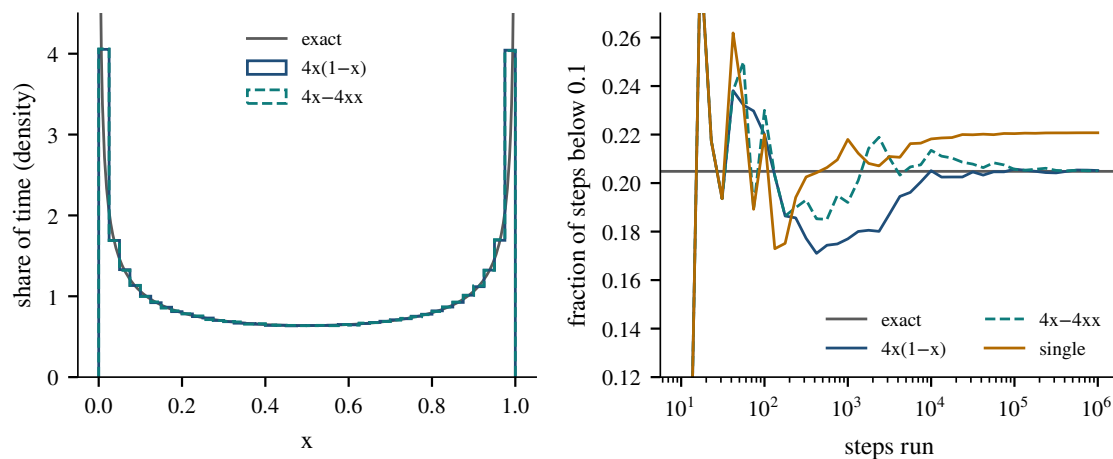


Figure 16.3: Left: where two double-precision runs of a million steps spend their time, against the exact answer. Right: the share of steps below 0.1 as the runs lengthen. The double runs settle on the exact value; the single-precision run, caught in a loop, on a wrong one.



Figure 16.4: A computed orbit stays close to the true orbit of a nearby start, and that orbit's statistics are the ones the picture shows.

16.6 Machines that must repeat

The plainest difference of all: between 0 and 1 there are 1 065 353 217 single-precision numbers and 4 607 182 418 800 017 409 doubles: enormous, and finite. So a computed orbit must sooner or later meet a value it has met before, and since the rule gives the same next value every time, it then goes round the same loop for ever.

In single precision the loops are short enough to find. From 0.1 the logistic map at $r = 4$ wanders for 1744 steps, then goes round a loop of 4344 values for ever; 199 different starts end in only 5 different loops. The loop spoils the statistics too: the single-precision run spends the fraction 0.221 of its time below 0.1, not 0.205, as Figure 16.3 shows. Doubles loop as well, but their loops are far too long to reach, which is the practical reason to compute chaos in double precision.

Stop and think 16.9

The logistic map at $r = 4$ never repeats, covers the whole of 0 to 1 and looks random at a glance. Would it make a good random-number generator?

Answer it before you read on.

Answer. No, for three reasons this chapter has measured. Its values pile up near 0 and 1: about a fifth of its time is below 0.1, not a tenth. Each value is fixed by the one before it. And on a computer it falls into a loop.

The trap

‘A chaotic map makes a good random-number generator.’ Chaos makes the future unpredictable to someone who does not know the start exactly, which is part of what randomness needs and not all of it. A good generator must also spread its values evenly, show no pattern from one value to the next, and repeat only after a loop nobody could ever go round. Those properties are designed with number theory and checked with statistical tests; they do not come free with chaos.

A random-number generator is a deterministic rule on a finite set of values, built to have one enormous loop. The simplest kind, the *linear congruential generator*, is one line: multiply by a , add c , and keep the remainder after dividing by m . With $a = 5$, $c = 3$ and $m = 16$, from 0 you get 3, then $5 \times 3 + 3 = 18$, remainder 2, then 13, 4, 7, 6, 1, 8, 11, 10, 5, 12, 15, 14, 9 and 0 again: all sixteen possible values before a repeat. The doubling map is a generator of this kind with $a = 2$ and $c = 0$, and on k binary digits it dies in k steps — the collapse of Section 16.2 in a new costume. Python’s own generator, the Mersenne Twister of Makoto Matsumoto and Takuji Nishimura (1998), has a loop of $2^{19937} - 1$ values. Because it is deterministic, the same starting value, the *seed*, gives the same ‘random’ numbers every time, which is how a random experiment is made repeatable.

Lab 16.3 — Every orbit ends in a loop

Write a linear congruential generator and a function that finds, for any rule and start, how many steps the orbit takes to enter its loop and how long the loop is. Try it on the generator above, then find the loop of the single-precision logistic map for yourself.

Starter: `code/labs/ch16/l16_03_loops.py`

Test: `code/labs/ch16/test_l16_03_loops.py`

Run: `cd code && uv run pytest -k l16_03_loops`

The practical lesson fits in a sentence: never trust a single long run of a chaotic computation. Run it again with the formula rearranged, with more precision and less, with a smaller step, and believe what survives. The path past fifty steps will not survive. The statistics will.

Summary

- A **double** keeps 53 significant binary digits. Most decimals, 0.1 among them, are stored inexactly, and every sum and product is **rounded**.
- The **doubling map** reads out its start one binary digit at a time, so on a computer every orbit dies at 0; the true orbit of one tenth loops for ever.

- Two programs for one rule round differently and part company. At $r = 4$ the logistic map uses **one binary digit per step**: about fifty steps in double precision, twenty in single.
- **Exact arithmetic** doubles its digits every step and cannot finish; **more digits** buy a fixed number of steps each.
- A computed orbit is **shadowed** by a true orbit from a nearby start, so its **statistics** are right when its path is wrong.
- A finite machine makes every orbit end in a **loop**; a random-number generator is a rule built to make that loop enormous.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

say what a computer stores when you type a number, and why $0.1 + 0.2$ does not come out as 0.3	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
predict the step at which the doubling map dies on a computer	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
estimate how many steps a chaotic computation stays right from the number of binary digits it keeps	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say which results of a chaotic simulation you can trust, and why	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
describe a random-number generator as a deterministic rule, and say why it must repeat	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Why does $0.1 + 0.2$ not equal 0.3 on a computer?
2. The computer stores 0.3 as a whole number over 2^{54} . When does its doubling-map orbit reach 0, and what does the true orbit of $\frac{3}{10}$ do?
3. Single precision keeps 24 binary digits. Roughly how many steps of the logistic map at $r = 4$ is it good for?
4. Run exactly from $\frac{1}{10}$, the bottom of the fraction has 716 digits after ten steps. About how many after twenty?
5. After a million steps of a chaotic simulation, which can you trust: the last value, or the fraction of time spent below 0.1?
6. Why must every orbit computed on a machine eventually repeat?

Part V

Living with chaos

Taming chaos: control and synchronisation

Balance a broom upright on the palm of your hand. Left alone it falls: upright is a position it cannot keep. Yet you can keep it there for as long as your patience lasts, with movements of the hand so small that somebody watching might not see them. You do not overpower the broom. You watch which way it is starting to go and make a small correction, again and again.

Part II of this book showed that a chaotic system turns a tiny difference into a large one, and measured how fast. This chapter turns that sentence round. If a tiny disturbance can change the future of a chaotic system a great deal, can a tiny, well-chosen disturbance change it the way you want? And if two chaotic systems left to themselves always drift apart, can they be made to agree?

What you will be able to do

When you have finished this chapter you will be able to:

- find a fixed point of the logistic map that the rule cannot keep, and count how often a chaotic orbit comes near it
- work out the small change of a parameter that cancels the miss one step would make
- run a small-nudge controller on a chaotic map, and measure how long it waits and how large its nudges are
- drive a copy of the Lorenz system with one number from the original and measure the gap between the two as time goes on
- say where control and synchronisation of chaos have been shown to work, and what they are not

17.1 A point the orbit cannot keep

Take the logistic map of Chapter 5, the rule that turns x into $rx(1-x)$, at $r = 3.9$, where the orbit never settles and never repeats. The rule still has a fixed point: the value $1 - 1/r$, which the rule leaves exactly where it is. At $r = 3.9$ that is about 0.744. Chapter 5 also said why an orbit does not stay there. Near a fixed point, one step multiplies a small miss by the slope of the rule at that point — the stretch factor of Chapter 3 — and here the slope is -1.9 . A miss of a thousandth becomes nearly two thousandths on the other side, then nearly four, then nearly seven. The fixed point is a broom standing on its end.

code/ch17/hidden.py

```

13 R = 3.9
14 rule = logistic(R)
15 POINT = 1.0 - 1.0 / R           # where the rule leaves x unchanged
16 STRETCH = logistic_slope(R)(POINT) # one step multiplies a miss by this

```

Listing 17.1: A fixed point, and what one step does to a miss there

The line that starts `STRETCH` asks the book's library for the slope of the rule at the fixed point: how much one step multiplies a small miss there.

Stop and think 17.1

The orbit at $r = 3.9$ is chaotic, and the fixed point pushes away anything that comes near it. Once the orbit has left, does it ever come back within a hundredth of 0.744: never, a handful of times, or again and again?

Answer it before you read on.

Answer. Again and again. In 10 000 steps from $x = 0.2$ it is within a hundredth of the fixed point on 112 steps, the first time at step 25.

A chaotic orbit wanders over its whole range and skips no part of it: Chapter 9 found that how often a chaotic system visits each region is one of the things you can predict about it. The listing's function `visits` counts the steps that land close:

```

fixed point      0.7436
one step later   0.7436
slope there      -1.90
steps within 0.01 of it, out of 10000: 112
the first ten: [25, 175, 176, 177, 285, 546, 547, 548, 549, 703]

```

Look at how the visits come: in runs of neighbouring steps. When the orbit arrives close, the slope of -1.9 flings it off only gradually, nearly doubling its miss and swapping its side at every step, so it lingers a few steps before it is thrown clear.

The fixed point is only the simplest thing the rule cannot keep. There are cycles of two values, of three, and of many more lengths, every one of them unstable, and the chaotic orbit passes near each of them in turn. A chaotic attractor is not a formless blur. It is a tangle of unstable cycles, and the orbit hops for ever from the neighbourhood of one to the next.

17.2 A nudge at the right moment

In 1990 Edward Ott, Celso Grebogi and James Yorke published a paper called 'Controlling chaos', with an idea as plain as the broom. Do not fight the chaos. Wait until the orbit passes close to the unstable orbit you want, then give the system small pushes, each only as big as it takes to cancel the miss that the next step would make. What they pushed was

a *parameter*: a setting of the system that you are allowed to adjust a little. For the logistic map that setting is r .

Stop and think 17.2

Suppose you may change r at every step, but never by more than one per cent: to anywhere between 3.861 and 3.939. Left alone at 3.9 the map is chaotic, and one per cent is hardly a change. Can nudges that small hold the orbit at 0.744?

Answer it before you read on.

Answer. Yes. In the run of the next section, a controller whose nudges never exceed one per cent is switched on at step 100, catches the orbit at step 175, and holds it at the fixed point for as long as the run lasts. The largest nudge it ever makes is 0.70 per cent of r .

The trap

‘Chaos cannot be controlled.’ What chaos forbids is prediction far ahead: Chapter 7 measured how a small error grows until a forecast is worthless. A controller never looks far ahead. It looks one step ahead, and over one step a chaotic rule is as predictable as a cup of tea: a miss is multiplied by -1.9 , not by a million. Chaos takes away the long view and leaves the short one, and the short one is all that steering needs. The chaos even does half the work, because it is what brings the orbit near the target in the first place.

How big must each push be? Two facts decide it. The first you have already: one step multiplies a miss by -1.9 . The second is what the knob does. The rule is r times $x(1-x)$, so raising r by a small amount c for one step lifts the next value by c times $x(1-x)$. Near the fixed point $x(1-x)$ is about 0.19; call it the *lever*.

Stop and think 17.3

The orbit is 0.001 above the fixed point. Left alone, the next step puts it $1.9 \times 0.001 = 0.0019$ below. Raising r by c lifts the next value by about 0.19 times c . What c cancels the miss?

Answer it before you read on.

Answer. $c = 0.0019/0.19$, about 0.01: raise r from 3.9 to 3.91 for this one step, about a quarter of one per cent.

In general the change is the miss times the slope, divided by the lever, with its sign turned round: about 9.97 times the miss. That is the whole controller.

code/ch17/control.py

```

14 R0 = 3.9
15 TARGET = 1.0 - 1.0 / R0          # the fixed point, unstable at R0
16 STRETCH = logistic_slope(R0)(TARGET) # one step multiplies a miss by this
17 LEVER = TARGET * (1.0 - TARGET)    # next x moves this much per unit of r
18
19
20 def nudge(x: float, largest: float) -> float:
21     """The change to r, for this step only, that cancels the coming miss.
22
23     Zero if the change needed is bigger than `largest`: then wait.
24     """
25     change = -STRETCH * (x - TARGET) / LEVER
26     return change if abs(change) <= largest else 0.0

```

Listing 17.2: The controller: cancel the coming miss, or wait

`nudge` is the rule, and its last line is the patience. If the change it would need is bigger than `largest`, it returns zero and the map runs on untouched. With `largest` set to one per cent of r , the controller can act only when the orbit is within 0.0039 of the fixed point, and it has to wait for the chaos to bring it there.

