Mathematical Modeling

Lecture Notes

Master M1 — 2025–2026

Yaë Ulrich Gaba

“Essentially, all models are wrong, but some are useful.”
— George E. P. Box

March 25, 2026

Contents

Preface

"All models are wrong, but some are useful."
— George E. P. Box

Mathematical modelling is the art and science of translating real-world phenomena into rigorous mathematical structures, analysing those structures to derive predictions, and confronting those predictions with observations. This course is aimed at second- and third-year undergraduate students (L2/L3) in mathematics, physics, quantitative biology, or engineering, with a solid background in calculus, linear algebra, and ordinary differential equations.

Learning Objectives

By the end of this course, the student will be able to:

1. **Formulate** a concrete problem as a mathematical model, identifying relevant variables, parameters, and assumptions.

2. **Analyse** the resulting equations qualitatively and quantitatively: equilibria, stability, bifurcations, asymptotic behaviour.

3. **Simulate** models numerically using Python (`scipy`, `numpy`, `matplotlib`).

4. **Validate** a model by comparing it with experimental data and discussing its limitations.

5. **Communicate** results clearly and rigorously, both in writing and through visualisations.

Course Philosophy

We follow a *concrete-to-abstract* approach. Each chapter starts from an observable phenomenon — population growth, epidemic spread, heat diffusion, price fluctuation — and builds the corresponding mathematical model step by step. Theoretical tools (dimensional analysis, stability theory, stochastic processes) are introduced *when they become necessary*, not as abstract prerequisites.

Model $\neq$ Reality

A model is always a simplification. Box's quote above should be kept in mind throughout the course. We will systematically emphasise the **assumptions** underlying each model and the consequences of violating them.

Course Structure

The course is organised into ten chapters, following a logical progression:

Chapter 1 — The Modelling Process. General methodology, stages of modelling, classification of models.

Chapter 2 — Dimensional Analysis and Scaling. Buckingham Π theorem, non-dimensionalisatio orders of magnitude.

Chapter 3 — Population Dynamics. Malthus, Verhulst, Lotka–Volterra models; stability and phase portraits.

Chapter 4 — Epidemiological Models. SIR, SEIR; basic reproduction number $\mathcal{R}_0$; vaccination strategies.

Chapter 5 — Discrete Dynamical Systems and Chaos. Logistic map, Lyapunov exponents, bifurcation diagrams.

Chapter 6 — Diffusion and Transport Models. Heat equation, advection-diffusion, Fick's law.

Chapter 7 — Economic and Financial Models. Solow model, Black–Scholes, supply-demand equilibrium.

Chapter 8 — Optimization and Variational Models. Linear programming, KKT conditions, calculus of variations.

Chapter 9 — Stochastic Models. Random walks, Markov chains, Brownian motion, SDEs.

Chapter 10 — Case Studies and Projects. Integrative applications spanning multiple domains.

Prerequisites

- **Calculus:** sequences, series, differential and integral calculus (single and multivariable), ordinary differential equations.
- **Linear Algebra:** vector spaces, matrices, eigenvalues, diagonalisation.
- **Probability:** probability spaces, random variables, expectation, variance, normal distribution.
- **Programming:** basic Python; additional libraries will be introduced as needed.

Conventions and Notation

- $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{C}$: standard number sets.
- $\mathbb{E}[X]$, $\mathrm{Var}(X)$: expectation and variance of a random variable X.
- $\|\cdot\|$, $|\cdot|$: norm and absolute value.
- Vectors in boldface: $\mathbf{x}$, $\mathbf{v}$.
- Time derivatives: $\dot{x} = dx/dt$.
- Dimensionless quantities: $\tilde{x} = x/x_0$.

Practical Organisation

Each chapter contains:

- Numbered **definitions**, **theorems**, and **propositions**.
- Detailed **examples** illustrating the concepts.
- Boxed **Key Formulas**, **Warnings**, **Best Practices**, and **Intuition** panels.
- Complete, commented **Python simulations**.
- **Exercises** of increasing difficulty.

Software

Python Libraries Used

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint, solve_ivp
from scipy.optimize import minimize, linprog
from scipy.linalg import eig
```

Acknowledgements

This course draws on the classic textbooks by C. Lin and L. Segel (*Mathematics Applied to Deterministic Problems in the Natural Sciences*), J. D. Murray (*Mathematical Biology*), S. H. Strogatz (*Nonlinear Dynamics and Chaos*), and E. A. Bender (*An Introduction to Mathematical Modeling*).

Happy reading and happy modelling!

Chapter 1

The Modelling Process

Galileo wrote that the book of nature is written in the language of mathematics. But between nature and equations, there is a bridge to build: the *model*. A model is a deliberate simplification of reality that captures the essential mechanisms while ignoring superfluous details. The art of modelling — for it is as much an art as a science — consists in choosing the right level of abstraction: enough detail for the model to be useful, enough simplicity for it to be tractable.

"Science is built of facts the way a house is built of bricks; but an accumulation of facts is no more a science than a pile of bricks is a house."
— Henri Poincaré

This opening chapter introduces the general approach to mathematical modelling. We describe the key steps from observing a phenomenon to validating the model, and classify the main families of mathematical models.

1.1 What Is a Mathematical Model?

Definition 1.1 (Mathematical model)**.** A **mathematical model** is a formal representation of a real-world system (physical, biological, economic, etc.) using mathematical objects (equations, inequalities, graphs, probability distributions) that capture the essential relationships between the system's quantities.

Remark 1.2. A model is never unique: the same phenomenon can be described by different models of varying complexity. The choice depends on the question asked and the data available.

1.2 Steps in the Modelling Process

The modelling process consists of five broad stages, forming an iterative cycle.

1. **Observation and problem identification.** Delineate the phenomenon, identify relevant quantities (state variables, parameters, inputs/outputs), and gather available data.

CHAPTER 1. THE MODELLING PROCESS

2. **Formulation of assumptions.** State explicitly which interactions are neglected, which quantities are assumed constant, and what the spatial and temporal domain of validity is.
3. **Mathematical formulation.** Translate assumptions into equations. This may yield:
   - an ordinary differential equation (ODE),
   - a partial differential equation (PDE),
   - an algebraic system,
   - an optimisation problem,
   - a stochastic model.
4. **Solution and analysis.** Solve the model analytically (when possible) or numerically. Study qualitative behaviour: existence and uniqueness, stability of equilibria, parameter dependence.
5. **Validation and interpretation.** Compare model predictions with experimental data. If the agreement is insufficient, return to Step 2 to revise the assumptions.

The modelling cycle

Modelling is not a linear process but a **cycle**. After validation, one often revisits the assumptions, enriches the model, or reduces its complexity. A good modeller goes through several iterations.

1.3 Classification of Models

Definition 1.3 (Types of models)**.** Models can be classified along several axes:

1. **Deterministic vs. stochastic.** A deterministic model predicts a unique trajectory for given initial conditions; a stochastic model incorporates randomness.
2. **Continuous vs. discrete.** Time and/or space may vary continuously or in discrete steps.
3. **Static vs. dynamic.** Static models describe equilibrium states; dynamic models describe temporal evolution.
4. **Linear vs. nonlinear.** Linearity of the relationships greatly affects the analysis methods.
5. **Parametric vs. non-parametric.** Whether the model structure is fixed a priori or learned from data.

Example 1.4. Consider a falling object. Neglecting air resistance, the model is:

$$m\ddot{y} = -mg, \qquad y(0) = h, \quad \dot{y}(0) = 0.$$

This is a deterministic, continuous, dynamic, linear model. Its solution is $y(t) = h - \frac{1}{2}gt^2$.

Adding quadratic air resistance yields a **nonlinear** model:

$$m\ddot{y} = -mg + k\dot{y}^2.$$

1.4 Fundamental Construction Principles

1.4.1 Conservation Laws

Many models rest on **conservation laws**: conservation of mass, energy, momentum, or number of individuals.

Theorem 1.5 (Generic balance equation)**.** *For a quantity $Q(t)$ contained in a domain Ω, the generic conservation law reads:*

$$\frac{dQ}{dt} = (\mathit{inflow}) - (\mathit{outflow}) + (\mathit{source}) - (\mathit{sink}).$$

Example 1.6. Let $N(t)$ be the size of a population with per-capita birth rate b and death rate d. The balance gives:

$$\frac{dN}{dt} = bN - dN = (b-d)N.$$

This is the Malthus model, studied in detail in Chapter 3.

1.4.2 Constitutive Laws

Constitutive laws relate fluxes to state variables.

Example 1.7.
- **Fourier's law:** heat flux is proportional to the temperature gradient: $\mathbf{q} = -\kappa\nabla T$.
- **Fick's law:** diffusive flux is proportional to the concentration gradient: $\mathbf{J} = -D\nabla c$.
- **Hooke's law:** stress is proportional to strain: $\sigma = E\varepsilon$.

1.4.3 Variational Principles

Some models are obtained by minimising (or making stationary) a functional.

Definition 1.8 (Action functional)**.** In mechanics, the principle of least action states that the true trajectory makes stationary the functional

$$\mathcal{S}[q] = \int_{t_1}^{t_2} L(q, \dot{q}, t)\, dt,$$

where L is the Lagrangian. The resulting Euler–Lagrange equations will be studied in Chapter 8.

1.5 Qualities of a Good Model

Quality criteria

A good model strikes a balance between:

- **Simplicity** (parsimony): the number of parameters should remain reasonable (Occam's razor).

CHAPTER 1. THE MODELLING PROCESS

- **Accuracy**: predictions must be close enough to observations.
- **Robustness**: small parameter changes should not cause dramatic prediction shifts.
- **Interpretability**: parameters should have a physical or biological meaning.

Proposition 1.9 (Parsimony principle)**.** Given two models of comparable accuracy, one prefers the one with fewer free parameters. This is formalised by information criteria such as the AIC (Akaike Information Criterion):

$$\mathrm{AIC} = 2k - 2\ln \hat{L},$$

where k is the number of parameters and $\hat{L}$ the maximum likelihood.

1.6 Sensitivity and Uncertainty

Definition 1.10 (Sensitivity analysis)**.** Sensitivity analysis studies how the output y of a model varies when a parameter p changes. The **local sensitivity index** is defined by:

$$S_p = \frac{p}{y}\frac{\partial y}{\partial p}.$$

An index $|S_p| \gg 1$ signals high sensitivity to p.

Example 1.11. For the exponential model $y(t) = y_0 e^{rt}$:

$$S_r = \frac{r}{y}\frac{\partial y}{\partial r} = \frac{r}{y_0 e^{rt}} \cdot y_0 t e^{rt} = rt.$$

Sensitivity to r grows linearly with time: small errors in r lead to large errors in y over long horizons.

Sensitivity analysis in Python

```python
import numpy as np
import matplotlib.pyplot as plt

def model(t, r, y0=1.0):
    return y0 * np.exp(r * t)

t = np.linspace(0, 10, 200)
r_nom = 0.5
dr = 0.05  # 10% perturbation

y_nom = model(t, r_nom)
y_plus = model(t, r_nom + dr)
y_minus = model(t, r_nom - dr)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(t, y_nom, 'b-', label=f'$r = {r_nom}$')
```

```
ax1.fill_between(t, y_minus, y_plus, alpha=0.3,
                 label=f'$r = {r_nom} \\pm {dr}$')
ax1.set_xlabel('$t$'); ax1.set_ylabel('$y(t)$')
ax1.legend(); ax1.set_title('Solutions')

S_r = r_nom * t
ax2.plot(t, S_r, 'r-')
ax2.set_xlabel('$t$'); ax2.set_ylabel('$S_r$')
ax2.set_title('Sensitivity index $S_r = rt$')
plt.tight_layout(); plt.savefig('sensitivity.pdf')
```

1.7 Fitting Models to Data

Once a model is built, its parameters must be estimated from data.

Definition 1.12 (Least squares)**.** Given observations (t_i, y_i^{obs}), $i = 1, \dots, n$, and a model $y(t; \mathbf{p})$ depending on a parameter vector $\mathbf{p}$, the **least-squares estimator** is:

$$\hat{\mathbf{p}} = \arg\min_{\mathbf{p}} \sum_{i=1}^{n} \left(y_i^{\text{obs}} - y(t_i; \mathbf{p})\right)^2.$$

Least-squares fitting

```
import numpy as np
from scipy.optimize import curve_fit

np.random.seed(42)
t_data = np.linspace(0, 5, 20)
r_true, y0_true = 0.8, 2.0
y_data = y0_true * np.exp(r_true * t_data) \
         + np.random.normal(0, 1, len(t_data))

def model_func(t, y0, r):
    return y0 * np.exp(r * t)

popt, pcov = curve_fit(model_func, t_data, y_data, p0=[1, 0.5])
print(f"Estimated: y0 = {popt[0]:.3f}, r = {popt[1]:.3f}")
print(f"Std errors: {np.sqrt(np.diag(pcov))}")
```

1.8 Complete Example: Cooling of a Cup of Coffee

Let us illustrate the full process on a classic example.

Step 1: Observation. A hot cup of coffee placed on a table cools down over time. We measure the temperature $T(t)$.

Step 2: Assumptions.

- The coffee temperature is spatially uniform (lumped model).
- Ambient temperature T_a is constant.
- Heat flux is proportional to the temperature difference (Newton's law of cooling).

Step 3: Model. Newton's law gives:

$$\frac{dT}{dt} = -k(T - T_a), \qquad T(0) = T_0,$$

where $k > 0$ is the heat transfer coefficient.

Step 4: Solution. This is a first-order linear ODE with solution:

$$T(t) = T_a + (T_0 - T_a)\, e^{-kt}.$$

Newton's Law of Cooling

$$T(t) = T_a + (T_0 - T_a)\, e^{-kt}, \qquad k > 0.$$

- T_0: initial temperature.
- T_a: ambient temperature.
- k: cooling constant (s^{-1}).
- Half-cooling time: $t_{1/2} = \ln 2/k$.

Step 5: Validation.

Simulation and fit — cooling

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

np.random.seed(0)
t_exp = np.arange(0, 31, 2)
T_a, T0_true, k_true = 22.0, 90.0, 0.07
T_exp = T_a + (T0_true - T_a) * np.exp(-k_true * t_exp) \
        + np.random.normal(0, 1.5, len(t_exp))

def newton_cooling(t, T0, k):
    return T_a + (T0 - T_a) * np.exp(-k * t)

popt, pcov = curve_fit(newton_cooling, t_exp, T_exp, p0=[85, 0.05])
print(f"T0 = {popt[0]:.1f} C,  k = {popt[1]:.4f} /min")

t_fine = np.linspace(0, 30, 300)
plt.figure(figsize=(8, 4))
plt.scatter(t_exp, T_exp, c='red', label='Data')
plt.plot(t_fine, newton_cooling(t_fine, *popt), 'b-',
         label='Fitted model')
plt.axhline(T_a, color='gray', ls='--', label=f'$T_a = {T_a}$ C')
plt.xlabel('Time (min)'); plt.ylabel('Temperature (C)')
plt.legend(); plt.title("Newton's Cooling Law")
plt.tight_layout(); plt.savefig('cooling.pdf')
```

1.9 Limitations and Common Pitfalls

Pitfalls to avoid

1. **Overfitting.** A model with too many parameters may fit the training data perfectly but fail at prediction.
2. **Reckless extrapolation.** A model validated on $[0, T]$ should not be extrapolated to $t \gg T$ without caution.
3. **Correlation vs. causation.** A good fit does not prove a causal link.
4. **Forgetting units.** Never add quantities with different dimensions (see Chapter 2).

