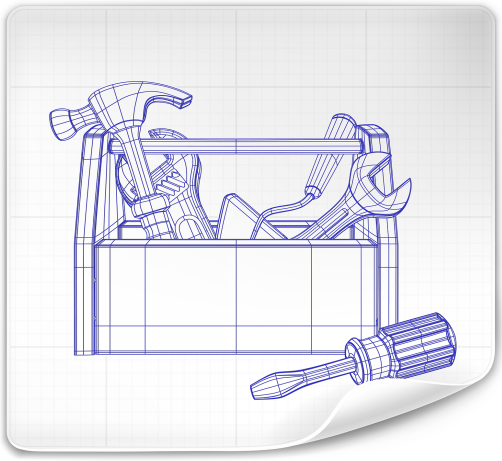
Continuous Mathematics

(complete)



Andrew D. Ker

16 Lectures, Hilary Term 2023

Oxford University Department of Computer Science

Contents

Introduction to the Lecture Notes

Course

The **Continuous Mathematics** course is compulsory for first-year undergraduates in Computer Science and Mathematics & Computer Science. There are 16 lectures, five tutorial sheets, and no formal practicals. (Some practical material will appear on the problem sheets.)

It also forms part of the option **Mathematics for Computer Science and Philosophy** for second- or third-year undergraduates in Computer Science & Philosophy. Approximately 10 hours of lecture material covers this option, for which there will be two tutorial sheets.

Prerequisites

Functions and their properties: 1-1 and onto, increasing and strictly increasing, invertible.

Basic familiarity with differentiation of real functions and turning points. Integration as area under a curve.

Linear algebra: vectors, matrices, multiplication, vector norm; eigenvalues and eigenvectors; rank of a matrix.

Syllabus

Derivatives, partial derivatives, differentiation with respect to a vector; gradient, Hessian, Jacobian. Taylor's theorem in 1 and n dimensions, with Lagrange remainder. Optimization in 1 and n dimensions, classification of stationary points. Constrained optimization

and the method of Lagrange multipliers. Methods for numerical integration in 1 and n dimensions, complexity and error analysis. Iterative numerical methods and rates of convergence. Methods for numerical root finding and numerical optimization in 1 and n dimensions, complexity and (simple cases of) error analysis. Applications.

Outline of Lectures

There are seven sections, very roughly one per week. Each builds on the previous material.

*Derivatives and Taylor's Theorem: Derivatives, partial derivatives, differentiation with respect to a vector, gradient, Hessian and Jacobian. Taylor's theorem in 1 dimension (Lagrange remainder), Taylor's theorem in n dimensions (remainder only briefly). Examples.

*Optimization: Optimization in 1 and n dimensions. Classification of turning points via Taylor's theorem. Convexity. Constrained optimization: Lagrange multipliers. Examples.

Algorithms for Numerical Integration: Numerical integration in 1 dimension: midpoint and Simpson's rules, complexity and error analysis. Briefly, integration in n dimensions and Monte Carlo methods. Examples.

*Accuracy: Floating-point numbers and rules of thumb for accuracy in practice. Convergence rates of iterative methods.

*Algorithms for Root Finding: Numerical root finding in 1 dimension by bisection, Newton's method, secant method. Root finding in n dimensions by Newton's method, and briefly quasi-Newton methods. Complexity and error analysis. Examples.

Algorithms for Optimization: Numerical optimization in 1 and n dimensions: root finding for gradient and gradient descent methods. Complexity and error analysis. Examples.

Applications: An extended example from steganography and steganalysis, illustrating constrained optimization, numerical root finding, and numerical optimization.

Sections marked * are relevant for the Mathematics for Computer Science & Philosophy option.

Reading Material

The lecture notes are designed to be self-contained, but it is a good idea to read around the subject. Sadly there seem to be few textbooks at the appropriate level: most either only cover one-dimensional numerical methods, or are graduate texts, or contain all the mathematical detail at a level unsuitable for this year.

- **H. Neill. Calculus: A Complete Introduction. Teach Yourself, 2013.**

  A *very* gentle introduction to calculus suitable for preparation before this course. Many, many, examples of simple differentiation. Does not cover our syllabus beyond the very first part.

- **U. M. Ascher and C. Greif. A First Course Numerical Methods. SIAM, 2011.**

  The least-bad option I could find. Nicely written, with good examples. Does not have much material on chapter 2 but otherwise is a reasonable place to find an alternative exposition. It is a bit more advanced than the lecture notes, and this course will only cover a fraction of what is in the book. Expensive.

- **E. W. Cheney and D. R. Kincaid. Numerical Mathematics and Computing (Seventh Edition). Wadsworth Publishing Co Inc, 2012.**

  Rarely goes beyond one-dimensional methods, but what it covers is done at an appropriate level. A good counterpart to the above. Second-hand copies seem to be cheap.

- **R. Burden and J. Faires. Numerical Analysis (Ninth Edition). Brooks Cole, 2010.**

  Goes a bit further into n-dimensional methods, but also includes a lot of material that is not in this course. There is an international edition that is cheaply available.

If you find any other textbooks that are a good match for this syllabus, please let me know!

Many universities include a numerical algorithms course in their Mathematics or Computer Science programme, often in the second year. I found one set of lecture notes that might prove a helpful second source for this course. If you find others, please let me know.

- **Anthony Yeates.** Numerical Analysis II Lecture Notes. Available from `http://www.maths.dur.ac.uk/~bmjg46/na17.pdf`.

  This is from a second year mathematics course at Durham. The notes cover topics in linear algebra as well as continuous maths, albeit in a different order to our syllabus. Some parts are more towards the mathematical end of the topic, or beyond our syllabus and at a higher level, but these notes are a nice read.

Notation

You are strongly advised to keep in mind the *type* of every mathematical object you manipulate: scalars, vectors, functions (of scalars or vectors) returning scalars or vectors, matrices, and so on. In the case of matrices it is often helpful to annotate them with their dimensions. In the notes, vectors and functions that return vectors will always be typeset boldface ($\boldsymbol{x}, \boldsymbol{f}$) in contrast to scalars and functions that return scalars (x, f). Matrices will be in boldface uppercase ($\mathbf{A}$), as will operators that return matrices ($\mathbf{H}, \mathbf{J}$). Sets will be italic uppercase (D).

Vectors will be column vectors by default, and we write $\boldsymbol{x}^T$ to transpose $\boldsymbol{x}$ into a row vector where necessary. Similarly we write $\mathbf{A}^T$ for the transpose of matrix $\mathbf{A}$. $\mathbf{I}$ is the identity matrix, $\mathbf{0}$ the zero vector, and $\mathbf{1}$ the vector all of whose entries is 1. The Euclidean norm of vector $\boldsymbol{x}$ will be written $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T\boldsymbol{x}}$.

△ Throughout the notes, some formulae are marked with the symbol △. This indicates a formula that you should understand, but are not obliged to memorize: if an exam question were to use the corresponding technique or formula, I would have advised the examiners to give it as a hint. In some cases it is indicated, underneath △, what part of the formula you *are* expected to learn.

Bonus Mathematics

Each chapter of the notes contains a 'bonus mathematics' section. These are written at a higher level for the most ambitious students, and are not remotely part of the syllabus. They exhibit some of the mathematical methods used to prove numerical analysis results, for students who are particularly interested.

Practice Questions and Tutorial Exercises

At the end of each chapter are some **practice questions**: straightforward, but may involve some longer calculations, they are to help you test your own understanding and practice the algebra. Brief answers are provided on the final page of the chapter. It is recommended to try the practice questions immediately after the relevant lectures, to help you correct any misunderstanding quickly.

More substantial questions are set in the **tutorial exercises**, provided separately from the notes. There will be **five** tutorial sheets, with the last intended for vacation work. Model answers are provided for tutors' use only.

It is not expected that your tutors will want to discuss the practice questions and so it is possible to get by without doing them. However, they have been designed so that some of the tutorial exercises will be easier if the practice questions have been attempted beforehand.

Course Website

Course material can be found at `https://courses.cs.ox.ac.uk/course/view.php?name=ContMath_2022_2023` (you may have to 'enrol' onto the course in Moodle first).

The page will include lecture notes, slides from the lectures (but only after the lecture), a small number of supplementary recorded lectures covering worked examples, tutorial exercises, and some past/sample exam questions.

If you find mistakes in the materials, please email them to `adk@cs.ox.ac.uk` so that I can fix them for next time. Particularly egregious errors will be noted on the website.

Chapter 1

Derivatives and Taylor's Theorem

This course assumes that you are familiar, at least informally, with the concept of continuity and the calculation of derivatives. In this chapter we survey the ideas of continuous and differentiable **univariate** functions ($f : \mathbb{R} \to \mathbb{R}$, or $f : D \to \mathbb{R}$ where $D \subseteq \mathbb{R}$) and Taylor's theorem. Then we move on to **multivariate** functions ($f : \mathbb{R}^n \to \mathbb{R}$, or $f : D \to \mathbb{R}$ where $D \subseteq \mathbb{R}^n$), which in fact has a lot in common with the univariate case as long as one uses vector notation. Taylor's theorem, in both univariate and multivariate versions, will be the basis of much of the material in this course. We also take a brief look at the calculus of **vector-valued functions** $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$.

1.1 Continuity

Continuous maths is about continuous functions. Continuous functions are 'smooth', or 'have no jumps'. We will not dwell on the technical definition, but some mathematical analysis is needed if we want to make it formal:

Definition A function f **is continuous at** x if

$$\lim_{h \to 0} f(x + h) = f(x).$$

f is called **continuous** if it is continuous at every point of its domain.

This definition makes sense whether x is a scalar ($\in \mathbb{R}$) or a vector ($\in \mathbb{R}^n$), and also whether the output of f is a scalar or a vector. Indeed, continuity also makes sense for functions on many other sets, even some that are not numerical at all, but we will leave this to mathematicians. We can also leave to them the formal proofs that continuous functions satisfy certain properties, instead taking such properties for granted.

Not all functions are continuous. A non-continuous function we are likely to encounter, apart from dividing by or taking logs at zero, is the **sign function**: it usually means one of the following three functions $\mathrm{sgn} : \mathbb{R} \to \mathbb{R}$,

$$\mathrm{sgn}(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}, \qquad \mathrm{sgn}(x) = \begin{cases} -1, & x \leq 0 \\ 1, & x > 0 \end{cases}, \qquad \mathrm{sgn}(x) = \begin{cases} -1, & x < 0, \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}.$$

(Which one depends on the context: beware!) Common sense tells us the following.

Lemma 1.1 If f and g are continuous functions on some domain $D \subseteq \mathbb{R}$ then so are:

$$\begin{aligned} f + g, & \\ cf, & \quad \text{for a constant } c \in \mathbb{R}, \\ f^n, & \quad \text{for a constant } n \in \mathbb{N}, \\ fg, & \\ f \circ g, & \\ \max(f, g), & \\ |f|, & \\ \exp f, & \\ f^\alpha, & \quad \text{for a constant } \alpha \in \mathbb{R}, \textbf{ where } f \textbf{ is strictly positive}, \\ \frac{f}{g}, & \quad \textbf{where } g \textbf{ is nonzero}, \\ \log f, & \quad \textbf{where } f \textbf{ is strictly positive}. \end{aligned}$$

Similar rules apply to functions that return vectors, when the operations are well-defined.

Note that it is *possible* that, for example, f/g is continuous where g is zero: this sometimes happens when f is also zero at that point, but is not guaranteed. We will leave these difficult cases to mathematicians.

It follows that the class of continuous functions on $\mathbb{R}$ includes polynomials and combinations of exponentials and polynomials. It also includes square roots, quotients, and logarithms of polynomials, as long as we restrict the domain to an interval which ensures that the function is properly defined. The trigonometric functions sin and cos are also continuous (this is not a consequence of the above Lemma).

Example 1.2 The function $f : \mathbb{R} \to \mathbb{R}$ is given by

$$f(x) = \ln\Big(\frac{1}{1+e^{-x}}\Big).$$

The function $l : \mathbb{R}^n \to \mathbb{R}$ is given by

$$l(\boldsymbol{w}) = \sum_{i=1}^{m} -y_i f(\boldsymbol{w}^T \boldsymbol{x_i}) - (1-y_i) f(-\boldsymbol{w}^T \boldsymbol{x_i}),$$

where $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}$ are fixed vectors in $\mathbb{R}^n$ and $y_1, \ldots, y_m$ are fixed scalars. The function $g : \mathbb{R}^n \to \mathbb{R}$ is given by

$$g(\boldsymbol{\mu}) = \sqrt{\exp\bigg(-\sum_{i=1}^{m} (\boldsymbol{x_i}-\boldsymbol{\mu})^T \mathbf{A} (\boldsymbol{x_i}-\boldsymbol{\mu})\bigg)},$$

where $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}$ are fixed vectors in $\mathbb{R}^n$ and $\mathbf{A}$ is a fixed n-by-n matrix.

They will be used in examples throughout this chapter. All of f, l, and g are continuous.

1.2 Differentiation (1 dimension)

Let us limit our attention to continuous **univariate** functions, $f : D \to \mathbb{R}$, where $D \subseteq \mathbb{R}$.

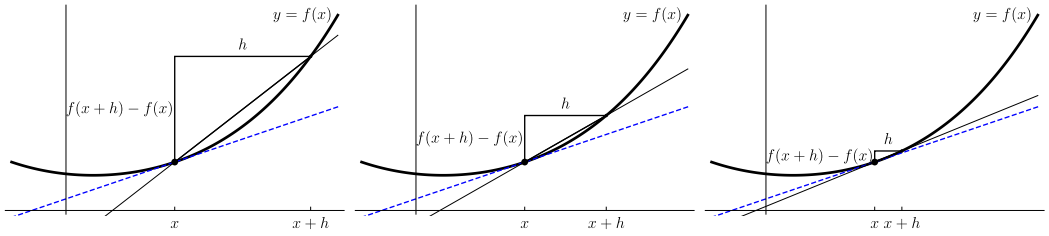


Figure 1.1: As $h \to 0$, the secant connecting $\big(x, f(x)\big)$ with $\big(x+h, f(x+h)\big)$ (solid line) approaches the tangent to the curve at x (dashed line), which has gradient $\frac{\mathrm{d}f}{\mathrm{d}x}$.

Most of this course is about making approximations to continuous functions, and then using them in algorithms. Sadly, not all continuous functions have good approximations. Our main mathematical tool, Taylor's theorem (sects. 1.3 and 1.7) applies to functions that have derivatives. Again, a technical definition:

Definition A continuous function $f : D \to \mathbb{R}$, where $D \subseteq \mathbb{R}$, is called **differentiable at** x if

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = d \tag{1.1}$$

exists for $d \in \mathbb{R}$. The value of d is called the **derivative** at x. f is called **differentiable** if it is differentiable at every point of D.

We write $\frac{\mathrm{d}f}{\mathrm{d}x}$ for the derivative function. If f is differentiable at some, but not all, points of D then this will be a partial function.

The derivative, if it exists, is the **gradient** of the function at x. See Figure 1.1 for the picture that shows how the gradient is approximated by the slope of the **secant** – the chord that runs from $\big(x, f(x)\big)$ to $\big(x+h, f(x+h)\big)$ – for small h.

Since $\frac{\mathrm{d}f}{\mathrm{d}x}$ is a function, we can apply it to an argument. We will write $\frac{\mathrm{d}f}{\mathrm{d}x}(a)$ to mean the value of the derivative function at a; in mathematics books this is commonly written $\frac{\mathrm{d}f}{\mathrm{d}x}\big|_{x=a}$ or $\frac{\mathrm{d}f}{\mathrm{d}x}\big|_{a}$, but our choice will reduce visual clutter. Do not confuse $\frac{\mathrm{d}f(a)}{\mathrm{d}x}$ with $\frac{\mathrm{d}f}{\mathrm{d}x}(a)$: the first is the constant zero function, because $f(a)$ is a constant; the second is the one we need.

Not all functions are differentiable. Non-continuous functions cannot be, but even some common continuous functions fail to be differentiable, for example $x \mapsto |x|$ is not differentiable at zero, nor is $x \mapsto \min(0, x)$. As with continuity, we leave to mathematicians the

formal proofs that differentiable functions have the following properties that we learned at school.

Lemma 1.3 The same facts about continuity in Lemma 1.1 also apply to derivatives, **except for $|f|$, which is not necessarily differentiable where f changes sign, and $\max(f,g)$, which is not necessarily differentiable where $f-g$ changes sign**. Some rules for differentiation include:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}(f+g) &= \frac{\mathrm{d}f}{\mathrm{d}x}+\frac{\mathrm{d}g}{\mathrm{d}x}, \\
\frac{\mathrm{d}}{\mathrm{d}x}(cf) &= c\frac{\mathrm{d}f}{\mathrm{d}x}, && \text{(the derivative is a \textbf{linear operator})} \\
\frac{\mathrm{d}}{\mathrm{d}x}(fg) &= f\frac{\mathrm{d}g}{\mathrm{d}x}+g\frac{\mathrm{d}f}{\mathrm{d}x}, && \text{(the \textbf{product rule})} \\
\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{f}{g}\right) &= \frac{g\frac{\mathrm{d}f}{\mathrm{d}x}-f\frac{\mathrm{d}g}{\mathrm{d}x}}{g^2}, && \text{(the \textbf{quotient rule})} \\
\frac{\mathrm{d}}{\mathrm{d}x}(g\circ f) &= \left(\frac{\mathrm{d}g}{\mathrm{d}x}\circ f\right)\frac{\mathrm{d}f}{\mathrm{d}x}, && \text{(the \textbf{chain rule})}
\end{aligned}$$

assuming all the functions are differentiable at the points concerned. Some standard derivatives are:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}x^\alpha &= \alpha x^{\alpha-1}, & \frac{\mathrm{d}}{\mathrm{d}x}\sin x &= \cos x, \\
\frac{\mathrm{d}}{\mathrm{d}x}e^x &= e^x, & \frac{\mathrm{d}}{\mathrm{d}x}\cos x &= -\sin x, \\
\frac{\mathrm{d}}{\mathrm{d}x}\ln x &= \frac{1}{x}, & \frac{\mathrm{d}}{\mathrm{d}x}\log_2 x &= \frac{1}{x\ln 2} = \frac{\log_2 e}{x}.
\end{aligned}$$

A sometimes-useful rule that can be derived from the chain and quotient rules is, if all the derivatives exist and $k\in\mathbb{N}$,

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{f}{g^k}\right)=\frac{g\frac{\mathrm{d}f}{\mathrm{d}x}-kf\frac{\mathrm{d}g}{\mathrm{d}x}}{g^{k+1}}. \tag{1.2}$$

If you think of $\frac{\mathrm{d}}{\mathrm{d}x}$ as an operator that works on functions, it makes sense to apply it twice. The derivative of the derivative of f, if it exists, is $\frac{\mathrm{d}}{\mathrm{d}x}\frac{\mathrm{d}}{\mathrm{d}x}f = \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$. This is the **second derivative**, and more generally we can attempt to compute a k-th derivative

$$\frac{\mathrm{d}^k f}{\mathrm{d}x^k}.$$

Example 1.4 With f as in **Example 1.2**, we can calculate

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{1}{1+e^x}, \quad \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = \frac{-e^x}{(1+e^x)^2}, \quad \frac{\mathrm{d}^3 f}{\mathrm{d}x^3} = \frac{e^{2x}-e^x}{(1+e^x)^3}, \quad \frac{\mathrm{d}^4 f}{\mathrm{d}x^4} = \frac{-e^{3x}+4e^{2x}-e^x}{(1+e^x)^4}.$$

We have used Leibniz's notation for derivatives; you may already know that there are various alternatives. Newton's notation uses a dot for derivative: if $y = f(x)$ then $\dot{y} = \frac{\mathrm{d}f}{\mathrm{d}x}$, $\ddot{y} = \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$, etc. This has its place, but is rarely used in computer science. A few books write Df or $\mathcal{D}f$ for $\frac{\mathrm{d}f}{\mathrm{d}x}$. Lagrange's notation uses the prime symbol $'$, so $f'(x)$ instead of $\frac{\mathrm{d}f}{\mathrm{d}x}$, $f''(x)$ for $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$, and so on. For the k-th derivative we can write $f^{(k)}(x)$; we need the parenthesis to avoid confusion with the k-th power of $f(x)$. Lagrange's notation can be convenient, and it is certainly a lot nicer to write

$$f^{(k)}(a) \quad \text{instead of} \quad \left.\frac{\mathrm{d}^k f}{\mathrm{d}x^k}\right|_{x=a}$$

but we prefer $\frac{\mathrm{d}^k f}{\mathrm{d}x^k}(a)$. Lagrange's notation does not work so well when we might differentiate with respect to multiple variables (or when f is a function of vectors) so we will mostly avoid it in these notes.

1.3 Taylor's Theorem (1 dimension)

A common and useful way to approximate a function is through Taylor's theorem. The theorem itself is exact, but it is often used to make an approximation to $f(x)$ if we know the value of a function f and some of its derivatives at a single point x_0. Typically, but not necessarily, x will be close to x_0.

Theorem 1.5 (Taylor's theorem) Let $k \geq 0$ be any integer. As long as $f : D \to \mathbb{R}$ satisfies some so-called **smoothness conditions**[1],

$$f(x) = f(x_0) + (x - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0) + \frac{(x - x_0)^2}{2!}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) + \cdots + \frac{(x - x_0)^k}{k!}\frac{\mathrm{d}^k f}{\mathrm{d}x^k}(x_0) + \frac{(x - x_0)^{k+1}}{(k+1)!}\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}}(\xi)\,,$$

for some $\xi \in (x_0, x)$.

The theorem holds for all x_0, x, and k (subject to f satisfying the smoothness conditions), but beware that ξ depends on them. A proof of Taylor's theorem is provided as bonus material in Section 1.8.

We will want to rewrite Taylor's theorem in some different forms. First, if we take $\frac{\mathrm{d}^0 f}{\mathrm{d}x^0}$ (differentiating zero times) to mean the original function f, we can write

$$f(x) = \hat{f}_k(x) + e_{k+1}(x, x_0)$$

where

$$\hat{f}_k(x) = \sum_{i=0}^{k} \frac{(x - x_0)^i}{i!}\frac{\mathrm{d}^i f}{\mathrm{d}x^i}(x_0)$$

is called the **order-k Taylor polynomial** or sometimes the **truncated Taylor series**. Note that $\hat{f}_k$ is a polynomial of degree k, and furthermore the *unique* polynomial of degree k that agrees with f as to its first k derivatives at x_0. We use it as an *approximation* for f near x_0, with $e_{k+1}(x, x_0)$ telling us the error in this approximation.

$e_{k+1}(x, x_0)$ is called the **error term**, also called the **Lagrange remainder term**,

$$e_{k+1}(x, x_0) = \frac{(x - x_0)^{k+1}}{(k+1)!}\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}}(\xi).$$

Mathematicians know equivalent formulae for the remainder: one is

$$e_{k+1}(x, x_0) = \int_{x_0}^{x} \frac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}}(t)\frac{(x - t)^k}{k!}\,\mathrm{d}t,$$

[1] Sufficient conditions are that f and its first $k+1$ derivatives all exist and are continuous on an interval including x_0 and x.

but the Lagrange version will work best for us. If $e_{k+1}(x, x_0)$ is the error in the approximation

$$f(x) \approx \hat{f}_k(x)$$

then we might hope that this error is small. If $|x - x_0| < 1$ then the term $(x - x_0)^{k+1}$ helps to make it small for large k, as does dividing by $(k + 1)!$, but it is possible that the k+1-st derivative of f gets larger as k does. e_{k+1} is guaranteed to be 'small' in the following sense: as $x \to x_0$, $e_{k+1}(x, x_0)/(x - x_0)^k \to 0$ (this is another fact that we will leave to mathematicians to prove). However this doesn't mean that it is actually small for a particular combination of f, x_0, and x, and indeed this error might be much bigger than the value of the function itself. When we apply Taylor's theorem, we will have to examine $e_{k+1}(x, x_0)$ and see.

Typically we want to find upper and lower bounds on the error, which often come from upper and lower bounds on the k+1-st derivative: if $\underline{C} \leq \frac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}} \leq \overline{C}$ on the interval (x_0, x), then

$$\hat{f}_k(x) + \underline{C}\frac{(x - x_0)^{k+1}}{(k + 1)!} \leq f(x) \leq \hat{f}_k(x) + \overline{C}\frac{(x - x_0)^{k+1}}{(k + 1)!},$$

with the inequalities reversed if k is even and $x < x_0$. This is the essence of Taylor's theorem as a numerical approximation: the k-th order Taylor polynomial has an error that depends on the k+1-st derivative of f.

Don't confuse Taylor's theorem with the concept of a **Taylor series**. The latter attempts to express the function f as an *infinite* sum with no remainder term:

$$f(x) \stackrel{?}{=} \sum_{i=0}^{\infty} \frac{(x - x_0)^i}{i!} \frac{\mathrm{d}^i f}{\mathrm{d}x^i}(x_0)$$

(this is called the **Taylor series of f around x_0**). Take care here: sometimes the infinite sum does not converge to any answer, and sometimes it does converge but to the wrong answer![2] A function where the error term in Taylor's theorem satisfies $e_k(x, x_0) \to 0$ as $k \to \infty$, for all x and small enough $x - x_0$, will always have a correct Taylor series and

[2] The classic example is $f(x) = \begin{cases} e^{-1/x^2}, & x \neq 0, \\ 0, & x = 0. \end{cases}$
It can be shown that f and all its derivatives are continuous, and at $x = 0$ *every derivative is zero.* So the Taylor series about $x = 0$ is $\sum_{i=0}^{\infty} \frac{x^i}{i!} \cdot 0 = 0$, which gives the wrong answer at every point except zero itself.

is called **analytic**. But as computer scientists we will find it more useful to use a finite sum (which we can compute) than an infinite one (which might or might not be correct anyway).

Example 1.6 (a) Consider $f(x) = \sin x$. Applying Taylor's theorem with $k = 8$ at $x_0 = 0$ gives

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}\cos\xi, \text{ for some } \xi \in (0, x).$$

It follows that

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \leq \sin x \leq x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!},$$

for $x \in [-\pi/2, \pi/2]$. So for $x = 0.1$ the approximation $\sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$ is accurate[3] to within $3 \cdot 10^{-15}$.

(b) Now consider $g(x) = \ln x$. Applying Taylor's theorem at $x_0 = 1$ gives

$$\ln x = \frac{(x-1)}{1} - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \cdots + (-1)^{k-1}\frac{(x-1)^k}{k} + (-1)^k\frac{(x-1)^{k+1}}{k+1}\frac{1}{\xi^{k+1}}.$$

The remainder terms tells us that we can expect this Taylor polynomial approximation to get more accurate with larger k, as long as $0 < x \leq 2$. For $x \leq 0$, g is not even defined, and for $x > 2$ the Taylor approximations get *worse* with larger k.

[3] Although this seems to provide an efficient method for computing $\sin x$, a) it needs more work to avoid large errors with large x, and b) in hardware there are better alternatives anyway (e.g. the CORDIC method). If you are really placed in a position where no libraries are available and you need to implement your own trigonometric functions, do not rely on this course: read a specialized textbook instead.

1.4 Partial Differentiation

Now let us move to functions of more than one variable, $f : D \to \mathbb{R}$, for $D \subseteq \mathbb{R}^n$. Depending on the context these are called **multivariate functions** or **scalar fields**. When we differentiate f with respect to one variable x, holding all others constant, this is called a **partial derivative** and written $\frac{\partial f}{\partial x}$. There is nothing otherwise special about partial differentiation: it is the same operation, and satisfies the same properties, as differentiation of univariate functions.

Example 1.7 (a) If $c(x, y) = 2x^2 + xy + y^2$ then $\frac{\partial c}{\partial x} = 4x + y$ and $\frac{\partial c}{\partial y} = x + 2y$.

(b) If $d(x_1, \ldots, x_n) = \sum a_i x_i$ then $\frac{\partial d}{\partial x_k} = a_k$.

(c) If $f(x_1, \ldots, x_n) = \sum_i \sum_j a_{ij} x_i x_j$ then $\frac{\partial f}{\partial x_k} = \sum_i (a_{ik} + a_{ki}) x_i$.

(d) If $g(x_1, \ldots, x_n) = \sqrt{\sum_{i=1}^n x_i^2}$ (the **Euclidean norm**) then $\frac{\partial g}{\partial x_k} = \frac{x_k}{\sqrt{\sum_{i=1}^n x_i^2}}$.

Higher partial derivatives can also be computed: $\frac{\partial}{\partial y}\frac{\partial}{\partial x} f$ means to differentiate first with respect to x, holding other variables constant, and then y. It is written $\frac{\partial^2 f}{\partial y \partial x}$. Note that the order of differentiation is the reverse of what is written in the denominator (because operators, like functions, apply on the left). The good news is that, in most circumstances, we do not need to distinguish between the so-called **mixed partials** $\frac{\partial^2 f}{\partial y \partial x}$ and $\frac{\partial^2 f}{\partial x \partial y}$: **Clairaut's theorem** says that they will be equal at least as long as they are both continuous.

The Lagrange notation for partial derivatives is f_x for $\frac{\partial f}{\partial x}$, and f_{xy} for $\frac{\partial^2 f}{\partial y \partial x}$. We will not use it.

The second partial derivatives can be collected together into a matrix called the **Hessian**.

If f is a function of $x_1, \ldots, x_n$ then its Hessian is

$$\mathbf{H}(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \frac{\partial^2 f}{\partial x_n \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix},$$

i.e. the (i, j) entry is $\frac{\partial^2 f}{\partial x_i \partial x_j}$. When everything is continuous, Clairaut's theorem says that $\mathbf{H}(f)$ is symmetric. The role of this matrix is similar to $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ in the case of univariate functions, and we will see much more of it soon.

Note that the Hessian is a function of $\boldsymbol{x}$, and will normally take different values at different points. When we want to evaluate the Hessian at a particular $\boldsymbol{x}^*$ we write $\mathbf{H}(f)(\boldsymbol{x}^*)$. Some books write $\mathbf{H}_f(\boldsymbol{x}^*)$, which avoids the consecutive parentheses.

Example 1.8 (a) If $c(x, y) = 2x^2 + xy + y^2$ then $\frac{\partial^2 c}{\partial x^2} = 4$, $\frac{\partial^2 c}{\partial y} = 2$, and $\frac{\partial^2 c}{\partial x \partial y} = 1$, so

$$\mathbf{H}(c) = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}.$$

(b) If $d(x_1, \ldots, x_n) = \sum a_i x_i$ then all the second partials are zero and $\mathbf{H}(d)$ is the zero matrix.

(c) If $f(x_1, \ldots, x_n) = \sum_i \sum_j a_{ij} x_i x_j$ then $\frac{\partial^2 f}{\partial x_k^2} = 2a_{kk}$ and $\frac{\partial^2 f}{\partial x_l \partial x_k} = a_{kl} + a_{lk}$. $\mathbf{H}(f) = \mathbf{A} + \mathbf{A}^T$, where $\mathbf{A}$ is the matrix with entries (a_{ij}).

(d) If $g(x_1, \ldots, x_n) = \sqrt{\sum_{i=1}^n x_i^2}$ then $\frac{\partial^2 g}{\partial x_k^2} = \frac{\sum x_i^2 - x_k^2}{(\sum x_i^2)^{3/2}}$ and $\frac{\partial^2 g}{\partial x_l \partial x_k} = \frac{-x_k x_l}{(\sum x_i^2)^{3/2}}$. After some fancy algebra, the Hessian can be written as

$$\mathbf{H}(g) = \frac{\mathbf{I} - \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}} \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}^T}{g(\boldsymbol{x})}.$$

1.5 Differentiation (n dimensions, scalar function)

If collecting the second partial derivatives into a matrix makes sense, so does collecting the first partial derivatives into a vector. For a function $f : D \to \mathbb{R}$, for $D \subseteq \mathbb{R}^n$ it is very convenient to write

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

Some books write $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$ as $\mathbf{D}(f)$ or $\mathbf{D}f$ or $\mathcal{D}_{\boldsymbol{x}}f$; applied maths and physics texts tend to write it ∇f or $\nabla_{\boldsymbol{x}} f$, the **gradient** function of f. (In such books the Hessian is often denoted $\nabla^2 f$.) Note that, unlike f which returns a scalar, ∇f is a vector-valued function. A few sources write it as a row vector rather than a column vector, but we will stick to column vectors.

Our interest in collecting partial derivatives into a vector is because it allows us to use familiar identities. All but one of the following are quite easy to prove from the corresponding rules for univariate functions, but we will leave it to mathematicians.

Lemma 1.9 As long as all the derivatives concerned exist,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(f+g) &= \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} + \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}, \\
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(cf) &= c\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}, && \text{(the derivative is a \textbf{linear operator})} \\
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(fg) &= f\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}} + g\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}, && \text{(the \textbf{product rule})} \\
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\Big(\frac{f}{g}\Big) &= \frac{g\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} - f\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}}{g^2}, && \text{(the \textbf{quotient rule})}
\end{aligned}$$

If $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$,

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(g \circ f) = \Big(\frac{\mathrm{d}g}{\mathrm{d}x} \circ f\Big)\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}, \qquad \text{(a simple \textbf{chain rule})}$$

If $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}$,

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(g \circ \boldsymbol{f}) = \mathbf{J}(\boldsymbol{f})^T\Big(\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}} \circ \boldsymbol{f}\Big),$$

for $\mathbf{J}(\boldsymbol{f})$ see Sect. 1.6, (a case of the partial **chain rule**)

Some standard derivatives are:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\boldsymbol{a}^T\boldsymbol{x} = \boldsymbol{a}, \qquad \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{a} = \boldsymbol{a}, \qquad \text{(where } \boldsymbol{a} \text{ is a column vector)}$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{x} = 2\boldsymbol{x}, \qquad \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} = (\mathbf{A} + \mathbf{A}^T)\boldsymbol{x}, \qquad \text{(where } \mathbf{A} \text{ is a matrix)}$$

Computing derivatives in this way can simplify notation and save a lot of time: it is a good idea to practice it.

Example 1.10 If

$$h(\boldsymbol{x}) = \frac{(\boldsymbol{x}^T\boldsymbol{a})(\boldsymbol{b}^T\boldsymbol{x})}{\boldsymbol{x}^T\boldsymbol{x}}$$

then

$$\frac{\mathrm{d}h}{\mathrm{d}\boldsymbol{x}} = \frac{(\boldsymbol{x}^T\boldsymbol{x})(\boldsymbol{x}^T\boldsymbol{a})\boldsymbol{b} + (\boldsymbol{x}^T\boldsymbol{x})(\boldsymbol{b}^T\boldsymbol{x})\boldsymbol{a} - 2(\boldsymbol{x}^T\boldsymbol{a})(\boldsymbol{b}^T\boldsymbol{x})\boldsymbol{x}}{(\boldsymbol{x}^T\boldsymbol{x})^2},$$

though neither is defined at the origin.

With l and g as in **Example 1.2**, we can calculate

$$\frac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}} = \sum_{i=1}^{m} \boldsymbol{x_i}\big(f'(-\boldsymbol{w}^T\boldsymbol{x_i}) - y_i\big)$$

where $f'(z) = 1/(1 + e^z)$, and

$$\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{\mu}} = g(\boldsymbol{\mu})\Big(\tfrac{\mathbf{A}+\mathbf{A}^T}{2}\Big)\Big(\textstyle\sum_i \boldsymbol{x_i} - n\boldsymbol{\mu}\Big).$$

If $\mathbf{A}$ is symmetric then the middle term simplifies to $\mathbf{A}$.

1.6 Vector Fields and the Jacobian

Briefly, let us consider vector-valued functions $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$, sometimes called a **vector field**. These are not our primary interest in this course (they are very interesting to mathematicians and physicists, particularly cases such as $m = n = 3$) but they allow us techniques for calculating the Hessian of a scalar function conveniently: notice that the derivative of a scalar-valued multivariate function *is* a vector-valued function.

It is best to consider a vector-valued function to be a vector of scalar-valued functions, each giving one of its 'co-ordinates'. So $\boldsymbol{f}(\boldsymbol{x})$ can be broken into the vector

$$\begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{pmatrix}.$$

The object that acts like the derivative of $\boldsymbol{f}$ is called the **Jacobian**[4], which is the matrix

$$\mathbf{J}(\boldsymbol{f}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \frac{\partial f_m}{\partial x_3} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

This contains the gradient of each f_i as one of its *rows*. This leads to the rather unfortunate fact that the Jacobian of a scalar function is the transpose of its gradient, the row vector $(\frac{\partial f_1}{\partial x_1} \ \frac{\partial f_1}{\partial x_2} \ \cdots \ \frac{\partial f_1}{\partial x_n})$. Indeed some books prefer to define the Jacobian to be the transpose of what we have used above, so take care; our version is probably the more common.

Do not confuse the Hessian matrix with the Jacobian. Both are matrices which are functions of $\boldsymbol{x}$, but the Hessian is always square and contains *second* partial derivatives of a multivariate scalar-valued function, whereas the Jacobian is m-by-n and contains *first* partial derivatives of a vector-valued function.

[4] The Jacobian works like a vector derivative because

$$\lim_{\boldsymbol{h} \to \mathbf{0}} \frac{\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{h}) - \boldsymbol{f}(\boldsymbol{x}) - \mathbf{J}(\boldsymbol{f})\boldsymbol{h}}{\|\boldsymbol{h}\|} = \mathbf{0}$$

(compare with (1.1)) but we need not pursue this.

The reason we will use a Jacobian is the following observation, which comes directly from the definitions. For $f : \mathbb{R}^n \to \mathbb{R}$,

$$\mathbf{H}(f) = \mathbf{J}\big(\tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}\big)^T. \tag{1.3}$$

Furthermore, as long as everything in sight is continuous, the Hessian is symmetric so the transpose changes nothing.

Like derivatives, Jacobians have rules for applying them to sums, products, and compositions. (There is no quotient rule because one cannot divide a vector by a vector; the product rules are for dot products and scalar products.) Most of these are easy to prove from the definitions and the rules for univariate scalar functions, but the partial chain rule is more difficult. We will leave the proofs to mathematicians.

You may not need all these rules this year, but they are included as a reference for the future.