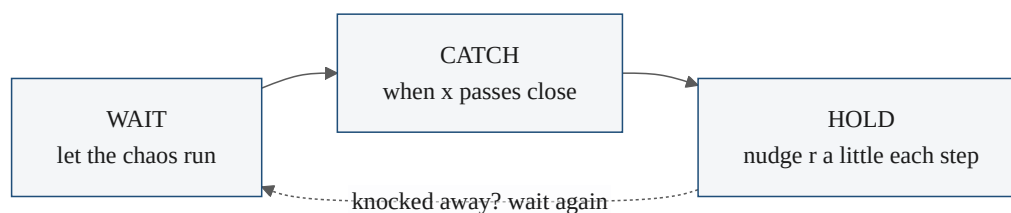


Figure 17.1: The controller's three moods. It waits while the orbit is out of reach, catches it when the chaos brings it close, and holds it with nudges; knocked away, it waits again.

17.3 Small change, not force

Now run the controller. The listing below leaves the map alone for 100 steps from $x = 0.2$, then switches the controller on with nudges of at most one per cent, and prints the state now and then.

Stop and think 17.4

Once the orbit is caught, how big must the nudges be to keep it there: as big as the first one, bigger because the chaos keeps pushing back, or smaller?

Answer it before you read on.

Answer. Smaller, and then almost nothing. Within 3 steps of the catch every nudge is below a billionth of r , and all the nudges of the run, added together, come to 0.028.

The trap

‘Controlling a system means overpowering it.’ The controller supplies none of the motion. The fixed point, and the orbit’s visits to it, belong to the map; the controller only cancels the small misses it can see coming, and a caught orbit has almost no miss left to cancel. You are choosing which of the system’s own behaviours to keep, not forcing a new one on it. That is why Ott, Grebogi and Yorke could hold a chaotic system on any of its many unstable cycles, and switch it between them, for the same small nudges.

code/ch17/control.py

```

37     largest = 0.01 * R0          # never more than one per cent of r
38     x, biggest = 0.2, 0.0
39     print("step x               change to r")
40     for n in range(301):
41         change = nudge(x, largest) if n >= 100 else 0.0
42         if n % 50 == 0 or abs(change) > 1e-9:
43             print(f"{n:4d} {x:.10f} {change:+.2e}")
44             biggest = max(biggest, abs(change))
45             x = step(x, change)
46     print(f"largest change: {100 * biggest / R0:.2f} per cent of r")

```

Listing 17.3: Chaos for a hundred steps, then the controller

```

step  x               change to r
  0  0.2000000000  +0.00e+00
 50  0.3686282977  +0.00e+00
100  0.9258735893  +0.00e+00
150  0.8891185779  +0.00e+00
175  0.7408600980  -2.72e-02
176  0.7435247144  -6.48e-04
177  0.7435897066  -3.69e-07
200  0.7435897436  +0.00e+00
250  0.7435897436  +0.00e+00
300  0.7435897436  +0.00e+00
largest change: 0.70 per cent of r

```

Python writes -2.72×10^{-2} as $-2.72\text{e-}02$, which is -0.0272 . Read the printout downwards. For 100 steps the orbit wanders. The controller is switched on and for 75 steps does nothing at all, because the orbit is out of reach. At step 175 it arrives within reach, is nudged, and lands much closer; every nudge after that is far smaller than the one before, and from step 200 on, x prints the same ten digits.

The tempting alternative is force: turn the knob until the chaos stops. Chapter 5 found that the fixed point is stable only while r is below 3. Set $r = 2.9$ and leave it there, and the orbit does settle, at 0.655. That is a change of 26 per cent of r , made for good, and it does not even give you the state you wanted: the calm map’s fixed point, $1 - 1/2.9$, is somewhere else. The value 0.744 is a fixed point only when $r = 3.9$, where it is unstable, so no fixed setting of the knob can give you that state. The controller holds it for small change: all its nudges added together come to less than a thirtieth of the change that force

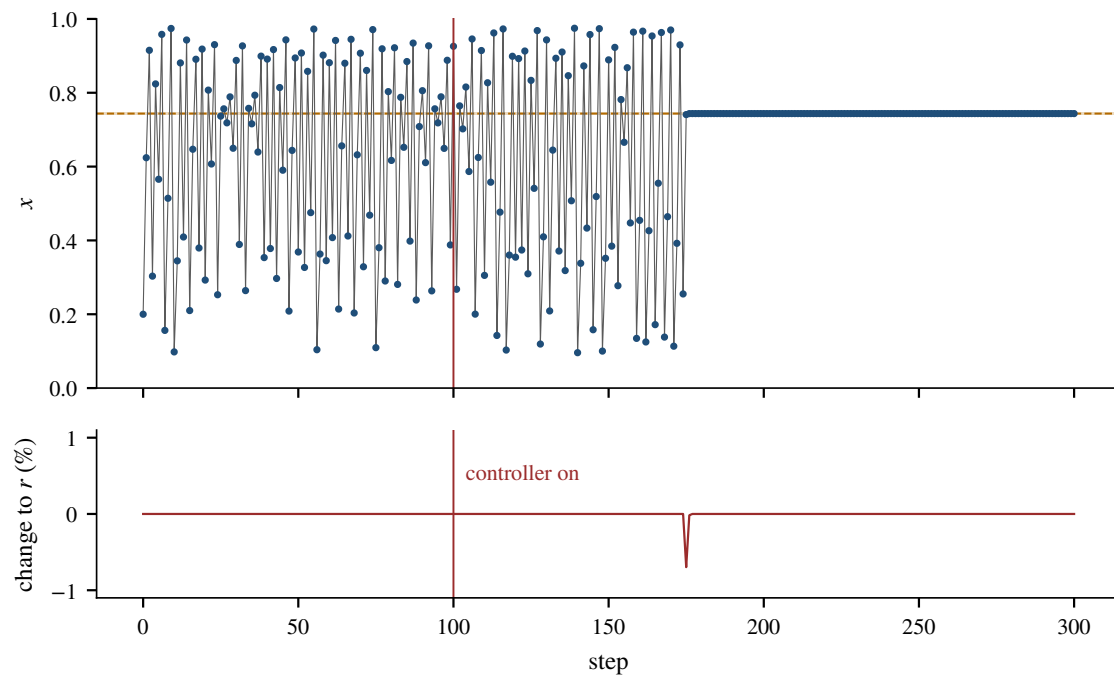


Figure 17.2: The logistic map at $r = 3.9$, left alone for 100 steps and then controlled. The dashed line is the fixed point. Below, the change made to r : zero, apart from one small dip at the catch.

makes at every single step.

Small nudges have a price, and it is paid in time. The controller can act only when the orbit is within reach, and the reach grows in step with the largest nudge you allow.

Stop and think 17.5

You allow nudges ten times smaller: a tenth of one per cent of r . What happens to the wait before the catch?

Answer it before you read on.

Answer. It becomes about ten times longer. Averaged over a thousand starting points the wait is 31 steps with nudges of up to ten per cent, 360 at one per cent and 3828 at a tenth of one per cent.

code/ch17/waiting.py

```

15 def wait(x: float, largest: float) -> int:
16     """Steps from x until the first nudge the controller is allowed."""
17     n = 0
18     while nudge(x, largest) == 0.0:
19         x = step(x, 0.0)
20         n += 1
21     return n

```

Listing 17.4: How long until the first nudge?

largest nudge	average wait	longest wait
10.0 per cent	31	199
1.0 per cent	360	2202
0.1 per cent	3828	29737

The reach shrinks tenfold, so the orbit has a target ten times narrower to pass through, and on average it takes ten times as long to find it. For any single run you cannot say in advance how long the wait will be — it is a chaotic orbit's first visit — and the longest of the thousand runs at one per cent waited 2202 steps. The average, like the statistics of Chapter 9, is steady.

Lab 17.1 — A controller of your own

Write `nudge` and `hold` for the logistic map at any r at which the fixed point $1 - 1/r$ has become unstable. At the fixed point the slope is $r(1 - 2x)$ and the lever is $x(1 - x)$. The tests catch and hold the orbit at $r = 3.9$ and again at $r = 3.7$, and check that no nudge is ever bigger than you allowed.

Starter: `code/labs/ch17/l17_01_controller.py`

Test: `code/labs/ch17/test_l17_01_controller.py`

Run: `cd code && uv run pytest -k l17_01_controller`

17.4 Two chaotic systems that agree

Chapter 9 ran Lorenz's three equations twice, from starts a hair apart, and watched the two runs part company for good. Two chaotic systems left to themselves never stay together. In 1990 Louis Pecora and Thomas Carroll asked what happens when one of them is not left to itself. Build a copy of Lorenz's system that has its own y and z , obeying the same rules as the original's, but no x of its own: at every moment it is handed the original's x instead.

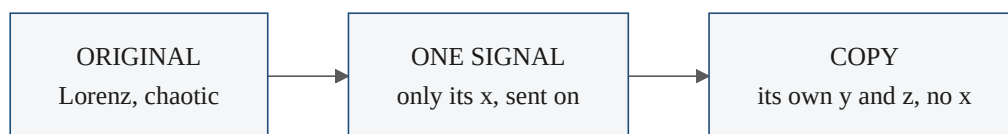


Figure 17.3: Drive and response. The original runs freely; one of its three numbers is passed to a copy that has only the other two.

Stop and think 17.6

Start the copy's y and z at zero, a distance of 36.7 from the original's. Both are chaotic. Will the copy's y and z drift away from the original's, as the two runs of Chapter 9 did, or come together?

Answer it before you read on.

Answer. Together, and fast. The gap is below a millionth at time 9.8, and at time 20.5 the copy's two numbers equal the original's in every digit the computer stores.

code/ch17/sync.py

```

20 def drive_and_copy(s: tuple[float, ...]) -> tuple[float, ...]:
21     """The original (x, y, z) and a copy (y2, z2) handed the original's x."""
22     x, y, z, y2, z2 = s
23     return (SIGMA * (y - x),
24             x * (RHO - z) - y,
25             x * y - BETA * z,
26             x * (RHO - z2) - y2,      # the copy's rules, fed the SAME x
27             x * y2 - BETA * z2)

```

Listing 17.5: The original, and a copy handed its x

time	driven copy	copy left alone
0.0	3.67e+01	3.67e+01
5.0	3.89e-03	1.11e+01
10.0	7.29e-07	2.64e+01
15.0	1.01e-10	1.24e+01
20.0	6.22e-15	2.78e+01
25.0	0.00e+00	2.16e+01
30.0	0.00e+00	2.21e+01

The middle column is the copy handed x . The last is a second copy, from the same wrong start, left to run on its own. The only difference between them is that one number passed across, and it makes the whole difference: the copy left alone never comes closer to the original than 5.5, while the driven copy's gap falls by a factor of about 6 with every unit of time. Once the two agree to the last digit they stay that way, because from then on the computer is doing the same arithmetic twice.

Why does it work? The copy is not in the loop that makes the original chaotic. In Lorenz's system x drives y and z , and y and z drive x back, and a small difference goes round that loop and grows. The copy only listens. Given the same x , its y and z forget where they started, the way two cups of tea in one room forget their difference in Chapter 1.

The trap

'Two chaotic systems can never agree.' Two chaotic systems left to themselves can never stay together, because each amplifies whatever difference there is between them. Coupled so that one listens to the other, they can agree exactly, chaos and all. Notice what the agreement is not: the copy agrees with the original now, not with the original's future. Synchronisation is not prediction.

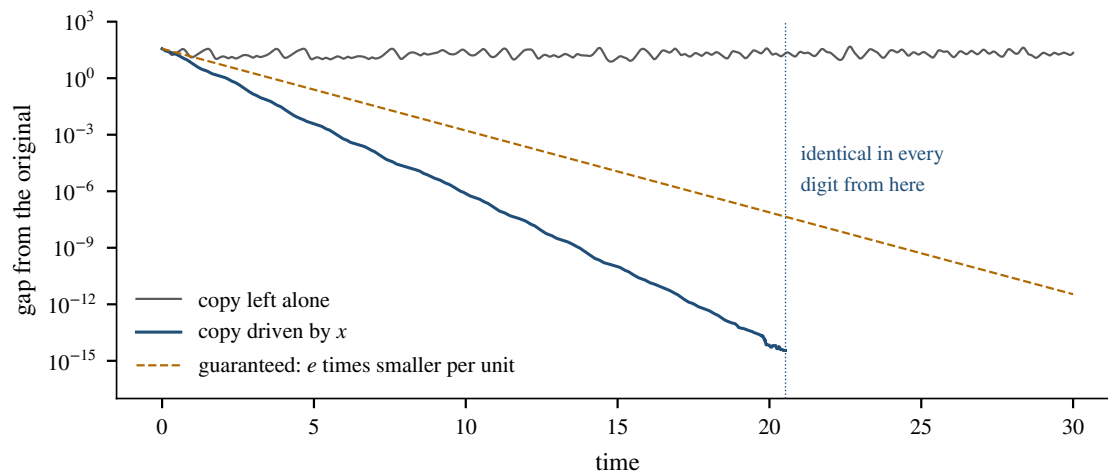


Figure 17.4: The gap between each copy and the original, on a logarithmic scale. Handed x , the copy falls into step along a straight line, faster than the guaranteed rate; left alone, it stays as far away as the attractor allows.