1.10 Exercises

Exercise 1.1. A tank initially contains $V_0 = 1000$ L of pure water. A salt solution of concentration $c_{\text{in}} = 30$ g/L flows in at rate $q_{\text{in}} = 5$ L/min. The well-mixed solution flows out at rate $q_{\text{out}} = 5$ L/min.

1. Write the ODE for the amount of salt $Q(t)$ in the tank.
2. Solve the ODE and find the steady-state concentration.
3. Plot $Q(t)$ for $t \in [0, 400]$ min using Python.
4. Discuss what changes if $q_{\text{out}} \neq q_{\text{in}}$.

Exercise 1.2. Consider the model $y(t) = A \sin(\omega t + \varphi)$ for a periodic signal.

1. Generate synthetic noisy data with $A = 3$, $\omega = 2\pi/5$, $\varphi = \pi/4$, and Gaussian noise of standard deviation 0.5.
2. Fit the model by least squares.
3. Compute 95% confidence intervals for A, ω, φ.

Exercise 1.3. For Newton's cooling model:

1. Show that $T(t)$ is strictly decreasing if $T_0 > T_a$ and strictly increasing if $T_0 < T_a$.
2. Compute the sensitivity index S_k and interpret.

3. Modify the model to account for radiation (Stefan–Boltzmann: flux $\propto T^4 - T_a^4$). Solve numerically and compare.

Exercise 1.4. The number of atoms $N(t)$ in a radioactive substance satisfies $dN/dt = -\lambda N$.

1. Show that the half-life is $t_{1/2} = \ln 2/\lambda$.
2. Write a Python program that estimates λ from data (t_i, N_i) by (a) nonlinear least squares and (b) linear regression on $\ln N$.
3. Compare the two methods and discuss their merits.

Chapter 2

Dimensional Analysis and Scaling

2.1 Introduction

In 1883, Osborne Reynolds was studying the flow of water through pipes at the University of Manchester. He noticed that the transition from smooth, laminar flow to chaotic, turbulent flow did not depend separately on the pipe diameter, the flow speed, or the fluid viscosity—it depended on a single dimensionless combination of all three, now called the *Reynolds number*. This was not a coincidence. It was a manifestation of a deep principle: the laws of physics cannot depend on the arbitrary units we choose to measure them in.

Dimensional analysis turns this seemingly obvious observation into a remarkably powerful tool. The Buckingham Π theorem, formalised by Edgar Buckingham in 1914, states that any physically meaningful relationship among n variables involving k independent dimensions can be rewritten as a relationship among $n - k$ dimensionless groups. The consequences are far-reaching: models are simplified, the number of experiments needed is drastically reduced, and scaling laws emerge naturally. From wind tunnel tests of aircraft to the energy of nuclear explosions (famously estimated by G. I. Taylor from a single photograph), dimensional analysis reveals structure hidden in plain sight.

2.2 Dimensions and Units

Definition 2.1 (Physical dimension)**.** Every physical quantity possesses a **dimension** expressed in terms of fundamental dimensions. In the SI system, the base dimensions are:

Quantity	Dimensional symbol	SI unit
Length	L	m
Mass	M	kg
Time	T	s
Temperature	Θ	K
Amount of substance	N	mol
Electric current	I	A

Definition 2.2 (Dimensional homogeneity)**.** A physical equation is **dimensionally homogeneous** if every term has the same dimension. This is a necessary (but not sufficient) condition for validity.

Example 2.3. Velocity v has dimension $[v] = LT^{-1}$. Kinetic energy $E_k = \frac{1}{2}mv^2$ has dimension $[E_k] = ML^2T^{-2}$, which is indeed the dimension of energy.

Remark 2.4. Arguments of transcendental functions (exp, ln, sin, etc.) must be dimensionless. Thus in $e^{-t/\tau}$, the ratio t/τ is dimensionless.

2.3 The Buckingham Π Theorem

Theorem 2.5 (Buckingham Π)**.** *Let a physical relationship involve n quantities $q_1, \dots, q_n$ whose dimensions are expressed using k independent fundamental dimensions. Then the relationship can be written as an equation among $n-k$ dimensionless groups $\Pi_1, \dots, \Pi_{n-k}$:*

$$f(\Pi_1, \Pi_2, \dots, \Pi_{n-k}) = 0.$$

Remark 2.6. The number k is the rank of the **dimensional matrix**, whose columns contain the exponents of the fundamental dimensions for each quantity.

2.3.1 Practical Procedure

1. List all n quantities involved in the problem.
2. Build the dimensional matrix and compute its rank k.
3. Form $n - k$ dimensionless groups Π_j by combining quantities.
4. Write the physical law as $\Pi_1 = \phi(\Pi_2, \dots, \Pi_{n-k})$.

Example 2.7 (Period of a simple pendulum)**.** The period τ depends on the length ℓ, mass m, gravitational acceleration g, and amplitude θ_0 (dimensionless). The dimensioned quantities are τ, ℓ, m, g ($n = 4$). Fundamental dimensions: M, L, T ($k = 3$). Dimensional matrix:

$$\left(\begin{array}{ccccc} & \tau & \ell & m & g \\ M & 0 & 0 & 1 & 0 \\ L & 0 & 1 & 0 & 1 \\ T & 1 & 0 & 0 & -2 \end{array}\right)$$

Rank $k = 3$, giving $n - k = 1$ dimensionless group:

$$\Pi_1 = \tau\sqrt{g/\ell}.$$

Conclusion: $\tau = C(\theta_0)\sqrt{\ell/g}$ for some function C. For small oscillations, $C \approx 2\pi$. Mass does not appear.

2.4 Non-dimensionalisation of Equations

Definition 2.8 (Non-dimensionalisation)**.** To **non-dimensionalise** an equation means to introduce dimensionless variables by dividing each quantity by a characteristic scale:

$$\tilde{x} = \frac{x}{x_c}, \qquad \tilde{t} = \frac{t}{t_c}, \qquad \tilde{u} = \frac{u}{u_c}.$$

The resulting equation involves only **dimensionless numbers** that reveal the relative importance of different terms.

Example 2.9 (Damped harmonic oscillator)**.** The equation

$$m\ddot{x} + c\dot{x} + kx = 0$$

involves mass m, damping c, stiffness k. Let x_c be a characteristic amplitude and $t_c = \sqrt{m/k}$. With $\tilde{x} = x/x_c$, $\tilde{t} = t/t_c$:

$$\frac{d^2\tilde{x}}{d\tilde{t}^2} + 2\zeta\frac{d\tilde{x}}{d\tilde{t}} + \tilde{x} = 0, \qquad \zeta = \frac{c}{2\sqrt{mk}}.$$

The only remaining parameter is the **damping ratio** ζ:

- $\zeta < 1$: underdamped (oscillatory decay),
- $\zeta = 1$: critically damped,
- $\zeta > 1$: overdamped (no oscillation).

Classic Dimensionless Numbers

Name	Expression	Meaning
Reynolds	$\mathrm{Re} = \rho v L/\mu$	inertia / viscosity
Péclet	$\mathrm{Pe} = vL/D$	advection / diffusion
Mach	$\mathrm{Ma} = v/c_s$	velocity / sound speed
Froude	$\mathrm{Fr} = v/\sqrt{gL}$	inertia / gravity
Damköhler	$\mathrm{Da} = \tau_{\text{transp}}/\tau_{\text{react}}$	transport / reaction

2.5 Scaling Laws and Similitude

Definition 2.10 (Similitude)**.** Two physical systems are said to be in **similitude** if they share the same dimensionless numbers. The study of a reduced model (prototype) can then be transferred to the full-scale system.

Theorem 2.11 (Scale invariance)**.** *If a quantity y depends on variables $x_1, \ldots, x_n$ through a power law:*

$$y = C\, x_1^{a_1}\, x_2^{a_2} \cdots x_n^{a_n},$$

then this law is invariant under change of units if and only if the exponents satisfy the dimensional constraints.

Example 2.12 (Drag law)**.** The drag force F_D on a body depends on ρ (fluid density), v (velocity), L (characteristic length), and μ (viscosity). We have $n = 5$, $k = 3$ (M, L, T), hence $n - k = 2$ groups:

$$\Pi_1 = \frac{F_D}{\rho v^2 L^2}, \qquad \Pi_2 = \frac{\rho v L}{\mu} = \mathrm{Re}.$$

Thus $C_D = F_D/(\frac{1}{2}\rho v^2 A)$ is a function of Re alone (for a given geometry).

2.6 Application: Diffusion Equation

Consider the one-dimensional diffusion equation:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}, \qquad u(x,0) = u_0(x),$$

on a domain of length L. Set $\tilde{x} = x/L$, $\tilde{t} = Dt/L^2$, $\tilde{u} = u/U$. The equation becomes:

$$\frac{\partial \tilde{u}}{\partial \tilde{t}} = \frac{\partial^2 \tilde{u}}{\partial \tilde{x}^2}.$$

All parameters have disappeared: the characteristic diffusion time is $t_c = L^2/D$.

Diffusion time in Python

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

L = 1.0; D = 0.01; N = 100
dx = L / (N + 1)
x = np.linspace(dx, L - dx, N)

u0 = np.exp(-((x - 0.5)**2) / (2 * 0.02**2))

def rhs(t, u):
    d2u = np.zeros_like(u)
    d2u[0] = (0 - 2*u[0] + u[1]) / dx**2
    d2u[-1] = (u[-2] - 2*u[-1] + 0) / dx**2
    d2u[1:-1] = (u[:-2] - 2*u[1:-1] + u[2:]) / dx**2
    return D * d2u

sol = solve_ivp(rhs, [0, 5], u0, t_eval=[0, 0.5, 1, 2, 5],
                method='RK45', max_step=0.01)

plt.figure(figsize=(8, 5))
for i, t_val in enumerate(sol.t):
    plt.plot(x, sol.y[:, i], label=f'$t = {t_val}$')
plt.xlabel('$x$'); plt.ylabel('$u(x,t)$')
plt.legend(); plt.title('1D Diffusion')
plt.tight_layout(); plt.savefig('diffusion_1d.pdf')
```

2.7 Regular Perturbations and Orders of Magnitude

Non-dimensionalisation often reveals a small parameter $\varepsilon \ll 1$ enabling a **perturbation expansion**.

Definition 2.13 (Regular perturbation expansion)**.** One seeks the solution in the form

$$\tilde{u} = \tilde{u}_0 + \varepsilon\, \tilde{u}_1 + \varepsilon^2\, \tilde{u}_2 + \cdots$$

and identifies terms order by order.

Example 2.14. The weakly nonlinear oscillator:

$$\ddot{x} + x + \varepsilon x^3 = 0, \qquad 0 < \varepsilon \ll 1.$$

At leading order: $\ddot{x}_0 + x_0 = 0 \Rightarrow x_0 = A\cos t$. At first order:

$$\ddot{x}_1 + x_1 = -x_0^3 = -A^3\cos^3 t = -\frac{A^3}{4}(3\cos t + \cos 3t).$$

The $\cos t$ term is resonant (secular term), signalling that the naive expansion breaks down at long times. One must use multiple scales or the Lindstedt–Poincaré method.

Secular terms

A regular perturbation expansion may contain terms that grow without bound in time (secular terms). This invalidates the approximation at long times and requires specialised techniques.

2.8 Order-of-Magnitude Analysis

Proposition 2.15 (Simplification by orders of magnitude)**.** After non-dimensionalisation, if a term $\varepsilon f(\tilde{x})$ is multiplied by $\varepsilon \ll 1$ while the other terms are $O(1)$, it may be neglected as a first approximation.

Example 2.16 (Low-Reynolds-number flow)**.** For an incompressible Newtonian fluid, the non-dimensionalised Navier–Stokes equations contain the term $\frac{1}{\mathrm{Re}}\nabla^2\mathbf{v}$. If $\mathrm{Re} \gg 1$, viscosity is negligible (Euler flow). If $\mathrm{Re} \ll 1$, inertia is negligible (Stokes flow):

$$\nabla p = \mu\nabla^2\mathbf{v}.$$

2.9 Exercises

Exercise 2.1. Using dimensional analysis, determine how the oscillation frequency f of a spherical droplet depends on its mass m, radius R, and surface tension γ.

Exercise 2.2. Non-dimensionalise the Lotka–Volterra equations:

$$\dot{x} = ax - bxy, \qquad \dot{y} = -cy + dxy,$$

by setting $\tilde{x} = bx/c$, $\tilde{y} = dy/a$, $\tilde{t} = at$. Verify that the resulting system depends on a single parameter $\alpha = c/a$.

Exercise 2.3. The sedimentation velocity v_s of a sphere of radius R and density ρ_s in a fluid of viscosity μ and density ρ_f under gravity g is given by Stokes' formula.

1. Recover $v_s \propto R^2$ by dimensional analysis.
2. For what values of Re is this formula valid?

Exercise 2.4. Consider the heat equation with a source:

$$\rho c_p \frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} + Q_0 e^{-x/\ell}.$$

CHAPTER 2. DIMENSIONAL ANALYSIS AND SCALING

1. Identify characteristic scales x_c, t_c, T_c.
2. Non-dimensionalise and identify the dimensionless numbers.
3. Discuss the limiting regimes.

Exercise 2.5. Using the Buckingham Π theorem, show that the pressure drop Δp in a pipe of length L and diameter d, for a fluid of density ρ and viscosity μ flowing at velocity v, takes the form:

$$\frac{\Delta p}{\rho v^2} = \phi\left(\mathrm{Re}, \frac{L}{d}\right).$$

Chapter 3

Population Dynamics Models

3.1 Introduction

In 1798, the Reverend Thomas Malthus published his famous *Essay on the Principle of Population*, arguing that population grows geometrically while resources grow only arithmetically. This pessimistic vision—the "Malthusian catastrophe"—would inspire Darwin and become the first mathematical model of population dynamics: the exponential. But Malthus was wrong on one crucial point: growth cannot be exponential forever. In 1838, Pierre-François Verhulst introduced a saturation term leading to the celebrated logistic equation. Then, in the 1920s, Alfred Lotka and Vito Volterra, working independently—one on chemical reactions, the other on Adriatic fisheries—discovered the predator-prey model that bears their names.

This chapter retraces this intellectual adventure and develops the fundamental models — Malthus, Verhulst, Lotka–Volterra — emphasising derivation from first principles, qualitative analysis (phase portraits, stability), and numerical simulations.

3.2 Exponential Growth: The Malthus Model

Definition 3.1 (Malthus model)**.** Let $N(t)$ be the population size at time t. If the birth rate b and death rate d are constant and proportional to population size:

$$\frac{dN}{dt} = rN, \qquad r = b - d, \qquad N(0) = N_0.$$

Theorem 3.2 (Solution of the Malthus model)**.** *The solution is* $N(t) = N_0\, e^{rt}$*.*

- $r > 0$*: exponential growth (explosion).*
- $r = 0$*: constant population.*
- $r < 0$*: exponential decay (extinction).*

Remark 3.3. The Malthus model is realistic only over short periods. No real population can grow exponentially indefinitely.

Malthus model in Python