Lemma 1.11 As long as all the derivatives concerned exist,

$$\begin{aligned}
\mathbf{J}(\boldsymbol{f}+\boldsymbol{g}) &= \mathbf{J}(\boldsymbol{f}) + \mathbf{J}(\boldsymbol{g}), \\
\mathbf{J}(c\boldsymbol{f}) &= c\mathbf{J}(\boldsymbol{f}), && \text{(the Jacobian is a } \textbf{linear operator}\text{)} \\
\mathbf{J}(\mathbf{A}\boldsymbol{f}) &= \mathbf{A}\mathbf{J}(\boldsymbol{f}), && \text{where } \mathbf{A} \text{ is a constant matrix,} \\
\mathbf{J}(\boldsymbol{f}^T\boldsymbol{g}) &= \boldsymbol{g}^T\mathbf{J}(\boldsymbol{f}) + \boldsymbol{f}^T\mathbf{J}(\boldsymbol{g}), && \text{(the } \textbf{dot product rule}\text{)} \quad \triangle \\
\mathbf{J}(f\boldsymbol{g}) &= \boldsymbol{g}\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}^T + f\mathbf{J}(\boldsymbol{g}), && \text{(the } \textbf{product of scalar and vector rule}\text{)} \quad \triangle \\
\mathbf{J}(\boldsymbol{g}\circ\boldsymbol{f}) &= \big(\mathbf{J}(\boldsymbol{g})\circ\boldsymbol{f}\big)\mathbf{J}(\boldsymbol{f}), && \text{(the } \textbf{partial chain rule}\text{)}
\end{aligned}$$

Some standard Jacobians are:

$$\mathbf{J}(\boldsymbol{x}) = \mathbf{I}, \qquad \mathbf{J}(\mathbf{A}\boldsymbol{x}) = \mathbf{A}, \qquad \text{(where } \mathbf{A} \text{ is a matrix)}$$

A special case of the product of scalar and vector rule is when the vector is constant: if $\boldsymbol{c}$ does not depend on $\boldsymbol{x}$ then

$$\mathbf{J}\big(f(\boldsymbol{x})\boldsymbol{c}\big) = \boldsymbol{c}\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}^T. \tag{1.4}$$ $\triangle$

Example 1.12 Let us use (1.3) to compute the Hessian of the function l from **Example 1.2**. In **Example 1.10** we showed

$$\frac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}} = \sum_{i=1}^{m} \boldsymbol{x_i}\big(f'(-\boldsymbol{w}^T\boldsymbol{x_i}) - y_i\big)$$

where $f'(z) = 1/(1+e^z)$. Recognizing $\frac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}}$ as a linear combination of products of the constant vector $\boldsymbol{x_i}$ with the scalar function $f'(-\boldsymbol{w}^T\boldsymbol{x_i}) - y_i$, we apply (1.4) to deduce:

$$\mathbf{H}(l) = \mathbf{J}\big(\tfrac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}}\big)^T = \sum_{i=1}^{m} -\boldsymbol{x_i} f''(-\boldsymbol{w}^T\boldsymbol{x_i})\boldsymbol{x_i}^T$$

where $f''(z) = -e^z/(1+e^z)^2$, which can compactly be written

$$\mathbf{H}(l) = \mathbf{X}^T\mathbf{D}\,\mathbf{X},$$

where $\mathbf{X}$ is the m-by-n matrix whose rows are the $\boldsymbol{x_i}$, and $\mathbf{D}$ is the m-by-m diagonal matrix with entries $-f''(-\boldsymbol{w}^T\boldsymbol{x_i})$ on the diagonal. We will return to this example in the next chapter.

1.7 Taylor's Theorem (n dimensions, scalar function)

Taylor's theorem has a version for multivariate functions, which has similar uses: we can approximate a function near a point by a multivariate polynomial, whose coefficients depend on its (partial) derivatives at that point. And we can write a formula for the error in this approximation. At first sight it is a mess of notation, but we will find ways of making it more comprehensible.

Theorem 1.13 (Taylor's theorem) Let $D \subseteq \mathbb{R}^n$ and $f : D \to \mathbb{R}$. Given vectors $\boldsymbol{x}, \boldsymbol{x_0} \in \mathbb{R}^d$, write their difference as $\boldsymbol{x} - \boldsymbol{x_0} = \boldsymbol{h} = (h_1, \ldots, h_n)^T$. Let $k \geq 0$ be an integer. As long as f satisfies some smoothness conditions[5] then

$$\begin{aligned} f(\boldsymbol{x}) &= f(\boldsymbol{x_0}) \\ &\quad + \Big[\big(h_1 \tfrac{\partial}{\partial x_1} + \ldots + h_n \tfrac{\partial}{\partial x_n}\big) f\Big](\boldsymbol{x_0}) \\ &\quad + \frac{1}{2!}\Big[\big(h_1 \tfrac{\partial}{\partial x_1} + \ldots + h_n \tfrac{\partial}{\partial x_n}\big)^2 f\Big](\boldsymbol{x_0}) \\ &\quad + \cdots \\ &\quad + \frac{1}{k!}\Big[\big(h_1 \tfrac{\partial}{\partial x_1} + \ldots + h_n \tfrac{\partial}{\partial x_n}\big)^k f\Big](\boldsymbol{x_0}) \\ &\quad + \frac{1}{(k+1)!}\Big[\big(h_1 \tfrac{\partial}{\partial x_1} + \ldots + h_n \tfrac{\partial}{\partial x_n}\big)^{k+1} f\Big](\boldsymbol{x_0} + \xi \boldsymbol{h}) \end{aligned} \tag{1.5}$$

for some $\xi \in (0, 1)$. The final term is the **error term**, denoted $e_{k+1}(\boldsymbol{x}, \boldsymbol{x_0})$.

A sketch of how to derive this from the univariate version is provided in Section 1.8.

This looks intimidating, but it has similarities to the one-dimensional version of Taylor's theorem: derivatives of order up to k evaluated at $\boldsymbol{x_0}$, and a remainder term that involves k+1-st order derivatives evaluated somewhere on the straight line between $\boldsymbol{x_0}$ and $\boldsymbol{x}$. Let us simplify it gradually.

First, what do we mean by a term like $\Big[\big(h_1 \frac{\partial}{\partial x_1} + \ldots + h_n \frac{\partial}{\partial x_n}\big)^2 f\Big]$? It is a shorthand: if we multiply out the square it becomes

$$\begin{aligned} & h_1^2 \tfrac{\partial}{\partial x_1}\tfrac{\partial}{\partial x_1} f \quad + h_1 h_2 \tfrac{\partial}{\partial x_1}\tfrac{\partial}{\partial x_2} f \quad + \cdots \quad + h_1 h_n \tfrac{\partial}{\partial x_1}\tfrac{\partial}{\partial x_n} f \\ + \; & h_2 h_1 \tfrac{\partial}{\partial x_2}\tfrac{\partial}{\partial x_1} f \quad + h_2 h_2 \tfrac{\partial}{\partial x_2}\tfrac{\partial}{\partial x_2} f \quad + \cdots \quad + h_2 h_n \tfrac{\partial}{\partial x_2}\tfrac{\partial}{\partial x_n} f \\ + \; & \cdots \\ + \; & h_n h_1 \tfrac{\partial}{\partial x_n}\tfrac{\partial}{\partial x_1} f \quad + h_n h_2 \tfrac{\partial}{\partial x_n}\tfrac{\partial}{\partial x_2} f \quad + \cdots \quad + h_n h_n \tfrac{\partial}{\partial x_n}\tfrac{\partial}{\partial x_n} f \\ = \; & \textstyle\sum_i h_i^2 \frac{\partial^2 f}{\partial x_i^2} + 2 \sum \sum_{i<j} h_i h_j \frac{\partial^2 f}{\partial x_i \partial x_j} \end{aligned}$$

[5] If f and all its first $k+1$ partial derivatives exist and are continuous on a region including $\boldsymbol{x_0}$ and $\boldsymbol{x}$.

(as long as everything in sight is continuous, so that the mixed partials are equal). The same applies to the other terms.

Let's take the case of a function of two variables $f(x, y)$, up to third order with a fourth-order error term. So $n = 2, k = 3$. We will use (x_0, y_0) for the vector $\boldsymbol{x_0}$, similarly (x, y) for $\boldsymbol{x}$. Then (1.5) becomes

$$
\begin{aligned}
f(x,y) \quad &= \\
& f(x_0,y_0) \\
&+ \ (x-x_0)\tfrac{\partial f}{\partial x}(\boldsymbol{x_0}) + (y-y_0)\tfrac{\partial f}{\partial y}(\boldsymbol{x_0}) \\
&+ \ \frac{1}{2!}\Big((x-x_0)^2\tfrac{\partial^2 f}{\partial x^2}(\boldsymbol{x_0}) + 2(x-x_0)(y-y_0)\tfrac{\partial^2 f}{\partial x\,\partial y}(\boldsymbol{x_0}) + (y-y_0)^2\tfrac{\partial^2 f}{\partial y^2}(\boldsymbol{x_0})\Big) \\
&+ \ \frac{1}{3!}\Big((x-x_0)^3\tfrac{\partial^3 f}{\partial x^3}(\boldsymbol{x_0}) + 3(x-x_0)^2(y-y_0)\tfrac{\partial^3 f}{\partial x^2\,\partial y}(\boldsymbol{x_0}) + 3(x-x_0)(y-y_0)^2\tfrac{\partial^3 f}{\partial x\,\partial y^2}(\boldsymbol{x_0}) \\
&\qquad + (y-y_0)^3\tfrac{\partial^3 f}{\partial y^3}(\boldsymbol{x_0})\Big) \\
&+ \ \frac{1}{4!}\Big((x-x_0)^4\tfrac{\partial^4 f}{\partial x^4}(\boldsymbol{x^*}) + 4(x-x_0)^3(y-y_0)\tfrac{\partial^4 f}{\partial x^3\,\partial y}(\boldsymbol{x^*}) + 6(x-x_0)^2(y-y_0)^2\tfrac{\partial^4 f}{\partial x^2\,\partial y^2}(\boldsymbol{x^*}) \\
&\qquad + 4(x-x_0)(y-y_0)^3\tfrac{\partial^4 f}{\partial x\,\partial y^3}(\boldsymbol{x^*}) + (y-y_0)^4\tfrac{\partial^4 f}{\partial y^4}(\boldsymbol{x^*})\Big)
\end{aligned}
$$

where $\boldsymbol{x^*} = \boldsymbol{x_0} + \xi(\boldsymbol{x} - \boldsymbol{x_0})$, with $\xi \in (0, 1)$.

In Taylor's theorem the remainder can be shown to be 'small' in the following sense: $e_{k+1}(\boldsymbol{x}, \boldsymbol{x_0})/\|\boldsymbol{x} - \boldsymbol{x_0}\|^k \to 0$ as $\|\boldsymbol{x} - \boldsymbol{x_0}\| \to 0$. As in the one-dimensional case, whether it is actually small in relation to $f(\boldsymbol{x})$ depends on the function f. The good news is that we will rarely need to examine a remainder term above second order.

Recall that Taylor's univariate theorem approximates $f(x)$ as a polynomial of order k in x, whose coefficients depend on f and its derivatives at x_0, and is the only such polynomial that agrees with the first k derivatives of f at x_0. The multivariate version approximates $f(\boldsymbol{x})$ as a multivariate polynomial of order k in $\boldsymbol{x}$, whose coefficients depend on f and its partial derivatives all at $\boldsymbol{x_0}$, and is the unique such that agrees with those derivatives.

Finally, we can make Taylor's theorem look a lot less messy if we use vector notation: notice that

$$\big(h_1\tfrac{\partial}{\partial x_1} + \ldots + h_n\tfrac{\partial}{\partial x_n}\big)f = \boldsymbol{h}^T\tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$$

and

$$\left(h_1 \tfrac{\partial}{\partial x_1} + \ldots + h_n \tfrac{\partial}{\partial x_n}\right)^2 f = \boldsymbol{h}^T \mathbf{H}(f) \boldsymbol{h}.$$

We can only do this up to the second term. The third term would involve a 'three-dimensional matrix' of all mixed third-order partials: such a thing does exist (it's called a rank-3 tensor) but it is way beyond the scope of this course. The good news is that we almost always only need to use Taylor's theorem at order 0, 1, or 2. Putting everything together, we have the following simplified cases:

Lemma 1.14 (The multivariate Taylor's theorem that you will want to learn)

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x^*}) \qquad \text{(0th order)}$$

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_0})^T \mathbf{H}(f)(\boldsymbol{x^*})(\boldsymbol{x} - \boldsymbol{x_0}) \qquad \text{(1st order)}$$

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_0})^T \mathbf{H}(f)(\boldsymbol{x_0})(\boldsymbol{x} - \boldsymbol{x_0}) + e_3 \qquad \text{(2nd order)}$$

where in each case $\boldsymbol{x^*}$ lies on the line between $\boldsymbol{x_0}$ and $\boldsymbol{x}$. The remainder term e_3 is complicated to write out, but at least it is 'small' in the sense that $\frac{e_3}{\|\boldsymbol{x}-\boldsymbol{x_0}\|^2} \to 0$ as $\|\boldsymbol{x} - \boldsymbol{x_0}\| \to 0$.

Example 1.15 (a) If $f(x, y, z) = 3x \sin y + 2(y + 1)e^z$ the Taylor 2nd-order approximation around $\boldsymbol{x_0} = (0, 0, 0)$ is

$$f_2(x, y, z) = 2 + 2y + 2z + 3xy + 2yz + z^2,$$

and the error e_3 satisfies $\frac{e_3}{\|\boldsymbol{x}\|^2} \to 0$ as $\|\boldsymbol{x}\| \to 0$. (In fact $|e_3| < 0.0015$ for $x, y, z \in (-0.1, 0.1)$.)

(b) Let us find the 1st-order Taylor approximation, with 2nd-order error term, for the function l from **Example 1.2**, at a point $\boldsymbol{w_0}$. In **Example 1.10** we found its gradient, and in **Example 1.12** its Hessian. Plugging them into Lemma 1.14,

$$l(\boldsymbol{w}) = l(\boldsymbol{w_0}) + (\boldsymbol{w} - \boldsymbol{w_0})^T \frac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}}(\boldsymbol{w_0}) + e_2$$

where

$$e_2 = \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w_0})^T \mathbf{H}(l)(\boldsymbol{w^*})(\boldsymbol{w} - \boldsymbol{w_0}).$$

for some $\boldsymbol{w^*}$ on a line between $\boldsymbol{w_0}$ and $\boldsymbol{w}$. With some algebra we can prove that $e_2 \geq 0$.

This tells us that, if we choose $\boldsymbol{w_0}$ so that $\frac{\mathrm{d}l}{\mathrm{d}\boldsymbol{w}}(\boldsymbol{w_0}) = \mathbf{0}$, i.e. all the partial derivatives are zero, then $l(\boldsymbol{w}) \geq l(\boldsymbol{w_0})$. We have proved that any stationary point of l must be a global minimum. We will see much more of this in the next chapter.

1.8 Bonus Mathematics: Proof of Taylor's Theorem

This is <u>not</u> part of the syllabus.

There are many ways to prove the one-dimensional version of Taylor's theorem. Maths & CS students will see one in the maths course that goes something like this: define

$$F(t) = f(t) + (x-t)\frac{\mathrm{d}f}{\mathrm{d}x}(t) + \frac{(x-t)^2}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(t) + \cdots + \frac{(x-t)^n}{n!}\frac{\mathrm{d}^n f}{\mathrm{d}x^n}(t),$$
$$G(t) = (t-x)^{n+1}.$$

Then apply Cauchy's mean value theorem to F and G on $[x_0, x]$.

The above gives a relatively short proof, but it is difficult to understand where Taylor's theorem 'comes from'. I like the following proof, which shows how Taylor's theorem is a generalization of the Fundamental Theorem of Calculus. Recall that

$$f(b) - f(a) = \int_a^b \frac{\mathrm{d}f}{\mathrm{d}x}(t)\,\mathrm{d}t, \quad \text{or} \quad f(b) = f(a) + \int_a^b \frac{\mathrm{d}f}{\mathrm{d}x}(t)\,\mathrm{d}t,$$

at least as long as $\frac{\mathrm{d}f}{\mathrm{d}x}$ is continuous.

Apply this in the setting of Taylor's theorem and we have

$$f(x) = f(x_0) + \int_{x_0}^{x} \frac{\mathrm{d}f}{\mathrm{d}x}(x_1)\,\mathrm{d}x_1$$

but (assuming enough continuous derivatives) we can apply it again to $\frac{\mathrm{d}f}{\mathrm{d}x}$:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x_1) = \frac{\mathrm{d}f}{\mathrm{d}x}(x_0) + \int_{x_0}^{x_1} \tfrac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_2)\,\mathrm{d}x_2$$

and repeatedly substituting (always moving constants to the front of an integral) leads to

$$\begin{aligned}
f(x) &= f(x_0) + \int_{x_0}^{x} \tfrac{\mathrm{d}f}{\mathrm{d}x}(x_1)\,\mathrm{d}x_1 \\
&= f(x_0) + \frac{\mathrm{d}f}{\mathrm{d}x}(x_0)\int_{x_0}^{x} 1\,\mathrm{d}x_1 + \int_{x_0}^{x}\int_{x_0}^{x_1} \tfrac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_2)\,\mathrm{d}x_2 x_1 \\
&= f(x_0) + \frac{\mathrm{d}f}{\mathrm{d}x}(x_0)\int_{x_0}^{x} 1\,\mathrm{d}x_1 + \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0)\int_{x_0}^{x}\int_{x_0}^{x_1} 1\,\mathrm{d}x_2 x_1 + \int_{x_0}^{x}\int_{x_0}^{x_1}\int_{x_0}^{x_2} \tfrac{\mathrm{d}^3 f}{\mathrm{d}x^3}(x_3)\,\mathrm{d}x_3 x_2 x_1 \\
&= \quad \cdots \\
&= \overbrace{f(x_0) + \frac{\mathrm{d}f}{\mathrm{d}x}(x_0)\int_{x_0}^{x} 1\,\mathrm{d}x_1 \quad + \cdots + \quad \frac{\mathrm{d}^k f}{\mathrm{d}x^k}(x_0)\int_{x_0}^{x}\int_{x_0}^{x_1}\cdots\int_{x_0}^{x_{k-1}} 1\,\mathrm{d}x_k \cdots x_2 x_1}^{\hat{f}_k(x)} \\
&+ \underbrace{\int_{x_0}^{x}\int_{x_0}^{x_1}\cdots\int_{x_0}^{x_k} \tfrac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}}(x_{k+1})\,\mathrm{d}x_{k+1}\cdots x_2 x_1}_{e_{k+1}(x,x_0)}.
\end{aligned}$$

Now it is straightforward to unwind the iterated integrals, doing the inner integral each time:

$$\begin{aligned}
\int_{x_0}^{x}\cdots\int_{x_0}^{x_{i-2}}\int_{x_0}^{x_{i-1}}\int_{x_0}^{x_i} 1\,\mathrm{d}x_{i+1}\cdots x_1 &= \int_{x_0}^{x}\cdots\int_{x_0}^{x_{i-1}}\int_{x_0}^{x_{i-1}} (x_i - x_0)\,\mathrm{d}x_i \cdots x_1 \\
&= \int_{x_0}^{x}\cdots\int_{x_0}^{x_{i-1}} \frac{(x_{i-1}-x_0)^2}{2}\,\mathrm{d}x_{i-1}\cdots x_1 = \cdots = \frac{(x-x_0)^{i+1}}{(i+1)!}.
\end{aligned}$$

This shows that the part labelled $\hat{f}_k(x)$ matches the Taylor polynomial.

For the remainder term, let $\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x}(\underline{a}) = \underline{m}$ and $\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x}(\overline{a}) = \overline{m}$ be the minimum and maximum values of $\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x}$ in $[x_0, x]$, and define $g(t) = \frac{\mathrm{d}^{k+1} f}{\mathrm{d}x}(t)\frac{(x-x_0)^{k+1}}{(k+1)!}$.

It follows that the remainder term is bounded by

$$g(\underline{a}) = \int_{x_0}^{x}\cdots\int_{x_0}^{x_k} \underline{m}\,\mathrm{d}x_{k+1}\cdots x_1 \le e_{k+1}(x, x_0) \le \int_{x_0}^{x}\cdots\int_{x_0}^{x_k} \overline{m}\,\mathrm{d}x_{k+1}\cdots x_1 = g(\overline{a}),$$

with the inequalities strict unless $\frac{\mathrm{d}^{k+1}f}{\mathrm{d}x^{k+1}}$ is constant. Since g is continuous, by the intermediate value theorem we must have

$$e_{k+1}(x, x_0) = g(\xi)$$

for some $\xi \in (\underline{a}, \overline{a}) \subseteq (x, x_0)$.

To prove the multivariate version of Taylor's theorem, one can simply apply the univariate version to the function $h(z) = f(\boldsymbol{g}(z))$, where $\boldsymbol{g}(z) = \boldsymbol{x_0} + z\boldsymbol{h}$ with $\boldsymbol{h} = \boldsymbol{x} - \boldsymbol{x_0}$. Using the partial chain rule we can compute

$$\frac{\mathrm{d}h}{\mathrm{d}z} = \boldsymbol{h}^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0} + z\boldsymbol{h}) \quad \text{and} \quad \frac{\mathrm{d}^2h}{\mathrm{d}z^2} = \boldsymbol{h}^T \mathbf{H}(f)(\boldsymbol{x_0} + z\boldsymbol{h})\boldsymbol{h}$$

and hence

$$\begin{aligned} f(\boldsymbol{x}) = h(1) &= h(0) + \frac{\mathrm{d}h}{\mathrm{d}z}(0) + \frac{1}{2}\frac{\mathrm{d}^2h}{\mathrm{d}z^2}(0) + \cdots \\ &= f(\boldsymbol{x_0}) + \boldsymbol{h}^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0}) + \frac{1}{2}\boldsymbol{h}^T \mathbf{H}(f)(\boldsymbol{x_0})\boldsymbol{h} + \cdots \end{aligned}$$

showing that the first two terms of the Taylor expansion of f match the statement of Lemma 1.14. Extending this to higher terms, as in Theorem 1.13, uses the same techniques but the algebra becomes messy.

Practice Questions

Don't forget to see if the function simplifies, before you start calculating.

1.1 For which $x \in \mathbb{R}$ are the following functions continuous? For which are they differentiable? Compute the derivative where it exists.

(i) $x^3 + x^2 + x + 1$, (ii) $\frac{x^3+x^2+x+1}{x^2+x}$, (iii) $\ln(1+x)$,

(iv) $\ln \sin x$, (v) $\sqrt{x^2-1}$, (vi) $\sqrt{\frac{x+1}{x-1}}$,

(vii) $f(x) = g(x)h(x)j(x)$, for some functions $g, h, j : \mathbb{R} \to \mathbb{R}$, .

1.2 For each of the following functions $f(x, y)$ compute $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial^2 f}{\partial x^2}$, $\frac{\partial^2 f}{\partial y^2}$, and $\frac{\partial^2 f}{\partial x \partial y}$.

(i) $x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$, (ii) $\sqrt{2x^2 + 3y^2}$,

(iii) $\frac{x-y}{(x+y)^4}$, (iv) x^y,

(v) $f(x, y) = g(x)h(y)$, for some functions $g, h : \mathbb{R} \to \mathbb{R}$, .

1.3 For each of the following scalar functions $f(\boldsymbol{x})$ compute $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$, and using (1.3) compute $\mathbf{H}(f)$. Here $\mathbf{A}$ and $\mathbf{B}$ are fixed matrices, $\boldsymbol{a}$ and $\boldsymbol{b}$ fixed vectors.

(i) $\boldsymbol{x}^T\boldsymbol{a}\boldsymbol{x}^T\boldsymbol{b}$, (ii) $(\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{B}\boldsymbol{x})$, (iii) $\sqrt{\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}}$, assuming $\mathbf{A}$ is symmetric.

1.4 Find the third-order Taylor polynomial, and the fourth-order error term, for $f(x) = e^x + e^{-x}$ around $x_0 = 0$. Find upper and lower bounds on the error term if this is used to approximate $f(x)$ for $x \in (-0.1, 0.1)$.

1.5 How many nonzero terms in the Taylor polynomial above (for $f(x) = e^x + e^{-x}$ around $x_0 = 0$) are sufficient if we want to ensure that it is accurate to within 10^{-6} in the region $x \in (-1, 1)$?

1.6 For each function $f : \mathbb{R}^n \to \mathbb{R}$ write its second-order Taylor expansion in the form of Lemma 1.14. Do not compute the error term.

(i) $f(x, y, z) = e^{xyz}$, around $(x, y, z) = (0, 1, 2)$,

(ii) $f(\boldsymbol{x}) = \ln\bigl(1 + \sum_i x_i\bigr)$ around $\boldsymbol{x} = \mathbf{0}$.

Answers to Chapter 1 Practice Questions

1.1 (i) Continuous and differentiable everywhere; $3x^2+2x+1$. (ii) Notice that you can cancel a factor of $1+x$; continuous and differentiable everywhere except $x=-1,0$; $\frac{x^2-1}{x^2}$. (iii) Continuous and differentiable for $x>-1$; $\frac{1}{1+x}$. (iv) Continuous and differentiable for $x\in(2n\pi,(2n+1)\pi)$ for any $n\in\mathbb{Z}$; $\frac{\cos x}{\sin x}=\cot x$. (v) Continuous and differentiable everywhere outside $[-1,1]$; $x/\sqrt{x^2-1}$. (vi) Continuous and differentiable for $x>1$ and $x<-1$; $-\sqrt{(x-1)/(x+1)}/(x-1)^2$. (vii) Continuous and differentiable at least when all of g, h, and j are (possibly also for other regions); $g'(x)h(x)j(x)+g(x)h'(x)j(x)+g(x)h(x)j'(x)$ (in Lagrange's notation).

1.2

	$\frac{\partial f}{\partial x}$	$\frac{\partial f}{\partial y}$	$\frac{\partial^2 f}{\partial x^2}$	$\frac{\partial^2 f}{\partial y^2}$	$\frac{\partial^2 f}{\partial x\,\partial y}$
*(i)	$4(x+y)^3$	$4(x+y)^3$	$12(x+y)^2$	$12(x+y)^2$	$12(x+y)^2$
(ii)	$\frac{2x}{\sqrt{2x^2+3y^2}}$	$\frac{3y}{\sqrt{2x^2+3y^2}}$	$\frac{6y^2}{(2x^2+3y^2)^{3/2}}$	$\frac{6x^2}{(2x^2+3y^2)^{3/2}}$	$\frac{-6xy}{(2x^2+3y^2)^{3/2}}$
**(iii)	$\frac{-3x+5y}{(x+y)^5}$	$\frac{-5x+3y}{(x+y)^5}$	$\frac{12x-28y}{(x+y)^6}$	$\frac{28x-12y}{(x+y)^6}$	$\frac{20x-20y}{(x+y)^6}$
(iv)	yx^{y-1}	$x^y\ln x$	$y(y-1)x^{y-2}$	$x^y(\ln x)^2$	$x^{y-1}(1+y\ln x)$
***(v)	$g'(x)h(y)$	$g(x)h'(y)$	$g''(x)h(y)$	$g(x)h''(y)$	$g'(x)h'(y)$

*save a lot of work by spotting that the function is just $(x+y)^4$. **rule (1.2) is useful here. ***this answer given using Lagrange's notation.

1.3 (i) $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}=(\boldsymbol{x}^T\boldsymbol{a})\boldsymbol{b}+(\boldsymbol{x}^T\boldsymbol{b})\boldsymbol{a}$; note the outer multiplications are scalar. $\mathbf{H}(f)=\boldsymbol{b}\boldsymbol{a}^T+\boldsymbol{a}\boldsymbol{b}^T$.
(ii) $f(\boldsymbol{x})$ can be simplified to $\boldsymbol{x}^T(\mathbf{A}^T\mathbf{B})\boldsymbol{x}$, so $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}=(\mathbf{A}^T\mathbf{B}+\mathbf{B}^T\mathbf{A})\boldsymbol{x}$ and $\mathbf{H}(f)=\mathbf{A}^T\mathbf{B}+\mathbf{B}^T\mathbf{A}$.
(iii) $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}=\frac{\mathbf{A}\boldsymbol{x}}{\sqrt{\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}}}$; $\mathbf{H}(f)=\frac{\mathbf{A}}{(\boldsymbol{x}^T\mathbf{A}\boldsymbol{x})^{1/2}}-\frac{\mathbf{A}\boldsymbol{x}(\mathbf{A}\boldsymbol{x})^T}{(\boldsymbol{x}^T\mathbf{A}\boldsymbol{x})^{3/2}}$. Compare with Example 1.8(d).

1.4 $f(0)=2$; $f'(0)=0$; $f''(0)=2$; $f'''(0)=0$; $f''''(\xi)=e^\xi+e^{-\xi}$. Putting it together, $f(x)=2+x^2+\frac{x^4}{24}(e^\xi+e^{-\xi})$ for some $\xi\in(0,x)$. Since

$$2\le e^\xi+e^{-\xi}\le e^{0.1}+e^{-0.1}\approx 2.01$$

for $\xi\in(-0.1,0.1)$, the error term is at least zero and bounded above by $\frac{0.1^4}{24}(e^{0.1}+e^{-0.1})\approx 8.36\cdot10^{-6}$. (In fact the highest error happens at $x=0.1$, and is about $8.34\cdot10^{-6}$: the upper bound for the error is close to tight, in this instance.)

1.5 Extending the above calculation, all odd derivatives of f are zero at $x_0=0$, and all even derivatives are 2, so we should examine even-order error terms. The $2k+1$-order Taylor polynomial (which has only $k+1$ nonzero terms: the even powers) has error term

$$0\le e_{2k+2}=\tfrac{x^{2k+2}}{(2k+2)!}(e^\xi+e^{-\xi})<\tfrac{e+\frac{1}{e}}{(2k+2)!}.$$

using the fact that e^x+e^{-x} is an even function which is increasing above zero. Checking by hand, this becomes less than 10^{-6} when $k=4$, i.e. the Taylor polynomial $f_9(x)=2+x^2+\frac{1}{12}x^4+\frac{1}{360}x^6+\frac{1}{20160}x^8$.

1.6 (i)

$$f(x,y,z)=1+\begin{pmatrix}2\\0\\0\end{pmatrix}^T\begin{pmatrix}x\\y-1\\z-2\end{pmatrix}+\frac{1}{2!}\begin{pmatrix}x\\y-1\\z-2\end{pmatrix}^T\begin{pmatrix}4&2&1\\2&0&0\\1&0&0\end{pmatrix}\begin{pmatrix}x\\y-1\\z-2\end{pmatrix}+e_3$$

or $f(x,y,z)=1-2x+2x^2+2xy+xz+e_3$, where e_3 is the third-order error term.
(ii) The gradient and Hessian are a constant vector and matrix, respectively. At $\mathbf{0}$, we have $f(\boldsymbol{x})=\sum_i x_i-\frac{1}{2}\big(\sum_i x_i\big)^2+e_3$.

Chapter 2

Optimization

Many of the computer science applications of continuous maths are in **optimization**: finding the minimum of a function. In this chapter we find them exactly, using derivatives. We also examine the related concept of **convexity**, a property which ensures that the minimum is unique (and generally makes it easier to find). As in the last chapter, we do these first for univariate functions and then for multivariate scalar functions.

If the function is not differentiable then we cannot apply the techniques of this chapter. Neither can we if the minimum is the root of an equation that we cannot solve. For an alternative, we will look at numerical algorithms for finding *approximate* minima in Chapter 6.

2.1 Optimization and Constraints

The general form of the **optimization** problem is as follows: find

$$\min_{\boldsymbol{x} \in F} f(\boldsymbol{x}).$$

The above expression denotes the *value* of the minimum; in practice we often care more about the *location*, which is written

$$\arg\min_{\boldsymbol{x} \in F} f(\boldsymbol{x}).$$

This is the $\boldsymbol{x}$ that attains the minimum value: $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for any $\boldsymbol{y} \in F$. If there is more than one such $\boldsymbol{x}$ (i.e. a tie for the least value of f) then $\arg\min$ is usually considered to return the set of points that attain the minimum. Symmetrically, $\arg\max f$ is the location or set of locations of the maximum value of f.

Instances of this problem are pervasive in computer science. We might want to minimize numerical error or running time of an algorithm. Minimizing the **loss function** is the keystone of machine learning. Minimizing each participant's cost arises in game theory and operations research. And maximizing entropy is often an aim of privacy-preserving technologies. There are countless more examples.

Some optimization terminology: f is called the **objective function**. F is called the **feasible** set. If f takes values in $\mathbb{R}^n$ and F is the whole of $\mathbb{R}^n$ then the optimization problem is called **unconstrained**, otherwise it is called **constrained**. In the case of constrained optimization, F is typically given in a **standard form**:

$$F = \{\boldsymbol{x} \in \mathbb{R}^n \mid g_1(\boldsymbol{x}) = 0, \ldots, g_l(\boldsymbol{x}) = 0, h_1(\boldsymbol{x}) \geq 0, \ldots, h_m(\boldsymbol{x}) \geq 0\}, \tag{2.1}$$

where $g_i(\boldsymbol{x}) = 0$ are l **equality constraints** and $h_i(\boldsymbol{x}) \geq 0$ are m **inequality constraints**. Sometimes l or m is zero: the techniques for solving optimization constrained only by equalities are simpler than those for inequalities (or a mixture of equalities and inequalities). Whatever the origin of the optimization problem we wish to solve, we will rewrite its constraints into a standard form like (2.1).

Optimization problems can fail to have a solution. If the constraints actually force $F = \emptyset$ then the problem is called **infeasible** or **inconsistent**. If there is no minimum because the objective function can take arbitrarily low values (for example $\arg\min_{x \in \mathbb{R}} x$) then the problem is called **unbounded**.

There is nothing special about minimization: we can convert maximization to it via $\max f(\boldsymbol{x}) = -\min -f(\boldsymbol{x})$. For historical reasons, methods and algorithms tend to focus on minimization rather than maximization. Optimization is also sometimes known as **programming**, in the old-fashioned military sense of allocating resources. There are some particular classes of optimization problems known as **linear programming**, **convex programming**, and so on.

Finally, a **local minimum** for a function f is a point $\boldsymbol{x} \in F$ such that

$$f(\boldsymbol{x}) \leq f(\boldsymbol{x} + \boldsymbol{h})$$

for vectors $\boldsymbol{h}$ that are sufficiently small. A local maximum is defined symmetrically. $\arg\min_x f(x)$ can be referred to as a/the **global minimum** if we want to make the distinction clear. It tends to be easier to find local minima, exactly or approximately, than global minima. In applications, however, we usually want the latter.

2.2 Optimization (1 dimension)

Let $f : \mathbb{R} \to \mathbb{R}$. We want to find

$$\arg\min_{x \in F} f(x),$$

for F is a closed interval: $\mathbb{R}$, $[a, b]$, $[a, \infty)$, or $(-\infty, b]$. In one dimension there aren't any interesting equality constraints, and the only useful inequality constraints set a maximum or minimum on x. In this chapter we assume that f is at least twice differentiable on F, and that those derivatives are continuous.

It would be rare to see an optimization problem on an *open* interval like (a, b), because if the minimum value occurs at an endpoint then it is not technically a minimum at all (it is not a member of F) but an infimum.

You probably already know a method to find the minimum. It begins with finding the **stationary points** of f, solving

The first-order condition:
$$\frac{\mathrm{d}f}{\mathrm{d}x} = 0.$$

This is called a first-order condition because it depends on the first derivative of f. In some books, the stationary points are called **critical points**. As you know, this only gets us part of the way, because not every stationary point is necessarily a local minimum (it might be a local maximum, or a **stationary point of inflection**) and not every local minimum is necessarily a global minimum. For unconstrained optimization, we might remember this as

$$\text{global minima} \subseteq \text{local minima} \subseteq \text{stationary points}$$

and for constrained optimization *we should also check whether the minimum occurs at an endpoint of the feasible interval.* Do not forget the last possibility.

A mathematical proof of why every local minimum/maximum has zero derivative is a bit fiddly (a result known as Fermat's theorem) but is also common sense, so we leave the formality to mathematicians.

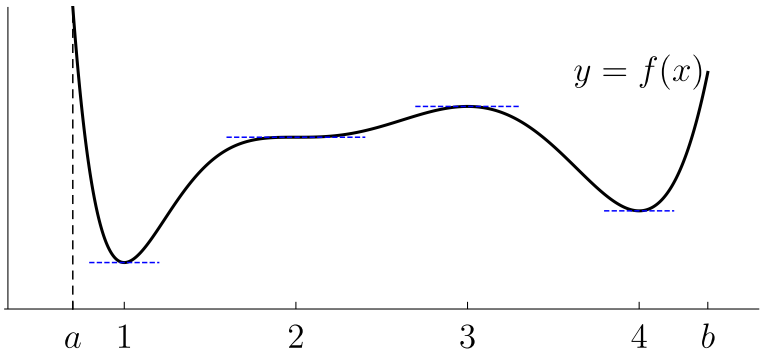


Figure 2.1: Different cases of the first-order condition. $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ at $x = 1$ and this is the global minimum of f on $[a, b]$. $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ at $x = 2$ and this is a stationary point of inflection. $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ at $x = 3$ and this is a local maximum, but not a global maximum: the global maximum of f on $[a, b]$ occurs at $x = a$. $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ at $x = 4$ and this is a local minimum.

Having found the set of stationary points, we need to **classify** them: determine which are local minima, local maxima, and stationary points of inflection. Naively, we might simply test some values of the function: if the first-order conditions tell us that x_0 is a stationary point, evaluate $f(x_0 + \epsilon)$ and $f(x_0 - \epsilon)$ for one or more small ϵ, and compare them with $f(x_0)$. Unfortunately this cannot classify the stationary point, because we cannot be certain that we picked a small enough values of ϵ to avoid crossing any other turning points. Or rather we cannot be sure unless we do more mathematics to rule out such behaviour.

