8 Chaos

Why Study Chaos?

- **It reveals the structure hidden behind apparent randomness**. Chaos theory shows that systems that look unpredictable can still follow precise deterministic laws. This helps distinguish true randomness from complex but regulated behaviour.

- **It highlights how systems are sensitivite to initial conditions**. Small uncertainties can grow exponentially in chaotic systems. Understanding this gives realistic limits on prediction in weather, climate, economics, and many physical processes.

- **It provides tools for modelling real-world complexity**. Many natural and engineered systems, such as turbulent fluids, population dynamics, electrical circuits, and chemical reactions, are chaotic. Their analysis utilizes ideas from chaos theory, including attractors, Lyapunov exponents, and bifurcations.

- **It improves forecasting and control**. Recognising when a system is near a bifurcation into chaos informa how we design stabilising systems, feedback controls, and simulations.

- **It unifies behaviour across disciplines**. Chaos reveals patterns, such as period-doubling and fractal structures, that recur in fields as different as biology, physics, engineering, and finance.

8.1 Introduction

Chaos occurs when a deterministic system behaves unpredictably, in a non-repeating and apparently random fashion. This is not due to some hidden external noise or randomness in the system, but a consequence of simple, deterministic actions.

One source of confusion is that this mathematical meaning doesn't really match chaos' common usage as a synonym for an absence of order. Another issue is that although the term originated in the mid-1970s, chaos is a part of the much larger nonlinear dynamic systems field, which dates back at least to Henri Poincaré in the 1890s.

A dynamic system is chaotic if it possesses the following properties:

(1) The system depends on a deterministic function (or functions) which, given the same input, always returns the same result.

(2) The results are aperiodic, with the function never entering a cycle.

(3) The results are bounded by upper and lower limits, so the system's aperiodicity is not just a consequence of being able to generate infinite sequences.

(4) The system exhibits *sensitive dependence on initial conditions* (SDIC), in that a very small change in its initial state will lead to very large changes in its outputs in a short time.

We'll look at two famous examples: the logistic map and the Lorenz model. For more details, Strogatz offers a great introduction [**Str15**].

8.2 The Logistic Map

The logistic map, a discrete-time version of the logistic equation for population growth, is one of the simplest nonlinear equations that exhibits chaos:

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

x_n represents the population in the n-th generation, which can range between $0 \le x \le 1$. r is the growth rate, which varies in the interval $[0, 4]$.

The generation of x_n from some initial population x_0 can be represented using a *cobweb* diagram, such as the one on the left of Fig. 8.1. The conversion of the map's output to become its next input (e.g. $x_0 \to x_1 \to x_2 \cdots$) is modeled by 'bouncing' the map's output off the line $y = x$.

Fig. 8.1 shows that in the $r = 2.5$ case the population converges to a steady state (a fixed point) where $x = rx(1 - x)$.

Another useful view of the map's execution is a time series plot of x_n against n, shown on the right of Fig. 8.1. x_n clearly converges upon a single value (0.6), which can be confirmed by solving $x = 2.5x(1 - x)$.

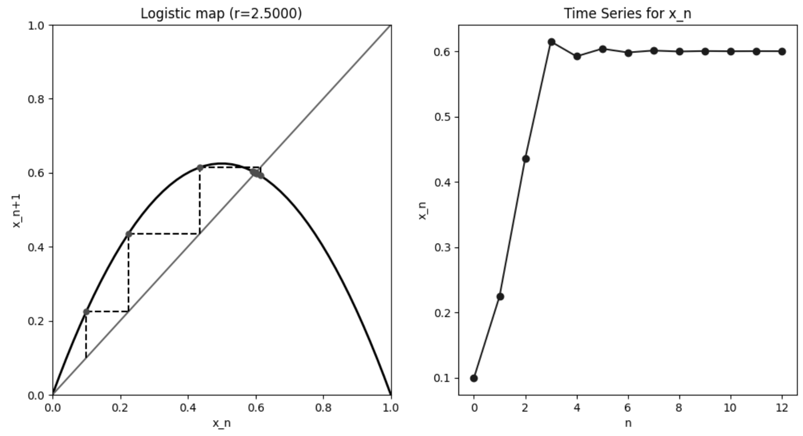


Figure 8.1. Cobweb and time series for $r = 2.5$

It's instructive to see what happens to the map by running logCobweb.py with different r values, but a more interactive approach offered by animLogCobweb.py uses a matplotlib slider (Fig. 8.2).

For small growth rates of $r < 1$, the population always goes extinct (see Fig. 8.2), while for $1 < r < 3$ the population eventually reaches a nonzero steady state (as in Fig. 8.1).

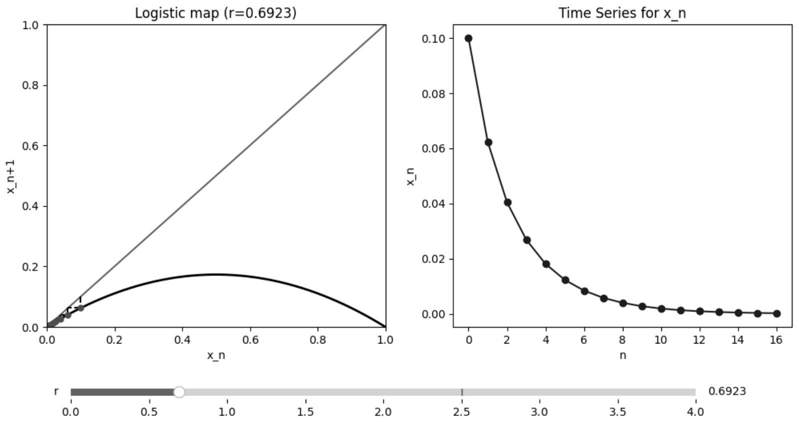


Figure 8.2. Cobweb and time series for $r = 0.6923$

Things become more interesting when $r \geq 3$. For $r = 3.3$, the population builds up but, instead of converging, gradually begins to oscillate between two population values (Fig. 8.3). This is called a period-2 cycle.