```python
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 10, 200)
N0 = 100

for r in [-0.3, 0, 0.2, 0.5]:
    N = N0 * np.exp(r * t)
    plt.plot(t, N, label=f'$r = {r}$')

plt.xlabel('Time $t$'); plt.ylabel('Population $N(t)$')
plt.legend(); plt.title('Malthus Model')
plt.ylim(0, 2000)
plt.tight_layout(); plt.savefig('malthus.pdf')
```

3.3 Logistic Growth: The Verhulst Model

Definition 3.4 (Verhulst logistic model)**.** To model resource limitation, Verhulst (1838) proposed:

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right), \qquad N(0) = N_0,$$

where $K > 0$ is the **carrying capacity**.

Interpreting the logistic term

The factor $(1 - N/K)$ models intraspecific competition: when $N \ll K$, growth is nearly exponential; as N approaches K, the effective growth rate tends to zero.

Theorem 3.5 (Solution of the logistic equation)**.** *The logistic ODE is separable. Its solution is:*

$$N(t) = \frac{K}{1 + \left(\frac{K}{N_0} - 1\right) e^{-rt}}.$$

We have $\lim_{t\to+\infty} N(t) = K$ *for all* $N_0 > 0$.

Proof. Separate variables:

$$\frac{dN}{N(1 - N/K)} = r\,dt.$$

Partial fractions:

$$\frac{1}{N(1 - N/K)} = \frac{1}{N} + \frac{1/K}{1 - N/K}.$$

Integration:

$$\ln N - \ln(1 - N/K) = rt + C.$$

Setting $C = \ln \frac{N_0}{1 - N_0/K}$ yields the result. □

3.3.1 Qualitative Analysis

Equilibria solve $rN(1 - N/K) = 0$: $N^* = 0$ and $N^* = K$.

Proposition 3.6 (Stability of logistic equilibria)**.**

- $N^* = 0$ is **unstable** (for $r > 0$).
- $N^* = K$ is **asymptotically stable**.

This follows from linearisation: $f'(N) = r(1-2N/K)$, giving $f'(0) = r > 0$ (unstable) and $f'(K) = -r < 0$ (stable).

Logistic growth and phase portrait

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

r, K = 0.5, 1000

def logistic(N, t):
    return r * N * (1 - N / K)

t = np.linspace(0, 30, 500)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

for N0 in [10, 50, 200, 500, 1200, 1500]:
    N = odeint(logistic, N0, t).flatten()
    ax1.plot(t, N, label=f'$N_0={N0}$')

ax1.axhline(K, color='gray', ls='--', label=f'$K={K}$')
ax1.set_xlabel('$t$'); ax1.set_ylabel('$N(t)$')
ax1.legend(fontsize=8); ax1.set_title('Trajectories')

N_vals = np.linspace(0, 1600, 300)
dNdt = r * N_vals * (1 - N_vals / K)
ax2.plot(N_vals, dNdt, 'b-')
ax2.axhline(0, color='gray', ls='--')
ax2.plot([0, K], [0, 0], 'ro', markersize=8)
ax2.set_xlabel('$N$'); ax2.set_ylabel("$dN/dt$")
ax2.set_title('Velocity field')
plt.tight_layout(); plt.savefig('logistic.pdf')
```

3.4 The Lotka–Volterra Predator–Prey Model

Definition 3.7 (Lotka–Volterra system)**.** Let $x(t)$ be the prey population and $y(t)$ the predator population. The classical Lotka–Volterra system is:

$$\dot{x} = ax - bxy, \tag{3.1}$$
$$\dot{y} = -cy + dxy, \tag{3.2}$$

where $a, b, c, d > 0$.

CHAPTER 3. POPULATION DYNAMICS MODELS

- a: intrinsic prey growth rate.
- b: predation rate.
- c: predator death rate.
- d: conversion efficiency prey $\to$ predator.

3.4.1 Equilibria

Setting the right-hand side to zero:

$$x(a - by) = 0, \qquad y(-c + dx) = 0.$$

Proposition 3.8 (Lotka–Volterra equilibria)**.** There are two equilibria:

1. $E_0 = (0, 0)$: total extinction.
2. $E^* = (c/d,\ a/b)$: coexistence.

3.4.2 Linear Stability

The Jacobian is:

$$J(x, y) = \begin{pmatrix} a - by & -bx \\ dy & -c + dx \end{pmatrix}.$$

Theorem 3.9 (Stability of Lotka–Volterra equilibria)**.** *1. At $E_0 = (0, 0)$: $J = \begin{pmatrix} a & 0 \\ 0 & -c \end{pmatrix}$, eigenvalues $\lambda_1 = a > 0$, $\lambda_2 = -c < 0$. E_0 is a* ***saddle point*** *(unstable).*

2. At $E^* = (c/d, a/b)$: $J = \begin{pmatrix} 0 & -bc/d \\ da/b & 0 \end{pmatrix}$, eigenvalues $\lambda = \pm i\sqrt{ac}$. E^* is a ***centre*** *(periodic orbits).*

3.4.3 First Integral and Closed Trajectories

Theorem 3.10 (First integral)**.** *The Lotka–Volterra system admits the first integral:*

$$H(x, y) = dx - c \ln x + by - a \ln y = \textit{constant}.$$

Phase-plane trajectories are level curves of H, forming closed curves around E^*.

Proof. Compute dH/dt:

$$\begin{aligned}
\frac{dH}{dt} &= d\dot{x} - \frac{c}{x}\dot{x} + b\dot{y} - \frac{a}{y}\dot{y} \\
&= d(ax - bxy) - \frac{c}{x}(ax - bxy) + b(-cy + dxy) - \frac{a}{y}(-cy + dxy) \\
&= dax - dbxy - ca + cby - bcy + bdxy + ac - adx = 0.
\end{aligned}$$

□

Lotka–Volterra simulation

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

a, b, c, d = 1.0, 0.1, 1.5, 0.075

def lotka_volterra(t, z):
    x, y = z
    return [a*x - b*x*y, -c*y + d*x*y]

t_span = (0, 40)
z0 = [40, 9]
sol = solve_ivp(lotka_volterra, t_span, z0,
                t_eval=np.linspace(0, 40, 1000),
                method='RK45', rtol=1e-10)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 5))

ax1.plot(sol.t, sol.y[0], 'b-', label='Prey $x(t)$')
ax1.plot(sol.t, sol.y[1], 'r-', label='Predators $y(t)$')
ax1.set_xlabel('$t$'); ax1.set_ylabel('Population')
ax1.legend(); ax1.set_title('Time series')

for x0 in [10, 20, 30, 40, 60]:
    sol_i = solve_ivp(lotka_volterra, (0, 50), [x0, 9],
                      t_eval=np.linspace(0, 50, 2000),
                      method='RK45', rtol=1e-10)
    ax2.plot(sol_i.y[0], sol_i.y[1], 'b-', alpha=0.6)

ax2.plot(c/d, a/b, 'ro', markersize=8, label="$E^*$")
ax2.set_xlabel('Prey $x$'); ax2.set_ylabel('Predators $y$')
ax2.legend(); ax2.set_title('Phase portrait')
plt.tight_layout(); plt.savefig('lotka_volterra.pdf')
```

3.5 Extensions of the Lotka–Volterra Model

3.5.1 Holling Functional Response

Definition 3.11 (Functional responses)**.** The functional response $g(x)$ models the per-predator predation rate as a function of prey density:

- **Type I (linear):** $g(x) = bx$ (classical Lotka–Volterra).
- **Type II (saturation):** $g(x) = \frac{bx}{1+bhx}$ where h is the handling time.
- **Type III (sigmoidal):** $g(x) = \frac{bx^2}{1+bhx^2}$.

CHAPTER 3. POPULATION DYNAMICS MODELS

The Type II system (Rosenzweig–MacArthur model) is:

$$\dot{x} = rx\left(1 - \frac{x}{K}\right) - \frac{bxy}{1 + bhx}, \tag{3.3}$$

$$\dot{y} = y\left(\frac{dbx}{1 + bhx} - c\right). \tag{3.4}$$

Paradox of enrichment

Rosenzweig (1971) showed that increasing the carrying capacity K can destabilise the coexistence equilibrium through a Hopf bifurcation, causing large-amplitude oscillations that may lead to extinction.

3.5.2 Interspecific Competition

Definition 3.12 (Lotka–Volterra competition model)**.** For two competing species:

$$\dot{N}_1 = r_1 N_1\left(1 - \frac{N_1 + \alpha_{12}N_2}{K_1}\right), \tag{3.5}$$

$$\dot{N}_2 = r_2 N_2\left(1 - \frac{N_2 + \alpha_{21}N_1}{K_2}\right), \tag{3.6}$$

where α_{ij} measures the competitive effect of species j on species i.

Proposition 3.13 (Competitive exclusion principle)**.** If $\alpha_{12}K_2/K_1 > 1$ and $\alpha_{21}K_1/K_2 > 1$, the two species cannot stably coexist (Gause's competitive exclusion). The outcome depends on initial conditions.

3.6 Discrete Models: Beverton–Holt Equation

Definition 3.14 (Beverton–Holt model)**.** For populations with non-overlapping generations:

$$N_{t+1} = \frac{RN_t}{1 + (R-1)N_t/K},$$

where $R > 1$ is the reproduction rate and K the carrying capacity.

Proposition 3.15. The nontrivial equilibrium $N^* = K$ is globally stable for $R > 1$. The Beverton–Holt model is the discrete analogue of the logistic equation and does *not* exhibit chaos, unlike the logistic map $N_{t+1} = RN_t(1 - N_t/K)$ studied in Chapter 5.

3.7 Limitations of Population Models

Assumptions and limitations

- **Spatial homogeneity:** the above models ignore spatial structure. In reality, populations are distributed in space (see Chapter 6).
- **Determinism:** for small populations, stochastic fluctuations become important (Chapter 9).

- **Age structure:** Leslie models introduce age classes.
- **Constant parameters:** in reality, r and K may depend on time (seasonality, climate change).

3.8 Exercises

Exercise 3.1. Show that the logistic population reaches half the carrying capacity at time $t_{1/2} = \frac{1}{r} \ln \frac{K-N_0}{N_0}$.

Exercise 3.2. Add constant harvesting h to the logistic model: $\dot{N} = rN(1 - N/K) - h$.

1. Find the equilibria as a function of h.
2. Determine the maximum sustainable yield h_{max}.
3. Draw the bifurcation diagram N^* vs. h.
4. Simulate in Python for $h < h_{\text{max}}$, $h = h_{\text{max}}$, $h > h_{\text{max}}$.

Exercise 3.3. For the Lotka–Volterra system, verify numerically that $H(x, y)$ is conserved over time. Compare results from `RK45` and `RK23` in `solve_ivp` for different tolerances.

Exercise 3.4. Study the Rosenzweig–MacArthur model with parameters $r = 1$, $b = 0.5$, $h = 0.5$, $d = 0.5$, $c = 0.3$ for $K \in \{5, 10, 20, 50\}$.

1. Find the equilibria and their stability.
2. Draw phase portraits.
3. Identify the critical value of K for the Hopf bifurcation.

Exercise 3.5. Two bacterial species compete in a confined environment with $r_1 = 0.8$, $r_2 = 0.6$, $K_1 = 500$, $K_2 = 400$, $\alpha_{12} = 0.7$, $\alpha_{21} = 1.2$.

1. Find the equilibria and analyse their stability.
2. Simulate the system for different initial conditions.
3. Conclude: is there coexistence or exclusion?

CHAPTER 3. POPULATION DYNAMICS MODELS

Chapter 4

Epidemiological Models

4.1 Introduction

In 1927, William Ogilvy Kermack and Anderson Gray McKendrick published a paper that would found mathematical epidemiology: their SIR model divides the population into three compartments—Susceptible, Infectious, Recovered—and predicts, with remarkable simplicity, the course of an epidemic. The basic reproduction number R_0 emerges naturally: if $R_0 > 1$, the epidemic spreads; if $R_0 < 1$, it dies out. This threshold, which became famous during the COVID-19 pandemic, is a purely mathematical result that guides public health policy.

Compartmental epidemiological models allow us to understand the spread of infectious diseases and evaluate the impact of interventions (vaccination, quarantine). This chapter develops the SIR and SEIR models from first principles, analyses the basic reproduction number $\mathcal{R}_0$, and provides Python simulations.

4.2 The Kermack–McKendrick SIR Model

Definition 4.1 (SIR compartments)**.** The total population N (constant) is divided into three compartments:

- $S(t)$: **Susceptible** (individuals who can be infected).
- $I(t)$: **Infectious** (contagious individuals).
- $R(t)$: **Recovered** (immunised or deceased).

Conservation constraint: $S(t) + I(t) + R(t) = N$.

4.2.1 Model Derivation

SIR assumptions

- The population is homogeneous and well-mixed.
- The transmission rate is proportional to the product SI (mass action).
- Infectious individuals recover at constant rate γ.
- No births, natural deaths, or demography.

CHAPTER 4. EPIDEMIOLOGICAL MODELS

The SIR equations are:

$$\frac{dS}{dt} = -\beta \frac{SI}{N}, \tag{4.1}$$
$$\frac{dI}{dt} = \beta \frac{SI}{N} - \gamma I, \tag{4.2}$$
$$\frac{dR}{dt} = \gamma I, \tag{4.3}$$

where $\beta > 0$ is the transmission rate and $\gamma > 0$ the recovery rate.

SIR model parameters

Parameter	Meaning	Unit
β	Transmission rate	time^{-1}
γ	Recovery rate	time^{-1}
$1/\gamma$	Mean infectious period	time
$\mathcal{R}_0 = \beta/\gamma$	Basic reproduction number	dimensionless

4.3 The Basic Reproduction Number $\mathcal{R}_0$

Definition 4.2 (Basic reproduction number)**.** The **basic reproduction number** $\mathcal{R}_0$ is the average number of secondary cases produced by one infectious individual in a fully susceptible population:

$$\mathcal{R}_0 = \frac{\beta}{\gamma}.$$

Theorem 4.3 (Epidemic threshold)**.** *An epidemic occurs if and only if $\mathcal{R}_0 > 1$. More precisely, $dI/dt > 0$ at the start of the epidemic if and only if $\mathcal{R}_0 S(0)/N > 1$.*

Proof. At the start, $S \approx N$. Equation (4.2) gives:

$$\left.\frac{dI}{dt}\right|_{t=0} \approx (\beta - \gamma)I_0 = \gamma(\mathcal{R}_0 - 1)I_0.$$

Thus $dI/dt > 0$ if and only if $\mathcal{R}_0 > 1$. □

Remark 4.4. Typical values of $\mathcal{R}_0$: measles ≈ 12–18, influenza ≈ 1.5–3, COVID-19 (original strain) ≈ 2.5–3.5, Ebola ≈ 1.5–2.5.

4.4 Qualitative Analysis of the SIR Model

4.4.1 Reduction to Two Equations

Since $R = N - S - I$, it suffices to study the (S, I) system in the triangle $\{S \geq 0,\ I \geq 0,\ S + I \leq N\}$.

4.4.2 Final Size of the Epidemic

Theorem 4.5 (Final size equation). *If $I(0) = I_0 \ll N$ and $S(0) \approx N$, the final fraction of susceptibles $s_\infty = S(\infty)/N$ satisfies the transcendental equation:*

$$\ln s_\infty = \mathcal{R}_0(s_\infty - 1).$$

Proof. Dividing dS/dI:

$$\frac{dS}{dI} = \frac{-\beta SI/N}{\beta SI/N - \gamma I} = \frac{-\beta S/N}{\beta S/N - \gamma}.$$

Setting $s = S/N$:

$$\frac{ds}{dI} = \frac{-\mathcal{R}_0 s}{\mathcal{R}_0 s - 1} \cdot \frac{1}{N}.$$

Integrating and using $I(\infty) = 0$ yields the final size relation. □

SIR model simulation

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 10000
beta, gamma = 0.3, 0.1  # R0 = 3
S0, I0, R0_init = N - 10, 10, 0

def sir(t, y):
    S, I, R = y
    dS = -beta * S * I / N
    dI = beta * S * I / N - gamma * I
    dR = gamma * I
    return [dS, dI, dR]

sol = solve_ivp(sir, [0, 200], [S0, I0, R0_init],
                t_eval=np.linspace(0, 200, 1000))

plt.figure(figsize=(10, 5))
plt.plot(sol.t, sol.y[0], 'b-', label='$S(t)$')
plt.plot(sol.t, sol.y[1], 'r-', label='$I(t)$')
plt.plot(sol.t, sol.y[2], 'g-', label='$R(t)$')
plt.xlabel('Time (days)'); plt.ylabel('Number of individuals')
plt.legend(); plt.title(f'SIR Model ($\\mathcal{{R}}_0 =
↪ {beta/gamma:.1f}$)')
plt.tight_layout(); plt.savefig('sir.pdf')
```

