

COURSE

How Epidemics Spread

10 Chapters

Philip Osztromok

Generated with Claude

HOW EPIDEMICS SPREAD

Contents

Chapter 1	Counting Cases	2
Chapter 2	Growth and Doubling	10
Chapter 3	The SIR Model	19
Chapter 4	Herd Immunity	29
Chapter 5	Adding Realism	38
Chapter 6	Chance and Fizzles	47
Chapter 7	Networks and Hubs	56
Chapter 8	Estimating R	66
Chapter 9	Controlling Spread	76
Chapter 10	Plan a Response	85

How Epidemics Spread 🏆

Chapter 1 · Counting Cases

Before you can ask how a disease spreads, you have to count it, and counting is harder than it sounds. Do you count people who feel ill, people who have been tested, or people who were infected last week and have already recovered? Do you count new cases or all current cases? Every model in this course, from the doubling of a first outbreak to the networks of later chapters, is fed by numbers that someone had to decide how to count. This chapter fixes the vocabulary and the habits: what an epidemic is, the difference between **incidence** and **prevalence**, how to draw an **epidemic curve**, and how to read one from real data. There is no calculus anywhere in this course. You need arithmetic, a little algebra, and a willingness to run small Python programs.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. Definitions come from the Wikipedia articles on epidemics, incidence and epidemic curves, fetched with a tool that often returns truncated pages or model-written summaries, so treat the wording as a paraphrase, not a quotation. The daily counts for the school outbreak were read from the data file of the `outbreaks` R package, which says that the original report gave them only as a figure, so exact numbers differ between sources. Numbers labelled *assumed* or *illustrative* are invented to show a method and are not measurements of any real disease.

What Is an Epidemic?

The Wikipedia article on epidemics defines an epidemic as the rapid spread of disease to a large number of hosts in a given population within a short period of time. The US Centers for Disease Control give a broader version: a sudden rise in cases above what is normally expected (as summarised by the same article). Both definitions hide a comparison. An epidemic is not a fixed number of cases; it is *more than usual* for that place and time, so you must know what "usual" is before you can call anything an epidemic. A handful of cases of a disease that never occurs in a region may be an outbreak worth investigating, while a thousand cases of a common cold in a winter week is routine.

TERM	MEANING	HOW IT IS USED
Outbreak	a rise in cases, often local or small	the article says it is usually restricted to smaller events, and can also apply to an epidemic
Epidemic	cases well above what is expected, over a short time	depends on the baseline for that population
Endemic	a disease that is present at a steady, expected level	the article gives no formal modern definition; this is the usual sense (check)
Pandemic	an epidemic that spreads to other countries or continents and affects a substantial number of people	defined by geography as well as size

Incidence and Prevalence

These two words are the most commonly confused in the whole subject. **Incidence** counts *new* cases in a period. **Prevalence** counts *existing* cases at a moment. Think of a bath: incidence is the water coming in from the tap, and prevalence is the depth of the water now. The Wikipedia article on incidence describes incidence as the number of new cases in a population and prevalence as the proportion of cases in the population at a given time. It also separates two forms of incidence: the **cumulative incidence** (or incidence proportion), which is new cases divided by the number of people followed, and the **incidence rate**, which divides new cases by the total person-time at risk, so people need not be watched for equally long.

	INCIDENCE	PREVALENCE
Counts	new cases	all current cases
Time	over a period (a day, a week)	at one moment (point) or over a period
Tells you	how fast the disease is arriving, the risk of catching it	how much disease there is, the burden at a time
Falls when	fewer people are being infected	people recover, die, or fewer are infected
Bath picture	the tap	the water level

The two are linked. In a steady state, where incidence stays roughly constant, the article gives `prevalence = incidence × duration`. A tap that adds 20 litres a minute to a bath that holds each litre for 6 minutes keeps 120 litres in the bath. The same arithmetic applies to people:

```
# Prevalence is roughly incidence x average duration (steady state).
# Illustrative numbers: assumed, not taken from a real disease.
population = 100_000
new_cases_per_day = 20        # incidence: assumed
mean_duration_days = 6        # assumed

incidence_rate = new_cases_per_day / population
ill_at_any_moment = new_cases_per_day * mean_duration_days
prevalence = ill_at_any_moment / population

print(f"incidence: {incidence_rate * 100_000:.0f} new cases per 100,000 per day")
print(f"ill at any moment: {ill_at_any_moment}")
print(f"point prevalence: {prevalence:.2%}")
yearly = new_cases_per_day * 365
print(f"new cases in a year: {yearly} ({yearly / population:.1%} of the population)")
# incidence: 20 new cases per 100,000 per day
# ill at any moment: 120
# point prevalence: 0.12%
# new cases in a year: 7300 (7.3% of the population)
```

With 20 new cases a day and an illness lasting 6 days, 120 people are ill at any moment, a prevalence of 0.12%. Note how different the two views look: a year of that incidence infects 7.3% of the population, while only 0.12% is ill on any given day. A disease with a long duration can have a high prevalence from a modest incidence, and a short, sharp illness can sweep through a town while its prevalence stays low on most days.

A Real Outbreak: The Boarding School

In 1978 an influenza outbreak struck a boys' boarding school in northern England. The R package documentation for the dataset (the pomp package calls it `bsflu`) says that 763 boys were at risk, that the index case was a boy who came back from holiday six days before the next case, and that 512 boys spent between 3 and 7 days away from class. The original report is an anonymous note in the British Medical Journal in 1978. One dataset entry words the 512

differently from the others (as the number who became ill), another sign that sources disagree in detail.

The data record, for each of 14 days, how many boys were *confined to bed*. That is a **prevalence**, not an incidence: a boy who was in bed for three days appears in three daily counts. The code below finds the peak, draws the curve in text, and then uses the relation above in reverse.

```
# Daily counts of boys confined to bed in a 1978 boarding-school outbreak
# (read from the 'outbreaks' R package data; the original was a figure).
in_bed = [3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4]
convalescent = [0, 0, 0, 0, 9, 17, 105, 162, 176, 166, 150, 85, 47, 20]
boys = 763
cases = 512          # boys who were ill at some point (sources differ)

peak = max(in_bed)
peak_day = in_bed.index(peak) + 1
print("peak day:", peak_day, "| in bed:", peak, f"({peak / boys:.1%} of the school)")
print("sum of the daily counts:", sum(in_bed))
print(f"average days in bed per case: {sum(in_bed) / cases:.2f}")
print(f"attack rate: {cases / boys:.1%}")
for day, n in enumerate(in_bed, start=1):
    print(f"day {day:2d} {n:3d} " + "#" * round(n / 6))
# peak day: 6 | in bed: 298 (39.1% of the school)
# sum of the daily counts: 1559
# average days in bed per case: 3.04
# attack rate: 67.1%
# day  1   3
# day  2   8 #
# day  3  26 ####
# day  4  76 #############
# day  5 225 ######################################
# day  6 298 ##################################################
# day  7 258 ############################################
# day  8 233 ########################################
# day  9 189 ################################
# day 10 128 ######################
# day 11  68 ###########
# day 12  29 #####
# day 13  14 ##
# day 14   4 #
```

The peak came on day 6, when 298 boys, 39.1% of the school, were in bed. Adding up the 14 daily counts gives 1,559, which is more than twice the 763 boys in the school, precisely because the counts are prevalences and add each boy once for every day. Dividing by the 512 cases gives an average of 3.04 days in bed per case, which is plausible against the statement that the cases lost 3 to 7 days of school. That is the identity at work: total person-days of illness equals cases times average duration. If 512 is right, the attack rate was 67.1%, an enormous share of a closed community for a single outbreak in about two weeks. Notice the shape of the curve: a slow start (3, 8, 26), a steep climb that roughly triples each day for the first four days, a peak, and a fall that is slower than the rise. How Epidemics Spread 2 turns that steep climb into a number.

Reading an Epidemic Curve

An **epidemic curve** (or epi curve) is a chart of the number of new cases against the date of onset, according to the Wikipedia article. It shows the size of the outbreak, its trend over time, whether cases cluster, the likely incubation period, and the likely mode of transmission. The article names three typical shapes.

SHAPE	SOURCE OF EXPOSURE	WHAT THE CURVE LOOKS LIKE
Point source	everyone exposed at about the same time, such as one meal	a sharp rise to one peak and a fall; the width of the curve matches the spread of incubation times
Continuous common source	exposure carries on, such as contaminated water that is not fixed	a rise followed by a plateau or a long, ragged tail
Propagated	person to person	successive waves, each starting from the cases of the one before; the generations may show as humps

A real curve is evidence, not proof. The boarding-school curve above is a prevalence, plotted by day; a proper epi curve plots new onsets. The simulation below builds both from the same invented outbreak so you can see how they differ. It is a simple day-by-day bookkeeping model: each day, the new cases are proportional to the number of people still susceptible multiplied by the fraction of the school that is infectious. Each case stays infectious for 3 days. The transmission rate `beta` of 0.9 and the 3-day duration are **assumed**; they are not a fit to the real data. Because it divides cases into fractions of people, it is a model, not a head count. How Epidemics Spread 3 builds this idea properly as the SIR model.

```
# A bookkeeping simulation: how incidence (new cases per day) and
# prevalence (people infectious right now) relate. All parameters are assumed.
def simulate(population=763, beta=0.9, duration=3, days=28):
    susceptible = population - 1.0
    infectious = [1.0]                    # one cohort per day; each stays 'duration' d
    rows = []
    for day in range(1, days + 1):
        prevalence = sum(infectious[-duration:])
        incidence = min(susceptible, beta * susceptible * prevalence / population)
        susceptible -= incidence
        infectious.append(incidence)
        rows.append((day, incidence, sum(infectious[-duration:])))
    return rows

rows = simulate()
for day, inc, prev in rows[:14]:
    print(f"day {day:2d}  new {inc:6.1f}  infectious now {prev:6.1f}")
peak_inc = max(rows, key=lambda r: r[1])
peak_prev = max(rows, key=lambda r: r[2])
print("incidence peaks on day", peak_inc[0], "| prevalence peaks on day", peak_prev[0])
print(f"total ever infected: {sum(r[1] for r in rows) + 1:.0f} of 763")
# day  1  new    0.9  infectious now    1.9
# day  2  new    1.7  infectious now    3.6
# day  3  new    3.2  infectious now    5.8
# day  4  new    5.2  infectious now   10.1
# day  5  new    9.0  infectious now   17.4
# day  6  new   15.2  infectious now   29.4
# day  7  new   25.2  infectious now   49.4
# day  8  new   40.9  infectious now   81.3
# day  9  new   63.4  infectious now  129.5
# day 10  new   91.2  infectious now  195.5
# day 11  new  116.7  infectious now  271.3
# day 12  new  124.6  infectious now  332.5
# day 13  new  103.8  infectious now  345.1
# day 14  new   65.5  infectious now  293.9
# incidence peaks on day 12 | prevalence peaks on day 13
# total ever infected: 726 of 763
```

Incidence peaks on day 12 with about 125 new cases, and prevalence peaks a day later at about 345 people infectious, because the people infected at the peak are still infectious on the next

day. This is the general pattern: **prevalence lags incidence** by an amount that depends on the duration of the illness. By the end about 726 of the 763 are infected in this model, whereas the real outbreak is recorded at 512, so the invented parameters overshoot; the real school may have had immune boys or a lower transmission rate (an assumption; the sources do not say).

Old Habits: Snow and the Pump

Counting by place and time is as old as the subject. In the 1854 Broad Street cholera outbreak in Soho, London, John Snow mapped the deaths and traced them to a water pump. The Wikipedia article says that within three days of the start of the major outbreak on 31 August, 127 people on or near Broad Street died, that more than 300 had died by 10 September, and that the pump handle was removed on 8 September. Total death figures differ between accounts: the page as fetched gave 420, and other accounts I remember give higher numbers (check). The lesson for this chapter is the method: Snow counted *who* was ill and *where*, and the concentration of the counts around one pump pointed to a single common source.

What Can Go Wrong When You Count

> Common mistakes
> Treating prevalence as incidence (adding up daily "currently ill" counts as if they were new cases, which counts each person many times); mixing the date of onset with the date of report or test (a curve by report date is shifted and lumpy, and weekends can hide cases); assuming that every case is counted (mild or symptom-free cases are often missed, so counts are a lower bound); comparing raw numbers between places of different size (use a rate, such as cases per 100,000); and calling a number an epidemic without knowing the baseline for that population.

EXERCISE 1

A village of 200 people (assumed) has 12 cases, each described by the first and last day of illness. For days 1 to 12, find the incidence (new cases) and the prevalence (people ill that day), check that prevalence equals the cases started so far minus those already recovered, and find the cumulative incidence, the mean duration and the peak prevalence.

View solution

EXERCISE 2

Using the 14 daily in-bed counts from the boarding school, find the day-to-day ratio for days 1 to 6, the days of the largest rise and the largest fall, the days on which more than half the peak number were in bed, and the day when most boys were in convalescence. Why does the convalescence peak come after the in-bed peak?

View solution

EXERCISE 3

Simulate a point-source outbreak: 80 guests share a dinner, each falls ill with probability 0.4 and the time to onset is between 6 and 48 hours (all assumed). Draw the epidemic curve in 6-hour bins and decide whether its shape looks like a point-source or a propagated outbreak, and why it is not a perfect bell.

View solution

QUICK REFERENCE

- **Epidemic:** cases above what is expected for that population and time
- **Incidence:** new cases in a period (cumulative incidence is a proportion; the incidence rate uses person-time)
- **Prevalence:** existing cases at a moment; in a steady state, prevalence = incidence × duration
- **Epidemic curve:** new cases against date of onset; shapes are point source, continuous common source and propagated
- **Prevalence lags incidence:** by roughly the duration of the illness
- **Check before trusting a count:** onset or report date, missed cases, rate or raw number

COMING NEXT

How Epidemics Spread 2 is about growth and doubling: exponential growth, doubling time, the reproduction number R0, and a first Python growth simulation.

How Epidemics Spread

Chapter 2 · Growth and Doubling

How Epidemics Spread 1 ended on the boarding-school curve: 3 boys in bed on the first day, then 8, 26, 76 and 225. The day-to-day ratios are 2.67, 3.25, 2.92 and 2.96, so the number in bed roughly tripled each day. That is **exponential growth**: the amount added each day is proportional to the amount already there. This chapter turns that observation into three tools. The first is the **doubling time**, a number you can feel. The second is the **reproduction number R0**, the number that says whether an outbreak grows at all. The third is a short Python simulation that shows why growth cannot go on for ever. There is no calculus here: only multiplication, powers, and the logarithm button on a calculator (or Python's `math.log`).

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The definition of R0, the quoted R0 ranges for measles and seasonal influenza, and the doubling-time formula and rule of 70 come from the Wikipedia articles on the basic reproduction number and on doubling time, fetched with a tool that often returns truncated pages or model-written summaries; the R0 page was longer than the part the tool read, so treat the wording as a paraphrase and the ranges as one source's figures, not settled facts. That article does not give a formula linking R0 to the growth rate, so the link used below (a fixed generation time) is a *modelling assumption* of this course, not a quotation. Growth rates, R0 values, populations and generation times in the code are *assumed* or *illustrative*: they show a method and are not measurements of any real disease.

Linear Versus Exponential

Suppose a town adds 10 cases every day. After 30 days there are 300: the line climbs steadily. Now suppose instead that the number of cases grows by 25% a day, starting from 10. The code below shows that after 31 days there are more than 10,000. Linear growth adds the same *amount* each step; exponential growth multiplies by the same *factor*. Disease is exponential at the start because each case infects others, and those infect others in turn: the more cases there are, the more new ones appear.

	LINEAR GROWTH	EXPONENTIAL GROWTH
Rule	add the same amount each step	multiply by the same factor each step
Example (illustrative)	10, 20, 30, 40, 50	10, 20, 40, 80, 160
Time to double	keeps getting longer	always the same
On a log chart	curves flatter and flatter	a straight line
Disease picture	a steady source, such as a polluted well	person-to-person spread with no limit yet

Doubling Time

The **doubling time** is how long it takes for the quantity to double. For a growth of r% per period, the Wikipedia article gives the doubling time as `t × ln(2) / ln(1 + r/100)`, where `ln` is the natural logarithm (just a button on the calculator, or `math.log` in Python). It also gives the **rule of 70**: divide 70 by the percentage growth per period for a quick estimate. At 2% a day, 70 / 2 = 35 days. The article says the rule is accurate to within 10% for growth rates below 25%. The table printed below checks that claim and shows where it fails.

```
# Doubling time from a constant daily growth rate, and the rule of 70.
# The growth rates are illustrative, not measurements.
import math

print("daily growth  exact doubling  rule of 70   error")
for pct in [1, 2, 5, 10, 25, 40]:
    exact = math.log(2) / math.log(1 + pct / 100)
    rule = 70 / pct
    print(f"{pct:9d}%  {exact:11.2f} d  {rule:9.2f} d  {abs(rule - exact) / exact:6.1%}"

cases = 10
days = 0
while cases < 10_000:
    cases *= 1.25
    days += 1
print(f"10 cases growing 25% a day pass 10,000 after {days} days")
# daily growth  exact doubling  rule of 70   error
#         1%        69.66 d      70.00 d    0.5%
#         2%        35.00 d      35.00 d    0.0%
#         5%        14.21 d      14.00 d    1.5%
```

```
#        10%          7.27 d       7.00 d    3.7%
#        25%          3.11 d       2.80 d    9.9%
#        40%          2.06 d       1.75 d   15.1%
# 10 cases growing 25% a day pass 10,000 after 31 days
```

The rule of 70 is off by only 0.5% at 1% a day, and by 3.7% at 10% a day. At 25% it reaches 9.9%, just inside the article's limit, and at 40% a day the error is 15.1%, so the rule stops being a good shortcut. Early epidemics can grow faster than 25% a day, which is why the exact formula is worth keeping. The practical lesson of the last line: even at 25% a day, 10 cases become 10,000 in a month. A doubling time of 3 days means a thousandfold increase in about 30 days, because ten doublings multiply a number by 1,024.

The Reproduction Number R0

The **basic reproduction number**, R0 ("R naught"), is, in the article's words, the expected number of cases directly generated by one case in a population where everyone is susceptible. It has no units: it is a count of people, not a rate. The key threshold comes from the same source: in common infection models, R0 above 1 means the infection can spread and the outbreak will expand, while R0 below 1 means it dies out. The article adds a caution that some models, such as those for diseases carried by insects, can sustain an outbreak even with R0 below 1. It also stresses that R0 by itself does not tell you how *fast* an infection spreads, because speed depends on how long each generation of infection takes.

R0	WHAT IT MEANS	OUTCOME IN THE SIMPLE MODEL
Below 1	each case infects fewer than one other, on average	the outbreak shrinks and dies out
Exactly 1	each case replaces itself	a steady trickle, neither growing nor shrinking
Above 1	each case infects more than one other	the outbreak grows, and the larger R0 is, the faster
Measles (one source)	quoted range 12 to 18	among the highest figures for any common infection (check)
Seasonal influenza (one source)	quoted as 1.3, range 1.2 to 1.4	growth per generation is much slower

A caution about the last two rows: R0 is not a fixed property of a germ. It depends on how people mix, how crowded they are, and what they do, so different studies report different values for the same disease, and an R0 measured in one country need not hold in another. Once some people are immune or have changed their behaviour, the average number of new infections per case is called the **effective reproduction number**, often written R or Rt (check the naming; later chapters use it). R0 is the special case where everyone can still be infected.

Generations

The simplest way to see what R0 does is to count by **generations**. The first case is generation 0. Its R0 contacts are generation 1, their contacts are generation 2, and so on. If every case infects exactly R0 others, generation g has $R0^g$ new cases. The code below prints eight generations for five invented values of R0. It ignores the fact that people get used up, so it is only valid early on.

```
# Cases by generation when every case infects R0 others (no one is used up yet).
# The R0 values are illustrative; the generation count is a model idea, not a clock.
for r0 in [0.8, 1.0, 1.5, 2.5, 6.0]:
    total = 1.0
    new = 1.0
    row = []
    for generation in range(1, 9):
        new *= r0
        total += new
        row.append(round(new, 1))
    print(f"R0 {r0:3.1f}: new cases per generation {row}  total after 8: {round(total)}"
# R0 0.8: new cases per generation [0.8, 0.6, 0.5, 0.4, 0.3, 0.3, 0.2, 0.2]  total after
# R0 1.0: new cases per generation [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]  total after
# R0 1.5: new cases per generation [1.5, 2.2, 3.4, 5.1, 7.6, 11.4, 17.1, 25.6]  total af
# R0 2.5: new cases per generation [2.5, 6.2, 15.6, 39.1, 97.7, 244.1, 610.4, 1525.9]  t
# R0 6.0: new cases per generation [6.0, 36.0, 216.0, 1296.0, 7776.0, 46656.0, 279936.0,
```

Compare the rows. With R0 of 0.8 the outbreak withers: 0.8 new cases in the first generation and 0.2 by the eighth. At exactly 1.0 it ticks along at one case per generation. At 1.5 it reaches about 26 new cases in the eighth generation, at 2.5 about 1,526, and at an R0 of 6.0 it reaches about 1.7 million. Small differences in R0 become huge differences in size within a few generations,

which is why a measured R0 of 1.3 and one of 1.5 are not interchangeable. Note that a generation is a model unit and not a day: how long it takes is the next question.