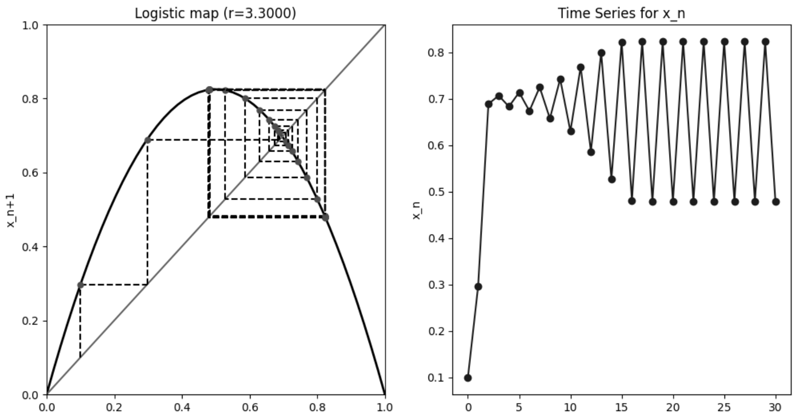


Figure 8.3. Cobweb and time series for $r = 3.3$

For a larger value, such as $r = 3.5$, the population starts repeating every four generations – a period-4 cycle (Fig. 8.4).

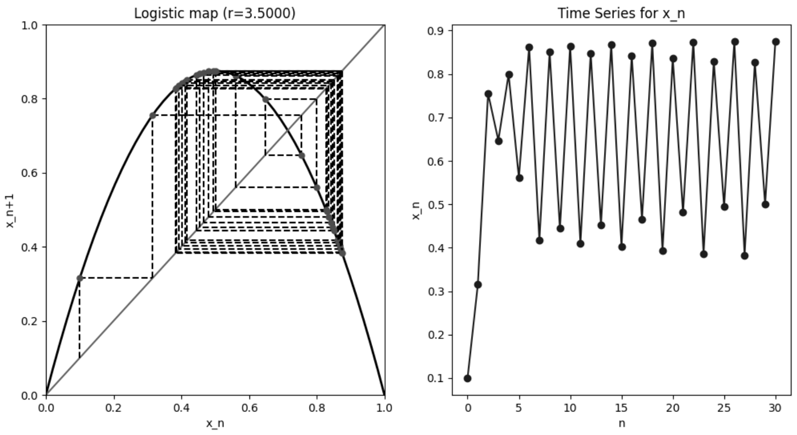


Figure 8.4. Cobweb and time series for $r = 3.5$

Doublings to cycles of period 8, 16, 32,... occur as r increases. Experimentation with `animLogCobweb.py` produces the results summarized in Table 8.1.

r	Period-i
$r_1 = 3$	2
$r_2 = 3.44949...\ (1+\sqrt{6})$	4
$r_3 = 3.54409...$	8
$r_4 = 3.5644...$	16
$r_5 = 3.568759...$	32

Table 8.1. Period doubling for r values

Splittings (*bifurcations*) of the population's cycles occur more and more quickly, with chaos seemingly breaking out at $r \approx 3.56995$ (e.g. see $r = 3.9$ in Fig. 8.5). However, the actual behavior is more subtle, requiring a different visualization.

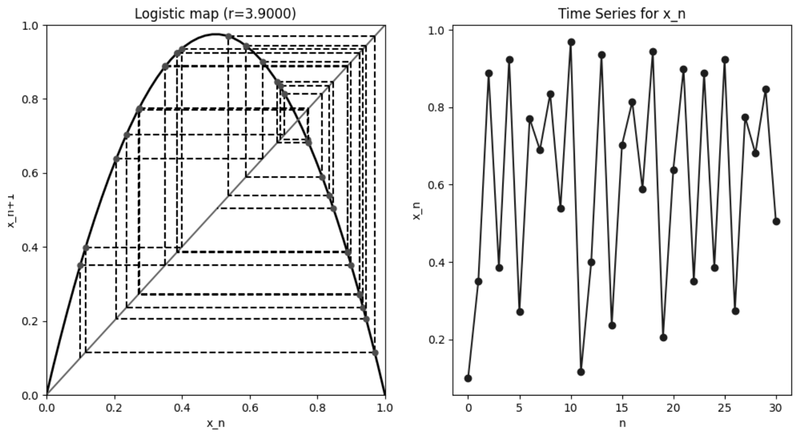


Figure 8.5. Cobweb and time series for $r = 3.9$

We switch to an orbit diagram (also called a *bifurcation diagram*) which plots values of r against the x_n's produced. Fig. 8.6 shows that the map splits into a period-2 cycle at $r = 3$ by dividing into two branches. As r increases, these branches multiply, until the map becomes chaotic.

Fig. 8.6 was produced using Listing 8.1 (bifurcation.py).

```
NUM_RS = 10000   # more r detail when zoomed-in
NUM_ITERS = 500

def logistic(r, x):
        return r * x * (1 - x)

def logistics(r):
  xs = [0.1]
```

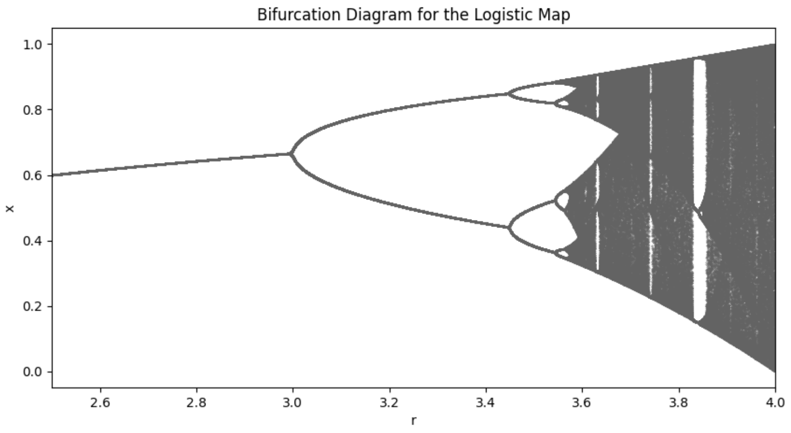


Figure 8.6. Bifurcation diagram for the logistic map

```
  for i in range(NUM_ITERS-1):
    xs.append(logistic(r,xs[i]))
  return xs[NUM_ITERS-100:]     #  last 100 iterations

rs = linspace(2.0, 4.0, NUM_RS)  # r between 2 and 4
xVals = []
rVals = []
for r in rs:
  xs = logistics(r)
  xVals.append(xs)
  rVals.append([r] * 100)
plt.figure(figsize=(10, 5))
plt.scatter(rVals, xVals, s=0.1)
plt.xlabel('r')
plt.ylabel('x')
plt.xlim(2.5, 4)
plt.title("Bifurcation Diagram for the Logistic Map")
plt.show()
```

Listing 8.1. Implementing bifurcation

The logistics() function generates a series of x values for a specific r (also called an *orbit*), but discards the first 100 since it takes some time for the map to settle down to a steady behavior. Each orbit is plotted as a points on the $x = r$ line, with r ranging between 2 and 4 in steps of 0.0001. This level of detail is necessary since matplotlib allows the graph to be panned and zoomed in upon, as in Fig. 8.7.