4.5 Vaccination and Herd Immunity

Theorem 4.6 (Herd immunity threshold). *If a fraction p of the population is vaccinated (and immunised), the fraction of susceptibles is $S(0)/N = 1 - p$. The epidemic does not start if $(1-p)\mathcal{R}_0 < 1$, i.e.:*

$$p > p_c = 1 - \frac{1}{\mathcal{R}_0}.$$

Example 4.7. For measles ($\mathcal{R}_0 \approx 15$): $p_c = 1 - 1/15 \approx 93.3\%$. Over 93% of the population must be vaccinated.

Effect of vaccination

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 10000
beta, gamma = 0.3, 0.1

def sir_vacc(t, y):
    S, I, R = y
    return [-beta*S*I/N, beta*S*I/N - gamma*I, gamma*I]

fig, axes = plt.subplots(2, 2, figsize=(12, 8))
vaccination_rates = [0, 0.3, 0.6, 0.75]

for ax, p in zip(axes.flat, vaccination_rates):
    S0 = N * (1 - p) - 10
    R0_init = N * p
    I0 = 10
    sol = solve_ivp(sir_vacc, [0, 200], [S0, I0, R0_init],
                    t_eval=np.linspace(0, 200, 500))
    ax.plot(sol.t, sol.y[1], 'r-', lw=2)
    ax.set_title(f'$p = {p:.0%}$, $I_{{\\max}} = {max(sol.y[1]):.0f}$')
    ax.set_xlabel('Time'); ax.set_ylabel('$I(t)$')

plt.suptitle(f'Effect of Vaccination ($\\mathcal{{R}}_0 =
↪   {beta/gamma:.1f}$)')
plt.tight_layout(); plt.savefig('vaccination.pdf')
```

4.6 The SEIR Model

Definition 4.8 (SEIR model)**.** A compartment **Exposed** (E) is added for individuals infected but not yet contagious (latency period $1/\sigma$):

$$\frac{dS}{dt} = -\beta\frac{SI}{N}, \tag{4.4}$$

$$\frac{dE}{dt} = \beta\frac{SI}{N} - \sigma E, \tag{4.5}$$

$$\frac{dI}{dt} = \sigma E - \gamma I, \tag{4.6}$$

$$\frac{dR}{dt} = \gamma I. \tag{4.7}$$

The basic reproduction number remains $\mathcal{R}_0 = \beta/\gamma$.

SEIR model simulation

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 10000
beta, sigma, gamma = 0.5, 0.2, 0.1
S0, E0, I0, R0_init = N - 10, 0, 10, 0

def seir(t, y):
    S, E, I, R = y
    dS = -beta * S * I / N
    dE = beta * S * I / N - sigma * E
    dI = sigma * E - gamma * I
    dR = gamma * I
    return [dS, dE, dI, dR]

sol = solve_ivp(seir, [0, 200], [S0, E0, I0, R0_init],
                t_eval=np.linspace(0, 200, 1000))

plt.figure(figsize=(10, 5))
plt.plot(sol.t, sol.y[0], 'b-', label='$S$')
plt.plot(sol.t, sol.y[1], color='orange', label='$E$')
plt.plot(sol.t, sol.y[2], 'r-', label='$I$')
plt.plot(sol.t, sol.y[3], 'g-', label='$R$')
plt.xlabel('Time'); plt.ylabel('Population')
plt.legend(); plt.title(f'SEIR Model ($\\mathcal{{R}}_0 =
↪  {beta/gamma}$)')
plt.tight_layout(); plt.savefig('seir.pdf')
```

4.7 Extensions and Advanced Models

4.7.1 SIR with Demography

Definition 4.9 (SIR with demography)**.** Adding births and natural deaths at rate μ:

$$\frac{dS}{dt} = \mu N - \beta\frac{SI}{N} - \mu S, \tag{4.8}$$

$$\frac{dI}{dt} = \beta\frac{SI}{N} - (\gamma + \mu)I, \tag{4.9}$$

$$\frac{dR}{dt} = \gamma I - \mu R. \tag{4.10}$$

The basic reproduction number becomes $\mathcal{R}_0 = \beta/(\gamma + \mu)$.

Proposition 4.10 (Endemic equilibrium)**.** For $\mathcal{R}_0 > 1$, there exists an endemic equilibrium (disease permanently present):

$$S^* = \frac{N}{\mathcal{R}_0}, \qquad I^* = \frac{\mu N}{\gamma + \mu}\left(1 - \frac{1}{\mathcal{R}_0}\right).$$

4.7.2 SIRS Model (Waning Immunity)

If immunity is temporary (mean duration $1/\delta$), recovered individuals become susceptible again:

$$\frac{dR}{dt} = \gamma I - \delta R, \qquad \frac{dS}{dt} = -\beta\frac{SI}{N} + \delta R.$$

This model can produce oscillations.

4.8 Fitting to Real Data

SIR fitting to data

```python
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import minimize

np.random.seed(42)
N = 10000; beta_true, gamma_true = 0.3, 0.1
sol_true = solve_ivp(lambda t, y: [-beta_true*y[0]*y[1]/N,
                     beta_true*y[0]*y[1]/N - gamma_true*y[1],
                     gamma_true*y[1]],
                     [0, 100], [N-10, 10, 0],
                     t_eval=np.arange(0, 100, 7))
I_obs = sol_true.y[1] + np.random.normal(0, 50, len(sol_true.t))
I_obs = np.maximum(I_obs, 0)

def cost(params):
    b, g = params
    if b <= 0 or g <= 0: return 1e10
    sol = solve_ivp(lambda t, y: [-b*y[0]*y[1]/N,
                    b*y[0]*y[1]/N - g*y[1], g*y[1]],
                    [0, 100], [N-10, 10, 0],
                    t_eval=np.arange(0, 100, 7))
    return np.sum((sol.y[1] - I_obs)**2)

result = minimize(cost, [0.2, 0.05], method='Nelder-Mead')
print(f"beta = {result.x[0]:.4f}, gamma = {result.x[1]:.4f}")
print(f"Estimated R0 = {result.x[0]/result.x[1]:.2f}")
```

4.9 Limitations of Compartmental Models

Key limitations

- **Homogeneity:** the population is assumed well-mixed; in reality, contacts are structured (social networks).
- **Constant parameters:** β may vary with seasons, public health measures, and behaviour.

- **Determinism:** for small case numbers, stochastic models are more appropriate.
- **Heterogeneity:** age, geography, and comorbidities affect transmission and severity.

4.10 Exercises

Exercise 4.1. For the SIR model with $N = 10000$, $\beta = 0.4$, $\gamma = 0.1$:

1. Compute $\mathcal{R}_0$ and the herd immunity threshold.
2. Simulate the epidemic and determine the epidemic peak.
3. Verify the final size equation numerically.

Exercise 4.2. Compare SIR and SEIR dynamics for the same β and γ with $\sigma = 0.2$. How does the latency period affect the peak and duration of the epidemic?

Exercise 4.3. Implement a stochastic SIR model (Gillespie algorithm) and compare with the deterministic model for $N = 100$ and $N = 10000$.

Exercise 4.4. Model a progressive vaccination campaign: $p(t)$ increases linearly from 0 to $p_{\max}$ over duration T_v. Compare the total number of cases for different values of T_v.

Exercise 4.5. Study the SIRS model with $\beta = 0.5$, $\gamma = 0.1$, $\delta = 0.01$.

1. Find the endemic equilibrium.
2. Show numerically the existence of damped oscillations.
3. Compare with a standard SIR model.

CHAPTER 4. EPIDEMIOLOGICAL MODELS

Chapter 5

Discrete Dynamical Systems and Chaos

In 1963, Edward Lorenz, a meteorologist at MIT, accidentally discovered that his simplified atmospheric model was extremely sensitive to initial conditions: a tiny difference in starting data produced radically different trajectories. This is the "butterfly effect," and the birth of chaos theory. But chaos is not disorder: it obeys precise rules, generates magnificent fractal structures (the Lorenz attractor, the Mandelbrot set), and possesses a rigorous mathematical theory. This chapter explores this fascinating world through discrete dynamical systems—iterations of functions—where chaos appears in its purest form.

5.1 Introduction

Discrete dynamical systems, defined by iterations $x_{n+1} = f(x_n)$, provide a remarkably simple framework in which complex behaviours emerge: bifurcations, period doubling, and deterministic chaos. This chapter studies the logistic map in depth, Lyapunov exponents, and bifurcation diagrams.

5.2 Iterations and Orbits

Definition 5.1 (Discrete dynamical system)**.** A one-dimensional **discrete dynamical system** is defined by a map $f : \mathbb{R} \to \mathbb{R}$ and the recurrence:

$$x_{n+1} = f(x_n), \qquad x_0 \text{ given.}$$

The **orbit** of x_0 is the sequence $(x_0, x_1, x_2, \ldots)$.

Definition 5.2 (Fixed point and cycle)**.**

- A **fixed point** is a point x^* such that $f(x^*) = x^*$.
- A **cycle of period** p is a set $\{x_1^*, \ldots, x_p^*\}$ with $f^p(x_i^*) = x_i^*$ and $f^k(x_i^*) \neq x_i^*$ for $0 < k < p$.

Theorem 5.3 (Stability of a fixed point)**.** *Let x^* be a fixed point of f with f differentiable at x^*.*

- *If $|f'(x^*)| < 1$, the fixed point is **asymptotically stable** (attractor).*

CHAPTER 5. DISCRETE DYNAMICAL SYSTEMS AND CHAOS

- *If $|f'(x^*)| > 1$, the fixed point is* ***unstable*** *(repeller).*
- *If $|f'(x^*)| = 1$, stability depends on higher-order terms.*

Example 5.4 (Cobweb diagram)**.** For $f(x) = \cos(x)$, the fixed point satisfies $x^* = \cos(x^*) \approx 0.7391$. We have $f'(x^*) = -\sin(x^*) \approx -0.6736$, so $|f'(x^*)| < 1$: the fixed point is stable.

Cobweb diagram

```python
import numpy as np
import matplotlib.pyplot as plt

def cobweb(f, x0, n_iter, ax, xmin=0, xmax=1):
    x = np.linspace(xmin, xmax, 500)
    ax.plot(x, f(x), 'b-', lw=2, label='$f(x)$')
    ax.plot(x, x, 'k--', lw=1, label='$y=x$')
    xn = x0
    for _ in range(n_iter):
        xn1 = f(xn)
        ax.plot([xn, xn], [xn, xn1], 'r-', lw=0.8)
        ax.plot([xn, xn1], [xn1, xn1], 'r-', lw=0.8)
        xn = xn1
    ax.set_xlabel('$x_n$'); ax.set_ylabel('$x_{n+1}$')

fig, ax = plt.subplots(figsize=(6, 6))
cobweb(np.cos, 0.1, 30, ax, xmin=-0.5, xmax=1.5)
ax.set_title('Cobweb diagram for $f(x) = \\cos(x)$')
plt.tight_layout(); plt.savefig('cobweb.pdf')
```

5.3 The Logistic Map

Definition 5.5 (Logistic map)**.** The **logistic map** is defined by:

$$f_r(x) = rx(1-x), \qquad x \in [0,1], \quad r \in [0,4].$$

It models population growth with saturation in discrete time.

5.3.1 Fixed Points

Fixed points satisfy $x = rx(1-x)$:

$$x_1^* = 0, \qquad x_2^* = 1 - \frac{1}{r} \quad (r > 1).$$

Proposition 5.6 (Stability of fixed points)**.** We have $f_r'(x) = r(1-2x)$.

- $x_1^* = 0$: $f_r'(0) = r$. Stable if $r < 1$, unstable if $r > 1$.
- $x_2^* = 1 - 1/r$: $f_r'(x_2^*) = 2 - r$. Stable if $1 < r < 3$.

5.4 Period-Doubling Cascade

Theorem 5.7 (Feigenbaum scenario)**.** *As r increases beyond 3, the fixed point x_2^* becomes unstable and gives birth to a period-2 cycle (period-doubling bifurcation). This process repeats: $2 \to 4 \to 8 \to \cdots$ The bifurcation values r_n converge geometrically to $r_\infty \approx 3.5699$ with the universal Feigenbaum ratio:*

$$\delta = \lim_{n\to\infty} \frac{r_n - r_{n-1}}{r_{n+1} - r_n} \approx 4.6692.$$

Remark 5.8. The Feigenbaum constant δ is **universal**: it does not depend on the precise form of f, provided f has a single quadratic maximum.