From R0 to a Growth Rate

To turn generations into days you need the **generation time**: the time from one person being infected to the people they infect. Suppose, as a simplification, that it is the same fixed number of days T for everyone (**assumed**: real generation times vary from person to person, and later chapters will deal with that, including the serial interval). Then the number of cases is multiplied by R0 every T days, so the growth per day is R0 raised to the power 1/T, and the doubling time is `T × ln(2) / ln(R0)`. The code uses a generation time of 4 days, an assumed value, to show how doubling time falls as R0 rises.

```
# From R0 and the generation time to a growth rate and doubling time.
# Assumed: every case infects people exactly T days after being infected.
# Then cases grow by a factor R0 every T days, so r = ln(R0) / T per day.
import math

T = 4  # assumed generation time in days
print("   R0   daily growth   doubling time")
for r0 in [1.2, 1.5, 2.0, 3.0, 5.0]:
    r = math.log(r0) / T
    doubling = math.log(2) / r
    print(f"{r0:5.1f}   {math.exp(r) - 1:9.1%}   {doubling:9.1f} d")

# Check the formula by stepping day by day at the daily growth rate for R0 = 2.
growth = 2.0 ** (1 / T) - 1          # the daily rate that gives a factor of 2.0 per T d
cases = 1.0
for day in range(1, 13):
    cases *= 1 + growth
    if day % T == 0:
        print(f"day {day:2d}: {cases:.2f} new cases a day (a factor of 2 every {T} days)
#    R0   daily growth   doubling time
#   1.2        4.7%        15.2 d
#   1.5       10.7%         6.8 d
#   2.0       18.9%         4.0 d
#   3.0       31.6%         2.5 d
#   5.0       49.5%         1.7 d
# day  4: 2.00 new cases a day (a factor of 2 every 4 days)
# day  8: 4.00 new cases a day (a factor of 2 every 4 days)
```

```
# day 12: 8.00 new cases a day (a factor of 2 every 4 days)
```

With an R0 of 2 and a 4-day generation, the number of new cases doubles every 4 days, as it must, because R0 is the factor per generation. With an R0 of 1.2 the same disease takes about 15 days to double; with an R0 of 5 it doubles in about 1.7 days, growing nearly 50% a day. The same R0 with a different generation time would give a different clock: halving T halves every doubling time in the table. That is why two diseases with the same R0 can feel very different, and why a quick-passing infection is harder to catch up with than a slow one.

Why Growth Cannot Go On

The mathematics so far says that outbreaks grow for ever, and that is plainly false. The final code block adds the missing piece: as people become infected they cannot be infected again, so the supply of susceptible people shrinks. Each case now infects only R0 × (the fraction still susceptible) others. The population is a closed community of 10,000, R0 is 2.5, the generation is 4 days, and there is one starting case, all **assumed**. The model is crude: a person who has been infected simply drops out of the susceptible pool. How Epidemics Spread 3 replaces it with a proper compartment model.

```
# Growth meets a limit: a closed population of 10,000 (assumed), R0 = 2.5 (assumed),
# generation time 4 days (assumed). New cases = R0 x susceptible fraction x
# cases one generation earlier. Run it and watch the doubling time stretch out.
import math

population = 10_000
r0 = 2.5
T = 4
susceptible = population - 1.0
new = [1.0] + [0.0] * (T - 1)
for day in range(T, 80):
    fraction = susceptible / population
    cases = min(susceptible, r0 * fraction * new[day - T])
    susceptible -= cases
    new.append(cases)

for day in range(T, 80, 4):
    if day >= 8 and new[day - 4] > 0.5:
```

```
        ratio = new[day] / new[day - 4]
        label = "growing" if ratio > 1 else "shrinking"
        print(f"day {day:2d}  new {new[day]:8.1f}  ratio to 4 days ago {ratio:5.2f}  {lab
peak = max(range(len(new)), key=lambda d: new[d])
print("peak day:", peak, "| peak new cases:", round(new[peak]))
print("ever infected:", round(population - susceptible), "of", population)
print(f"naive 100-day prediction, no limit: {r0 ** (100 / T):.2e} cases")
# day  8  new      6.2  ratio to 4 days ago  2.50  growing
# day 12  new     15.6  ratio to 4 days ago  2.50  growing
# day 16  new     38.9  ratio to 4 days ago  2.49  growing
# day 20  new     96.6  ratio to 4 days ago  2.48  growing
# day 24  new    237.7  ratio to 4 days ago  2.46  growing
# day 28  new    570.6  ratio to 4 days ago  2.40  growing
# day 32  new   1288.3  ratio to 4 days ago  2.26  growing
# day 36  new   2493.6  ratio to 4 days ago  1.94  growing
# day 40  new   3272.2  ratio to 4 days ago  1.31  growing
# day 44  new   1617.0  ratio to 4 days ago  0.49  shrinking
# day 48  new    145.4  ratio to 4 days ago  0.09  shrinking
# day 52  new      7.8  ratio to 4 days ago  0.05  shrinking
# day 56  new      0.4  ratio to 4 days ago  0.05  shrinking
# peak day: 40 | peak new cases: 3272
# ever infected: 9794 of 10000
# naive 100-day prediction, no limit: 8.88e+09 cases
```

For about three weeks the growth ratio stays close to 2.5 every 4 days (2.46 on day 24), just as the simple formula says. Then it starts to slip: 2.26 on day 32, 1.94 on day 36, and 1.31 on day 40. The peak of 3,272 new cases a day comes on day 40, and by day 44 the outbreak is shrinking. At the end 9,794 of the 10,000 people have been infected in this model. The last line shows why the early formula must not be extended: 2.5 raised to the power of 25 generations is about 8.9 billion cases, more than the roughly 8 billion people on Earth (check). Unchecked exponential growth always meets a limit, and the shape of the curve you saw in the boarding school, a climb, a peak and a fall, is the result. The overshoot (nearly everyone infected) is a feature of this crude model, which has no separate recovered group and ignores how long people stay infectious.

> Common mistakes
> Treating R0 as a fixed constant of the pathogen rather than something that depends on the population and its behaviour; quoting an R0 without saying where it came from (sources differ, and the ranges here are one article's figures); forgetting that doubling time depends on the generation time as well as R0, using the rule of 70 for fast growth (above about 25% a day it is no longer within 10%), extending

an exponential forecast far into the future without a limit; and confusing the reproduction number with a rate or with a count of total cases.

EXERCISE 1

New cases on days 1 to 6 of an outbreak are 12, 15, 19, 24, 30 and 38 (assumed, invented). Find the day-to-day ratios, the average daily growth, the doubling time and what the rule of 70 says, then find the day on which new cases per day first reach 1,000 if growth never changes, by a loop and by a formula.

View solution

EXERCISE 2

For R0 values of 1.2, 1.5, 2, 2.5 and 3, find how many generations (and how many days, if a generation is 5 days, assumed) it takes for one case to lead to 1,000 cases in a single generation. Then compare the cases in the third generation for an R0 of 12, 18 and 1.3, the figures quoted for measles and seasonal influenza, and decide how much the uncertainty in R0 matters.

View solution

EXERCISE 3

Wrap the limited-growth simulation in a function and run it for R0 values of 0.9, 1, 1.5, 2, 3 and 5 in a population of 10,000 (generation time 4 days, assumed). Report the peak day, the peak number of new cases and the share ever infected, and explain what happens when R0 is below 1.

View solution

QUICK REFERENCE

- **Exponential growth:** the same factor each step; the doubling time is constant
- **Doubling time:** $t \times \ln(2) / \ln(1 + r/100)$; the rule of 70 ($70 / r$) is good for r below about 25%
- **R0:** expected number of cases one case generates in a fully susceptible population; above 1 the outbreak can grow, below 1 it dies out
- **By generations:** generation g has $R0^g$ new cases (while few have been used up)

- **With a fixed generation time T (assumed):** doubling time = T × ln(2) / ln(R0)
- **Limit:** as susceptibles run out, growth slows, peaks and falls

COMING NEXT

How Epidemics Spread 3 is about the SIR model: susceptible, infected and recovered compartments, differential equations, and stepping them in Python.

How Epidemics Spread

Chapter 3 · The SIR Model

How Epidemics Spread 2 ended with a crude simulation: new cases were R0 times the fraction still susceptible times the cases one generation earlier. It showed the right shape, a climb, a peak and a fall, but it had no real notion of who was ill *now*, and it ignored how long people stay infectious. This chapter replaces it with the model that nearly every later model grows from, the **SIR model**. Everyone is in exactly one of three groups: **S**usceptible (can catch it), **I**nfected, meaning currently infectious (can pass it on), or **R**ecovered (no longer infectious and, in this model, immune, or removed). The model is a pair of rules for how many people move from S to I and from I to R each day. The textbook calls such rules *differential equations*, but you will not need calculus: we simply apply the rules over small steps of time, one step after another, in Python. That is called **stepping** the equations (the method used here is Euler's method).

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The model's equations, the meaning of the two rate constants, the relation R0 = beta/gamma, the 1927 origin (Kermack and McKendrick), the average infectious period 1/gamma and the final-size equation come from the Wikipedia article on compartmental models in epidemiology, fetched with a tool that often returns truncated pages or model-written summaries; the page was far longer than the part the tool read, and the summary noted that the article writes the equations with and without dividing by the population in different sections, so treat the wording as a paraphrase. The statement that infections peak when S falls to N/R0 is not quoted in the article: it is worked out below from the equations. All rates, populations and R0 values in the code are *assumed* or *illustrative*; the boarding-school fit in the third exercise is a classroom fit, not a published estimate.

Three Boxes and Two Arrows

Picture three boxes in a row: S, then I, then R, with an arrow from S to I and another from I to R. People only move forwards. The population is closed: nobody is born, dies or arrives (**assumed**, and relaxed in How Epidemics Spread 5), so S + I + R stays equal to the total N all the time. In How Epidemics Spread 1 the boarding school counted boys "in bed", and a rough match is that the in-bed count is the I box and the convalescent boys are heading for R (an assumption, since being in bed is not quite the same as being infectious).

BOX	WHO IS IN IT	HOW PEOPLE ENTER	HOW PEOPLE LEAVE
S susceptible	can catch the infection	start here (nobody enters later)	get infected, move to I
I infected	infectious now	infected by someone in I	recover (or are removed), move to R
R recovered	no longer infectious, assumed immune	recovery from I	nobody leaves

The Two Rules

Two numbers control the flows. The **infection rate constant** beta (written β) says how fast infectious people infect susceptible ones: the Wikipedia article gives it in units of people infected per day per infectious person. The **recovery rate constant** gamma (γ) is the fraction of the infectious who recover each day. With the population N, the rules for a small step of time are:

- **New infections** in the step = β × S × I / N × (step length). The product S × I / N is large when there are many people who can pass it on *and* many who can catch it; it shrinks as S falls or as I falls.
- **New recoveries** in the step = γ × I × (step length).
- S falls by the new infections; I gains the new infections and loses the new recoveries; R gains the new recoveries.

If the infectious recover at the rate γ = 0.2 per day, then on average a person stays infectious for 1/γ = 5 days. (The article states that the mean infectious period is 1/γ; this holds when the time to recover varies in the particular way built into the model, which is a simplification of real diseases.) Compare this with How Epidemics Spread 2:

	GENERATION MODEL (HOW EPIDEMICS SPREAD 2)	SIR MODEL
Time	jumps of one generation (a fixed number of days)	small steps, such as a day or a tenth of a day
Who is infectious	only the latest generation	the whole box I, with people leaving it at rate γ

	GENERATION MODEL (HOW EPIDEMICS SPREAD 2)	SIR MODEL
What it needs	R0 and a generation time	β and γ (so R0 and the infectious period)
Where it stops	a rough limit on the susceptible fraction	infections fall as S and I both shrink
Origin	this course's simplification	Kermack and McKendrick, 1927 (per the article)

Stepping the Model in Python

The code below follows the rules exactly, one day at a time, for a closed community of 10,000 with a single infectious person at the start. The values β = 0.5 and γ = 0.2 are **assumed** for illustration; they give R0 = 2.5. Each loop turns the rules into three lines of arithmetic: the new infections, the new recoveries, and the three updates. It also records the largest I along the way.

```
# SIR stepped one day at a time. All numbers are assumed, for illustration.
population = 10_000
beta = 0.5      # assumed: contacts that could transmit, per infectious person per day
gamma = 0.2     # assumed: 1/5 of the infectious recover each day (average 5 days)
dt = 1.0        # step length in days

S, I, R = population - 1.0, 1.0, 0.0
peak_I, peak_day, S_at_peak = I, 0, S
print("day       S        I        R")
for day in range(0, 121):
    if day % 10 == 0:
        print(f"{day:3d} {S:8.1f} {I:8.1f} {R:8.1f}")
    if I > peak_I:
        peak_I, peak_day, S_at_peak = I, day, S
    new_infections = beta * S * I / population * dt
    new_recoveries = gamma * I * dt
    S -= new_infections
    I += new_infections - new_recoveries
    R += new_recoveries
print(f"R0 = beta/gamma = {beta / gamma}")
print(f"peak: {peak_I:.0f} infectious on day {peak_day}, when S was {S_at_peak:.0f}")
print(f"S = N/R0 predicts the peak at S = {population / (beta / gamma):.0f}")
print(f"total never infected at the end: {S:.0f}; S+I+R = {S + I + R:.1f}")
# day       S        I        R
```

```
# day  susceptible  infectious  recovered
#   0   9999.0      1.0      0.0
#  10   9977.7     13.8      8.5
#  20   9693.4    182.7    123.9
#  30   6983.1   1610.1   1406.8
#  40   2161.6   2009.9   5828.5
#  50   1129.7    539.4   8330.9
#  60    974.5    108.6   8916.9
#  70    946.7     20.9   9032.4
#  80    941.5      4.0   9054.6
#  90    940.5      0.8   9058.8
# 100    940.3      0.1   9059.6
# 110    940.3      0.0   9059.7
# 120    940.3      0.0   9059.7
# R0 = beta/gamma = 2.5
# peak: 2487 infectious on day 36, when S was 3570
# S = N/R0 predicts the peak at S = 4000
# total never infected at the end: 940; S+I+R = 10000.0
```

Read the table like a story. For ten days almost nothing seems to happen: 13.8 people are infectious on day 10, then 183 on day 20 and 1,610 on day 30. That quiet start is the exponential growth of How Epidemics Spread 2, and it is why outbreaks often look harmless until they are not. The infectious number peaks at 2,487 on day 36; by day 60 only about 109 remain, and the epidemic ends with 940 people never infected. Two checks are built in. First, S + I + R is 10,000 to the last decimal (a sign the arithmetic moves people between boxes without losing any). Second, the peak line and the prediction line compare the S at the peak with N/R0 = 4,000. They do not agree exactly (3,570 against 4,000), and the next sections explain both the prediction and the gap.

Why R0 = β / γ

Take one infectious person in a population where everyone else is susceptible, so S/N is about 1. Each day they pass the infection to β people. They stay infectious for 1/γ days on average. So over their whole infectious period they infect β × (1/γ) = β/γ people. That is exactly R0, the expected number of cases one case generates in a fully susceptible population: the same definition as in How Epidemics Spread 2, now built from two things you can measure separately, how infectious the disease is per day and how long it lasts. The same reasoning shows how the effective reproduction number falls: when only a fraction S/N of the population can still be infected, each case infects R0 × S/N people.

That gives the peak rule. Rewrite the change in I per day as (R0 × S/N − 1) × γ × I. The number of infectious people keeps growing while R0 × S/N is above 1 and starts to shrink when it drops below 1, so the peak is at S = N/R0. For R0 = 2.5 and N = 10,000 that is S = 4,000. The article states this threshold only as a herd-immunity threshold; reading it as the peak condition is this course's own inference from the equation. It is the seed of How Epidemics Spread 4. Notice what the peak is *not*: it is not the end. After I peaks, I still falls only gradually while it keeps infecting people, so S continues to drop below 4,000. In the run above, S was 2,162 by day 40 and ended at 940.

How Small Must the Step Be?

The run above used a one-day step, and the peak S came out at 3,570 rather than 4,000. Stepping the rules is an approximation: during a whole day the rule assumes the rate of infection stays fixed even though S and I change as the day goes on. Shorter steps follow the real curve more closely. The code below runs the same model with steps from 5 days down to 0.01 days (1/100 of a day). It also stops the day's new infections from exceeding S, a guard that matters with very large steps.

```
# How much does the step length matter? Same model, smaller and smaller steps.
population, beta, gamma = 10_000, 0.5, 0.2   # assumed values

def run(dt, days=150):
    S, I, R = population - 1.0, 1.0, 0.0
    peak_I, peak_t = I, 0.0
    for k in range(int(days / dt)):
        new_inf = min(S, beta * S * I / population * dt)
        new_rec = gamma * I * dt
        S -= new_inf
        I += new_inf - new_rec
        R += new_rec
        if I > peak_I:
            peak_I, peak_t = I, (k + 1) * dt
    return peak_I, peak_t, S

print("step (days)  peak infectious  peak time (days)  never infected")
for dt in [5.0, 2.0, 1.0, 0.5, 0.1, 0.01]:
    peak_I, peak_t, S = run(dt)
    print(f"{dt:10.2f} {peak_I:16.0f} {peak_t:17.1f} {S:15.0f}")
# step (days)  peak infectious  peak time (days)  never infected
```

```
#       5.00            3272            50.0            206
#       2.00            2636            40.0            791
#       1.00            2487            36.0            940
#       0.50            2411            34.0           1009
#       0.10            2350            32.4           1061
#       0.01            2337            32.1           1072
```

The 5-day step is clearly wrong: it reports a peak of 3,272 on day 50 when the answer is closer to 2,337 on day 32. The 1-day step used above overshoots the peak by about 6% (2,487 against 2,337) and by 4 days. Halving to 0.5 days and then 0.1 days brings the answers close to the 0.01 results (peak around 2,340 on day 32, about 1,070 never infected), and the peaks in the last two rows differ by less than 1%. The lesson is practical: when a result barely changes as the step is halved, it is converged; when it keeps changing, the step is too coarse. Always repeat a simulation with a smaller step before you trust a number. (The simple method used here has a known weakness: it needs a small step. Better methods exist, and calculus-based treatments use them, but this one is enough for everything in this course.)

Where Does the Epidemic End?