The enlarged bifurcation diagram in Fig. 8.7 reveals an intriguing mixture of order and chaos, with periodic empty spaces (called *windows*) interspersed

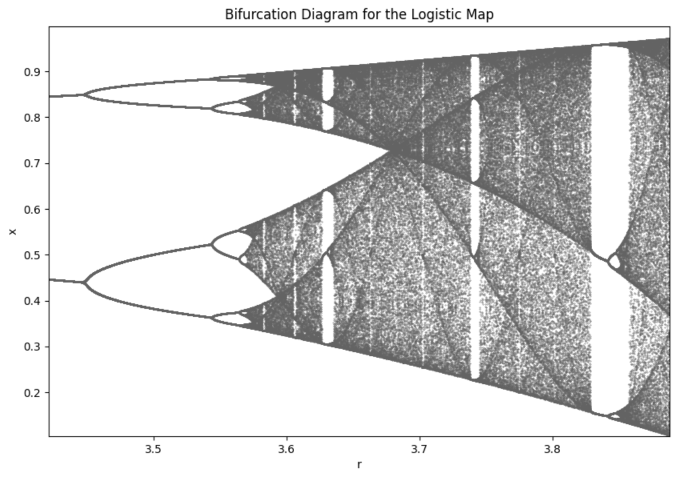


Figure 8.7. Bifurcation diagram between 3.4 and 3.9

between chaotic clouds of dots. It's also possible to discern elements of self-similarity, as in the lower-left corner of Fig. 8.7, enlarged in Fig. 8.8, which is nearly identical to the entire diagram.

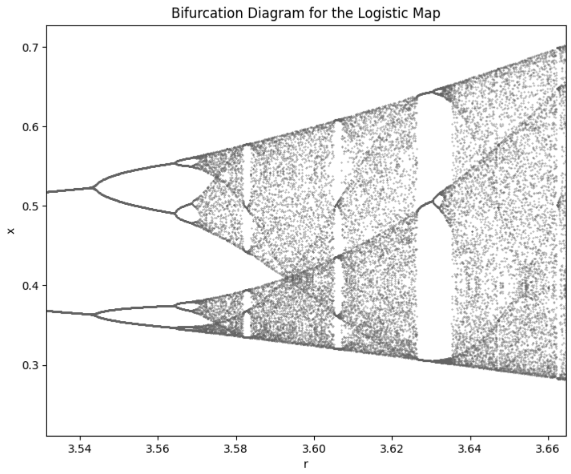


Figure 8.8. Bifurcation diagram between 3.54 and 3.66

8.3 Stability and Period Doubling

Fixed points (x_f) for the logistic map occur when $x_f = rx_f(1 - x_f)$. Hence, $x_f = 0$ and $1 - \frac{1}{r}$.

A fixed point is *stable* when mapping a x_n that's close to x_f produces a new point x_{n+1} that's even nearer. An unstable fixed point is one where the new point is further away than the original. This can be judged by looking at the map's gradient at the fixed point: x_f is stable if $|f'(x_f)| < 1$. The logistic map derivative is $f'(x) = r(1-2x)$. So for $x_f = 0$, $f'(x_f) = r$, and for $x_f = 1-1/r$, $f'(x_f) = 2-r$. This means that $x_f = 0$ is stable when $r < 1$, and $x_f = 1 - 1/r$ is stable when $1 < r < 3$.

Another concept is fixed point *superstability*. This is a fixed-point x_f where $|f'(x_f)| = 0$, causing nearby points to converge upon it at the fastest rate.

One issue is how to choose a r value to represent a given period in the logistic map. The approach we used in Table 8.1 was to list the first r that exhibited that behavior. In Fig. 8.9, these are labeled r_0, r_1, and r_2. However, it's more useful to choose an r where its sequence of x values contain a superstable fixed point. For the logistic map, where $f'(x) = r(1 - 2x)$, $|f'(x)| = 0$ occurs when x = 0.5.

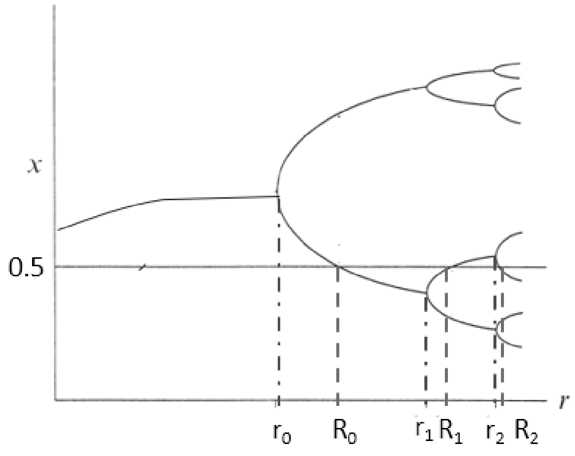


Figure 8.9. Choosing r values for period doubling

We'll denote these superstable sequences of period length 2^k using R_k, and R_0, R_1, and R_2 are shown on Fig. 8.9. From now on, when we need to compare r values of period 1, 2, 4,... We'll use these R values.

Fig. 8.10 shows the logistic map when $r = R_1 = 3.2361$. The sequence has a period length 2, which includes 0.5 as one of its fixed points.

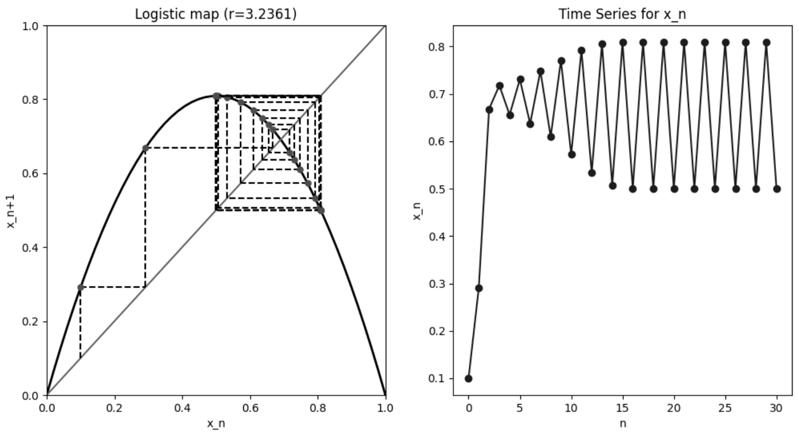


Figure 8.10. A superstable fixed point (R_1) at $r = 3.2361$

R_1 can be calculated exactly by solving $f(f(0.5)) = 0.5$ with $r = R_1$ and $x = 0.5$. Note that we are using two applications of $f()$ since the fixed point is in a sequence of period length 2.

$$\begin{aligned}
f(f(0.5)) &= 0.5 \\
f(R_1 0.5(1-0.5)) &= \\
f(R_1/4) &= \\
R_1(R_1/4)(1-R_1/4) &= \\
R_1^2/4 - R_1^3/16 &= 0.5 \\
R_1^3 - 4R_1^2 + 8 &= 0 \\
(R_1-2)(R_1^2 - 2R_1 - 4)) &= 0
\end{aligned}$$

So $R_1 = 2$ or $1 \pm \sqrt{5}$, of which only $R_1 = 1 + \sqrt{5} \approx 3.2361$ is applicable here. Note that R_1 is related to the golden ratio ϕ, with $R_1 = 2\phi$.