Instead, we can apply Taylor's theorem, which is an ideal tool to examine the local behaviour of a function. Suppose that x_0 is a stationary point. Consider the first-order Taylor polynomial for f at x_0, with second-order remainder:

$$f(x) = f(x_0) + \overbrace{(x - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)}^{=0} + \frac{(x - x_0)^2}{2!}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi) \tag{2.2}$$

for some $\xi \in (x_0, x)$. Since x_0 is a stationary point, $\frac{\mathrm{d}f}{\mathrm{d}x}(x_0) = 0$ and so the second term vanishes.

(i) If $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) > 0$ then it follows that $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi) > 0$ for all ξ close to x_0, by continuity of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$. The third term of (2.2) is non-negative, which means that $f(x) \geq f(x_0)$ for x

near x_0: a local minimum.

(ii) If $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) < 0$ then, for the same reasons, the third term is non-positive and $f(x) \leq f(x_0)$ for x near x_0: a local maximum.

Sometimes we write this as

A second-order sufficient condition: (for a minimum) $\dfrac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) > 0.$

It is common to find second-order conditions that are *sufficient* to classify x_0 as a local minimum, but are not *necessary*, and that is the case here too. *Some local minima do not satisfy this condition.*

On the other hand, $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) \geq 0$ is a *necessary* condition (if it fails then the stationary point is in fact a local maximum), but not a *sufficient* one.

If we want to classify all stationary points, we may have to work harder, because of the possibility that $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) = 0$. In that case we should look at the higher-order Taylor polynomial:

$$f(x) = f(x_0) + \overbrace{(x - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)}^{=0} + \overbrace{\frac{(x - x_0)^2}{2!}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0)}^{=0} + \frac{(x - x_0)^3}{3!}\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}(\xi)$$

for some $\xi \in (x_0, x)$. If $\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}(x_0) \neq 0$, the third term changes sign when $(x - x_0)$ changes sign, meaning that this is neither a local minimum nor local maximum; it must be a stationary point of inflection. If $\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}(x_0) = 0$,

$$f(x) = f(x_0) + \overbrace{(x - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)}^{=0} + \overbrace{\frac{(x - x_0)^2}{2!}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0)}^{=0} + \overbrace{\frac{(x - x_0)^3}{3!}\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}(x_0)}^{=0} + \frac{(x - x_0)^4}{4!}\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}(\xi)$$

and we are back where we started.

This shows that classifying stationary points can be complicated. In one dimension, if the first nonzero derivative is of odd order then we have a stationary point of inflection, and if it is of even order than we have a local minimum/maximum according to whether that nonzero derivative is positive/negative.

That is not even the end of the complications. For non-analytic functions it is possible that *every* derivative is zero at x_0, in which case Taylor's theorem fails to help classify the

stationary point. In higher dimensions, as we will see, it can be even more tricky and we will be happy to use sufficient second-order conditions to classify some of the stationary points, and hope that no difficult cases arise.

To summarise the one-dimensional case, we find $\arg\min_{x\in F} f(x)$ by the following process: differentiate f and, if you can solve the first-order condition, find its stationary points. Differentiate again (and as many times as necessary) to try to classify the stationary points. The global minimum is attained at one of the local minima, or on a boundary of the feasible interval if there is one.

Example 2.1 Let

$$f(x) = (x-1)^3(x-2)^4.$$

Find $\arg\min_{x\in[1,3]} f(x)$ and $\arg\max_{x\in[1,3]} f(x)$.

Finally, notice that optimization is a lot easier if $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x) > 0$ for all x: in that case there can be at most one stationary point, and if there is one it is the global minimum. We will return to this in Section 2.4.

2.3 Unconstrained Optimization (n dimensions)

Let us now consider multivariate functions, $f : \mathbb{R}^n \to \mathbb{R}$, and the same problem of optimization (minimization). For now let us assume no constraints, so the feasible set is the whole of $\mathbb{R}^n$. We use much the same method as in 1 dimension: find stationary points, classify them, and select the global minimum if there is more than one local minimum.

The stationary points are defined by

The first-order condition:
$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mathbf{0}.$$

Note that this is a vector equation, i.e. a set of simultaneous equations $\frac{\partial f}{\partial x_1} = 0, \ldots, \frac{\partial f}{\partial x_n} = 0$. It is not always easy to solve such a set of equations and there might be zero, one, or many solutions. For now, let us assume that we can solve the first-order condition to get a set of stationary points.

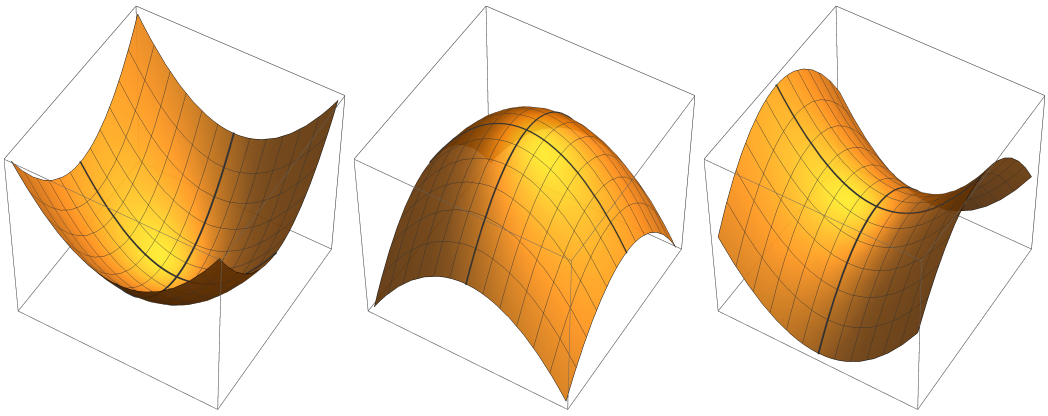


Figure 2.2: Stationary points in n dimensions (here $n = 2$). From left to right: a local minimum, a local maximum, a saddle point.

In multiple dimensions, a stationary point can be a local minimum, local maximum, or a **saddle point**: $\boldsymbol{x}$ is a saddle point if the local behaviour of f near $\boldsymbol{x}$ depends on the direction from which you approach $\boldsymbol{x}$. (For this reason some people even call the one-dimensional equivalent case – the stationary point of inflection – a saddle point, even though it does not look like a saddle.) See Figure 2.2 for examples of stationary points in n dimensions.

As before, let us use Taylor's theorem to examine the local behaviour of f near a stationary point $\boldsymbol{x_0}$. Using Lemma 1.14,

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + \overbrace{(\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})}^{=\mathbf{0}} + \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{x_0})^T \mathbf{H}(f)(\boldsymbol{x^*})(\boldsymbol{x} - \boldsymbol{x_0})$$

for some $\boldsymbol{x^*}$ between $\boldsymbol{x_0}$ and $\boldsymbol{x}$.

This brings us to a property of matrices which is extremely important for optimization:

Definition A symmetric matrix $\mathbf{A}$ is $\left\{ \begin{array}{c} \textbf{positive definite} \\ \textbf{positive semidefinite} \\ \textbf{negative definite} \\ \textbf{negative semidefinite} \end{array} \right\}$ if $\left\{ \begin{array}{c} \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} > 0 \\ \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} \geq 0 \\ \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} < 0 \\ \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} \leq 0 \end{array} \right\}$

for all nonzero $\boldsymbol{x}$.

A symmetric matrix that satisfies none of the above is called **indefinite**. Positive semidefinite is also sometimes called **non-negative definite** and negative semidefinite **non-positive definite**.

This allows us to classify stationary points in many, but not all, cases:

(i) If $\mathbf{H}(f)(\boldsymbol{x_0})$ is positive definite then (by a tricky continuity argument) so is $\mathbf{H}(f)(\boldsymbol{x^*})$ for $\boldsymbol{x^*}$ close enough to $\boldsymbol{x_0}$. It follows that $f(\boldsymbol{x}) \geq f(\boldsymbol{x_0})$ for $\boldsymbol{x}$ close to $\boldsymbol{x_0}$, so that $\boldsymbol{x_0}$ is a local minimum.

(ii) If $\mathbf{H}(f)(\boldsymbol{x_0})$ is negative definite then, for similar reasons, $\boldsymbol{x_0}$ is a local maximum.

(iii) If $\mathbf{H}(f)(\boldsymbol{x_0})$ is indefinite (omitting some mathematics), $\boldsymbol{x_0}$ is a saddle point.

Note that this does not cover the semidefinite cases, and we might have to go to a higher-order Taylor expansion, or use some other analytical tricks, to classify such stationary points.

But at least we have the following second-order condition, which is *sufficient* to guarantee that $\boldsymbol{x_0}$ is a local minimum:

A second-order sufficient condition: (for a minimum) $\mathbf{H}(f)(\boldsymbol{x_0})$ is positive definite.

Furthermore, $\mathbf{H}(f)(\boldsymbol{x_0})$ being positive semidefinite is a *necessary* condition: if not, there is some direction (an eigenvector corresponding to a negative eigenvalue) in which moving from $\boldsymbol{x_0}$ decreases f.

As you can see, the positive/negative definiteness condition on $\mathbf{H}(f)$ works similarly to the positivity/negativity of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ in the one dimension (after all, the 1-by-1 matrix $\left(\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right)$ *is* the Hessian), but there are additional cases. Furthermore, these results are only useful if you have techniques to establish whether a matrix is positive or negative (semi)definite. In two dimensions it is trivial:

Lemma 2.2 The symmetric 2-by-2 matrix $\mathbf{A} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ is

$$\left\{ \begin{array}{c} \text{positive definite} \\ \text{positive semidefinite, not definite} \\ \text{negative definite} \\ \text{negative semidefinite, not definite} \\ \text{indefinite} \end{array} \right\} \text{ if and only if } \left\{ \begin{array}{c} |\mathbf{A}| > 0 \text{ and } a > 0 \\ |\mathbf{A}| = 0 \text{ and } a + c \geq 0 \\ |\mathbf{A}| > 0 \text{ and } a < 0 \\ |\mathbf{A}| = 0 \text{ and } a + c \leq 0 \\ |\mathbf{A}| < 0 \end{array} \right\}.$$

Note that if $|\mathbf{A}| \geq 0$ then necessarily a and c have the same sign, or one is zero.

Example 2.3 Find and classify the stationary points of $f, g : \mathbb{R}^2 \to \mathbb{R}$,

$$f(x, y) = 3x - x^3 - 2y^2 + y^4,$$
$$g(x, y) = x^4 + y^2.$$

For higher dimensions there are various methods for deciding whether a matrix is positive/negative semidefinite/definite. Here are some that can be helpful:

Lemma 2.4

(i) $\mathbf{A}$ is $\left\{ \begin{array}{c} \text{positive definite} \\ \text{positive semidefinite} \\ \text{negative definite} \\ \text{negative semidefinite} \end{array} \right\}$ if and only if all its **eigenvalues** are $\left\{ \begin{array}{c} > \\ \geq \\ < \\ \leq \end{array} \right\} 0.$

(ii) $\mathbf{A}$ is $\left\{ \begin{array}{c} \text{positive definite} \\ \text{positive semidefinite} \\ \text{negative definite} \\ \text{negative semidefinite} \end{array} \right\}$ if and only if all its **pivots** are $\left\{ \begin{array}{c} > \\ \geq \\ < \\ \leq \end{array} \right\} 0.$

(The pivots are the leading entries in the unreduced row echelon form: where row additions have been used to convert $\mathbf{A}$ to upper triangular, but rows have not been divided to make their leading entry equal one. If the echelon form contains a row of all zeros, we call that a zero pivot.)

(iii) $\mathbf{A}$ is positive (semi)definite if and only if $-\mathbf{A}$ is negative (semi)definite.

(iv) The sum of positive semidefinite matrices is positive semidefinite. The sum of a positive definite and a positive (semi)definite matrix is positive definite. Similarly negative.

(v) For *any* symmetric matrix $\mathbf{C}$, $\mathbf{C} - \lambda\mathbf{I}$ is positive semidefinite as long as $\lambda \leq$ the △ smallest eigenvalue of $\mathbf{C}$. It is positive definite if the inequality is strict.

(vi) If $\mathbf{A}$ and $\mathbf{B}$ are positive (semi)definite matrices of the same size then so are:

(a) $c\mathbf{A}$, for any constant $c > 0$,

(b) $\mathbf{A}^{-1}$, which exists when $\mathbf{A}$ is positive definite,

△ (c) any upper-left submatrix of $\mathbf{A}$,

△ (d) $\mathbf{ABA}$,

(e) $\mathbf{C}^T\mathbf{AC}$, for any matrix $\mathbf{C}$ of full *column* rank (square or not),

△ (f) $\mathbf{A} - \boldsymbol{v}\boldsymbol{v}^T$, if $\boldsymbol{v}^T\mathbf{A}^{-1}\boldsymbol{v} < 1$.

In (e), even if $\mathbf{C}$ is not of full rank then we can still guarantee positive **semi**definiteness of $\mathbf{C}^T\mathbf{AC}$. In (f) we can still guarantee positive **semi**definiteness if the inequality in the side condition is non-strict.

Example 2.5 For which values of a, b, and c are the following positive definite or semidefinite?

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}, \qquad \begin{pmatrix} a^2 & a^2 & a^2 \\ a^2 & -b^2 & a^2 \\ a^2 & a^2 & a^2 \end{pmatrix}, \qquad \begin{pmatrix} 1-a^2 & -ab & -ac \\ -ab & 1-b^2 & -bc \\ -ac & -bc & 1-c^2 \end{pmatrix}.$$

Sometimes all this complexity can be avoided, in the helpful case where the Hessian matrix is positive definite everywhere: this is the topic of the next section.

Example 2.6 In a machine-learning example called **Linear Discriminant Analysis**, we have to find the *maximum* of the following function on $\mathbb{R}^n$,

$$S(\boldsymbol{w}) = \frac{(\boldsymbol{w}^T\boldsymbol{\mu})^2}{\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}},$$

where $\boldsymbol{\mu}$ is a fixed vector and $\boldsymbol{\Sigma}$ a fixed, positive definite, matrix.

The stationary points are:

(i) Whenever $\boldsymbol{w}^T\boldsymbol{\mu} = 0$. These are always local minima.

(ii) At $\boldsymbol{w} = \lambda\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$, for any $\lambda \neq 0$. Regardless of λ, S takes the same value at these stationary points. Here the Hessian is

$$\frac{2\boldsymbol{\mu}\boldsymbol{\mu}^T}{\lambda^2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}} - \frac{2\boldsymbol{\Sigma}}{\lambda^2}.$$

Using Lemma 2.4, we can argue that this is negative semidefinite (for all $\lambda \neq 0$), but unfortunately not negative definite and so not enough to prove that the stationary points are maxima. A more nuanced analysis of $S(\boldsymbol{w_0} + \boldsymbol{x})$, for small $\boldsymbol{x}$ using the multivariate version of Taylor's theorem, is needed to show that these points are (all) global maxima.

2.4 Convexity

There is a class of optimization problems where classifying the stationary points is much easier. We have already seen that when $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} > 0$ everywhere, or $\mathbf{H}(f)$ is positive definite everywhere, there can only be at most one stationary point, and if there is one then it is the global minimum. These are examples of functions that are **convex**, something you will meet often in optimization.

Intuitively, a convex function is 'bowl shaped', whether in 1 or more dimensions and whether differentiable or not. Only functions whose domains are convex sets can be convex:

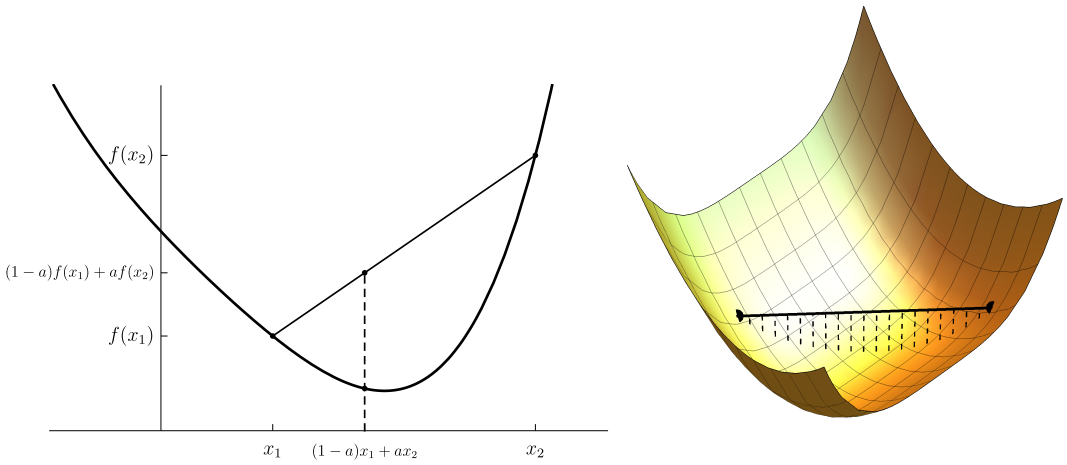


Figure 2.3: Convex functions in 1 and 2 dimensions. The secant connecting $\big(\boldsymbol{x_1}, f(\boldsymbol{x_1})\big)$ with $\big(\boldsymbol{x_2}, f(\boldsymbol{x_2})\big)$ lies above the curve or surface of f.

Definition A set $D \subseteq \mathbb{R}^n$ is called **convex** if, for all $\boldsymbol{x_1}, \boldsymbol{x_2} \in D$ and $\alpha \in (0,1)$,

$$(1-\alpha)\boldsymbol{x_1} + \alpha\boldsymbol{x_2} \in D.$$

A function $f : D \to \mathbb{R}$, for some convex set $D \subseteq \mathbb{R}^n$, is **convex** if for all $\boldsymbol{x_1}, \boldsymbol{x_2} \in D$ and $\alpha \in (0,1)$,

$$f\big((1-\alpha)\boldsymbol{x_1} + \alpha\boldsymbol{x_2}\big) \leq (1-\alpha)f(\boldsymbol{x_1}) + \alpha f(\boldsymbol{x_2}). \tag{2.3}$$

f is **strictly convex** if the inequality, above, is strict. It is called **concave** if the inequality is reversed, and **strictly concave** analogously.

Thus convexity of f ensures that any secant joining $\big(\boldsymbol{x_1}, f(\boldsymbol{x_1})\big)$ to $\big(\boldsymbol{x_2}, f(\boldsymbol{x_2})\big)$ lies above graph, and strict convexity that it lies strictly above the graph (except at the endpoints, where of course it meets the graph). See Figure 2.3. For concavity, the secants lie below the graph. You can see why the domain must be convex for the function to be convex, otherwise $f\big((1-\alpha)\boldsymbol{x_1} + \alpha\boldsymbol{x_2}\big)$ may not be defined.

The reason convexity matters to optimization is the following: in unconstrained optimization of convex f,

$$\text{global minima} = \text{local minima} = \text{stationary points}.$$

If furthermore f is strictly convex, there can be at most one stationary point. If there are none, there is no global minimum. *To find the global minimum of a convex objective function, therefore, we only need to solve the first-order condition.*

In the case of constrained optimization, it is still possible that the global minimum occurs on the boundary of F. When f is *strictly* convex this can only happen if there are no stationary points.

However, the definition of convexity (2.3) is usually not easy to check. In practice, we might establish it using the following lemma, which will be no surprise given the discussion in the preceding sections:

Lemma 2.7 Suppose that $f : D \to \mathbb{R}$, for some convex set $D \subseteq \mathbb{R}^n$, and that its second-order partial derivatives are all continuous.

(i) f is convex if and only if $\mathbf{H}(f)$ is positive semidefinite.
(ii) f is strictly convex if, but not only if, $\mathbf{H}(f)$ is positive definite.

The proof of this lemma is quite fiddly, so we will not cover it here. There are other characterisations of convexity in terms of derivatives, but we will not use them.

Often it is better to demonstrate that a function f is convex by expressing it as a suitable combination of functions we already know to be convex, saving a lot of calculation.

Lemma 2.8 (i) Linear functions (only linear functions!) are both convex and concave.

(ii) $\exp(x)$ is strictly convex, $\log x$ is strictly concave on $(0, \infty)$, and $|x|^p$ is strictly convex for $p > 1$ (convex for $p = 1$).

(iii) f is (strictly) convex if and only if $-f$ is (strictly) concave.

(iv) $\boldsymbol{x} \mapsto \boldsymbol{x}^T \mathbf{A} \boldsymbol{x}$ is convex/concave when $\mathbf{A}$ is positive/negative semidefinite, and strictly convex/concave when $\mathbf{A}$ is positive/negative definite.

(v) If f and g are (strictly) convex functions on some convex domain $D \subseteq \mathbb{R}^n$ then so are:

$$\begin{aligned} f + g, & \\ cf, & \quad \text{for any constant } c > 0, \\ \boldsymbol{x} \mapsto f(\mathbf{A}\boldsymbol{x} + \boldsymbol{b}), & \quad \text{(to preserve } \textit{strict} \text{ convexity, } \mathbf{A} \text{ must have full rank)} \\ \max(f, g), & \\ \exp f. & \end{aligned}$$ △

(vi) Let $f : \mathbb{R}^n \to \mathbb{R}$, $g_1, \ldots, g_n : \mathbb{R}^m \to \mathbb{R}$. If f is (strictly) convex, and for each $i = 1, \ldots, n$ either (a) f is (strictly) increasing in its i-th argument and g_i is (strictly) convex, or (b) f is (strictly) decreasing in its i-th argument and g_i is (strictly) concave, then $f \circ \boldsymbol{g}$ is (strictly) convex. △

Example 2.9 (a) The function $f(x) = \ln\left(\frac{1}{1+e^{-x}}\right)$ is strictly concave.

(b) Recall from **Example 1.2**, the logistic regression loss function $l : \mathbb{R}^n \to \mathbb{R}$ is given by

$$l(\boldsymbol{w}) = \sum_{i=1}^{m} -y_i f(\boldsymbol{w}^T \boldsymbol{x_i}) - (1 - y_i) f(-\boldsymbol{w}^T \boldsymbol{x_i}).$$

We computed its Hessian in **Example 1.12**, which is positive semidefinite and hence l is convex. But much easier is to use that f is concave, hence $\boldsymbol{w} \mapsto f(\boldsymbol{w}^T \boldsymbol{x_i})$ is concave (Lemma 2.8(v)), and $-y_i \leq 0$ so that $\sum_{i=1}^m -y_i f(\boldsymbol{w}^T \boldsymbol{x_i})$ is a sum of convex functions, ditto the other term. We have deduced that l is convex, so every stationary point is a global minimum. (In practice it is not generally *strictly* convex.)

Although we will not use it much in this course, it would remiss not to mention perhaps the most useful property that convex functions possess:

Lemma 2.10 (Jensen's Inequality) If f is convex and X any random variable taking values in its domain, then

$$f\big(\mathrm{E}[X]\big) \leq \mathrm{E}[f(X)]. \qquad \triangle$$

The inequality is reversed for concave functions. It is strict if f is strictly convex/concave under the further condition that X is not constant.

2.5 Optimization Tricks

Before we proceed to constrained optimization, we should mention a few methods that can make optimization easier.

Lemma 2.11 If $g : \mathbb{R} \to \mathbb{R}$ is a strictly increasing function, then

$$\arg\min_{x \in F} f(x) = \arg\min_{x \in F} g\big(f(x)\big).$$

We can generalize to functions g that are defined and strictly increasing on at least the image of F under f.

For example, if you are asked to minimize $(x + e^{-2x})^2$ (whether constrained or not), it is probably easier to minimize $x + e^{-2x}$ instead (note: this is okay because $x + e^{-2x}$ is positive, and the square-root function is defined and increasing on positive reals). A commonly-used example is to minimize $\log f$ in place of f: this is particularly helpful if f is a product, because then $\log f$ is a sum and, hopefully, easier to differentiate.

Lemma 2.12 If $g : \mathbb{R} \to \mathbb{R}$ is a 1-1 function, then

$$\arg\min_{x \in F} f(x) = g\Big(\arg\min_{x \in F} f\big(g(x)\big)\Big).$$

For example, the maxima and minima of $\cos(\exp(x))$ will be found when $\exp(x) = n\pi$. We can generalize to functions g that are defined and 1-1 on F.

2.6 Optimization with Equality Constraints (n dimensions)

Now let us turn to optimization in n dimensions where there are equality constraints. First, imagine just one equality constraint. In standard form,

$$\text{minimize } f(\boldsymbol{x}) \text{ subject to } g(\boldsymbol{x}) = 0. \tag{2.4}$$

Sometimes we might be able to re-parameterize so that the constraint is always satisfied, reducing the problem to unconstrained optimization of one fewer variable. (Example: minimize $f(x, y)$ subject to $x^2 + y^2 = 1$ can sometimes be solved by minimizing $f(\cos\theta, \sin\theta)$.) This is rare. Sometimes $g(\boldsymbol{x}) = 0$ has no solutions, in which case the problem is infeasible, or sometimes it is clear that f is unbounded even in the feasible set. (Example: minimize $x + y$ subject to $x - y = 0$.)

Most of the time, we need another trick. There is one that allows us to solve the problem using unconstrained optimization methods. See Figure 2.4, which in two dimensions shows the contours of a function f (some lines where $f(\boldsymbol{x})$ is constant) and the constraint $g(\boldsymbol{x}) = 0$. The key observation is that, when f is maximized or minimized on the constraint, the gradient of f and g must be parallel. That is, $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \lambda \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}$ for some scalar λ. If this were not so, imagine moving 'around' the constraint, following the line $g(\boldsymbol{x}) = 0$: if the contours of f are *not* in the same direction that we are travelling ($\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}$) then we can increase or decrease f by going a small distance forward or back, so f is *not* at a stationary point here on the constraint.

This leads to the **method of Lagrange multipliers**. To solve (2.4), we form the **Lagrangian**

$$\Lambda(\lambda, \boldsymbol{x}) = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$$

and we use the fact that *the stationary points of f on the constraint are all stationary points of Λ*. You should be careful about using the method of Lagrange multipliers mindlessly, because it will not tell you if the problem is unbounded, but it is a powerful method when used carefully[1]. A formal proof of the correctness of this method is difficult; one is provided (for the simplest case of two variables and one constraint) as bonus material in Section 2.8.

[1] There is actually a technical caveat to the method of Lagrange multipliers: a situation in will it may not work. However, this is too rare for us to worry about this year. See the end of Section 2.8 if you are interested.

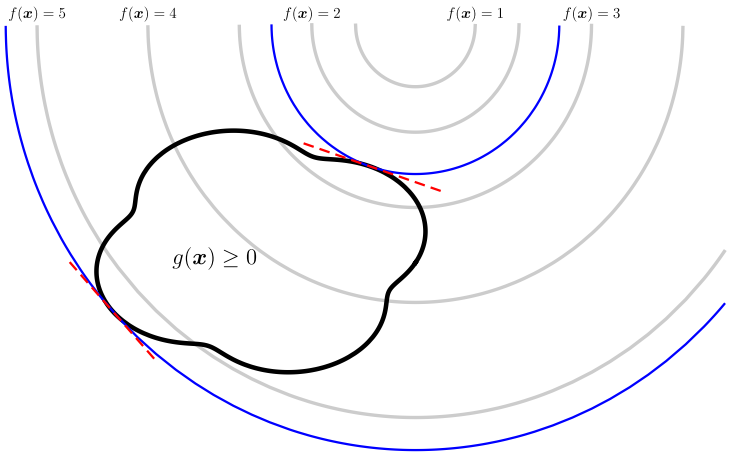


Figure 2.4: The stationary points of $f(\boldsymbol{x})$, subject to the constraint $g(\boldsymbol{x}) = 0$, must occur when the gradients of f and g are parallel.

If f is a function of n variables then the Lagrangian is a function of $n+1$ variables. λ is the **Lagrange multiplier**. We apply the first-order condition which gives the system of $n+1$ equations $\frac{\mathrm{d}\Lambda}{\mathrm{d}(\lambda,\boldsymbol{x})} = \mathbf{0}$, i.e.

$$\begin{aligned} g(\boldsymbol{x}) &= 0, \quad \text{(the constraint again)} \\ \frac{\partial f}{\partial x_1} &= \lambda \frac{\partial g}{\partial x_1}, \\ &\vdots \\ \frac{\partial f}{\partial x_n} &= \lambda \frac{\partial g}{\partial x_n}. \end{aligned}$$

There is nothing to say that this system of equations must be easy to solve! λ is a 'dummy' variable that we do not normally care about (though it does have an economic interpretation as a **shadow price**) so it can be convenient to eliminate it by manipulating the system, but take care not to divide by zero while doing this. Also do not forget that square roots have positive and negative values!

Example 2.13 Find $\arg\min_{(x,y)\in\mathbb{R}^2} xy$, subject to $3x^2 + y^2 = 6$.

The same method works for multiple constraints. For

$$\text{minimize } f(\boldsymbol{x}) \text{ subject to } g_1(\boldsymbol{x}) = 0,\ \ldots,\ g_l(\boldsymbol{x}) = 0,$$

form the Lagrangian with one Lagrange multiplier for each constraint,

$$\Lambda(\lambda_1, \ldots, \lambda_l, \boldsymbol{x}) = f(\boldsymbol{x}) - \lambda_1 g_1(\boldsymbol{x}) - \cdots - \lambda_l g_l(\boldsymbol{x}).$$

The stationary points of the original problem will all be stationary points of the Lagrangian.

The method of Lagrange multipliers is good for finding the stationary points of f subject to constraints. But how do we know whether they are local/global minima or maxima, or merely stationary points of inflection? This is rather difficult: it is not as simple as requiring that the Hessian of Λ is positive definite at the stationary point. Indeed, it can never be, because $\frac{\partial^2 \Lambda}{\partial \lambda^2} = 0$, so that the $\mathbf{H}(\Lambda)$ has a zero in the upper-left entry, and so $\boldsymbol{x}^T \mathbf{H}(\Lambda)\boldsymbol{x} = 0$ when $\boldsymbol{x}$ is the first unit vector.

It *is* the case that $\boldsymbol{x}$, assuming it solves the first-order condition, is a local minimum of f subject to $g(\boldsymbol{x}) = 0$ as long as $\boldsymbol{v}^T \mathbf{H}(\Lambda)\boldsymbol{v} > 0$ for all $\boldsymbol{v}$ where $\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}}^T \boldsymbol{v} = 0$[2] but this is not easy to check.

It is also not sufficient to establish that f is convex, since equality constraints typically describe a non-convex set, unless they are all linear. In practice, it is typically easiest either to check by hand that f is not unbounded, and then evaluate it at the stationary points to find which are the global extrema.

[2] This level of detail is well beyond the first year syllabus.

2.7 Optimization with Inequality Constraints (n dimensions)

Consider optimization subject to *inequality* constraints:

$$\text{minimize } f(\boldsymbol{x}) \text{ subject to } h_1(\boldsymbol{x}) \geq 0,\ \ldots,\ h_m(\boldsymbol{x}) \geq 0.$$

These are harder than optimization problems with equality constraints. First, some terminology. If, at the minimum, $h_i(\boldsymbol{x}) > 0$ we say that constraint i is **slack**. If, at the minimum, $h_i(\boldsymbol{x}) = 0$ we say that the constraint is **tight**.

An elementary observation that can help solve such problems is simply that each constraint is either slack, in which case it can be removed (since it did not affect the solution), or tight, in which case it can be converted into an equality constraint. Once all contraints are either removed or converted to equalities we can apply the Lagrange multiplier method as before. Unfortunately, we do not know in advance which constraints will be tight and which will be slack: in the worst case we might have to try all 2^m options. We should certainly look for reasons why some contraints must be tight (if moving towards the constraint always improves the objective, for example) or intuition that some must be slack (if when tight they lead to clearly non-optimal solutions).

After guessing which constraints are tight and which are slack, we can solve the equality-constrained optimization problem that arises, using the methods of the previous section. If the solution satisfies the slack inequalities, we have found the a minimum for this combination of tight and slack constraints. That does not necessarily mean that we guessed correctly and we do need to try all possibilities for tight and slack constraints that we have not ruled out by other means. Although laborious, this technique should be sufficient for us this year.

The analysis can be made a bit more precise if we return to Figure 2.4. Imagine one inequality constraint $h(\boldsymbol{x}) \geq 0$. Observe that, if $f(\boldsymbol{x})$ is to find a *minimum* on the boundary of the feasible region when $h(\boldsymbol{x}) = 0$, the gradients of f and h have to be parallel and *in the same direction* (otherwise we can move to the interior of $h(\boldsymbol{x}) \geq 0$ while decreasing f. That is, $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mu\frac{\mathrm{d}h}{\mathrm{d}\boldsymbol{x}}$ for $\mu \geq 0$. On the other hand, if the minimum happens in the interior $h(\boldsymbol{x}) > 0$ then $\mu = 0$, because f attains a local maximum regardless of the constraint. This condition can be written as $\mu h(\boldsymbol{x}) = 0$, which is called the **complementary slackness** condition and encodes our intuition that each constraint is either slack ($\mu = 0$) or tight ($h(\boldsymbol{x}) = 0$).

This leads to a mathematical formulation similar in style to Lagrange multipliers: find stationary points of

$$\Lambda(\mu, \boldsymbol{x}) = f(\boldsymbol{x}) - \mu h(\boldsymbol{x})$$

satisfying the constraint plus

$$\mu \geq 0,\ \mu h(\boldsymbol{x}) = 0.$$

Finally, we can combine m inequality constraints with l equality constraints into what are known as the **Karush-Kuhn-Tucker conditions**: to solve

$$\text{minimize } f(\boldsymbol{x}) \text{ subject to } \begin{array}{ll} g_i(\boldsymbol{x}) = 0 & i = 1, \ldots, l, \\ h_i(\boldsymbol{x}) \geq 0 & i = 1, \ldots, m. \end{array}$$

some necessary (but not sufficient) conditions for $\boldsymbol{x}$ to be the minimum are:

△

$$\begin{array}{ll} \dfrac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\Big[f(\boldsymbol{x}) - \displaystyle\sum_i \lambda_i g_i(\boldsymbol{x}) - \sum_i \mu_i h_i(\boldsymbol{x})\Big] = \mathbf{0}, & \\ g_i(\boldsymbol{x}) = 0, & i = 1, \ldots, l, \\ h_i(\boldsymbol{x}) \geq 0, & i = 1, \ldots, m, \\ \mu_i \geq 0, & i = 1, \ldots, m, \\ \mu_i h_i(\boldsymbol{x}) = 0, & i = 1, \ldots, m. \end{array}$$

We will not use this method this year, but it is helpful in many applications including support vector machines and other kinds of machine learning. There must be further conditions to ensure that $\boldsymbol{x}$ locates a global minimum, for example a convexity condition on f and the feasible region. That takes us well beyond this year's syllabus.

Example 2.14 In steganography, we make *changes* to an apparently-innocent cover in order to conceal a secret message within it. In one formulation of the steganography problem we enumerate all possible changes and assign them *costs* $c_1, \ldots, c_n$. Each change will be made with corresponding probability π_i, which conveys up to $H(\pi_i)$ bits of secret information (using an asymptotically-perfect code whose operation need not concern us here). Here, H is the **binary entropy function**,

$$H(x) = -x \log_2 x - (1 - x) \log_2 (1 - x), \quad H(0) = H(1) = 0.$$

The best hiding is one that minimizes the total average cost, while having enough entropy to hide a secret message of some fixed length m bits. That is, find $\pi_1, \ldots, \pi_n$ to

$$\text{minimize } \sum_{i=1}^{n} \pi_i c_i \text{ subject to } \sum_{i=1}^{n} H(\pi_i) \geq m, \quad 0 \leq \pi_i \leq 1.$$

We can argue that the first constraint must be tight, and test the intuition that the others are loose (as a zero or one probability simply wastes a possible change). We find the solution

$$\pi_i = \frac{1}{1 + \exp(\lambda c_i)}$$

for some constant $\lambda > 0$. This is the unique stationary point, the objective is a convex (linear) function and the constraint describes a convex set, so this solution must be the global minimum.

λ can be found by substituting back into the nonlinear equation

$$\sum_{i=1}^{n} H(\pi_i) = m,$$

which is solved numerically using the techniques we will learn in Chapter 5.

2.8 Bonus Mathematics: Proof of the Lagrange Method

This is <u>not</u> part of the syllabus.

Consider the very simplest case of the Lagrange method: equality-constrained optimization with two variables and one constraint:

$$\text{minimize } f(x, y) \text{ subject to } g(x, y) = 0. \tag{2.5}$$

Suppose that this problem has the solution (x_0, y_0). The method of Langrange multipliers rests on the following result.

Claim 2.15 Assume that at least one of $\frac{\partial g}{\partial x}(x_0, y_0) \neq 0$ or $\frac{\partial g}{\partial y}(x_0, y_0) \neq 0$. Let

$$\Lambda(\lambda, x, y) = f(x, y) - \lambda g(x, y).$$

Then there exists some λ_0 such that (λ_0, x_0, y_0) is a stationary point of Λ.

It is a common mistake to believe that the *converse* claim – that every stationary point of Λ is a stationary point of f (subject to g) – proves the method correct. It does not: this would leave the possibility that the solution of (2.5) is not found as a stationary point of the Lagrangian. We need to show that every stationary point of f (subject to g) does show up as a stationary point of Λ. The proof will require a result from real analysis.

Theorem 2.16 (Implicit function theorem for $\mathbb{R}^2$) Let $g : \mathbb{R}^2 \to \mathbb{R}$ be a smooth function, and $(a, b) \in \mathbb{R}^2$ such that $g(a, b) = 0$ and $\frac{\partial g}{\partial y}(a, b) \neq 0$. Then there exists an interval I, strictly containing a, such that there is a unique function $h : I \to \mathbb{R}$ with the properties that

$$h(a) = b \quad \text{and} \quad g(x, h(x)) = 0 \text{ for all } x \in I.$$

Then

$$\frac{\mathrm{d}h}{\mathrm{d}x} = -\frac{\frac{\partial g}{\partial x}(x, h(x))}{\frac{\partial g}{\partial y}(x, h(x))}. \tag{2.6}$$

Although in general one cannot expect to eliminate y by turning $g(x, y) = 0$ into $y = h(x)$, the implicit function theorem says that we can do this locally to x_0, and then tells us how to differentiate h.