Bifurcation diagram of the logistic map

```python
import numpy as np
import matplotlib.pyplot as plt

r_vals = np.linspace(2.5, 4.0, 2000)
n_skip = 300
n_plot = 200

r_list, x_list = [], []

for r in r_vals:
    x = 0.5
    for _ in range(n_skip):
        x = r * x * (1 - x)
    for _ in range(n_plot):
        x = r * x * (1 - x)
        r_list.append(r)
        x_list.append(x)

plt.figure(figsize=(12, 7))
plt.scatter(r_list, x_list, s=0.01, c='black', alpha=0.5)
plt.xlabel('$r$'); plt.ylabel('$x^*$')
plt.title('Bifurcation Diagram --- Logistic Map')
plt.tight_layout(); plt.savefig('bifurcation.pdf', dpi=200)
```

5.5 Deterministic Chaos

Definition 5.9 (Chaos (Devaney))**.** A discrete dynamical system $x_{n+1} = f(x_n)$ is said to be **chaotic** in the sense of Devaney if it satisfies:

1. **Sensitive dependence on initial conditions:** $\exists\, \varepsilon > 0$ such that for every x_0 and every neighbourhood U of x_0, $\exists\, y_0 \in U$ and $\exists\, n$ with $|f^n(x_0) - f^n(y_0)| > \varepsilon$.
2. **Topological transitivity:** for every pair of non-empty open sets U, V, $\exists\, n$ with $f^n(U) \cap V \neq \emptyset$.
3. **Dense periodic orbits:** periodic points are dense.

Theorem 5.10. *The logistic map $f_4(x) = 4x(1 - x)$ is chaotic on* $[0, 1]$.

Chaos and predictability

A chaotic system is **deterministic**: there is no randomness. However, sensitive dependence on initial conditions makes long-term predictions practically impossible, as any measurement error is amplified exponentially.

5.6 Lyapunov Exponents

Definition 5.11 (Lyapunov exponent)**.** The **Lyapunov exponent** of an orbit $(x_0, x_1, \ldots)$ under f is:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} \ln |f'(x_k)| .$$

- $\lambda < 0$: convergence to an attractor (stable fixed point or cycle).
- $\lambda = 0$: marginally stable behaviour.
- $\lambda > 0$: **chaos** (sensitive dependence).

Computing the Lyapunov exponent

```python
import numpy as np
import matplotlib.pyplot as plt

def lyapunov_exponent(r, x0=0.5, n_skip=1000, n_calc=5000):
    x = x0
    for _ in range(n_skip):
        x = r * x * (1 - x)
    lyap = 0.0
    for _ in range(n_calc):
        deriv = abs(r * (1 - 2 * x))
        if deriv == 0:
            return -np.inf
        lyap += np.log(deriv)
        x = r * x * (1 - x)
    return lyap / n_calc

r_vals = np.linspace(2.5, 4.0, 2000)
lyap_vals = [lyapunov_exponent(r) for r in r_vals]

plt.figure(figsize=(12, 4))
plt.plot(r_vals, lyap_vals, 'b-', lw=0.5)
plt.axhline(0, color='red', ls='--')
plt.xlabel('$r$'); plt.ylabel('$\\lambda$')
plt.title('Lyapunov exponent of the logistic map')
plt.tight_layout(); plt.savefig('lyapunov.pdf')
```

5.7 Periodic Windows

Remark 5.12. Within the chaotic region ($r > r_\infty$), there exist **periodic windows** where the orbit becomes periodic again. The largest is the period-3 window around $r \approx 3.83$.

Theorem 5.13 (Li–Yorke, 1975)**.** *If f has a periodic point of period 3, then f has periodic orbits of* ***every*** *period ("period three implies chaos").*

5.8 The Hénon Map

Definition 5.14 (Hénon map)**.** The **Hénon map** is a two-dimensional discrete dynamical system:

$$\begin{cases} x_{n+1} = 1 - ax_n^2 + y_n, \\ y_{n+1} = bx_n. \end{cases}$$

For $a = 1.4$ and $b = 0.3$, it possesses a strange attractor of fractal dimension ≈ 1.26.

Hénon attractor

```python
import numpy as np
import matplotlib.pyplot as plt

a, b = 1.4, 0.3
n_iter = 50000
x, y = np.zeros(n_iter), np.zeros(n_iter)
x[0], y[0] = 0.1, 0.1

for n in range(n_iter - 1):
    x[n+1] = 1 - a * x[n]**2 + y[n]
    y[n+1] = b * x[n]

plt.figure(figsize=(10, 6))
plt.scatter(x[1000:], y[1000:], s=0.1, c='blue', alpha=0.5)
plt.xlabel('$x$'); plt.ylabel('$y$')
plt.title("Henon Attractor ($a=1.4$, $b=0.3$)")
plt.tight_layout(); plt.savefig('henon.pdf', dpi=200)
```

5.9 Continuous-Time Chaos: The Lorenz System

Definition 5.15 (Lorenz system)**.** The Lorenz system models atmospheric convection:

$$\dot{x} = \sigma(y - x), \tag{5.1}$$
$$\dot{y} = \rho x - y - xz, \tag{5.2}$$
$$\dot{z} = xy - \beta z. \tag{5.3}$$

For $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, it exhibits a butterfly-shaped **strange attractor**.

Lorenz attractor

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from mpl_toolkits.mplot3d import Axes3D

sigma, rho, beta_l = 10, 28, 8/3

def lorenz(t, state):
    x, y, z = state
    return [sigma*(y-x), rho*x - y - x*z, x*y - beta_l*z]

sol = solve_ivp(lorenz, [0, 50], [1, 1, 1],
                t_eval=np.linspace(0, 50, 20000),
                method='RK45', rtol=1e-10)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(sol.y[0], sol.y[1], sol.y[2], lw=0.3, color='blue')
ax.set_xlabel('$x$'); ax.set_ylabel('$y$'); ax.set_zlabel('$z$')
ax.set_title('Lorenz Attractor')
plt.tight_layout(); plt.savefig('lorenz.pdf')
```

5.10 Exercises

Exercise 5.1. For the logistic map with $r = 3.2$:

1. Find the period-2 cycle analytically.
2. Verify by simulation.
3. Show the cycle is stable by computing $(f_r^2)'$.

Exercise 5.2. Write a program that numerically computes the first bifurcation values $r_1, r_2, \ldots, r_6$ and verify convergence to the Feigenbaum ratio $\delta \approx 4.6692$.

Exercise 5.3. Plot the Lyapunov exponent of the logistic map as a function of r and verify that $\lambda > 0$ in chaotic regions and $\lambda < 0$ in periodic windows.

Exercise 5.4. For the Hénon map with different values of a (fixing $b = 0.3$), plot the attractor and compute the two Lyapunov exponents.

Exercise 5.5. Study sensitive dependence on initial conditions in the Lorenz system: plot two trajectories starting from $(1, 1, 1)$ and $(1.001, 1, 1)$ and measure the exponential divergence.

Exercise 5.6. Consider the tent map $f(x) = \min(2x, 2(1 - x))$ on $[0, 1]$.

1. Show that the Lyapunov exponent is $\lambda = \ln 2$.
2. Draw the cobweb diagram.
3. Why is this map exactly conjugate to the logistic map f_4?

Chapter 6

Diffusion and Transport Models

6.1 Introduction

Diffusion is everywhere: a dye disperses in water, heat spreads through a metal, a pollutant disseminates in the atmosphere. This fundamental phenomenon, first studied quantitatively by Adolf Fick in 1855 (by analogy with Fourier's law for heat), obeys the diffusion equation—the same heat equation, but interpreted in terms of concentration rather than temperature. Coupled with a velocity field, it becomes the advection-diffusion equation, which models the transport of substances in a moving medium. This chapter develops these models and their applications.

Diffusion and transport phenomena are ubiquitous: heat conduction, pollutant dispersion, cell migration. This chapter derives the heat equation from Fick's law, studies the advection-diffusion equation, and introduces numerical methods for parabolic PDEs.

6.2 Fick's Law and the Diffusion Equation

Definition 6.1 (Fick's law)**.** The diffusive flux J of a substance with concentration $c(x, t)$ is proportional and opposite to the concentration gradient:

$$J = -D\frac{\partial c}{\partial x},$$

where $D > 0$ is the **diffusion coefficient** (m^2/s).

Theorem 6.2 (Diffusion (heat) equation)**.** *Combining Fick's law with mass conservation $\partial c/\partial t = -\partial J/\partial x$ yields:*

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2}.$$

This is the one-dimensional heat equation.

Proof. Local conservation:

$$\frac{\partial c}{\partial t} = -\frac{\partial J}{\partial x} = -\frac{\partial}{\partial x}\left(-D\frac{\partial c}{\partial x}\right) = D\frac{\partial^2 c}{\partial x^2}.$$

□

Fundamental solution (heat kernel)

For the problem on $\mathbb{R}$ with initial condition $c(x,0) = \delta(x)$:

$$c(x,t) = \frac{1}{\sqrt{4\pi Dt}} \exp\left(-\frac{x^2}{4Dt}\right), \qquad t > 0.$$

This is a Gaussian whose standard deviation grows as $\sigma(t) = \sqrt{2Dt}$.

Diffusion and random walks

Diffusion is the macroscopic limit of a random walk. If a particle takes steps of length ℓ at rate $1/\Delta t$, the diffusion coefficient is $D = \ell^2/(2\Delta t)$.

6.3 Boundary Conditions

Definition 6.3 (Classical boundary conditions)**.** On a domain $[0, L]$, the most common boundary conditions are:

- **Dirichlet:** $c(0,t) = c_0$, $c(L,t) = c_L$ (prescribed concentration).
- **Neumann:** $\partial c/\partial x(0,t) = 0$ (zero flux, insulating wall).
- **Robin:** $-D\partial c/\partial x + hc = hc_\infty$ (convective transfer).

6.4 Finite Difference Methods

Definition 6.4 (Discretisation)**.** Discretise $[0, L]$ into $N + 1$ points $x_j = j\Delta x$ and time into steps Δt. The second derivative is approximated by:

$$\left.\frac{\partial^2 c}{\partial x^2}\right|_{x_j} \approx \frac{c_{j-1} - 2c_j + c_{j+1}}{(\Delta x)^2}.$$

Theorem 6.5 (Explicit scheme (FTCS))**.** *The Forward Time, Central Space scheme is:*

$$c_j^{n+1} = c_j^n + \mu(c_{j-1}^n - 2c_j^n + c_{j+1}^n), \qquad \mu = \frac{D\Delta t}{(\Delta x)^2}.$$

This scheme is stable if and only if $\mu \leq 1/2$.

CFL condition

The stability condition $\mu = D\Delta t/(\Delta x)^2 \leq 1/2$ links the time and space steps. For fine meshes, Δt must be very small, making the explicit scheme costly. Implicit schemes (Crank–Nicolson) remove this constraint.

1D Diffusion — explicit scheme

```python
import numpy as np
import matplotlib.pyplot as plt

L, D, T_final = 1.0, 0.01, 2.0
Nx = 50; dx = L / Nx
dt = 0.4 * dx**2 / D  # CFL: mu = 0.4 < 0.5
Nt = int(T_final / dt)
mu = D * dt / dx**2

x = np.linspace(0, L, Nx + 1)
c = np.exp(-((x - 0.5)**2) / (2 * 0.05**2))

plt.figure(figsize=(10, 6))
plt.plot(x, c, label='$t=0$')

for n in range(Nt):
    c_new = c.copy()
    c_new[1:-1] = c[1:-1] + mu * (c[:-2] - 2*c[1:-1] + c[2:])
    c_new[0] = 0; c_new[-1] = 0
    c = c_new
    if (n+1) * dt in [0.05, 0.2, 0.5, 1.0, 2.0]:
        plt.plot(x, c, label=f'$t={((n+1)*dt):.2f}$')

plt.xlabel('$x$'); plt.ylabel('$c(x,t)$')
plt.legend(); plt.title('1D Diffusion (explicit scheme)')
plt.tight_layout(); plt.savefig('diffusion_explicit.pdf')
```

6.5 The Advection-Diffusion Equation

Definition 6.6 (Advection-diffusion equation)**.** When the substance is also transported by a velocity field v:

$$\frac{\partial c}{\partial t} + v\frac{\partial c}{\partial x} = D\frac{\partial^2 c}{\partial x^2}.$$

The Péclet number $\mathrm{Pe} = vL/D$ measures the relative importance of advection to diffusion.

Proposition 6.7 (Limiting regimes)**.**

- $\mathrm{Pe} \ll 1$: diffusion dominates (heat equation).
- $\mathrm{Pe} \gg 1$: advection dominates (transport equation).
- $\mathrm{Pe} \sim 1$: both effects are comparable.

Advection-diffusion

```python
import numpy as np
import matplotlib.pyplot as plt

L, D, v = 1.0, 0.005, 0.5
```

```
Nx = 200; dx = L / Nx
dt = 0.001; Nt = 1000

x = np.linspace(0, L, Nx + 1)
c = np.exp(-((x - 0.2)**2) / (2 * 0.03**2))

plt.figure(figsize=(10, 5))
plt.plot(x, c, 'k--', label='$t=0$')

for n in range(Nt):
    c_new = c.copy()
    c_new[1:-1] = (c[1:-1]
                   - v * dt / dx * (c[1:-1] - c[:-2])
                   + D * dt / dx**2 * (c[:-2] - 2*c[1:-1] + c[2:]))
    c_new[0] = 0; c_new[-1] = 0
    c = c_new
    if (n+1) in [200, 500, 800, 1000]:
        plt.plot(x, c, label=f'$t={(n+1)*dt:.2f}$')

plt.xlabel('$x$'); plt.ylabel('$c(x,t)$')
plt.legend(); plt.title(f'Advection-diffusion (Pe = {v*L/D:.0f})')
plt.tight_layout(); plt.savefig('advection_diffusion.pdf')
```

6.6 Diffusion in Higher Dimensions

Definition 6.8 (Diffusion equation in 2D/3D)**.** In d spatial dimensions:

$$\frac{\partial c}{\partial t} = D\nabla^2 c = D\sum_{i=1}^{d}\frac{\partial^2 c}{\partial x_i^2}.$$

The fundamental solution in d dimensions is:

$$c(\mathbf{x}, t) = \frac{1}{(4\pi Dt)^{d/2}}\exp\left(-\frac{\|\mathbf{x}\|^2}{4Dt}\right).$$

6.7 Reaction-Diffusion Models

Definition 6.9 (Reaction-diffusion equation)**.** Adding a reaction term $f(c)$:

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} + f(c).$$

Example 6.10 (Fisher–KPP equation)**.** With $f(c) = rc(1-c)$ (logistic growth), we obtain the Fisher equation:

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} + rc(1-c).$$

It admits **travelling wave fronts** with minimum speed $v_{\min} = 2\sqrt{Dr}$.

Theorem 6.11 (Minimum front speed for Fisher–KPP)**.** *The Fisher–KPP equation admits travelling wave solutions $c(x,t) = \phi(x - vt)$ for all speeds $v \geq v_{\min} = 2\sqrt{Dr}$. For compactly supported initial data, the front propagates at speed $v_{\min}$.*

Fisher wave front