Look again at the first table: 940 people were never infected. An SIR epidemic does not stop because the susceptible run out. It stops because, as S falls, each case infects fewer than one other (R0 × S/N drops below 1), so I declines to nothing while some people remain in S. The article gives a **final-size** equation for the share who escape, and the code below uses a rearranged version: if z is the share of the population ever infected (starting with almost nobody infected), then $z = 1 - e^{-R0 \times z}$, which you can solve with no calculus by guessing z, plugging it into the right-hand side, and repeating until z stops changing. The code compares that answer with the full step-by-step simulation, using a step of 0.05 days. (The equation as printed in the article uses its own notation; the rearrangement here is mine, for the case of almost everyone starting susceptible.)

```
# Final size: simulate, then check against a number found by repeated guessing.
import math

population, gamma, dt = 10_000, 0.2, 0.05    # assumed values
```

```
def simulate(r0):
    beta = r0 * gamma
    S, I, R = population - 1.0, 1.0, 0.0
    while I >= 0.01:
        new_inf = beta * S * I / population * dt
        new_rec = gamma * I * dt
        S -= new_inf
        I += new_inf - new_rec
        R += new_rec
    return (population - S) / population

def final_size_guess(r0):
    z = 0.5                       # start with any guess for the share ever infected
    for _ in range(200):
        z = 1 - math.exp(-r0 * z)  # a self-consistency equation; repeat until steady
    return z

print("  R0   simulated share infected   self-consistency answer")
for r0 in [0.8, 1.0, 1.2, 1.5, 2.0, 2.5, 3.0, 5.0]:
    print(f"{r0:5.1f} {simulate(r0):14.1%} {final_size_guess(r0):24.1%}")
#   R0   simulated share infected   self-consistency answer
#   0.8           0.0%                     0.0%
#   1.0           1.4%                     1.0%
#   1.2          31.4%                    31.4%
#   1.5          58.3%                    58.3%
#   2.0          79.7%                    79.7%
#   2.5          89.3%                    89.3%
#   3.0          94.1%                    94.0%
#   5.0          99.3%                    99.3%
```

The two columns agree to within about a tenth of a percentage point for every R0 of 1.2 and above (for example 58.3% at R0 = 1.5 by both methods), which is a good check that both are right. Three points stand out. First, R0 = 1.2 infects 31% in the end, and R0 = 2 infects 80%: the share rises steeply near 1 and then flattens, so the extra infected from doubling R0 from 2.5 to 5 is only about 10 percentage points. Second, at R0 = 0.8 the final number is effectively zero: with R0 below 1 there is no epidemic at all, just a few cases. Third, at R0 exactly 1 the simulation reaches 1.4% while the repeated-guess method shows 1.0% after only 200 repeats: convergence is very slow at R0 = 1, so that row mostly shows the limits of the quick method, not a real difference. At R0 = 1 the true answer for a large population is a tiny share.

Early Growth, Revisited

How Epidemics Spread 2 turned R0 into a doubling time using a fixed generation time. The SIR model gives a different answer with no extra assumption. At the start almost everyone is susceptible (S/N is about 1), so the rule for I becomes: change per day = (β − γ) × I = γ × (R0 − 1) × I. That is exponential growth at the rate γ × (R0 − 1) per day, and so a doubling time of ln(2) / (γ × (R0 − 1)). The code checks this on a large population (so S really is near N) and compares it with the fixed-generation idea.

```
# Early growth in the SIR model compared with the generation-time idea.
import math

population, gamma, dt = 1_000_000, 0.2, 0.01     # assumed values
r0 = 2.5
beta = r0 * gamma
S, I = population - 10.0, 10.0
t, history = 0.0, []
while t < 10:
    history.append((t, I))
    new_inf = beta * S * I / population * dt
    S -= new_inf
    I += new_inf - gamma * I * dt
    t += dt
t1, i1 = history[int(2 / dt)]
t2, i2 = history[int(8 / dt)]
measured = (t2 - t1) * math.log(2) / math.log(i2 / i1)
rate = gamma * (r0 - 1)
print(f"measured doubling time, days 2 to 8: {measured:.2f} days")
print(f"growth rate gamma*(R0-1) = {rate:.2f} per day -> doubling {math.log(2) / rate:.2
print(f"fixed 5-day generation idea (Chapter 2 style): {5 * math.log(2) / math.log(r0):.
# measured doubling time, days 2 to 8: 2.31 days
# growth rate gamma*(R0-1) = 0.30 per day -> doubling 2.31 days
# fixed 5-day generation idea (Chapter 2 style): 3.78 days
```

The measured doubling time, 2.31 days, matches the formula exactly. The fixed 5-day generation of How Epidemics Spread 2 would give 3.78 days for the same R0 and the same average infectious period. The two models make different *assumptions* about timing: in the SIR model, people stop being infectious at random moments, and many pass the infection on early in their

infectious period. So the same R0 and the same mean infectious time can give a faster or slower epidemic depending on how the timing is spread out, which is the point How Epidemics Spread 8 returns to when it covers serial intervals. Do not treat either number as the truth for a real disease.

Common mistakes

Using β as if it were R0 (R0 is β divided by γ); forgetting to divide by N in the infection rule, which makes the epidemic depend on the population size; updating S, I and R one after another using already-changed values instead of working out both flows first; trusting a result from one step size without halving the step; reading "the epidemic peaks when S = N/R0" as "the epidemic ends when S = N/R0"; expecting everyone to be infected in the end (a share always escapes unless R0 is huge); and forgetting that the model assumes a closed population, equal mixing, and permanent immunity, none of which is exactly true of a real disease.

EXERCISE 1

Keep R0 = 2.5 in a population of 10,000 with one starting case, but give the disease infectious periods of 2, 5 and 10 days (assumed). Work out β and γ for each, run the model, and report the peak day, the peak number infectious and the number never infected. What changes and what stays the same?

View solution

EXERCISE 2

With a strongly infectious disease (assumed β = 2.0, γ = 0.2, so R0 = 10), run the model for 60 days with steps of 1, 0.5, 0.1 and 0.01 days, once with and once without the guard that stops new infections exceeding S. Report the lowest S reached in each run, and explain what goes wrong with a step that is too large.

View solution

EXERCISE 3

Use the 14 daily in-bed counts from the boarding school in How Epidemics Spread 1 (763 boys, 3 infectious on day 1, assumed to match the I box). Search a grid of β and γ values for the pair that gives the smallest sum of squared differences from the data, then print the observed and model numbers side by side, the fitted R0, and the share of the school the model infects against the 512 recorded cases.

View solution

QUICK REFERENCE

- **Boxes:** S (susceptible), I (infectious), R (recovered or removed); S + I + R = N in a closed population
- **New infections per step:** $\beta \times S \times I / N \times$ step
- **New recoveries per step:** $\gamma \times I \times$ step
- **R0 = β / γ;** the average infectious period is $1/\gamma$
- **Peak of I:** when $R0 \times S/N = 1$, that is $S = N/R0$ (worked out from the equation)
- **Early growth rate:** $\gamma \times (R0 - 1)$ per day
- **Final size:** $z = 1 - e^{-R0 \times z}$ for the share ever infected (almost everyone starting susceptible); never 100%
- **Check:** halve the step; if the answer moves, the step was too big

COMING NEXT

How Epidemics Spread 4 is about herd immunity: the threshold 1 - 1/R0, vaccination, imperfect vaccines, and what coverage is needed.

How Epidemics Spread

Chapter 4 · Herd Immunity

How Epidemics Spread 3 ended with a hint: infections peak when the susceptible count falls to N/R0, because from then on each case infects fewer than one other person. Nothing in that argument says *how* the susceptible count falls. It can fall because people catch the disease and recover, or because they were protected beforehand by vaccination. This chapter follows that second route. The idea is called **herd immunity**: once enough of a community is immune, the few who are not are protected too, because an infection that meets mostly immune people cannot keep itself going. By the end you will have the threshold formula, you will have checked it by simulation, and you will know why an imperfect vaccine needs more than the threshold coverage and why a good average can hide a dangerous pocket.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The threshold formulas, the measles figures (R0 of 12 to 18 and a threshold of 92 to 94 per cent) and the history of the term (first used by the American veterinary scientist Daniel Elmer Salmon in 1894 for the disease resistance of well-fed herds of hogs, then for cattle in 1916 and for experimental epidemics in mice in 1923) come from the Wikipedia article on herd immunity, fetched with a tool that often returns truncated pages or model-written summaries; this fetch covered only the first 100,000 of about 166,000 characters, so treat the wording as a paraphrase. **Source conflict:** the same article gives the measles threshold as 92 to 94 per cent in its table but speaks elsewhere of a threshold “exceeding 95%”, and I could not tell which reflects the better estimate. The reasoning that the threshold is the point where R0 × S/N reaches 1 is the same inference made in How Epidemics Spread 3. Every population size, R0, vaccine effectiveness and coverage in the code is *assumed* or *illustrative*, not an estimate for any real place or disease.

The Threshold in One Line

How Epidemics Spread 3 showed that, with a fraction S/N of the population still susceptible, each case infects R0 × S/N people. An outbreak can grow only while that number is above 1. So the infection is held down once R0 × S/N is at or below 1, that is, once the susceptible share is at most 1/R0. Everyone else must be immune, so the **herd immunity threshold** is the immune share

$$p_c = 1 - 1/R0$$

The Wikipedia article gives the same formula and says it comes from setting R0 × S = 1, where S is the susceptible proportion. Try some numbers: R0 = 2 gives 1 − 1/2 = 50 per cent; R0 = 4 gives 75 per cent; R0 = 12 gives about 92 per cent; R0 = 18 gives about 94 per cent. The last two match the measles range in the article. Notice how quickly the threshold climbs: the more infectious the disease, the more of the population must be immune, but the share never reaches 100 per cent for any finite R0. A disease with R0 below 1 needs no immunity at all, as How Epidemics Spread 3 found.

	PEAK OF THE EPIDEMIC (HOW EPIDEMICS SPREAD 3)	HERD IMMUNITY THRESHOLD (THIS CHAPTER)
Condition	R0 × S/N = 1	R0 × S/N = 1 (same equation)
When it is met	part-way through a natural epidemic	before the first case, if enough people are vaccinated
Result	cases start to fall, but many more are still infected	an introduced case does not start an epidemic
Cost	a large share of the population falls ill	vaccination; the immune have not been ill

A Name from the Farm

The phrase did not begin with people. According to the article, Daniel Elmer Salmon, an American veterinary scientist, first used "herd immunity" in 1894 about the health and disease resistance of well-fed herds of hogs, and later uses concerned cattle with brucellosis (1916) and experimental epidemics among mice (1923). The article also stresses that the immune system does not distinguish between immunity from a natural infection and immunity from a vaccine. The mathematics does not care either: a person in R is a person in R, and what differs is the price paid to get there. The simulation below makes that price visible.

Vaccinating Before the Outbreak

The code below uses the SIR rules from How Epidemics Spread 3 with R0 = 2.5, so the threshold is 1 − 1/2.5 = 60 per cent. Before the first case arrives, a share of the population is moved straight into R, as if by a vaccine that is perfect and protects completely (**assumed**; the next

sections drop that assumption). Then one infectious person appears in the 10,000 and the model runs until the outbreak ends, with a step of 0.05 days. The table reports the largest number infectious at once, the share of the *whole* population that caught the disease, and the share of the *unvaccinated* who did.

```
# Vaccinating before the outbreak: what fraction ends up infected? All numbers assumed.
population = 10_000
gamma = 0.2          # assumed: average infectious period 5 days
r0 = 2.5             # assumed, as in the earlier chapters
beta = r0 * gamma
dt = 0.05

def outbreak(coverage):
    """Perfect, all-or-nothing vaccine given before the first case (assumed)."""
    vaccinated = coverage * population
    S, I, R = population - vaccinated - 1.0, 1.0, vaccinated
    peak_I, t = I, 0.0
    while I >= 0.01 and t < 2000:
        new_inf = beta * S * I / population * dt
        new_rec = gamma * I * dt
        S -= new_inf
        I += new_inf - new_rec
        R += new_rec
        t += dt
        peak_I = max(peak_I, I)
    unvaccinated = population - vaccinated
    return peak_I, (unvaccinated - S) / population, (unvaccinated - S) / unvaccinated

print(f"threshold 1 - 1/R0 = {1 - 1 / r0:.0%}")
print("coverage  peak infectious  share of everyone infected  share of unvaccinated infe
for coverage in [0.0, 0.2, 0.4, 0.5, 0.6, 0.7, 0.8]:
    peak, everyone, unvax = outbreak(coverage)
    print(f"{coverage:7.0%} {peak:16.0f} {everyone:20.1%} {unvax:28.1%}")
# threshold 1 - 1/R0 = 60%
# coverage  peak infectious  share of everyone infected  share of unvaccinated infected
#       0%             2343                89.3%                        89.3%
#      20%             1231                63.8%                        79.7%
#      40%              379                35.0%                        58.3%
#      50%              108                18.6%                        37.2%
#      60%                1                 0.9%                         2.2%
#      70%                1                 0.0%                         0.1%
```

```
#     80%                1                0.0%                        0.1%
```

Three features deserve attention. First, with no vaccination the outbreak infects 89.3 per cent, matching the figure for R0 = 2.5 in How Epidemics Spread 3. Natural herd immunity is reached on the way, when 60 per cent have been infected, yet the epidemic carries on to 89.3 per cent: the cases already infectious keep infecting people after the threshold is passed (an **overshoot**). Vaccination before the outbreak avoids that overshoot. Second, the benefit is not linear. Vaccinating 20 per cent of the population cuts the peak from 2,343 to 1,231, and 40 per cent cuts it to 379, but at 60 per cent the peak is 1: the single introduced case never grows. Third, look at the last column: at 60 per cent coverage the unvaccinated, who have no protection of their own, are infected 2.2 per cent of the time, against 89.3 per cent with no vaccination. That drop is the “herd” part of herd immunity. The 0.9 and 2.2 per cent at exactly 60 per cent are leftovers from the first case: the outbreak sits on the knife-edge R = 1 and dies slowly. At 70 per cent it has all but vanished.

Imperfect Vaccines

A real vaccine does not protect everyone. If its effectiveness is E (the share of vaccinated people who end up protected) and a share v of the population is vaccinated, the immune share is v × E. Setting that equal to the threshold gives the coverage needed. The article states the result as $V_c = (1 - 1/R0) / E$, and adds that if E is less than 1 − 1/R0, eliminating the disease is impossible even if the whole population is vaccinated. The next code prints that formula for a few illustrative cases.

```
# Coverage needed when the vaccine is imperfect: Vc = (1 - 1/R0) / E.
# R0 values are illustrative; E is vaccine effectiveness (all assumed here).
r0_values = [2, 3, 5, 12, 18]
effectiveness = [1.00, 0.95, 0.90, 0.80]

print("R0   threshold   coverage needed at E = " + "  ".join(f"{e:.0%}" for e in effecti
for r0 in r0_values:
    threshold = 1 - 1 / r0
    cells = []
    for e in effectiveness:
        needed = threshold / e
        cells.append(f"{needed:>4.0%}" if needed <= 1 else "none")
```

```
    print(f"{r0:2d} {threshold:10.1%}   " + "     ".join(f"{c:>6}" for c in cells))
print("'none' means even 100% coverage cannot push R below 1 (E < 1 - 1/R0).")
# R0   threshold   coverage needed at E = 100%  95%  90%  80%
#  2      50.0%      50%        53%        56%        62%
#  3      66.7%      67%        70%        74%        83%
#  5      80.0%      80%        84%        89%       100%
# 12      91.7%      92%        96%       none       none
# 18      94.4%      94%        99%       none       none
# 'none' means even 100% coverage cannot push R below 1 (E < 1 - 1/R0).
```

Read across the rows. For R0 = 3 a 95 per cent effective vaccine needs 70 per cent coverage, not 67 per cent. For the high R0 values the demands become severe: with R0 = 12 and E = 95 per cent the coverage needed is 96 per cent, and at E = 90 per cent the threshold of 91.7 per cent is above what the vaccine can deliver, so even vaccinating everybody cannot stop spread by this route. The R0 = 5 row at E = 80 per cent lands on exactly 100 per cent because the threshold (80 per cent) equals the vaccine's effectiveness: with every single person vaccinated R would be exactly 1, which is no safety margin at all. Real programmes also deal with things this table ignores, such as people who cannot be vaccinated and the time it takes to reach everyone.

Two Ways a Vaccine Can Fail

"80 per cent effective" can mean two quite different things (the two models below are standard textbook cartoons, **assumed** here rather than taken from the article). In the **all-or-nothing** picture, 80 per cent of vaccinated people are fully protected and 20 per cent have no protection at all. In the **leaky** picture, every vaccinated person is still susceptible, but their chance of infection per exposure is cut by 80 per cent. The code compares them at R0 = 4, where the perfect-vaccine threshold is 75 per cent and the coverage needed at E = 80 per cent is (1 − 1/4)/0.8 = 93.75 per cent.

```
# Two kinds of imperfect vaccine, same average effectiveness (E = 80%, assumed).
#  all-or-nothing: 80% of vaccinated people are fully protected, 20% not at all
#  leaky: every vaccinated person has their chance of infection cut by 80%
population, gamma, r0, E, dt = 10_000, 0.2, 4.0, 0.8, 0.05
beta = r0 * gamma

def all_or_nothing(coverage):
```

```
    protected = coverage * E * population
    S, I = population - protected - 1.0, 1.0
    while I >= 0.01:
        new_inf = beta * S * I / population * dt
        S -= new_inf
        I += new_inf - gamma * I * dt
    return (population - protected - S) / population

def leaky(coverage):
    Su = (1 - coverage) * population - 1.0     # unvaccinated susceptibles
    Sv = coverage * population                 # vaccinated, still partly susceptible
    I, ever = 1.0, 0.0
    while I >= 0.01:
        force = beta * I / population * dt
        inf_u = Su * force
        inf_v = Sv * force * (1 - E)
        Su -= inf_u
        Sv -= inf_v
        I += inf_u + inf_v - gamma * I * dt
        ever += inf_u + inf_v
    return ever / population

print(f"threshold with perfect vaccine: {1 - 1 / r0:.0%}; coverage needed at E=80%: {(1
print("coverage   all-or-nothing   leaky   (share of whole population infected)")
for coverage in [0.0, 0.5, 0.8, 0.9, 0.9375, 1.0]:
    print(f"{coverage:8.2%} {all_or_nothing(coverage):15.1%} {leaky(coverage):9.1%}")
# threshold with perfect vaccine: 75%; coverage needed at E=80%: 93.8%
# coverage   all-or-nothing   leaky   (share of whole population infected)
#    0.00%           98.1%     98.1%
#   50.00%           52.8%     67.5%
#   80.00%           19.5%     33.7%
#   90.00%            5.8%     12.5%
#   93.75%            0.7%      1.1%
#  100.00%            0.0%      0.0%
```

	ALL-OR-NOTHING	LEAKY
Vaccinated people	some fully protected, some not at all	all partly protected
At 50% coverage (R0 = 4, E = 80%)	52.8% of the population infected	67.5% infected

	ALL-OR-NOTHING	LEAKY
At 90% coverage	5.8% infected	12.5% infected
At 93.75% coverage	0.7% infected	1.1% infected
Hidden danger	the unprotected look like the unvaccinated	vaccinated people can be infected and may pass it on

Both models bring the outbreak almost to nothing at about the 93.75 per cent the formula predicts, and both give little protection when coverage is low, but at middle coverages the leaky vaccine lets through noticeably more infection (12.5 against 5.8 per cent at 90 per cent coverage). One reason is that every vaccinated person in the leaky model remains a possible host, whereas in the all-or-nothing case the protected people drop out of the chain entirely. My simple leaky model also treats a vaccinated person who catches the disease as equally infectious as anyone else, an assumption that could overstate the damage. The point to keep is that one number, "E = 80 per cent", does not tell you how a vaccine fails.

When the Average Lies

The formula assumes everyone mixes equally. Real communities do not: children share schools, families share homes, and some neighbourhoods have lower coverage than others. If non-vaccinators cluster together, the average coverage can be above the threshold while the infection finds a pocket in which R is still above 1. The third exercise puts numbers on that. It is also why public-health writing looks at coverage by district and not only the national figure; this is the course's own point, not a quotation from the article. How Epidemics Spread 7 returns to uneven contact patterns with networks.

> Common mistakes
> Forgetting the vaccine's effectiveness and using 1 − 1/R0 as the coverage; reading the threshold as a switch (just above it large outbreaks stop, just below it they are possible but not certain, as How Epidemics Spread 6 will show); assuming the threshold is fixed for a disease (it depends on R0, which depends on behaviour and place); calling the unvaccinated "protected" as if that were their own immunity (it lasts only while others stay immune); averaging coverage over places that do not mix; and treating the simple formula as exact for real populations, which have varying contact patterns, age structure and waning immunity, the subject of How Epidemics Spread 5.

EXERCISE 1

Take R0 = 4 and a vaccine with E = 90% (assumed, all-or-nothing). Compute the coverage the formula says is needed, then simulate coverages 10 points below it, 5 points below, exactly at it, and 5 points above. Print the peak number infectious and the share of the population infected for each.

View solution

EXERCISE 2

With R0 = 6 and E = 80% (assumed), show that vaccinating everyone cannot prevent a (smaller) outbreak, by simulating coverages of 50%, 80% and 100%. Then show that a vaccine with effectiveness 5 points above the threshold does stop it at full coverage.

View solution

EXERCISE 3

Build two communities of 5,000, one with 98% coverage and one with 70% (average 84%), with a perfect vaccine and R0 = 5 (threshold 80%, all assumed). Let each person have a share of contacts inside their own community of 50%, 80% and 95%, seed one case in each community in turn, and report how many people each community loses (not counting the first case) and the share of everyone infected.