Proof of Claim 2.15

Without loss of generality, let us assume that $\frac{\partial g}{\partial y}(x_0, y_0) \neq 0$. (If it is zero, but the other partial derivative is nonzero, then swap x and y.)

We will show that

$$\lambda_0 = \frac{\frac{\partial f}{\partial y}(x_0, y_0)}{\frac{\partial g}{\partial y}(x_0, y_0)} \tag{2.7}$$

has the required property, i.e. that (λ_0, x_0, y_0) is a stationary point of Λ. This happens if all three of Λ's partial derivatives are zero. Two are easy:

$$\frac{\partial\Lambda}{\partial y}(\lambda_0, x_0, y_0) = \frac{\partial f}{\partial y}(x_0, y_0) - \lambda_0 \frac{\partial g}{\partial y}(x_0, y_0) = 0 \text{ by (2.7), and}$$

$$\frac{\partial\Lambda}{\partial \lambda}(\lambda_0, x_0, y_0) = -g(x_0, y_0) = 0.$$

For the third,

$$\frac{\partial\Lambda}{\partial x}(\lambda_0, x_0, y_0) = \frac{\partial f}{\partial x}(\lambda_0, x_0, y_0) - \frac{\frac{\partial f}{\partial y}(x_0, y_0)}{\frac{\partial g}{\partial y}(x_0, y_0)} \frac{\partial g}{\partial x}(x_0, y_0).$$

We show that this zero using the implicit function theorem on g at $(a, b) = (x_0, y_0)$. Since $\frac{\partial g}{\partial y}(x_0, y_0) \neq 0$, it tells us that there is a function h defined around x_0, such that the constraint $g(x, h(x)) = 0$ is always satisfied. Near x_0, then, the constrained optimization problem becomes simply the unconstrained problem

$$\text{minimize } \tilde{f}(x) = f(x, h(x)).$$

Differentiating $\tilde{f}$ gives

$$\begin{aligned} \frac{\mathrm{d}\tilde{f}}{\mathrm{d}x} &= \frac{\partial f}{\partial x}(x, h(x)) + \frac{\partial f}{\partial y}(x, h(x)) \frac{\mathrm{d}h}{\mathrm{d}x}(x) && \{ \text{ partial chain rule from p. 13 } \} \\ &= \frac{\partial f}{\partial x}(x, h(x)) - \frac{\partial f}{\partial y}(x, h(x)) \frac{\frac{\partial g}{\partial x}(x, h(x))}{\frac{\partial g}{\partial y}(x, h(x))}. && \{ \text{ (2.6) } \} \end{aligned}$$

If x_0 is to be a stationary point of $\tilde{f}$ then

$$0 = \frac{\mathrm{d}\tilde{f}}{\mathrm{d}x}(x_0) = \frac{\partial f}{\partial x}(x_0, y_0) - \frac{\partial f}{\partial y}(x_0, y_0) \frac{\frac{\partial g}{\partial x}(x_0, y_0)}{\frac{\partial g}{\partial y}(x_0, y_0)}.$$

This is precisely $\frac{\partial\Lambda}{\partial x}(\lambda_0, x_0, y_0) = 0$. ■

The implicit function theorem, and the claim, generalize to higher dimensions and more constraints. The condition that at least one of $\frac{\partial g}{\partial x}(x_0, y_0) \neq 0$ or $\frac{\partial g}{\partial y}(x_0, y_0) \neq 0$ becomes that the Jacobian of the l constraints $\boldsymbol{g} = (g_1, \ldots, g_l)$ has rank l at the optimum.

You might notice that we did not mention this condition in the chapter. Indeed, it is commonly omitted from textbooks that describe the Lagrange method. Some such condition is, however, necessary! An elementary example where the condition fails, and the method of Lagrange multipliers does not work, is:

$$\text{minimize } x^2 + y \text{ subject to } y^2 = 0.$$

Check for yourself. This illustrates the need to understand the statement of the mathematical theorems underpinning our methods.

Practice Questions

Keep an eye open for simplifications, before you start calculating.

2.1 For what values of a are these matrices positive definite/semidefinite/indefinite/etc.?

$$\mathbf{A} = \begin{pmatrix} 1 & a \\ a & 2 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 2 & 4 \\ 1 & 4 & a \end{pmatrix}, \quad \text{and} \quad \mathbf{C} = \mathbf{B}^2.$$

2.2 '$\boldsymbol{x}$ is the point with non-negative co-ordinates, closest to $\boldsymbol{a}$, no further than 1 unit from $\boldsymbol{b}$, on the plane through the origin whose normal is $\boldsymbol{c}$.' Write this as a constrained optimization problem in standard form.

2.3 Classify the stationary points of f, and find $\arg\min_{x \in F} f(x)$ for each of:

(i) $f(x) = \frac{x}{1+x^2}$, (ii) $g(x) = 3x^4 + 4x^3 - 12x^2 + 1$,

(iii) $h(x) = 3e^{4(x-2)} + 4e^{3(x-2)} - 12e^{2(x-2)} + 3$, (iv) $j(x) = e^{x^3 - 3x}$.

$F = \mathbb{R}$ in each case, except (iv) where $F = [-3, 3]$.

2.4 Find and classify the stationary points of these functions on $\mathbb{R}^2$:

(i) $f(x, y) = x^3 - 3x + y^3 - 3y$, (ii) $g(x, y) = x^4 + 4x^2y^2 - 2x^2 + 2y^2 - 1$,

(iii) $h(x, y) = \frac{x^4}{32} + x^2y^2 - x - y^2$, (iv) $j(x, y) = \frac{x}{1+y^2}$.

2.5 Which are strictly convex, convex, strictly concave, concave, or none of these?

(i) $b(x) = x^4 + x^2$, (ii) $c(x) = \exp(-\exp(x))$, (iii) $^*d(x) = \log(x + 1)$,

(iv) $^*f(x, y) = \frac{y^2}{x}$, (v) $g(x, y) = x^2 - 3xy + y^2$, (vi) $^*h(x, y) = \frac{(y^4+y^2)^2}{\log(x+1)}$,

(vii) $j(\boldsymbol{x}) = (\boldsymbol{x}^T\mathbf{A}\boldsymbol{x})^3$, (viii) $k(\boldsymbol{x}) = \sum_{i=1}^m \max(1 - \boldsymbol{b_i}^T\boldsymbol{x}, 0)$, (ix) $l(\boldsymbol{x}) = \log \prod_{i=1}^m \boldsymbol{x}^T\boldsymbol{b_i}$.

For *, the domain of x is $(0, \infty)$. The symmetric matrix $\mathbf{A}$ is positive semidefinite, and $\boldsymbol{b_1}, \ldots, \boldsymbol{b_m}$ are arbitrary vectors.

2.6 Use the method of Lagrange multipliers to solve these optimization problems:

(i) minimize $x^2 + y^2$ subject to $5x^2 + 6xy + 5y^2 - 4 \geq 0$,

(ii) minimize $x + y + z$ subject to $z - x^2 - y^2 + 1 = 0$, $z + x^2 + y^2 = 0$,

(iii) minimize $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}$ subject to $\boldsymbol{b}^T\boldsymbol{x} = 1$,
(where $\mathbf{A}$ a fixed symmetric positive definite matrix, and $\boldsymbol{b}$ a fixed vector).

Answers to Chapter 2 Practice Questions

2.1 (i) Eigenvalues are $\frac{3\pm\sqrt{4a^2-1}}{2}$. $\mathbf{A}$ positive definite for $|a| < \sqrt{2}$, positive semidefinite for $a = \pm\sqrt{2}$, and indefinite for $|a| > \sqrt{2}$. (ii) Computing the eigenvalues is not fun, but reducing $\mathbf{B}$ to echelon forms gives pivots $(1, 1, a - 26)$. $\mathbf{B}$ positive definite for $a > 26$, positive semidefinite $a = 26$, indefinite for $a < 26$. (iii) Eigenvalues of $\mathbf{B}^2$ are square of eigenvalues of $\mathbf{B}$: $\mathbf{C}$ positive definite unless $a = 26$, when positive semidefinite.

2.2 $\arg\min(\boldsymbol{x} - \boldsymbol{a})^T(\boldsymbol{x} - \boldsymbol{a})$, where $\boldsymbol{c}^T\boldsymbol{x} = 0, 1 - (\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{x} - \boldsymbol{b}) \geq 0, x_1 \geq 0, \ldots, x_n \geq 0$.

2.3 (i) global min at $x = -1$, global max at $x = 1$. (ii) global min at $x = -2$, local max at $x = 0$, local min at $x = 1$. (iii) don't compute $\frac{\mathrm{d}h}{\mathrm{d}x}$, instead notice that $h(x) = g(e^{x-2}) + 2$. Its stationary points will be where $e^{x-2} = -2, 0, 1$, only the last of which is possible at $x = 2$. Furthermore, like g at 1, it must be a local minimum, which is also the global minimum. (iv) a monotone function of $x^3 - 3x$, which has local min at $x = 1$, local max at $x = -1$, and the global minimum on the boundary at $x = -3$.

2.4

	Hessian	local mins	local maxs	saddle pts
(i)	$\begin{pmatrix} 6x & 0 \\ 0 & 6y \end{pmatrix}$	$(1,1)$	$(-1,-1)$	$(1,-1),\ (-1,1)$
(ii)	$\begin{pmatrix} 12x^2+8y^2-4 & 16xy \\ 16xy & 8x^2+4 \end{pmatrix}$	$(\pm1, 0)$	–	$(0,0)$
(iii)	$\begin{pmatrix} 3x^2/8+2y^2 & 4xy \\ 4xy & 2x^2-2 \end{pmatrix}$	$(2,0)$	–	$(1, \pm\sqrt{7}/4)$
(iv)	$\frac{\partial j}{\partial x} = \frac{1}{1+y^2} \neq 0$, so there cannot be any stationary points.			

2.5 (i) Strictly convex as $\frac{\mathrm{d}^2b}{\mathrm{d}x^2} = 12x^2 + 2 > 0$. (ii) $\frac{\mathrm{d}^2c}{\mathrm{d}x^2} = (\exp(x) - 1)\exp(x - \exp(x))$, which crosses the x-axis, so c is neither convex nor concave. (iii) Strictly concave as $\frac{\mathrm{d}^2d}{\mathrm{d}x^2} = -1/x^2 < 0$. (iv) Hessian is $\begin{pmatrix} 2y^2/x^3 & -2y/x^2 \\ -2y/x^2 & 2/x \end{pmatrix}$: pivots 0 and $2y^2/x^3$ so positive semidefinite, hence g is convex (it is not strictly convex). (v) Hessian is $\begin{pmatrix} 2 & -3 \\ -3 & 2 \end{pmatrix}$, indefinite, so g neither convex nor concave. (vi) $h(x, y) = f(b(x), d(y))$ meets the conditions of Lemma 2.8(vi), hence convex (but not strictly). (vii) $x \mapsto x^3$ is convex increasing on $[0, \infty)$, $\boldsymbol{x} \mapsto \boldsymbol{x}^T\mathbf{A}\boldsymbol{x}$ is convex non-negative, Lemma 2.8(vi) applies so j is convex (not strictly). (viii) $x \mapsto 1 - \boldsymbol{b_i}^T\boldsymbol{x}$ is linear hence convex, $x \mapsto 0$ is convex; sums and maximum preserve convexity. (ix) Rewrite as $-\sum_{i=1}^{m} -\log \boldsymbol{x}^T\boldsymbol{b_i}$. $-\log$ is decreasing and convex $\boldsymbol{x} \mapsto \boldsymbol{x}^T\boldsymbol{b_i}$ is concave, Lemma 2.8(vi) applies, l is concave.

2.6 (i) The constraint is either slack or tight. If slack, we simply minimize x^2+y^2, whose solution does not meet the constraint, a contradiction. So it is tight: $\Lambda = x^2 + y^2 - \lambda(5x^2 + 6xy + 5y^2 - 4)$, $\frac{\partial\Lambda}{\partial x} = 2x - \lambda(10x + 6y)$, $\frac{\partial\Lambda}{\partial y} = 2y - \lambda(6x + 10y)$. Divide these equations and rearrange into a quadratic for x/y: the solutions are $x/y = \pm1$. Substituting $x = y$ in the constraint gives $x = y = \pm\frac{1}{2}$ and $f(x, y) = \frac{1}{2}$. Substituting $x = -y$ in the constraint gives $x = -y = \pm1$ and $f(x, y) = 2$. Two global minima are therefore found at $x = y = \pm\frac{1}{2}$.
(ii) $\Lambda = x + y + z - \lambda(z - x^2 - y^2 + 1) - \mu(z + x^2 + y^2)$. From $\frac{\partial\Lambda}{\partial x} = 1 + 2\lambda x - 2\mu x = 0$ and $\frac{\partial\Lambda}{\partial y} = 1 + 2\lambda y - 2\mu y = 0$ it follows that $x = y$. Adding the two constraints tells us that $2z + 1 = 0$, hence $z = -\frac{1}{2}$. Substituting $x = y$ into either constraints gives $x = y = \pm\frac{1}{2}$. At $(\frac{1}{2}, \frac{1}{2}, -\frac{1}{2})$, $f = \frac{1}{2}$; at $(-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$, $f = -\frac{3}{2}$ and this is the global minimum.
(iii) $\Lambda = \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} - \lambda(\boldsymbol{b}^T\boldsymbol{x} - 1)$. $\frac{\partial\Lambda}{\partial\boldsymbol{x}} = 2\mathbf{A}\boldsymbol{x} - \lambda\boldsymbol{b} = 0$ if $\boldsymbol{x} = \frac{\lambda}{2}\mathbf{A}^{-1}\boldsymbol{b}$. Substituting into the constraint gives $\lambda = 2/\boldsymbol{b}^T\mathbf{A}^{-1}\boldsymbol{b}$ and hence $\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{b}/(\boldsymbol{b}^T\mathbf{A}^{-1}\boldsymbol{b})$. ($\mathbf{A}^{-1}$ exists because $\mathbf{A}$ is positive definite, so all its eigenvalues are positive, hence its determinant is positive. How do we know it is a minimum? The objective is strictly convex, so the only stationary point is the global minimum.)

Chapter 3

Algorithms for Numerical Integration

Many mathematicians are first introduced to numerical algorithms through integration. Finding the exact area under a curve can often be extremely hard or impossible, but it is a problem that is amenable to approximation. We first look at the problem in one dimension (integration of a univariate function) and then move to integration over regions of a multivariate scalar function. This will illustrate a problem known as the **curse of dimensionality**.

Numerical algorithms tend to come in two parts. First, make some approximation to the problem, and solve the approximate problem exactly. We will use various approximation methods including Taylor's theorem from Chapter 1. Second, use mathematics to perform **error analysis**: an upper bound on the difference between our approximation and the true answer. Again, Taylor's theorem will be key here. We can compare different numerical algorithms by the amount of calculation they need to do, and the guarantees on the maximum error.

In high dimensions, it is *randomized* algorithms that tend to work best. In such a case there are no guarantees, but we can make probabilistic statements about the error.

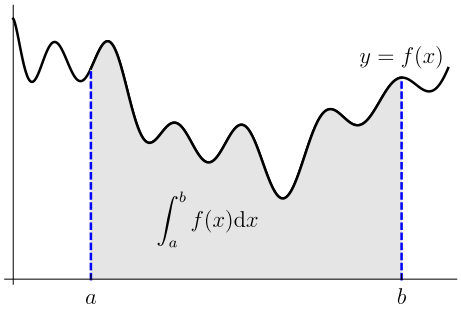


Figure 3.1: Integration as area under the curve.

3.1 Integration

The **integral of f from a to b**,

$$\int_a^b f(x)\,\mathrm{d}x$$

is defined (in a mathematically rigorous way that need not concern us for now) to be the area bounded by the x-axis, the lines $x = a$ and $x = b$, and the graph of $y = f(x)$, with the interpretation of area *below* the x-axis as negative and above as positive. See Figure 3.1. In such an integral the function $f(x)$ is known as the **integrand**.

As you probably learned in school, the elementary way to compute this area is to find an **antiderivative**, a function $F : \mathbb{R} \to \mathbb{R}$ with the property that

$$\frac{\mathrm{d}F}{\mathrm{d}x} = f(x). \tag{3.1}$$

An antiderivative is also called an **indefinite integral**, and written without limits as $\int f(x)\,\mathrm{d}x$: the conventional '$+\,c$' normally added to indefinite integrals reflects the fact that any constant added to $F(x)$ still satisfies (3.1). If an antiderivative can be found, then

$$\int_a^b f(x)\,\mathrm{d}x = \big[F(x)\big]_a^b = F(b) - F(a).$$

(Quite *why* the area under a curve is related to an antiderivative is another matter for mathematicians to prove: it is one of the theorems called the **fundamental theorem of calculus**.)

As a simple example, the area under $y = \cos x$ (whose antiderivative is $\sin x$) between $x = 1$ and $x = 3$ is $\sin 3 - \sin 1 \approx -0.700$, a negative answer because most of the curve lies below the x-axis.

In this course we will only consider integration on finite intervals, rather than integrals like $\int_{-\infty}^{\infty} f(x)\,\mathrm{d}x$. They require more advanced techniques.

Unfortunately, finding an antiderivative is not nearly as simple as finding a derivative. You may have learned some tricks such as integration by substitution or integration by parts, but in general there is no 'product rule' or 'chain rule' for integration.

Worse than simply difficult, in some cases it is known to be impossible: although every continuous function *has* an antiderivative, some cannot ever be expressed in terms of elementary functions. (The **elementary functions** are compositions of polynomials, trigonometric functions, exponentials, logarithms, and roots of polynomials.) For example, none of e^{-x^2}, $\frac{e^x}{x}$, $\ln\ln x$, $\frac{\sin x}{x}$ has an elementary antiderivative.

3.2 Numerical Methods for Integration (1 dimension)

If we wish to find the area under a curve, but cannot find an antiderivative that we can evaluate, we must use an approximation. We approximate the function that we cannot integrate by a function that we can, and then bound how much error this introduces. In the rest of this section, we suppose that we want to approximate

$$I = \int_a^b f(x)\,\mathrm{d}x$$

for some function $f : [a, b] \to \mathbb{R}$. The region $[a, b]$ is called a **strip**, and later we will combine many strips to integrate a function over a larger range. Throughout this section we will also write

$$m = \frac{a+b}{2}$$

for the *midpoint* of the strip.

3.2.1 A Method Based on Taylor's Theorem

A natural idea is to approximate $f(x)$ by a Taylor polynomial around $x = m$. The zero-order (constant) approximation is simply $\hat{f}(x) = f(m)$, and its integral is called the

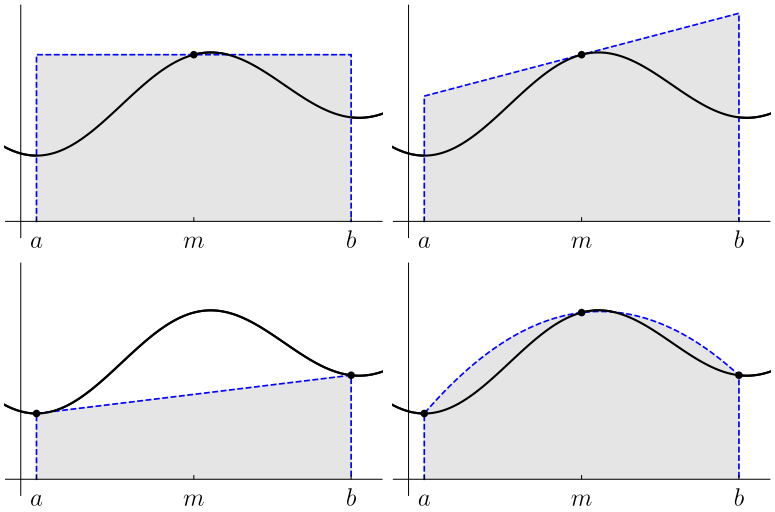


Figure 3.2: Methods of approximating an integral: Midpoint rule (3.2), the linear Taylor approximation around $x = m$ which gives the same answer as the Midpoint rule (and hence is not useful), the Trapezium rule (3.3), and Simpson's rule (3.4).

Midpoint rule since it approximates f by its value at the midpoint of the strip $[a, b]$. Its integral is

$$M_1[f, a, b] = \int_a^b f(m)\,\mathrm{d}x = \big[xf(m)\big]_a^b = (b-a)f(m), \tag{3.2}$$

the subscript 1 indicating that this is integration over a single strip.

You can check for yourself that the first-order (linear) Taylor polynomial leads to exactly the same estimate: the linear term does not contribute to the integral.

Higher-order Taylor expansions are possible: all the odd powers cancel, as did the linear term, but the even terms persist. One can also perform the Taylor approximation around a different point: $x = a$ or $x = b$, perhaps. These are sometimes called the **Left-hand rule** and the **Right-hand rule**, respectively, but they are only useful in special circumstances.

3.2.2 A Method Based on Interpolating Polynomials

More popular methods for numerical integration are based on **interpolating polynomials**. That is, approximating f not by its Taylor polynomial (which, recall, has the

value and first few derivatives that agree with f at the single point $x = m$) but by a polynomial that agrees with the value of f at some set of points across the strip.

The simplest is a linear approximation that agrees with f at the endpoints of the interval: a linear $\hat{f}$ such that $\hat{f}(a) = f(a)$ and $\hat{f}(b) = f(b)$. It is easy to see that this is $\hat{f}(x) = \frac{(b-x)f(a)+(x-a)f(b)}{(b-a)}$. The approximation to I is therefore

$$\begin{aligned} T_1[f,a,b] &= \int_a^b \frac{(b-x)f(a)+(x-a)f(b)}{b-a}\,\mathrm{d}x \\ &= \frac{1}{b-a}\left[-\frac{(b-x)^2}{2}f(a)+\frac{(x-a)^2}{2}f(b)\right]_a^b \\ &= \frac{b-a}{2}\big(f(a)+f(b)\big). \end{aligned} \tag{3.3}$$

This is the **Trapezium** or **Trapezoid rule** which you may have seen in school. Despite its frequent occurrence in school mathematics, it is rarely a good choice as its error is generally worse than the Midpoint rule: compare equations (3.6) and (3.7).

Instead we can try a quadratic approximation that agrees with f at the endpoints and midpoint of the interval. Finding such a quadratic function can be messy; perhaps easiest is to write

$$\hat{f}(x) = \alpha(x-m)(x-b) + \beta(x-a)(x-b) + \gamma(x-a)(x-m)$$

and then:

$$\begin{array}{rcllcl} \hat{f}(a) & = & f(a) & \text{implies} & \alpha & = & \frac{2}{(b-a)^2}f(a), \\ \hat{f}(m) & = & f(m) & \text{implies} & \beta & = & -\frac{4}{(b-a)^2}f(m), \\ \hat{f}(b) & = & f(b) & \text{implies} & \gamma & = & \frac{2}{(b-a)^2}f(b). \end{array}$$

The approximation to I, using $\int_a^b (x-a)(x-b)\,\mathrm{d}x = (a-b)^3/6$, is

$$\begin{aligned} S_2[f,a,b] &= \int_a^b \frac{2f(a)(x-m)(x-b) - 4f(m)(x-a)(x-b) + 2f(b)(x-a)(x-m)}{(b-a)^2}\,\mathrm{d}x \\ &= \frac{b-a}{6}\big(f(a)+4f(m)+f(b)\big). \end{aligned} \tag{3.4}$$

This is called **Simpson's rule** and we have written it S_2 to comply with convention, which regards our 'strip' of length $b-a$ as if they were two strips of length $\frac{b-a}{2}$. This

makes some sense, as we shall see in Section 3.4, since we had to evaluate the integrand at every multiple of $\frac{b-a}{2}$.

The approximations made by the Midpoint rule (3.2), the Trapezium rule (3.3), and Simpson's rule (3.4), are all displayed in Figure 3.2.

We could go further with polynomial approximation, for example to a fourth-order polynomial that matches f at $x = a, \frac{3a+b}{4}, m, \frac{a+3b}{4}, b$. This is called **Boole's rule** and it turns out to be

$$B_4[f,a,b] = \tfrac{b-a}{90}\big(7f(a) + 32f(\tfrac{3a+b}{4}) + 12f(m) + 32f(\tfrac{a+3b}{4}) + 7f(b)\big),$$

but it is no longer widely used.

3.3 Error Analysis

The second part of designing a numerical method is to analyze its error. We will write the error of the Midpoint, Trapezium, and Simpson's rules for f on $[a,b]$ as

$$\mathrm{err}(M_1)[f,a,b] = M_1[f,a,b] - I,$$

and so on, and want to find upper and lower bounds on these errors. Only the proof for the Midpoint rule is within our syllabus, as well as the statements (but not proof) of the bounds for the other methods. The error bounds come from Taylor's theorem, although sometimes applied in surprising ways. In research, finding the expansion (Taylor or otherwise) that gives the tightest error bound is something of a dark art.

In this section we will assume that f can be differentiated at least five times. If not, some of the results may not hold.

The Midpoint rule: f is approximated by $\hat{f}(x) = f(m)$, but Taylor's theorem *with second order remainder* says that

$$f(x) = \hat{f}(x) + (x-m)\frac{\mathrm{d}f}{\mathrm{d}x}(m) + \frac{(x-m)^2}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)$$

for some $\xi \in (a,b)$. Rearranging and bounding the second derivative,

$$\frac{(x-m)^2}{2}\underline{D_2} \le f(x) - \hat{f}(x) - (x-m)\frac{\mathrm{d}f}{\mathrm{d}x}(m) \le \frac{(x-m)^2}{2}\overline{D_2}, \tag{3.5}$$

where $\underline{D_k} = \min_{x\in(a,b)} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}$ and $\overline{D_k} = \max_{x\in(a,b)} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}$. Since $\int_a^b (x-m)\,\mathrm{d}x = 0$ and $\int_a^b \frac{(x-m)^2}{2}\,\mathrm{d}x = \frac{(b-a)^3}{24}$, rearranging (3.5) gives

$$-\frac{1}{24}(b-a)^3\overline{D_2} \le \mathrm{err}(M_1)[f,a,b] = \int_a^b \hat{f}(x) - f(x)\,\mathrm{d}x \le -\frac{1}{24}(b-a)^3\underline{D_2}. \tag{3.6}$$

As we would expect, the Midpoint rule is generally more accurate for small strips. The error is no worse than $O((b-a)^3)$, and the hidden constant depends on the *second derivative* of the integrand. If the case when $\underline{D_2} \ge 0$ (which happens when f is convex) we can be certain that the error is negative (an underestimate), and conversely the error is positive if $\overline{D_2} \le 0$.

The Trapezium rule: Error analysis of the single-strip Trapezium rule is not on our syllabus. The classic way to analyze this error is to use the **interpolation error bound** for polynomial interpolation[1]. We give an alternative proof as bonus material in Section 3.9. The result is

$$\frac{1}{12}(b-a)^3\underline{D_2} \le \mathrm{err}(T_1)[f,a,b] \le \frac{1}{12}(b-a)^3\overline{D_2}. \tag{3.7}$$

Like the Midpoint rule, the error of the Trapezium rule is $O((b-a)^3)$ and its bounds depend on the second derivative of the integrand, but the sign is opposite that of the Midpoint rule: for convex functions it will overestimate and for concave functions underestimate. Also note that the error bound is a factor of 2 larger than for the Midpoint rule. That is not to say that the Trapezium rule is always twice as inaccurate as the Midpoint rule (but it often is).

Simpson's rule: Error analysis of single-strip Simpson's rule is notoriously fiddly and also not on our syllabus. We give a simplified method in Section 3.9. The result is

$$\frac{1}{2880}(b-a)^5\underline{D_4} \le \mathrm{err}(S_2)[f,a,b] \le \frac{1}{2880}(b-a)^5\overline{D_4}. \tag{3.8}$$

Simpson's rule has error that is $O((b-a)^5)$: impressive for such a simple formula. The hidden constant depends on the fourth derivative of the integrand.

[1] An accessible source is E. Süli and D. Mayers, *An Introduction to Numerical Analysis* (CUP, 2006), Thm 6.2.

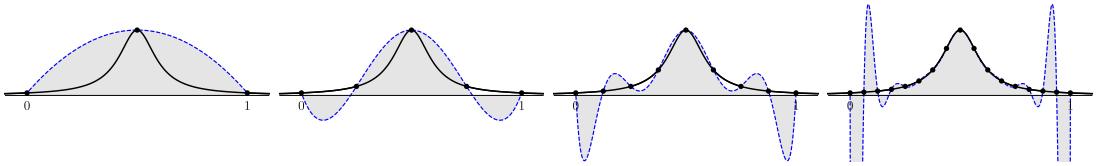


Figure 3.3: Failure of polynomial interpolation. Interpolation of (3.9) on $[0, 1]$ by (from left to right) degree 2, 4, 8, and 16 polynomials. The integrals of the polynomials are, respectively to 4 decimal places: $0.6795, 0.2374, 0.1500, -0.6241$. This is not approaching the true integral 0.27468.

Summary of the single-strip integration rules, and their error bounds, for approximating $\int_a^b f(x)\,\mathrm{d}x$:

Rule	Approximation	Error bounded by
Midpoint	$(b-a)f(m)$	$-\frac{1}{24}(b-a)^3\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$
Trapezium	$\frac{b-a}{2}\big(f(a)+f(b)\big)$	$\frac{1}{12}(b-a)^3\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$
Simpson's	$\frac{b-a}{6}\big(f(a)+4f(m)+f(b)\big)$	$\frac{1}{2880}(b-a)^5\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}$
for interest only:		
Boole's	$\frac{b-a}{90}\big(7f(a)+32f\big(\frac{3a+b}{4}\big)+12f(m)+32f\big(\frac{a+3b}{4}\big)+7f(b)\big)$	$-\frac{1}{1935360}(b-a)^7\frac{\mathrm{d}^6 f}{\mathrm{d}x^6}$

3.4 Composite Methods

How can we improve the accuracy? Following the discussion in Section 3.2, we might try higher-order Taylor expansions for f, or interpolating polynomials of higher degree. These are complicated, much more so than the simple rules we derived in Section 3.2, and sometimes offer no improvement. In the case of higher-order Taylor expansions, it may not improve accuracy if the Taylor series for f fails to converge or converges to the wrong answer. Or f may fail to be differentiable enough times for the error analysis to be correct. In the case of higher-order polynomial interpolation, there are examples of perfectly sensible functions where polynomial interpolation goes haywire as the order increases. A classic example is the **Runge function**, one version of which is

$$f(x) = \frac{1}{1+25(2x-1)^2}. \tag{3.9}$$

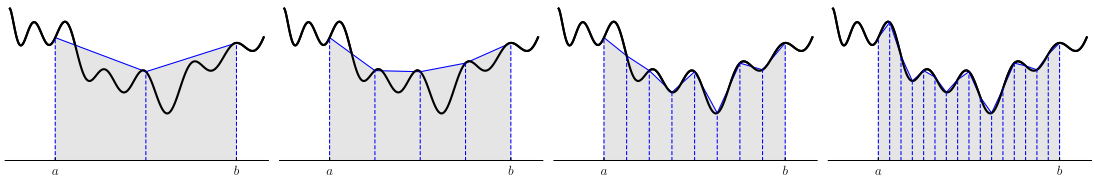


Figure 3.4: Composite integration rules use piecewise, rather than global, approximations.

In Figure 3.3 we show polynomial interpolation, on the interval $[0, 1]$, for the Runge function using order-2, order-4, order-8, and order-16 polynomials. Although they become more exact on the centre of the interval, they become less accurate towards the edges, and the integral of the interpolating polynomials does not approach the integral of f.

Higher-order interpolation is certainly not useless, either for integration or other purposes. The Runge phenomenon is avoidable, if we use interpolation points that are not equally spaced across the interval (see for example Chebyshev polynomials). It is also commonly believed that polynomial interpolation is unstable: that rounding errors in the calculations can cause large errors in the answer. This is also avoidable, with carefully-chosen algorithms. For more information, albeit at an advanced level, see the entertaining article *Six Myths of Polynomial Interpolation and Quadrature* by Nick Trefethen[2].

An alternative, though, is to divide the interval of integration into small strips, and approximate each strip separately. See Figure 3.4 for an illustration of how this would work with the Trapezium rule. By making the individual strips very small we only need the approximations to be valid in a small range each. As well as avoiding the worst of the problems with high-order Taylor expansions and polynomial interpolation, it allows us to use the extremely simple methods of Section 3.2 to achieve high accuracy. Since for a strip of width w the error of the Midpoint and Trapezium rule is $O(w^3)$, and of Simpson's rule is $O(w^5)$, accuracy should increase rapidly as the strips become small.

So we divide the integral $I = \int_a^b f(x)\,\mathrm{d}x$ into n strips:

$$I = \int_a^b f(x)\,\mathrm{d}x = \int_{x_0}^{x_1} f(x)\,\mathrm{d}x + \int_{x_1}^{x_2} f(x)\,\mathrm{d}x + \cdots + \int_{x_{n-1}}^{x_n} f(x)\,\mathrm{d}x. \tag{3.10}$$

[2] `https://people.maths.ox.ac.uk/trefethen/mythspaper.pdf`

Using (3.10) with n equal-sized strips so that $x_i = a + i\big(\frac{b-a}{n}\big)$, we approximate $\int_a^b f(x)\,\mathrm{d}x$ by the **composite Midpoint rule**:

$$M_n[f,a,b] = \sum_{i=1}^{n} M_1[f,x_{i-1},x_i] = \frac{b-a}{n}\Big(f\big(\tfrac{x_0+x_1}{2}\big) + f\big(\tfrac{x_1+x_2}{2}\big) + \cdots + f\big(\tfrac{x_{n-1}+x_n}{2}\big)\Big).$$

We can also bound the error, using (3.6): since $\max_{x\in(x_{i-1},x_i)} \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \le \overline{D_2}$, where $\overline{D_2} = \max_{x\in(a,b)} \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$, we have

△ (know that it is $O(n^{-2})$ and depends on $-\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$)

$$-\frac{(b-a)^3}{24n^2}\overline{D_2} \le \mathrm{err}(M_n)[f,a,b] = \sum_{i=1}^{n} \mathrm{err}(M_1)[f,x_{i-1},x_i] \le -\frac{(b-a)^3}{24n^2}\underline{D_2}. \tag{3.11}$$

The same calculations apply for the Trapezium rule. The **composite Trapezium rule** with n strips is:

$$T_n[f,a,b] = \sum_{i=1}^{n} T_1[f,x_{i-1},x_i] = \frac{b-a}{2n}\Big(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\Big)$$

and the error bound is

△

$$\frac{(b-a)^3}{12n^2}\underline{D_2} \le \mathrm{err}(T_n)[f,a,b] = \sum_{i=1}^{n} \mathrm{err}(T_1)[f,x_{i-1},x_i] \le \frac{(b-a)^3}{12n^2}\overline{D_2}. \tag{3.12}$$

Simpson's rule with what is conventionally called n strips requires dividing into $n/2$ intervals, and applying S_2 to each (n must be even to apply Simpson's rule in this way). It approximates $\int_a^b f(x)\,\mathrm{d}x$ by the **composite Simpson's rule**:

$$\begin{aligned} S_n[f,a,b] &= \sum_{i=1}^{n/2} S_2[f,x_{2i-2},x_{2i}] \\ &= \frac{b-a}{3n}\Big(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 4f(x_{n-1}) + f(x_n)\Big). \end{aligned}$$

The error bound is

△ (know that it is $O(n^{-4})$ and depends on $\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}$)

$$\frac{(b-a)^5}{180n^4}\underline{D_4} \le \mathrm{err}(S_n)[f,a,b] = \sum_{i=1}^{n/2} \mathrm{err}(S_2)[f,x_{2i-2},x_{2i}] \le \frac{(b-a)^5}{180n^4}\overline{D_4}. \tag{3.13}$$

Example 3.1 The integral

$$\int_0^{12} \frac{\sin x}{x}\,\mathrm{d}x$$

provides an illustration of the composite methods and their error bounds. The integrand has no elementary antiderivative, so numerical methods are necessary. The value of the integral is approximately

1.504971241526373370527148532118162608929215879356371424570210244429143 57.

It is routine to show that

$$\max_{x\in[0,12]}\left|\tfrac{\mathrm{d}^2(\sin x/x)}{\mathrm{d}x^2}\right| = \tfrac{1}{3} \quad \text{and} \quad \max_{x\in[0,12]}\left|\tfrac{\mathrm{d}^4(\sin x/x)}{\mathrm{d}x^4}\right| = \tfrac{1}{5},$$

and estimating the integral using the composite Midpoint or Trapezium rule exhibits error proportional to n^{-2} or n^{-4}, as predicted by the error analysis, but several multiples smaller than the worst-case bounds that we have derived.

Observe that all of the composite rules are of the form

$$\frac{(b-a)}{\sum w_i}\Big(w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n)\Big).$$

These are collectively known as **Newton-Cotes formulae**, and there are many such. The more complex Newton-Coates formulae used to be more popular than they are now, but with modern computing power it is easier to apply the simplest rules (Midpoint or Simpson's) with more strips, rather than to make the interpolation more complex.

What do the error bounds mean? First, let us measure the amount of 'work' required to compute $M_n[f,a,b]$, $T_n[f,a,b]$, and $S_n[f,a,b]$. In the first case, $O(n)$ arithmetic operations plus n evaluations of the function f. In the others, the same but $n+1$ evaluations of the function, which is practically the same thing. We assume that the expensive part of this operation is evaluating f. (If f is quicker to evaluate than some scalar additions, and divisions by constants, it can probably be integrated exactly!) Thus all composite methods do approximately the same amount of work, n evaluations of f. This is why the composite Simpson's rule conventionally counts $n/2$ intervals as n 'strips'.

Now how does the approximation error depend on n? In the case of the Midpoint or Trapezium rule, $O(n^{-2})$. That is, to reduce the error by a factor of about 10 (one more decimal place of accuracy) we should increase n by a factor of $\sqrt{10}$. In the case of Simpson's rule, the error decreases rapidly, $O(n^{-4})$, and it even has a small hidden constant. To reduce the error by a factor of about 10 only requires about 1.8 times as much work. Considering its extreme simplicity, Simpson's rule is tremendously good value.