$f(f(x))$ – often abbreviated as $f^{(2)}(x)$ – has two fixed points at $x_0 = 0.5$ and $x_1 = 0.809...$ (see Fig. 8.10). If r is increased, then these two fixed points will become unstable at the same time, and split into a period length of 4. This can be confirmed by applying the chain rule:

$$\frac{d}{dx} f(f(x))\Big|_{x=x_0} = f'(f(x))\Big|_{x=x_0} \cdot f'(x)\Big|_{x=x_0}$$

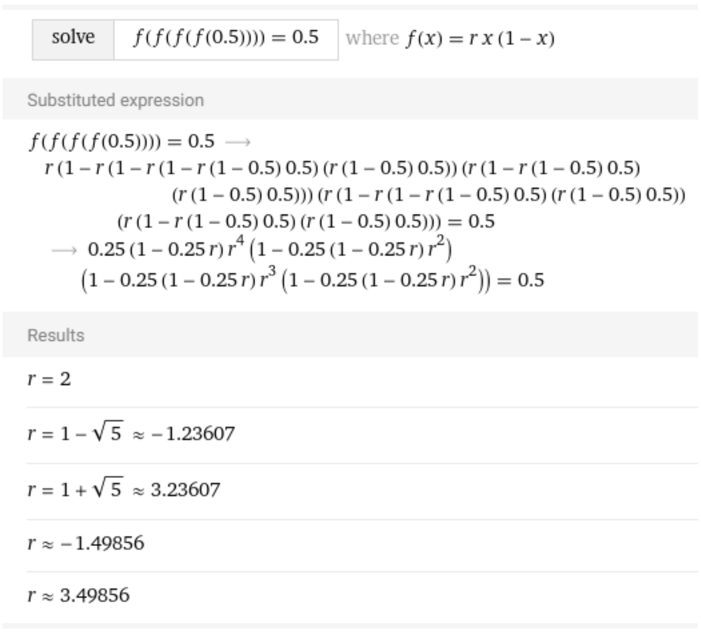
or, with $f(x_0) = x_1$,

$$\frac{d}{dx} f(f(x)) = f'(x_1) \cdot f'(x_0)$$

Note that the same result is obtained at $x = x_1$, by the symmetry of the final term. So if x_0 becomes unstable because $|f^{(2)'}(x_0)| > 1$, so does x_1 at the same r.

Now $f^{(2)}(f^{(2)}(x)) = f^{(4)}(x) = f(f(f(f(x))))$ will have four fixed points. For $r = R_2 = 3.49856...$, they are $x_0 = 0.5$, $x_1 = 0.874...$, $x_2 = 0.383...$, and $x_4 = 0.0.827....$

We checked the answer by running WolframAlpha (`https://www.wolframalpha.com`), as shown in Fig. 8.11.



solve $f(f(f(f(0.5)))) = 0.5$ where $f(x) = r\,x\,(1-x)$

$$f(f(f(f(0.5)))) = 0.5 \longrightarrow r(1-r(1-r(1-r(1-0.5)\,0.5)(r(1-0.5)\,0.5))(r(1-r(1-0.5)\,0.5)(r(1-0.5)\,0.5)))(r(1-r(1-r(1-0.5)\,0.5)(r(1-0.5)\,0.5))(r(1-r(1-0.5)\,0.5)(r(1-0.5)\,0.5))) = 0.5$$

$$\longrightarrow 0.25\,(1-0.25\,r)\,r^4\left(1-0.25\,(1-0.25\,r)\,r^2\right)\left(1-0.25\,(1-0.25\,r)\,r^3\left(1-0.25\,(1-0.25\,r)\,r^2\right)\right) = 0.5$$

$r = 2$

$r = 1-\sqrt{5} \approx -1.23607$

$r = 1+\sqrt{5} \approx 3.23607$

$r \approx -1.49856$

$r \approx 3.49856$

Figure 8.11. Calculating R_2 in the logistic map

8.4 The Lyapunov Exponent

One useful indicator of chaotic behavior is the Lyapunov exponent, which measures how an infinitesimally small distance between two close states grows exponentially quickly over time.

Given some initial condition x_0, consider a nearby point $x_0 + \delta_0$. Let δ_n be the separation after n iterations of the map such that $|\delta_n| = |\delta_0|e^{n\lambda}$, where λ is the Lyapunov exponent. A more computationally useful formula can be derived by utilizing

$$\delta_n = f^{(n)}(x_0 + \delta_0) - f^{(n)}(x_0)$$

Taking logs of the exponent:

$$\begin{aligned}
\lambda &\approx \frac{1}{n}\ln\left|\frac{\delta_n}{\delta_0}\right| \\
&= \frac{1}{n}\ln\left|\frac{f^{(n)}(x_0+\delta_0)-f^{(n)}(x_0)}{\delta_0}\right| \\
&= \frac{1}{n}\ln\left|f^{(n)'}(x_0)\right|, \quad \text{when } \delta_0 \to 0
\end{aligned}$$

The term inside the logarithm can be expanded by generalizing the chain rule technique used earlier so that:

$$f^{(n)'}(x_0) = \prod_{i=0}^{n-1} f'(x_i)$$

So:

$$\begin{aligned}
\lambda &\approx \tfrac{1}{n}\ln\left|\textstyle\prod_{i=0}^{n-1} f'(x_i)\right| \\
&= \tfrac{1}{n}\textstyle\sum_{i=0}^{n-1} |f'(x_i)|
\end{aligned}$$

λ will be negative for stable fixed points and cycles, and positive for chaotic behavior.

logLyapunov.py extends the code in bifurcation.py to generate a bifurcation diagram *and* a Lyapunov exponent curve, as in Fig. 8.12.