```python
import numpy as np
import matplotlib.pyplot as plt

L, D, r = 20.0, 1.0, 1.0
Nx = 400; dx = L / Nx
dt = 0.4 * dx**2 / D
x = np.linspace(0, L, Nx + 1)

c = np.where(x < 2, 1.0, 0.0)

plt.figure(figsize=(10, 5))
times_to_plot = [0, 2, 4, 6, 8]
t_current = 0

for t_target in times_to_plot:
    while t_current < t_target:
        c_new = c.copy()
        c_new[1:-1] = (c[1:-1]
            + dt * D * (c[:-2] - 2*c[1:-1] + c[2:]) / dx**2
            + dt * r * c[1:-1] * (1 - c[1:-1]))
        c_new[0] = 1.0; c_new[-1] = 0.0
        c = c_new
        t_current += dt
    plt.plot(x, c, label=f'$t = {t_target}$')

v_min = 2 * np.sqrt(D * r)
plt.xlabel('$x$'); plt.ylabel('$c(x,t)$')
plt.title(f'Fisher Wave Front ($v_{{\\min}} = {v_min:.2f}$)')
plt.legend(); plt.tight_layout()
plt.savefig('fisher_front.pdf')
```

6.8 Turing Patterns

Definition 6.12 (Turing instability)**.** A two-species reaction-diffusion system:

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u,v), \tag{6.1}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + g(u,v), \tag{6.2}$$

may exhibit a **Turing instability**: a spatially homogeneous equilibrium that is stable without diffusion can become unstable with diffusion if $D_v \gg D_u$. This gives rise to **spatial patterns** (spots, stripes).

Remark 6.13. Turing instability typically requires an activator (slow diffusion, small D_u) and an inhibitor (fast diffusion, large D_v).

6.9 Exercises

Exercise 6.1. Verify that the heat kernel $c(x,t) = \frac{1}{\sqrt{4\pi Dt}}\exp(-x^2/(4Dt))$ satisfies the diffusion equation.

Exercise 6.2. Solve the heat equation on $[0, L]$ with homogeneous Dirichlet boundary conditions by separation of variables. Show that the general solution is:

$$c(x,t) = \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L}\right) \exp\left(-\frac{n^2\pi^2 D}{L^2}t\right).$$

Exercise 6.3. Implement the Crank–Nicolson scheme for 1D diffusion and compare with the explicit scheme in terms of accuracy and stability.

Exercise 6.4. Simulate the advection-diffusion equation for different values of Pe and observe the transition between the diffusive and advective regimes.

Exercise 6.5. For the Fisher–KPP equation, verify numerically that the front speed converges to $v_{\min} = 2\sqrt{Dr}$ for compactly supported initial data.

Exercise 6.6. Implement a Schnakenberg-type reaction-diffusion system in 2D and observe the formation of Turing patterns.

Chapter 7

Economic and Financial Models

7.1 Introduction

Economists have long envied physics for the precision of its models. In 1956, Robert Solow proposed an economic growth model of Newtonian elegance: capital accumulates, labour grows, and technology progresses, all governed by a differential equation whose steady state illuminates the wealth disparities between nations. This model earned him the Nobel Prize in 1987. But economics is also the domain of uncertainty: financial markets oscillate, option prices defy intuition. It is there that the Black–Scholes model, rooted in Itô's stochastic calculus, revolutionized finance in 1973. This chapter weaves the link between deterministic and stochastic differential equations applied to economics and finance, from the Solow model to option pricing.

7.2 The Solow Growth Model

Definition 7.1 (Solow model)**.** The Solow model describes long-run economic growth. Let $K(t)$ be capital, $L(t)$ labour, and $Y(t)$ output. The production function is Cobb–Douglas:

$$Y = AK^{\alpha}L^{1-\alpha}, \qquad 0 < \alpha < 1,$$

where A is total factor productivity (TFP). Capital accumulates according to:

$$\dot{K} = sY - \delta K,$$

where $s \in (0,1)$ is the savings rate and $\delta > 0$ the depreciation rate.

7.2.1 Per-Capita Capital

Assume $L(t) = L_0 e^{nt}$ (population growth at rate n). Setting $k = K/L$ (capital per worker) and $y = Y/L$:

Theorem 7.2 (Fundamental Solow equation)**.** *Capital per worker satisfies:*

$$\dot{k} = sAk^{\alpha} - (n+\delta)k.$$

Proof. $\dot{k} = \dot{K}/L - Kn/L = (sY - \delta K)/L - nk = sy - \delta k - nk = sAk^{\alpha} - (n+\delta)k.$ □

Proposition 7.3 (Solow equilibrium)**.** The positive equilibrium k^* satisfies $sAk^{*\alpha} = (n+\delta)k^*$:

$$k^* = \left(\frac{sA}{n+\delta}\right)^{1/(1-\alpha)}.$$

This equilibrium is **globally stable** for $k > 0$.

Solow equilibrium

$$k^* = \left(\frac{sA}{n+\delta}\right)^{1/(1-\alpha)}, \qquad y^* = A\left(\frac{sA}{n+\delta}\right)^{\alpha/(1-\alpha)}.$$

Solow model simulation

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

A, alpha = 1.0, 0.3
s, delta, n = 0.2, 0.05, 0.02

def solow(t, k):
    return s * A * k**alpha - (n + delta) * k

k_star = (s * A / (n + delta))**(1 / (1 - alpha))
print(f"Equilibrium: k* = {k_star:.4f}")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 5))

for k0 in [0.5, 1, 2, 5, 10, 15]:
    sol = solve_ivp(solow, [0, 100], [k0],
                    t_eval=np.linspace(0, 100, 500))
    ax1.plot(sol.t, sol.y[0], label=f'$k_0={k0}$')

ax1.axhline(k_star, color='gray', ls='--', label=f'$k^*={k_star:.2f}$')
ax1.set_xlabel('$t$'); ax1.set_ylabel('$k(t)$')
ax1.legend(fontsize=8); ax1.set_title('Convergence to $k^*$')

k_vals = np.linspace(0, 15, 200)
ax2.plot(k_vals, s * A * k_vals**alpha, 'b-', label='$sAk^\\alpha$')
ax2.plot(k_vals, (n + delta) * k_vals, 'r-', label='$(n+\\delta)k$')
ax2.plot(k_star, (n+delta)*k_star, 'ko', markersize=8)
ax2.set_xlabel('$k$'); ax2.set_ylabel('Investment / Depreciation')
ax2.legend(); ax2.set_title('Solow Diagram')
plt.tight_layout(); plt.savefig('solow.pdf')
```

7.3 Supply-Demand Equilibrium

Definition 7.4 (Price adjustment model)**.** Let $D(p)$ be demand and $S(p)$ supply as functions of price p. The price adjusts proportionally to excess demand:

$$\dot{p} = \kappa\big(D(p) - S(p)\big), \qquad \kappa > 0.$$

Example 7.5 (Linear functions)**.** With $D(p) = a - bp$ and $S(p) = c + dp$ $(a, b, c, d > 0)$:

$$\dot{p} = \kappa\big((a - c) - (b + d)p\big).$$

The equilibrium $p^* = (a - c)/(b + d)$ is globally stable.

7.3.1 Cobweb Model

Definition 7.6 (Cobweb model)**.** In discrete time, producers set supply based on past price:

$$S_t = S(p_{t-1}), \qquad p_t = D^{-1}(S_t).$$

Proposition 7.7. If $|S'(p^*)/D'(p^*)| < 1$, the model converges to equilibrium. If $|S'(p^*)/D'(p^*)| > 1$, oscillations diverge.

7.4 The Black–Scholes Model

Definition 7.8 (Geometric Brownian motion)**.** The price of a financial asset $S(t)$ follows geometric Brownian motion:

$$dS = \mu S\, dt + \sigma S\, dW_t,$$

where μ is the drift, σ the volatility, and W_t a standard Brownian motion.

Theorem 7.9 (Black–Scholes equation)**.** *The price $V(S, t)$ of a European option on an asset of price S satisfies the PDE:*

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0,$$

where r is the risk-free rate.

Black–Scholes formula for a European call

$$C(S, t) = S\,\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2),$$

where K is the strike price, T the maturity, Φ the standard normal CDF, and

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}}, \qquad d_2 = d_1 - \sigma\sqrt{T - t}.$$

Black–Scholes formula in Python

```
import numpy as np
from scipy.stats import norm
```

```python
import matplotlib.pyplot as plt

def black_scholes_call(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * norm.cdf(d1) - K * np.exp(-r*T) * norm.cdf(d2)

def black_scholes_put(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return K * np.exp(-r*T) * norm.cdf(-d2) - S * norm.cdf(-d1)

S0, K, T, r, sigma = 100, 100, 1.0, 0.05, 0.2
print(f"Call = {black_scholes_call(S0, K, T, r, sigma):.4f}")
print(f"Put  = {black_scholes_put(S0, K, T, r, sigma):.4f}")

S_range = np.linspace(60, 140, 200)
calls = [black_scholes_call(S, K, T, r, sigma) for S in S_range]

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(S_range, calls, 'b-', lw=2, label='Call price')
ax.plot(S_range, np.maximum(S_range - K, 0), 'r--', label='Payoff')
ax.set_xlabel('$S$'); ax.set_ylabel('Call price')
ax.legend(); ax.set_title('European Call (Black-Scholes)')
plt.tight_layout(); plt.savefig('black_scholes.pdf')
```

7.5 Monte Carlo Simulation

Definition 7.10 (Monte Carlo pricing). Simulate M price paths $S(T)$ and estimate the option price by the discounted average payoff:

$$\hat{C} = e^{-rT}\frac{1}{M}\sum_{i=1}^{M}\max(S_i(T) - K,\ 0).$$

Monte Carlo for European call

```python
import numpy as np

S0, K, T, r, sigma = 100, 100, 1.0, 0.05, 0.2
M = 100000
np.random.seed(42)

Z = np.random.standard_normal(M)
S_T = S0 * np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*Z)
payoffs = np.maximum(S_T - K, 0)
C_mc = np.exp(-r*T) * np.mean(payoffs)
se = np.exp(-r*T) * np.std(payoffs) / np.sqrt(M)

print(f"MC price = {C_mc:.4f} +/- {1.96*se:.4f} (95% CI)")
print(f"BS price = {black_scholes_call(S0, K, T, r, sigma):.4f}")
```

7.6 Markowitz Portfolio Optimisation

Definition 7.11 (Portfolio optimisation)**.** For n assets with expected returns $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix Σ, the Markowitz optimal portfolio minimises variance for a target return r_c:

$$\min_{\mathbf{w}} \mathbf{w}^\top \Sigma \mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^\top \boldsymbol{\mu} = r_c, \quad \mathbf{w}^\top \mathbf{1} = 1.$$

Proposition 7.12 (Efficient frontier)**.** The set of optimal portfolios forms a parabola in the (risk, return) plane, called the **efficient frontier**.

7.7 Leontief Input-Output Model

Definition 7.13 (Leontief model)**.** Let $A = (a_{ij})$ be the matrix giving the amount of good i needed to produce one unit of good j. Production equilibrium satisfies:

$$\mathbf{x} = A\mathbf{x} + \mathbf{d} \quad \Leftrightarrow \quad \mathbf{x} = (I - A)^{-1}\mathbf{d},$$

where $\mathbf{d}$ is final demand.

Proposition 7.14. If the spectral radius satisfies $\rho(A) < 1$, then $(I - A)$ is invertible and $(I - A)^{-1} = \sum_{k=0}^{\infty} A^k \geq 0$.

7.8 Limitations of Financial Models

Limitations of Black–Scholes

- Volatility σ is not constant (volatility smile).
- Real returns have heavy tails (extreme events more frequent than predicted by the normal distribution).
- Markets are not always liquid and frictionless.
- Financial crises reveal nonlinear correlations that Gaussian models ignore.

7.9 Exercises

Exercise 7.1. For the Solow model with $A = 1$, $\alpha = 0.3$, $s = 0.25$, $\delta = 0.05$, $n = 0.01$:

1. Compute k^* and y^*.
2. Study the effect of increasing the savings rate s.
3. Find the optimal savings rate (Phelps' golden rule).

Exercise 7.2. Simulate the cobweb model with $D(p) = 100 - 2p$ and $S(p) = -10 + 3p$. Determine whether oscillations converge or diverge.

Exercise 7.3. Compute the "Greeks" of the European call (Delta, Gamma, Vega, Theta, Rho) both analytically and numerically.

Exercise 7.4. Compare the Monte Carlo price of a European put with the Black–Scholes formula for different numbers of paths M. Plot the error as a function of M and verify $O(1/\sqrt{M})$ convergence.

Exercise 7.5. For a portfolio of 3 assets with given returns and covariances, plot the Markowitz efficient frontier.

Chapter 8

Optimization and Variational Models

8.1 Introduction

Nature is economical. Light follows the fastest path (Fermat's principle), a soap bubble minimizes its surface area (Plateau's problem), a hanging chain adopts the shape that minimizes its potential energy. This idea — that the laws of physics are optimization problems — has fascinated mathematicians since Euler and Lagrange in the eighteenth century. The *calculus of variations*, which seeks the function optimizing a functional, is the tool that follows from it. But optimization also permeates the applied sciences: linear programming for logistics, Karush–Kuhn–Tucker conditions for constrained optimization, numerical methods for large-scale problems. This chapter covers these three facets, from classical theory to modern algorithms.

8.2 Linear Programming

Definition 8.1 (Linear programming problem)**.** A **linear programming** (LP) problem consists of minimising a linear objective under linear constraints:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad A\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}.$$

Theorem 8.2 (Fundamental theorem of LP)**.** *If an LP problem has an optimal solution, then it has one at a **vertex** (extreme point) of the constraint polyhedron.*

Example 8.3 (Production problem)**.** A factory produces two goods x_1, x_2 with resource constraints and unit profits:

$$\begin{aligned} \max \quad & 40x_1 + 30x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 40, \\ & 2x_1 + x_2 \leq 60, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Linear programming with scipy

```python
import numpy as np
from scipy.optimize import linprog
```

```
c = [-40, -30]  # min -40x1 - 30x2  <=>  max 40x1 + 30x2
A_ub = [[1, 1], [2, 1]]
b_ub = [40, 60]
bounds = [(0, None), (0, None)]

result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds,
                 method='highs')
print(f"Optimal solution: x1 = {result.x[0]:.1f}, "
      f"x2 = {result.x[1]:.1f}")
print(f"Maximum profit: {-result.fun:.1f}")
```

Definition 8.4 (Duality)**.** The **dual problem** associated with the primal LP $\min \mathbf{c}^\top \mathbf{x}$ s.t. $A\mathbf{x} \geq \mathbf{b}$, $\mathbf{x} \geq 0$ is:

$$\max_{\mathbf{y}\geq 0} \mathbf{b}^\top \mathbf{y} \quad \text{s.t.} \quad A^\top \mathbf{y} \leq \mathbf{c}.$$

Theorem 8.5 (Strong duality theorem)**.** *If the primal and dual have feasible solutions, then their optimal values are equal:* $\mathbf{c}^\top \mathbf{x}^* = \mathbf{b}^\top \mathbf{y}^*$.