As with the single-strip case, the error bound also depends on the second or fourth derivative of f. This forms part of the hidden constant (along with 1/24, 1/12, 1/180) in the $O(n^{-2})$ or $O(n^{-4})$. Remember that these are *bounds* on the error, and sometimes the error will be a lot smaller. For example, if the second (or fourth) derivative is only large in one of the strips, and very small elsewhere, the error bound will be very loose.

Indeed, this suggests one possible improvement to the composite rules: unequal-width strips. There are **adaptive** methods that start with one strip, then recursively subdivide until an error tolerance is achieved: much more sophisticated than the methods we have described, but a bit advanced for us yet.

Summary of the composite integration rules, and their error bounds, for approximating $\int_a^b f(x)\,\mathrm{d}x$, with $x_i = a + i\frac{b-a}{n}$:

Rule	Approximation	Error bounded by
Midpoint	$\frac{b-a}{n}\big(f\big(\frac{x_0+x_1}{2}\big) + f\big(\frac{x_1+x_2}{2}\big) + \cdots + f\big(\frac{x_{n-1}+x_n}{2}\big)\big)$	$-\frac{(b-a)^3}{24n^2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$
Trapezium	$\frac{b-a}{2n}\big(f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\big)$	$\frac{(b-a)^3}{12n^2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$
Simpson's	$\frac{b-a}{3n}\big(f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 4f(x_{n-1}) + f(x_n)\big)$	$\frac{(b-a)^5}{180n^4}\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}$

3.5 Integration in d dimensions

Consider a function $f : \mathbb{R}^2 \to \mathbb{R}$. Integration of f calculates the volume underneath the surface, as depicted in Figure 3.5. The notation for this is

$$\int_R f(x,y)\,\mathrm{d}(x,y).$$

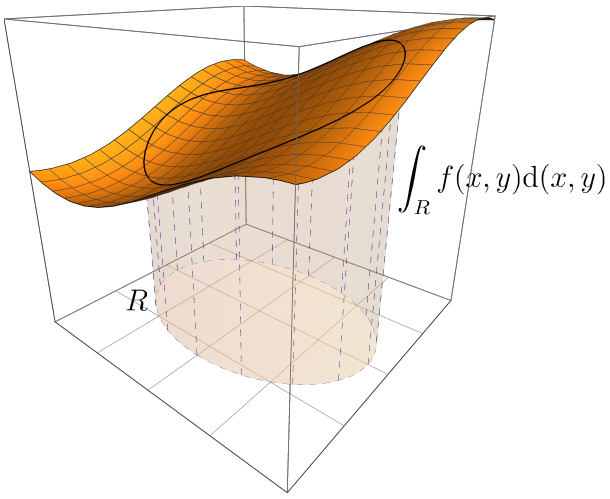


Figure 3.5: Integration in higher dimensions. $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ represents the volume of the region between $f(\boldsymbol{x})$ and $\boldsymbol{x} = 0$ inside the region R.

Such integrals can sometimes be evaluated exactly by converting them into nested integrals over univariate functions:

$$\int_R f(x,y)\,\mathrm{d}(x,y) = \int_{x_0}^{x_1}\int_{y_0}^{y_1} f(x,y)\,\mathrm{d}y\,\mathrm{d}x, \tag{3.14}$$

providing that all the integrals are finite, which is a result called **Fubini's theorem**. Note that the inner integral treats x as a constant, a bit like partial derivatives do. However, one has to identify the limits (x_0, x_1) and (y_0, y_1) (the latter can be a function of x) such that the region $R \subset \mathbb{R}^2$ is covered: easy in the case of rectangles, triangles, and circles; difficult for more general shapes. The order of integration does not matter and you can choose whether to integrate first with respect to y, then x (as in (3.14)), or the other way around. Sometimes it is more convenient to describe the limits one way than the other.

The same concept generalizes to multivariate functions in d dimensions,

$$\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},$$

for $R \subseteq \mathbb{R}^d$, which can sometimes be computed by converting to d nested integrals, each over one dimension of R. The one-dimensional integration trick you might know as

'integration by substitution' applies in higher dimensions (and uses the Jacobian of the map), but there are many examples of functions where the antiderivatives simply cannot be found. Then we need numerical methods.

3.6 Numerical Methods for Integration (d dimensions)

The methods of Section 3.2 can be adapted for higher dimensions.

For simplicity's sake, consider just the (composite) Midpoint rule, and suppose we wish to evaluate

$$I = \int_R f(x,y)\,\mathrm{d}(x,y)$$

where R is the rectangle $[0,1]\times[0,1]$. Fubini's theorem and then the Midpoint rule, with n strips, on the *outer* integral gives

$$I = \int_0^1 \int_0^1 f(x,y)\,\mathrm{d}y\,\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n} \int_0^1 f\big(\tfrac{2i-1}{2n}, y\big)\,\mathrm{d}y. \tag{3.15}$$

Then we can apply the Midpoint rule again to each inner integral, perhaps again with n strips (although there could be different numbers of strips in each dimension), giving

$$I \approx \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} f\big(\tfrac{2i-1}{2n}, \tfrac{2j-1}{2n}\big).$$

Notice that this is just evaluating f on the midpoint of the squares of a $n\times n$ grid over the region $R=[0,1]\times[0,1]$, and taking the average value.

One can extend the same method to higher dimensions in the same way. There are also versions of the Trapezium rule and Simpson's rule. In the case of 2 dimensions, the latter evaluates f on a $(n+1)\times(n+1)$ grid over R, and then multiplies the results by constants

given by the $(n+1) \times (n+1)$ matrix

$$\frac{1}{9n^2}\begin{pmatrix} 1 & 4 & 2 & 4 & 2 & \cdots & 2 & 4 & 1 \\ 4 & 16 & 8 & 16 & 8 & \cdots & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & 4 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & 8 & \cdots & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & 4 & \cdots & 4 & 8 & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2 & 8 & 4 & 8 & 4 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & 8 & \cdots & 8 & 16 & 4 \\ 1 & 4 & 2 & 4 & 2 & \cdots & 2 & 4 & 1 \end{pmatrix}.$$

Seemingly a pretty simple calculation, even generalized to d dimensions, but there is a hitch.

Formal error analysis of the d-dimensional versions of Midpoint, Trapezium, and Simpson's rules is tricky, but an interesting thought experiment is to imagine that all the inner integrals are done with zero error, then consider just the effect of the relevant rule on the outer integral (3.15). Whatever answer this gives, it can only be an underestimate for the error when the inner integrals are also subject to approximation.

There will be some unpleasant dependence on partial derivatives of f, but that only contributes a constant factor. Whatever else happens, from our analysis of the one-dimensional cases we can say that the error decreases at best as $O(n^{-2})$ in the case of the Midpoint and Trapezium rules, and $O(n^{-4})$ for Simpson's rule. In fact, it can be shown that the error of the d-dimensional Midpoint and Trapezium rules are $O(dn^{-2})$ and Simpson's rule is $O(dn^{-4})$. This feels like fast convergence.

But consider the amount of work required, to apply the numerical methods. In d dimensions we need to evaluate the function in $N = n^d$ (or $N = (n+1)^d$, which is practically the same thing) locations, because this is the number of vertices in the d-dimensional grid of side n or $n+1$. This is a phenomenon called the **curse of dimensionality**, which occurs in numerical algorithms and machine learning in high dimensions. The amount of 'work' increases exponentially with the dimension.

It means that the error of the algorithm is only $O(N^{-2/d})$ or $O(N^{-4/d})$, if N represents the amount of 'work' performed. So even Simpson's rule, which is wonderful in 1 dimension, converges as $O(N^{-1/3})$ when attempting a 12-dimensional integral: to reduce the error

by a factor of 10 we need around 1000 times as many function evaluations. For a lot of decimal places of accuracy (all depending on the hidden constants, of course), the work becomes infeasible. And when we come to integration over hundreds of dimensions, which is quite commonplace in machine learning and Bayesian inference, we will need an alternative method.

3.7 Monte Carlo Integration

A completely different approach to numerical integration is found in **Monte Carlo methods**. These are randomized algorithms that give random answers[3] but where the random answer is likely close to the correct answer.

The random part of Monte Carlo integration comes from choosing where to evaluate the integrand $f(\boldsymbol{x})$. Instead of calculating $f(\boldsymbol{x})$ over points in a regular d-dimensional grid, and then returning a weighted or (in the case of the Midpoint rule) an unweighted average of the answers, we calculate $f(\boldsymbol{x_i})$ for *randomly generated* points $\boldsymbol{x_i}$, and then return a weighted or unweighted average of the answers. What is surprising is how powerful this method is, in high dimensions.

Consider first a one-dimensional integration problem, $\int_a^b f(x)\,\mathrm{d}x$. Suppose that we pick a random number X, uniformly distributed on $[a, b]$. Its probability density function (pdf) is $p(x) = \frac{1}{b-a}$ for $x \in [a, b]$, 0 otherwise. Then, using a property of expectation of a function of a random variable,

$$\mathrm{E}[f(X)] = \int f(x)p(x)\,\mathrm{d}x = \frac{1}{b-a}\int_a^b f(x)\,\mathrm{d}x.$$

This suggests a way to approximate $\int_a^b f(x)\,\mathrm{d}x$: pick a large number of continuous random variables $X_1, \ldots, X_N$, all independently uniformly distributed on $[a, b]$; the Monte Carlo estimate for $\int_a^b f(x)\,\mathrm{d}x$ is

$$MC_N[f, a, b] = (b-a)\frac{1}{N}\sum_i f(X_i).$$

[3] in contrast to **Las Vegas methods**, which are randomized algorithms that give nonrandom answers, but have random running time.

This has a natural intuition: the area under f in an interval $[a, b]$ is the average value of f times the width of the interval.

Note that $MC_N[f, a, b]$ is a random variable: it will give a different answer every time, if the X_i are regenerated randomly. As with all Monte Carlo algorithms, we seek a proof that it is *likely* to be close to the true answer.

The method applies equally well in higher dimensions. For a multivariate function $f : \mathbb{R}^d \to \mathbb{R}$, we can approximate $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ by generating random vectors $\boldsymbol{X_1}, \ldots, \boldsymbol{X_N}$ uniformly on the region R, and then

$$MC_N[f, R] = A(R)\frac{1}{N}\sum_i f(\boldsymbol{X_i}), \tag{3.16}$$

where $A(R)$ denotes the area or volume of the region R (note that the pdf of $\boldsymbol{X_i}$ is the constant $\frac{1}{A(R)}$).

Is this method any use? Here is a sequence of increasingly precise things that we can say about Monte Carlo integration.

Lemma 3.2 Assuming that f is well-behaved[4],

(i) $\mathrm{E}\big[MC_N[f, R]\big] = \displaystyle\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$.

(So the estimate is on average correct, or **unbiased***.)*

(ii) $\mathrm{Var}\big[MC_N[f, R]\big] = \dfrac{V}{N}$, for a constant V. Furthermore, V can be estimated[5] by

$$\hat{V} = \frac{A(R)^2}{N}\sum_i f^2(\boldsymbol{X_i}) - \left(\frac{A(R)}{N}\sum_i f(\boldsymbol{X_i})\right)^2. \tag{3.17}$$

△ (know that you can get it from $\sum f(\boldsymbol{X_i})$ and $\sum f^2(\boldsymbol{X_i})$)

(The square root of the estimated variance, $SE = \sqrt{\hat{V}/N}$, is called the **standard error** *and sometimes used to measure accuracy of Monte Carlo methods.)*

[4] Both $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ and $\int_R f^2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ are finite, and f is not constant.

[5] Strictly speaking, we should multiply this estimate by $\frac{N-1}{N}$, which makes it unbiased: a theorem from statistics. However, when N is large this factor is so close to 1 than we can ignore it.

Now denote the error of the Monte Carlo estimate

$$\mathrm{err}(MC_N)[f,R] = A(R)\frac{1}{N}\sum_i f(\boldsymbol{X_i}) - \int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

(iii) $\mathrm{P}\Big[\big|\mathrm{err}(MC_N)[f,R]\big| \geq \epsilon\Big] \to 0$ as $N \to \infty$, for any $\epsilon > 0$.
(So the estimate gets arbitrarily close to the right answer, for large enough N; it is **consistent***.)*

(iv) For large N, $\mathrm{err}(MC_N)[f,R]$ approaches the Normal distribution $\mathrm{N}\big(0, \frac{V}{N}\big)$. The same holds if V is replaced by $\hat{V}$.
(So the estimate is within $\pm SE$ about 68% of the time, within $\pm 2SE$ about 95% of the time, and within $\pm 3SE$ about 99.7% of the time.)

This lemma tells us that Monte Carlo integration works. Furthermore its error decreases as $O(N^{-1/2})$ *regardless of the dimension*. This is vastly inferior to Trapezium or Simpson's rule for one-dimensional integration, but superior in higher dimensions, and when the number of dimensions is dozens or hundreds it is the only game in town. Of course, we might expect f to be slower to evaluate when the dimensionality is large, but this need not and should not be the exponential relationship that manifests in the curse of dimensionality.

On the other hand, we no longer have any *guarantees* about the error of the approximate integral. We only have a probabilistic statement that its standard error is a certain quantity, or that (assuming N is 'large enough' – whatever that means) it is about X% likely to be within a certain range.

It is a common feature of randomized algorithms that they can sometimes avoid the curse of dimensionality, at the cost of certainty on the error bounds.

Proof of Lemma 3.2

(i) is simple: by linearity of expectation,

$$E\left[A(R)\frac{1}{N}\sum_i f(\boldsymbol{X_i})\right] = \frac{1}{N}\sum_i A(R)\mathrm{E}[f(\boldsymbol{X_i})] = \frac{1}{N}\sum_i \int_R \tfrac{A(R)}{A(R)} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

(ii) Using the additive property of variance, because $\boldsymbol{X_1}, \ldots, \boldsymbol{X_n}$ are independent and identically distributed,

$$\mathrm{Var}\big[MC_N[f,R]\big] = \mathrm{Var}\Big[\tfrac{A(R)}{N}\textstyle\sum_i f(\boldsymbol{X_i})\Big] = \tfrac{A(R)^2}{N^2}\textstyle\sum_i \mathrm{Var}[f(\boldsymbol{X_i})] = \tfrac{A(R)^2\mathrm{Var}[f(\boldsymbol{X_i})]}{N}.$$

So

$$V = A(R)^2\mathrm{Var}[f(\boldsymbol{X_i})].$$

Furthermore, the formula for variance is

$$\mathrm{Var}[f(\boldsymbol{X_i})] = \mathrm{E}[f^2(\boldsymbol{X_i})] - \big(E[f(\boldsymbol{X_i})]\big)^2 = \frac{1}{A(R)}\int_R f^2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} - \left(\frac{1}{A(R)}\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\right)^2;$$

for the same reasons that Monte Carlo integration is useful, the first term can be estimated by $\frac{1}{N}\sum_i f^2(\boldsymbol{X_i})$ and the second by the square of $\frac{1}{N}\sum_i f(\boldsymbol{X_i})$. With the factor $A(R)^2$ we get (3.17).

(iii) Let us quote **Chebyshev's inequality** from probability[6]: if Y is any random scalar with finite mean μ and finite, nonzero, variance σ^2 then

$$\mathrm{P}(|Y-\mu| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}.$$

In our case setting $Y = MC_N[f,R]$, (i) proves that $\mu = \int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, so the LHS is $\mathrm{P}\Big[\big|\mathrm{err}(MC_N)[f,R]\big| \geq \epsilon\Big]$. The RHS is $\frac{V}{N\epsilon^2}$, which certainly tends to zero as $N \to \infty$.

(iv) Let us quote, slightly loosely, the **central limit theorem** from probability/statistics: if $Y_1, \ldots, Y_N$ are independent, identically distributed random scalars with finite mean μ and finite, nonzero, variance σ^2 then, for large N, $\frac{1}{N}\sum_i Y_i$ approaches the Normal distribution $\mathrm{N}(\mu, \sigma^2/N)$. In our case use $Y_i = A(R)f(\boldsymbol{X_i}) - \int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$. By (i), $\mu = 0$. By (ii), $\sigma^2 = V$. ■

[6]Look up the proof: it is simple and very beautiful.

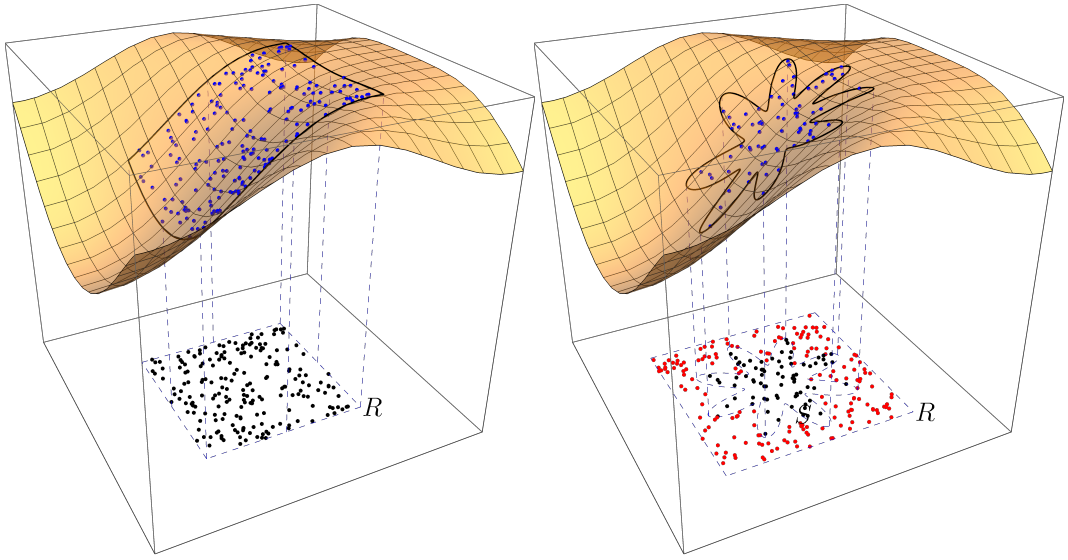


Figure 3.6: Monte Carlo integration over a region that we can sample from uniformly, and a region that we cannot. In the latter case, red dots indicate samples that were rejected for being outside R. The integral is estimated to be the average 'height' of the blue dots, multiplied by the area of R.

3.8 A Monte Carlo Algorithm

How could we use Monte Carlo integration in practice? Generate the random $\boldsymbol{X_i}$, then compute the estimate (3.16) and the standard error $SE = \sqrt{\hat{V}/N}$ using (3.17). Simple?

The difficult part is generating a uniformly distributed random vector $\boldsymbol{X_i}$ from the region R. If R is a rectangle, it is as simple as calling your random number generator[7] for each dimension, and scaling the answer. For circles and other simple geometrical objects there are also easy methods. But generating random vectors from complex regions is a topic in its own right, known as **simulation**.

The simplest way to deal with a complex region R is to find a simple region S with $R \subseteq S$. For example S, could be an n-dimensional rectangle that surrounds R. Then

[7]There are actually a lot of pitfalls in random number generation, but that would take us off topic. Remember John von Neumann: 'Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.'

generate uniform random vectors from S but reject any that do not come from R. Only the accepted vectors contribute to the Monte Carlo integration. If $A(R)$ is a lot smaller than $A(S)$ then it means drawing a lot more samples. Figure 3.6 displays the procedure of Algorithm 1. An alternative is to stop, rather than after a fixed number of samples N, only when the standard error reduces to a desired level.

Algorithm 1 Simple Monte Carlo integration, and calculation of standard error.

$M, Sum, SumSq \leftarrow 0$
for $i \leftarrow 1, \ldots, N$ **do**
  $\boldsymbol{X_i} \leftarrow$ GENERATEUNIFORM(S) ▷ Generates random vector uniformly from S
  if $\boldsymbol{X_i} \in R$ **then**
    $Sum \leftarrow Sum + f(\boldsymbol{X_i})$
    $SumSq \leftarrow SumSq + f(\boldsymbol{X_i})^2$
    $M \leftarrow M + 1$
  end if
end for
$MC_N[f, R] \leftarrow A(R)\frac{Sum}{M}$ ▷ Estimate of $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$
$SE \leftarrow \sqrt{\frac{1}{M}\left(A(R)^2\frac{SumSq}{M} - \left(A(R)\frac{Sum}{M}\right)^2\right)}$ ▷ Standard error

If R is so complex that we do not even know its area/volume $A(R)$, we can estimate that too: $A(S)$ times the proportion of accepted samples from S is a good estimate for $A(R)$; indeed, once you see that

$$A(R) = \int_S \begin{cases} 1, & \boldsymbol{x} \in R \\ 0, & \boldsymbol{x} \notin R \end{cases} \mathrm{d}\boldsymbol{x},$$

this just is another Monte Carlo integral.

There are many ways to improve Monte Carlo integration, which aim to reduce the standard error. That would increase the accuracy of the estimate or, equivalently, reduce the number of samples needed to obtain a desired accuracy.

One method is **stratified sampling**. This divides the region R into sub-regions and samples each region separately, allocating more samples to the regions that would have higher variance, and fewer samples to the 'easier' regions. You will explore a simple

version in one of the tutorial exercises. In practice, stratified sampling is often performed by dividing regions recursively, which leads to a more complex algorithm.

Another variance-reduction method is **importance sampling**. Here, the random vectors $\boldsymbol{X_i}$ are not generated uniformly. Instead, they are generated according to some pdf $g(\boldsymbol{x})$, and the Monte Carlo estimator is altered to

$$MC_N[f,R] = \frac{1}{N}\sum_i \frac{f(\boldsymbol{X_i})}{g(\boldsymbol{X_i})}.$$

It is a one-line proof that this is an unbiased estimator, and when g is chosen well it will have a lower V and hence lower SE. A good g is higher when f is higher, and lower when f is lower: this captures the intuitive idea that parts of R with higher values of f contribute more to the integral, and so should be examined more closely, than parts of R with lower f.

Lastly, there are **control variates**. A control variate is another function $\hat{f}$ that is close to f, but easy to integrate. Then use

$$\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \int_R f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_R \hat{f}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

The second integral can be done exactly, and if $\hat{f}$ is well-chosen and close to f then the error in estimating the first integral should be smaller than in $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$. You could see this as a hybrid between pure Monte Carlo integration and methods such as the Trapezium rule or Simpson's rule that approximate the integrand.

3.9 Bonus Mathematics: Some More Error Bounds

This is <u>not</u> part of the syllabus.

It is reasonably straightforward to find error bounds for single-strip integration rules that are based on Taylor's theorem, as we did for the Midpoint rule in Section 3.3. Methods based on interpolating polynomials are rather trickier. To avoid using the **interpolation error bound** for polynomial interpolation, we try a different approach based on Taylor's theorem. Define

$$F(t) = t\big(f(m+t) + f(m-t)\big) - \int_{m-t}^{m+t} f(x)\,\mathrm{d}x.$$

Note that $F\big(\frac{b-a}{2}\big) = T_1 - I = \mathrm{err}(T_1)[f,a,b]$, so bounding the error of the 1-strip Trapezium rule approximation reduces to bounding F at $\frac{b-a}{2}$. Apply Taylor's theorem with $x_0 = 0$ up to second order,

$$F\Big(\frac{b-a}{2}\Big) = F(0) + \frac{b-a}{2}\frac{\mathrm{d}F}{\mathrm{d}t}(0) + \frac{(b-a)^2}{8}\frac{\mathrm{d}^2F}{\mathrm{d}t^2}(0) + \frac{(b-a)^3}{48}\frac{\mathrm{d}^3F}{\mathrm{d}t^3}(\xi) \tag{3.18}$$

for some $\xi \in \big(0, \frac{b-a}{2}\big)$. Now routine computation of the derivatives of F, using the fact that $\int_a^{m+t} f(x)\,\mathrm{d}x$ is an antiderivative for $f(m+t)$, gives

$$\begin{array}{rclcrcl}
\frac{\mathrm{d}F}{\mathrm{d}t} &=& t\frac{\mathrm{d}f}{\mathrm{d}t}(m+t) - t\frac{\mathrm{d}f}{\mathrm{d}t}(m-t) & & \frac{\mathrm{d}F}{\mathrm{d}t}(0) &=& 0 \\
\frac{\mathrm{d}^2F}{\mathrm{d}t^2} &=& t\frac{\mathrm{d}^2f}{\mathrm{d}t^2}(m+t) + t\frac{\mathrm{d}^2f}{\mathrm{d}t^2}(m-t) + \frac{\mathrm{d}f}{\mathrm{d}t}(m+t) - \frac{\mathrm{d}f}{\mathrm{d}t}(m-t) & & \frac{\mathrm{d}^2F}{\mathrm{d}t^2}(0) &=& 0 \\
\frac{\mathrm{d}^3F}{\mathrm{d}t^3} &=& t\frac{\mathrm{d}^3f}{\mathrm{d}t^3}(m+t) - t\frac{\mathrm{d}^3f}{\mathrm{d}t^3}(m-t) + 2\frac{\mathrm{d}^2f}{\mathrm{d}t^2}(m+t) + 2\frac{\mathrm{d}^2f}{\mathrm{d}t^2}(m-t) & & & &
\end{array}$$

The third derivative is bounded below by $t(\underline{D_3} - \overline{D_3}) + 4\underline{D_2}$ and above by $t(\overline{D_3} - \underline{D_3}) + 4\overline{D_2}$, using the same notation as above. Plugging in to (3.18),

$$\frac{1}{12}(b-a)^3\underline{D_2} - O\big((b-a)^4\big) \le F\Big(\frac{b-a}{2}\Big) \le \frac{1}{12}(b-a)^3\overline{D_2} + O\big((b-a)^4\big).$$

A more careful analysis[8] reaches the same conclusion without the $O((b-a)^4)$ terms, but when we use it to analyze composite methods (which is really the goal) then each strip will have $(b-a) \to 0$, so we can disregard the higher-order term anyway. Up to the most significant term as $(b-a) \to 0$, we have shown

$$\frac{1}{12}(b-a)^3\underline{D_2} \lesssim \mathrm{err}(T_1)[f,a,b] \lesssim \frac{1}{12}(b-a)^3\overline{D_2}.$$

We can use a similar trick to get an error bound for Simpson's rule on the 2-strip interval $[a,b]$, up to the most significant term as $(b-a) \to 0$. This time, apply Taylor's theorem to

$$F(t) = \tfrac{t}{3}\big(f(m-t) + 4f(m) + f(m+t)\big) - \int_{m-t}^{m+t} f(x)\,\mathrm{d}x.$$

Now $\mathrm{err}(S_2)[f,a,b] = F\big(\frac{b-a}{2}\big)$. It is routine to check that F, and all of its first four derivatives, are zero at $t = 0$, so

$$F\Big(\frac{b-a}{2}\Big) = \frac{(b-a)^5}{5!2^5}\frac{\mathrm{d}^5F}{\mathrm{d}t^5}(\xi) \tag{3.19}$$

[8] e.g. Süli & Mayers, Eq. (7.6).

for some ξ. The fifth derivative is found to be

$$\frac{\mathrm{d}^5 F}{\mathrm{d}t^5} = \frac{2}{3}\left(\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}(m+t) + \frac{\mathrm{d}^4 f}{\mathrm{d}x^4}(m-t)\right) + \frac{t}{3}\left(\frac{\mathrm{d}^5 f}{\mathrm{d}x^5}(m+t) - \frac{\mathrm{d}^5 f}{\mathrm{d}x^5}(m-t)\right)$$

which is bounded below by $\frac{t}{3}(\underline{D_5} - \overline{D_5}) + \frac{4}{3}\underline{D_4}$ and above by $\frac{t}{3}(\overline{D_5} - \underline{D_5}) + \frac{4}{3}\overline{D_4}$. Plugging in to the 4th-order Taylor expansion (with fifth-order remainder) for (3.19) gives

$$\frac{1}{2880}(b-a)^5\underline{D_4} - O\big((b-a)^6\big) \leq \operatorname{err}(S_2)[f,a,b] \leq \frac{1}{2880}(b-a)^5\overline{D_4} + O\big((b-a)^6\big).$$

As with the Trapezium rule, a more delicate analysis[9] can prove the same bounds without the $O\big((b-a)^6\big)$ term.

[9] e.g. Süli & Mayers, Thm. 7.2.

Practice Questions

3.1 Compute the following integrals exactly. a is a nonzero constant.

(i) $\int_0^1 x\sqrt{x^2+a}\,\mathrm{d}x$, (ii) $\int_0^1 x(x^2+1)^a\,\mathrm{d}x$,

(iii) $\int_0^1\int_0^1 xy\sqrt{x^2+y^2}\,\mathrm{d}y\,\mathrm{d}x$, (iv) $\int_0^1\int_0^x xy\sqrt{x^2+y^2}\,\mathrm{d}y\,\mathrm{d}x$,

(v) $\displaystyle\int_{x^2+y^2<1} xy\sqrt{x^2+y^2}\,\mathrm{d}(x,y)$, (vi) $\displaystyle\int_{\substack{x,y>0,\\ x^2+y^2<1}} xy\sqrt{x^2+y^2}\,\mathrm{d}(x,y)$.

3.2 Compute the following integrals numerically, using a) the Midpoint rule, b) the Trapezium rule, and c) Simpson's rule. Feel free to write a program to help you.

(i) $\int_{\frac{1}{2}}^1 -x\log_2 x\,\mathrm{d}x$, using 2 intervals,

(ii) $\int_0^\pi \cos\cos x\,\mathrm{d}x$, using 4 intervals,

(iii) $\int_3^4 \log\log x\,\mathrm{d}x$, using 8 intervals.

3.3 Find upper and lower bounds for $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ and $\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}$ when

(i) $f(x) = xe^x$ on $[0,1]$, (ii) $f(x) = e^{e^x}$ on $[0,a]$, (iii) $f(x) = x\tan^{-1}x$ on $[0,2]$.

3.4 It is desired to compute the following integrals numerically, with a maximum error of 10^{-6}. How many intervals would you use, with a) the Midpoint rule, b) Simpson's rule?

(i) $\int_0^1 xe^x\,\mathrm{d}x$, (ii) $\int_0^2 x\tan^{-1}x\,\mathrm{d}x$, (iii) $\int_{-1}^1 \frac{\sin x}{e^x+e^{-x}}\,\mathrm{d}x$.

3.5 Imagine approximating $\int_{x,y\in[0,8]} f(x,y)\,\mathrm{d}(x,y)$, where $f(x,y) = 3xy + 18x^2y - 3xy^2$, by the 2D Midpoint rule. Given a choice between a grid that has 4 strips in the x direction and 2 in the y direction, or vice versa, which might you guess is more accurate? Evaluate both numerical approximations, as well as the exact integral.

3.6 In estimating $\int_R f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, where $R \subseteq S$ and $A(S) = 2$, $N = 1000$ samples were drawn uniformly from S, 100 of which fell outside R and were rejected. Of the others, $\sum f(\boldsymbol{x_i})$ was 314 and $\sum f(\boldsymbol{x_i})^2$ was 271.

Compute the estimate $MC_N[f,R]$, and its standard error. What is an approximate 95% confidence interval for the true integral? Estimate approximately how large N should have been, if we wanted the standard error to be 10^{-4}.

Answers to Chapter 3 Practice Questions

3.1 (i) $\left[\frac{(x^2+a)^{3/2}}{3}\right]_0^1 = \frac{(a+1)^{3/2}-a^{3/2}}{3}$. (ii) $\left[\frac{(x^2+1)^{a+1}}{2(a+1)}\right]_0^1 = \frac{2^{a+1}-1}{2(a+1)}$. (iii) $\frac{2^{5/2}-2}{15}$. (iv) $\frac{2^{3/2}-1}{15}$. (v) 0 (any half of the integral cancels the other). (vi) $\int_0^1 \int_0^{\sqrt{1-x^2}} xy\sqrt{x^2+y^2}\,\mathrm{d}y\,\mathrm{d}x = \frac{1}{10}$. The last two parts are also conveniently computed by changing to polar co-ordinates.

3.2 To 8 significant figures,
(i) $M_2 = 0.14808985$, $T_2 = 0.14031953$, $S_2 = 0.14542604$. (True integral is 0.14550532.)
(ii) $M_4 = 2.4039388$, $T_4 = 2.4039400$, $S_4 = 2.3987530$. (True integral is 2.4039394.)
(iii) $M_8 = 0.22047714$, $T_8 = 0.22023685$, $S_8 = 0.22039685$. (True integral is 0.22039706.)

3.3 Tediously,

	$\frac{\mathrm{d}}{\mathrm{d}x}$	$\frac{\mathrm{d}^2}{\mathrm{d}x^2}$	$\frac{\mathrm{d}^3}{\mathrm{d}x^3}$	$\frac{\mathrm{d}^4}{\mathrm{d}x^4}$
(i)	$e^x(x+1)$	$e^x(x+2)$	$e^x(x+3)$	$e^x(x+4)$
(ii)	e^{x+e^x}	$e^{x+e^x}(e^x+1)$	$e^{x+e^x}(e^{2x}+3e^x+1)$	$e^{x+e^x}(e^{3x}+6e^{2x}+7e^x+1)$
(iii)	$\frac{x+(1+x^2)\tan^{-1}x}{1+x^2}$	$\frac{2}{(1+x^2)^2}$	$\frac{-8x}{(1+x^2)^3}$	$\frac{8(5x^2-1)}{(1+x^2)^4}$

In the first two cases it is clear that the derivatives are strictly increasing, so the min and max are to be found at the start and end of the intervals.
(i) $2 \le \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \le 3e$ and $4 \le \frac{\mathrm{d}^4 f}{\mathrm{d}x^4} \le 5e$.
(ii) $2e \le \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \le (e^a+1)e^{a+e^a}$ and $15e \le \frac{\mathrm{d}^4 f}{\mathrm{d}x^4} \le (e^{3a}+6e^{2a}+7e^a+1)e^{a+e^a}$.
For (iii) the second derivative is decreasing: $\frac{2}{25} \le \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \le 2$. We will have to look at the fifth derivative to find the stationary points of the fourth! It is $\frac{-48x(5x^2-3)}{(1+x^2)^5}$, zero when $x = 0$ or $x = \sqrt{3/5}$. The first is the minimum, the second the maximum: $-8 \le \frac{\mathrm{d}^4 f}{\mathrm{d}x^4} \le \frac{625}{256}$.

3.4 (i) Midpoint rule: need $\frac{1}{24n^2}3e \le 10^{-6}$, hence $n \ge 583$. Simpson's rule: need $\frac{1}{180n^4}5e \le 10^{-6}$, hence $n \ge 17$.
(ii) Midpoint rule: need $\frac{2^3}{24n^2}2 \le 10^{-6}$, hence $n \ge 817$. Simpson's rule: need $\frac{2^5}{180n^4}8 \le 10^{-6}$, hence $n \ge 35$.
(iii) I would not use numerical integration. The integrand is odd: the area above the x-axis in $[0,1]$ equals the area below the x-axis in $[-1,0]$, so the integral is exactly zero.

3.5 Since $\frac{\partial^2 f}{\partial x^2}$ is larger than $\frac{\partial^2 f}{\partial y^2}$ we might expect to prefer more strips in the x direction. That estimate is $\frac{64}{8}\big(f(1,2)+f(3,2)+f(5,2)+f(7,2)+f(1,6)+f(3,6)+f(5,6)+f(7,6)\big) = 84480$; the other is $\frac{64}{8}\big(f(2,1)+f(6,1)+f(2,3)+f(6,3)+f(2,5)+f(6,5)+f(2,7)+f(6,7)\big) = 79104$; the true integral is $\int_0^8 \int_0^8 f(x,y)\,\mathrm{d}y\,\mathrm{d}x = \int_0^8 576x^2 - 416x\,\mathrm{d}x = 84992$.

3.6 $A(R) \approx A(S)\frac{900}{1000} = 1.8$. $MC_N[f,R] = A(R)\frac{314}{900} = 0.628$. $\hat{V} = A(R)^2\frac{271}{900} - \big(A(R)\frac{314}{900}\big)^2 \approx 0.581$. $SE = \sqrt{\frac{\hat{V}}{900}} \approx 0.0254$. The 95% confidence interval is approximately $0.628 \pm 2SE = (0.577, 0.679)$. $SE = \sqrt{\frac{\hat{V}}{M}} = 10^{-4}$ would imply $M = \hat{V}10^8 \approx 58$ million, and since we expect $M \approx 0.9N$ (10% of samples from S were rejected), we would need around 64 million samples to achieve this standard error.

Chapter 4

Accuracy

Once we start using algorithms that return an approximation – be it Taylor's theorem to approximate the value of a function, Simpson's rule to approximate an integrand and hence the integral, or Monte Carlo integration that returns a random number that should be close to the true answer – we encounter error. This short chapter is about how we measure error, and some of the sources of error. We should have some idea of what levels of accuracy are feasible for 'normal' implementations of numbers on computers. And for *iterative* algorithms, that refine the accuracy of an answer step-by-step, we need to measure the rate at which the answer improves.

4.1 Error and Relative Error

Let $u \in \mathbb{R}$ be the number that we want to approximate, and $\tilde{u} \in \mathbb{R}$ its approximation. There are three ways to measure the error:

$$\begin{array}{ll} \tilde{u} - u & \text{is the } \textbf{error}, \\ |\tilde{u} - u| & \text{is the } \textbf{absolute error}, \\ \dfrac{|\tilde{u} - u|}{|u|} & \text{is the } \textbf{relative error}. \end{array}$$

[1] Some books use the opposite sign for the error: $u - \tilde{u}$. But in the spirit of Taylor's theorem we would call this the **remainder** rather than the error.

If we approximate a vector $\boldsymbol{u} \in \mathbb{R}^n$ by $\tilde{\boldsymbol{u}} \in \mathbb{R}^n$, then we would replace the absolute value $|\cdot|$ by the Euclidean norm $\|\cdot\|$. Absolute and relative error are always scalars.

For an example, suppose that we wish to approximate the value of $\sin\left(\frac{355}{113}\right)$ using the Taylor polynomial we derived in **Example 1.6**, but for even more accuracy we decide to use the 14th-order Taylor polynomial

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}. \tag{4.1}$$

Before starting, we computed an upper bound on the error (the remainder term), which in **Example 1.6** we found was $\frac{x^{15}}{15!}\cos(\xi)$ for some ξ, which is less than $(\frac{355}{113})^{15}/15! \approx 0.000021915381$. The *absolute error* of (4.1) is going to be small.