View solution

QUICK REFERENCE

- **Herd immunity threshold:** $p_c = 1 - 1/R0$ (immune share needed, from $R0 \times S/N = 1$)
- **With vaccine effectiveness E:** coverage needed = $(1 - 1/R0) / E$; impossible if $E < 1 - 1/R0$
- **Examples:** R0 2, 3, 5, 12, 18 give thresholds 50%, 67%, 80%, 92%, 94% (measles R0 is 12 to 18 per the article)
- **Overshoot:** a natural epidemic passes the threshold; with R0 = 2.5 it infected 89.3% against a 60% threshold
- **Indirect protection:** at the threshold the unvaccinated are infected far less often
- **Failure modes:** all-or-nothing and leaky vaccines can have the same E but different outcomes

- **Assumes:** equal mixing; averages can hide low-coverage pockets

COMING NEXT

How Epidemics Spread 5 is about adding realism: latent periods (SEIR), deaths, waning immunity, births, endemic diseases.

How Epidemics Spread

Chapter 5 · Adding Realism

The model in How Epidemics Spread 3 had three boxes, and How Epidemics Spread 4 used it to find the herd immunity threshold. Real outbreaks break its rules in familiar ways. People are infected a few days before they can infect anyone else. Some die. Immunity can fade. Babies are born, and each new baby is a fresh susceptible. This chapter adds those four things one at a time and watches what each does. Two lessons emerge: some changes alter only the *timing* of an epidemic, and some change the long-run *outcome*, including the possibility that a disease never leaves and becomes **endemic**.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The model equations with births and deaths ($R0 = \beta/(\mu + \gamma)$, a disease-free state, and an endemic state with the susceptible share at $(\gamma + \mu)/\beta$) come from the Wikipedia article on compartmental models in epidemiology, fetched with a tool that often returns truncated pages or model-written summaries; this fetch covered only the first 100,000 of about 229,000 characters and stopped before the SEIR section, so treat the wording as a paraphrase and the article as unchecked for SEIR and SIRS equations. I found no conflict between sources, but I only had the one. The SEIR and SIRS rules, the death rule, and the formulas for the endemic level with waning immunity or newborn vaccination are **my own algebra on the standard set-up**, checked here only against the simulations; they are inferences, not quotations. Every population size, R0, latent period, infectious period, fatality share, lifetime and waning time in the code is *assumed* or *illustrative*, not an estimate for any real disease or place.

A Latent Period: the E Box

After catching a disease, a person is usually not infectious straight away. The Wikipedia article notes that SEIS and SEIR models include a latent period during which the person is not infectious. The extra compartment is called E, for **exposed**, and the model is called **SEIR**: people move S to E at the infection rate (exactly as before), then E to I at a rate $\sigma = 1 /$ (latent period), then I to R at γ. The code below runs $R0 = 2.5$ with a 5-day infectious period and latent periods of 0 (the plain SIR of earlier chapters), 2, 4 and 8 days, and reports the day of the peak, the peak number *infectious*, and the share ever infected. The step is 0.05 days, and this is just numerical stepping, no calculus.

```python
# SEIR: adding a latent period. R0 = 2.5 and a 5-day infectious period are assumed.
population = 10_000
gamma = 0.2            # assumed: average infectious period 5 days
r0 = 2.5
beta = r0 * gamma
dt = 0.05

def run(latent_days):
    """latent_days = 0 means no latent compartment (plain SIR)."""
    S, E, I, R = population - 1.0, 0.0, 1.0, 0.0
    if latent_days > 0:
        sigma = 1 / latent_days
    peak_I, peak_t, t = I, 0.0, 0.0
    while (E + I) >= 0.01 and t < 2000:
        new_exposed = beta * S * I / population * dt
        if latent_days > 0:
            new_infectious = sigma * E * dt
        else:
            new_infectious = new_exposed          # skip the E stage
            E = 0.0
        new_rec = gamma * I * dt
        S -= new_exposed
        if latent_days > 0:
            E += new_exposed - new_infectious
        I += new_infectious - new_rec
        R += new_rec
        t += dt
        if I > peak_I:
            peak_I, peak_t = I, t
    return peak_t, peak_I, R / population

print("latent days   peak day   peak infectious   share ever infected")
for latent in [0, 2, 4, 8]:
    day, peak, share = run(latent)
    print(f"{latent:11d} {day:10.0f} {peak:17.0f} {share:21.1%}")
# latent days   peak day   peak infectious   share ever infected
#           0         32              2343                 89.3%
#           2         55              1653                 89.3%
#           4         72              1279                 89.3%
#           8        102               886                 89.3%
```

The final size is the same in all four runs: 89.3 per cent, the figure for R0 = 2.5 in How Epidemics Spread 3 and How Epidemics Spread 4. What changes is the schedule. A latent period of 8 days moves the peak from day 32 to day 102 and cuts the peak number infectious from 2,343 to 886. In these runs R0 is untouched (it is still β/γ, the number of people one case infects), but each step along the chain now takes longer, so the epidemic grows more slowly, spreads the same total over a longer time, and at any instant fewer people are infectious (many are waiting in E). That is my reading of the output, not a result from the article. Reading for practice: the same number of eventual cases can mean a very different load at the peak, which is why timing matters for hospitals.

	SIR (HOW EPIDEMICS SPREAD 3)	SEIR (THIS CHAPTER)
Compartments	S, I, R	S, E, I, R
Path of a case	infected and infectious at once	infected, wait (latent), then infectious
Peak day (R0 = 2.5)	day 32	day 55 / 72 / 102 for 2 / 4 / 8 days latent
Final share infected	89.3%	89.3% in each of those runs
Best for	diseases with little delay	diseases with an incubation gap before infectiousness

Adding Deaths

So far people left the infectious stage into R and the population stayed the same size. A simple way to add deaths is to say that a share f of the people leaving the infectious stage die rather than recover. The code does that, with the SEIR rules above (3-day latent, 5-day infectious). It also divides by the number still alive when working out how often a susceptible meets an infectious person. That is a modelling choice, **assumed**. The fatality shares 0.1, 1 and 5 per cent are illustrative.

```
# Deaths: a share f of the people who leave the infectious stage die instead of recoveri
# R0 = 2.5, a 3-day latent and 5-day infectious period, and every f below are assumed.
population = 10_000
gamma, sigma, r0, dt = 0.2, 1 / 3, 2.5, 0.05
```

```
beta = r0 * gamma

def run(f):
    S, E, I, R, D = population - 1.0, 0.0, 1.0, 0.0, 0.0
    t = 0.0
    while (E + I) >= 0.01 and t < 2000:
        alive = population - D              # the dead no longer take part in mixing
        new_exposed = beta * S * I / alive * dt
        new_infectious = sigma * E * dt
        leaving = gamma * I * dt
        S -= new_exposed
        E += new_exposed - new_infectious
        I += new_infectious - leaving
        R += (1 - f) * leaving
        D += f * leaving
        t += dt
    return D, (R + D) / population

print("fatality share f   deaths per 10,000   share ever infected")
for f in [0.001, 0.01, 0.05]:
    deaths, ever = run(f)
    print(f"{f:15.1%} {deaths:19.1f} {ever:21.1%}")
# fatality share f   deaths per 10,000   share ever infected
#            0.1%                 8.9                 89.3%
#            1.0%                89.5                 89.5%
#            5.0%               450.2                 90.0%
```

Deaths scale almost directly with f: 8.9, 89.5 and 450.2 per 10,000. The share ever infected barely moves (89.3, 89.5 and 90.0 per cent). The 90.0 comes from my choice to divide by the survivors: as people die, each remaining susceptible meets a slightly larger share of infectious people, so a

of the epidemic. It is also a different thing from a deadly disease that kills people before they can infect others, which would shorten the infectious period and so lower R0, an effect not modelled here.

Births, Waning Immunity and the Endemic State

The big change comes from a longer horizon. Over decades, people are born and die for other reasons too. The article's version of the model has a birth rate and a death rate; with them, R0

becomes β/(μ + γ), because people leave the infectious state by dying as well as recovering. It gives two long-run outcomes: if R0 is 1 or less, the disease dies out and the population returns to the all-susceptible, disease-free state; if R0 is above 1 and some infection is present, the system tends to an **endemic** state in which the disease stays permanently, with the susceptible share settling at (γ + μ)/β, which is 1/R0. The disease stays because births keep refilling S.

The code compares two worlds with R0 = 2.5, a population of 1,000,000 and people living on average 70 years (birth rate equal to death rate, so the population stays constant). In the first, immunity lasts for life, so only births refill S. In the second, immunity wanes: recovered people return to S after 2 years on average, giving the **SIRS** model (mentioned briefly in the article, with no equations in the portion I saw). It prints, for each 25-year window, the lowest, highest and mean number infectious per 100,000, plus the endemic level my algebra predicts. Setting the rates of change to zero and solving is just algebra: for births only it gives μ(R0 − 1)/β, the article's endemic infectious share; with waning ω, it gives (1 − 1/R0) / (1 + γ/(μ + ω)).

```
# Births and waning immunity: SIR with births vs SIRS with births (all values assumed).
population = 1_000_000
gamma, r0, dt = 0.2, 2.5, 0.25
mu = 1 / (70 * 365)                  # birth rate = death rate; mean lifetime 70 years
beta = r0 * (gamma + mu)             # so that beta / (gamma + mu) = 2.5

def run(omega, years=150):
    S, I, R = 0.9 * population, 100.0, 0.1 * population
    steps_per_year = int(365 / dt)
    windows = {}                      # 25-year windows: lowest, highest, mean infectiou
    for year in range(years):
        for _ in range(steps_per_year):
            infect = beta * S * I / population
            S += (mu * population - mu * S - infect + omega * R) * dt
            I += (infect - (gamma + mu) * I) * dt
            R += (gamma * I - (mu + omega) * R) * dt
            window = year // 25
            lo, hi, total, n = windows.get(window, (1e18, 0.0, 0.0, 0))
            windows[window] = (min(lo, I), max(hi, I), total + I, n + 1)
    return windows, S / population

for label, omega in [("births only (lifelong immunity)", 0.0), ("waning, mean 2 years",
    print(label)
    windows, s_end = run(omega)
    for w, (lo, hi, total, n) in windows.items():
        print(f"  years {25 * w:3d}-{25 * w + 24:3d}: infectious per 100,000: lowest {lo
```

```
    print(f"  susceptible share at the end: {s_end:.3f}   (1/R0 = {1 / r0:.3f})")
    if omega == 0:
        formula = mu * (r0 - 1) / beta           # (mu / beta)(R0 - 1)
    else:
        formula = (1 - 1 / r0) / (1 + gamma / (mu + omega))
    print(f"  formula for the endemic level: {formula * 100_000:.1f} per 100,000")
# births only (lifelong immunity)
#   years   0- 24: infectious per 100,000: lowest     0.000, highest 17832.605, mean    4
#   years  25- 49: infectious per 100,000: lowest     0.000, highest     0.000, mean
#   years  50- 74: infectious per 100,000: lowest     0.000, highest  4139.665, mean    1
#   years  75- 99: infectious per 100,000: lowest     0.000, highest  1842.713, mean    1
#   years 100-124: infectious per 100,000: lowest     0.000, highest  1041.567, mean    1
#   years 125-149: infectious per 100,000: lowest     0.000, highest   669.340, mean    1
#   susceptible share at the end: 0.346   (1/R0 = 0.400)
#   formula for the endemic level: 11.7 per 100,000
# waning, mean 2 years
#   years   0- 24: infectious per 100,000: lowest     0.000, highest 18157.649, mean   44
#   years  25- 49: infectious per 100,000: lowest   419.843, highest   419.886, mean   41
#   years  50- 74: infectious per 100,000: lowest   419.813, highest   419.843, mean   41
#   years  75- 99: infectious per 100,000: lowest   419.792, highest   419.813, mean   41
#   years 100-124: infectious per 100,000: lowest   419.778, highest   419.792, mean   41
#   years 125-149: infectious per 100,000: lowest   419.767, highest   419.778, mean   41
#   susceptible share at the end: 0.400   (1/R0 = 0.400)
#   formula for the endemic level: 419.7 per 100,000
```

The two worlds behave very differently. With waning immunity the system settles within 25 years at 419.8 infectious per 100,000, a steady background level that matches the formula (419.7) and keeps the susceptible share at exactly 1/R0 = 0.400. With lifelong immunity the disease does not settle quickly. The first wave peaks at 17,832.6 per 100,000, and then infection fades to a lowest and highest of under 0.0005 per 100,000 in the years 25 to 49, which in a population of one million is far less than a single person. Only as births rebuild S does it rebound (to 4,139.7, then 1,842.7, 1,041.6 and 669.3 per 100,000 in later windows). The waves shrink and slowly home in on the formula's 11.7 per 100,000 (the final-window mean is 14.9 and the susceptible share has reached 0.346, not yet 0.400).

The deterministic model keeps a vanishing fraction of a person infected and lets the disease recover from it. A real population has whole people. My inference (not a result from the article) is that those deep troughs would end in real extinction, with the disease coming back only if someone brought it in from outside. This is where chance matters, and How Epidemics Spread 6 is about it.

	EPIDEMIC (EARLIER CHAPTERS)	ENDEMIC (THIS CHAPTER)
Time scale	weeks to months	years to decades
Susceptible supply	only the starting population	refilled by births and by waning immunity
Long-run susceptible share	falls and stays low	settles near 1/R0 (0.400 for R0 = 2.5)
Shape	one wave, then it ends	waves that shrink, or a steady background level
Effective reproduction number	falls below 1 as S falls	hovers around 1 (R0 × S/N = 1 at the endemic state)

Why Vaccination Looks Different in an Endemic World

How Epidemics Spread 4 vaccinated people before one outbreak. When births keep arriving, vaccination has to be an ongoing programme, and the same threshold appears in a new form: if a share p of newborns is protected, the formula for the endemic level becomes μ(R0(1 − p) − 1)/β, which falls to zero when p reaches 1 − 1/R0. This is my own derivation, checked by Exercise 3. Notice how a plain 1 − 1/R0 still governs elimination, but here it applies to each new generation of susceptibles.

> Common mistakes
> Believing a latent period changes the final size (in these runs it only changed the timing); counting people in E as infectious (they are not, and the peak reported is of I only); forgetting that R0 changes to β/(μ + γ) once deaths or births are in the model; reading an endemic level of 11.7 per 100,000 as a promise that a real town always has cases (very small populations die out by chance); mixing up a *low* endemic level with a *harmless* disease; and using one set of assumed numbers as if it were data (every rate here is illustrative).

EXERCISE 1

Run the SEIR model with R0 = 3 (assumed) and latent periods of 2, 4 and 8 days with a 5-day infectious period, population 10,000. Print the day of the peak, the peak number infectious and the share ever infected for each, and say which of the three numbers changes with the latent period and which does not.

View solution

EXERCISE 2

Using the SIRS model with births in a population of 1,000,000 (R0 = 2.5, 70-year lifetime, assumed), let immunity last 0.5, 1 and 3 years. Simulate 60 years and compare the infectious level per 100,000 at the end with the formula $(1 - 1/R0) / (1 + \gamma/(\mu + \omega))$.

View solution

EXERCISE 3

Start from the unvaccinated endemic state (R0 = 2.5, lifelong immunity, assumed). Switch on a perfect vaccine for 0, 30, 50, 60 and 70 per cent of newborns, run 100 years, and compare the final infectious level per 100,000 with the prediction $\mu(R0(1 - p) - 1)/\beta$. Which coverages eliminate the disease?

View solution

QUICK REFERENCE

- **SEIR:** S to E at the infection rate, E to I at $\sigma = 1$ / latent period, I to R at γ
- **Latent period effect:** later and lower peak; final size unchanged in these runs (R0 = 2.5: 89.3%)
- **Deaths:** a share f leaves I by dying; the toll scales with f, the epidemic barely changes
- **With births and deaths:** $R0 = \beta/(\mu + \gamma)$; disease dies out if $R0 \leq 1$, otherwise tends to an endemic state
- **Endemic state:** susceptible share $(\gamma + \mu)/\beta = 1/R0$; infectious share $\mu(R0 - 1)/\beta$ with lifelong immunity
- **SIRS (waning ω):** infectious share $(1 - 1/R0) / (1 + \gamma/(\mu + \omega))$
- **Newborn vaccination p:** level $\mu(R0(1 - p) - 1)/\beta$, zero once $p \geq 1 - 1/R0$
- **Assumes:** well-mixed population, constant rates, illustrative numbers

COMING NEXT

How Epidemics Spread 6 is about chance and small outbreaks: stochastic simulation, branching processes, super-spreaders, why outbreaks sometimes fizzle.

How Epidemics Spread

Chapter 6 · Chance and Fizzles

A traveller steps off a train in a small town, carrying an infection that is, on paper, perfectly capable of causing an epidemic. Most of the time, nothing happens. The traveller recovers, perhaps passes it to one or two people, and the chain of cases quietly ends. Every so often the same arrival lights a fire. The models in How Epidemics Spread 3, How Epidemics Spread 4 and How Epidemics Spread 5 could not tell these two stories apart, because they tracked averages and gave the same answer every time. How Epidemics Spread 5 ended by suggesting that the deep troughs in its endemic runs would, in a real population, end in extinction. This chapter puts chance into the simulations and asks three questions: how often does a single case fizzle out, why does the answer depend on more than R0, and what do super-spreaders change?

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed (random seeds are fixed, so you should get the same numbers). The criterion for branching processes (extinction certain if the mean number of offspring is 1 or less, uncertain if above 1, and the extinction probability found by starting at 0 and repeatedly applying the offspring generating function) comes from the Wikipedia article on branching processes, fetched with a tool that often returns truncated pages or model-written summaries, so treat the wording as a paraphrase. The super-spreading set-up (a gamma-distributed individual reproductive number, giving negative binomial offspring with mean R0 and dispersion k, with k = 1 the geometric and k going to infinity the Poisson case, and a reported SARS estimate of k = 0.16 with a 90% interval of 0.11 to 0.64) is from Lloyd-Smith and colleagues' 2005 paper in *Nature*; I saw it only through search-result summaries, not the full paper, so check the paper before quoting the numbers. Those summaries **conflict in emphasis** on one point: some retellings speak of a "20/80 rule", while the paper itself is reported as finding that no general 20/80 rule holds for directly transmitted diseases, so I do not claim one. The statement that a one-case outbreak fizzles with probability 1/R0 in the stochastic SIR model, and the formulas for the geometric and negative binomial cases, are **my own algebra**, checked here only against the simulations; they are inferences, not quotations. Every population size, R0, k, run count and cap in the code is *assumed* or *illustrative*, not an estimate for any real disease.

Putting Chance into the SIR Model

In a stochastic simulation, people are counted as whole individuals and each step is a random event. With S susceptibles and I infectious people, the next event is either an infection (at rate βSI/N) or a recovery (at rate γI). The chance that it is an infection is the first rate divided by the sum of the two. The code below ignores the exact time of each event, because we only want to know *how many* people end up infected. It starts with a single case in a population of 1,000, repeats the whole outbreak 2,000 times for each R0, and calls a run "fizzled" if 20 or fewer people were ever infected. That threshold of 20 is an arbitrary choice for this chapter. There is no calculus here, only a loop and a random number.

```
# Stochastic SIR: one random event at a time. Population, rates and run counts are assum
import random

gamma = 0.2                     # assumed: average infectious period 5 days
population = 1000

def outbreak(r0):
    """Start with one case; return how many people were ever infected."""
    beta = r0 * gamma
    S, I, ever = population - 1, 1, 1
    while I > 0:
        infect_rate = beta * S * I / population
        recover_rate = gamma * I
        if random.random() < infect_rate / (infect_rate + recover_rate):
            S -= 1
            I += 1
            ever += 1
        else:
            I -= 1
    return ever

random.seed(1)
runs = 2000
print("R0    fizzled (<=20 cases)   1/R0    mean size of the rest")
for r0 in [0.8, 1.5, 2.5, 4.0]:
    sizes = [outbreak(r0) for _ in range(runs)]
    small = [s for s in sizes if s <= 20]
    big = [s for s in sizes if s > 20]
    mean_big = sum(big) / len(big) / population if big else float("nan")
    print(f"{r0:3.1f} {len(small) / runs:20.1%} {min(1, 1 / r0):7.1%} {mean_big:20.1%}")
# R0    fizzled (<=20 cases)   1/R0    mean size of the rest
# 0.8                95.1%  100.0%                4.3%
# 1.5                66.5%   66.7%               54.7%
```

```
# 2.5                41.1%   40.0%                89.2%
# 4.0                25.1%   25.0%                98.0%
```

At R0 = 2.5, the plain model of How Epidemics Spread 3 says the epidemic always happens and reaches 89.3 per cent of the population. The simulation says that about 41 runs in 100 fizzle, and the others reach 89.2 per cent on average, almost exactly the deterministic figure. At R0 = 1.5 and 4.0 the picture is the same: a mixture of fizzles and full epidemics, and the share that fizzle is almost exactly 1/R0 (66.5 against 66.7 per cent, and 25.1 against 25.0 per cent). At R0 = 0.8 nearly every run fizzles; the 4.9 per cent that passed 20 cases are unlucky long chains, and they average only 4.3 per cent of the population. Run the deterministic rule (repeating $z = 1 - \exp(-R0 \times z)$) and you get about 58.3 per cent for R0 = 1.5 and 98.0 per cent for R0 = 4.0; the simulated means are 54.7 and 98.0, so the model gets the size of a real outbreak roughly right, and the fizzle rate is the part the earlier chapters could not see. That 1/R0 rule is my inference, and the next section explains why it should hold.