What we are not proving

Why it must work, whatever x does. Write e_y and e_z for how far the copy's y and z are from the original's. Subtract the copy's rules from the original's and work out how fast $e_y^2 + e_z^2$ changes: every term that contains x cancels, and what is left, $-2e_y^2 - 2\beta e_z^2$ with $\beta = 8/3$, is never positive. From that one line it follows that the gap shrinks at least by the factor e , about 2.718, in every unit of time; the measured factor of 6 is better than the guarantee. An argument of this kind is in Strogatz's textbook *Nonlinear Dynamics and Chaos*. Not every choice of the number handed across works, and Pecora and Carroll's paper gives the test that tells.

Lab 17.2 — How long does the copy take?

Write `copy_field`, the five rates of change of the original and the copy; `error`, the distance between the copy's y and z and the original's; and `sync_time`, which steps the five numbers together and returns the first time the error falls below a tolerance. The tests check that three more factors of ten take no longer than the guarantee above allows.

Starter: `code/labs/ch17/117_02_sync.py`

Test: `code/labs/ch17/test_117_02_sync.py`

Run: `cd code && uv run pytest -k 117_02_sync`

17.5 What has been done with it, and what has not

Pecora and Carroll's result led at once to a proposal for keeping secrets. Add a faint message to the original's x and send the sum. To anyone listening it sounds like noise. A receiver built as the matching copy falls into step with the chaotic part of the signal, and subtracting what it rebuilds leaves the message. Circuits that do this were built soon

afterwards. Later work showed that many such schemes can be broken by an eavesdropper who studies the signal, and chaos has not replaced ordinary encryption.

Where you meet this

Within a few years of 1990 the small-nudge method had held chaotic lasers and electronic circuits on orbits they would not keep by themselves. In 1992 Alan Garfinkel and colleagues applied it in an experiment on isolated rabbit heart tissue, whose beating had been made irregular with a drug: electrical stimuli, timed by the same wait-then-nudge logic, made the rhythm regular again. Space missions use the same double edge of sensitivity. Near the places where the pulls of the Sun, the Earth and the Moon nearly balance, the three-body problem of Chapter 13 is chaotic, and a small burn of the engines changes a path a great deal. NASA's ISEE-3 spacecraft was steered away from its station between the Sun and the Earth in this way, with a series of close passes of the Moon and little fuel, and in 1985 it flew through the tail of a comet.

Watch out

Keep these in proportion. Control of chaos has been demonstrated many times in the laboratory. It is not a routine clinical treatment, and it is not a standard industrial technique; whether the heart's own irregularity is chaos at all is a question that Chapter 15 calls contested.

What this chapter adds to the book's argument is this. Part II measured sensitivity as a limit: it is why the horizon of a forecast is short. It is also a lever. A calm system has one behaviour, and changing it takes force. A chaotic system is full of behaviours it cannot keep, and any of them can be chosen, and held, for small change, as long as you are willing to watch and wait. Chaos is not only a limit on what you can know. It is a resource. Chapter 18 gathers what the whole book has measured into a guide for living with both.

Summary

- At $r = 3.9$ the logistic map has a **fixed point it cannot keep**: one step multiplies a miss by the slope, -1.9 . A chaotic orbit keeps passing close to it, and to many **unstable cycles**.
- A **controller** waits for a close pass, then changes the **parameter** r by the amount that cancels the coming miss: the miss times the slope, divided by the **lever**, with the sign turned round. It predicts only one step ahead, and chaos allows that.
- Once the orbit is caught the nudges shrink to almost nothing. Force, a large and permanent change of r , cannot even give the same state. Smaller nudges cost a longer **wait**, whose average is steady.
- A copy of the Lorenz system handed the original's x falls into step with it: **synchronisation**. Left alone it never does. Agreement is not prediction.

- Control and synchronisation of chaos are **laboratory results**, in lasers, circuits and heart tissue, and a tool of spacecraft steering. Sensitivity is a **resource** as well as a limit.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

find a fixed point of the logistic map that the rule cannot keep, and count how often a chaotic orbit comes near it	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
work out the small change of a parameter that cancels the miss one step would make	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
run a small-nudge controller on a chaotic map, and measure how long it waits and how large its nudges are	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
drive a copy of the Lorenz system with one number from the original and measure the gap between the two as time goes on	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say where control and synchronisation of chaos have been shown to work, and what they are not	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

- For the logistic map at $r = 3.5$, what is the fixed point $1 - 1/r$, what is the slope $2 - r$ there, and can an orbit stay at it?
- At $r = 3.9$ the orbit is 0.002 below the fixed point. Where does one uncontrolled step put it, and what change of r cancels the miss?
- Why can a controller steer a chaotic system when nobody can forecast it far ahead?
- You halve the largest nudge you allow. What happens to the reach of the controller, and roughly to the average wait?
- Why does setting $r = 2.9$ for good not give you the state 0.744?
- In Pecora and Carroll's arrangement, what is passed from the original to the copy, and does the copy then predict the original's future?
- Name two places where control of chaos has been demonstrated, and say what it is not.

Why you need chaos: a field guide to prediction

Here are four predictions you might meet in a single week. A table says there will be a total eclipse of the Sun on 2 August 2027. A weather app gives a 70 per cent chance of rain on Saturday. A pension statement says what your savings will be worth in 2060. And a computer program answers a question about next year's interest rates without a flicker of doubt.

Which of them would you trust, and how far? You can now answer that with numbers you computed yourself. This chapter turns the book into a field guide: five questions to ask of any prediction, each answered by a measurement, and then the case for why everybody needs them.

What you will be able to do

When you have finished this chapter you will be able to:

- ask of any prediction whether a small change in its start grows
- compute a prediction horizon, and say what better data buys
- say what can be predicted beyond the horizon, and read odds
- separate chaos from noise, and a small cause from a big effect
- apply the field guide's seven questions to a forecast or a model
- explain, with numbers you computed, why some things can be predicted for centuries and others only for days

18.1 Question one: does a small change grow?

Start with the cleanest pair in the list: the eclipse and the rain. Both are forecast the same way. A computer takes a measured present, applies exact laws of physics, and steps forward in time.

Stop and think 18.1

An eclipse table is good for thousands of years and a detailed weather forecast for about a week. Both come from exact laws stepped forward from a measured start. In the language of this book, what is the difference?

Answer it before you read on.

Answer. How fast a small error in the start grows. In the motions of the Earth and the Moon it grows so slowly that it stays small for millennia; in the air it grows until it is as large as the weather itself.

The test is the one Chapter 6 turned into a measurement: run the rule twice, from two starts closer than any instrument could tell apart, and watch the gap. Here are three rules the book has met, each started twice a billionth apart:

code/ch18/grows.py

```

24 print("gap after step:      0      10      20      30      40")
25 for name, rule, start in RULES:
26     a, b = start, start + 1e-9      # two starts a billionth apart
27     gaps = []
28     for step in range(41):
29         if step % 10 == 0:
30             gaps.append(f"{abs(b - a):8.1e}")
31             a, b = rule(a), rule(b)
32     print(f"{name:18s}" + " ".join(gaps))

```

Listing 18.1: The first question, asked of three rules

gap after step:	0	10	20	30	40
cooling tea	1.0e-09	3.5e-10	1.2e-10	4.2e-11	1.5e-11
logistic, $r = 2.8$	1.0e-09	9.1e-11	1.0e-11	1.1e-12	1.2e-13
logistic, $r = 4$	1.0e-09	4.6e-07	5.4e-04	6.1e-01	2.4e-01

The cooling tea forgets its disturbance: the gap is multiplied by 0.9 every minute. The logistic map at $r = 2.8$ forgets too, because it settles to its fixed point. At $r = 4$ the billionth grows until it is as big as the map allows. If the gap shrinks, the path is predictable for as long as the rule is right. If it grows, the next question is how fast.

Question one has a question in front of it: is there an exact rule at all? For the planets and for Lorenz's three equations there is. For an economy or an epidemic the rule itself changes as people react to the forecast, which is a different difficulty from chaos.

18.2 Question two: how far is the horizon?

A growing error usually grows by roughly the same factor each step, and Chapter 7 turned that factor into one number: the Lyapunov exponent, written λ (the Greek letter lambda), which is the natural logarithm $\ln$ of the average factor. A factor of 2 is an exponent of $\ln 2$, about 0.693. Suppose your start is wrong by e_0 and the forecast is useless once the error passes a tolerance T . After n steps the error is e_0 times the factor n times over, so it reaches T when

$$n \approx \frac{\ln(T/e_0)}{\lambda}.$$

In words: the horizon is the logarithm of how many times too small your error is, divided by the exponent. The book's library has it in one line (`e**exponent` there is the factor, rebuilt from the exponent):

code/src/chaoslab/measure.py

```

65 def horizon(exponent: float, error: float, tolerance: float) -> float:
66     """Steps (or time) until an error of size `error` grows to `tolerance`,
67     if it grows by the factor e**exponent per step."""
68     return math.log(tolerance / error) / exponent

```

Listing 18.2: The horizon formula in the book's library

Try it on the logistic map at $r = 4$, whose exponent this chapter measures as 0.693 per step ($\ln 2$, as Chapter 7 shows). Know the start to six decimal places, so e_0 is a millionth, and call the forecast useless once it is out by more than $T = 0.1$. Then $T/e_0 = 100\,000$, and $n \approx 11.513/0.693 \approx 16.6$ steps.

Stop and think 18.2

You buy an instrument a thousand times more precise, and now know the start to nine decimal places. How long does the forecast last: a thousand times as long, about three times as long, or a few steps longer?

Answer it before you read on.

Answer. A few steps longer: about 10.0 more, so 26.6 in all. Dividing e_0 by a thousand adds $\ln 1000/\lambda = 6.908/0.693$ steps, and nothing more.

That is the formula's answer. Here is the map's own, measured the long way over 201 different starts:

code/ch18/thousandfold.py

```

20 def typical_horizon(error: float, tolerance: float = 0.1) -> float:
21     """The median, over many starts, of the steps two runs `error` apart
22     stay within `tolerance` of each other."""
23     return median(steps_until_apart(RULE, x0, error, tolerance)
24                   for x0 in STARTS)
25
26
27 def main() -> None:
28     six = typical_horizon(1 / 10**6)      # start known to six digits
29     nine = typical_horizon(1 / 10**9)     # a thousand times better
30     print(f"start right to 6 digits: {six:.0f} steps")
31     print(f"start right to 9 digits: {nine:.0f} steps")
32     print(f"a thousandfold better measurement bought {nine - six:.0f} steps")

```

Listing 18.3: A thousand times better data, measured

```

start right to 6 digits: 17 steps
start right to 9 digits: 27 steps
a thousandfold better measurement bought 10 steps

```

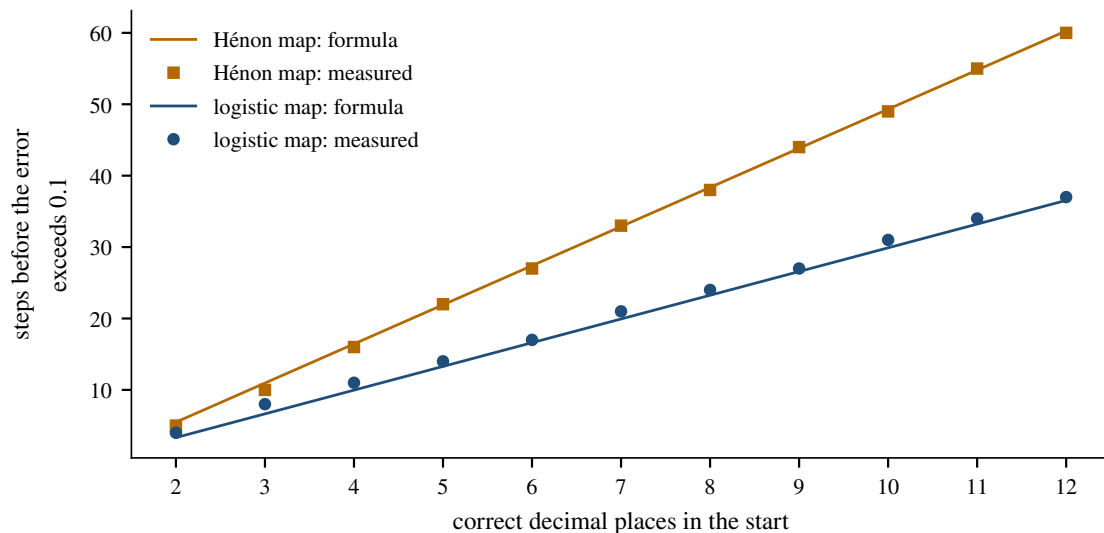


Figure 18.1: How far ahead a forecast lasts, against how many decimal places of the start are right. Lines are the formula, dots the median of many measured pairs of runs. Each extra digit buys the same few steps.