We feel less good about the error when we compute (4.1) as, to 12 decimal places, 0.000020875830. In fact,

$$\sin\left(\tfrac{355}{113}\right) = -0.000000266764189062419148406 37\cdots$$

and the *relative error* was approximately -79.26! We did not even get the sign correct.

This is a case where the absolute error is small, but the relative error is large. Furthermore, we might try to reduce the relative error by increasing the order of the Taylor polynomial. We can do this to some extent, reducing the relative error to around 10^{-9} with a 26th-order polynomial. But, after that, additional terms do not reduce the (absolute or relative) error at all. This is because of how real numbers are stored with limited precision (see the next section).

Whether relative or absolute error is appropriate depends on the application. We might think that an error of ± 1 is significant if the true value is 2, and insignificant if the true value is $2 \cdot 10^9$: this suggests that relative error is a better quantity, and more advanced books on numerical analysis tend to focus on relative error. On the other hand, in this introductory course we will often only be able to deal with absolute error. It is generally simpler to find bounds on absolute error; you might notice that the error bounds for numerical integration, in Chapter 3, were all in absolute terms.

Something that advanced books also discuss is the distinction between **forward error** and **backward error**. Loosely speaking, when we want to approximate the value of $f(x)$ (for example f integrates something that depends on x, or finds the root of a function

that depends on x), by a numerical algorithm that computes $\tilde{f}(x)$, the forward error is $\tilde{f}(x) - f(x)$ and the backward error is $\tilde{x} - x$ when $\tilde{f}(x) = f(\tilde{x})$. The forward error is how much the approximate answer is wrong, and the backward error is how much you would have to change the input to make the approximate answer right! This does not fall within our syllabus, but the distinction between forward and backward error analysis is important in numerical linear algebra.

4.2 Floating-Point Numbers

Computers rarely store a real number precisely. Although it is conceivable to store arbitrarily many decimal places, it is impossible to store infinitely many. And to keep memory requirements small and predictable, almost every language (and every processor) stores a discrete approximation to a real number called a **floating-point number**.

Just as scientists often write numbers as some multiple of a power of ten, e.g. $c = 2.99792 \cdot 10^8$, the most common computer representations of real numbers are powers of two, of the form

$$\pm\left(1 + \frac{m}{2^p}\right) \cdot 2^e$$

where $0 \leq m < 2^p$ and $e \in \mathbb{Z}$. $\frac{m}{2^p}$ is called the **mantissa**[2] and e is the **exponent**. In the IEEE 754 technical standard for floating-point[3] there is a particular format for storing this representation, which need not concern us. Sufficient to know that m is stored in binary using p bits, e (limited to $e_{min} < e < e_{max}$) using $\lceil \log_2(e_{max} - e_{min}) \rceil$ bits, and the $\pm$ sign using 1 bit. The number of bits in the mantissa, p, is called the **precision**. The standard also specifies representations of error (Inf and $-Inf$ for infinite values, and NaN (**not a number**) for the result of calculations like $0/0$), and some special cases for numbers very close to zero, which we will not consider.

There are two widely-adopted formats from the IEEE standard:

Single precision takes 4 bytes of storage: the sign bit, 23 bits of precision, and 8 bits for the exponent with $e_{min} = -126$, $e_{max} = 127$. The type associated with single-precision numbers is commonly called `float` (in C-like languages). In Haskell and Scala it is `Float`.

[2] Donald Knuth dislikes this terminology, as it means something else in the context of logarithms. He suggests calling it the 'fraction part'; who am I to disagree?

[3] The latest version is from 2019: `https://ieeexplore.ieee.org/document/8766229`.

Double precision takes 8 bytes of storage: the sign bit, 52 bits of precision, and 11 bits for the exponent with $e_{min} = -1022$, $e_{max} = 1023$. The type associated with double-precision numbers is commonly called `double` (in C-like languages). In Haskell and Scala it is `Double`. Watch out: in Python it is called `float`.

Some processors also use registers that have **extended precision**, taking 10 bytes of memory and supplying 63 bits of precision. There is also a less-widely implemented quadruple-precision format.

It is probably true that the default for computing today is double precision; the exception would be GPUs, where single precision is still common, as 23 bits of mantissa is sufficient resolution for storing colours of textures and pixels.

What does this mean for numerical computing? First, that (unlike integers) there is almost no chance of overflow. The largest positive number (negative numbers are symmetrical) that can be stored in double precision is just under $2 \cdot 2^{1023}$, which is more than 10^{308}. Instead, the danger is of **rounding error**, because only approximately 15 significant decimal digits are stored. Note that the rounding errors of floating-point numbers are relative, not absolute. Understanding when floating-point rounding is going to cause significant errors could be a lecture course in its own right, but here are some condensed pieces of advice.

(i) Because few floating-point numbers are stored exactly, and will be subject to rounding at intermediate stages of computation, it is dangerous to test them for equality. In Scala (on my system, at least), `var x=0.0; while(x!=1.0) x+=0.1;` fails to terminate because `x` just misses the exact value 1.0. Instead of `if(a==b)` it is common to see `if(abs(a-b)<SMALL)`, for some small constant `SMALL`. Choose `SMALL` with some thought.

(ii) The accuracy of floating-point numbers is usually measured by **machine epsilon**[4], the quantity ϵ such that every number $x \in \mathbb{R}$ can be stored as $\tilde{x}$ with relative error no more than ϵ. For single-precision numbers, $\epsilon = 2^{-24} \approx 6 \cdot 10^{-8}$; for double-precision numbers, $\epsilon = 2^{-53} \approx 1.1 \cdot 10^{-16}$. You should never rely on any floating-point computation having a relative error of less than ϵ. Given that most computations involve a sequence of operations, each of which can introduce or compound previous errors, relative errors a few multiples of ϵ are more common. For example, Brent's

[4] Unfortunately there are two competing definitions for machine epsilon, the other twice the value we have defined here. Both are quite widely used. Be careful when reading other sources.

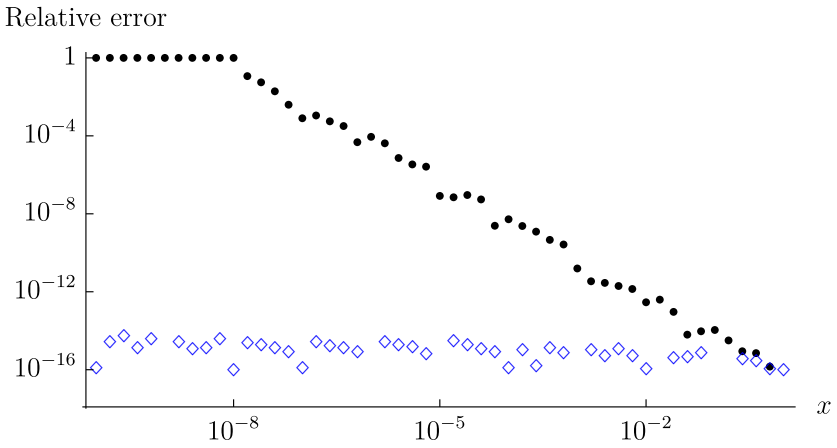


Figure 4.1: Relative error (log-log axes) when computing $1 - \cos x$ (*black dots*) and the equal expression $2\sin^2 \frac{x}{2}$ (*blue diamonds*) in double-precision arithmetic. The latter avoids catastrophic cancellation and has a smaller, more stable, relative error for small x. Note that the more accurate expression still has relative error several multiples of $\epsilon \approx 10^{-16}$.

version of the famous `zeroin` one-dimensional root-finder guarantees relative error of no more than 4ϵ.

(iii) Particular peril lurks where the result of a computation is orders of magnitude smaller than the inputs, because then small relative errors on the inputs can lead to large relative errors on the outputs. A notoriously nasty operation is subtraction: if $a \approx b$ then $a - b \approx 0$, so small *relative* errors in a and/or b can cause large *relative* errors in $a - b$. When this happens it is called **catastrophic cancellation**. Sometimes catastrophic cancellation can be avoided simply be replacing one formula by another, equivalent, formula that avoids subtracting close numbers. For example, in Figure 4.1 we compute $1 - \cos x$ and the equivalent formula $2\sin^2 \frac{x}{2}$, which avoids subtracting close numbers. For small x, the latter is much more accurate. On the other hand, if this occurs in a context like $\sqrt{2} + 1 - \cos x$ then there is probably no need to worry about the cancellation, as the error will remain small relative to the overall answer.

4.3 Sources of Error

What are the sources of error in numerical computing? There are many, and different authors categorize them in different ways. We will mention two main types of error:

Truncation error (in some contexts called discretization error or procedural error) is the error introduced when approximating an infinite procedure by a finite procedure. For example, when we use a finite Taylor polynomial to approximate a function, we have *truncated* a power series and the remainder term is the truncation error. Or when we perform numerical integration we have truncated a hypothetical infinite sum (perfectly accurate for continuous functions) by the finite sum over n strips.

Roundoff error (also called numerical error) is error introduced by approximating a real number, for example in floating-point format. A common mistake is to think that roundoff error is called truncation error because it involves truncating the binary representation of a floating-point number. Roundoff error is **not** truncation error.

We could also identify other sources of error: model error, if we are computing the output of a model but the model itself is an approximation (e.g. in fluid dynamics); input error (if the data fed to our algorithm is measured with some error); programmer error. Some texts also mention **propagation error**, which refers to errors arising in an early part of a numerical method being amplified by a later part. This phenomenon is called **instability** and some classic problems are subject to it. For some $\mathbf{A}$, solving $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ can be very unstable.

To illustrate a problem with two sources of error, let us approximate $\frac{\mathrm{d}f}{\mathrm{d}x}$ by the **numerical derivative**

$$\tilde{d}(x,h) = \frac{f(x+h) - f(x)}{h}.$$

By definition, $\tilde{d}(x,h)$ tends to $\frac{\mathrm{d}f}{\mathrm{d}x}$ as $h \to 0$. So we approximate $\frac{\mathrm{d}f}{\mathrm{d}x}$ by $\tilde{d}(x,h)$ with small, nonzero, h. (Numerical differentiation is a topic in its own right, but cannot fit into this course.) Let us apply this method to the function $f(x) = e^x$, at $x = 0$. Of course, we know the true answer $\frac{\mathrm{d}f}{\mathrm{d}x}(0) = 1$, so the relative error equals the absolute error in this instance. Implementing $\frac{f(x+h)-f(x)}{h}$ on a computer, we show this error for values of h between 10^{-16} and 1. Reading the graph from right-to-left, observe that the error reduces as h reduces, until h gets to a certain level; then it *increases*. The error for large

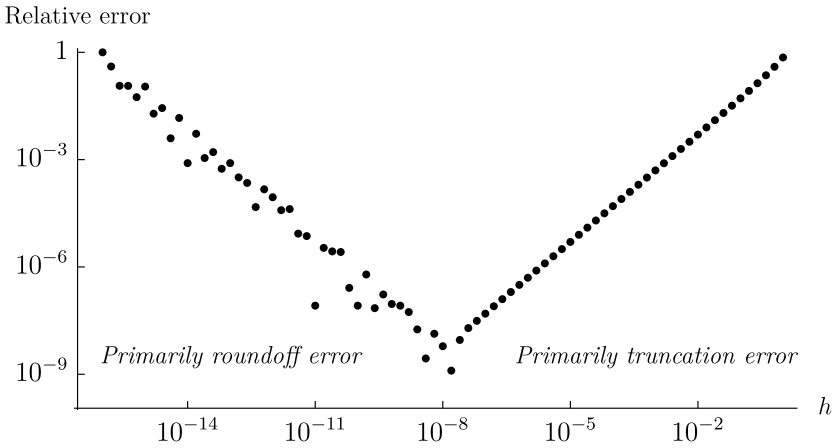


Figure 4.2: Relative error (log-log axes) when estimating $\frac{\mathrm{d}f}{\mathrm{d}x}$ by $\frac{f(x+h)-f(x)}{h}$, for $f(x) = e^x$ and $x = 0$. Note that relative error equals absolute error for this problem. For h not too close to zero, most of the error is truncation error (approximating a limit by a small finite value). When h gets small enough, the relative error is mainly caused by roundoff in $f(x+h) - f(x)$; for very small h the cancellation is catastrophic.

h is mainly due to truncation error, approximating a limit by a small finite value. The error for small h is due to roundoff in calculating $f(x+h) - f(x)$, which is an example of catastrophic cancellation: a small relative error in $f(x+h)$ leads to a large relative error in $f(x+h) - f(x)$ because $f(x+h) - f(x)$ is close to zero. For example, although $f(10^{-16})$ is actually just over $1 + 10^{-16}$, its double-precision floating point representation is 1, so $f(10^{-16}) - f(0)$ comes out to zero.

Truncation error is usually something that we can analyse, and pick suitable parameters to keep it small. In the above example, we could use the remainder of Taylor's theorem to show that $h < 10^{-6}$ should give relative truncation error no more than $5 \cdot 10^{-7}$, for example. But, as we said earlier, roundoff is more difficult to analyze. Scientists using numerical computing have to spot the types of computation where roundoff error is significant, such as catastrophic cancellation.

4.4 Rates of Convergence

Suppose that we have designed an **iterative numerical algorithm**: a procedure that attempts to approximate a true value x^* (or vector $\boldsymbol{x}^*$) by starting with an initial guess

then repeatedly refining the answer, (hopefully) making it more and more accurate. We will see many such algorithms in Chapters 5 and 6. Starting with some x_0 or $\boldsymbol{x_0}$[5], let

$$x_{n+1} = f(x_n), \quad \text{or} \quad \boldsymbol{x_{n+1}} = \boldsymbol{f}(\boldsymbol{x_n}).$$

We identify the absolute error at the n-th step as

$$\epsilon_n = x_n - x^*, \quad \text{or} \quad \epsilon_n = \|\boldsymbol{x_n} - \boldsymbol{x^*}\|.$$

The first goal of our algorithm is that it should converge to the correct answer: $|\epsilon_n| \to 0$ as $n \to \infty$. But how to compare two methods, both of which achieve this goal? In numerical algorithms, just as in exact algorithms, we want to look at the time complexity: in our case, the time complexity of computing x_{n+1} from x_n. But we also need to look at how *quickly* does $|\epsilon_n| \to 0$. The speed of convergence is measured in the following way.

Definition Let ϵ_n be the error of some iterative numerical algorithm A_n after n steps. If $|\epsilon_n| \to 0$ we say that

(i) A_n **converges linearly** if, for some $0 < a < 1$, $\dfrac{|\epsilon_{n+1}|}{|\epsilon_n|} \to a$.

(ii) A_n **converges sublinearly** if $\dfrac{|\epsilon_{n+1}|}{|\epsilon_n|} \to 1$,

a special case, **logarithmic convergence**, if also $\dfrac{|\epsilon_{n+2} - \epsilon_{n+1}|}{|\epsilon_{n+1} - \epsilon_n|} \to 1$.

(iii) A_n **converges superlinearly** if $\dfrac{|\epsilon_{n+1}|}{|\epsilon_n|} \to 0$,

a special case, **order-q convergence**, if $\dfrac{|\epsilon_{n+1}|}{|\epsilon_n|^q} \to a$ for $a > 0, q > 1$.

(All limits as $n \to \infty$. Note that q need not be an integer.)

Order-2 convergence is also known as **quadratic convergence**. Linear convergence is sometimes known as order-1 convergence.

[5] Understanding the region of starting points, for which the iteration will converge to the right answer, is a key part of numerical computing.

We might measure the performance of a numerical algorithm in terms of its actual error, or alternatively in terms of an upper bound on its error (as we did in Chapter 3). The latter is more appropriate when the algorithm approaches the true answer non-monotonically, in which case the error might go up before it goes down again, all while still tending to zero. In some literature the convergence of the actual error is called Q-linear convergence, Q-quadratic convergence, etc., while the convergence of an error *bound* is called R-linear convergence, R-quadratic convergence, etc. We will not need such a distinction.

We can also use slightly relaxed definitions that do not require the ratio of successive errors to converge to a limit, merely to have an upper bound. For example, if

$$\frac{|\epsilon_{n+1}|}{|\epsilon_n|} < a, \text{ for } a < 1,$$

then we have **at least linear convergence**, and if

$$\frac{|\epsilon_{n+1}|}{|\epsilon_n|^q} < a, \text{ for } a > 0, q > 1,$$

then we have **at least order-q convergence**. They are 'at least because the limit of the ratio might be zero.

Example 4.1 For $c < 1$,

(i) $x_n = \frac{1}{n}$ converges **logarithmically** to zero.
(ii) $x_n = c^n$ converges **linearly** to zero.
(iii) $x_n = c^{n^2}$ converges **superlinearly**, but not with order q for any $q > 1$, to zero.
(iv) $x_n = c^{2^n}$ converges **quadratically** to zero.

The words *quadratic* and *linear* do not mean that the error itself is a quadratic or linear function of n. Instead, it is best to understand a linearly convergent algorithm as one that increases the number of correct decimal places ($-\log_{10}|\epsilon_n|$) by a constant at each step, and a quadratically convergent algorithm as one that approximately *doubles* the number of correct decimal places at each step, provided that the error is fairly small to begin with. Quadratic algorithms are highly prized because they need so few steps to

achieve accuracy that reaches the limits of rounding. However, it is a common property of quadratically convergent numerical algorithms that they *fail* to converge unless the starting point is chosen carefully.

When comparing a slower algorithm with a fast convergence against a fast algorithm with slow convergence, you should bear in mind:

Lemma 4.2 If A_n converges with order q, then A_{2n} converges with order q^2.

Proof

$$\frac{|\epsilon_{2(n+1)}|}{|\epsilon_{2n}|^{q^2}} = \frac{|\epsilon_{2n+2}|}{|\epsilon_{2n+1}|^q} \cdot \left(\frac{|\epsilon_{2n+1}|}{|\epsilon_{2n}|^q} \right)^q \to a^{q+1} \quad \text{if} \quad \frac{|\epsilon_{n+1}|}{|\epsilon_n|^q} \to a.$$

■

So, multiplicative constants aside, given an order-1.5 numerical procedure where each step runs twice as fast as a quadratic procedure, the former is probably preferable. It can achieve order-2.25 in equivalent time simply by taking twice as many steps. Of course, our decision could be different if we take into account to numerical stability of the methods, or whether one method is guaranteed to converge starting from a wider range of starting points.

Example 4.3 The iteration

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n},$$

starting with $x_0 = 1$, converges quadratically to $\sqrt{2}$.

(In fact this is true for any $x_0 > 0$, and for any $x_0 < 0$ it converges quadratically to $-\sqrt{2}$.)

4.5 Bonus Mathematics: Acceleration Methods

This is not part of the syllabus.

If a scalar sequence $(x_0, x_1, \ldots)$ converges linearly, we might expect that

$$\frac{x_{n+2} - x^*}{x_{n+1} - x*} \approx \frac{x_{n+1} - x^*}{x_n - x*} \approx a.$$

Setting it to equality and rearranging gives a 'solution' for x^* which is typically more accurate than x_{n+2}, and can be used instead. This is one of many **acceleration methods**, which by clever manipulation can turn a convergent iteration into one that converges more rapidly. In other contexts (notably integration) they are also called **extrapolation methods**, because they can be seen as extrapolating a sequence of approximations to the point at which it appears to be converging.

This is probably the simplest acceleration method, known as **Aitken extrapolation** or the Δ^2 **method**:

Definition If $x_0, x_1, \ldots$ is a sequence in $\mathbb{R}$ then the sequence

$$\tilde{x}_n = x_n - \frac{(x_{n+1} - x_n)^2}{(x_{n+2} - x_{n+1}) - (x_{n+1} - x_n)}$$

(where $\tilde{x}_n$ can be computed once x_{n+2} has been) is called the Δ^2 **process**.

Note that this generates a parallel sequence $(\tilde{x}_n)$, from (x_n).

Extrapolation methods, like all numerical methods, come with conditions and convergence results. For example,

---

Theorem 4.4 If $x_0, x_1, \ldots$ converges linearly to x^* then the Δ^2 process converges strictly faster, in the sense that

$$\lim_{n\to\infty} \frac{\tilde{x}_n - x^*}{x_n - x^*} = 0.$$

Proof Set

$$a_n = \frac{x_{n+1} - x^*}{x_n - x^*};$$

the hypothesis of linear convergence is that $a_n \to a$ for some $0 < a < 1$. Then calculate

$$\begin{aligned}
\frac{\tilde{x}_{n+1} - x^*}{x_n - x^*} &= \frac{x_n - \frac{(x_{n+1}-x_n)^2}{(x_{n+2}-x_{n+1})-(x_{n+1}-x_n)} - x^*}{x_n - x^*} \\
&= 1 - \frac{(x_{n+1} - x_n)^2}{(x_n - x^*)(x_{n+2} - 2x_{n+1} + x_n)} \\
&= 1 - \frac{\left(\frac{(x_{n+1}-x^*)-(x_n-x^*)}{x_n-x^*}\right)^2}{\frac{(x_{n+2}-x^*)-2(x_{n+1}-x^*)+(x_n-x^*)}{x_n-x^*}} \\
&= 1 - \frac{(a_n - 1)^2}{a_{n+1}a_n - 2a_n + 1} \\
&\to 1 - \frac{(a-1)^2}{a^2 - 2a + 1} = 0.
\end{aligned}$$

■

In some cases, Aitken extrapolation can generate a quadratically convergent sequence. An example is Steffensen's method, which is an iterative algorithm for approximating the root of a univariate function: the subject of chapter 5.

The formula for Aitken extrapolation must be treated with care. It is less numerically stable if the denominator is simplified to

$$(x_{n+2} - x_{n+1}) - (x_{n+1} - x_n) = x_{n+2} - 2x_{n+1} + x_n,$$

and suffers from catastrophic cancellation if the entire formula is 'simplified' to

$$\tilde{x}_n = \frac{x_n x_{n+2} - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n}.$$

Practice Questions

4.1 Using the results in Section 3.4, find upper bounds for the absolute and relative errors when integrating

$$\int_0^1 x^4 + \tfrac{11}{47}x^3 - \tfrac{38}{49}x^2 \,\mathrm{d}x$$

using Simpson's rule with n strips.

4.2 In the **quadratic formula** for solving $ax^2 + bx + c = 0$,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \qquad (*)$$

identify possible catastrophic cancellation when (i) $b^2 - 4ac$ is many orders of magnitude smaller than b^2, and (ii) b^2 is many orders of magnitude greater than $4ac$.

Given the choice between $(*)$ and the equivalent so-called **citardauq formula**,

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}, \qquad (\dagger)$$

which would you use?

4.3 Determine the rate of convergence of a hypothetical iterative numerical algorithm where the error after n steps, ϵ_n, is:

(i) $3/(n+3)$, (ii) $3^{-n}n^{-3}$, (iii) $3^{(-3^n - n^3)}$, (iv) $1/(3n)!$.

4.4 Determine the rate of convergence of a hypothetical iterative numerical algorithm where the error after n steps ϵ_n, satisfies the recurrences:

(i) $|\epsilon_{n+1}| = 3|\epsilon_n^3|,\ |\epsilon_0| < \frac{1}{\sqrt{3}}$ (ii) $|\epsilon_{n+1}| = \frac{n+1}{n+2}|\epsilon_n|,\ |\epsilon_0| < 1$ (iii) $\frac{|\epsilon_{n+2}|}{|\epsilon_{n+1}(\epsilon_n)^6|} \to 1.$

Hint: for (iii), assume $\frac{|\epsilon_{n+2}|}{|\epsilon_{n+1}|^q} \to a$*, and solve for* q *and* a*.*

4.5 The iteration $x_0 = 1$,

$$x_{n+1} = \frac{x_n(x_n^2 + 6)}{3x_n^2 + 2}$$

converges to $\sqrt{2}$ faster than **Example 4.3**; determine its order of convergence. Which of the two methods would you prefer to approximate $\sqrt{2}$ on a computer without a square-root function?

Answers to Chapter 4 Practice Questions

4.1 The exact answer is $\int_0^1 x^4 + \frac{11}{47}x^3 - \frac{38}{49}x^2 \,\mathrm{d}x = \frac{1}{5} + \frac{11}{47\cdot 4} - \frac{38}{49\cdot 3} = \frac{1}{138180} \approx 7.237 \cdot 10^{-6}$. The fourth derivative of the integrand is (the constant) 24. By (3.13), the absolute error is bounded by $\frac{(1-0)^5}{180n^4}24 = \frac{2}{15n^4}$. The relative error is the absolute error divided by $\frac{1}{138180}$, which simplifies to $\frac{18424}{n^4}$.

4.2 (i) If $b^2 - 4ac \approx 0$ then $b^2 - 4ac$ looks like potential catastrophic cancellation. In fact it should not be, because the answer is next added to $-b$, which is orders of magnitude larger, so a small absolute (even if large relative) error in $b^2 - 4ac$ will not cause a large relative error in the overall answer.
(ii) But if $b^2 \gg 4ac$, for the lower root the subtraction $-b + \sqrt{b^2 - 4ac}$ could be catastrophic. It could lead to a large relative error in the answer. In that case the equivalent formula avoids the close-to-zero subtraction.
The citardauq formula suffers from the same problem at the upper root. Assuming $b > 0$ (for $b < 0$ the answers are reversed), the two roots are most stably given by

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad \frac{2c}{-b - \sqrt{b^2 - 4ac}}.$$

4.3 (i) Logarithmic. (ii) Linear. (iii) Order-q for any $q < 3$, but not order-3. (iv) Superlinear, but not order-q for any $q > 1$.

4.4 (i) Cubic. (ii) Logarithmic. (iii) Dividing $\frac{|\epsilon_{n+2}|}{|\epsilon_{n+1}(\epsilon_n)^6|} \to 1$ by $\frac{|\epsilon_{n+2}|}{|\epsilon_{n+1}|^q} \to a$ gives $\frac{|\epsilon_{n+1}|^{q-1}}{|\epsilon_n|^6} \to \frac{1}{a}$, hence $\frac{|\epsilon_{n+1}|}{|\epsilon_n|^{6/(q-1)}} \to \frac{1}{a^{1/(q-1)}}$. This means $q = 6/(q-1)$, hence $q = 3$: cubic convergence.

4.5

$$\frac{x_{n+1} - \sqrt{2}}{(x_n - \sqrt{2})^q} = \frac{\frac{x_n(x_n^2+6)}{3x_n^2+2} - \sqrt{2}}{(x_n - \sqrt{2})^q} = \frac{x_n^3 - 3\sqrt{2}x_n^2 + 6x_n - 2\sqrt{2}}{(3x_n^2+2)(x_n - \sqrt{2})^q} = \frac{(x_n - \sqrt{2})^3}{(3x_n^2+2)(x_n - \sqrt{2})^q}$$

so with $q = 3$

$$\frac{x_{n+1} - \sqrt{2}}{(x_n - \sqrt{2})^q} = \frac{1}{3x_n^2 + 2}$$

which tends to $\frac{1}{5}$, given that $x_n \to \sqrt{2}$. The convergence is cubic.
But two steps of **Example 4.3** would have order-4 convergence, and should be faster (two reciprocals are required for two steps of **Example 4.3**, but fewer multiplications).

Chapter 5

Algorithms for Root-Finding

Finding the **roots** or **zeros** of a continuous function is an ancient art: methods for solving quadratic equations go back to the Babylonians (though they had no concept of *equation*) and mathematical history charts progress in finding roots of wider classes of functions. But some equations cannot be solved, and methods for locating *approximate* roots were developed as early as the 17th century. They are still useful today.

Following the usual pattern, we will begin with root-finding methods for univariate functions $f : \mathbb{R} \to \mathbb{R}$. They will all be iterative, in the sense of the last chapter: given some starting point they generate either a sequence $(x_0, x_1, x_2, \ldots)$ which aims to converge to some x^* where $f(x^*) = 0$, or a sequence of smaller-and-smaller intervals in which x^* is located. They can be compared by a) the simplicity of the iterative step, b) their order of convergence, and c) the range of starting points for which they do converge and the smoothness conditions for the error analysis to hold. Some trade-offs are inevitable here.

Then we consider root-finding in d dimensions. This solves $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ for a vector function $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d$ (i.e. a set of d simultaneous equations). The techniques are more advanced, though they still use only matrix-vector operations at each step. Their error analysis is not on the syllabus, but a convergence result is given as bonus material.

One can also attempt to find $\boldsymbol{x}$ to make $\boldsymbol{f}(\boldsymbol{x})$ as close a possible to $\mathbf{0}$, usually when $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^m$ with $m > d$. This is called an **overdetermined system**, but there is rarely a zero because there are more equations than variables (unless some of the m

simultaneous equations are redundant). Instead, overdetermined systems are solved by finding

$$\arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \|\boldsymbol{f}(\boldsymbol{x})\|.$$

This is a problem of optimization rather than root-finding, but root-finding methods can be adapted to it. We will come to it in the next chapter.

5.1 When to Stop

We will be running iterative algorithms generating a sequence $(x_0, x_1, \ldots)$ that, hopefully, converges to a root x^*. We are most unlikely to find an exact solution to $f(x) = 0$ (not least because the exact solution might not be representable as a floating-point number), so at some point we stop and take x_n as our approximate solution. Termination conditions are usually phrased in terms of a **tolerance** parameter tol, a small positive number, and can include:

(i) $|x_n - x_{n-1}| < tol$,
the last iteration did not change significantly; because of equation (5.2), $|x_n - x_{n-1}|$ can also be an estimate of forward error.

(ii) $|x_n - x_{n-1}| < tol|x_n|$,
the same, but measuring relative change or error.

(iii) $|f(x_n)| < tol$,
the function was sufficiently close to zero (a bound on backward error).

(iv) $n = N$,
we have run out of time (and we should warn the user that their specified tolerance was not met).

(v) an iteration step was ill-defined (we must raise an error).

It is common to combine (i) and (ii) into a single condition

(i)* $|x_n - x_{n-1}| < tol(1 + |x_n|)$ or sometimes $|x_n - x_{n-1}| < tol\max(1, |x_n|)$,
the iteration either made a small relative change *or* a small absolute change.

Whether to prefer conditions (i)/(ii)/(i)* or (iii) depends on the application. If f is particularly shallow near the root, and tol is small, there might not exist any floating-point number where (iii) holds. And a word of warning about relative forward error: seeking a root very close to zero is prone to roundoff error problems.

We should always include condition (iv), to prevent nontermination if, for example, an iteration gets stuck in a loop. We must, of course, always include condition (v).

How do we choose the tolerance parameter *tol*? It depends on the application and may be specified by the user. It often defaults to 2ϵ or 4ϵ, where ϵ is machine epsilon, but this can be ambitious if the iterative step suffers from roundoff error.

5.2 Interval Bisection (1 dimension)

The most reliable way to locate a root in 1 dimension is to start with a **bracket**, an interval (a_0, b_0) where $f(a_0)$ and $f(b_0)$ have different signs. We can be sure that f has at least one root in this interval, as long as f is continuous (this is the **intermediate value theorem**). A method that finds and refines a bracket is called an **inclusion method**; most of the other root-finding algorithms we will see are not inclusion methods.

Given a bracket (a_n, b_n) we can refine it to a bracket exactly half as big, by testing the sign of $f(x_n)$ where $x_n = \frac{a_n+b_n}{2}$. Either (a_n, x_n) or (x_n, b_n) becomes the new bracket (a_{n+1}, b_{n+1}). That is not to say that there is no root in the other half – f might have lots of roots – but that there is guaranteed to be at least one in the smaller bracket.

Just as binary search does in discrete algorithms, this halves the problem, hence the name **interval bisection**. Unlike binary search, it is unlikely to terminate by finding an exact root (and recall: we try to avoid testing floating-point numbers for equality). Instead we should terminate when the answer is accurate enough for our purposes, or when any further subdivision is impossible because of floating-point accuracy. Because bisection has a precise error bound, it is common simply to specify the number of iterations in advance.

As an aside: when practising bisection, or indeed any of the methods in this chapter, do not be tempted by the simplicity of a recursive implementation[1]. The overhead of the recursive call is often significant, and avoided by using iteration instead.

Interval bisection has two drawbacks: it is slow, and it does not generalize to higher dimensions.

[1] unless you are using a purely functional language, in which case recursion is inevitable and you need to ask yourself why you are using a purely functional language.

Algorithm 2 Interval bisection method. Replace $|b_n - a_n| < tol$ by another termination condition, if desired.

procedure BISECTION(f, a_0, b_0, N, tol)
  require $f(a_0)f(b_0) < 0$ ▷ Must begin with a bracket for a root
  $n \leftarrow 0$
  $x_0 \leftarrow \frac{a_0+b_0}{2}$
  repeat
    if $f(a_n)f(x_n) < 0$ **then** ▷ We should terminate now if $f(x_n) = 0$
      $(a_{n+1}, b_{n+1}) \leftarrow (a_n, x_n)$
    else
      $(a_{n+1}, b_{n+1}) \leftarrow (x_n, b_n)$
    end if
    $n \leftarrow n + 1$
    $x_n \leftarrow \frac{a_n+b_n}{2}$
  until $n = N \vee |b_n - a_n| < tol$
  return x_n
end procedure

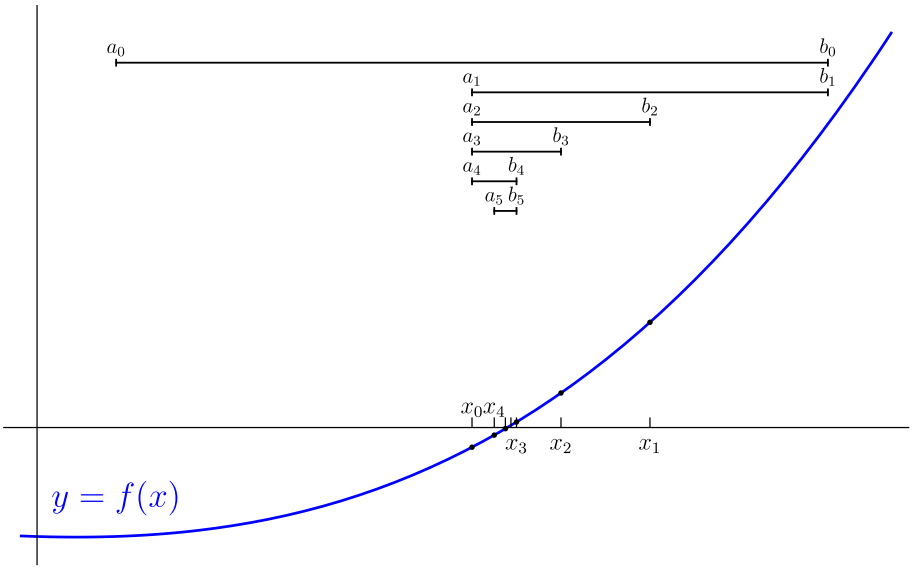


Figure 5.1: Interval bisection: because the signs of $f(a_n)$ and $f(b_n)$ differ, every interval contains at least one root. Each is exactly half the width of the last.

5.2.1 Error Analysis

There is almost nothing to analyze: at each step the bisection method exactly halves the interval. If we take the midpoint of the interval as the n-th approximation, $x_n = \frac{a_n+b_n}{2}$, the error $x_n - x^*$ is *not* guaranteed to reduce at every step (see Figure 5.1, where x_0 is closer to x^* than x_1 or x_2). But the upper bound on the error $\epsilon_n = \frac{1}{2}|b_n - a_n|$ satisfies

$$\epsilon_{n+1} = \tfrac{1}{2}\epsilon_n.$$

This is linear convergence. We need $n = \lceil \log_2 \frac{b_0-a_0}{2tol} \rceil$ to guarantee absolute error less than tol. Alternatively, we can set a relative error bound, or stop iterating when we achieve $|f(x_n)| < tol$.

5.2.2 Finding a Starting Bracket

Sometimes there is no obvious bracket (a_0, b_0). There are simple algorithms to help find one, but they require some information about the function. This is necessary because an arbitrary function might be positive almost everywhere, ducking below the x-axis for a brief time only. Without any information as to whether this could happen, or a minimum

on the interval that it is below the x-axis, there is no algorithm that guarantees to find a bracket.

But, for example, if we know that the function is increasing (but have no other information about it) then we could test $(-1, 1)$ as a bracket, and if not then $(-2, 2)$, ..., $(-2^k, 2^k)$. This guarantees to find a starting bracket in a number of steps logarithmic in the size of the root. In other situations we might search for a bracket of the form $(0, 2^k)$ or $(0, 2^{-k})$. Failing that, we will need to do some mathematics to bound $f(x)$ and derive a bracket using algebra.

Example 5.1 Let us return to **Example 2.14**. After solving the Lagrange system, we found that

$$\pi_i(\lambda) = \frac{1}{1 + \exp(\lambda c_i)},$$

where $c_i \geq 0$ are known constants, but λ is unknown. We determine λ from the payload constraint

$$f(\lambda) = \sum_{i=1}^{n} H\big(\pi_i(\lambda)\big) - m = 0,$$

where $m < n$ is a known constant.

Observe that $\pi_i(0) = \frac{1}{2}$, so $f(0) > 0$. Also, all π_i and f are decreasing in λ. Therefore we first seek a starting bracket of the form $(0, 2^k)$, and having found it use interval bisection to locate λ. In this application we are not concerned about setting a relative or absolute error bound on λ, but are instead content to stop iterating when $\sum_{i=1}^{n} H\big(\pi_i(x_n)\big) \in (m, m+1)$.