	DETERMINISTIC (HOW EPIDEMICS SPREAD 3)	STOCHASTIC (THIS CHAPTER)
Unit	a fraction of the population	whole people
Same inputs, two runs	identical output	different output every run (unless the seed is fixed)
One case, R0 = 2.5	always an epidemic	about 4 runs in 10 fizzle
Result reported	one curve	a spread of outcomes, a fizzle rate, a mean size
Cost	one run	hundreds or thousands of runs

Branching Processes: the Early Days

the others. A case infects some number of people, each of whom infects some number, and so on. That is a **branching process**. If the average number of offspring is 1 or less, the chain dies out for certain; if it is above 1, it may still die out, with a probability q below 1 (this is the criterion from the article). To find q, write the probability of having k offspring, build the function

f(q) = sum of P(k) × q^k, and repeat q ← f(q) starting from q = 0 until the number stops changing. The method is just repeated substitution.

The offspring distribution matters. Two natural choices both have mean R0. With **Poisson** offspring (many independent contacts, each with a small chance of transmission) f(q) = exp(R0(q − 1)). In the stochastic SIR model, each step before the susceptibles run low is a race between an infection and the case's own recovery, so the number of people one case infects is **geometric**: f(q) = 1 / (1 + R0(1 − q)). Solving q = f(q) for that case gives q = 1/R0, which explains the fizzle rates above (my algebra). The code checks both against direct simulation, calling a chain "taken off" once it has reached 500 people (an assumed cap).

```
# Branching process: each case independently infects a random number of others.
# Poisson and geometric offspring, both with mean R0. R0 values and run counts are assum
import math
import random

def poisson(lam):
    """Knuth's method, in chunks so exp(-lam) never underflows."""
    total = 0
    while lam > 30:
        total += poisson(30)
        lam -= 30
    limit, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= random.random()
        if p <= limit:
            return total + k
        k += 1

def geometric(mean):
    """P(k) = (1 - p) * p**k, with p chosen so the mean is `mean`."""
    p = mean / (1 + mean)
    return int(math.log(1 - random.random()) / math.log(p))

def extinct_probability(generating, steps=2000):
    """Smallest solution of q = generating(q), found by repeating from q = 0."""
    q = 0.0
    for _ in range(steps):
        q = generating(q)
    return q

def escapes(draw, cap=500):
```

```
    """True if the chain of cases reaches `cap` people before dying out."""
    active, total = 1, 1
    while active > 0 and total < cap:
        new = sum(draw() for _ in range(active))
        active = new
        total += new
    return active > 0

random.seed(2)
runs = 4000
print("R0    Poisson: formula  simulated    geometric: formula  simulated")
for r0 in [0.9, 1.2, 1.5, 2.5, 4.0]:
    q_pois = extinct_probability(lambda q: math.exp(r0 * (q - 1)))
    q_geom = extinct_probability(lambda q: 1 / (1 + r0 * (1 - q)))
    sim_p = 1 - sum(escapes(lambda: poisson(r0)) for _ in range(runs)) / runs
    sim_g = 1 - sum(escapes(lambda: geometric(r0)) for _ in range(runs)) / runs
    print(f"{r0:3.1f} {q_pois:17.3f} {sim_p:10.3f} {q_geom:20.3f} {sim_g:10.3f}")
# R0    Poisson: formula  simulated    geometric: formula  simulated
# 0.9             1.000      1.000                1.000      0.999
# 1.2             0.686      0.681                0.833      0.837
# 1.5             0.417      0.430                0.667      0.663
# 2.5             0.107      0.106                0.400      0.404
# 4.0             0.020      0.017                0.250      0.247
```

The formula and the simulation agree (to within noise from 4,000 runs each; the 0.999 at R0 = 0.9 is a rare chain that reached the cap of 500 before dying out). The surprise is how far apart the two columns are. With R0 = 2.5 the Poisson case fizzles only 10.7 per cent of the time, while the geometric case fizzles 40.0 per cent of the time, nearly four times as often. Both have exactly the same average number of secondary cases. The difference is the spread: the geometric distribution has many more zeros and a few large counts. **R0 alone does not say how likely an outbreak is to fizzle.** The spread of the offspring counts matters too.

Super-Spreaders: When Transmission Is Uneven

Real transmission is often more uneven than either of those. The 2005 paper models each person's own reproductive number as a random value with mean R0, gamma-distributed with a dispersion parameter k; the resulting offspring counts are negative binomial. Small k means very uneven: most cases infect nobody and a few infect many. k = 1 gives the geometric case and k going to infinity gives Poisson. The paper reports k of about 0.16 for SARS. The code samples

from this model with R0 = 2.5 and several assumed values of k, and reports four things: the share of cases who infect nobody, the fizzle probability from the formula $q = (1 + R0(1 - q)/k)^{-k}$ (same repeated substitution; my derivation from the standard generating function, checked below), the fizzle rate from direct simulation, and the share of all onward infections caused by the 20 per cent most infectious cases.

```
# Super-spreading: negative binomial offspring with mean R0 and dispersion k (small k = r
# R0 = 2.5 and every k below are assumed, except that k = 0.16 echoes a published SARS es
import math
import random

r0 = 2.5

def poisson(lam):
    total = 0
    while lam > 30:
        total += poisson(30)
        lam -= 30
    limit, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= random.random()
        if p <= limit:
            return total + k
        k += 1

def offspring(k):
    """Gamma-Poisson mixture: each case gets its own mean, then a Poisson draw."""
    if k == float("inf"):
        return poisson(r0)
    return poisson(random.gammavariate(k, r0 / k))

def generating(q, k):
    if k == float("inf"):
        return math.exp(r0 * (q - 1))
    return (1 + r0 * (1 - q) / k) ** (-k)

def extinct_probability(k):
    q = 0.0
    for _ in range(3000):
        q = generating(q, k)
    return q
```

```
def escapes(k, cap=500):
    active, total = 1, 1
    while active > 0 and total < cap:
        active = sum(offspring(k) for _ in range(active))
        total += active
    return active > 0

random.seed(3)
runs, sample = 3000, 100_000
print("   k   infect nobody   fizzle: formula  simulated   share of spread from top 20%"
for k in [0.1, 0.3, 1.0, 10.0, float("inf")]:
    draws = sorted((offspring(k) for _ in range(sample)), reverse=True)
    top_share = sum(draws[: sample // 5]) / sum(draws)
    nobody = sum(1 for d in draws if d == 0) / sample
    sim = 1 - sum(escapes(k) for _ in range(runs)) / runs
    label = "inf" if k == float("inf") else f"{k:.1f}"
    print(f"{label:>4} {nobody:14.1%} {extinct_probability(k):16.3f} {sim:10.3f} {top_sh
#    k   infect nobody   fizzle: formula  simulated   share of spread from top 20%
#  0.1          72.2%            0.861      0.859                    96.5%
#  0.3          51.1%            0.675      0.678                    79.0%
#  1.0          28.6%            0.400      0.398                    58.0%
# 10.0          10.7%            0.144      0.152                    41.3%
#  inf           8.3%            0.107      0.113                    38.9%
```

As k falls, everything shifts in the same direction. At k = 0.1, 72.2 per cent of cases infect nobody and the top fifth of cases cause 96.5 per cent of the onward spread, yet the average is still 2.5. A single introduction fizzles 86 times in 100, against about 11 times in 100 for Poisson. The formula and simulation agree throughout (0.861 and 0.859 at k = 0.1, 0.400 and 0.398 at k = 1). Notice also the last column: even with Poisson offspring the top 20 per cent cause 38.9 per cent of spread, simply because some cases happen to have more offspring than others. These figures come from my simulation with assumed parameters; they are not the paper's results, and they are not a general 20/80 rule.

The practical lesson is modest but important. When transmission is uneven, a single introduction is much less likely to start an epidemic, but when the arrival is a super-spreading event the same R0 can be reached in one step. Control measures aimed at the unusually infectious events can then matter more than measures aimed at everyone equally. That last sentence is an inference from the model, not a result I tested; How Epidemics Spread 7 and How Epidemics Spread 9 will test targeted control properly.

	POISSON (K = INFINITY)	GEOMETRIC (K = 1)	VERY UNEVEN (K = 0.1)
Mean offspring (R0)	2.5	2.5	2.5
Cases who infect nobody	8.3%	28.6%	72.2%
One case fizzles	about 11%	40%	about 86%
Share of spread from top 20% of cases	38.9%	58.0%	96.5%
Typical story	almost every introduction grows	a coin flip with a bias	many dead ends, a few explosive clusters

Common mistakes

Believing that R0 > 1 guarantees an epidemic (it only makes one possible; the fizzle chance can be large); reading “the epidemic fizzled” as proof the disease is mild or that a control measure worked (many untreated introductions fizzle anyway); using a branching-process formula late in an outbreak, when the susceptibles are no longer plentiful and cases are no longer independent; treating the 20/80 idea as a law (the paper reports it does not hold in general); running one simulation and quoting its result (a single run is one random story, so report a rate from many runs); and using the illustrative numbers here (R0, k, the 500-person cap, the 20-case threshold) as if they were estimates for a real disease.

EXERCISE 1

Run the stochastic SIR model with R0 = 2 (assumed) for populations of 50, 200 and 1,000, 2,000 runs each. Print the share of runs that fizzle (10 or fewer cases) and the mean share infected in the rest, and compare them with 1/R0 and with the deterministic final share.

View solution

EXERCISE 2

With Poisson offspring and R0 = 1.1, 1.5, 2 and 3 (assumed), find the extinction probability q by repeated substitution. Print the chance that 1, 3 and 5 independent introductions take off (1 − q to the power of the number of seeds), and check the 1-seed and 3-seed values by simulation.

View solution

EXERCISE 3

For R0 = 1.2 and 2.5 and k = 0.16, 1 and infinity (all assumed), print the chance a case infects nobody, the fizzle probability from the formula, and the fizzle rate from 3,000 simulated chains. For which R0 does overdispersion change the fizzle chance more?

View solution

QUICK REFERENCE

- **Stochastic model:** whole people, one random event at a time; report rates from many runs, not one run
- **Branching process:** early on each case acts independently; extinction certain if mean offspring is 1 or less
- **Extinction probability q:** smallest solution of $q = f(q)$; repeat $q \leftarrow f(q)$ from $q = 0$
- **Poisson offspring:** $f(q) = \exp(R0(q - 1))$
- **Geometric offspring (stochastic SIR, one case):** $q = 1/R0$ (my algebra, matched by the runs)
- **Negative binomial (R0, k):** $q = (1 + R0(1 - q)/k)^{-k}$; small k means many dead ends and a few big spreaders; SARS reported near k = 0.16
- **With several introductions:** chance at least one takes off = $1 - q^{seeds}$
- **Assumes:** independent cases, illustrative numbers, an arbitrary cap and threshold

COMING NEXT

How Epidemics Spread 7 is about networks and contacts: random, clustered and hub networks, simulating spread on a graph, targeted control.

How Epidemics Spread

Chapter 7 · Networks and Hubs

Picture two towns where the average person has six close contacts. In the first, those contacts are scattered: a neighbour, a colleague in another street, a cousin across town. In the second, nearly everyone you know also knows each other, like a village where the same few families share every kitchen. In a third place, a handful of people (a barber, a bus driver, a teacher) know hundreds, while most people know three or four. The models in How Epidemics Spread 3, How Epidemics Spread 4 and How Epidemics Spread 5 treated all three places identically, because they assumed everyone mixes with everyone at random. How Epidemics Spread 6 added chance but kept that assumption. This chapter drops it and asks: if we keep the *average* number of contacts fixed, how much does the *shape* of the contact network change who gets infected, and does it change whom we should vaccinate?

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed (random seeds are fixed, so you should get the same numbers with the same Python random generator). The history is from search-result summaries, which are often truncated or model-written, so check the originals before quoting: the small-world model is Watts and Strogatz, *Nature* 393 (1998); preferential attachment is Barabási and Albert, *Science* 286 (1999); the finding that epidemics on a wide range of scale-free networks show no threshold is Pastor-Satorras and Vespignani, *Physical Review Letters* 86 (2001), which concerns idealised, very large networks; the treatment of an SIR outbreak on a network as a percolation problem, with a critical transmissibility, is Newman, *Physical Review E* 66 (2002); and acquaintance immunisation is Cohen, Havlin and ben-Avraham, *Physical Review Letters* 91 (2003). I did not see the formulas inside those papers, so the quantities below (the "excess degree" and R = T × excess degree) are **my own reasoning**, checked only against my simulations; they are inferences, not quotations. Where my simulation disagrees with that reasoning (clustered networks), I say so. The network sizes, mean degrees, transmission chances T, vaccination shares, run counts and the 20-case fizzle threshold are all *assumed* or *illustrative*, not estimates for any real disease or population. No calculus is used.

Three Kinds of Network

A network (graph) is people, called nodes, joined by links that mean "could pass the infection to". The number of links a person has is their **degree**. The code below builds three networks of

2,000 people, all with a mean degree of about 6, using only the standard library. A **random** network joins random pairs. A **clustered** network starts as a ring where everyone knows the three people on each side, then rewires 5 per cent of links to random far-away people, the recipe of the small-world model. A **hub** network grows one person at a time, each newcomer linking to three existing people with a bias towards those who already have many links (preferential attachment). The function `describe` reports the mean degree, the biggest hub, the clustering coefficient (the share of a person's pairs of contacts who are also linked to each other), and the **excess degree**: the average number of *further* contacts that a randomly followed contact has. It is computed as (mean of degree squared − mean degree) divided by mean degree. The excess degree is larger than the mean degree whenever degrees vary, because following a link is more likely to land on a well-connected person (the so-called friendship paradox).

```
# Three kinds of contact network, built from scratch. Sizes and mean degree are assumed.
import random

def random_network(n, mean_degree):
    """Random network: pick random pairs until there are n * mean_degree / 2 links."""
    nbrs = [set() for _ in range(n)]
    links = 0
    while links < n * mean_degree // 2:
        a, b = random.randrange(n), random.randrange(n)
        if a != b and b not in nbrs[a]:
            nbrs[a].add(b)
            nbrs[b].add(a)
            links += 1
    return nbrs

def clustered_network(n, mean_degree, rewire):
    """Ring of neighbours (everyone knows the people nearby); each link is rewired at ra
    nbrs = [set() for _ in range(n)]
    for i in range(n):
        for step in range(1, mean_degree // 2 + 1):
            j = (i + step) % n
            if random.random() < rewire:
                j = random.randrange(n)
                while j == i or j in nbrs[i]:
                    j = random.randrange(n)
            nbrs[i].add(j)
            nbrs[j].add(i)
    return nbrs
```

```python
def hub_network(n, m):
    """Growth with preferential attachment: each newcomer links to m people, favouring t
    nbrs = [set() for _ in range(n)]
    pool = []                       # a person appears here once per link they have
    targets = list(range(m))
    for new in range(m, n):
        for t in set(targets):
            nbrs[new].add(t)
            nbrs[t].add(new)
        pool.extend(targets)
        pool.extend([new] * m)
        targets = [random.choice(pool) for _ in range(m)]
    return nbrs

def describe(nbrs):
    n = len(nbrs)
    degrees = [len(x) for x in nbrs]
    mean = sum(degrees) / n
    excess = (sum(d * d for d in degrees) / n - mean) / mean   # average number of *othe
    tri, trip = 0, 0
    for i in range(n):
        d = degrees[i]
        if d >= 2:
            trip += d * (d - 1) / 2
            tri += sum(len(nbrs[i] & nbrs[j]) for j in nbrs[i]) / 2
    return mean, max(degrees), excess, tri / trip

random.seed(7)
n = 2000
networks = {
    "random":    random_network(n, 6),
    "clustered": clustered_network(n, 6, 0.05),
    "hubs":      hub_network(n, 3),
}
print("network     mean degree  biggest hub  excess degree  clustering")
for name, g in networks.items():
    mean, biggest, excess, clustering = describe(g)
    print(f"{name:10s} {mean:12.2f} {biggest:12d} {excess:14.2f} {clustering:11.3f}")
# network     mean degree  biggest hub  excess degree  clustering
# random             6.00           16           6.05       0.003
# clustered          6.00            9           5.05       0.507
# hubs               5.96          215          15.58       0.009
```

All three have a mean degree of 6, yet they look nothing alike. The random network has almost no clustering (0.003) and a biggest hub of only 16 contacts. The clustered network has clustering of 0.507: half of a person's pairs of contacts know each other. The hub network has someone with 215 contacts, and its excess degree of 15.58 is far above its mean of 5.96, while the random network's excess degree is 6.05, almost equal to its mean. The clustered network's excess degree is 5.05, just below 6, because nearly everyone has exactly 6 contacts and, of those, one is the person who infected you.

	RANDOM	CLUSTERED	HUBS
How it is built	random pairs linked	ring of neighbours, 5% rewired	newcomers favour well-connected people
Mean degree	6.00	6.00	5.96
Biggest hub	16	9	215
Excess degree	6.05	5.05	15.58
Clustering	0.003	0.507	0.009
Rough everyday picture (an analogy, not data)	scattered acquaintances	village or school year	a few very busy people among many quiet ones

Spreading on a Network

Now we spread an infection. Pick a random person, infect them, and give them one chance to infect each of their contacts, each with probability T (the **transmissibility**: the chance an infection passes along one link during the whole infectious period). Newly infected people get their own chance, and the outbreak stops when no new cases appear. Each run reports how many people were ever infected, and many runs give a fizzle rate, as in How Epidemics Spread 6. Two numbers are printed for comparison. The "simple R" is T times the mean degree, which is what a model with no network structure would use. The "network R" is T times the excess degree, my reasoning being that a newly infected person was reached through one link, so what matters is how many *other* links they have. This reuses the three networks built above, so run it in the same file, below the previous snippet.

```
# Spread on each network: every infected person gets one chance to infect each neighbour
```

```python
import random

def outbreak(nbrs, T, blocked=frozenset()):
    """One outbreak from a random unblocked person; returns how many people were ever in
    start = random.randrange(len(nbrs))
    while start in blocked:
        start = random.randrange(len(nbrs))
    reached, frontier = {start}, [start]
    while frontier:
        following = []
        for u in frontier:
            for v in nbrs[u]:
                if v not in reached and v not in blocked and random.random() < T:
                    reached.add(v)
                    following.append(v)
        frontier = following
    return len(reached)

def summarise(nbrs, T, runs, blocked=frozenset()):
    sizes = [outbreak(nbrs, T, blocked) for _ in range(runs)]
    big = [s for s in sizes if s > 20]
    fizzled = 1 - len(big) / runs
    share = sum(big) / len(big) / len(nbrs) if big else 0.0   # 0 if none took off
    return fizzled, share, sum(sizes) / runs / len(nbrs)

random.seed(8)
runs = 400
print("T      network     simple R=T*mean   network R=T*excess   fizzled   size if it ta
for T in [0.10, 0.25]:
    for name, g in networks.items():
        mean, biggest, excess, clustering = describe(g)
        fizzled, share, average = summarise(g, T, runs)
        print(f"{T:4.2f}  {name:10s} {T * mean:15.2f} {T * excess:20.2f} {fizzled:9.1%}
# T      network     simple R=T*mean   network R=T*excess   fizzled   size if it takes o
# 0.10  random                0.60                0.60     99.8%
# 0.10  clustered             0.60                0.50    100.0%
# 0.10  hubs                  0.60                1.56     88.5%
# 0.25  random                1.50                1.51     39.2%
# 0.25  clustered             1.50                1.26     77.2%
# 0.25  hubs                  1.49                3.89     42.8%
```

With T = 0.25 and a simple R of 1.5, the random network behaves like the well-mixed model of How Epidemics Spread 6: 39.2 per cent of outbreaks fizzle (the branching-process figure for Poisson offspring would be a little higher, near 42 per cent, but the offspring here are binomial and the networks finite, so I do not expect an exact match) and the rest reach 58.2 per cent of the population. The network R of 1.51 matches. The hub network has the same simple R but a network R of 3.89, and the fizzle rate is similar (42.8 per cent). The size of those that take off (57.9 per cent) is also similar to the random network, which I did not expect. A plausible reason (an inference I did not test) is that most people in the hub network have only a few links, and these cap how far an infection can reach. The difference shows at low transmissibility. At T = 0.10 the simple R is 0.60 for every network, so a model without structure predicts a certain fizzle; in the random network the outbreak does fizzle in 99.8 per cent of runs, but in the hub network 11.5 per cent of runs take off, and the average share infected is 0.8 per cent, against 0.1 per cent. Hubs make an epidemic possible when the average contact count says it should not be.