λ is negative for $r < 3.57$, but approaches zero at the period-doubling bifurcations. Chaos begins near $r \approx 3.57$, when λ first becomes positive, and increases for $r > 3.57$, except for the dips caused by the periodic windows.

The main change to logLyapunov.py compared to bifurcation.py is the calculation of the exponent inside the logistics() function in Listing 8.2 (logLyapunov.py).

```
def logistics(r):
  xs = [0.3]
  lyapunov = 0
  for i in range(NUM_ITERS-1):
    xs.append( logistic(r,xs[i]))
    lyapunov += math.log(abs(r - 2*r*xs[-1]))
        # sum of log( | f'(x) | )
  return xs[NUM_ITERS-100:], lyapunov/NUM_ITERS
      #  last 100 iterations
```

Listing 8.2. logistics() in logLyapunov.py

8.5 The Feigenbaum Constants

The Feigenbaum constants, δ and α, are ratios present in every bifurcation diagram of a nonlinear map with a single quadratic maximum (such as the logistic map).

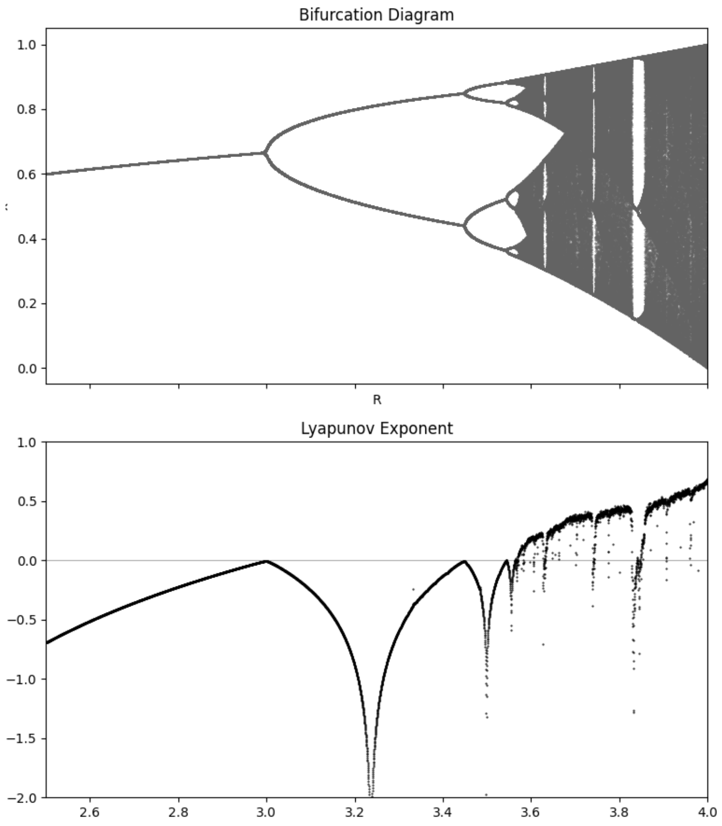


Figure 8.12. Bifurcation diagram and lynapunov exponent for the logistic map

Mitchell Feigenbaum began to study period-doubling in the mid 1970s, focusing on r_n, the value of r when a 2^n-cycle first appears. He noticed that r_n converges geometrically, with the distance between successive transitions, δ, defined as the limit:

$$\delta = \lim_{n\to\infty} \frac{r_n - r_{n-1}}{r_{n+1} - r_n} = 4.6692016091029...$$

The first few transitions are shown in Fig. 8.13.

Calculating r_i to a high level of accuracy is difficult, but a much simpler alternative is to determine the superstable values R_i. These can be used in the same way to calculate δ:

$$\delta = \lim_{n\to\infty} \frac{R_n - R_{n-1}}{R_{n+1} - R_n}$$

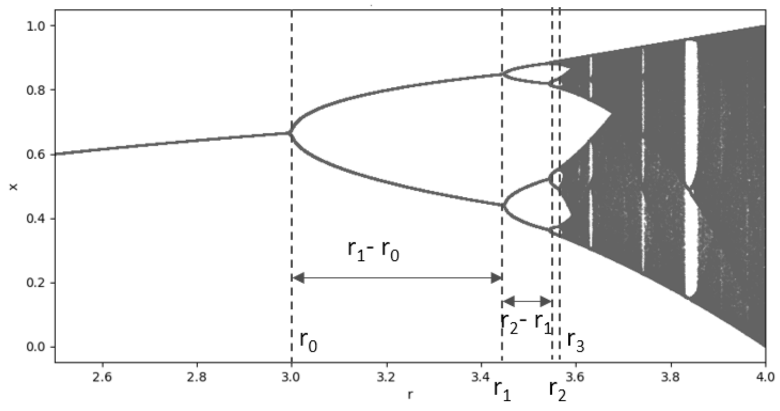


Figure 8.13. δ convergence

Feigenbaum's other constant, α, is the limit of the ratios of the distance a_i and its successor at a_{i+1}:

$$\alpha = \lim_{n\to\infty} \frac{a_n}{a_{n+1}} = 2.502907876...$$

a_i is the distance between the two branches of bifurcation diagram when R_i is detected.

Fig. 8.14 shows the placement of the first few R_i and a_i values.

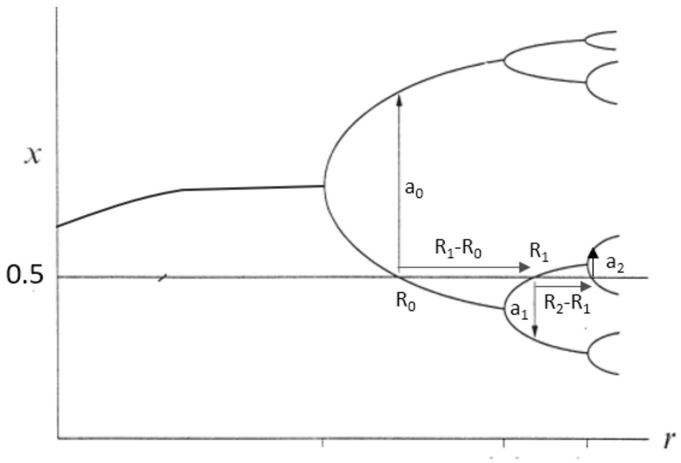


Figure 8.14. Calculating Feigenbaum's constants

Keith Briggs developed an algorithm for calculating the Feigenbaum's constants [**Bri89**], which is implemented in feigenbaum.py. His approach uses a simple quadratic equation ($x_{n+1} = a - x_n^2$). Its derivation is $f'(x) = 2x$ which means that its R_i values all occur at $x = 0$, which makes it easy to find them using Newton's root finding method. The corresponding map is:

$$\begin{aligned} b_k(a) &= a - [b_{k-1}(a)]^2, \quad k = 1, 2, 3, \ldots \\ b_0(a) &= 0 \end{aligned}$$