The formula said 16.6 and 26.6; the map says 17 and 27, a gain of 10. Every extra correct digit buys the same number of steps, $\ln 10/\lambda = 2.303/0.693$, about 3.3 here. The horizon grows like the logarithm of your precision, so better instruments and bigger computers can only add a fixed stretch. A small calculator puts three of the book's systems side by side:

code/ch18/horizon.py

```

16 def forecast_length(exponent: float, digits: int,
17                     tolerance: float = 0.1) -> float:
18     """How far ahead a forecast stays within `tolerance`, when the start is
19     right to `digits` decimal places and an error is multiplied by
20     e**exponent every step (or every unit of time)."""
21     error = 1 / 10**digits          # six digits: an error of 0.000001
22     return horizon(exponent, error, tolerance)

```

Listing 18.4: The horizon calculator

	exponent	horizon with the start right to			
		3 digits	6 digits	9 digits	
logistic map	0.693	6.6	16.6	26.6	steps
Henon map	0.420	11.0	27.4	43.8	steps
Lorenz system	0.909	5.1	12.7	20.3	time units

The Hénon map's errors grow more slowly, at 0.420 per step, so each digit buys it 5.5 steps; Lorenz's system, at 0.909 per unit of time, gains 2.5 units of time per digit. Plotted against the number of correct digits, every horizon is a straight line (Figure 18.1).

The same arithmetic explains the eclipse. Jacques Laskar's 1989 computation of the solar system, quoted in Chapter 13, gave the inner planets an exponent of about one fifth per million years: a Lyapunov time, one over the exponent, of about five million years. Know the Earth's position to a metre, and call the forecast useless when the error

is the size of its orbit, 150 million kilometres. The horizon is five million years times $\ln 150\,000\,000\,000 \approx 25.7$, some 130 million years, and a millimetre would add only about 35 million. The solar system is chaotic; on the scale of an eclipse table its errors have not yet begun to matter.

Lab 18.1 — The horizon calculator

Write `forecast_length`, which turns an exponent and a number of correct digits into a horizon; `extra_steps`, which says what a more precise measurement buys; and `digits_needed`, which runs the formula backwards. Then ask how many digits a hundred steps of the logistic map would need, against the roughly sixteen decimal digits a computer's ordinary numbers carry (Chapter 16).

Starter: `code/labs/ch18/l18_01_horizon.py`

Test: `code/labs/ch18/test_l18_01_horizon.py`

Run: `cd code && uv run pytest -k l18_01_horizon`

18.3 Question three: what survives the horizon?

Past the horizon the path is lost. That is not the same as knowing nothing, and it is the part of chaos theory most often missed.

Stop and think 18.3

Forty steps ahead, the logistic map's path is lost several times over. Is there anything about step forty you can still predict?

Answer it before you read on.

Answer. Yes: the odds. The chance that x is below 0.25 at step forty is about one in three, and so is the fraction of time any long run spends there. The path is gone; its statistics are not.

Forecasters get such odds by running the model many times, from starts spread across everything the measurement allows, and counting. That is an ensemble, and Chapter 14 builds one for the weather. Here is the smallest version:

`code/ch18/beyond.py`

```

18 def odds(centre: float, width: float, steps: int) -> list[float]:
19     """After each step, the fraction of MEMBERS runs below 0.25. The runs
20     start spread evenly across `width` around `centre`."""
21     runs = [centre + width * (k / (MEMBERS - 1) - 0.5)
22             for k in range(MEMBERS)]
23     fractions = []
24     for _ in range(steps + 1):
25         fractions.append(sum(1 for x in runs if x < 0.25) / MEMBERS)
26         runs = [RULE(x) for x in runs]
27     return fractions
28
29 
```

```

30 def long_run(start: float, steps: int = 100_000) -> float:
31     """The fraction of one long run's steps spent below 0.25."""
32     x, below = start, 0
33     for _ in range(steps):
34         below += x < 0.25
35         x = RULE(x)
36     return below / steps

```

Listing 18.5: A forecast made from an ensemble, and one long run

step	6 digits	9 digits
0	0.00	0.00
4	1.00	1.00
8	0.00	0.00
12	0.00	0.00
16	0.00	0.00
20	0.00	0.00
24	0.34	0.00
28	0.33	0.76
32	0.33	0.36
36	0.34	0.34
40	0.34	0.33

one long run from 0.3: 0.33 of its steps below 0.25
one long run from 0.8: 0.33 of its steps below 0.25

With the start right to six digits, all 1000 runs agree until step 18, and the forecast is one certain answer. Then they spread, the forecast becomes odds, and within a few steps the odds settle at 0.33: the fraction of time one long run spends below 0.25, from either start. With nine digits the runs agree until step 27, and then the same thing happens (Figure 18.2).

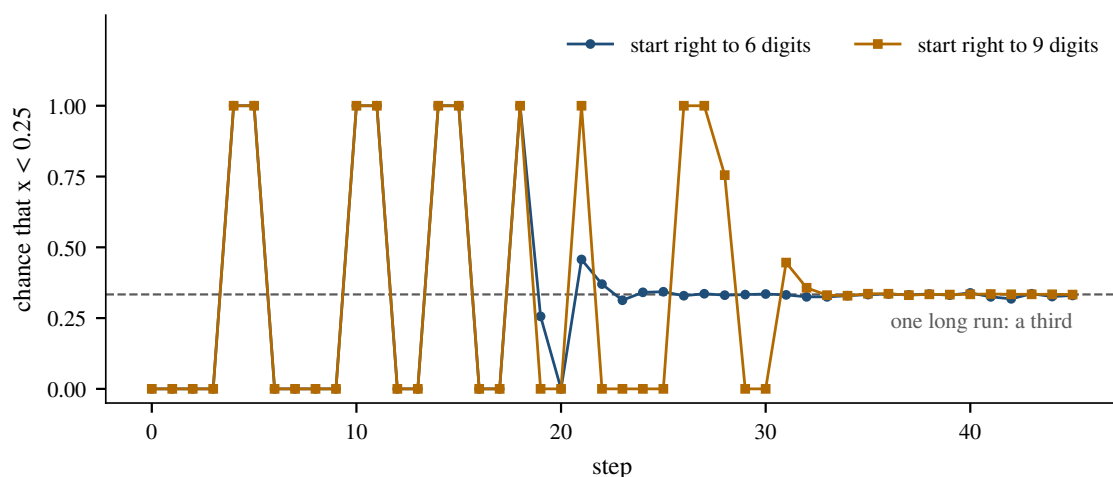


Figure 18.2: An ensemble forecast of the chance that x is below 0.25. It is certain while the runs agree, gives odds as they spread, and settles at the long-run fraction. Three more digits delay this and do not prevent it.

So a forecast of a chaotic system comes in three forms, one after another: a path while

the runs agree, odds as they spread, and the long-run statistics after that. The last is what Chapter 9 calls the statistics of the attractor, and for the weather it is the climate.

The trap

‘Unpredictable means unknowable.’ Beyond the horizon the path cannot be known, and nearly everything else can: where the system can go and where it cannot, how often it visits each region, the odds for any question you care to ask. A 70 per cent chance of rain is not a forecaster hedging; it is the correct answer to a question chaos has made impossible to answer with a yes or a no. A climate projection forecasts statistics, not Tuesdays.

Where you meet this

The ‘cone of uncertainty’ the US National Hurricane Center draws around a storm’s forecast track widens with every day of lead time. It is sized from the errors of the centre’s own recent forecasts, so that the storm’s centre can be expected to stay inside it about two times in three: a horizon and a set of odds in one picture, published for millions of people to act on.

18.4 Question four: chaos, noise, or a small cause?

Irregular data invites a story, and two stories need a question each.

The first says ‘this is chaos’. Chapter 15 shows that a chaotic series and a random one can share a histogram, and that structure tells them apart: plot each value against the next and chaos draws a curve where noise draws a cloud, and chaos can be forecast a step or two ahead where noise cannot. It matters which you have, and how much data there is: a short, noisy record can look like almost anything.

The second story runs backwards, from a big effect to a small cause.

Stop and think 18.4

Nudge the cooling tea and the logistic map at $r = 4$ by a billionth each, as in Section 18.1. Forty steps later, which nudge has changed something? Has it changed the fraction of time the map spends below 0.25?

Answer it before you read on.

Answer. Only the logistic map’s, and only its path: the gap has grown to the size of the map. The fraction of time below 0.25 is still about a third from either start. The tea’s nudge has faded away.

The trap

‘Small causes always matter.’ In a stable system they die away, as the tea’s did. In a chaotic one a small cause changes the details beyond the horizon and nothing about the statistics, and because every other small cause changes the details too, no single one can be picked out afterwards as the reason. The butterfly of Chapter 6 did not cause the storm. Blaming one small cause for a big effect after the fact is a story, not a measurement.

18.5 Question five: can it be steered?

The last question turns the limit into a tool.

Stop and think 18.5

If a tiny change in the start changes the future completely, is that bad news or good news for somebody who wants to control the system?

Answer it before you read on.

Answer. Both. It is why you cannot forecast the system far ahead, and it is why a tiny push, well timed, can move it a long way.

Edward Ott, Celso Grebogi and James Yorke made that the idea of their 1990 paper on controlling chaos: wait until the system passes near one of the unstable cycles hidden in its attractor, nudge a parameter slightly, and it can be held on that cycle. Chapter 17 does it with nudges under one per cent, and makes two chaotic systems keep in step, as Louis Pecora and Thomas Carroll showed in the same year. So chaos is no excuse for fatalism. A system you cannot forecast may be one you can steer, and changing a parameter, such as the forcing in a climate model, moves the statistics even when no path can be forecast.

18.6 The field guide on one page

Here are the five questions, two more the book has earned, and where you measured each:

1. Is there an exact rule, and does a small change in the start grow? (Chapter 6)
2. How far is the horizon, and what would better data buy? (Chapter 7)
3. What survives the horizon: the attractor, the statistics, the odds? (Chapters 9 and 14)
4. Is the irregularity chaos or noise, and is there enough data to tell? (Chapter 15)
5. Can it be steered, or can a parameter move its statistics? (Chapter 17)
6. Does the answer survive a smaller step and more digits? (Chapters 4 and 16)
7. Is a small cause being blamed for a big effect after the fact? (Chapter 6)

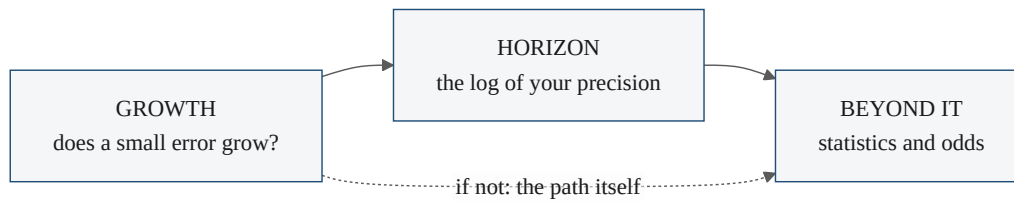


Figure 18.3: The field guide in three moves. If a small error does not grow, the path itself survives; if it does, the horizon says for how long, and beyond it you ask for the statistics and the odds.

And here are familiar forecasts sorted by horizon. Only one figure is worked out in this chapter; the rest are orders of magnitude, some of them folklore, labelled by where they come from.

Forecast	Horizon	Source of the figure	Beyond it, ask for
Where the planets are	about 130 million years	worked above	where their orbits lie
Eclipses	thousands of years	published tables	nothing more
The astronomical tide	years	published tables	the storm surge, which is weather
Detailed weather	a week in practice, two weeks at best	quoted (Lorenz, 1969)	the odds, then the climate
A double pendulum	seconds	order of magnitude	where its energy lets it go
A share price, day to day	none established	irregular is not chaos	the spread of past moves

Now the pension statement and the confident program. A pension projection is what an assumed rate of return would give, so ask for the range of rates. A confident answer about next year's economy has no horizon until somebody says what its rule is, and a confident tone is not an error bar.

Lab 18.2 — The field guide, on a model you write

Write `error_grows`, `growth_rate` and `long_run_fraction`: the first three questions as code you can point at any rule that turns one number into the next. The test runs them on the cooling tea and the logistic map. Then write a rule of your own at the bottom of the file (a tent, a sine, a population with a harvest) and put all seven questions to it.

Starter: `code/labs/ch18/l18_02_guide.py`

Test: `code/labs/ch18/test_l18_02_guide.py`

Run: `cd code && uv run pytest -k l18_02_guide`

18.7 Why everyone needs it

It ended the clockwork, without adding a single random ingredient: an exact rule and a nearly exact start give a future known only as far as a horizon that grows like the logarithm

of your precision.

It says which forecasts to trust, and for how long. An eclipse and a thunderstorm obey laws of the same kind; what differs is the exponent. That teaches humility about long-range certainty, because a single number far beyond the horizon is a guess dressed as a measurement, and trust in short-range models, because inside the horizon they are astonishingly good.

It makes probabilities respectable. Odds are what a correct forecast looks like beyond the horizon, not a failure of forecasting.

It is beautiful and universal. Feigenbaum's constant comes out of a line of algebra and out of real fluids and circuits (Chapter 8); fractals give a coastline a dimension between one and two (Chapter 11); the Mandelbrot set holds the whole road to chaos (Chapter 12).

And it is a resource: control, synchronisation, and the scrambling a computer's random numbers are designed to have (though Chapter 16 warns that a chaotic map alone is a poor source of them).

Chaos theory is not a theory of everything, nor of disorder in general: most of the world's irregularity is noise, ignorance or a rule that keeps changing.