The clustered network goes the other way. Its network R (1.26 at T = 0.25) predicts spread, but 77.2 per cent of outbreaks fizzle, and those that pass 20 cases infect only 2.1 per cent of the population on average. Here my formula fails. A plausible reason (again an inference, untested) is that when your contacts know each other, many of the people you might infect are already infected, or about to be, by someone else, and the infection can only creep around the ring. The formula assumes that a contact's other contacts are new people, which is true in a random network and false in a clustered one. Exercise 1 explores this.

T = 0.25	RANDOM	CLUSTERED	HUBS
Simple R = T × mean degree	1.50	1.50	1.49
Network R = T × excess degree	1.51	1.26	3.89
Outbreaks that fizzle	39.2%	77.2%	42.8%
Average share infected, all runs	35.4%	0.7%	33.2%
Does the network R predict it?	yes	no (clustering breaks it)	partly (low T, yes; size, no)

Targeted Control

How Epidemics Spread 4 treated vaccination as a share of the population chosen at random. On a network we can choose *whom*. The code vaccinates 10 per cent of people, treated as removed from the network, by four rules: nobody; random people; the busiest people (most contacts);

and **acquaintance** vaccination, where you pick random people and vaccinate one of their contacts, which needs no knowledge of who the hubs are. The idea is credited to the 2003 paper by Cohen, Havlin and ben-Avraham (per search summaries, which describe it as lowering the immunisation threshold a great deal). The numbers below are from my simulation, not theirs. The outbreak is T = 0.25 on the random and hub networks. As before, this continues the same file.

```
# Targeted control: vaccinate 10% of people and compare how they are chosen. T and the 1
import random

def choose_random(nbrs, count):
    return frozenset(random.sample(range(len(nbrs)), count))

def choose_busiest(nbrs, count):
    ranked = sorted(range(len(nbrs)), key=lambda i: len(nbrs[i]), reverse=True)
    return frozenset(ranked[:count])

def choose_acquaintance(nbrs, count):
    """Pick a random person, then vaccinate one of their contacts (no need to know who t
    chosen = set()
    while len(chosen) < count:
        person = random.randrange(len(nbrs))
        if nbrs[person]:
            chosen.add(random.choice(sorted(nbrs[person])))
    return frozenset(chosen)

random.seed(9)
T, runs = 0.25, 400
count = len(networks["random"]) // 10
rules = [("nobody", lambda g, c: frozenset()), ("random", choose_random),
         ("busiest", choose_busiest), ("acquaintance", choose_acquaintance)]
print("average share infected, T = 0.25, 10% vaccinated")
print("network     " + "".join(f"{label:>14s}" for label, _ in rules))
for name in ["random", "hubs"]:
    g = networks[name]
    row = ""
    for label, rule in rules:
        fizzled, share, average = summarise(g, T, runs, rule(g, count))
        row += f"{average:14.1%}"
    print(f"{name:10s} {row}")
# average share infected, T = 0.25, 10% vaccinated
# network             nobody        random       busiest  acquaintance
```

```
# random          31.3%        18.3%        2.5%         15.5%
# hubs            35.9%        21.1%        0.1%          6.7%
```

Vaccinating 10 per cent at random cuts the average share infected from 31.3 to 18.3 per cent on the random network and from 35.9 to 21.1 per cent on the hub network. Vaccinating the busiest 10 per cent cuts it to 2.5 and 0.1 per cent. The hub network is the clearest case: removing the best-connected people takes away a large share of all the links at once. Acquaintance vaccination lands in between: 15.5 per cent on the random network and 6.7 per cent on the hub network, a better result than random vaccination, achieved without knowing who the hubs are. It works because a random contact is likelier to be a hub than a random person is (my explanation, consistent with the friendship paradox above). Exercise 3 asks how few people you need on the hub network.

Compare this with the herd-immunity threshold 1 − 1/R0 of How Epidemics Spread 4. That rule assumes random vaccination in a well-mixed population. On a hub network it overstates what is needed if you can find the hubs, and where you cannot find them it may not apply at all. The idea that a few well-chosen people can matter more than many random people is an inference from this model, not a claim about any real disease.

10% VACCINATED, T = 0.25	NOBODY	RANDOM	ACQUAINTANCE	BUSIEST FIRST
Random network	31.3%	18.3%	15.5%	2.5%
Hub network	35.9%	21.1%	6.7%	0.1%
Needs to know the hubs?	no	no	no	yes

Common mistakes

Treating the mean number of contacts as the whole story (two networks with the same mean can behave very differently); applying T × excess degree to a highly clustered network, where it overpredicts; quoting the “no epidemic threshold” result for scale-free networks as if it applied to every network with hubs (it concerns idealised, very large networks, and my 2,000-person hub network does show a gradual takeoff rather than none); assuming the busiest people are known in real life (finding them is the hard part, which is why acquaintance vaccination is interesting); forgetting that these networks are static, while real contacts change from day to day; reporting one run instead of a rate from many; and using the illustrative numbers here (T, 6 contacts, 2,000 people, 10 per cent vaccinated, the 20-case threshold) as if they were estimates for a real disease.

EXERCISE 1

Build clustered networks of 2,000 people with mean degree 6 and rewiring chances of 0, 0.01, 0.05, 0.2 and 1 (assumed), and spread an infection with T = 0.25 (assumed) through each, 300 runs. Print the clustering, the excess degree, the fizzle rate and the average share infected. Does the excess degree explain the differences?

View solution

EXERCISE 2

For a random network and a hub network (2,000 people, mean degree 6), print the predicted threshold T = 1 / excess degree, then the average share infected for T from 0.03 up to 0.25 (assumed values). Roughly where does each network start to take off, and does it match the prediction?

View solution

EXERCISE 3

On a hub network of 2,000 people with T = 0.25 (assumed), vaccinate 0, 1, 2, 3, 5, 8 and 10 per cent of people, chosen at random or busiest first. Print the average share infected for each. How small a share, chosen well, is enough to stop almost all spread?

View solution

QUICK REFERENCE

- **Network:** people as nodes, possible transmissions as links; degree = number of links
- **Random / clustered / hub:** scattered links; neighbours who know each other; a few people with very many links
- **Excess degree:** (mean of degree squared − mean degree) / mean degree; the further contacts of a contact
- **Transmissibility T:** chance of passing the infection along one link; simple R = T × mean degree, network R = T × excess degree (my reasoning)
- **Hubs:** lower the threshold; make outbreaks possible when the mean contact count says no
- **Clustering:** wastes transmissions on people already infected, so the excess-degree rule overpredicts

- **Targeted control:** busiest-first beats random; acquaintance vaccination beats random without finding the hubs
- **Assumes:** static networks, equal T on every link, illustrative numbers

COMING NEXT

How Epidemics Spread 8 is about measuring and estimating: estimating R from data, serial intervals, case-control and cohort studies, bias in testing.

How Epidemics Spread

Chapter 8 · Estimating R

Every model so far began with a number handed to us: an R of 2, a contact rate, a transmission chance. Nobody hands those to a health officer. What arrives each morning is a list of positive tests, a few hospital admissions, and some interviews with people who fell ill. In 1948 the Framingham Heart Study began following thousands of ordinary residents of one Massachusetts town for years, to see who developed heart disease and what they had in common (per search summaries, around 5,200 people were recruited). In 1950 Richard Doll and Austin Bradford Hill published a very different design in the *British Medical Journal*: they asked hospital patients with lung cancer, and matched patients with other illnesses, about their smoking, and compared the two groups. The first design starts with the exposure and waits for the illness; the second starts with the illness and looks back at the exposure. This chapter uses simulations to ask how we get from messy counts to a number like R, which studies answer which questions, and how the way we count can mislead us before any model is applied.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed (random seeds are fixed, so you should get the same numbers with the same Python random generator). The history and methods come from search-result summaries, which are often truncated or model-written, so check the originals before quoting: Doll and Hill, "Smoking and carcinoma of the lung: preliminary report", *British Medical Journal* 1950 (the summaries I saw gave its design as hospital patients with lung cancer compared with matched patients with other diseases, but I did not see its tables); the Framingham Heart Study began in 1948 as a prospective study; the link between a growth rate and R through the shape of the generation interval is Wallinga and Lipsitch, *Proceedings of the Royal Society B* 274 (2007); and the correction of test results for sensitivity and specificity is Rogan and Gladen, *American Journal of Epidemiology* 107 (1978). The search summaries gave me the general form R = 1 / M(−r), where M is the moment generating function of the generation interval. The special case for a gamma-shaped interval, R = (1 + r × scale) ^ shape, is **my own derivation** from that form (one search summary said the same), and it is checked here only against my simulations: treat it as an inference, not a quotation. I did not check the methods named in passing in this chapter's text (the Cori and Wallinga-Teunis ways of estimating R over time). Every population size, risk, sensitivity, specificity, test count, serial interval and true R is *assumed* or *illustrative*. No calculus is used; the only maths beyond arithmetic is logarithms and a least-squares line.

Estimating R from a Growth Curve

In the early days of an outbreak, with nobody immune and nothing changed, cases grow exponentially, as in How Epidemics Spread 2. If we plot the logarithm of daily cases against the day, the points fall near a straight line, and the slope of that line is the **growth rate** r per day. A growth rate is easy to measure. R is what we want, and it needs one more ingredient: how long it takes for one case to infect the next. The **serial interval** is the time between the start of symptoms in one case and the start of symptoms in the person they infected. It is measured from interviews. The **generation interval** is the time between the two infections, which usually cannot be seen. The two have the same mean in simple models but different spreads, and in practice the serial interval is used as a stand-in (a point I am relying on from general knowledge, not from a source I checked this session).

A shortcut is R = exp(r × mean interval). It is exact only if every infection takes exactly the mean time. When the interval varies, as it always does, the correct conversion is smaller. For an interval with a gamma shape (a flexible curve with a mean and a spread), my derivation gives R = (1 + r × scale) ^ shape. The code simulates outbreaks with a known R, fits the growth rate on days 10 to 20, and compares the two conversions. Each case has a random number of children (Poisson, mean R) and each child is infected a random whole number of days later. The serial interval is assumed gamma-shaped with a mean of 5 days and a standard deviation of 2.5 days. The four helper functions are reused in the exercises.

```
# Estimating R from early growth. Serial interval, window, sizes and true R are all assu
import math
import random
import statistics

# ---- helpers (the same four functions are reused in Exercises 1 and 2) ----
def poisson(lam):
    """Poisson random number (Knuth's method; fine for small lam)."""
    limit, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= random.random()
        if p <= limit:
            return k
        k += 1

def simulate(R, days, shape, scale, start=20):
    """New cases per day. Each case has Poisson(R) children; each child is infected a wh
```

```
    of days later, drawn from a gamma 'serial interval' (shape, scale) and rounded, mini
    daily = [0] * (days + 1)
    daily[0] = start
    for day in range(days + 1):
        for _ in range(daily[day]):
            for _ in range(poisson(R)):
                lag = max(1, round(random.gammavariate(shape, scale)))
                if day + lag <= days:
                    daily[day + lag] += 1
    return daily

def growth_rate(daily, first, last):
    """Slope of log(cases) against day, by least squares over days first..last (None if 
    if min(daily[first:last + 1]) == 0:
        return None
    xs = list(range(first, last + 1))
    ys = [math.log(daily[d]) for d in xs]
    mx, my = sum(xs) / len(xs), sum(ys) / len(ys)
    return sum((x - mx) * (y - my) for x, y in zip(xs, ys)) / sum((x - mx) ** 2 for x in 

def r_to_R(r, shape, scale):
    """Growth rate to R for a gamma serial interval: R = (1 + r * scale) ** shape."""
    return (1 + r * scale) ** shape

random.seed(3)
shape, scale = 4, 1.25            # assumed serial interval: gamma, mean 5 days, sd 2.5 
print("true R   mean estimate   sd     naive exp(r * 5)   died out")
for true_R in [1.2, 1.5, 2.0, 3.0]:
    est, naive, died = [], [], 0
    for _ in range(100):
        r = growth_rate(simulate(true_R, 30, shape, scale), 10, 20)
        if r is None:                 # the outbreak died out before day 20: nothing to 
            died += 1
            continue
        est.append(r_to_R(r, shape, scale))
        naive.append(math.exp(r * shape * scale))
    print(f"{true_R:6.1f} {statistics.mean(est):14.2f} {statistics.stdev(est):6.2f} {sta

print()
print("true R = 2.0, but we assume a different mean serial interval (same shape):")
daily_runs = [simulate(2.0, 30, shape, scale) for _ in range(100)]
rates = [r for r in (growth_rate(d, 10, 20) for d in daily_runs) if r is not None]
for assumed_mean in [3, 4, 5, 6, 7]:
    est = [r_to_R(r, shape, assumed_mean / shape) for r in rates]
    print(f"assumed mean {assumed_mean} days: mean R estimate {statistics.mean(est):5.2f
```

```
# true R   mean estimate   sd     naive exp(r * 5)   died out
#    1.2          1.24   0.22            1.25          2
#    1.5          1.56   0.19            1.61          0
#    2.0          2.04   0.12            2.18          0
#    3.0          3.01   0.07            3.56          0
#
# true R = 2.0, but we assume a different mean serial interval (same shape):
# assumed mean 3 days: mean R estimate  1.56
# assumed mean 4 days: mean R estimate  1.78
# assumed mean 5 days: mean R estimate  2.04
# assumed mean 6 days: mean R estimate  2.32
# assumed mean 7 days: mean R estimate  2.62
```

The gamma conversion recovers the true R well: 2.04 for a true 2.0 and 3.01 for a true 3.0. The shortcut overshoots, by a little at R = 1.2 (1.25) and by a lot at R = 3 (3.56), because it pretends that nobody infects anyone sooner than the average. The small upward drift at the low values (1.24 for 1.2 and 1.56 for 1.5) has two likely causes that I did not separate: rounding of lags to whole days, and, at R = 1.2, the two runs that died out and were left out, which removes the unluckiest outbreaks (both are inferences, untested). The second table shows the weak point. The simulation knows the serial interval exactly; a real analyst does not. If the true mean is 5 days but we assume 3, the estimate for a true R of 2.0 falls to 1.56, and an assumed 7 gives 2.62. An estimate of R is only as good as the interval it was converted with.

	GROWTH RATE R	SERIAL INTERVAL	R
What it is	slope of log daily cases	onset of symptoms to onset in the next case	new cases per case
Observed directly?	yes, from the counts	from interviews (a sample)	no, always inferred
Main risk	noisy when counts are small	wrong mean gives wrong R	inherits both errors
Result in the simulation (true R = 2.0)	fitted on days 10 to 20	mean 5 days, sd 2.5 assumed	2.04 (gamma), 2.18 (shortcut)

Cohort and Case-Control Studies

Counts tell us how fast an epidemic grows, but not *why*. To find what puts people at risk (a food, a gathering, a type of work) we compare groups. A **cohort study** takes a group of people, records who was exposed, follows everyone, and compares the **risk** of illness in the exposed with the risk in the unexposed. Their ratio is the **risk ratio** (RR). It is the cleaner design but needs everyone followed, which is expensive when the illness is rare. A **case-control study** goes the other way: collect the ill people (cases), pick some well people (controls), and ask both groups about the exposure. It cannot give risks, because the numbers of cases and controls were chosen by the investigator, but it gives the **odds ratio** (OR): the odds of exposure among cases divided by the odds of exposure among controls. The code builds a made-up town of 20,000 people with 30 per cent exposed, then runs both designs on it. Two illnesses are tried with the same true risk ratio of 5: a rare one (5 per cent of the exposed ill against 1 per cent of the unexposed) and a common one (50 per cent against 10 per cent). The intervals use the standard log method and are approximate.

```
# Cohort versus case-control study in a made-up population. All risks and sizes are assu
import math
import random

def make_population(n, exposed_share, risk_exposed, risk_unexposed):
    """List of (exposed, ill) pairs."""
    people = []
    for _ in range(n):
        exposed = random.random() < exposed_share
        ill = random.random() < (risk_exposed if exposed else risk_unexposed)
        people.append((exposed, ill))
    return people

def cohort(people):
    """Follow everyone: risk ratio with an approximate 95% interval (log method)."""
    a = sum(1 for e, i in people if e and i)          # exposed, ill
    b = sum(1 for e, i in people if e and not i)      # exposed, well
    c = sum(1 for e, i in people if not e and i)      # unexposed, ill
    d = sum(1 for e, i in people if not e and not i)  # unexposed, well
    rr = (a / (a + b)) / (c / (c + d))
    se = math.sqrt(1 / a - 1 / (a + b) + 1 / c - 1 / (c + d))
    return rr, rr * math.exp(-1.96 * se), rr * math.exp(1.96 * se)

def case_control(people, n_cases, n_controls):
    """Sample some ill people and some well people: odds ratio with an approximate 95% i
    cases = random.sample([p for p in people if p[1]], n_cases)
```

```
    controls = random.sample([p for p in people if not p[1]], n_controls)
    a = sum(1 for e, i in cases if e)
    b = n_cases - a
    c = sum(1 for e, i in controls if e)
    d = n_controls - c
    orr = (a * d) / (b * c)
    se = math.sqrt(1 / a + 1 / b + 1 / c + 1 / d)
    return orr, orr * math.exp(-1.96 * se), orr * math.exp(1.96 * se)

random.seed(11)
print("scenario                    study size  true RR  cohort RR (95% interval)  true (
scenarios = [("rare illness (5% v 1%)", 0.05, 0.01, 200), ("common illness (50% v 10%)",
for label, risk_e, risk_u, size in scenarios:
    people = make_population(20000, 0.3, risk_e, risk_u)
    rr, rr_lo, rr_hi = cohort(people)
    orr, or_lo, or_hi = case_control(people, size, size)
    true_or = (risk_e / (1 - risk_e)) / (risk_u / (1 - risk_u))
    print(f"{label:27s} {size:5d}+{size:<5d} {risk_e / risk_u:6.1f} {rr:8.2f} ({rr_lo:.2
# scenario                    study size  true RR  cohort RR (95% interval)  true OR  c
# rare illness (5% v 1%)       200+200      5.0     4.60 (3.77 to 5.61)     5.21       
# common illness (50% v 10%)  1000+1000     5.0     4.96 (4.69 to 5.24)     9.00       
```

In the rare case the two designs agree with the truth within their intervals: the cohort risk ratio is 4.60 (3.77 to 5.61), against a true 5.0, and the case-control odds ratio is 4.53 (2.97 to 6.90), against a true odds ratio of 5.21 calculated by hand. The case-control interval is much wider, even though the cohort followed 20,000 people and the case-control study interviewed only 400. In the common case the story changes. The cohort still recovers the risk ratio (4.96), but the case-control study estimates 8.44 (6.88 to 10.35), close to the true odds ratio of 9.00 and well away from the risk ratio of 5. The odds ratio is a different quantity from the risk ratio. It is close to the risk ratio only when the illness is rare, a standard result in epidemiology that this run illustrates, not proves. The practical lesson: a case-control odds ratio from a common illness should not be quoted as “times more likely”.