Its R_i values will occur at the zeros of b_{2^i}. These are found by starting with an approximation a_i, then using Newton's method to zero in on the correct value by looking for a root. This approach is defined using six equations:

$$\begin{aligned} a_i^0 &= a_{i-1} + \frac{a_{i-1} - a_{i-2}}{\delta_{i-1}}, \quad i = 2, 3, 4, \ldots \\ a_i^{j+1} &= a_i^j - \frac{b_{2^i}(a_i^j)}{b'_{2^i}(a_i^j)}, \quad j = 0, 1, 2, \ldots \\ b'_k(a) &= 1 - 2b'_{k-1}(a)b_{k-1}(a), \quad k = 1, 2, 3, \ldots \\ a_i &= \lim_{j\to\infty} a_i^j \\ \delta_i &= \frac{a_{i-1} - a_{i-2}}{a_i - a_{i-1}} \\ \delta &= \lim_{i\to\infty} \delta_i \end{aligned}$$

The first equation produces an initial approximation to the next superstable a value, which is refined using Newton's method using the derivative db/da of b. a_i^j will converge on the i-th root of b_{2^i}.

Listing 8.3 (feigenbaum.py) is an almost direct translation of these equations using Python's decimal to ensure extended precision.

```
NUM_ITERS = 20
A_ITERS = 10

def calcAIter(n, a):
  # perform Briggs equations 7, 8, 9 (first 3)
  for _ in range(A_ITERS):
    b = D(0)
    bp = D(0)
    for _ in range(2**n):
      bp = 1 - 2*bp*b    # represents bPrime()
      b = a - b**2       # represents b(); equ 5
    a = a - (b / bp)
  return a, bp


d = D('3.2')     # will become the feigenbaum constant
a2 = D(0)  # the ordering is [ a2, a1, a, ...]
```

```
a1 = D(1)
bpOld = D(1)
# perform Briggs equations 10, 11, 12 (last three)
print(" i      delta          alpha")
for i in range(2, NUM_ITERS+1):
  a, bp = calcAIter(i, a1 + ((a1 - a2) / d))
  d = (a1 - a2)/(a - a1)
  alpha = d*bpOld/bp
  bpOld = bp
  print(f"{i:2d}  {d:.10f}   {alpha:.10f}")
  # prepare for next iteration; move elems backwards
  a2 = a1
  a1 = a
```

Listing 8.3. Implementing Feigenbaum

The output:

```
> python feigenbaum.py
 i      delta          alpha
 2  3.2185114220   1.8560344182
 3  4.3856775986   -2.3948809906
 4  4.6009492765   -2.4753300767
 5  4.6551304954   -2.4974031887
          :   # more lines, not shown
18  4.6692016091   -2.5029078751
19  4.6692016091   -2.5029078751
20  4.6692016091   -2.5029078751
```

The rate of convergence is roughly linear, with one significant decimal digit of improvement every two iterations.

The calculation of α is based on the fact that:

$$\lim_{i\to\infty} \frac{b'_{i+1}(a_{i+1})}{b'_i(a_i)} = \delta/\alpha$$

This is derived from the connection between the b^i derivative and the lengths $R_i - R_{i-1}$ and a_i in Fig. 8.15.

We know

$$b_i \approx \frac{a_i}{R_i - R_{i-1}} \quad \text{and} \quad b_{i+1} \approx \frac{a_{i+1}}{R_{i+1} - R_i}$$

So

$$\begin{aligned} \frac{b_{i+1}}{b_i} &\approx \frac{a_{i+1}}{R_{i+1} - R_i} \cdot \frac{R_i - R_{i-1}}{a_i} \\ &= \frac{R_i - R_{i-1}}{R_{i+1} - R_i} \cdot \frac{a_{i+1}}{a_i} \\ &= \frac{\delta_i}{\alpha_i} \end{aligned}$$

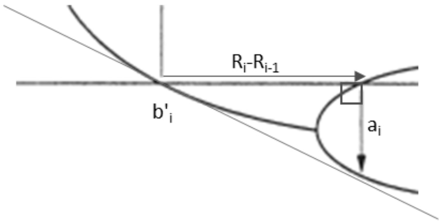


Figure 8.15. Relating b_i to the Feigenbaum constants

8.6 Self-similarity Again

We've already seen that the bifurcation diagram contains self-similarity; another example utilizes Feigenbaum's α constant = -2.5029....

R_0 ($r = 2$) is the superstable value for r in $f(x) = rx(1 - x)$, while R_1 ($r = 3.236$) is the superstable value for r in $f(f(x))$. These two equations are plotted in Fig. 8.16.

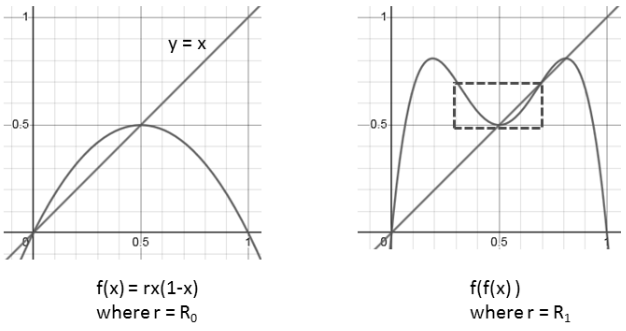
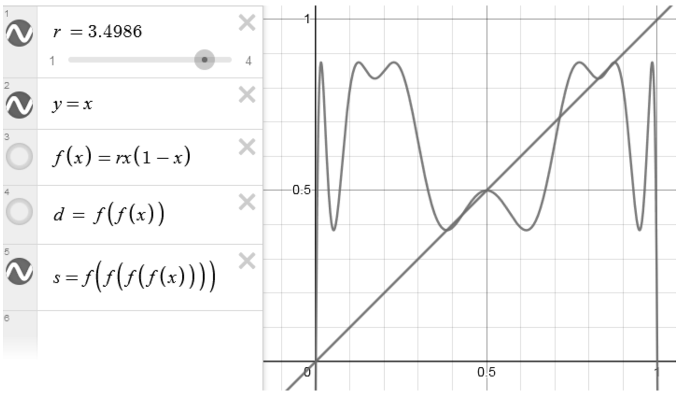


Figure 8.16. Self-similarity for the logistic map, $f(x)$

The dotted box in the plot of $f(f(x))$ when $r = R_1$ outlines an area of the curve which is similar to the $f(x)$ plot except that it's scaled by about half and reflected. This transformation, called *renormalization*, can be obtained by employing Feigenbaum's α as a scaling factor ($x \rightarrow x/\alpha$). The negative sign inverts the curve.