Stop and think 18.6

A colleague says: 'The economy is chaotic, so any forecast is as good as any other.' What is wrong with that?

Answer it before you read on.

Answer. Both halves. Nobody has shown that an economy follows a fixed chaotic rule, and even a chaotic system is not free to do anything: it keeps to its attractor, its odds are fixed, and a short-range forecast of it can be very good.

The trap

'Chaos means anything goes.' Chaos is an exact rule whose motion stays inside a bounded attractor, with steady statistics and a horizon you can compute. Calling something chaotic is a claim that can be tested, not a licence to call anything unpredictable butterfly-shaped, and not a reason to stop forecasting.

18.8 Laplace's demon, one last time

Chapter 1 ended by planting a question, and you can now answer it.

Stop and think 18.7

Suppose the rule is exact and you know it perfectly. Is the future then known, as far ahead as you like?

Answer it before you read on.

Answer. No. It is known only as far as the horizon, which grows like the logarithm of how precisely you know the start. Beyond it, what is known is the attractor, the statistics and the odds.

Laplace's demon needs an infinitely precise present, and every extra digit buys it only a few more steps. The clockwork is real for the eclipse and the tide, and it runs out for the weather, the double pendulum and a single line of algebra.

That leaves the forecasts you meet every day: the train seven minutes away, the rain on Saturday, the projection, the confident answer. Ask each the five questions. You will trust some forecasts more than you used to and some a great deal less, and for the first time you will be able to say why, with a number.

Summary

- **Question one:** run the rule twice from starts a hair apart. If the gap shrinks, the path can be forecast as far as the rule is right.
- **Question two:** the **horizon** is about $\ln(T/e_0)$ divided by the exponent λ , so each correct digit buys the same few steps.
- **Question three:** beyond the horizon the **statistics** and the **odds** survive, and an ensemble shows a path turning into odds.
- **Question four:** irregular is not chaos until structure shows it, and a big effect cannot be pinned on one small cause afterwards.
- **Question five:** sensitivity makes chaos **steerable**.
- The **field guide** adds two questions and sorts familiar forecasts by horizon.
- Chaos ended the **clockwork** and makes odds respectable; it is not a licence for 'anything goes'.
- **Laplace's demon** would know the future only to the horizon.

Can you?

Rate yourself against each item, 1 (no) to 5 (yes, unaided). Anything below 4 is not a matter for reflection: go back to the section that taught it and work its questions again before you move on.

ask of any prediction whether a small change in its start grows	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
compute a prediction horizon, and say what better data buys	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
say what can be predicted beyond the horizon, and read odds	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
separate chaos from noise, and a small cause from a big effect	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
apply the field guide's seven questions to a forecast or a model	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
explain, with numbers you computed, why some things can be predicted for centuries and others only for days	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Checkpoint

Answer these on paper before you look at Appendix A. They test what the chapter taught, in the order it taught it.

1. Two runs of a rule start a billionth apart; thirty steps later they are a trillionth apart. What does question one say?
2. Errors in a system grow with an exponent of 0.5 per day. You know the start to within 0.001 and can tolerate an error of 1. Given that $\ln 1000 \approx 6.908$, what is the horizon?
3. You now measure that start ten times more precisely. Given that $\ln 10 \approx 2.303$, how much longer does the forecast last?
4. Far beyond its horizon, an ensemble of the logistic map at $r = 4$ gives a one-in-three chance that x is below 0.25. Is that forecast worthless?
5. A series of daily measurements looks irregular. Name two things to check before calling it chaos.
6. Why does the property that ruins long-range forecasts of a chaotic system make it possible to steer?
7. Explain why eclipses can be predicted for thousands of years and the weather only for about a week.

Part VI

Appendices

Answers

The checkpoint answers of every chapter, in chapter order. Each was written beside its question and collected here by the build, so an answer cannot drift away from the question it answers.

Chapter 1

1. The state is the time now and the stop you are at; the rule is the timetable (together with the assumption that the bus keeps to it); the prediction is when the next bus arrives.
2. About 19.6 metres in two seconds and about 78.5 in four. Twice the time, four times the distance.
3. 74 °C and then 68.6 °C: the gap goes from 60 to 54 to 48.6.
4. 1.8 degrees: each cup loses a tenth of its own gap to the room, so their difference loses a tenth too.
5. Because every model leaves something out, and usefulness is a question of whether its answers are close enough for the question being asked, over the range it is used for.
6. The rule and the exact present state. The rule may be known exactly; the present state never is, because every measurement has an error.

Chapter 2

1. 4, 10 and 28.
2. $x \mapsto x - 0.2(x - 20)$, which is $0.8x + 4$.
3. About $70/3$, which is 23 years. Counting, it first passes double in year 24; the logarithm says 23.45.
4. $0.2x = 4$, so $x = 20$, the room's temperature; stable, because $|0.8| < 1$.
5. Unstable: every step triples the gap. The orbit from 2 is one away, then 3, 9 and 27 away: 4, 10, 28.
6. 1 degree too hot: the error is halved and flipped.

Chapter 3

1. 1024 on the eleventh and 1023 on the first ten: each square holds one grain more than all the squares before it.
2. Negative, positive, negative.
3. It goes from 0.5 to 0.48 and to 0.42: down by 0.02 and by 0.08. Twice the push does four times the harm.
4. At 0.6 and at 0.2: $1 - 1/2.5$ and $1 - 1/1.25$.
5. 0.005 below the fixed point, then 0.0025 above it. It settles, overshooting, with each swing half the one before.
6. It reacts to the crowding as it builds up, not a generation late: the overshoot comes from the delay, not the speed of growth.

Chapter 4

1. About half a kilometre: 90 kilometres an hour is 1.5 kilometres a minute, and 20 seconds is a third of a minute.
2. $x = 0.1$ and $v = 1$, so the energy is $0.5 + 0.005 = 0.505$: one per cent more, the factor $1 + 0.1 \times 0.1$.
3. About a thousand seconds: a step a hundred times smaller gains a hundred times less per second.
4. The rates at the start; half a step ahead at the first rates; half a step ahead at the second; a whole step ahead at the third. It moves at their average with weights 1, 2, 1.
5. Swinging back and forth; going over the top and turning; losing energy to friction and settling to hang at rest.
6. Its paths can never cross, because each state has one future, so a bounded path can only settle to a fixed point or wind onto a loop (Poincaré–Bendixson). Chaos in continuous time needs at least three.

Chapter 5

1. $x_1 = 3 \times 0.5 \times 0.5 = 0.75$ and $x_2 = 3 \times 0.75 \times 0.25 = 0.5625$.
2. $1 - 1/2.5 = 0.6$. The slope there is $2 - 2.5 = -0.5$, between -1 and 1 , so it holds.
3. The slope there is $2 - 3.2 = -1.2$: a nudge flips side and grows at every step, so the orbit is pushed off, onto a cycle of two.
4. Going round a cycle of period four; the corners are the four values it visits in turn.
5. That no cycle up to the length it checked was found in the stretch it looked at. Not that the orbit is chaotic: a longer cycle, or an orbit still settling, gives None too.
6. The weather or other chance; errors in counting; and the population's own deterministic rule of growth, as in the logistic map at $r = 3.9$.
7. The same start always gives exactly the same orbit. At $r = 3.9$ it never settles or repeats, so it shows no pattern, although every value is fixed by the one before.

Chapter 6

1. At step 30. Until step 16 they differ by less than a millionth.
2. After 20 steps: 2^{20} is a little over a million, while 2^{19} is only about half a million.
3. Because every x lies between 0 and 1, so two values can never be more than 1 apart. Once the gap reaches the size of the whole range it wanders, averaging about 0.4.
4. $\frac{1}{2} + \frac{1}{4}$, so 0.11 in binary. The halves are upper, upper, lower: 0.75, then 0.5, then 0. The map reads out the digits 1, 1, 0.
5. About 44 steps. Each decimal digit adds a little over three steps, so four more add about thirteen.
6. The first run again, digit for digit, because the rule is deterministic: the same start always gives the same future.
7. No: every value follows exactly from the one before. It is deterministic, because the same start gives the same future; it is not predictable for long, because no measured start has enough digits for the rule to use.

Chapter 7

1. It has not grown at all: the product is 1. The ordinary average of the factors is $\frac{5}{3}$, which suggests growth that never happened.
2. $\ln 20 = \ln 2 + \ln 10 \approx 2.99$. And $0.2 \times 10 = 2$, so $\ln 0.2 = \ln 2 - \ln 10 \approx -1.61$.

3. $\ln 0.5 \approx -0.69$. It is negative: errors shrink, halving at every step.
4. About 9: $\ln(0.1/0.001) = \ln 100 = 2 \ln 10 \approx 4.6$, divided by 0.5.
5. About 9 steps each time: $\ln 100/0.5 \approx 9.2$. Each hundredfold improvement buys the same fixed number of steps.
6. Because the horizon depends on the error in the start only through its logarithm, and a faster computer does not measure the start any better. Doubling the horizon needs twice as many correct digits in the start.
7. A window: inside the chaotic range, the orbit has settled into a cycle, so errors shrink again and the exponent dips below zero.

Chapter 8

1. Where the orbit lives in the long run at that r , after the transient: one dot for a fixed point, two for a two-cycle, a smear for chaos.
2. About 4.7, from $0.08 \div 0.017$; so the next gap should be about $0.017 \div 4.7$, roughly 0.0036.
3. Near a split the cycle losing its grip multiplies a nudge by nearly -1 each time round, so it settles extremely slowly, and a finite run still shows it flipping between values.
4. The r at which the cycle passes through the top of the hump, $x = 0.5$, where the slope is zero. It solves one equation (start at 0.5, take the period's number of steps, be back at 0.5), which bisection solves by halving a bracket.
5. No. Universality says the ratio of successive gaps tends to the same number, about 4.6692, for every rule with one smooth hump; where each road ends depends on the rule.
6. No: 3.83 lies in the period-three window, which opens at $1 + \sqrt{8}$, about 3.82843. Li and Yorke proved that a continuous rule taking an interval into itself, with a point of period three, has cycles of every length and starts that never settle into any cycle.

Chapter 9

1. It makes x grow: the roll speeds up towards the temperature difference.
2. The still pan, $(0, 0, 0)$, and the two steady rolls, $x = y = \pm\sqrt{72}$ (about 8.49) with $z = 27$.
3. In a plane a path cannot cross itself, so it can only settle, spiral or go round a fixed loop. Lorenz's model uses three, the fewest that let a path pass over and under itself.
4. About $\ln(1000)/0.9$, roughly 8 time units more: from about twenty to about twenty-eight, not a thousand times as long.
5. It will not: every run is drawn onto the same bounded attractor, where z stays between about 1 and 48.
6. A half. Changing the signs of x and y together leaves the rules unchanged, so the two wings are mirror images.
7. The average height z , between 23.48 and 23.57 from five different starts; the Lyapunov exponent, about 0.9, is another.

Chapter 10

1. At 0.6, then at 0.8: doubling 0.6 gives 1.2, past the end, so it folds back to $2 - 1.2 = 0.8$.
2. 0.08. The strip is only one long, so sooner or later the fold catches one raisin and not the other.
3. $1 - 1.4 \times 0.25 + 0 = 0.65$ and $0.3 \times 0.5 = 0.15$: the point $(0.65, 0.15)$.

4. $2 \times 0.027 = 0.054$. No: the patch is squeezed one way and stretched another, and its points keep moving.
5. The attractor is the set the motion settles onto and never leaves; its basin is every start that ends up on it.
6. A smooth curve looks more and more like a straight line. Each line of the attractor splits into a bundle of lines, again and again.
7. It keeps only the points where a flow's path crosses a surface; the rule from one crossing to the next is a map. The three-number rule is about flows, and a map jumps, so it can fold.

Chapter 11

1. Both can be right if they measured with different rulers (maps of different scale): the shorter ruler follows more bays. Each should have said what ruler or map scale it used.
2. $\left(\frac{4}{3}\right)^3 = \frac{64}{27}$, about 2.37.
3. $\frac{3}{5}$ of the length. Three copies, each shrunk by five, give $D = \ln 3 / \ln 5$, about 0.68.
4. $2^3 = 8$, so $D = 3$: a solid cube.
5. About 1.5, because $2^{1.5} = 2 \times \sqrt{2}$, about 2.83.
6. Too few points leave some of the smallest boxes unvisited, which undercounts them and flattens the fitted line. More points fill them in.
7. Over what range of sizes, and measured how. A real object can be fractal only between a largest and a smallest size.

Chapter 12

1. $3 + 2i$, and 2: multiplying out gives $1 - i + i - i \times i = 1 + 1$.
2. $1 + i$ has angle 45 degrees, so its square has angle 90 and length 2 (the length of $1 + i$, squared): it is $2i$. Multiplying out, $1 + 2i + i \times i = 2i$.
3. At step 2: the orbit is 0, 2, 6, and 2 is not past 2 but 6 is.
4. Yes. Every real c from -2 to $\frac{1}{4}$ is in it; the orbit of 0 stays between -2 and 2.
5. Yes. Its length is the square root of $0.36 + 0.81$, a little more than 1, and squaring keeps making a length above 1 larger.
6. $c = 1.6 - 2.56 = -0.96$, inside the big bulb, which runs from -0.75 to -1.25 : a cycle of two, as Chapter 5 found at that r .
7. Its heart and bulbs are values of c at which the orbit of 0 settles into a cycle. The chaotic values sit on the thin edge, which on the real axis is the needle between its windows.