	COHORT STUDY	CASE-CONTROL STUDY
Starts from	exposure	illness
Measures	risk, so risk ratio	odds of exposure, so odds ratio

	COHORT STUDY	CASE-CONTROL STUDY
People needed	everyone at risk (20,000 here)	cases plus a few controls (400 or 2,000 here)
Good for	common exposures, several outcomes	rare illnesses, quick outbreak investigations
Main risks	cost, loss to follow-up	choosing controls, recall of past exposure
Rare illness (RR 5 true)	4.60	4.53 (true OR 5.21)
Common illness (RR 5 true)	4.96	8.44 (true OR 9.00)

Bias in Testing

So far the counts were correct. Real counts are not. A **sensitive** test (sensitivity: the share of infected people who test positive) misses few cases; a **specific** test (specificity: the share of uninfected people who test negative) raises few false alarms. Even a decent test can mislead when the infection is uncommon. Worse, the people tested are rarely a random sample: those with symptoms, or contacts of a known case, or people with the time to queue, are tested more. The code makes a town of 100,000 with 2 per cent truly infected, a test with 80 per cent sensitivity and 98 per cent specificity, and two plans. Plan A tests 5,000 random people. Plan B tests only people who come forward, which we assume 40 per cent of infected people do against 3 per cent of the rest. Each plan reports the share positive and a corrected figure from the Rogan-Gladen formula, (observed + specificity − 1) / (sensitivity + specificity − 1), which removes the effect of test errors only.

```
# Testing bias: random testing versus people who choose to be tested. Every number is as
import random

random.seed(5)
n, prevalence = 100000, 0.02          # assumed true share infected
sens, spec = 0.80, 0.98               # assumed test sensitivity and specificity
infected = [random.random() < prevalence for _ in range(n)]

def positive(is_infected):
    """Test result: infected people test positive with chance sens, others with chance 1
```

```
    return random.random() < (sens if is_infected else 1 - spec)

def rogan_gladen(observed):
    """Correct for test errors only: (observed + spec - 1) / (sens + spec - 1)."""
    return (observed + spec - 1) / (sens + spec - 1)

# Plan A: test 5,000 people chosen at random.
sample = random.sample(range(n), 5000)
observed_a = sum(positive(infected[i]) for i in sample) / len(sample)

# Plan B: test only people who come forward (40% of infected people do, 3% of the rest).
tested = [i for i in range(n) if random.random() < (0.40 if infected[i] else 0.03)]
observed_b = sum(positive(infected[i]) for i in tested) / len(tested)

print(f"true prevalence                       {sum(infected) / n:6.2%}")
print(f"A random sample ({len(sample)} tests):   positive {observed_a:6.2%}  corrected {
print(f"B volunteers ({len(tested)} tests):     positive {observed_b:6.2%}  corrected {r
# true prevalence                       1.93%
# A random sample (5000 tests):   positive  3.24%  corrected  1.59%
# B volunteers (3716 tests):     positive 18.25%  corrected 20.83%
```

The true share infected is 1.93 per cent. Plan A sees 3.24 per cent positive, about 1.7 times the truth, because at this low prevalence the false alarms from the uninfected outnumber the real finds: by my arithmetic, using the expected values, roughly 56 per cent of positive results here are false. The corrected figure, 1.59 per cent, is close but not exact, and the gap is plausibly sampling noise (about 160 positives out of 5,000 tests, so a small change in the count moves the result by a few tenths of a point; my reasoning, not tested here). Plan B is badly off: 18.25 per cent positive, and 20.83 per cent after correction, more than ten times the truth. The formula repaired the test errors but could not repair the selection, since infected people were far more likely to be tested. A bigger sample would not fix this either: Plan B's sample is large (3,716 tests) and still wrong. The common remedy is a random survey, even a small one; the size and the randomness are separate questions, and randomness is the one that matters here.

SOURCE OF ERROR	WHAT GOES WRONG	DOES IT FIX ITSELF WITH MORE TESTS?	REMEDY
Test errors	false positives and missed cases distort the share positive	no	correct with sensitivity and specificity

SOURCE OF ERROR	WHAT GOES WRONG	DOES IT FIX ITSELF WITH MORE TESTS?	REMEDY
Who gets tested	the tested are more often the infected (Plan B)	no	random sampling; report who was tested
Changing test effort	more tests find more cases with no change in infections (Exercise 2)	makes it worse	report the share positive as well as counts
Result in this run	truth 1.93%	A 3.24%, 1.59% corrected	B 18.25%, 20.83% corrected

Common mistakes

Using exp(r × mean interval) as if it were exact (it overestimates when the interval varies); assuming the serial interval is known when it is itself an estimate from a handful of interviews; fitting a growth rate on days when interventions or immunity have already changed the epidemic, so that the answer is no longer the R you wanted; reading a case-control odds ratio as a risk ratio when the illness is common; choosing controls who differ from the cases in ways other than the exposure; trusting a share-positive figure without asking who was tested and how good the test is; treating a rise in reported cases as a rise in infections when testing has also risen; correcting for test errors and believing that also corrects for who got tested; reporting a single run instead of a spread over many; and using the illustrative numbers here (5-day interval, 2 per cent prevalence, 80 and 98 per cent, 20,000 people) as estimates for a real disease.

EXERCISE 1

Simulate 100 outbreaks with a true R of 2.0 and the 5-day gamma serial interval (assumed), over 30 days. Estimate R from the growth rate in five windows (days 0 to 10, 5 to 15, 10 to 20, 15 to 25, 20 to 30), then, for the window 10 to 20, using assumed serial-interval standard deviations of 1.0, 2.5 and 4.0 days with the mean fixed at 5. Which window is least noisy here, and what would make a later window misleading in a real epidemic?

View solution

EXERCISE 2

Keep the true share infected fixed at 1 per cent (assumed) but let the number of daily tests double every 14 days, from 2,000 tests on day 0, over 28 days, with 80 per cent sensitivity and 98 per cent specificity (assumed). Estimate the growth rate and the apparent R from the raw positive counts, then from the share of tests that are positive. Which is misleading, and by how much?

View solution

EXERCISE 3

In a town of 20,000 (30 per cent exposed, 5 per cent of the exposed and 1 per cent of the unexposed ill, assumed), repeat a case-control study with 100 cases 500 times, using 1, 2, 4 and 10 controls per case. Print the mean odds ratio and the spread of its logarithm for each. How much does adding controls help, and what limits it?

View solution

QUICK REFERENCE

- **Growth rate r:** slope of the log of daily cases, fitted over a window where nothing else changes
- **Serial interval:** symptom onset to symptom onset between infector and infected; stands in for the generation interval (infection to infection)
- **From r to R:** R = exp(r × mean) only if the interval never varies; for a gamma interval R = (1 + r × scale) ^ shape (my derivation, checked by simulation)
- **Cohort study:** follow exposed and unexposed, compare risks; risk ratio = risk if exposed / risk if unexposed
- **Case-control study:** compare exposure in cases and controls; odds ratio approximates the risk ratio only if the illness is rare
- **Sensitivity / specificity:** share of infected who test positive / share of uninfected who test negative
- **Rogan-Gladen:** true share = (observed + specificity − 1) / (sensitivity + specificity − 1); fixes test errors, not selection
- **Testing bias:** volunteers, symptomatic people and rising test numbers distort counts; a random sample and the share positive help
- **Assumes:** constant R during the fitting window, known test accuracy, illustrative numbers

COMING NEXT

How Epidemics Spread 9 is about controls and interventions: quarantine, contact tracing, closures, timing effects, comparing strategies by simulation.

How Epidemics Spread

Chapter 9 · Controlling Spread

On 27 July 1377 the Great Council of Ragusa, today's Dubrovnik, passed a law that kept travellers from plague-stricken places out of the city until they had waited a month on the islands offshore. The waiting period was first thirty days (a *trentino*) and was later lengthened to forty (a *quarantena*, from which we get "quarantine"). In 1423 Venice put a hospital for plague victims on the island of Santa Maria di Nazareth, an early *lazaretto*. Nobody then knew about germs, and the logic was simply that sick people spread sickness and keeping people apart for long enough should stop it. Six hundred years later, the logic is the same, but we can ask sharper questions. Which separation matters most: of the sick, of the possibly exposed, or of everyone? How quickly must it happen? And what happens when we stop? This chapter answers those with two small simulations: a model of a closure that cuts contacts, and a model of case isolation plus contact tracing.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed (random seeds are fixed, so you should get the same numbers with the same Python random generator; the closure model uses no random numbers). The history comes from search-result summaries, which are often truncated or model-written, so check the originals before quoting. The sources agreed on 27 July 1377 as the date of the Ragusa law, on thirty days at first and forty later, and on 1423 for the Venice lazaretto, but they **conflicted on details**: one account gave thirty days for ships and forty for land travellers, another gave thirty days first and forty after a change in 1397, and one source wrongly placed Ragusa under Venice (Ragusa was an independent republic). I have not resolved these. The summary of Hellewell and colleagues, "Feasibility of controlling COVID-19 outbreaks by isolation of cases and contacts", *The Lancet Global Health* 8 (2020), gave their design (a stochastic model, varying R0, the delay to isolation, the share of contacts traced and the amount of transmission before symptoms) and the broad finding that the more infectious the disease, the more contacts must be traced; I did not see the full text. The simulations here are **my own simplified models, not a reproduction of theirs**. Every population, R0, delay, tracing share and contact cut is *assumed* or *illustrative*, and each result is a property of the model, not a forecast. John Snow removing the Broad Street pump handle in 1854 is general knowledge I did not check this session. No calculus is used: the closure model steps forward in small time steps, as in How Epidemics Spread 3.

Four Kinds of Control

The words are used loosely in the news, so fix them. **Isolation** separates people known to be infected from everyone else. **Quarantine** separates people who may have been exposed but are not known to be infected, in case they are. **Contact tracing** is the work of finding those people: ask each case who they met, then get in touch. **Closures and distancing** cut contacts for everyone, whether or not they have been exposed. Vaccination, the other great control, was the subject of How Epidemics Spread 4. All of these work on the same two things: they lower the number of people each case infects (R), or they shorten the time a case spends spreading. John Snow's pump handle is a different kind again: it removes a shared source. The table compares the four main ones.

	ISOLATION	QUARANTINE	CONTACT TRACING	CLOSURES AND DISTANCING
Who is affected	known cases	people possibly exposed	contacts of known cases	everyone
Needs	testing or symptoms to find cases	a list of who was exposed	interviews, staff, speed	no information, only compliance
Cost per person	high, but few people	moderate, some are not infected	staff time	low each, enormous in total
Weak point	transmission before cases are found	missed contacts, long waits	delays and unknown contacts	the epidemic returns when it ends
Model below	detection delay	traced share and delay	traced share and delay	contact cut c over a window

Closures: When and How Long

A closure lowers the contact rate for a while. In the model that is one multiplication: for the days the closure is in force, the transmission rate is multiplied by (1 − c), where c is the share of contacts cut. The effective reproduction number becomes R0 × (1 − c) × (share still susceptible). The code uses the S-I-R model of How Epidemics Spread 3 with an assumed R0 of 2.5 and an assumed infectious period of 5 days, in a population of 1,000,000 with 10 people infected at the start. It steps in quarter-day steps, and reports three things for six scenarios: the peak share of the population infected at one time, the day of that peak, and the share who are ever infected.

```python
# A closure that cuts contacts, started early or late, lasting a short or a long time. A
def run_sir(R0, c, start, stop, days=400, n=1_000_000, i0=10, infectious_days=5.0, dt=0.
    """Step a simple S-I-R model forward in small time steps.
    The contact factor is (1 - c) from day `start` up to (not including) day `stop`, oth
    gamma = 1.0 / infectious_days
    beta = R0 * gamma
    s, i, r = n - i0, float(i0), 0.0
    peak, peak_day = i, 0.0
    t = 0.0
    while t < days:
        factor = (1 - c) if start <= t < stop else 1.0
        new_inf = beta * factor * s * i / n * dt
        new_rec = gamma * i * dt
        s -= new_inf
        i += new_inf - new_rec
        r += new_rec
        t += dt
        if i > peak:
            peak, peak_day = i, t
    return peak / n, peak_day, (n - s) / n

scenarios = [
    ("no control", 0.0, 0, 0),
    ("70% cut, days 20-80", 0.7, 20, 80),
    ("70% cut, days 40-100", 0.7, 40, 100),
    ("70% cut, days 20-200", 0.7, 20, 200),
    ("40% cut, days 20-80", 0.4, 20, 80),
    ("70% cut, from day 20, never lifted", 0.7, 20, 10_000),
]
print(f"{'scenario':32s} {'peak infected':>14s} {'peak day':>9s} {'ever infected':>14s}"
for name, c, a, b in scenarios:
    pk, pd, fin = run_sir(2.5, c, a, b)
    print(f"{name:32s} {pk*100:13.2f}% {pd:9.0f} {fin*100:13.1f}%")
# Output when run:
# scenario                         peak infected  peak day  ever infected
# no control                              23.72%        41          89.6%
# 70% cut, days 20-80                     22.86%       112          89.3%
# 70% cut, days 40-100                    23.47%        40          65.7%
# 70% cut, days 20-200                    22.83%       255          89.2%
# 40% cut, days 20-80                      6.49%        65          78.1%
# 70% cut, from day 20, never lifted          0.32%        20           1.5%
```

With no control, 23.72 per cent of the population are infected at the peak on day 41, and 89.6 per cent are ever infected. A 70 per cent cut from day 20 to day 80 changes almost nothing in the end: the epidemic is pushed back, the peak arrives on day 112 instead of 41, but it is as tall (22.86 per cent) and the total is the same (89.3 per cent). Extending the closure to day 200 delays the peak to day 255 and again the total is 89.2 per cent. This is the central lesson of closures that end: unless something has changed by the time they lift, the susceptible people are still there and the epidemic picks up where it left off. What could change is a vaccine, a treatment, better tests, or time to build a tracing system. A weaker cut makes a different trade. A 40 per cent cut leaves R at 1.5 during the closure, so cases still grow, but more slowly: the peak is only 6.49 per cent, on day 65, and the total 78.1 per cent. Timing matters too. A 70 per cent cut starting on day 40, just when the epidemic peaks, leaves the peak where it was (23.47 per cent, on day 40, the day the cut started) but lowers the total to 65.7 per cent. My reading is that stopping transmission at the peak prevents much of the "overshoot" in which an uncontrolled epidemic keeps infecting people after most are already immune; I have not tested that. The last row, a cut that is never lifted, keeps cases to 1.5 per cent, but that is a different question from the others: nobody can hold a 70 per cent cut for ever, and the model says nothing about what that costs.

One more caution about the 20 to 200 row. A model with continuous amounts never lets the infection die out completely; a tiny fraction of a person remains and grows back. Real epidemics are made of whole people, and as shown in How Epidemics Spread 6, a tiny number of cases can simply vanish by chance, or be reseeded from outside. Treat the late peak as a feature of the model rather than a prediction.

SCENARIO (R0 2.5 ASSUMED)	PEAK INFECTED	PEAK DAY	EVER INFECTED
No control	23.72%	41	89.6%
70% cut, days 20 to 80	22.86%	112	89.3%
70% cut, days 40 to 100	23.47%	40	65.7%
70% cut, days 20 to 200	22.83%	255	89.2%
40% cut, days 20 to 80	6.49%	65	78.1%
70% cut, never lifted	0.32%	20	1.5%

Isolation, Tracing and the Race Against Time

A closure treats everyone. Isolation and tracing treat only the people who matter, but they have to win a race: each case has to be removed before it has infected too many others. The second model is a **branching process**, as in How Epidemics Spread 6. Each case infects a random number of people (Poisson, mean R = 2.5, assumed), each after a random lag (gamma-shaped, mean 5 days, as in How Epidemics Spread 8). Each case is detected a random time after being infected, with a mean you can set, and isolated at once; a case infects nobody after it is isolated. With contact tracing, each newly infected person is traced with some probability p: they are quarantined a fixed delay after their infector was isolated, if that is sooner than their own detection. The code starts every outbreak with 3 cases and calls the outbreak *escaped* if it reaches 300 cases (an assumed cut-off; the Hellewell summary gave 5,000 cases or 12 weeks for its definition of control). It runs 300 outbreaks per plan. Every case is eventually detected here, which is optimistic, and isolation is perfect, which is also optimistic.

```
# Case isolation and contact tracing as a branching process. All parameters assumed.
import math
import random
import statistics

def one_outbreak(R, detect_mean, p_trace, trace_delay, cap=300, seed_cases=3):
    """Return (escaped, total_cases). Each case infects Poisson(R) people, each after a
    (gamma, mean 5 days). A case is isolated at its own detection time (infection time +
    or, if traced, at parent's isolation time + trace_delay, whichever is first. Isolate
    nobody afterwards."""
    def poisson(lam):
        limit, k, p = math.exp(-lam), 0, 1.0
        while True:
            p *= random.random()
            if p <= limit:
                return k
            k += 1
    queue = []  # (infection_time, isolation_time)
    total = 0
    for _ in range(seed_cases):
        t0 = 0.0
        queue.append((t0, t0 + random.gammavariate(4, detect_mean / 4)))
    while queue:
        t, iso = queue.pop()
        total += 1
        if total >= cap:
            return True, total
        for _ in range(poisson(R)):
```

```
            lag = random.gammavariate(4, 1.25)
            if t + lag >= iso:
                continue                    # parent already isolated: no infection
            child_t = t + lag
            child_iso = child_t + random.gammavariate(4, detect_mean / 4)
            if random.random() < p_trace:     # child is traced and quarantined
                child_iso = min(child_iso, max(child_t, iso + trace_delay))
            queue.append((child_t, child_iso))
    return False, total

random.seed(9)
print(f"{'plan':42s} {'escaped':>8s} {'mean size if contained':>24s}")
plans = [
    ("no isolation (detect after 1000 days)", 1000, 0.0, 0),
    ("isolate cases, detection mean 5 days", 5, 0.0, 0),
    ("isolate cases, detection mean 3 days", 3, 0.0, 0),
    ("detect 5 days + trace 50%, 2-day delay", 5, 0.5, 2),
    ("detect 5 days + trace 90%, 2-day delay", 5, 0.9, 2),
    ("detect 5 days + trace 90%, 5-day delay", 5, 0.9, 5),
]
for name, d, p, td in plans:
    runs = [one_outbreak(2.5, d, p, td) for _ in range(300)]
    esc = sum(1 for e, _ in runs if e)
    small = [n for e, n in runs if not e]
    shown = f"{statistics.mean(small):.1f}" if small else "-"
    print(f"{name:42s} {esc / 300 * 100:7.0f}% {shown:>24s}")
# Output when run:
# plan                                       escaped   mean size if contained
# no isolation (detect after 1000 days)         100%                        -
# isolate cases, detection mean 5 days           59%                     12.3
# isolate cases, detection mean 3 days            0%                      7.9
# detect 5 days + trace 50%, 2-day delay         11%                     37.5
# detect 5 days + trace 90%, 2-day delay          0%                     16.3
# detect 5 days + trace 90%, 5-day delay         63%                     14.3
```

Without isolation every outbreak escapes. Isolating cases with a mean detection time of 5 days is not enough: 59 per cent of outbreaks escape, because by the time a case is found it has already passed the infection on, and each of those cases will do the same. Cutting the detection time to 3 days stops all 300 in this run. Detection speed is a threshold effect in this model, not a gentle slope. Tracing with the 5-day detection and a 2-day delay brings escape down to 11 per cent at 50 per cent tracing and to 0 at 90 per cent. But the same 90 per cent tracing with a 5-day

delay does no better than isolation alone (63 per cent escape, against 59 per cent, which is within what 300 runs can scatter by, about 3 points either way by a rough binomial estimate of mine). A trace that takes as long as the detection itself does not catch anyone early. The mean size of the outbreaks that were contained is also informative: 37.5 cases for the 50 per cent trace, since a weaker control lets contained outbreaks run longer before they die out. The direction agrees with the summary of Hellewell and colleagues, that faster isolation and more tracing both help, and that more infectious diseases need more of both. That is a qualitative agreement with a summary, nothing more.

PLAN (R = 2.5 ASSUMED, 300 RUNS)	ESCAPED	MEAN SIZE IF CONTAINED	WHAT IT SHOWS
No isolation	100%	none contained	the baseline
Isolate, 5-day detection	59%	12.3	not fast enough alone
Isolate, 3-day detection	0%	7.9	speed alone can be enough
5-day detection, trace 50%, 2-day delay	11%	37.5	tracing helps a lot
5-day detection, trace 90%, 2-day delay	0%	16.3	high coverage, fast
5-day detection, trace 90%, 5-day delay	63%	14.3	coverage without speed fails

Comparing Strategies Honestly

The two models make different promises. The closure model asks how much total harm a measure prevents, in a well-mixed population. The branching model asks whether a small outbreak can be stopped before it grows. Often the right answer is a combination: a short closure while tracing capacity is built, then tracing and isolation to keep numbers down after reopening. Comparing strategies by simulation needs the same discipline as comparing drugs. Change one thing at a time; run many random repeats and report the spread, as in How Epidemics Spread 6; write down every assumed number; and look for the scenario where your favourite plan fails. The most important numbers to vary are the ones you are least sure of: the delay, the share of cases that are ever found, and the share of contacts you can name. The measures are also not independent in real life (people who isolate may also be the ones who

distance), and none of the models here counts the cost of a measure, which is what makes the choice hard.