8.3 Unconstrained Nonlinear Optimisation

Definition 8.6 (Necessary optimality conditions)**.** For $f : \mathbb{R}^n \to \mathbb{R}$ of class $\mathcal{C}^2$, if $\mathbf{x}^*$ is a local minimum, then:

1. $\nabla f(\mathbf{x}^*) = \mathbf{0}$ (first-order condition).
2. $\nabla^2 f(\mathbf{x}^*)$ is positive semi-definite (second-order condition).

Definition 8.7 (Gradient descent)**.** Gradient descent iterates:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k),$$

where $\alpha_k > 0$ is the step size (learning rate).

Gradient descent in Python

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return (x[0]-1)**2 + 10*(x[1]-x[0]**2)**2  # Rosenbrock

def grad_f(x):
    dx0 = 2*(x[0]-1) - 40*x[0]*(x[1]-x[0]**2)
    dx1 = 20*(x[1]-x[0]**2)
    return np.array([dx0, dx1])

x = np.array([-1.0, 1.0])
alpha = 0.002
trajectory = [x.copy()]
```

```
for _ in range(5000):
    x = x - alpha * grad_f(x)
    trajectory.append(x.copy())

trajectory = np.array(trajectory)
print(f"Minimum found: x = ({x[0]:.6f}, {x[1]:.6f})")
print(f"f(x) = {f(x):.8f}")

x1 = np.linspace(-2, 2, 200)
x2 = np.linspace(-1, 3, 200)
X1, X2 = np.meshgrid(x1, x2)
Z = (X1-1)**2 + 10*(X2-X1**2)**2

plt.figure(figsize=(8, 6))
plt.contour(X1, X2, Z, levels=np.logspace(-1, 3, 30), cmap='viridis')
plt.plot(trajectory[:, 0], trajectory[:, 1], 'r-', lw=0.5)
plt.plot(1, 1, 'r*', markersize=15)
plt.xlabel('$x_1$'); plt.ylabel('$x_2$')
plt.title('Gradient descent on the Rosenbrock function')
plt.tight_layout(); plt.savefig('gradient_descent.pdf')
```

8.4 Constrained Optimisation: KKT Conditions

Theorem 8.8 (Karush–Kuhn–Tucker conditions)**.** *Consider the problem:*

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad s.t. \quad g_i(\mathbf{x}) \le 0, \; i = 1, \ldots, m, \qquad h_j(\mathbf{x}) = 0, \; j = 1, \ldots, p.$$

If $\mathbf{x}^*$ is a local minimum and a constraint qualification holds, then there exist multipliers $\lambda_i \ge 0$ and μ_j such that:

1. $\nabla f(\mathbf{x}^*) + \sum_i \lambda_i \nabla g_i(\mathbf{x}^*) + \sum_j \mu_j \nabla h_j(\mathbf{x}^*) = \mathbf{0}$ *(stationarity).*
2. $g_i(\mathbf{x}^*) \le 0$ *for all i (primal feasibility).*
3. $\lambda_i \ge 0$ *for all i (dual feasibility).*
4. $\lambda_i g_i(\mathbf{x}^*) = 0$ *for all i (complementary slackness).*

Example 8.9. Minimise $f(x, y) = x^2 + y^2$ subject to $x + y \ge 1$. The Lagrangian is $\mathcal{L} = x^2 + y^2 - \lambda(x + y - 1)$. KKT conditions: $2x = \lambda$, $2y = \lambda$, $\lambda(x + y - 1) = 0$, $\lambda \ge 0$. Thus $x = y = \lambda/2$. If $\lambda > 0$: $x + y = 1$, so $\lambda = 1$, $x^* = y^* = 1/2$.

Constrained optimisation with scipy

```
from scipy.optimize import minimize

def objective(x):
    return x[0]**2 + x[1]**2

constraints = [{'type': 'ineq', 'fun': lambda x: x[0] + x[1] - 1}]
```

```
x0 = [0.0, 0.0]
result = minimize(objective, x0, method='SLSQP',
                  constraints=constraints)
print(f"Solution: x = ({result.x[0]:.4f}, {result.x[1]:.4f})")
print(f"f(x*) = {result.fun:.4f}")
```

8.5 Calculus of Variations

Definition 8.10 (Variational problem)**.** Find the function $y : [a, b] \to \mathbb{R}$ making stationary the functional:

$$J[y] = \int_a^b F(x, y, y')\, dx,$$

with boundary conditions $y(a) = y_a$, $y(b) = y_b$.

Theorem 8.11 (Euler–Lagrange equation)**.** *If y^* makes J stationary, then y^* satisfies:*

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\frac{\partial F}{\partial y'} = 0.$$

Proof. Let η be an admissible variation ($\eta(a) = \eta(b) = 0$). Set $y_\varepsilon = y^* + \varepsilon\eta$. The stationarity condition is:

$$\frac{d}{d\varepsilon} J[y_\varepsilon]\bigg|_{\varepsilon=0} = 0.$$

Computing:

$$\int_a^b \left(\frac{\partial F}{\partial y}\eta + \frac{\partial F}{\partial y'}\eta' \right) dx = 0.$$

Integrating the second term by parts (boundary terms vanish) and applying the fundamental lemma of the calculus of variations yields the Euler–Lagrange equation. □

Example 8.12 (Geodesics in the plane)**.** The length of a curve $y(x)$ between (a, y_a) and (b, y_b) is $J[y] = \int_a^b \sqrt{1 + y'^2}\, dx$. Here $F = \sqrt{1 + y'^2}$, $\partial F/\partial y = 0$. The Euler–Lagrange equation gives $y''/(1 + y'^2)^{3/2} = 0$, i.e. $y'' = 0$: geodesics are straight lines.

Example 8.13 (Brachistochrone)**.** Find the curve $y(x)$ along which a bead slides without friction from $(0, 0)$ to (x_1, y_1) in minimum time. The descent time is:

$$T[y] = \int_0^{x_1} \frac{\sqrt{1 + y'^2}}{\sqrt{2gy}}\, dx.$$

The Euler–Lagrange equation leads to a **cycloid**.

Numerical brachistochrone

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
```

```python
N = 30
x1, y1 = 1.0, 0.5
x = np.linspace(0, x1, N + 2)
dx = x[1] - x[0]
g = 9.81

def travel_time(y_inner):
    y = np.concatenate([[0], y_inner, [y1]])
    y_mid = 0.5 * (y[:-1] + y[1:])
    y_mid = np.maximum(y_mid, 1e-10)
    ds = np.sqrt(dx**2 + (np.diff(y))**2)
    v = np.sqrt(2 * g * y_mid)
    return np.sum(ds / v)

y0_inner = np.linspace(0, y1, N + 2)[1:-1]
result = minimize(travel_time, y0_inner, method='L-BFGS-B',
                  bounds=[(0.001, 2.0)] * N)
y_opt = np.concatenate([[0], result.x, [y1]])

plt.figure(figsize=(8, 5))
plt.plot(x, y_opt, 'b-', lw=2, label='Numerical brachistochrone')
plt.plot(x, np.linspace(0, y1, N+2), 'r--', label='Straight line')
plt.gca().invert_yaxis()
plt.xlabel('$x$'); plt.ylabel('$y$')
plt.legend(); plt.title('Brachistochrone Curve')
plt.tight_layout(); plt.savefig('brachistochrone.pdf')
```

8.6 Isoperimetric Problems

Theorem 8.14 (Isoperimetric problem). *Among all closed curves of fixed perimeter L, the one enclosing maximum area is the* ***circle****.*

Definition 8.15 (Lagrange multiplier for functionals). For a variational problem with integral constraint $\int_a^b G(x, y, y')\,dx = C$, form the augmented Lagrangian:

$$\tilde{J}[y] = \int_a^b \left[F(x, y, y') + \lambda G(x, y, y')\right] dx,$$

and apply the Euler–Lagrange equation to $\tilde{F} = F + \lambda G$.

8.7 Exercises

Exercise 8.1. Solve graphically and with `linprog`:

$$\begin{aligned} \max \quad & 5x_1 + 4x_2 \\ \text{s.t.} \quad & 6x_1 + 4x_2 \le 24,\ x_1 + 2x_2 \le 6,\ x_1, x_2 \ge 0. \end{aligned}$$

Exercise 8.2. Show that gradient descent converges for a strongly convex, L-smooth function f with step size $\alpha = 1/L$. What is the convergence rate?

CHAPTER 8. OPTIMIZATION AND VARIATIONAL MODELS

Exercise 8.3. Solve via KKT conditions: min $x_1^2 + x_2^2$ s.t. $x_1 + 2x_2 = 3$, $x_1 \geq 0$.

Exercise 8.4. Find the catenary equation (shape of a hanging cable) by minimising gravitational potential energy $J[y] = \int_{-a}^{a} \rho g y \sqrt{1 + y'^2}\, dx$ subject to fixed length $\int \sqrt{1 + y'^2}\, dx = L$.

Exercise 8.5. Implement Newton's method for unconstrained optimisation and compare its convergence with gradient descent on the Rosenbrock function.

Exercise 8.6. Solve the brachistochrone problem numerically for different endpoint positions and compare with the analytical solution (cycloid).

Chapter 9

Stochastic Models

"Randomness is not disorder; it is a different kind of order."
— Nassim Nicholas Taleb

Until now, our models have been deterministic: knowing the initial state and the evolution laws, one could predict the future with certainty. But the real world is noisy. Financial markets fluctuate, genes mutate randomly, molecules jitter under thermal agitation, and even epidemics do not propagate deterministically. Modelling these phenomena requires integrating randomness into the equations. This chapter introduces stochastic models — from random walks to Itô stochastic differential equations — and shows how randomness, far from being an obstacle to understanding, becomes an essential ingredient of modelling. We cover birth-death processes in biology, queueing theory, random walks, the Gillespie algorithm, and Monte Carlo simulation.

9.1 Birth-Death Processes

Definition 9.1 (Birth-Death Process)**.** A **birth-death process** is a continuous-time Markov chain $\{X(t)\}_{t\geq 0}$ taking values in $\mathbb{N}$ with transitions from state n:

- $n \to n+1$ (birth) at rate $\lambda_n \geq 0$,
- $n \to n-1$ (death) at rate $\mu_n \geq 0$, where $\mu_0 = 0$.

Intuition

Think of a bacterial colony in a petri dish. Each bacterium divides (birth) or dies independently. The total count fluctuates randomly around a general trend. The birth-death process captures exactly this stochastic dynamics at the individual level.

Proposition 9.2 (Kolmogorov Forward Equations)**.** Let $p_n(t) = \mathbb{P}(X(t) = n)$. The state probabilities satisfy:

$$\frac{dp_n}{dt} = \lambda_{n-1}\, p_{n-1}(t) + \mu_{n+1}\, p_{n+1}(t) - (\lambda_n + \mu_n)\, p_n(t), \quad n \geq 1,$$

with $\dfrac{dp_0}{dt} = \mu_1\, p_1(t) - \lambda_0\, p_0(t)$.

Proof. Condition on the state at time t. In a small interval $[t, t+h]$:

$$p_n(t+h) = p_n(t)\big(1 - (\lambda_n + \mu_n)h\big) + p_{n-1}(t)\,\lambda_{n-1}\,h + p_{n+1}(t)\,\mu_{n+1}\,h + o(h).$$

Subtracting $p_n(t)$, dividing by h, and letting $h \to 0$ yields the stated equation. □

Theorem 9.3 (Stationary Distribution)**.** *If $\sum_{n=1}^{\infty} \prod_{k=0}^{n-1} \frac{\lambda_k}{\mu_{k+1}} < \infty$, then a stationary distribution $\pi = (\pi_n)_{n \geq 0}$ exists:*

$$\pi_n = \pi_0 \prod_{k=0}^{n-1} \frac{\lambda_k}{\mu_{k+1}}, \qquad \pi_0 = \left(1 + \sum_{n=1}^{\infty} \prod_{k=0}^{n-1} \frac{\lambda_k}{\mu_{k+1}}\right)^{-1}.$$

Example 9.4. Consider the Yule process with $\lambda_n = \lambda\, n$ and $\mu_n = 0$ (pure birth). Starting from $X(0) = 1$, the expected population grows exponentially: $\mathbb{E}[X(t)] = e^{\lambda t}$.

9.2 Queueing Theory

Definition 9.5 (M/M/1 Queue)**.** An **M/M/1 queue** is a birth-death process with constant arrival rate $\lambda_n = \lambda$ and constant service rate $\mu_n = \mu$ for $n \geq 1$. The letter "M" stands for "Markovian" (exponential inter-arrival and service times).

Theorem 9.6 (M/M/1 Equilibrium)**.** *Let $\rho = \lambda/\mu$ denote the traffic intensity. If $\rho < 1$, the M/M/1 queue has a geometric stationary distribution:*

$$\pi_n = (1 - \rho)\,\rho^n, \quad n \geq 0.$$

The expected number of customers is $L = \rho/(1-\rho)$, and the mean sojourn time is $W = 1/(\mu - \lambda)$.

Proof. By Theorem 9.3, $\pi_n = \pi_0\, \rho^n$. Normalization $\sum_{n \geq 0} \pi_n = 1$ gives $\pi_0 = 1 - \rho$ when $\rho < 1$. The mean number follows from $L = (1-\rho) \sum_{n \geq 1} n\, \rho^n = \rho/(1-\rho)$. Little's law $L = \lambda W$ yields W. □

Key Formulas

Queueing Summary

$$\text{M/M/1:} \quad L = \frac{\rho}{1-\rho}, \quad W = \frac{1}{\mu - \lambda}, \quad L_q = \frac{\rho^2}{1-\rho}$$

$$\text{M/M/c:} \quad \pi_0 = \left[\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} + \frac{(c\rho)^c}{c!\,(1-\rho)}\right]^{-1}, \quad \rho = \frac{\lambda}{c\mu} < 1$$

Definition 9.7 (M/M/c Queue)**.** An **M/M/c queue** has c identical servers in parallel. The effective service rate in state n is $\mu_n = \min(n, c)\,\mu$.