Chapter 13

1. Fixing the energy of a single pendulum's two-number state leaves one direction, round a closed loop. Fixing it for the double pendulum's four numbers leaves three, which is enough for chaos.
2. The bobs are one and two metres up, so $9.81 \times (1 + 2)$, about 29.4 joules.
3. The stepper's error is too large to trust. Shrink the step until the energy holds, then check that your answer survives halving it again.
4. That the parting was made at least partly by the stepper's errors, so neither number can be trusted. A parting that is physics does not move when the step changes.
5. Its energy, which never changes and decides which states can ever be reached.
6. By e twice: 2.7×2.7 , a little over seven.

7. No. Their positions cannot be predicted beyond some tens of millions of years, but computations over billions of years find their orbits keeping much the same size and shape in almost every run.

Chapter 14

1. It is the best estimate of the present state, built from measurements that are never exact. Its error grows, as in any chaotic system, until the forecast has forgotten the start.
2. 24 per cent: 12 out of 50.
3. No. A reliable 80 per cent is followed by rain about 80 times in 100; these forecasts were too sure of themselves.
4. By then it is one more day of weather, unrelated to the truth, and two unrelated days differ by about $\sqrt{2}$ times the typical swing, while the average is off by only the swing.
5. Twice 1.35, about 2.7 time units: each tenfold improvement buys the same extra time.
6. The first is a weather question, far beyond the horizon, so the best answer is the climate's odds. The second is a climate question about statistics under a stated forcing, which chaos does not rule out.

Chapter 15

1. It keeps every value, so the histogram. It destroys the order, which is where a rule linking each value to the next would show.
2. That each value is fixed by the one before it: a rule of one number made it. Noise would spread the points over the square.
3. The first: a curve passes through only so many squares, however many points land on it. The second keeps filling the square, as noise does.
4. About eight: $2^8 = 256$, and 0.001×256 is about 0.26.
5. Both tests are unreliable on so little: in this chapter's measurement the squares test on ten noisy values was right less often than a tossed coin would be.
6. Its years go from all alike, to alternating, to repeating every four years, to never repeating; and its chaotic troughs are so deep that a real disease would die out.
7. No: noise is irregular too. Evidence would be a forecast that beats surrogates, with an error that starts small and grows steadily, in a long, clean record.

Chapter 16

1. Neither 0.1 nor 0.2 has an exact binary form, so each is stored as the nearest double, and the sum is rounded again.
2. At step 54, one step per binary digit. The true orbit goes $\frac{3}{5}, \frac{1}{5}, \frac{2}{5}, \frac{4}{5}, \frac{3}{5}$ and loops for ever.
3. About twenty (measured: 23): the map doubles the rounding error every step, using up one binary digit per step.
4. About 732 924: the digits double every step, and ten doublings multiply them by about a thousand.
5. The fraction of time. The path has followed a different orbit since about step fifty, but it stays close to a true orbit from a nearby start, and that orbit spends its time as true orbits do.
6. The machine holds only finitely many numbers, so the orbit must revisit a value, and the rule then repeats everything after it.

Chapter 17

1. About 0.714, with slope -1.5 . A miss is multiplied by 1.5 and flips side at every step, so the orbit cannot stay.
2. About 0.0038 above it. Lower r by about $0.0038/0.19$, that is by 0.02, for that one step.
3. Because the controller needs to predict only one step ahead, and over one step a miss grows by a factor of about two, not exponentially without end.
4. The reach halves, so the target is half as wide and the average wait roughly doubles.
5. Because the calm map settles at its own fixed point, $1 - 1/2.9$, about 0.655. The value 0.744 is a fixed point only at $r = 3.9$, where it is unstable.
6. The original's x . No: the copy agrees with the original at each moment, which is synchronisation, not prediction.
7. Any two of lasers, electronic circuits and isolated heart tissue in the laboratory. It is not a routine clinical or industrial technique.

Chapter 18

1. The error shrinks, so the path can be forecast as far ahead as the rule itself is right.
2. About $6.908/0.5 \approx 13.8$ days.
3. About $2.303/0.5 \approx 4.6$ days more: a fixed extra stretch, not ten times as long.
4. No. It is the correct forecast: the fraction of time the map spends there, the same from any start.
5. Whether each value plotted against the next draws a curve rather than a cloud; whether it can be forecast a step or two ahead from similar past moments; and whether the record is long enough to tell.
6. It is sensitive to small changes: a tiny error grows, and so does the effect of a tiny, well-timed push.
7. The horizon is $\ln(T/e_0)$ divided by the exponent. The solar system's exponent is about one fifth per million years, which gives a horizon of about a hundred million years; the atmosphere's errors grow as much in days. Better data adds a fixed stretch to either.

Misreadings of chaos

Every misreading the book sets out to correct, in the order the chapters elicit them. Each is written in the voice of somebody who believes it, followed by what is true instead and the section where the book lets you walk into it first. If you arrived here before reading that section, the entry will make more sense after it: the point of each trap was to be caught by it.

Chapter 1 *A model is a machine for predicting*

- 11 **‘A model is a copy of reality.’** A model is a machine for answering questions: numbers in, numbers out, and the only fair test is whether its answers agree with what you then measure. § 1.1
- 12 **‘A model that is wrong is useless.’** Every model is wrong, because it leaves something out. The useful question is whether it is good enough, for this question, over this range. § 1.4

Chapter 2 *The rule of the next step*

- 21 **‘Exponential growth means fast growth.’** It means growth in proportion to what is already there, and the multiplier can be close to one: at 1 per cent a year a sum takes about 70 years to double. What exponential growth guarantees is not speed but that it eventually overtakes any growth that adds a fixed amount. § 2.3
- 22 **‘A fixed point is where the system ends up.’** It is where the system stays if it is exactly there. Only a stable fixed point, where each step shrinks the gap ($|a| < 1$ for $x \mapsto ax + b$), draws nearby states in; an unstable one, like a loan’s break-even balance, pushes them away. § 2.5

Chapter 3 *Feedback: why nothing grows forever*

- 31 **‘The effect of two pushes is the sum of the effects.’** Only for a straight-line rule. On a curved one, which is what feedback makes, the effect of a push depends on where the system already stands: two pushes can together do much more, much less or the opposite of their sum. § 3.4
- 32 **‘A limit to growth means growth stops smoothly.’** With a gentle growth rate it does. With a strong one, or with a delay between cause and effect, the population overshoots its level and oscillates, and turned up further the swings need not die away at all. § 3.6

Chapter 4 *Motion: rules of change and the map of every state*

- 41 **‘A smaller step makes a simulation right.’** It makes it right for longer. A stepping method’s error is made fresh at every step and piles up: Euler with steps ten times smaller gains the same energy on a spring, ten times later. § 4.2

- 42** ‘**A pendulum is a simple system, so its motion must be simple.**’ One pendulum has two numbers of state, and a smooth rule in a plane can only settle, cycle or run off. Hang a second from it and the state has four numbers, and the motion can be chaotic. § 4.5
- 43** ‘**A pendulum’s swing takes the same time however far it swings.**’ Only for small swings. A one-metre pendulum released at 90 degrees takes about 18 per cent longer than one released at 10, and near the upside-down position the swing time grows without limit. § 4.4

Chapter 5 *One line of algebra, every kind of behaviour*

- 51** ‘**A simple rule gives simple behaviour.**’ One line of arithmetic, $x_{n+1} = r x_n (1 - x_n)$, gives a steady state, cycles and orbits that never repeat as the single knob r is turned. How complicated behaviour looks says very little about how complicated its rule is. § 5.2
- 52** ‘**If it has not repeated after a thousand steps, it is about to.**’ At $r = 3.9$ no cycle pulls the orbit in: in a hundred thousand steps of the logistic map not one value comes back, and a longer run only produces more values never seen before. § 5.5
- 53** ‘**Irregular output needs a random or complicated cause.**’ A simple deterministic rule with no randomness in it can produce output as irregular as chance. The output alone does not say which of the two made it; that was May’s 1976 warning to ecologists. § 5.6

Chapter 6 *The butterfly effect, measured*

- 61** ‘**If I run a chaotic model again, I will get something different.**’ Not from the same start: a chaotic rule gives the same future from the same start, to the last digit. What changes the future is a changed start, however small. § 6.5
- 62** ‘**The butterfly causes the tornado.**’ Lorenz’s question was about prediction, not power: a tiny disturbance changes which weather happens, and since every other disturbance does the same, no single one of them is the cause. § 6.5
- 63** ‘**Chaos means random.**’ A chaotic system obeys its rule exactly at every step. What looks like randomness is the rule reading out digits of the start that nobody knows. § 6.6

Chapter 7 *A number for chaos: the Lyapunov exponent*

- 71** ‘**To find the typical stretch per step, average the stretches.**’ Growth compounds, so the typical factor is the one that, used at every step, gives the same product. Average the logarithms of the stretches, not the stretches: a step of $\times 4$ and a step of $\times \frac{1}{4}$ leave a gap where it was. § 7.1
- 72** ‘**Ten times better data gives ten times longer forecasts.**’ The horizon is about $\ln(T/\delta)/\lambda$, so a tenfold better start buys a fixed $\ln 10/\lambda$ extra steps. For the logistic map at $r = 4$ a thousandfold improvement buys about ten steps, whatever the error was before. § 7.4
- 73** ‘**A bigger computer will eventually make any system predictable.**’ A faster computer does not measure the start any better, and the horizon depends on the error in the start only through its logarithm. Doubling the horizon needs twice as

many correct digits; a thousand steps of the logistic map at $r = 4$ would need the start to about three hundred decimal places. § 7.4

Chapter 8 *The road to chaos, and a number nobody expected*

- 81 **‘The change from order to chaos is a sudden switch.’** On the logistic map’s road, order breaks down in an infinite cascade of period doublings, each arriving sooner than the last; chaos begins where the cascade runs out, at $r \approx 3.5699$. § 8.1
- 82 **‘A different equation gives a different route to chaos.’** Any rule with a single smooth hump takes the same road: its doublings sit at different values of the knob, but the gaps between them shrink by the same ratio, Feigenbaum’s $\delta \approx 4.6692$. § 8.4
- 83 **‘Once a system has reached chaos, it is chaotic everywhere beyond that point.’** Beyond the cascade the logistic map returns to order infinitely many times, in windows of every period; the widest, of period three, opens at $r = 1 + \sqrt{8}$. § 8.5

Chapter 9 *Three equations for the weather: the Lorenz attractor*

- 91 **‘Chaos needs many variables.’** Lorenz’s model has three numbers and two multiplications, and it is chaotic. Three is the fewest a flow needs, because a path in a plane cannot cross itself; it is also enough. § 9.2
- 92 **‘A chaotic system can go anywhere.’** Unpredictable is not unbounded. Every Lorenz run, from almost any start, is drawn onto the same bounded butterfly-shaped set, the attractor, and never leaves it. § 9.4
- 93 **‘If the path is unpredictable, nothing about it is predictable.’** The path is unpredictable beyond about twenty time units, but the statistics of the attractor — the fraction of time on each wing, the average height, even the Lyapunov exponent — come out the same from every start. § 9.5

Chapter 10 *Stretch and fold: strange attractors*

- 101 **‘If a system can never leave a small region, it cannot be very sensitive to its start.’** The region caps how *big* an error can get, not how fast it grows. A tiny error doubles and doubles until it is as large as the region, and from then on two runs are as unrelated as two points dropped in it at random. § 10.1
- 102 **‘If the area keeps shrinking, the motion must be dying out.’** Hénon’s map multiplies every area by 0.3 each step because it squeezes hard in one direction while it stretches in another. A patch becomes a longer and thinner ribbon whose points keep moving for ever. § 10.4
- 103 **‘Zoom in far enough on any curve and it turns into a straight line.’** That is true of smooth curves. Each line of a strange attractor splits under magnification into a bundle of lines, and each of those splits again: layers within layers at every scale. § 10.5

Chapter 11 *Fractals: measuring what has no length*

- 111 **‘A coastline has a length; it is only hard to measure.’** A length is what measurements settle on as the ruler shrinks, and a coast’s measurements do not settle: the Koch curve’s grow by $\frac{4}{3}$ every time the ruler shrinks by three. A coastline’s length is a fact about the coast and the ruler together. § 11.2

- 112** **‘A dimension must be a whole number.’** Counted from copies or from boxes, a dimension is a measurement, and it need not be whole: the Koch curve’s is $\ln 4 / \ln 3$, between one and two, and the Cantor set’s $\ln 2 / \ln 3$, between nothing and one. Whole numbers belong to smooth shapes. § 11.3
- 113** **‘Fractals are decoration: pretty pictures, no use.’** Fractal branching in lungs, blood vessels and rivers does work: it packs a large surface or a long edge into a bounded space. Real objects are fractal only over a range of sizes, and a claimed dimension should say which. § 11.6