Common mistakes

Treating a temporary closure as a cure when the susceptible people are still there afterwards; reading the late peak of a model that never lets the infection die out as a prediction; judging tracing by the share of contacts reached and ignoring how long it takes; assuming every case is found and every isolation is perfect (both are optimistic in the model here); comparing two plans on one random run instead of many; mixing up quarantine (possibly exposed) with isolation (known infected); judging a measure by cases prevented without asking what it costs; calling the model a reproduction of a published study when it is only a simplified model in the same spirit; and using the illustrative numbers here (R0 of 2.5, a 5-day interval, 300 cases, a 70 per cent cut) as estimates for a real disease.

EXERCISE 1

Using the closure model (R0 2.5, a 70 per cent contact cut, both assumed), try starting days of 10, 30 and 50 with closure lengths of 30, 60 and 120 days. Print the peak share infected and the share ever infected for each of the nine. When does a closure change the peak, when does it change only the total, and when does it change neither?

View solution

EXERCISE 2

Using the branching model with R = 2.5 and a 5-day mean detection time (assumed), find the share of 200 outbreaks that escape for tracing coverages of 30, 50, 70 and 90 per cent and tracing delays of 1, 3 and 5 days. Which matters more at low coverage, and what does a 5-day delay do to the benefit of high coverage?

View solution

EXERCISE 3

Combine a contact cut with isolation (no tracing). Cut contacts by 0, 20, 40 and 50 per cent, so that R becomes 2.5 × (1 − cut), and use mean detection times of 7, 5 and 3 days. For each pair print the share of 200 outbreaks that escape. How much of a contact cut is needed to make slow detection enough, and how optimistic are the model's assumptions?

View solution

QUICK REFERENCE

- **Isolation:** separate known cases; **quarantine:** separate people who may have been exposed; **tracing:** find those people
- **Closure model:** transmission multiplied by (1 − c) between a start and an end day; effective R = R0 × (1 − c) × share susceptible
- **A closure that ends** leaves the susceptible people; unless something changes the epidemic returns (peak delayed, total similar)
- **Timing:** started at the peak, a cut lowered the total but not the peak; started early and short, it only delayed the epidemic (in this model)
- **Branching model:** Poisson children, gamma lags, detection delay, isolated cases infect nobody; escaped = reaches the case cap
- **Race against time:** tracing needs coverage and speed; a trace as slow as detection adds little
- **Compare strategies:** many runs, one change at a time, all assumptions stated, include a scenario where the plan fails
- **Assumes:** well-mixed population, perfect isolation, every case eventually detected, illustrative numbers

COMING NEXT

How Epidemics Spread 10 is the capstone, Plan a Response: the history of famous outbreaks, building a small outbreak simulator and judging a control plan.

How Epidemics Spread

Chapter 10 · Plan a Response

Imagine you are handed a report at nine in the morning: ten people in a town of one hundred thousand have a new illness, and three more have turned up since breakfast. Somebody is going to ask you what to do. You cannot know the answer, but you now know how to ask good questions: how fast is it growing, what is R, who is infectious and when, how many cases are we missing, what does each measure buy, and what happens when the measure ends? This last chapter puts the whole course to work. It first looks back at five famous outbreaks and asks what each of them taught, then builds one small outbreak simulator out of the pieces of the earlier chapters, and finally tests a response plan against the numbers we are least sure of.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed (random seeds are fixed, so the same Python random generator should give the same numbers). The history comes from search-result summaries, which are often truncated, so check the originals before quoting. Sources **conflicted** on several points and I have not resolved them: estimates of the Black Death's toll run from about 39 per cent of Western Europe's population (one economists' synthesis) to over 60 per cent of Europe's (one demographer), and the number of deaths in the 1854 Broad Street outbreak is given as 616 in older accounts and 420 in the current English Wikipedia article. The 1918 influenza figure of at least 50 million deaths comes from a CDC page; other studies put it at up to 100 million. The date of the World Health Organization's declaration that smallpox was eradicated, 8 May 1980 (resolution WHA33.3), appeared in a search summary. The ring-vaccination strategy for smallpox, the 2014 to 2016 West African Ebola outbreak and the March 2020 pandemic declaration are general knowledge that I did not check this session, so I mark them "(check)". The simulator is **my own simplified model**. Every population, R0, delay, detection rate, contact cut and cost weight is *assumed* or *illustrative*, and each result is a property of the model, not a forecast for any real disease. No calculus is used: the model steps forward one day at a time, as in How Epidemics Spread 3 and How Epidemics Spread 5.

Five Outbreaks, Five Lessons

Each famous outbreak is remembered for a number, but what makes it useful to a planner is the question it forced people to ask. The table pairs each with the part of this course that explains it. The numbers are rough, and where the sources disagree the table says so.

OUTBREAK	WHAT THE NUMBERS SAY	LESSON FOR A PLANNER	WHERE WE MET IT
Black Death, 1347 to 1351	roughly 39% of Western Europe's population in one synthesis; 51 to 58%, or over 60%, in other estimates (sources conflict)	separation can work without knowing the cause; ports learned to wait before letting people in	How Epidemics Spread 9 (quarantine)
Broad Street cholera, 1854	616 deaths in older accounts, 420 in the current English Wikipedia article (conflict)	map the cases and look for the shared source; a pump handle is a control	How Epidemics Spread 1 (counting cases) and How Epidemics Spread 8 (case-control reasoning)
Influenza, 1918 to 1919	at least 50 million deaths and about a third of the world infected (CDC); up to 100 million in other studies	a pandemic arrives in waves, and every estimate is wide, so plan for a range	How Epidemics Spread 2 (growth) and How Epidemics Spread 8 (estimating)
Smallpox eradication, declared 8 May 1980	the WHO declaration; the strategy of vaccinating the contacts of each case, called ring vaccination, was central (check)	find every case fast and protect the people around it: tracing plus vaccination	How Epidemics Spread 4 (herd immunity) and How Epidemics Spread 9 (tracing)
Ebola in West Africa, 2014 to 2016 (check)	tens of thousands of cases (I did not verify the totals)	when hospital beds and staff run out, isolation fails, so capacity is part of the plan	How Epidemics Spread 6 (small outbreaks) and How Epidemics Spread 9 (race against time)

One more is too recent to settle: the pandemic that the World Health Organization declared in March 2020 (check) produced more public modelling, and more arguments about modelling, than all earlier outbreaks put together. Many of the arguments were about exactly the questions in the next sections: which numbers were assumed, how sensitive the answer was to them, and what the models left out.

A Small Outbreak Simulator

The pieces are already built. From How Epidemics Spread 5 the model keeps an exposed (latent) stage between catching the infection and being able to pass it on: a person is **S**usceptible, then **E**xposed, then **I**nfectious, then **R**ecovered. From How Epidemics Spread 6 it keeps chance: the number of new infections each day is a random draw (Poisson) and so is the number of people

who move between stages (binomial), so every run is different. From How Epidemics Spread 9 it takes two controls: **isolation**, in which each infectious person is found with some chance each day and then infects nobody, and a **closure** that multiplies the contact rate by (1 − cut) for a fixed number of days once enough cases have been found. A plan is then just a set of settings. The code runs 100 repeats of each plan and reports the 10th, 50th (median) and 90th percentile of the share ever infected, plus the average number of closed days.

```
# A small outbreak simulator: stochastic SEIR in daily steps, with isolation and a trigg
# Every number is assumed or illustrative.
import math
import random

def poisson(lam):
    """Poisson draw: Knuth's method for small means, a normal approximation for large on
    if lam <= 0:
        return 0
    if lam > 30:
        return max(0, round(random.gauss(lam, math.sqrt(lam))))
    limit, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= random.random()
        if p <= limit:
            return k
        k += 1

def binom(n, p):
    """Binomial draw: exact for small n, normal approximation for large n."""
    if n <= 0 or p <= 0:
        return 0
    if p >= 1:
        return n
    if n < 40:
        return sum(1 for _ in range(n) if random.random() < p)
    return min(n, max(0, round(random.gauss(n * p, math.sqrt(n * p * (1 - p))))))

def simulate(R0=2.5, detect=0.0, trigger=None, cut=0.5, closure_days=60,
             n=100_000, seeds=10, latent=3.0, infectious=5.0, days=300):
    """Return (share ever infected, peak infectious, closure days used).
    detect: chance per day that an infectious person is found and isolated (0 = never).
    trigger: close when this many cases have been found (None = never close)."""
    s, e, i, q, r = n - seeds, 0, seeds, 0, 0
    beta = R0 / infectious
```

```
    found, closed_left, closed_used, peak = 0, 0, 0, seeds
    for day in range(days):
        factor = (1 - cut) if closed_left > 0 else 1.0
        if closed_left > 0:
            closed_left -= 1
            closed_used += 1
        new_e = min(s, poisson(beta * factor * s * i / n))
        new_i = binom(e, 1 / latent)
        new_found = binom(i, detect)
        new_r = binom(i - new_found, 1 / infectious)
        s -= new_e
        e += new_e - new_i
        i += new_i - new_found - new_r
        q += new_found
        r += new_r
        found += new_found
        peak = max(peak, i)
        if trigger is not None and closed_used == 0 and closed_left == 0 and found >= tr
            closed_left = closure_days
    return (n - s) / n, peak, closed_used

def summary(label, **kwargs):
    random.seed(10)
    runs = [simulate(**kwargs) for _ in range(100)]
    attack = sorted(a for a, _, _ in runs)
    print(f"{label:34s} {attack[10]*100:6.1f}% {attack[50]*100:6.1f}% {attack[90]*100:6.
          f"{sum(c for _, _, c in runs) / 100:8.1f}")

print(f"{'plan (100 runs, R0 2.5 assumed)':34s} {'p10':>7s} {'median':>7s} {'p90':>7s} {
summary("no control")
summary("isolate (find 25% a day)", detect=0.25)
summary("closure at 200 found, 50% cut", detect=0.1, trigger=200)
summary("isolate + closure at 200 found", detect=0.25, trigger=200)
summary("isolate + early closure (30 found)", detect=0.25, trigger=30)
# Output when run:
# plan (100 runs, R0 2.5 assumed)        p10  median     p90  closed days
# no control                           89.9%   90.1%   90.3%      0.0
# isolate (find 25% a day)             35.4%   37.1%   38.4%      0.0
# closure at 200 found, 50% cut        71.7%   72.1%   72.6%     60.0
# isolate + closure at 200 found        0.5%    0.5%    0.8%     57.6
# isolate + early closure (30 found)    0.1%    0.1%    0.1%     57.6
```

With no control about 90 per cent of the population is infected in every run. Isolation alone, finding a quarter of the infectious each day (a mean of 4 days from becoming infectious to being found, against a 5-day infectious period; both assumed), cuts that to 37 per cent. A closure by itself, with slow finding (10 per cent a day) and a 50 per cent cut for 60 days, does little: 72 per cent infected, because the closure ends and the epidemic resumes, the lesson of How Epidemics Spread 9. Put the two together and the picture changes: isolation plus a closure triggered at 200 found cases leaves only 0.5 per cent ever infected, and triggering it at 30 found cases leaves 0.1 per cent. The closure holds numbers down while isolation, now facing a small number of cases, catches up. The average number of closed days is a little under 60 in the last two rows; my reading, which I did not test, is that in a few runs the outbreak died out before the trigger was reached.

PLAN (100 RUNS, R0 2.5 ASSUMED)	MEDIAN EVER INFECTED	SPREAD (10TH TO 90TH)	CLOSED DAYS
No control	90.1%	89.9% to 90.3%	0
Isolation only (25% found a day)	37.1%	35.4% to 38.4%	0
Closure at 200 found, slow finding	72.1%	71.7% to 72.6%	60.0
Isolation + closure at 200 found	0.5%	0.5% to 0.8%	57.6
Isolation + closure at 30 found	0.1%	0.1% to 0.1%	57.6

A caution about the tidy spread. In a population of 100,000 with 10 starting cases the simulation draws large numbers from a normal approximation, so the runs differ less than they would for a small village, and the random events that decide whether a tiny outbreak fizzles, as in How Epidemics Spread 6, are largely absent once the cases are many. The plan comparisons are still fair among themselves, but the narrow spreads should not be read as real-world certainty.

Judging a Plan: Stress-Test It

A plan that wins in the base case is only a candidate. The next step is to ask where it breaks. In the stress test below the plan is fixed (isolation plus a closure at 30 found cases) and the two numbers we know least about are varied: R0 (2.0, 2.5 or 3.5, all assumed) and how fast cases are found (from 35 per cent a day down to 5 per cent). Each cell is the median share ever infected over 60 runs.

```python
# Stress-testing a plan: the same plan under different assumed values of the disease and
import math
import random

def poisson(lam):
    if lam <= 0:
        return 0
    if lam > 30:
        return max(0, round(random.gauss(lam, math.sqrt(lam))))
    limit, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= random.random()
        if p <= limit:
            return k
        k += 1

def binom(n, p):
    if n <= 0 or p <= 0:
        return 0
    if p >= 1:
        return n
    if n < 40:
        return sum(1 for _ in range(n) if random.random() < p)
    return min(n, max(0, round(random.gauss(n * p, math.sqrt(n * p * (1 - p))))))

def attack_rate(R0, detect, trigger, cut=0.5, closure_days=60, n=100_000, seeds=10):
    """Share ever infected for one stochastic SEIR run (latent 3 d, infectious 5 d, both
    s, e, i = n - seeds, 0, seeds
    beta, found, left, used = R0 / 5.0, 0, 0, 0
    for _ in range(300):
        factor = (1 - cut) if left > 0 else 1.0
        if left > 0:
            left -= 1
            used += 1
        new_e = min(s, poisson(beta * factor * s * i / n))
        new_i = binom(e, 1 / 3.0)
        new_f = binom(i, detect)
        new_r = binom(i - new_f, 1 / 5.0)
        s -= new_e
        e += new_e - new_i
        i += new_i - new_f - new_r
        found += new_f
        if used == 0 and left == 0 and found >= trigger:
```

```
            left = closure_days
        return (n - s) / n

    def median_attack(R0, detect, trigger, runs=60):
        vals = sorted(attack_rate(R0, detect, trigger) for _ in range(runs))
        return vals[runs // 2]

    random.seed(21)
    print("Plan: isolate + closure triggered at 30 found cases. Median share ever infected."
    print(f"{'':16s} {'R0 = 2.0':>10s} {'R0 = 2.5':>10s} {'R0 = 3.5':>10s}")
    for detect in (0.35, 0.25, 0.10, 0.05):
        cells = [median_attack(R0, detect, 30) * 100 for R0 in (2.0, 2.5, 3.5)]
        print(f"detect {detect:4.2f}/day " + " ".join(f"{c:9.1f}%" for c in cells))
    # Output when run:
    # Plan: isolate + closure triggered at 30 found cases. Median share ever infected.
    #                     R0 = 2.0   R0 = 2.5   R0 = 3.5
    # detect 0.35/day       0.0%       0.1%       0.2%
    # detect 0.25/day       0.1%       0.1%      71.9%
    # detect 0.10/day       0.3%      73.2%      89.5%
    # detect 0.05/day      68.0%      82.1%      92.1%
```

The plan looks safe, with 0.1 per cent infected, in the middle of the table, and then it falls off a cliff. At R0 = 3.5 a find rate of 25 per cent a day is not enough (71.9 per cent infected). At R0 = 2.5 a find rate of 10 per cent a day is not enough (73.2 per cent). Even at R0 = 2.0 a find rate of 5 per cent fails (68.0 per cent). There is no gentle middle: in this model the outcome is either a small outbreak or a large one, and which one you get depends on whether faster finding and the closure together push the effective reproduction number below 1, as in How Epidemics Spread 4 and How Epidemics Spread 9. Because the outcomes are one or the other, the median is a blunt summary; with real data you would report the share of runs that escape, as in How Epidemics Spread 9. The lesson for a planner is to work out how close to the edge you are, since a small error in R0 or in detection speed can flip the result completely, and these are the numbers hardest to measure early (How Epidemics Spread 8).

A Checklist for a Response Plan

The simulator answers a narrow question. A real plan needs the questions around it. This list is my own summary of the course, not an official protocol.

QUESTION	WHY IT MATTERS	WHERE IT CAME FROM
How fast is it growing, and how sure are we?	doubling time and R0 set how little time there is	How Epidemics Spread 2, How Epidemics Spread 8
How many cases are we missing?	testing bias distorts every curve	How Epidemics Spread 1, How Epidemics Spread 8
What does each measure buy, and for how long?	a closure that ends leaves the susceptible people behind	How Epidemics Spread 9
Who is reached and who is missed?	hubs and clusters matter more than averages	How Epidemics Spread 7
What if the key number is wrong?	plans fail at cliff edges, as the stress test shows	this capstone
What does it cost, and who decides the weight?	the best plan changes with the value placed on a closed day	Exercise 3
What is the exit?	immunity, tracing capacity or a vaccine must change before a closure lifts	How Epidemics Spread 4, How Epidemics Spread 9

Common mistakes

Quoting one number from a model as if it were a forecast; judging a plan only in the base case and never stress-testing it; reading a median when the outcomes fall into two groups; trusting a narrow spread between runs that comes from the model's own simplifications; treating the cost of a measure as zero, or choosing its weight without saying so; assuming every case is found and every isolation is perfect (both are optimistic here); copying the illustrative numbers in this chapter (R0 of 2.5, a 5-day infectious period, a 50 per cent cut) into a real decision; quoting the historical death tolls as if the sources agreed; and forgetting that a model can only answer the question it was built for.

EXERCISE 1

Using the simulator with slow isolation (10 per cent of the infectious found each day), a 50 per cent contact cut and a closure of 60 days, try triggers of 30, 100, 300, 1,000 and 3,000 found cases. Print the median share ever infected and the mean number of closed days over 100 runs each. Does an earlier or a later closure do better, and does either come close to stopping the epidemic?

View solution

EXERCISE 2

With slow isolation (10 per cent a day) and a trigger of 100 found cases, try contact cuts of 30, 50 and 70 per cent and closure lengths of 30, 60 and 120 days. Print the median share ever infected and the cost as cut times mean closed days. Is there a threshold, and is a longer closure always worth its cost?

View solution

EXERCISE 3

Score four plans with a loss equal to infections per 1,000 people plus w times the full-closure days, for w of 0.5, 2 and 8: isolation only, isolation plus a closure at 30 found cases, isolation plus a closure at 200, and slow finding (10 per cent a day) plus a closure at 30. Which plan is best for each w, and who should choose w?

View solution

QUICK REFERENCE

- **Simulator:** S, E, I, R in daily steps; Poisson new infections, binomial transitions; isolation finds a share of the infectious each day; a closure multiplies contacts by (1 − cut) for a set number of days after a trigger
- **Plan comparison:** many runs, one change at a time, report a spread and the cost, state every assumed number
- **Isolation plus an early closure** beat either alone in this model; a closure alone was only a delay
- **Stress test:** vary the uncertain numbers (R0, detection speed); expect cliff edges, not gentle slopes
- **Median caution:** if outcomes are either small or large, report the share of runs that escape
- **The weight on cost is a value judgement:** the model cannot choose it
- **History:** estimates conflict (Black Death, Broad Street, 1918); give ranges and say which source
- **Assumes:** well-mixed population, perfect isolation, illustrative numbers throughout

WHERE TO GO FROM HERE

You now have the working parts of the field: counting cases, growth and doubling, compartment models, herd immunity, chance, networks, estimation and controls. To go further, pick one original source and read it in full rather than a summary; the paper on isolation and contact tracing that How

Epidemics Spread 9 described is a good start. Then extend the simulator yourself. Add vaccination, as in How Epidemics Spread 4, or loss of immunity, as in How Epidemics Spread 5, or replace the well-mixed population with the contact network of How Epidemics Spread 7 and see which plans survive. Fit the model to a real epidemic curve using the methods of How Epidemics Spread 8, and always write down which numbers you assumed. A model is best treated as a way of asking sharper questions, not as a source of answers.