5.3 Newton's Method (1 dimension)

A powerful, if sometimes temperamental, root-finding method is to approximate a function $f(x)$ by its 1st-order Taylor polynomial, and then solve the approximation:

$$\hat{f}(x) = f(x_0) + (x - x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0) = 0 \quad \Leftrightarrow \quad x = x_0 - \frac{f(x_0)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)}.$$

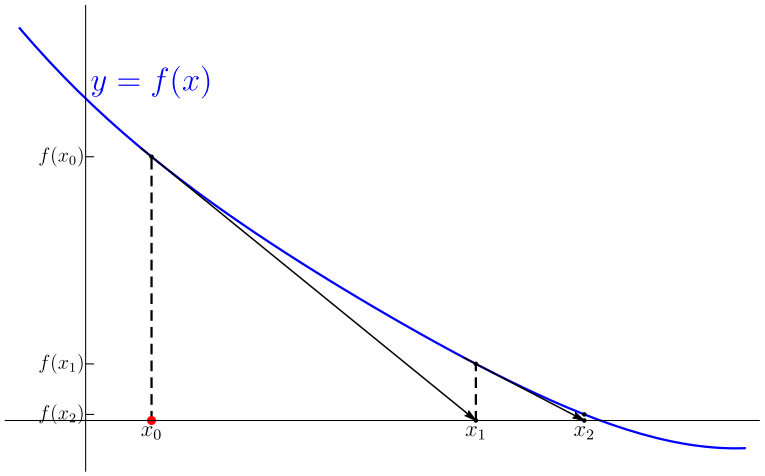


Figure 5.2: Newton's method, which repeatedly approximates f by a 1st-order Taylor polynomial at $x = x_n$, and solves the approximation to give x_{n+1}.

Using the right-hand side as x_1, then iterating the same step at x_1 to derive x_2, and so on, gives **Newton's method**:

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)}. \tag{5.1}$$

See Figure 5.2. The method is known as **Newton's method** or the **Newton-Raphson method**, because Isaac Newton used it (for polynomials only, and without mentioning the derivative) in 1669, and it was refined by Joseph Raphson in 1690. Its modern version emerged in the 19th century.

As we shall see, Newton's method can converge rapidly, but it does require evaluation of the derivative of f. This can be significantly slower than the evaluation of f, and in some cases it is simply unknown (for example if the value of $f(x_n)$ is evaluated by measuring a physical quantity).

To make the method practical we should state termination criteria, including a maximum number of iterations. We must also handle an error if $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n) = 0$, where the iteration is undefined. Unfortunately, there are other ways that Newton's method can fail: the iteration can in fact diverge; it can converge to a different root than we hoped (although demanding that a root-finding method converge to a *particular* root is a tricky problem

anyway); it can get stuck in an oscillation. See Figure 5.3 for some illustrations. We might *hope* that these cannot happen if we start with x_0 that is 'reasonably close' to x^*. For 'well-behaved' functions, that is so, but we need to prove this.

A useful observation about Newton's method concerns the termination criteria. By the same Taylor expansion that derived (5.1),

$$f(x^*) = f(x_n) + (x^* - x_n)\frac{\mathrm{d}f}{\mathrm{d}x}(x_n) + e_2$$

where e_2 is a remainder term of order $(x^* - x_n)^2$. Neglecting the error term, and using $f(x^*) = 0$, this rearranges to

$$x^* - x_n \approx -\frac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} = x_{n+1} - x_n. \tag{5.2}$$

This is an example of an **a posteriori error estimate**, and it justifies the use of $|x_n - x_{n-1}| < tol$ as a termination condition: this approximates a bound on the forward error $x^* - x_{n-1}$, which is not normally known. The cost is that we must perform one further iteration x_n in order to learn that x_{n-1} was sufficient! On the other hand, a termination condition $|f(x_n)| < tol$ is a bound on the backward error.

Algorithm 3 Newton's method in 1 dimension. Replace $|x_n - x_{n-1}| < tol(1 + |x_n|)$ by another termination condition, if desired.

---

procedure NEWTON1D(f, x_0, N, tol)
  $n \leftarrow 0$
  repeat
    $x_{n+1} \leftarrow x_n - \dfrac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)}$ ▷ Should raise an error if $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n) = 0$!
    $n \leftarrow n + 1$
  until $n = N \vee |x_n - x_{n-1}| < tol(1 + |x_n|)$
  return x_n ▷ Should raise a warning if $n = N$
end procedure

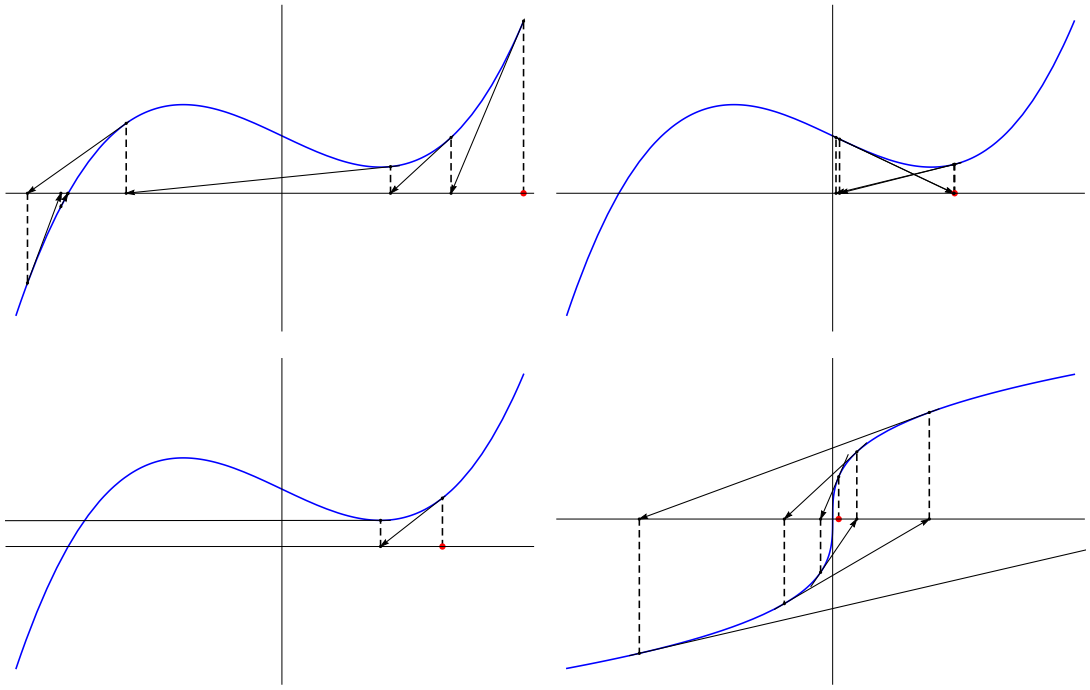


Figure 5.3: Failure of Newton's method. *Top left*: starts a long way from the root but does converge. *Top right*: get stuck in a loop. *Bottom left*: hits a location where $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$. *Bottom right*: diverges.

Example 5.2 The equation

$$x^3 - 2x + 2$$

has one root. Bisection shows that the root lies between -2 and -1. Newton's iteration is

$$x_{n+1} = x_n - \frac{x_n^3 - 2x_n + 2}{3x_n^2 - 2}.$$

As we will see, this iteration does not always converge, but it is extremely fast when it does converge.

5.3.1 Error Analysis

Newton's method is analyzed using Taylor's theorem. We want to find conditions on f to make it 'well-behaved', and how 'reasonably close' we must be to the root to guarantee quadratic convergence. We cannot find necessary conditions for the latter (in fact the regions of convergence for Newton's method are often *fractal*) but we can find useful sufficient conditions. A good combination is to start root-finding using interval bisection, then switch to Newton's method when the bracket is small enough to guarantee quadratic convergence from then on: we need sufficient conditions for such a bracket.

We first establish a bound between the error at step n and step $n+1$, then give an interval $(x^* - c, x^* + c)$ for which convergence is guaranteed and of quadratic order. Since x^* is not usually known, we convert this into a sufficient condition when we only know a bracketing interval for x^*.

Lemma 5.3 Let f have two continuous derivatives, and $f(x^*) = 0$. Write $\epsilon_n = x_n - x^*$.

(i) If $x_n \in I = (x^* - c, x^* + c)$ and

$$A(c) = \frac{\max_{\beta \in I} \left| \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\beta) \right|}{\min_{\alpha \in I} \left| \frac{\mathrm{d}f}{\mathrm{d}x}(\alpha) \right|}$$

then[2] $|\epsilon_{n+1}| \leq \frac{A(c)}{2} \epsilon_n^2$.

(ii) If $x_0 \in I$, and further $\frac{cA(c)}{2} < 1$, then Newton's method converges at least quadratically to x^*.

(iii) If $x^* \in (m - c, m + c)$ and for $I = (m - 2c, m + 2c)$,

$$B(c) = \frac{\max_{\beta \in I} \left| \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\beta) \right|}{\min_{\alpha \in I} \left| \frac{\mathrm{d}f}{\mathrm{d}x}(\alpha) \right|}$$

and further $\frac{cB(c)}{2} < 1$, then Newton's method starting with $x_0 = m$ converges at least quadratically to x^*.

(iv) Such an interval I exists if $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) \neq 0$.

---

[2] We use max and min interchangably with sup and inf, here: since everything in sight is continuous, the sup and inf must exist.

Proof (i) Rearranging,

$$\begin{aligned} x_{n+1} - x^* &= \left(x_n - \frac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} \right) - x^* \\ &= \frac{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)(x_n - x^*) - f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} && \{ \text{ algebra } \} \\ &= \frac{f(x^*) - \big(f(x_n) + (x^* - x_n)\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)\big)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} && \{ f(x^*) = 0 \} \\ &= \frac{\frac{1}{2}(x^* - x_n)^2 \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)}, \quad \xi \in (x_n, x^*) && \{ \text{ Taylor's theorem at } x_n \} \end{aligned}$$

and the result follows because

$$\left| \frac{\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} \right| \leq A(c).$$

(ii) Let $\rho = \frac{cA(c)}{2}$. The hypothesis is $\rho < 1$. We show by induction that $|\epsilon_n| \leq \rho^n |\epsilon_0|$. The base case is immediate. Given $|\epsilon_k| \leq \rho^k |\epsilon_0| < c$, we know that x_k lies in I and so (i) applies to it, giving

$$|\epsilon_{k+1}| \leq \tfrac{A(c)}{2}|\epsilon_k|^2 \leq \tfrac{A(c)}{2} c |\epsilon_k| = \rho |\epsilon_k| \leq \rho^{k+1} |\epsilon_0|.$$

This shows that $\epsilon_n \to 0$ as $n \to \infty$. Furthermore, (i) gives

$$\frac{|\epsilon_{n+1}|}{|\epsilon_n|^2} \leq \frac{A(c)}{2},$$

for all n, which is at least quadratic convergence (it might be even higher-order convergence if $A(c) \to 0$).

It is important that $\rho < 1$, otherwise some x_k might fall outside I and prevent us using (i). Even if we start with a bracketing interval for x^*, Newton's method is not in general guaranteed to stay within it.

(iii) Let $\rho = \frac{cB(c)}{2}$. If $x^* \in (m - c, m + c)$ and $x_0 = m$ then $|\epsilon_0| < c$, so we can deduce $|\epsilon_1| < \rho c < c$, but we cannot be sure that $x_1 \in (m - c, m + c)$ (x^* might lie very close to $m + c$, say, and x_1 on the other side of x^*). We can only be sure that

$x_1 \in (m-2c, m+2c)$. By the same inductive argument as (ii), so do all subsequent x_n, and $|\epsilon_n| \leq \rho^n |\epsilon_0|$. Quadratic convergence follows.

(iv) If $\frac{\mathrm{d}f}{\mathrm{d}x}$ and $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ are both continuous then, as $c \to 0$,

$$A(c) \to \frac{\left|\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^*)\right|}{\left|\frac{\mathrm{d}f}{\mathrm{d}x}(x^*)\right|}.$$

This is finite by assumption, and so $\frac{cA(c)}{2} < 1$ for sufficiently small c. The same is true for $B(c)$. ■

This lemma highlights the problem of a **double root**, which is x^* such that $f(x^*) = \frac{\mathrm{d}f}{\mathrm{d}x}(x^*) = 0$. Then the conditions of the Lemma can never be met, and normally Newton's method does *not* have quadratic convergence. (It can, however, be modified to have quadratic convergence: you will explore this on a tutorial sheet.) As long as x^* is not a double root, and f has two continuous derivatives, f is 'well-behaved' for Newton's method. On the other hand, in the happy case where $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) \neq 0$ and $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^*) = 0$ then convergence might be even faster than quadratic. (Try solving $x^3 + x = 0$ using Newton's method for an example of this.)

We have talked about 'the' root x^*. Note that, if the conditions of Lemma 5.3(ii) or (iii) are met, there can only be one root in I. Otherwise there would have to be a stationary point in I, which is impossible if $A(c)$ or $B(c)$ is to be finite.

We should stress that the conditions in Lemma 5.3 are certainly not necessary. There are many functions f and starting points x_0 that do not meet the conditions, yet f converges quadratically. For example, *every* convex (or concave) function that does not have a double root, with *any* starting point. You will explore a case of this on one of the problem sheets. Our proof required f to have two continuous derivatives, yet there are functions where Newton's method converges quadratically where the 2nd derivative does not even exist.

Finally, we should note that we have only analysed the **truncation error** of Newton's method. We have not analyzed the roundoff error, which is difficult. On one hand, if the calculation of x_{n+1} introduces a small roundoff error, this should to some extent be removed at the next step. But because the iteration involves division there is scope for roundoff errors in $f(x_n)$ to be magnified when $\frac{\mathrm{d}f}{\mathrm{d}x}$ is close to zero. Even when x^* is not a

double root, if f has a very small derivative at the root (such f are called **ill-conditioned**) then roundoff can become a significant source of error or instability.

Example 5.4 In applied probability, a **branching process** is a sequence of integer-valued random variables $(Z_0, Z_1, \ldots)$ representing the size of some population at discrete time intervals (*generations*) $t = 0$, $t = 1$, ... Branching processes can be used to model disease, the spread of malware, and genetics. The population evolves as follows: at first there is one individual, $Z_0 = 1$. At each time interval every individual of the existing population generates some (independent and identically distributed) random number of successors and then dies. Thus Z_{t+1} is the sum of Z_t (independent and identically distributed) random variables.

One of the key questions to ask about a branching process is the **probability of ultimate extinction**: the probability that $Z_t = 0$ at any point t. Using only elementary probability techniques, it can be shown that this probability is a solution of the equation

$$G(x) = x, \tag{5.3}$$

where G is the probability generating function for each individual's number of successors. Note that $x = 1$ is always a solution of (5.3), but when ultimate extinction is not certain there is also a smaller root: that is the probability we seek.

For example, if each member of the population generates a Poisson number of successors, with parameter $\lambda > 1$, then the probability of ultimate extinction is the *lower* root of

$$f(x) = e^{\lambda(x-1)} - x = 0.$$

We must apply Newton's method carefully, to avoid converging to the 'wrong' root $x = 1$. Also note that $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ for some value of x, which we need to avoid.

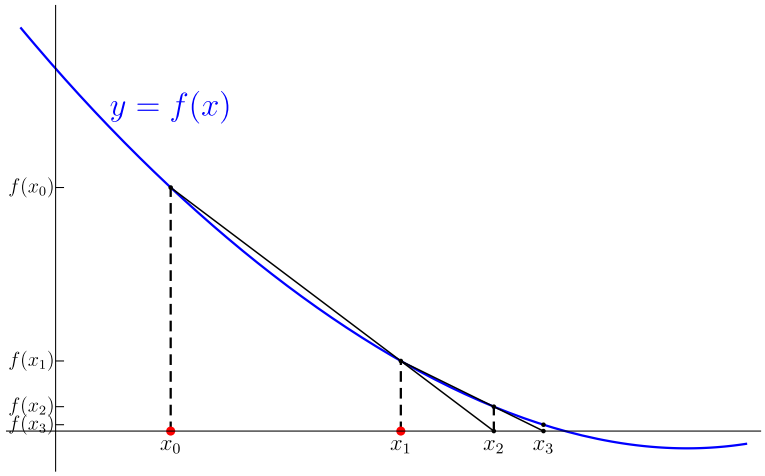


Figure 5.4: The Secant method approximates f by the secant connecting points on f at the last two values of x.

5.4 The Secant Method (1 dimension)

Computing derivatives can be expensive, or impossible in some cases. An alternative to approximating f by a linear Taylor polynomial is to use **linear interpolation**. Given two points on the curve $\big(x_0, f(x_0)\big)$ and $\big(x_1, f(x_1)\big)$, we approximate and solve:

$$\hat{f}(x) = \frac{(x - x_0)f(x_1) - (x - x_1)f(x_0)}{x_1 - x_0} = 0 \quad \Leftrightarrow \quad x = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}.$$

Using the right-hand side as x_2, and iterating this method, gives us the **Secant method**:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}. \tag{5.4}$$

See Figure 5.4. An advantage of this method is that $f(x_{n-1})$ was already computed at the previous iteration, so it only requires evaluating f once per step. And of course it does not use derivatives at all. The Secant method predates Newton's method, some say by thousands of years.

Another way to understand the Secant method is to think of $\frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})}$ as an approximation of $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)$: it is the truncation of the infinite process as $(x_n - x_{n-1}) \to 0$, which

should happen when x_n converges. The class of Newton-like methods that approximate the derivative/gradient are called **quasi-Newton methods**.

Just as with Newton's method, the Secant method can fail. If we hit a point where $f(x_n) = f(x_{n-1})$, the quotient is undefined. Sometimes we will wish to terminate if $|f(x_n) - f(x_{n-1})| < tol|f(x_n)|$, otherwise the next step might go far from the root. The iteration can diverge, oscillate, or converge to a root other than the intended. To know when this cannot happen, we need to analyze the Secant method's error.

Algorithm 4 The Secant method. Replace $|x_n - x_{n-1}| < tol(1 + |x_n|)$ by another termination condition, if desired.

---

procedure SECANT(f, x_0, x_1, N, tol)
  $y_0 \leftarrow f(x_0)$
  $n \leftarrow 1$
  repeat
    $y_n \leftarrow f(x_n)$
    $x_{n+1} \leftarrow x_n - \dfrac{y_n(x_n - x_{n-1})}{y_n - y_{n-1}}$ ▷ Should raise an error if $f(x_n) - f(x_{n-1}) = 0$!
    $n \leftarrow n + 1$
  until $n = N \vee |x_n - x_{n-1}| < tol(1 + |x_n|)$
  return x_n ▷ Should raise a warning if $n = N$
end procedure

Example 5.5 We can apply the Secant method, instead of interval bisection, to **Example 5.1**. We might not want to use Newton's method, because the derivative is moderately expensive to compute, but the Secant method uses fewer evaluations of $f(\lambda)$ and converges as long as we impose an additional test on the iterative step.

5.4.1 Error Analysis

Error analysis of the Secant method is similar to Newton's method. You will perform it, on a problem sheet. You will need the following lemma from real analysis:

Lemma 5.6 If $f : \mathbb{R} \to \mathbb{R}$ has at least two continuous derivatives, and $f(x) = 0$, then for any $a, b \neq 0$,

$$\frac{f(x+a)}{a} - \frac{f(x+b)}{b} = \frac{(a-b)}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)$$

for some ξ in an interval containing x, $x+a$, and $x+b$.

You will show that the Secant method has order-ϕ convergence, for some ϕ between 1.5 and 1.7, and that conditions of Lemma 5.3 are also sufficient to ensure this case, as long as *both* x_0 and x_1 lie in the interval in question. Given a well-behaved function (two continuous derivatives, no double root) the Secant method is superlinear but not as high order as Newton's method. Whether the lower order is worth the cheaper iterative step depends on the cost of computing the derivative: remember that two steps of the secant method will have order $\phi^2 > 2$ convergence (Lemma 4.2), so if we can perform two Secant method steps in the time it takes to do one Newton method step, we are probably winning.

5.5 Other Methods (1 dimension)

There is a connection between Newton's method for root-finding and the Midpoint rule for numerical integration: both approximate the function by its first-order Taylor polynomial. There is a similar connection between the Secant method and the Trapezium rule: both approximate the function by a linear interpolation of two points on the curve.

It is natural to ask whether we can do better with higher-order Taylor approximations or higher-order interpolation. Using the solution of the second-order Taylor approximation as an iteration gives rise to **Halley's method**, which has cubic convergence as long as certain conditions are met. However a second-order polynomial might have zero roots (what to do next?), or two roots (which to pick?) and solving them requires the square-root function (slow). Halley's method avoids these problems by making a further approximation.

The quadratic equivalent of the Secant method is called **Muller's method**. It keeps track of three points on the function, interpolating them by a parabola and following the parabola down to its intersection with the x-axis. Similar problems arise as with

the quadratic Taylor approximation. When the parabola has no real roots, one way to progress is to allow complex roots. Thus Muller's method may find a sequence of complex numbers that (given sufficient conditions) converge to a root. Unfortunately, it will sometimes be a complex root, which may not be what we want. Muller's method has order of convergence approximately 1.84.

There is a better option than either of these higher-order methods. Instead of finding a function $y = ax^2 + bx + c$ that interpolates three known values of $f(x)$, instead fit $x = ay^2 + by + c$. This brilliant idea is called **Inverse Quadratic Interpolation** (IQI) and it is no more difficult than quadratic interpolation, yet it always has exactly one real solution to $y = 0$, and it is algebraically simple to find: $x = c$. Like Muller's method it has order of convergence approximately 1.84.

The gold standard in one-dimensional root-finding is a combination of bisection, the Secant method, and IQI called **Brent's method**. (It is Brent's modification to an idea originally by Dekker.) It is an inclusion method, so it maintains a bracket that contains a root. Given an initial bracket, it *always* converges: linearly if the function is badly-behaved (for example if it has a double root), and with order approximately 1.84 if the function is well-behaved near the root. It does not need to evaluate the derivative at all (but having a continuous third derivative is a hypothesis for the proof of superlinear convergence to be correct). Brent's method works by keeping track of three values of the function, using IQI where possible but falling back to the Secant method if IQI tries to go outside the known bracket, and to bisection if the IQI/Secant step does not provide sufficient reduction in the bracket size. Requiring only about 30 lines of code[3], it is at the core of most one-dimensional root-finding libraries.

5.6 Newton's Method (d dimensions)

Root-finding in d dimensions means solving

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}. \tag{5.5}$$

This is more difficult than the one-dimensional case, and not just because dealing with vectors is more difficult than scalars. In more than one dimension there is (almost) no way to bracket a root, hence nothing as safe as interval bisection to help us find a good starting point for other iterative methods, to fall back on if they fail to converge, or even

[3]See §9.3 of the famous textbook *Numerical Recipes in C* (or *Fortran*, or *C++*, as you prefer).

to prove that a root exists. For now, we will accept this uncertainty, but see Section 5.6.1 for some indication of how this might be dealt with in practice.

To derive a Newton method, break down (5.5) into d simultaneous equations

$$f_1(\boldsymbol{x}) = 0, \ldots, f_d(\boldsymbol{x}) = 0,$$

where $f_i : \mathbb{R}^d \to \mathbb{R}$ are the components of $\boldsymbol{f}$. Given an initial point $\boldsymbol{x_0}$, we approximate *each* component using its 1st-order Taylor polynomial at $\boldsymbol{x_0}$, then solve for all approximate components simultaneously equal to zero:

$$\begin{aligned}
\hat{f}_1(\boldsymbol{x}) = f_1(\boldsymbol{x_0}) + \left(\tfrac{\mathrm{d}f_1}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})\right)^T(\boldsymbol{x} - \boldsymbol{x_0}) = 0 \quad &\Leftrightarrow \quad \left(\tfrac{\mathrm{d}f_1}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})\right)^T(\boldsymbol{x} - \boldsymbol{x_0}) = -f_1(\boldsymbol{x_0}), \\
\vdots \qquad & \qquad \vdots \\
\hat{f}_d(\boldsymbol{x}) = f_d(\boldsymbol{x_0}) + \left(\tfrac{\mathrm{d}f_d}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})\right)^T(\boldsymbol{x} - \boldsymbol{x_0}) = 0 \quad &\Leftrightarrow \quad \left(\tfrac{\mathrm{d}f_d}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})\right)^T(\boldsymbol{x} - \boldsymbol{x_0}) = -f_d(\boldsymbol{x_0}).
\end{aligned}$$

These d equations can be gathered into one $d \times d$ matrix equation

$$\big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_0})\big)(\boldsymbol{x} - \boldsymbol{x_0}) = -\boldsymbol{f}(\boldsymbol{x_0})$$

(recall that, by definition, the rows of $\mathbf{J}(\boldsymbol{f})$ are the vector derivatives of the f_i). This gives rise to an iteration that is often written

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})\big)^{-1}\boldsymbol{f}(\boldsymbol{x_n}). \tag{5.6}$$

This shows how the one-dimensional Newton method is simply a special case of the d-dimensional case, since $\mathbf{J}(\boldsymbol{f})^{-1}$ is the d-dimensional analogue of $1/\frac{\mathrm{d}f}{\mathrm{d}x}$. However, in practice we should *not* invert the Jacobian, because finding a solution $\boldsymbol{\Delta x}$ to

$$\big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})\big)\boldsymbol{\Delta x} = -\boldsymbol{f}(\boldsymbol{x_n}) \tag{5.7}$$

by Gaussian elimination is typically about three times faster than inverting[4]. Instead, solve (5.7) for $\boldsymbol{\Delta x}$, then set $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \boldsymbol{\Delta x}$.

[4]Recall that solving the $d \times d$ system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, and computing $\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{b}$, both take $O(d^3)$ floating-point operations. But solving has a hidden constant about three times smaller, and is also more numerically stable. Recall also: multiplying two $d \times d$ matrices is typically $O(d^3)$ operations although advanced methods, such as **Strassen's algorithm** and its successors, reduce the exponent slightly at the cost of much higher hidden constants; multiplying a $d \times d$ matrix by a vector takes $O(d^2)$, and the scalar product of two vectors is $O(d)$ floating-point operations.

If we reach a point $\boldsymbol{x_n}$ where $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})$ is singular (has zero determinant) then (5.7) may fail to have a solution. In that case, which is the d-dimensional analogue of $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n) = 0$, we must terminate with an error. Otherwise we terminate the iteration under the usual conditions: either we reach a maximum number of steps, or we have converged sufficiently close to a root. This might be specified in terms of forward error $\|\boldsymbol{x_n} - \boldsymbol{x_{n-1}}\| < tol(1 + \|\boldsymbol{x_n}\|)$ (recall that inaccuracy in vectors is measured by the norm: in practice use squared norm to avoid a square-root step in the calculation) or backward error $\|\boldsymbol{f}(\boldsymbol{x_n})\| < tol$.

The computational cost of Newton's method is in evaluating the Jacobian ($O(d^2)$ entries, which might be slow to compute) and solving the system (5.7) ($O(d^3)$ floating-point operations). These are required at each step of the iteration.

Algorithm 5 Newton's method in d-dimensions. Replace $\|\boldsymbol{\Delta x}\| < tol(1 + \|\boldsymbol{x_n}\|)$ by another termination condition, if desired.

---

procedure NEWTON($\boldsymbol{f}$, $\boldsymbol{x_0}$, N, tol)
  $n \leftarrow 0$
  repeat
    Solve for $\boldsymbol{\Delta x}$: $\big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})\big)\boldsymbol{\Delta x} = -\boldsymbol{f}(\boldsymbol{x_n})$ ▷ Raise an error if no solutions!
    $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \boldsymbol{\Delta x}$
    $n \leftarrow n + 1$
  until $n = N \vee \|\boldsymbol{\Delta x}\| < tol(1 + \|\boldsymbol{x_n}\|)$
  return $\boldsymbol{x_n}$ ▷ Should raise a warning if $n = N$
end procedure

Error analysis of the d-dimensional Newton method is beyond the syllabus, but a result is included in Section 5.8. Under smoothness conditions on $\boldsymbol{f}$, and assuming that its Jacobian is nonsingular at $\boldsymbol{x^*}$, there is a region around $\boldsymbol{x^*}$ in which quadratic convergence is guaranteed.

Example 5.7 An interesting two-dimensional root-finding problem arises if we modify **Example 5.4**. Let us suppose that the population consists of *two* types of individual, say type A and type B. At each generation, each individual of type A independently and identically gives rise to some number of type-A offspring (generating function G_{AA}) plus some number of type-B offspring (generating function G_{AB}),

before dying. Similarly, each individual of type B gives rise to type-A offspring (whose number has generating function G_{BA}) and type-B offspring (generating function G_{BB}). What is the probability that the entire population becomes extinct, if we start with one individual?

Using only elementary probability, it can be shown that the probabilities of extinction after starting with one individual of type A, x, and extinction after starting with one individual of type B, y, satisfy the simultaneous nonlinear equations

$$x = G_{AA}(x)G_{AB}(y)$$
$$y = G_{BA}(x)G_{BB}(y).$$

For specified generating functions, Newton's method (applied carefully) is a fast way to solve this system.

5.6.1 Globalizing Newton's Method

We briefly mention a way to make Newton's method less fragile. At each step we compute $\boldsymbol{\Delta x}$, but we *reject* it if $\boldsymbol{x_{n+1}}$ is not a 'better' solution than $\boldsymbol{x_n}$. A sensible way to define 'better' is to insist that

$$\|\boldsymbol{f}(\boldsymbol{x_{n+1}})\| < \|\boldsymbol{f}(\boldsymbol{x_n})\|. \tag{5.8}$$

When this does not hold, we set $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \lambda\boldsymbol{\Delta x}$ for some $\lambda \in (0, 1)$. We will see more of this idea in the next chapter, which is called **backtracking**, because we take a step back towards $\boldsymbol{x_n}$ from $\boldsymbol{x_{n+1}}$.

This is a so-called **damped Newton method**. It can be shown that, for sufficiently small λ, the damped iteration always does satisfy (5.8), but damping every step leads only to linear convergence. There are then algorithms which choose suitable λ at each step, which do achieve quadratic convergence near a root, and which terminate after a finite number of steps instead of oscillating indefinitely. That is to say: they do not guarantee convergence, but they do guarantee that the user knows whether convergence was going to happen or not. Even if it happens, convergence may still be slow if $\boldsymbol{x_0}$ was not close to a root.

5.7 Quasi-Newton Methods (d dimensions)

Newton's method requires a lot of derivatives: all d^2 elements of the Jacobian, which need to be derived algebraically and then evaluated at each step. Naturally we might consider **quasi-Newton methods** which are analogous to the Secant method. Recall that the Secant method can be understood as approximating $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)$ by $\frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})}$. How do we do this in d dimensions? We need to approximate the entire Jacobian $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})$ which, note, depends on n. Thus the approximated Jacobian will evolve through the iterations, preferably without doing too much work at each step.

Let us derive an approximate Jacobian for the n-th step, which we will call $\hat{\mathbf{J}}_{\boldsymbol{n}}$. If $\boldsymbol{y_{n-1}} = \boldsymbol{f}(\boldsymbol{x_{n-1}})$ and $\boldsymbol{y_n} = \boldsymbol{f}(\boldsymbol{x_n})$, then we want $\hat{\mathbf{J}}_{\boldsymbol{n}}$ to satisfy the **secant equation**

$$\boldsymbol{y_n} = \boldsymbol{y_{n-1}} + \hat{\mathbf{J}}_{\boldsymbol{n}}(\boldsymbol{x_n} - \boldsymbol{x_{n-1}}), \tag{5.9}$$

but in more than one dimension this does not determine $\hat{\mathbf{J}}_{\boldsymbol{n}}$: it is a system of d equations, but $\hat{\mathbf{J}}_{\boldsymbol{n}}$ has d^2 entries to find! We need to add further constraints on $\hat{\mathbf{J}}_{\boldsymbol{n}}$.

One sensible way to constrain $\hat{\mathbf{J}}_{\boldsymbol{n}}$ is to note that (5.9) gives no information about $\hat{\mathbf{J}}_{\boldsymbol{n}}\boldsymbol{z}$ for $\boldsymbol{z}$ orthogonal to $\boldsymbol{x_n} - \boldsymbol{x_{n-1}}$ (i.e. $\boldsymbol{z}^T(\boldsymbol{x_n} - \boldsymbol{x_{n-1}}) = 0$). We could therefore specify that $\hat{\mathbf{J}}_{\boldsymbol{n}}$ has no reason to change from $\hat{\mathbf{J}}_{\boldsymbol{n-1}}$ when applied to such vectors:

$$\hat{\mathbf{J}}_{\boldsymbol{n}}\boldsymbol{z} = \hat{\mathbf{J}}_{\boldsymbol{n-1}}\boldsymbol{z}, \quad \text{whenever} \quad \boldsymbol{z}^T(\boldsymbol{x_n} - \boldsymbol{x_{n-1}}) = 0. \tag{5.10}$$ △

This supplies enough constraints to determine $\hat{\mathbf{J}}_{\boldsymbol{n}}$. We would need to couple this with a choice of initial $\hat{\mathbf{J}}_{\mathbf{0}}$.

An alternative way to constrain $\hat{\mathbf{J}}_{\boldsymbol{n}}$ is to say that we want it to be 'close to' $\hat{\mathbf{J}}_{\boldsymbol{n-1}}$. The reasoning is that, once we get close to the root, $\boldsymbol{x_n}$ should not be very different from $\boldsymbol{x_{n-1}}$, and hence the true Jacobians will not be very different when evaluated there. A principle we could apply is

$$\text{minimize } \|\hat{\mathbf{J}}_{\boldsymbol{n}} - \hat{\mathbf{J}}_{\boldsymbol{n-1}}\|, \tag{5.11}$$ △

where $\|\cdot\|$ is a **matrix norm**. There are many matrix norms, and in this case we choose the **Frobenius norm** $\|\mathbf{A}\| = \sqrt{\sum_{i,j} a_{ij}^2}$.

Happily, the solutions to (5.10) combined with (5.9), and (5.11) subject to (5.9), are identical. Write $\boldsymbol{\Delta y} = \boldsymbol{y_n} - \boldsymbol{y_{n-1}}$ and $\boldsymbol{\Delta x} = \boldsymbol{x_n} - \boldsymbol{x_{n-1}}$, then

$$\hat{\mathbf{J}}_{\boldsymbol{n}} = \hat{\mathbf{J}}_{\boldsymbol{n-1}} + \frac{\boldsymbol{\Delta y} - \hat{\mathbf{J}}_{n-1}\boldsymbol{\Delta x}}{\|\boldsymbol{\Delta x}\|^2}\boldsymbol{\Delta x}^T. \tag{5.12}$$ △

Updating the Jacobian in this way, then using the quasi-Newton step

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \boldsymbol{\Delta x}, \quad \text{where} \quad \hat{\mathbf{J}}_{\boldsymbol{n}} \boldsymbol{\Delta x} = -\boldsymbol{f}(\boldsymbol{x_n}) \tag{5.13}$$

is known as **Broyden's method**, and it is one of a family of quasi-Newton methods invented by Charles Broyden in the 1960s. At each step of the iteration we need only evaluate $\boldsymbol{f}$ once, update $\boldsymbol{\Delta x}$, $\boldsymbol{\Delta y}$, and compute $\hat{\mathbf{J}}_{\boldsymbol{n}}$.

As with Newton's method, we need a starting point $\boldsymbol{x_0}$, but we also need a starting Jacobian $\hat{\mathbf{J}}_{\mathbf{0}}$. Absent other information, this might be $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_0})$, which unfortunately does require us to compute the Jacobian of $\boldsymbol{f}$ (and evaluate it just the once). We cannot use a second point $\boldsymbol{x_1}$ to determine $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_0})$, as the Secant method did, because then $\mathbf{J}(\boldsymbol{f})$ is underdetermined as described above. To avoid computing the Jacobian of $\boldsymbol{f}$ at all, in some cases it is safe to use a generic starting Jacobian $\hat{\mathbf{J}}_{\mathbf{0}} = -\mathbf{I}$, or a suitably-scaled version of this. $\hat{\mathbf{J}}_{\boldsymbol{n}}$ will still become a good approximation to $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})$ after $O(d)$ iterations. That is beyond the scope of this course.

As described in Algorithm 6, each step of Broyden's method requires $O(d^3)$ work in the dominant step, which is solving the system in (5.13). Compared with Newton's method, we avoided evaluating the Jacobian at each stage (and perhaps computing it at all, if we began with a generic $\hat{\mathbf{J}}_{\mathbf{0}}$), at the cost of more complex formulae, but did not change the complexity of the matrix operations. There are some efficiency improvements that can be made to Broyden's method, but perhaps the most important is to avoid this dominant step of solving a $d \times d$ system. Instead of updating $\hat{\mathbf{J}}_{\boldsymbol{n}}$ at each iteration, equivalent formulae can be found that keep track of $\hat{\mathbf{J}}_{\boldsymbol{n}}^{-1}$ (we do need to invert the very first Jacobian $\hat{\mathbf{J}}_{\mathbf{0}}$, unless it is generic, but this happens only once). You will explore this in the tutorial exercises.

Like the Secant method in 1 dimension, Broyden's method converges if started suitably close to an isolated root $\boldsymbol{x^*}$, but not as fast as Newton's method. Sufficient conditions for superlinear convergence have been found, but the order of convergence depends on the nature of the problem. If d is large, and the efficient version of the algorithm that tracks $\hat{\mathbf{J}}_{\boldsymbol{n}}^{-1}$ is used, the need for more iteration steps (in Broyden's method over Newton's) is easily outweighed by the *much* faster speed of computing each step. Termination conditions can be similar to other d-dimensional methods, in terms of the norm of the change at each step, or the backward error.