Remark 9.8. The Erlang-C formula gives the probability of waiting in an M/M/c queue: $C(c, \lambda/\mu) = \frac{(c\rho)^c}{c!\,(1-\rho)}\,\pi_0$. It is widely used in call center dimensioning.

9.3 Random Walks

Definition 9.9 (Simple Random Walk)**.** A **simple random walk** on $\mathbb{Z}$ is a sequence $(S_n)_{n\geq 0}$ with $S_0 = 0$ and $S_n = \sum_{k=1}^{n} X_k$, where the (X_k) are i.i.d. with $\mathbb{P}(X_k = +1) = p$ and $\mathbb{P}(X_k = -1) = 1 - p$.

Theorem 9.10 (Recurrence of Random Walks)**.** *The simple random walk on $\mathbb{Z}$ is recurrent if and only if $p = 1/2$. More generally, the simple symmetric random walk on $\mathbb{Z}^d$ is recurrent if and only if $d \leq 2$ (Pólya's theorem).*

Proof sketch for $d = 1$. When $p = 1/2$, the return probability involves $\sum_{n\geq 1} \mathbb{P}(S_{2n} = 0) = \sum_{n\geq 1} \binom{2n}{n} 2^{-2n}$. By Stirling's formula, $\binom{2n}{n} \sim 4^n/\sqrt{\pi n}$, so the series diverges, ensuring recurrence. For $p \neq 1/2$, $\mathbb{E}[X_k] \neq 0$ and the law of large numbers implies $S_n \to \pm\infty$. □

Example 9.11. The random walk models a particle suspended in a fluid (discrete Brownian motion) and the fortune of a gambler in a fair game.

9.4 The Gillespie Algorithm

Definition 9.12 (Gillespie Algorithm (SSA))**.** The **Stochastic Simulation Algorithm** (SSA) of Gillespie exactly simulates the trajectory of a stochastic chemical system. At each step:

1. the time τ until the next reaction is drawn from $\text{Exp}(a_0)$, where $a_0 = \sum_j a_j$,
2. reaction j is selected with probability a_j/a_0,

where a_j is the *propensity* of reaction j.

Attention

The Gillespie algorithm simulates *every single* reaction event. For systems with large molecule counts, it becomes computationally prohibitive. Approximate methods such as τ-leaping are preferred in such cases.

Example 9.13. Consider mRNA production-degradation: $\emptyset \xrightarrow{k_1} \text{mRNA} \xrightarrow{k_2} \emptyset$, with propensities $a_1 = k_1$ and $a_2 = k_2 n$. At stochastic equilibrium, n follows a Poisson distribution with parameter k_1/k_2.

```python
import numpy as np

def gillespie_production_degradation(k1, k2, n0, t_max):
    """Gillespie SSA for production-degradation."""
    t, n = 0.0, n0
    times, states = [t], [n]
    while t < t_max:
        a1 = k1            # production propensity
        a2 = k2 * n        # degradation propensity
        a0 = a1 + a2
        if a0 == 0:
            break
        tau = np.random.exponential(1.0 / a0)
        t += tau
        if np.random.uniform() < a1 / a0:
            n += 1          # production
        else:
            n -= 1          # degradation
        times.append(t)
        states.append(n)
    return np.array(times), np.array(states)
```

9.5 Monte Carlo Simulation

Definition 9.14 (Monte Carlo Method)**.** A **Monte Carlo method** uses random samples to estimate a deterministic quantity. To estimate $\theta = \mathbb{E}[g(X)]$, generate $X_1, \ldots, X_N$ i.i.d. from the distribution of X and compute:

$$\hat{\theta}_N = \frac{1}{N}\sum_{i=1}^{N} g(X_i).$$

Theorem 9.15 (Monte Carlo Convergence)**.** *By the strong law of large numbers, $\hat{\theta}_N \to \theta$ a.s. Moreover, by the CLT, the estimation error is of order $O(1/\sqrt{N})$:*

$$\sqrt{N}\,(\hat{\theta}_N - \theta) \xrightarrow{d} \mathcal{N}(0, \mathrm{Var}(g(X))).$$

Intuition

The $O(1/\sqrt{N})$ convergence rate is *dimension-independent*. This is the major advantage of Monte Carlo over deterministic methods (quadrature), whose convergence degrades with dimension—the so-called "curse of dimensionality."

Proposition 9.16 (Antithetic Variates)**.** If g is monotone and $X \sim \mathcal{U}(0,1)$, the estimator

$$\hat{\theta}_N^{\text{anti}} = \frac{1}{N}\sum_{i=1}^{N/2} \frac{g(U_i) + g(1-U_i)}{2}$$

has variance less than or equal to that of the crude estimator.

Exercise 9.1. Estimate $\int_0^1 e^x\,dx$ by Monte Carlo with $N = 10{,}000$ samples. Compare the accuracy with and without antithetic variates.

Exercise 9.2. Simulate an M/M/1 queue with $\lambda = 3$ and $\mu = 5$ over a time horizon $T = 1000$. Estimate the average number of customers in the system and compare with the theoretical value $L = 3/2$.

Exercise 9.3. Implement the Gillespie algorithm for the stochastic Lotka–Volterra model: $X \xrightarrow{\alpha} 2X$, $X + Y \xrightarrow{\beta} 2Y$, $Y \xrightarrow{\gamma} \emptyset$. Compare stochastic trajectories with the deterministic model.

Chapter 10

Case Studies and Projects

"All models are wrong, but some are useful."

— George E. P. Box

This chapter applies the tools developed in earlier chapters to five concrete case studies. Each study illustrates the full modeling cycle: formulation, mathematical analysis, calibration with data, and validation.

10.1 Epidemiological Modeling: SIR and SEIR

Definition 10.1 (SIR Model)**.** The **SIR** (Susceptible–Infected–Recovered) model of Kermack–McKendrick describes disease spread in a population of size N:

$$\frac{dS}{dt} = -\beta\,\frac{SI}{N}, \quad \frac{dI}{dt} = \beta\,\frac{SI}{N} - \gamma\, I, \quad \frac{dR}{dt} = \gamma\, I,$$

where $\beta > 0$ is the transmission rate and $\gamma > 0$ the recovery rate.

Theorem 10.2 (Epidemic Threshold)**.** *Let $R_0 = \beta/\gamma$ be the* basic reproduction number*. If $R_0 > 1$ and $S(0) \approx N$, then $I(t)$ initially increases (epidemic outbreak). If $R_0 < 1$, the number of infected individuals decreases monotonically.*

Proof. At the onset of the epidemic, $S \approx N$, so $dI/dt \approx (\beta - \gamma)\, I = \gamma(R_0 - 1)\, I$. If $R_0 > 1$, this linearized equation yields exponential growth of I. If $R_0 < 1$, I decays exponentially. □

Intuition

The number R_0 represents the average number of secondary infections caused by a single infected individual during their entire infectious period (of mean duration $1/\gamma$). The epidemic spreads only if each patient infects more than one person on average ($R_0 > 1$).

Definition 10.3 (SEIR Model)**.** The **SEIR** model adds an "Exposed" (incubation) compartment:

$$\frac{dS}{dt} = -\beta\,\frac{SI}{N}, \quad \frac{dE}{dt} = \beta\,\frac{SI}{N} - \sigma\, E, \quad \frac{dI}{dt} = \sigma\, E - \gamma\, I, \quad \frac{dR}{dt} = \gamma\, I,$$

where $1/\sigma$ is the mean incubation period.

Key Formulas

Key Epidemiological Parameters

$$R_0 = \frac{\beta}{\gamma} \quad \text{(SIR and SEIR)}, \qquad R_{\text{eff}}(t) = R_0 \cdot \frac{S(t)}{N}$$

$$\text{Final size:} \quad S_\infty \text{ solves } S_\infty = S_0\, e^{-R_0(N-S_\infty)/N}$$

Example 10.4. Calibrating the SIR model on influenza data from an English boarding school (1978): 763 students, daily data over 14 days. Least-squares fitting gives $\beta \approx 1.66$ and $\gamma \approx 0.44$, yielding $R_0 \approx 3.78$.

```python
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import minimize

def sir_model(y, t, beta, gamma, N):
    S, I, R = y
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return [dSdt, dIdt, dRdt]

# Observed infected counts (daily)
data_I = np.array([1, 3, 8, 28, 76, 222, 293, 257, 237,
                   192, 126, 70, 28, 12])
t_data = np.arange(len(data_I))
N = 763

def objective(params):
    beta, gamma = params
    y0 = [N - 1, 1, 0]
    sol = odeint(sir_model, y0, t_data, args=(beta, gamma, N))
    return np.sum((sol[:, 1] - data_I)**2)

result = minimize(objective, [1.5, 0.5], method='Nelder-Mead')
```

10.2 Population Dynamics: Lotka–Volterra

Definition 10.5 (Lotka–Volterra Model)**.** The predator–prey Lotka–Volterra model reads:

$$\frac{dx}{dt} = \alpha\, x - \beta\, x\, y, \qquad \frac{dy}{dt} = \delta\, x\, y - \gamma\, y,$$

where $x(t)$ is the prey population, $y(t)$ the predator population, and $\alpha, \beta, \gamma, \delta > 0$.

Proposition 10.6 (First Integral)**.** The Lotka–Volterra system admits the conserved quantity:

$$H(x, y) = \delta\, x - \gamma\, \ln x + \beta\, y - \alpha\, \ln y.$$

Phase-plane trajectories are therefore closed curves around the equilibrium $(\gamma/\delta,\ \alpha/\beta)$.

Proof. Compute $\frac{dH}{dt} = \delta\,\dot{x} - \frac{\gamma}{x}\,\dot{x} + \beta\,\dot{y} - \frac{\alpha}{y}\,\dot{y}$. Substituting the system equations, all terms cancel: $dH/dt = 0$. □

Attention

The classical Lotka–Volterra model is structurally unstable: it has no attracting limit cycle. Any perturbation changes the oscillation amplitude permanently. For a more realistic model, add a carrying capacity (logistic term) or a Holling-type functional response.

Example 10.7. Historical data from the Hudson's Bay Company (hare and lynx pelts, 1845–1935) show phase-shifted oscillations characteristic of predator–prey dynamics. Least-squares calibration yields periods of approximately 10 years.

10.3 Traffic Flow Models

Definition 10.8 (LWR Model)**.** The **Lighthill–Whitham–Richards** (LWR) model is a scalar conservation law for the vehicle density $\rho(x,t)$:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}\left[Q(\rho)\right] = 0,$$

where $Q(\rho) = \rho\, v(\rho)$ is the flux and $v(\rho)$ is the speed-density relation.

Proposition 10.9 (Greenshields Relation)**.** The Greenshields model assumes $v(\rho) = v_{\max}(1 - \rho/\rho_{\max})$, giving a parabolic flux:

$$Q(\rho) = v_{\max}\,\rho\left(1 - \frac{\rho}{\rho_{\max}}\right).$$

The capacity (maximum flux) is $Q_{\max} = v_{\max}\rho_{\max}/4$, attained at $\rho = \rho_{\max}/2$.

Remark 10.10. Solutions of the LWR model can develop discontinuities (*shocks*), corresponding physically to traffic jam formation. The Rankine–Hugoniot condition gives the shock speed: $s = [Q(\rho_R) - Q(\rho_L)]/(\rho_R - \rho_L)$.

10.4 Introduction to Climate Modeling

Definition 10.11 (Energy Balance Model)**.** The **Budyko–Sellers energy balance model** describes the mean temperature T of the Earth:

$$C\,\frac{dT}{dt} = (1 - \alpha(T))\,Q - \varepsilon\,\sigma\,T^4,$$

where C is the heat capacity, Q the incoming solar flux, $\alpha(T)$ the albedo (temperature-dependent via ice cover), and $\varepsilon\,\sigma\,T^4$ the emitted infrared radiation.

Theorem 10.12 (Bifurcation and "Snowball Earth")**.** *The Budyko–Sellers model generically admits three equilibria: a warm state (little ice), a "snowball" state (fully glaciated Earth), and an unstable intermediate equilibrium. When the solar flux Q decreases, a saddle-node bifurcation triggers an abrupt transition to the glaciated state.*

Intuition

Ice albedo (~ 0.7) is much higher than ocean albedo (~ 0.06). The colder it gets, the more ice forms, the more sunlight is reflected, the colder it gets... This is a positive feedback loop that can lead to a runaway glaciation.

10.5 Model Comparison and Validation

Definition 10.13 (Akaike Information Criterion (AIC))**.** Let $\hat{L}$ be the maximum likelihood of a model with k parameters. The **Akaike Information Criterion** is:

$$\text{AIC} = -2 \ln \hat{L} + 2k.$$

The model with the smallest AIC is preferred.

Definition 10.14 (Bayesian Information Criterion (BIC))**.** The **BIC** (or Schwarz criterion) is:

$$\text{BIC} = -2 \ln \hat{L} + k \ln n,$$

where n is the number of observations. BIC penalizes complexity more heavily than AIC when $n \geq 8$.

Proposition 10.15 (Cross-Validation)**.** k-fold **cross-validation** estimates the prediction error by splitting the data into k parts. The estimated error is:

$$\text{CV}(k) = \frac{1}{k} \sum_{j=1}^{k} \text{Error}_j,$$

where Error_j is the error of the model fitted on the other $k-1$ parts and evaluated on part j.

Attention

A good fit to training data does not guarantee good predictive power. *Overfitting* occurs when the model captures noise rather than signal. Always validate on independent data.

Key Formulas

Model Comparison Criteria

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \quad \text{(coefficient of determination)}$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \quad \text{(root mean squared error)}$$

$$\text{AIC} = -2 \ln \hat{L} + 2k, \qquad \text{BIC} = -2 \ln \hat{L} + k \ln n$$

Exercise 10.1. Implement the SEIR model with COVID-19 parameters ($\sigma^{-1} \approx 5.2$ days, $\gamma^{-1} \approx 10$ days, $R_0 \approx 2.5$). Simulate the dynamics for a city of 100,000 inhabitants and study the effect of lockdown (reduced R_0).

Exercise 10.2. Calibrate a Lotka–Volterra model on the Hudson's Bay Company hare–lynx dataset. Estimate parameters by nonlinear least squares and assess fit quality via R^2 and RMSE.

Exercise 10.3. Compare SIR, SEIR, and SIR-with-vaccination models on an epidemic dataset. Use AIC and BIC to determine the most parsimonious model.

Exercise 10.4. Solve the LWR model numerically with the Greenshields relation to simulate traffic jam formation at a red light. Use a Godunov scheme and visualize the density $\rho(x, t)$ in space and time.

CHAPTER 10. CASE STUDIES AND PROJECTS

Bibliography

[1] Murray, J.D., *Mathematical Biology*, 3rd ed., 2 vols., Springer, 2002–2003.

[2] Strogatz, S.H., *Nonlinear Dynamics and Chaos*, 2nd ed., Westview, 2015.

[3] Fowler, A.C., *Mathematical Models in the Applied Sciences*, Cambridge, 1997.