This scaling can be seen in each curve defined by R_i. For example, for R_2 ($r = 3.4986$), the curve is $f^{(4)}(x) = f(f(f(f(x))))$ which is plotted using Desmos (`https://www.desmos.com/calculator`) in Fig. 8.17. The scaled version of $f(x)$ can be observed around the fixed point (i.e. when the curve touches the $y = x$ line).


$r = 3.4986$

$y = x$

$f(x) = rx(1-x)$

$d = f(f(x))$

$s = f(f(f(f(x))))$

Figure 8.17. Self-similarity for R_2 in $f^{(4)}(x)$

We've included the Desmos parameters in Fig. 8.17 so you can reproduce these graphs for yourself.

8.7 The Lorenz System

The Lorenz system is a simplified model of the motion of the atmosphere as it's heated from below and cooled from above. The air rises and falls along the opposite edges of long cylinders that rotate because of the temperature difference (as depicted in Fig. 8.18).

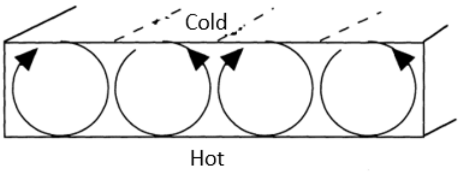


Figure 8.18. Convection in the Lorenz model

The temperature difference reduces as the air turns, causing the rotation to slow down and eventually stop. The difference will eventually build up again and the turning resumes, but now in the opposite direction. This pattern keeps repeating.

This behavior can be modeled as a system of three ordinary differential equations:

$$\begin{aligned} dx/dt &= \sigma(y-z) \\ dy/dt &= x(\rho-z)-y \\ dz/dt &= xy-\beta z \end{aligned}$$

x is the speed of the convective circulation with $x > 0$ representing a clockwise direction (and $x < 0$ being counter-clockwise). y is the temperature difference, and z the bottom-to-top linear temperature gradient. The constants σ, ρ, and β are system parameters, typically set to 10, 28, and 8/3.

lorenz.py (Listing 8.4) plots the model by treating the derivatives as difference equations and using Euler's forward method to step through time, calculating the partial derivatives at the current (x, y, z) in order to estimate the next coordinate.

```
DT = 0.01
N = 10000

def lorenz(xyz, s=10, p=28, b=8/3):
  """
  xyz : Point in 3D space as a tuple
  s, p, b : Parameters defining the Lorenz attractor
  Returns Lorenz attractor's partial derivatives at xyz
  """
  x, y, z = xyz
  x_dot = s*(y - x)
  y_dot = p*x - y - x*z
  z_dot = x*y - b*z
  return (x_dot, y_dot, z_dot)

xyz = [(0,0,0) for x in range(N+1)]
xyz[0] = (0, 1.0, 1.05)
for i in range(N):
  xyz[i+1] = tuple(map(lambda x,f: x+f*DT,
                              xyz[i], lorenz(xyz[i])))
              # xyz[i] + lorenz(xyz[i]) * DT
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(projection ='3d')
xs, ys, zs = zip(*xyz)
ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title("Lorenz Attractor")
plt.show()
```

Listing 8.4. Implementing Lorenz

Fig. 8.19 is perhaps the most famous image in chaos theory (although the bifurcation diagram (Fig. 8.6) for the logistic map is a close contender).

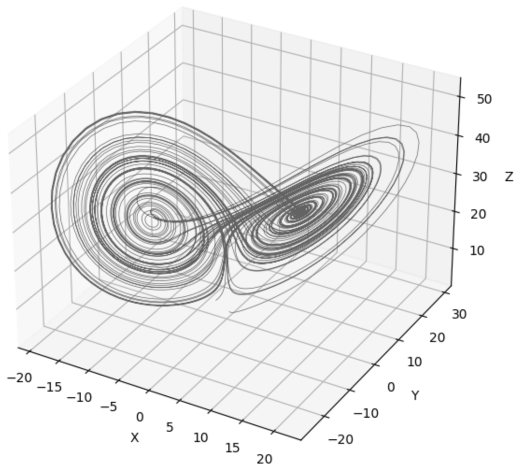


Figure 8.19. Plot of the Lorenz model

Lorenz soon noted that changing his initial conditions by even a tiny fraction would rapidly lead to totally different behavior – the SDIC hallmark of chaotic systems. This property is shown in Fig. 8.20 (lorenzSDIC.py) which uses the same lorenz() function as above but plots two curves, one starting at (0, 1.0, 1.05), the other at (0.001, 1.0, 1.05). Their x values are also charted separately to show that although the curves begin in a similar fashion, they soon diverge.

There's a strong resemblance between the Lorenz model and a butterfly's wings, and it's occasionally thought that this is the reason why SDIC is sometimes known as the butterfly effect. This isn't the case. Lorenz's original work didn't include a 3D plot, and he originally used a seagull, not a butterfly, as an example of how a small perturbation could lead to large changes in the weather system. Hilborn discusses the origins of the butterfly metaphor in his short paper "Sea gulls, butterflies, and grasshoppers", available online at `https://physics.csuchico.edu/ayars/427/handouts/AJP000425.pdf`.

Another variant of the lorenz() code appears in lorenzAnim.py which utilizes matplotlib's animation support to draw the curve incrementally. This makes its dynamic aspects much more apparent, including its clockwise and counter-clockwise rotations around the two centers, and the variations in speed. Near a center, the curve slows down, almost to a stop, then speeds up as it moves further away. The two centers are at $(6\sqrt{2}, 6\sqrt{2}, 27)$ and $(-6\sqrt{2}, -6\sqrt{2}, 27)$

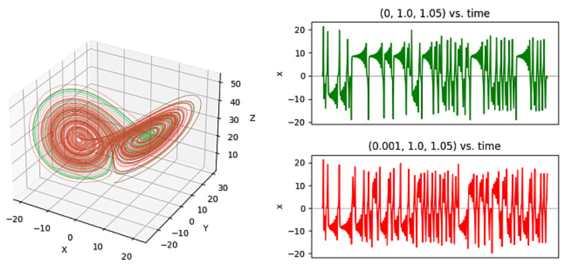


Figure 8.20. SDIC of the Lorenz model

Two snapshots of the animation are shown in Fig. 8.21. The large dot marks the current coordinate being drawn.