Chapter 12 *Iterating in the plane: Julia and Mandelbrot*

- 121** **‘Complex numbers are imaginary, so they describe nothing real.’** The name is history, not a description. A complex number is a point of the plane with a rule for multiplying that stretches and turns, so anything that stretches and turns — a wave, a rotation, an alternating current — is described by complex numbers. § 12.1
- 122** **‘Infinite detail means infinite information.’** Every detail of the edge of the Mandelbrot set follows from one line of algebra, $z \rightarrow z^2 + c$, and one question: does the orbit of 0 escape? The detail is generated, not stored, and a short description can have an endless consequence. § 12.3
- 123** **‘The Mandelbrot set is a picture of chaos.’** Most of what is black is order: in the heart the orbit of 0 settles on one point, and in every bulb on a cycle whose length the bulb fixes. The never-settling orbits of the logistic map sit on the thin edge, which on the real axis is the needle between its windows. § 12.5

Chapter 13 *Pendulums and planets*

- 131** **‘A pendulum is a simple machine, so its motion must be simple.’** True for one pendulum, whose state is two numbers. Hang a second from it and the state is four numbers, and a release from high up never settles into a repeating pattern. § 13.1
- 132** **‘A system that keeps its energy, with no friction and nothing random in it, cannot be chaotic.’** Fixing the energy of a four-number state still leaves three directions to move in, which is enough for chaos. Two double pendulums a millionth of a radian apart part within seconds, with the energy of each constant to about one part in 10^8 . § 13.3
- 133** **‘The solar system is a perfect clock.’** Over thousands of years it is an excellent one. Laskar found the inner planets chaotic, with a Lyapunov time of about five million years, so where Earth will be on its orbit a hundred million years from now cannot be predicted. § 13.6
- 134** **‘A chaotic solar system is an unstable one: the planets will fly apart.’** Unpredictable is not unstable. Computations over billions of years find the planets keeping orbits of much the same size and shape in almost every run; only their positions along those orbits cannot be foretold. § 13.6

Chapter 14 *Weather, climate and the ensemble forecast*

- 141** **‘A forecast is a single number.’** A forecast is a number and how far to trust it. The same ensemble mean deserves confidence on a day the members agree and a shrug on a day they scatter, and the spread of the ensemble tracks the size of the error without anyone seeing the truth. § 14.2

- 142** ‘**A 30 per cent chance of rain, followed by rain, was a wrong forecast.**’ One day cannot refute a probability. A forecast of 30 per cent is right if it rains on about 30 per cent of the days it is issued; that check, reliability, needs many forecasts, and a good ensemble passes it. § 14.3
- 143** ‘**If the weather cannot be predicted, the climate cannot be predicted either.**’ Weather is the path and climate is the statistics of the attractor. Two runs that disagree about every day agree about the average, the swing and how often it rains; a change in the forcing moves those statistics, which is the question a climate projection asks. § 14.5

Chapter 15 *Life, markets and noise: telling chaos from chance*

- 151** ‘**A random-looking sequence must come from randomness.**’ The logistic map at $r = 4$ looks as random as a shuffle of its own values and has no chance in it at all. Looking random is a fact about the look; only a test of order says anything about the cause. § 15.1
- 152** ‘**Irregular data proves chaos.**’ Chaos and noise are both irregular. A claim of chaos needs evidence of a rule: structure in the return map, a forecast that starts good and worsens at a steady rate, and a record long and clean enough for either to mean something. § 15.5
- 153** ‘**Chaos in markets means you can predict them.**’ Chaos is forecastable only a few steps ahead and not at all beyond its horizon, so chaotic prices would promise a short horizon with a hard limit, not a crystal ball; and the evidence for low-dimensional chaos in prices is weak and contested. § 15.6
- 154** ‘**A good forecast one step ahead proves chaos.**’ A series with memory, like daily temperature, is forecastable a day ahead with no chaos at all. The signature of chaos is a forecast that is good a step ahead and worsens at a steady rate, and that beats surrogates keeping the series’ slow ups and downs. § 15.6

Chapter 16 *Chaos inside the computer*

- 161** ‘**A deterministic program always gives the same answer.**’ On the same machine, with the same compiler and libraries, it does. Change any of those, or the order of a sum, and the roundings change; a chaotic computation carries a last-digit difference to the first digit in about fifty steps. § 16.3
- 162** ‘**More decimal places make a simulation exact.**’ Each extra digit buys a fixed number of correct steps, never all of them: at $r = 4$ the logistic map uses up one binary digit per step, and exact fractions double their digits at every step. § 16.4
- 163** ‘**A chaotic map makes a good random-number generator.**’ Chaos gives unpredictability and nothing else a generator needs: the logistic map’s values pile up near 0 and 1, each fixes the next, and on a computer it falls into a short loop. Good generators are designed with number theory to have one enormous loop. § 16.6

Chapter 17 *Taming chaos: control and synchronisation*

- 171** ‘**Chaos cannot be controlled.**’ Chaos forbids prediction far ahead, and a controller only needs to predict one step: wait until the orbit passes near an unstable orbit,

then cancel each coming miss with a nudge to a parameter. The logistic map at $r = 3.9$ is held at its unstable fixed point with nudges under one per cent. § 17.2

- 172** **‘Controlling a system means overpowering it.’** The controller supplies none of the motion: it keeps one of the system’s own unstable behaviours by cancelling small misses, and once the orbit is caught the nudges shrink to almost nothing. Force, a large permanent change of the parameter, cannot even give the same state. § 17.3

- 173** **‘Two chaotic systems can never agree.’** Left to themselves they never stay together. Coupled so that one listens to the other, they can agree exactly: a copy of the Lorenz system handed the original’s x falls into step with it. The agreement is with the present, not a prediction of the future. § 17.4

Chapter 18 *Why you need chaos: a field guide to prediction*

- 181** **‘Unpredictable means unknowable.’** Beyond the horizon the path is lost, and nearly everything else is known: where the system can go, how often it visits each region, and the odds. A forecast given as odds is the correct answer, not a hedge. § 18.3

- 182** **‘Small causes always matter.’** In a stable system they die away. In a chaotic one a small cause changes the details beyond the horizon and nothing about the statistics, and no single small cause can be blamed for a big effect after the fact. § 18.4

- 183** **‘Chaos means anything goes.’** Chaos is an exact rule confined to a bounded attractor, with steady statistics and a horizon you can compute. Calling something chaotic is a claim to be tested, not a reason to stop forecasting. § 18.7

The mathematics you need, on four pages

Every tool the book uses, in one place, with the chapter that introduces it. If a symbol stops you in the middle of a chapter, it is here.

C.1 Very large and very small numbers

A number like 0.00000000015 is hard to read, so it is written 1.5×10^{-10} : the digits that matter, and how many places to move the decimal point. 10^3 is a thousand, 10^6 a million, and a negative power moves the other way, so 10^{-3} is a thousandth. When the book says two numbers differ ‘in the tenth decimal place’, it means by something like 10^{-10} . The book rounds what it prints and says how many figures it keeps: a result is only worth the digits the measurement behind it can support (Chapter 1).

C.2 Rules, steps and orbits

A rule that turns the present into the next moment is written

$$x_{n+1} = f(x_n),$$

read ‘the next x is f of this x ’. The small n below the line counts the steps: x_0 is the start, x_1 the state after one step, and so on. The whole list $x_0, x_1, x_2, \dots$ is called an *orbit*. A *fixed point* is a state the rule leaves where it is, a solution of $x = f(x)$ (Chapter 2).

C.3 Powers, growth and halving

Multiplying by the same factor a again and again gives powers: $a \times a \times a = a^3$, and n steps of multiplying by a multiply by a^n . If a is bigger than 1 the result grows, faster and faster; if it is between 0 and 1 it shrinks towards zero. Doubling ten times multiplies by $2^{10} = 1024$, a little over a thousand, which is worth remembering: ten doublings make a thousand, twenty make a million, and thirty a billion (Chapters 2, 3 and 6).

C.4 Logarithms

A logarithm answers the question ‘how many times?’. The natural logarithm, written $\ln$, turns multiplying into adding:

$$\ln(a \times b) = \ln a + \ln b.$$

So if an error is multiplied by 2 at every step, $\ln$ of the error grows by $\ln 2 \approx 0.693$ at every step, a steady amount, and a steady amount is something you can average and compare. The number whose $\ln$ is 1 is called $e \approx 2.718$, and $\ln 10 \approx 2.303$.

The question ‘how many doublings multiply by N ?’ has the answer $\ln N / \ln 2$. Multiplying by a thousand takes about 9.97 doublings, and one extra decimal digit of precision is worth about 3.32 doublings. That single fact is behind the forecasting horizon of Chapter 7 and the book’s whole argument about prediction: precision is paid for in digits, and each digit buys only a fixed number of steps.

C.5 The slope of a curve

How steep is a curve at a point? Move the input a tiny amount h either side, see how much the output changes, and divide:

$$\text{slope of } f \text{ at } x \approx \frac{f(x+h) - f(x-h)}{2h}.$$

For $f(x) = x^3$ at $x = 2$ the exact slope is 12, and with $h = 0.01$ the recipe gives 12.0001. A slope tells you how much an error in the input is magnified in the output: a slope of 3 triples it, a slope of $-\frac{1}{2}$ halves it and flips its sign. The size of a number without its sign is written with bars, $|-3| = 3$ (Chapter 2).

C.6 Rates of change

When time runs smoothly rather than in steps, a rule says how fast each quantity is changing right now. That is written

$$\frac{dx}{dt} = (\text{some formula}),$$

read ‘the rate of change of x with time’. You never need to solve such a rule with algebra in this book: a computer follows it forward in many small steps of time, and Chapter 4 shows how to do that well.

C.7 Sums and averages

A long sum is written with a capital Greek sigma:

$$\sum_{n=1}^N a_n = a_1 + a_2 + \cdots + a_N,$$

and the average of N numbers is that sum divided by N . The Lyapunov exponent of Chapter 7 is an average of this kind: the average of $\ln$ of how much each step stretches a small error.

C.8 Remainders

‘ $x \bmod 1$ ’ means ‘throw away the whole-number part and keep what is after the decimal point’, so $2.75 \bmod 1 = 0.75$. Chapter 6 uses it to keep a number between 0 and 1.

C.9 Complex numbers

A complex number is a point in a flat plane, written $a + bi$: a steps across and b steps up. Adding two of them adds the steps. Multiplying uses ordinary algebra plus one new rule, $i \times i = -1$, so

$$(1 + 2i)(3 + 4i) = 3 + 4i + 6i + 8i^2 = -5 + 10i.$$

The distance of $a + bi$ from the centre of the plane is $\sqrt{a^2 + b^2}$ (Chapter 12).

C.10 The Greek letters

The book uses a handful, always for the same thing:

Letter	Name	What it is in this book
λ	lambda	the Lyapunov exponent (Chapter 7)
δ	delta	Feigenbaum’s constant (Chapter 8)
σ, ρ, β	sigma, rho, beta	Lorenz’s three settings (Chapter 9)
ε	epsilon	a small error or a small nudge
Σ	capital sigma	a sum

Further reading

Two lists: the papers in which the ideas of this book first appeared, and the books to read next. The papers are for looking up where something came from; most of them need more mathematics than this book assumes, but their first pages are often clearer than anything written about them since. The books are for going further, and each comes with a note on what it is good for and what it asks of you.

D.1 Where the ideas first appeared

Pierre-Simon Laplace, *Essai philosophique sur les probabilités*, 1814. The clockwork universe stated at its most confident; quoted in Chapter 1.

Henri Poincaré, *Science et méthode*, 1908. The first clear statement, in words, that small differences at the start can produce very large ones in the end, and that prediction then becomes impossible.

Edward N. Lorenz, ‘Deterministic nonperiodic flow’, *Journal of the Atmospheric Sciences* 20 (1963), 130–141. The three equations of Chapter 9, and the paper that started the modern subject.

Tien-Yien Li and James A. Yorke, ‘Period three implies chaos’, *American Mathematical Monthly* 82 (1975), 985–992. The paper that gave the subject its name (Chapter 8).

Robert M. May, ‘Simple mathematical models with very complicated dynamics’, *Nature* 261 (1976), 459–467. The logistic map as a warning to everyone who builds models (Chapter 5).

Michel Hénon, ‘A two-dimensional mapping with a strange attractor’, *Communications in Mathematical Physics* 50 (1976), 69–77. The map of Chapter 10.

Mitchell J. Feigenbaum, ‘Quantitative universality for a class of nonlinear transformations’, *Journal of Statistical Physics* 19 (1978), 25–52. The constant of Chapter 8.

Benoit B. Mandelbrot, ‘How long is the coast of Britain? Statistical self-similarity and fractional dimension’, *Science* 156 (1967), 636–638. Fractal dimension, before the word fractal (Chapter 11).

Floris Takens, ‘Detecting strange attractors in turbulence’, in *Dynamical Systems and Turbulence*, Lecture Notes in Mathematics 898 (1981), 366–381. Rebuilding a system’s shape from one measured quantity (Chapter 15).

Jacques Laskar, ‘A numerical experiment on the chaotic behaviour of the Solar System’, *Nature* 338 (1989), 237–238. Chaos in the orbits of the planets (Chapter 13).

Edward Ott, Celso Grebogi and James A. Yorke, ‘Controlling chaos’, *Physical Review Letters* 64 (1990), 1196–1199. Chapter 17.