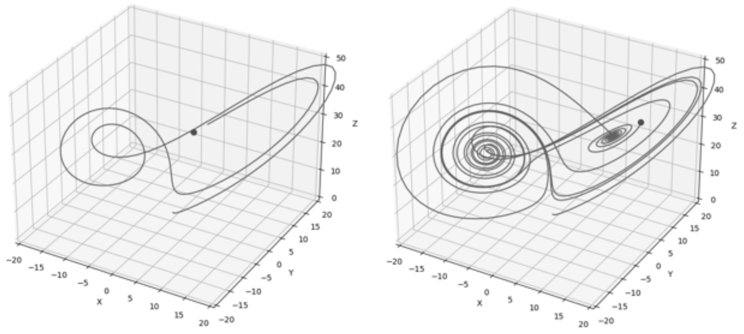


Figure 8.21. Animation of the Lorenz model

8.8 A Poincaré Section

One approach used by Lorenz to analyze the system was to plot Poincaré sections. This entails placing a constraint on one or more of the (x, y, z) coordinates to project the 3D curve onto a 2D surface or a 1D map.

Lorenz set $dZ/dt = 0$ which flattens the curve onto $z = b^{-1}xy$. Another way to understand this is that the derivative focuses only on the maximum and minimum z values. In fact, Lorenz generated a map of only the successive maximum values. He plotted the n-th maximum z value against the $(n + 1)$-th maximum, resulting in a graph similar to Fig. 8.22.

This captures the way that once a z value crosses a certain threshold, the coordinate jumps into an orbit around the other center, and the z component

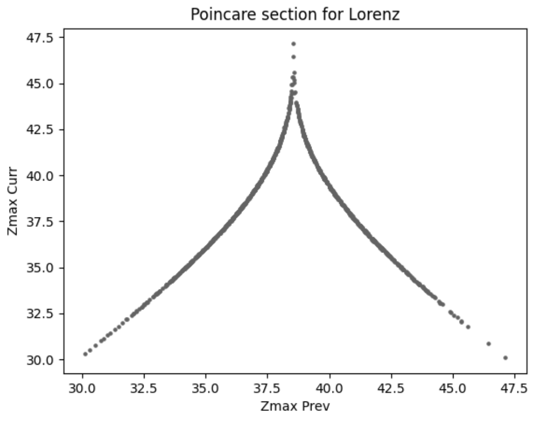


Figure 8.22. Poincaré section of the Lorenz model

shrinks. The value then starts increasing again until it reaches the threshold, then jumps back to the original orbit, and so on. Listing 8.5 (poincare.py) generates the Lorenz curve as before, but now looks for z points that are local maximums. These are stored in two lists which allow the n-th maximum z value to be plotted against the $(n+1)$-th value. Incidentally, the curve in Fig. 8.22 is sometimes called the *cusp map*.

```
currs = []
prevs = []
xyz = [(0,0,0) for x in range(N+1)]
xyz[0] = (0, 1.0, 1.05)

localMax = 0   # dummy start value
for i in range(N):
  xyz[i+1] = tuple(map(lambda x,f: x+f*DT,
                       xyz[i], lorenz(xyz[i])))
  if i > 100:   # let things settle down
    zOld = xyz[i-1][2]
    z = xyz[i][2]
    zNew = xyz[i+1][2]
    if zNew < z and zOld < z:
      prevMax = localMax
      localMax = z
      currs.append(localMax)
      prevs.append(prevMax)

plt.scatter(prevs[1:], currs[1:], s=5)  # s is point size
            # skip dummy value
plt.xlabel("Zmax Prev")
plt.ylabel("Zmax Curr")
plt.title("Poincare section for Lorenz")
plt.show()
```

Listing 8.5. Implementing Poincare

Lorenz used the fact that the absolute gradient of the map is > 1 at all points to argue that the underlying model is unstable.

As in previous examples, the curve is generated using Euler's forward step approximation, which in this case introduces some minor rounding errors on the right hand end. These disappear if we switch to a more accurate approximation such as 4th order Runge Kutta, but we decided not to complicate the code with that approach.

Exercises

(1) Modify logCobweb.py to explore the map $x_{n+1} = 1 + \frac{1}{2}\sin x_n$.

(2) Modify logCobweb.py to explore the map $x_{n+1} = 3x_n - x_n^3$. In particular consider the cobwebs starting at $x_0 = 1.9$ and 2.1.

(3) Modify bifurcation.py (Listing 8.1) to plot the following maps. Be sure to use a large enough range for both r and x to capture all the features of interest.

  (a) $x_{n+1} = r\cos x_n$
  (b) $x_{n+1} = r\tan x_n$
  (c) $x_{n+1} = rx_n - x_n^3$

(4) The constants used in the Lorenz model, σ, ρ, and β, are typically set to 10, 28, and 8/3. What happens when they are varied? Most researchers have left σ and β alone, and experimented with different values for ρ. Modify Listing 8.4, or one of its variants, to investigate the meaning of Fig. 8.23 which is taken from Section 9.5 of Strogatz [**Str15**].

It's also interesting to try out much larger values for ρ, such as 166.3 (sometimes termed 'intermittent chaos'), 212 (noisy periodicity) and the interval $145 < \rho < 166$ (period-doubling).

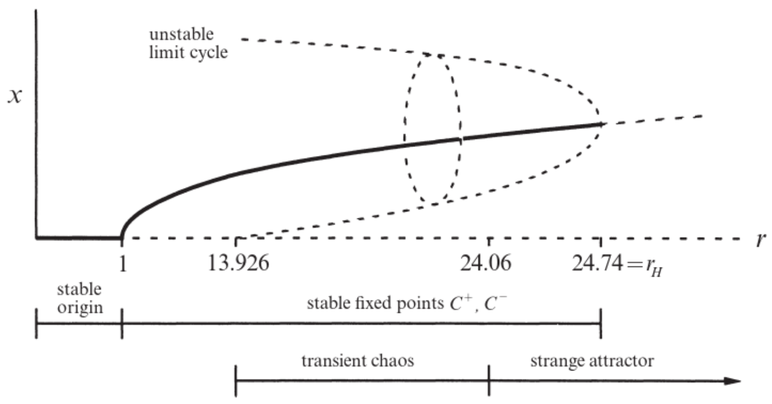


Figure 8.23. Varying ρ in the Lorenz model