Louis M. Pecora and Thomas L. Carroll, ‘Synchronization in chaotic systems’, *Physical Review Letters* 64 (1990), 821–824. Chapter 17.

Edward N. Lorenz, ‘Predictability: a problem partly solved’, *Seminar on Predictability*, European Centre for Medium-Range Weather Forecasts, 1996. The toy atmosphere of Chapter 14.

D.2 Books to read next

James Gleick, *Chaos: Making a New Science*, 1987. The story of the people who found chaos, told as history rather than mathematics. The best book to read alongside this one.

Edward N. Lorenz, *The Essence of Chaos*, 1993. One of the founders explaining the subject to a general reader, gently and with care; a natural next step after Parts II and III.

Ian Stewart, *Does God Play Dice? The Mathematics of Chaos*, 1989. A lively popular account by a mathematician, with more mathematics than Gleick and less than a textbook.

Leonard A. Smith, *Chaos: A Very Short Introduction*, 2007. Short and careful, and especially good on what chaos does and does not mean for forecasting.

Tim Palmer, *The Primacy of Doubt*, 2022. Ensemble forecasting and the uses of uncertainty, from a meteorologist who helped build it; the book to read after Chapter 14.

Benoit B. Mandelbrot, *The Fractal Geometry of Nature*, 1982. The founding book of fractals, discursive and full of pictures.

Heinz-Otto Peitgen, Hartmut Jürgens and Dietmar Saupe, *Chaos and Fractals: New Frontiers of Science*, 1992. A very large, very generous book that builds everything from experiments you can do; the natural continuation of this one for a reader who liked the labs.

Steven H. Strogatz, *Nonlinear Dynamics and Chaos*, 1994. The textbook to read when you are ready for calculus: clear, applied and much loved. It is where this book’s subject becomes a university course.

Kathleen T. Alligood, Tim D. Sauer and James A. Yorke, *Chaos: An Introduction to Dynamical Systems*, 1996. A mathematics textbook written by one of the subject’s founders and his colleagues, with the proofs.

Manifest

What the book contains, counted rather than claimed. Every number below is computed from the repository by the build, which refuses to finish if the book has changed and this page has not.

The ledger

Chapters written, of 18	18
Listings, every one run by the build	80
Labs, each with a starter that fails and a solution that passes	37
Tests that check the labs	132
Stop-and-think questions	131
Checkpoint questions, each answered in Appendix A	119
Misreadings in Appendix B	53
Plots	52
Diagrams	17
Printed outputs of listings	80
Computed numbers in the text	416
Lines of the chaoslab library	447
Pieces of code printed without being run	0

Lab manifest

1.1	101_01_stone — The stone, both ways round	4
1.2	101_02_cooling — How long until I can drink it?	6
2.1	102_01_until — How many steps until?	18
2.2	102_02_fixed — Where does it stop, and does it stay?	21
3.1	103_01_verdict — Read a fixed point's fate off its slope	33
3.2	103_02_instalments — Late brakes and early brakes	36
4.1	104_01_rk4 — One step of RK4	44
4.2	104_02_drift — Measure the drift	45
5.1	105_01_stability — Where the steady state lets go	58
5.2	105_02_period — A cycle detector of your own	62
6.1	106_01_apart — When do two runs part company?	70
6.2	106_02_digits — Digits known, steps predicted	71
7.1	107_01_exponent — The exponent, from an orbit	83
7.2	107_02_horizon — How far ahead, and at what price	86
8.1	108_01_superstable — Find a landmark yourself	97
8.2	108_02_ratio — Feigenbaum's ratio, and where the doublings end	100

9.1	109_01_rates — Lorenz’s rules, in your own hand	107
9.2	109_02_wings — Two numbers the start does not decide	114
10.1	110_01_knead — A strip of dough	120
10.2	110_02_henon — One step, and its area	126
11.1	111_01_cantor — The Cantor set by rule	137
11.2	111_02_boxes — Count the boxes yourself	139
12.1	112_01_escape — Escape time without complex numbers . . .	148
12.2	112_02_convert — From r to c and back	156
13.1	113_01_energy — The number that must not move	161
13.2	113_02_apart — When do two releases part?	164
14.1	114_01_chance — A chance of rain, and keeping score	176
14.2	114_02_saturation — When does the ensemble stop spreading?	178
15.1	115_01_return — Curve or cloud?	187
15.2	115_02_forecast — Ask several analogues	191
16.1	116_01_collapse — When does the doubling map die?	199
16.2	116_02_single — Single against double	201
16.3	116_03_loops — Every orbit ends in a loop	206
17.1	117_01_controller — A controller of your own	215
17.2	117_02_sync — How long does the copy take?	217
18.1	118_01_horizon — The horizon calculator	224
18.2	118_02_guide — The field guide, on a model you write	228

Plot manifest

ch01-two-models — a formula and a stepping rule	6
ch01-gap-shrinks — an error that dies away	8
ch02-three-fates — three linear rules and their three fates . .	15
ch02-two-accounts — linear against exponential growth	16
ch02-cobweb — a cobweb on a line	23
ch03-s-curve — exponential against logistic growth	30
ch03-zoom — zooming in until a curve looks straight	32
ch03-overshoot — the overshoot, and the same law without delay	35
ch04-euler-rk4 — Euler spiralling out against RK4	45
ch04-pendulum-map — a pendulum’s phase portrait	47
ch04-settle — settling onto a point and onto a loop	48
ch05-four-series — four time series of the logistic map	56
ch05-four-cobwebs — four cobwebs on the logistic map	60
ch05-window — a cycle of three inside the chaos	62
ch05-two-starts — two nearby starts at one setting	63
ch06-peel — two runs peeling apart	67
ch06-gap-log — a gap that grows, then a ceiling	69
ch07-running-average — an average that settles	83
ch07-horizon-line — horizon against the error in the start . .	85
ch07-exponent-r — the exponent against r	88
ch08-diagram — the bifurcation diagram	92
ch08-ratios — the ratios for two rules converging	98

ch08-zoom — the period-three window, magnified	101
ch09-butterfly — the butterfly from two sides	108
ch09-apart — two runs parting company	111
ch09-histograms — histograms that ignore the start	115
ch10-attractor — the attractor and its basin	123
ch10-square-folds — a square stretched and folded	125
ch10-zooms — the attractor, zoomed twice	127
ch11-koch-stages — the Koch curve, round by round	133
ch11-three-fates — three fates of a length	137
ch11-box-counts — box counts on a log-log plot	140
ch12-mandelbrot — the Mandelbrot set and three zooms	150
ch12-julia — a Julia set in one piece and one in dust	152
ch12-real-axis — the real axis over the bifurcation diagram	154
ch13-two-releases — two releases parting	163
ch13-energy-error — the energy check	164
ch13-gentle-wild — calm and wild releases	166
ch14-plume — an ensemble plume fanning out	174
ch14-skill — spread and error against lead time	177
ch14-climate — climate statistics under two forcings	180
ch15-twins — two series with one histogram	185
ch15-return-maps — a curve and a cloud	186
ch15-forecast — forecast error against steps ahead	189
ch15-noisy — a curve thickening in noise	190
ch16-collapse — the doubling map dying on a computer	199
ch16-parting — two precisions parting company	201
ch16-statistics — statistics that survive a wrong path	205
ch17-control — control switched on mid-run	213
ch17-sync — a copy falling into step	216
ch18-digits — horizon against correct digits	222
ch18-odds — a forecast turning into odds	225

Diagram manifest

ch01-model-loop.mmd — state, rule, prediction, check	9
ch02-feed-back.mmd — the loop of iteration	13
ch03-crowding-loop.mmd — crowding as a negative feedback loop	28
ch04-step-loop.mmd — a rule of change, followed in steps	42
ch06-digits.mmd — digits in, steps out	71
ch07-three-steps.mmd — stretch, exponent, horizon	84
ch08-universal.mmd — any hump, one route, one ratio	99
ch09-feedback.mmd — three numbers in a loop	106
ch10-section.mmd — flow, section, map	128
ch11-box-recipe.mmd — cover, count, slope	138
ch12-two-questions.mmd — one rule, two sets	152
ch13-bodies.mmd — two bodies, three bodies, the planets	167
ch14-forecast-cycle.mmd — analysis, ensemble, forecast	175

ch16-shadow.mmd — a computed orbit and its shadow	205
ch17-wait-catch.mmd — wait, catch, hold	212
ch17-drive.mmd — one signal, from original to copy	215
ch18-field-guide.mmd — the field guide in three moves	228

Index

- analysis (weather), 171
- area, 123
- attractor, 47, 112, 122
 - statistics, 224
 - statistics of, 113
- bacteria, 26
- basin of attraction, 122
- bifurcation diagram, 91
- binary digits, 70
- binary numbers, 196
- bisection, 93
- box counting, 137
- butterfly effect, 66, 226
- Cantor set, 135
- chaos, 54
 - definition, 63
 - telling from noise, 183
 - why it matters, 228
- climate, 179
- climate projection, 179
- climatology, 177
- clockwork universe, 9
- coastline paradox, 131
- cobweb diagram, 22, 59
- complex numbers, 144
 - multiplying, 144
- control of chaos, 209, 227
 - cost, 212
 - in the laboratory, 217
 - nudge, 210
- convection, 105
- cycle, 36, 60
- delay, 33
- determinant, 123
- determinism, 75
- deterministic, 63
- differential equation, 40
- dimension
 - box-counting, 137
 - from copies, 135
- dissipation, 123
- double pendulum, 159
 - state, 159
- double precision, 196
- doubling, 26
- doubling map, 70, 198
- doubling time, 16
- energy, 161
- ensemble forecast, 170, 224
 - spread, 171
- epidemic, 191
- error
 - in the start, 7
- escape time, 146
- Euler's method, 41
- exact arithmetic, 202
- exponential growth, 16
 - of an error, 67
- feedback, 26
 - negative, 27
 - positive, 27
- Feigenbaum's constant, 91
- Feigenbaum, Mitchell, 95
- fixed point, 18, 31, 56
 - stability, 56
 - stable, 20
 - unstable, 20, 209
- floating point, 196
- forecast, 220
- forecast skill, 177
- fractal, 126, 131
 - in nature, 141
- growth
 - exponential, 26
- half-life, 16
- Halley's comet, 9
- Hénon map, 120

- Hénon, Michel, 120
- Hyperion, 167
- Hénon attractor
 - dimension, 139
- islands of regular motion, 165
- iteration, 12
- Julia set, 152
- KAM theorem, 165
- Kepler's third law, 50
- kneading, 118
- Koch curve, 133
- laboratory, 3
- Laplace's demon, 9, 229
- Laskar, Jacques, 167
- Li–Yorke theorem, 100
- limit cycle, 47
- linear congruential generator, 205
- linear rule, 14
- logarithm, 16
 - natural, 80
- logistic growth, 28
- logistic map, 54, 66
 - and the Mandelbrot set, 153
 - four behaviours, 55
 - on a computer, 200
 - rule, 54
- Lorenz system, 105
 - butterfly, 107
- Lorenz, Edward, 73, 105
- Lorenz-96 model, 170
- Lyapunov exponent, 78, 221
 - across r , 87
 - of a flow, 110
 - of the logistic map, 81
- Lyapunov time, 84
- Mandelbrot set, 144
 - drawing, 149
- Mandelbrot, Benoit, 131
- markets, 193
- May, Robert, 63
- method of analogues, 188
- model, 2
 - error, 7
 - prediction, 2
 - rule, 2
 - state, 2
 - stepping, 4
- motion, 40
- nearest neighbour, 188
- Newton's law of cooling, 4
- noise, 183, 226
 - measurement, 190
- nonlinear, 30
- orbit, 12
- oscillation
 - damped, 33
- Ott, Grebogi and Yorke, 210
- overshoot, 33
- Pecora and Carroll, 215
- pendulum, 45
- period, 60
- period doubling, 93
- periodic window, 62, 100
- phase portrait, 45
- phase space, 45
- Poincaré section, 128
- Poincaré, Henri, 167
- Poincaré–Bendixson theorem, 47
- population dynamics, 191
- predictability, 75
- prediction
 - field guide, 220, 227
 - how long it lasts, 72
- prediction horizon, 78, 84, 221
- probability forecast, 175
- Python, 3
- random numbers, 205
- reliability, 175
- reproducibility, 200
- return map, 186
- Richardson, Lewis Fry, 131
- RK4, 43, 107
- rounding, 196
- rule
 - linear, 14
 - of the next step, 12
- rule of change, 40
- Runge–Kutta method, 43

- S-curve, 28
- secure communication, 217
- self-similarity, 133
- sensitive dependence, 110
 - double pendulum, 162
 - test for, 220
- sensitive dependence on initial conditions, 66
- shadowing, 203
- simulation
 - energy check, 161
- sine map, 97
- SIR model, 191
- slope, 21, 31
- solar system, 159
- stability, 20, 31
- state, 12
- strange attractor, 118
 - as fractal, 139
 - layers, 126
- stretch, 78
- stretch and fold, 118
- superposition, 30
- superstable cycle, 95
- surrogate data, 183
- synchronisation, 209, 227
 - drive and response, 215
- tent map, 118
- three-body problem, 167
- time step, 41
- transient, 91
- two-body problem, 50
- universality, 97
- velocity, 40
- Verhulst, Pierre-François, 28
- weather forecast, 170