As with Newton's method in d dimensions, convergence is not guaranteed unless $\boldsymbol{x_0}$ is sufficiently close to a root, and it is difficult to determine in advance whether a given

Algorithm 6 Broyden's method. Replace $\|\boldsymbol{\Delta x}\| < tol(1 + \|\boldsymbol{x_n}\|)$ by another termination condition, if desired.

procedure BROYDEN($\boldsymbol{f}$, $\boldsymbol{x_0}$, N, tol)
  $\boldsymbol{y_0} \leftarrow \boldsymbol{f}(\boldsymbol{x_0})$
  $\hat{\mathbf{J}}_\mathbf{0} \leftarrow \mathbf{J}(\boldsymbol{f})(\boldsymbol{x_0})$ ▷ Or allow the user to supply an initial $\hat{\mathbf{J}}_\mathbf{0}$
  Solve for $\boldsymbol{\Delta x}$: $\hat{\mathbf{J}}_\mathbf{0}\boldsymbol{\Delta x} = -\boldsymbol{y_0}$ ▷ Should raise an error if no solutions!
  $\boldsymbol{x_1} \leftarrow \boldsymbol{x_0} + \boldsymbol{\Delta x}$
  $n \leftarrow 1$
  repeat
    $\boldsymbol{y_n} \leftarrow \boldsymbol{f}(\boldsymbol{x_n})$
    $\boldsymbol{\Delta y} \leftarrow \boldsymbol{y_n} - \boldsymbol{y_{n-1}}$
    $\boldsymbol{\Delta \mathbf{J}} \leftarrow \dfrac{\boldsymbol{\Delta y} - \hat{\mathbf{J}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}}{\|\boldsymbol{\Delta x}\|^2}\boldsymbol{\Delta x}^T.$
    $\hat{\mathbf{J}}_\boldsymbol{n} \leftarrow \hat{\mathbf{J}}_{\boldsymbol{n-1}} + \boldsymbol{\Delta \mathbf{J}}$
    Solve for $\boldsymbol{\Delta x}$: $\hat{\mathbf{J}}_\boldsymbol{n}\boldsymbol{\Delta x} = -\boldsymbol{y_n}$ ▷ Should raise an error if no solutions!
    $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \boldsymbol{\Delta x}$
    $n \leftarrow n + 1$
  until $n = N \vee \|\boldsymbol{\Delta x}\| < tol(1 + \|\boldsymbol{x_n}\|)$
  return $\boldsymbol{x_n}$ ▷ Should raise a warning if $n = N$
end procedure

$\boldsymbol{x_0}$ is 'sufficiently close'. There are backtracking techniques to make Broyden's method more robust to bad starting points. They are generally do not guarantee convergence, but make failure less likely. There are also **limited-memory methods** which avoid having to store the Jacobian $\hat{\mathbf{J}}_{\boldsymbol{n}}$ (or inverse $\hat{\mathbf{J}}_{\boldsymbol{n}}^{-1}$): bear in mind that these matrices are $O(d^2)$ in size, which might be impractical to store if the dimension is, say, one million. Limited-memory methods store only some recent vectors which are sufficient to perform calculations with an approximate Jacobian, reducing the space requirement to $O(d)$.

5.8 Bonus Mathematics: Convergence of Newton's Method in d dimensions

This is not part of the syllabus.

Here is an elementary proof of quadratic convergence for the d-dimensional Newton root-finding method; I deliberately phrased it to parallel Lemma 5.3. This section is probably best read in your second or third year, if you find an interest in numerical analysis.

We first need to define a different matrix norm from the Frobenius norm we used in Section 5.7:

Definition If $\mathbf{A}$ is a matrix (not necessarily square) then

$$\|\mathbf{A}\| = \max_{\|\boldsymbol{x}\|=1} \|\mathbf{A}\boldsymbol{x}\|$$

is the **matrix 2-norm** of $\mathbf{A}$.

It has the property that, by definition,

$$\|\mathbf{A}\boldsymbol{x}\| \leq \|\mathbf{A}\|\|\boldsymbol{x}\|. \tag{5.14}$$

We can connect the matrix 2-norm with the eigenvalues of a matrix, as follows.

Lemma 5.8 Let $\lambda_{max}(\mathbf{A})$ denote the largest absolute eigenvalue of $\mathbf{A}$. Then

(i) For symmetric square $\mathbf{A}$,

$$|\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}| \leq \lambda_{max}(\mathbf{A})\|x\|^2. \tag{5.15}$$

(ii) For arbitrary $\mathbf{A}$,

$$\|\mathbf{A}\| = \sqrt{\lambda_{max}(\mathbf{A}^T\mathbf{A})}. \tag{5.16}$$

Proof (i) was proved on the first exercise sheet (after replacing 'eigenvalue' by 'absolute eigenvalue'). Furthermore, the inequality is an equality if $\boldsymbol{x}$ is an eigenvector with absolute eigenvalue λ_{max}.

(ii) follows from (i), as $\mathbf{A}^T\mathbf{A}$ is always square and symmetric, and

$$\max_{\|\boldsymbol{x}\|=1} \|\mathbf{A}\boldsymbol{x}\| = \sqrt{\max_{\|\boldsymbol{x}\|=1} \boldsymbol{x}^T\mathbf{A}^T\mathbf{A}\boldsymbol{x}} = \sqrt{\lambda_{max}(\mathbf{A}^T\mathbf{A})}.$$

(The maximum is attained when $\boldsymbol{x}$ is a unit eigenvector with absolute eigenvalue λ_{max}.) ■

Now we can state a convergence result for the d-dimensional Newton method.

Theorem 5.9 Let $\boldsymbol{f}$ have components $f_1, \ldots, f_d$, where each f_i has at least two continuous partial derivatives, and $\boldsymbol{f}(\boldsymbol{x}^*) = \mathbf{0}$. Write $\boldsymbol{\epsilon_n} = \boldsymbol{x_n} - \boldsymbol{x}^*$.

(i) If $\boldsymbol{x_n} \in D = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| < c\}$ and

$$A(c) = \max_{\boldsymbol{x}\in D} \left\| \big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x})\big)^{-1} \right\| \cdot \max_{\boldsymbol{x}\in D} \sqrt{\lambda^2_{max}\big(\mathbf{H}(f_1)(\boldsymbol{x})\big) + \cdots + \lambda^2_{max}\big(\mathbf{H}(f_d)(\boldsymbol{x})\big)}$$

then $\|\boldsymbol{\epsilon_{n+1}}\| \leq \frac{A(c)}{2}\|\boldsymbol{\epsilon_n}\|^2$.

(ii) If $\boldsymbol{x_0} \in D$, and further $\frac{cA(c)}{2} < 1$, then Newton's method converges at least quadratically to $\boldsymbol{x}^*$.

(iii) Such a c exists if $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x}^*)$ is nonsingular.

Notice the similarity with Lemma 5.3: the sphere D plays the role of the interval I, the maximum on $\left\|\big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x})\big)^{-1}\right\|$ is analogous to the minimum on $\frac{\mathrm{d}f}{\mathrm{d}x}$, and the maximum on the eigenvalues of the Hessians corresponds to the maximum of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$.

Proof Let us apply Taylor's theorem with second-order remainder term to each component f_i of $\boldsymbol{f}$:

$$f_i(\boldsymbol{x^*}) = f_i(\boldsymbol{x}) + \Big(\tfrac{\mathrm{d}f_i}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x})\Big)^T(\boldsymbol{x^*}-\boldsymbol{x}) + \tfrac{1}{2}(\boldsymbol{x^*}-\boldsymbol{x})^T\mathbf{H}(f_i)(\boldsymbol{x'})(\boldsymbol{x^*}-\boldsymbol{x}),$$

for some $\boldsymbol{x'}$. You might notice that this is the same Taylor expansion that motivates the Newton method in the first place, but with an explicit remainder term.

Since the LHS is zero, and using (5.15),

$$\left|f_i(\boldsymbol{x}) - \Big(\tfrac{\mathrm{d}f_i}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x})\Big)^T(\boldsymbol{x}-\boldsymbol{x^*})\right| \le \tfrac{1}{2}\lambda_{max}\big(\mathbf{H}(f_i)(\boldsymbol{x'})\big)\|\boldsymbol{x}-\boldsymbol{x^*}\|^2.$$

This establishes a bound for each component of $\boldsymbol{f}$. If both $\boldsymbol{x}, \boldsymbol{x^*} \in D$ then squaring these bounds, summing for $i = 1, \ldots, d$, then taking square root, gives

$$\begin{aligned}&\big\|\boldsymbol{f}(\boldsymbol{x}) - \big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x})\big)(\boldsymbol{x}-\boldsymbol{x^*})\big\| \le \\ &\quad\frac{1}{2}\left(\max_{\boldsymbol{x}\in D}\sqrt{\lambda^2_{max}\big(\mathbf{H}(f_1)(\boldsymbol{x})\big) + \cdots + \lambda^2_{max}\big(\mathbf{H}(f_d)(\boldsymbol{x})\big)}\right)\big\|\boldsymbol{x}-\boldsymbol{x^*}\big\|^2. \qquad (5.17)\end{aligned}$$

Proof of (i)

We will abbreviate $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x_n})$ as $\mathbf{J}$. Let us assume for a moment that $\mathbf{J}$ is invertible (we will prove it in (iii), for sufficiently small c) so that the Newton iteration

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \mathbf{J}^{-1}\boldsymbol{f}(\boldsymbol{x_n})$$

is well defined. Then

$$\begin{aligned}\|\boldsymbol{x_{n+1}} - \boldsymbol{x^*}\| &= \big\|\boldsymbol{x_n} - \boldsymbol{x^*} - \mathbf{J}^{-1}\boldsymbol{f}(\boldsymbol{x_n})\big\| \\ &= \big\|\mathbf{J}^{-1}\big(\mathbf{J}(\boldsymbol{x_n}-\boldsymbol{x^*}) - \boldsymbol{f}(\boldsymbol{x_n})\big)\big\| \\ &\le \big\|\mathbf{J}^{-1}\big\|\big\|\mathbf{J}(\boldsymbol{x_n}-\boldsymbol{x^*}) - \boldsymbol{f}(\boldsymbol{x_n})\big\| && \big\{\,(5.14)\,\big\} \\ &\le \frac{A(c)}{2}\|\boldsymbol{x_n}-\boldsymbol{x^*}\|^2. && \big\{\,(5.17)\text{ and definition of } A(c)\,\big\}\end{aligned}$$

Proof of (ii)

Identical to the proof of Lemma 5.3(ii).

Proof of (iii)

Note that eigenvalues are a continuous function of the elements of a matrix, and by assumption the elements of $\mathbf{J}(\boldsymbol{f})$ and $\mathbf{H}(f_i)$ are all continuous functions. It follows that

$$\max_{\boldsymbol{x}\in D} \sqrt{\lambda_{max}^2\big(\mathbf{H}(f_1)(\boldsymbol{x})\big) + \cdots + \lambda_{max}^2\big(\mathbf{H}(f_d)(\boldsymbol{x})\big)}$$

is finite and tends to

$$\sqrt{\lambda_{max}^2\big(\mathbf{H}(f_1)(\boldsymbol{x}^*)\big) + \cdots + \lambda_{max}^2\big(\mathbf{H}(f_d)(\boldsymbol{x}^*)\big)}$$

as $c \to 0$.

If $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x}^*)$ is nonsingular then none of its eigenvalues is zero. It follows that the same is true for $\mathbf{J}(\boldsymbol{f})(\boldsymbol{x})$ for $\boldsymbol{x} \in D$ for sufficiently small c, so the Newton iteration is indeed well-defined in a neighbourhood of $\boldsymbol{x}^*$. Furthermore,

$$\max_{\boldsymbol{x}\in D} \big\| \big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x})\big)^{-1} \big\| \to \big\| \big(\mathbf{J}(\boldsymbol{f})(\boldsymbol{x}^*)\big)^{-1} \big\|$$

as $c \to 0$. Multiplying by c, $cA(c) \to 0$ as $c \to 0$, so the condition $\frac{cA(c)}{2} < 1$ holds for sufficiently small c. ■

Practice Questions

Use a calculator or computer to do the iterations.

5.1 Find a starting bracket for a root of these functions, then calculate how many steps of interval bisection will be necessary to locate it within 10^{-6}. m is a large integer.

(i) $f(x) = \prod_{i=1}^{m} |x - i| - \frac{m!}{2}$, (ii) $g(x) = \frac{\ln x}{(1-\frac{1}{x})^2} - m$, (iii) $h(x) = e^{\frac{x}{100}} - (x+100)^{100}$.

Hint for (ii): a useful inequality is $1 - \frac{1}{x} \leq \ln x \leq x - 1$.
Do you foresee any difficulties implementing interval bisection in the case of (iii)?

5.2 An engineer suggests using Newton's method to find a root of

$$g(x) = (x-1)^{51}(x+1)^{51},$$

starting at $x_0 = 1.1$. Compute the first four iterations. Setting aside that we can see the roots are ± 1, why is this iteration a bad idea? Estimate roughly how many steps it would take to find an answer accurate to within 10^{-6} using a) Newton's method, and b) interval bisection starting with the bracket $(0, 2)$.

5.3 All of the following functions have a root at $\sqrt{2}$. Write down the Newton iteration for each, simplifying where possible. Does Lemma 5.3 guarantee that it converges quadratically to $\sqrt{2}$ if $x_0 = 1$?

(i) $f(x) = x^2 - 2$, (ii) $f(x) = \frac{1}{x^2} - \frac{1}{2}$, (iii) $f(x) = x - \frac{2}{x}$,

(iv) $f(x) = \frac{x^2}{2} - 1$, (v) $f(x) = x^4 - 4$, (vi) $f(x) = x^4 - 4x^2 + 4$.

5.4 Why might it be a bad idea, in practice, to simplify the iterative step of the Secant method to the equivalent formula

$$x_{n+1} = \frac{x_{n-1}f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}\,?$$

5.5 'Find the point on the line $y = \ln x$ that is closest to the origin.' Show that this happens when $x^2 + \ln x = 0$ and use the Secant method, with $x_0 = 1$ and $x_1 = 2$, to approximate the root.

5.6 Use three steps of the 2D Newton method, starting at $(0, 0)$, to approximate the solution of the simultaneous equations

$$x + e^x = 3 \quad \wedge \quad x + y + e^y = 5.$$

Answers to Chapter 5 Practice Questions

5.1 (i) $f(m) = -\frac{m!}{2} < 0$; $f(m+1) = \frac{m!}{2} > 0$, so $(m, m+1)$ is a bracket.
(ii) $\frac{x}{x-1} = \frac{1}{1-\frac{1}{x}} \le \frac{\ln x}{(1-\frac{1}{x})^2} \le \frac{x-1}{(\frac{x-1}{x})^2} = \frac{x^2}{x-1}$. Hence $\left(\frac{m}{m-1}, \sqrt{m}\right)$ is a bracket. (The upper bound could be a lot tighter).
(iii) $f(0) < 0$ and it seems likely that $f(x) > 0$ for sufficiently large x. To find it, try brackets of the form $(0, 2^k)$ until one works: the first that does is $(0, 131072)$. It is quite difficult to overflow double-precision arithmetic, but $(x+100)^{100}$ will do it for $x \ge 1110$. Avoid this by using $e^{\frac{x}{100}} < (x+100)^{100} \Leftrightarrow \frac{x}{100} < 100 \ln(x+100)$.

5.2 $x_{n+1} = x_n - \frac{x_n^2 - 1}{102 x_n}$. The sequence begins 1.1, 1.098128, 1.096290, 1.094485, 1.092712. Since $102x_n \approx 100$, the sequence is decreasing by less than 0.002 at each step. It will be at least 50 iterations before it gets close to 1 (in fact it slows down, taking 585 to get within 10^{-6}). Newton's method fails to have quadratic convergence because $\frac{\mathrm{d}g}{\mathrm{d}x}$ is zero at the root. Indeed, the first 50 derivatives are all zero. Interval bisection, starting on $(0, 2)$, will find an interval of width 10^{-6} in $\lceil \log_2 \frac{2}{2 \cdot 10^{-6}} \rceil = 20$ steps.

5.3 Let $I = (\sqrt{2} - c, \sqrt{2} + c)$, assuming $c < \sqrt{2}$ so that the interval avoids zero, and $A(c) = \max_I \left|\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right| / \min_I \left|\frac{\mathrm{d}f}{\mathrm{d}x}\right|$. Lemma 5.3 guarantees quadratic convergence if $\frac{cA(c)}{2} < 1$.

	iteration	$\min_I \left\|\frac{\mathrm{d}f}{\mathrm{d}x}\right\|$	$\max_I \left\|\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right\|$	$(\frac{cA(c)}{2} < 1$ for $c = \sqrt{2} - 1)$?
*(i)	$x_{n+1} = \frac{1}{x_n} + \frac{x_n}{2}$	$2\sqrt{2} - 2c$	2	✓
(ii)	$x_{n+1} = \frac{6x_n - x_n^3}{4}$	$\frac{2}{(c+\sqrt{2})^3}$	$\frac{6}{(c-\sqrt{2})^4}$	×
(iii)	$x_{n+1} = \frac{4x_n}{x_n^2 + 2}$	$1 + \frac{2}{(c+\sqrt{2})^2}$	$\frac{4}{(\sqrt{2}-c)^3}$	✓
(iv)	exactly the same as (i)			✓
(v)	$x_{n+1} = \frac{3x_n^4 + 4}{4x_n^3}$	$4(\sqrt{2} - c)^3$	$12(\sqrt{2} + c)^2$	×
**(vi)	$x_{n+1} = \frac{x_n^2 - 2}{4x_n}$	0	irrelevant	×

*this is the iteration from **Example 4.3**. **since $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) = 0$ (a double root), $A(c) = \infty$ for any c: no quadratic convergence for *any* initial value.

5.4 Risk of catastrophic cancellation, since the values in the numerator are both close to zero. In the original iteration, the quotient is added to x_n, which unless $x^* \approx 0$ should mean that the cancellation errors are small in relation to the size of x_{n+1}. In this iteration, there is no further addition, so very large relative errors can remain.

5.5 Let $D(x)$ be the squared distance from $(0, 0)$ to $(x, \ln x)$: $D(x) = x^2 + \ln^2 x$, so $\frac{\mathrm{d}D}{\mathrm{d}x} = 2x + 2\frac{\ln x}{x} = 0$ if $x^2 + \ln x = 0$. The secant method sequence $(x_0, \ldots)$ goes (1, 2, 0.729228, 0.667918, 0.652889, 0.652919, 0.652919), at which point we terminate the iteration. (The root is indeed 0.652919, to 6 d.p.)

5.6 Write $\boldsymbol{f}(x, y) = \begin{pmatrix} x + e^x - 3 \\ x + y + e^y - 5 \end{pmatrix}$. $\mathbf{J}(\boldsymbol{f}) = \begin{pmatrix} 1 + e^x & 0 \\ 1 & 1 + e^y \end{pmatrix}$ which has inverse $\mathbf{J}^{-1} = \frac{1}{(1+e^x)(1+e^y)} \begin{pmatrix} 1 + e^y & 0 \\ -1 & 1 + e^x \end{pmatrix}$. The Newton iteration is $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \mathbf{J}^{-1}\boldsymbol{f}(\boldsymbol{x})$ which proceeds (to 5 d.p.) $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 1 \\ 1.5 \end{pmatrix}$, $\begin{pmatrix} 0.80682 \\ 1.17373 \end{pmatrix}$, $\begin{pmatrix} 0.79213 \\ 1.12652 \end{pmatrix}$. In fact the root is, to 5 d.p., $\begin{pmatrix} 0.79206 \\ 1.12567 \end{pmatrix}$.

Chapter 6

Algorithms for Optimization

A major application of numerical methods is in finding the **minimum** of a function. In this chapter we will only consider unconstrained optimization, although numerical methods for constrained optimization are equally important. In Chapter 2 we saw that a local minimum of $f : \mathbb{R}^d \to \mathbb{R}$ occurs only when $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mathbf{0}$, but that more is needed to rule out a saddle point or local maximum: f having a positive definite Hessian at the stationary point, or being convex, are sufficient conditions.

In applications, however, finding solutions of $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mathbf{0}$ can be impossible analytically. Such applications include most of machine learning, image and signal processing, and resource allocation. In each case a **loss function**, in some fields called **energy**, is to be minimized, but the minimum has no closed form. So we turn to iterative methods.

We must be modest in our ambitions. A **local minimum** is all we expect to achieve. For convex functions, a local minimum is the global minimum and all is well, but for non-convex functions it can be extremely difficult to find the global minimum. (Some heuristics involve starting many local-minimum searches from a wide variety of initial points, or adding noise to the iterations to try to 'jump' out of a bad local minimum.) We will assume that the objective function f is bounded below (in practice this is something you should check), and many methods will assume that f has a continuous derivative and perhaps Hessian.

The astute reader might ask: why not simply apply the d-dimensional root-finding methods of the previous chapter to the simultaneous equations $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mathbf{0}$? We will come to this in Section 6.4.

6.1 When to Stop

Once again we will be iterating, in the case of d-dimensional optimization giving a sequence of vectors $(\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots)$ that hopefully converges to a local minimum $\boldsymbol{x^*}$. We need to give termination criteria, and things are not quite the same as for root-finding. First, because we are not trying to find a particular value of f, we cannot give a backward error criterion in terms of $f(\boldsymbol{x_n})$. Instead, we might have conditions on the (relative or absolute) amount that $\boldsymbol{x_n}$ changed, the amount that $f(\boldsymbol{x_n})$ changed, or the gradient of $f(\boldsymbol{x_n})$. Picking good termination criteria is tricky, but some common conditions include:

(i) $\|\boldsymbol{x_n} - \boldsymbol{x_{n-1}}\| < tol(1 + \|\boldsymbol{x_n}\|)$,
the location of the approximate minimum did not change much (absolutely or relatively) on the last iteration.

(ii) $|f(\boldsymbol{x_n}) - f(\boldsymbol{x_{n-1}})| < tol(1 + |f(\boldsymbol{x_n})|)$,
the value of the approximate minimum did not change much (absolutely or relatively) on the last iteration.

(iii) $\|\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})\| < tol$,
the gradient of the function, for methods that compute it (Sections 6.3, 6.4, and 6.5) was sufficiently close to zero. Under some conditions this is also an estimate of **forward error**.

(iv) $n = N$,
we have run out of time (and we should warn the user that their specified tolerance was not met).

(v) an iteration step was ill-defined (we must raise an error).

Whether to pick (i), (ii), (iii), or a combination, depends on the requirements of the application and may depend on the properties of f. Condition (ii) is considered rather risky, as some functions can decrease ever more slowly even as they diverge to $-\infty$. Condition (i) is good for Newton-like methods, but in other methods it can simply be an indication that the step size (see Section 6.3) is too small. When the gradient is available, condition (iii) can be a good choice. We should always include conditions (iv) and (v).

Another difference from root-finding is the tolerance. How precisely can we approximate the *location* of the minimum? Less than you might imagine! Consider the one-dimensional case: whenever

$$\frac{|f(x) - f(x^*)|}{|f(x^*)|} < \epsilon$$

(where ϵ is machine epsilon) the values of $f(x)$ and $f(x^*)$ are, or might be, indistinguishable as floating-point numbers. Since $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) = 0$, the left-hand side is

$$\frac{\frac{1}{2}(x - x^*)^2 \left|\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)\right|}{|f(x^*)|},$$

for some $\xi \in (x, x^*)$. So $f(x)$ and $f(x^*)$ may be indistinguishable when

$$\frac{|x - x^*|}{|x^*|} < \sqrt{\epsilon} \sqrt{\frac{2|f(x^*)|}{|x^*|^2 \left|\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)\right|}}.$$

$\triangle$ (know that it is $O(\sqrt{\epsilon})$)

The second square root is practically a constant – since $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\xi)$ will be almost exactly $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^*)$ – which depends on the problem we are solving. But apart from esoteric cases it will not be many orders of magnitude different from 1. Thus we cannot distinguish $f(x)$ and $f(x^*)$ better than machine roundoff if $|x - x^*|$ is of the order of $\sqrt{\epsilon}$. The same applies, with more involved linear algebra, in d dimensions.

So a sensible default value of *tol* for optimization is $\sqrt{\epsilon}$, not $O(\epsilon)$. If working in double-precision arithmetic, do not expect to find the location of a minimum with relative error better than about 10^{-8}. Knowing this limitation can save a lot of pointless iterations.

6.2 Golden Section Search (1 dimension)

In one-dimensional root-finding we had the powerful concept of a **bracket**, an interval in which a root must lie. The same exists for minimization. Here, a **bracket** is a triple where the value of f is lower in the middle than at either end:

$$(a, b, c) \text{ with } a < b < c,\ f(b) < f(a),\ f(b) < f(c).$$

If (a, b, c) is a bracket then at least one local minimum must occur somewhere between a and c. (For this section we will ignore the possibility that $f(x) = f(y)$ for *any* distinct points x and y: dealing with such cases is left as an exercise.)

We can refine a bracket using a method similar to interval bisection. If $z \in (a, b)$ and $f(z) < f(b)$ then (a, z, b) is a smaller bracket; conversely if $f(z) > f(b)$ then (z, b, c) is a smaller bracket. A symmetrical argument holds if we pick $z \in (b, c)$ instead. Then we iterate to produce smaller and smaller brackets.

How best to pick z? We want to subdivide the largest part of the original bracket, so $z \in (a, b)$ if $b - a > c - b$, otherwise $z \in (b, c)$. Suppose the latter (the other case is symmetrical), and let

$$c - z = \phi(c - b), \tag{6.1}$$

so that z divides the interval (b, c) in the ratio $1 - \phi : \phi$. The length of the new bracket will be either $z - a$ or $c - b$, but we cannot know which until we pick z and evaluate $f(z)$. So that bad luck does not penalize us, we ensure that these lengths are equal:

$$z - a = c - b. \tag{6.2}$$

This is enough to determine z, but (depending on the initial choice of (a, b, c)) it can lead to brackets that do not shrink as fast as they might. Instead, we make the further assumption that the *previous* bracket (a, b, c) had the same proportions as (b, z, c), i.e.

$$c - b = \phi(c - a). \tag{6.3}$$

Combining equations (6.1), (6.2), and (6.3), we have

$$c - b = \phi(c - a) = \phi(c - z + z - a) = (\phi^2 + \phi)(c - b)$$

and hence

$$\phi = \frac{\sqrt{5} - 1}{2} \approx 0.618$$

(using $0 < \phi < 1$ to eliminate the other root). This says that we should divide the bigger interval (a, b) or (c, b) in the ratio $1 - \phi : \phi$, approximately $0.382 : 0.618$, to find the location of z. The number $\frac{\sqrt{5}-1}{2}$ is sometimes called the **golden ratio**, hence the name **golden section search**.

For starting an iteration using the above bracket-shrinking process, the starting bracket (a_0, b_0, c_0) would ideally also have $b_0 - a_0 : c_0 - b_0$ in the same ratio. But even if it does not, choosing subsequent z using $z = a_n + \phi(b_n - a_n)$ or $z = c_n - \phi(c_n - b_n)$ will still give a sequence of brackets whose ratio rapidly approaches the optimal.

A careful implementation of golden section search needs only one evaluation of f (at z) at each step. The value of $f(b)$ can be remembered from the previous step. No derivatives are evaluated, and f need not even be differentiable. (It does need to be continuous.) The method has linear convergence because the size of the new bracket $c_{n+1} - a_{n+1} = c_n - b_n = \phi(c_n - a_n)$. Convergence is not quite as fast as interval bisection

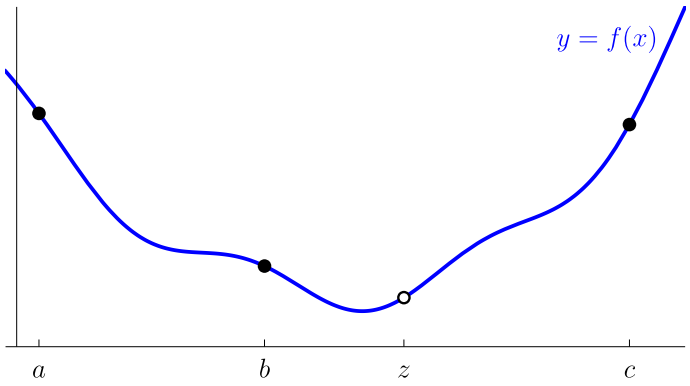


Figure 6.1: Golden section search. (a, b, c) is a bracket for a minimum; because $f(z) < f(b)$, the new bracket is (b, z, c). The ratio of the differences $(b - a) : (c - b)$ and $(z - b) : (c - z)$ are both $1 - \phi : \phi$. z is chosen symmetrically if $b - a > c - b$.

for a root, where the bracket shrinks by a factor of 0.5 on each step; here it shrinks by a factor of approximately 0.618. Golden section search normally terminates when the width of the interval is less than *tol* times the middle point b_n. Remember again: *tol* should generally not be less than $\sqrt{\epsilon}$.

Just like interval bisection, the method does need the user to find a starting bracket. Sometimes this is easy to find algebraically. Otherwise we might try an expanding search for a bracket: for example, if we can prove that the minimum must be at some positive x, we can test brackets of the form $\left(0, (\phi{+}1)^n, (\phi{+}1)^{n+1}\right)$, or $\left((\phi{+}1)^{n-1}, (\phi{+}1)^n, (\phi{+}1)^{n+1}\right)$ – these have the golden ratio property, and testing a new bracket requires only one further evaluation of f – until we find one. There are many other possible approaches to finding an initial bracket.

Finally, we should mention that it does not matter if the ratio ϕ used is not exactly $\frac{\sqrt{5}-1}{2}$. It changes the rate of convergence only slightly if we use an approximation such as 0.618.

Algorithm 7 Golden section search. Replace $|c_n - a_n| < tol|b_n|$ by another termination condition, if desired.

procedure GOLDENSECTION(f, a_0, b_0, c_0, N, tol)
  require $a_0 < b_0 < c_0 \wedge f(a_0) > f(b_0) < f(c_0)$ ▷ Must begin with a bracket
  $f_b \leftarrow f(b_0)$
  $n \leftarrow 0$
  repeat
    if $c_n - b_n > b_n - a_n$ **then** ▷ (b_n, c_n) was the bigger interval
      $z \leftarrow c_n - \frac{\sqrt{5}-1}{2}(c_n - b_n)$
      $f_z \leftarrow f(z)$
      if $f_b < f_z$ **then**
        $(a_{n+1}, b_{n+1}, c_{n+1}, f_b) \leftarrow (a_n, b_n, z, f_b)$ ▷ New bracket (a_n, b_n, z)
      else
        $(a_{n+1}, b_{n+1}, c_{n+1}, f_b) \leftarrow (b_n, z, c_n, f_z)$ ▷ New bracket (b_n, z, c_n)
      end if
    else ▷ (a_n, b_n) was the bigger interval
      $z \leftarrow a_n + \frac{\sqrt{5}-1}{2}(b_n - a_n)$
      $f_z \leftarrow f(z)$
      if $f_b < f_z$ **then**
        $(a_{n+1}, b_{n+1}, c_{n+1}, f_b) \leftarrow (z, b_n, c_n, f_b)$ ▷ New bracket (z, b_n, c_n)
      else
        $(a_{n+1}, b_{n+1}, c_{n+1}, f_b) \leftarrow (a_n, z, b_n, f_z)$ ▷ New bracket (a_n, z, b_n)
      end if
    end if
    $n \leftarrow n + 1$
  until $n = N \vee |c_n - a_n| < tol|b_n|$
  return b_n
end procedure

Example 6.1 Golden section search is ideal for finding the minimum of a non-differentiable function like

$$f(x) = \max\Big(\tfrac{10}{1+10\sin x}, 2-(x-2)^4, (x-1)^3\Big).$$

There are superlinear improvements to golden section search, which are also derivative-free one-dimensional minimization algorithms. One is **successive parabolic interpolation**, which uses a quadratic function to interpolate f at a, b, and c, setting z as the minimum of the quadratic. It has some drawbacks that you will investigate in the exercises. There is a version of **Brent's method** for minimization that uses a combination of parabolic interpolation and golden section search, selecting the latter when the former does not provide much improvement. Methods that make use of derivatives of f are also available.

It would be remiss not to mention that a d-dimensional minimization algorithm, that uses no derivatives, also exists. It is called the **Nelder-Mead method** (or simplex method, not to be confused with the completely different simplex method of linear programming) and it walks a $(d+1)$-dimensional polyhedron downhill across a d-dimensional function's surface. But there is no way to 'bracket' a local minimum in more than 1 dimension, and the Nelder-Mead method is not guaranteed to find a local minimum.

6.3 Gradient Descent Methods (d dimensions)

In more than 1 dimension, bracketing a local minimum is not feasible. Optimization of functions in higher dimensions is a vast discipline, and we can only give some introductory methods.

The major class of numerical optimization algorithms are **line search** methods. Broadly, they work as follows: given $\boldsymbol{x_n}$, and perhaps some information about the gradient and Hessian of f at or near $\boldsymbol{x_n}$, choose a **direction** $\boldsymbol{d_n}$. Then search for a **step length** $\alpha_n > 0$. The next iteration is

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}. \tag{6.4}$$

Since we will refer to the gradients of f at $\boldsymbol{x_n}$ often, for the rest of this chapter we will write

$$\boldsymbol{g_n} = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}).$$

Desirable properties of $\boldsymbol{d_n}$ and α_n include:

(i) $\boldsymbol{d_n}$ should be a **descent direction**, in which f initially decreases: $\frac{\mathrm{d}f(\boldsymbol{x_n}+\alpha\boldsymbol{d_n})}{\mathrm{d}\alpha}(0) < 0$, which is equivalent to

$$\boldsymbol{g_n^T d_n} < 0; \tag{6.5}$$

(ii) the directions $\boldsymbol{d_n}, \boldsymbol{d_{n-1}}, \boldsymbol{d_{n-2}}, \ldots$ should not veer wildly, with each undoing some of the work of previous steps;

(iii) α_n should 'loosely' minimize $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n})$;

(iv) the **infinite travel condition** to prevent the method slowing to a complete stop

$$\sum_{n=1}^{\infty} \alpha_n = \infty.$$

△

Having made good progress at reducing f in the direction $\boldsymbol{d_n}$, we move on to the next iterative step in a new direction.

First, let us discuss some ways to choose $\boldsymbol{d_n}$. The simplest is called **co-ordinate gradient descent**. Here $\boldsymbol{d_n}$ repeatedly cycles through unit vectors $\boldsymbol{e_i}$ (in some order). We should allow $\alpha < 0$, as some of these vectors will point in the wrong direction and fail to satisfy (6.5). We are minimizing f with respect to each of its arguments one at a time, holding the others constant. Co-ordinate gradient descent is naive, because $\boldsymbol{d_n}$ is independent of the function f; this also means that the iterative step can be fast, and derivatives are not needed.

A more sophisticated direction is to choose

$$\boldsymbol{d_n} = -\boldsymbol{g_n};$$

then (6.5) is automatically satisfied. This is called **gradient descent** or sometimes **steepest descent**, because it is (up to a positive scalar multiple) the solution to

$$\underset{\boldsymbol{d_n}}{\text{minimize}} \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})^T \boldsymbol{d_n} \text{ subject to } \|\boldsymbol{d_n}\| \leq 1,$$

i.e. it is the direction in which f decreases the most rapidly.

We will see another possibility for direction, which is informed by both the gradient and the Hessian of f, in the next section.

After deciding $\boldsymbol{d_n}$, in principle we could find α_n by a one-dimensional minimization algorithm (e.g. golden section search, or a simple one-dimensional Newton method from Section 6.4). This can lead to convergence in relatively few iterative steps and, furthermore, it is possible to prove that convergence to a local minimum is certain for any choice of $\boldsymbol{x_0}$ except one directly on a stationary point. But in practice then each iterative step, which itself involves solving a one-dimensional optimization problem, is too slow. It is better to find α_n that is only very roughly optimal, and move on to the next iteration. After all, even the α_n that truly minimizes $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n})$ is only a step on the way to solving the original optimization problem.

The simplest gradient descent algorithms use a predetermined α_n, either a constant, or decaying as $n \to \infty$. But even when $\boldsymbol{d_n}$ is a descent direction, we are only guaranteed that $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d}) < f(\boldsymbol{x_n})$ for sufficiently small α_n, so we are on the horns of a dilemma: choose α_n too small and the iteration makes little progress, choose α_n too large and the next iteration might be a worse solution. See Figure 6.2, right.

It is probably fair to say that entire textbooks can be written on the subject of choosing a good step length. We will discuss a simple method called **backtracking**: choose an initial step length $\alpha_n = \alpha'$, and then repeatedly multiply it by a factor of $0 < \rho < 1$ until we have reduced to it a 'good enough' α_n. (Optionally, having found a 'good enough' α_n, polish it to a better one by a few steps of a one-dimensional method; we will not pursue this.)

We have three things to choose. The constant ρ is often $\frac{1}{2}$. The very first step size α' is notoriously tricky, as the right magnitude of the steps depends on the function being

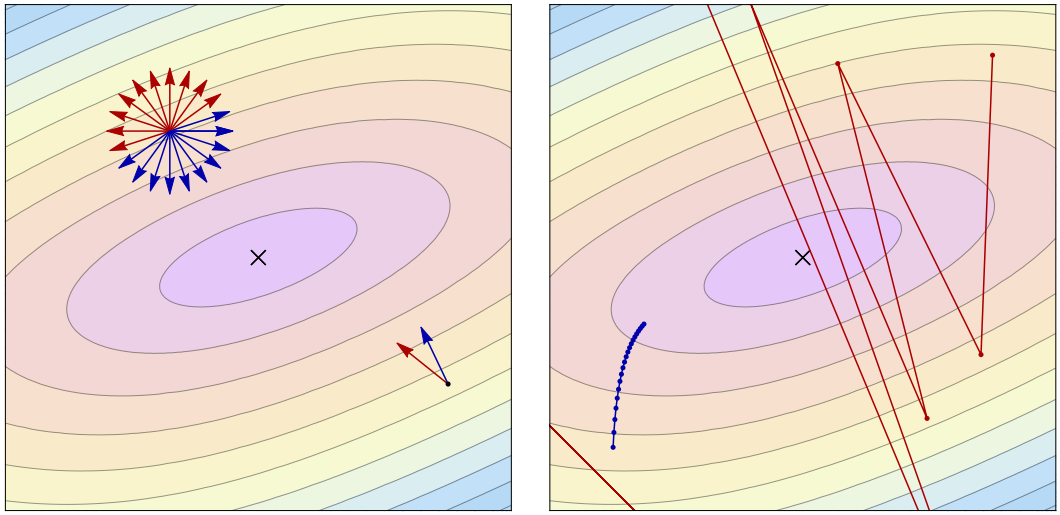


Figure 6.2: Gradient descent methods. The minimum is marked with $\times$. *Left figure, upper left*: blue arrows indicate descent directions and red arrows non-descent directions. *Left figure, lower right*: blue arrow is the steepest descent direction, red is the Newton's method direction. *Right figure:* gradient descent without backtracking, *blue steps*: step size too small, *red steps*: step size too big.

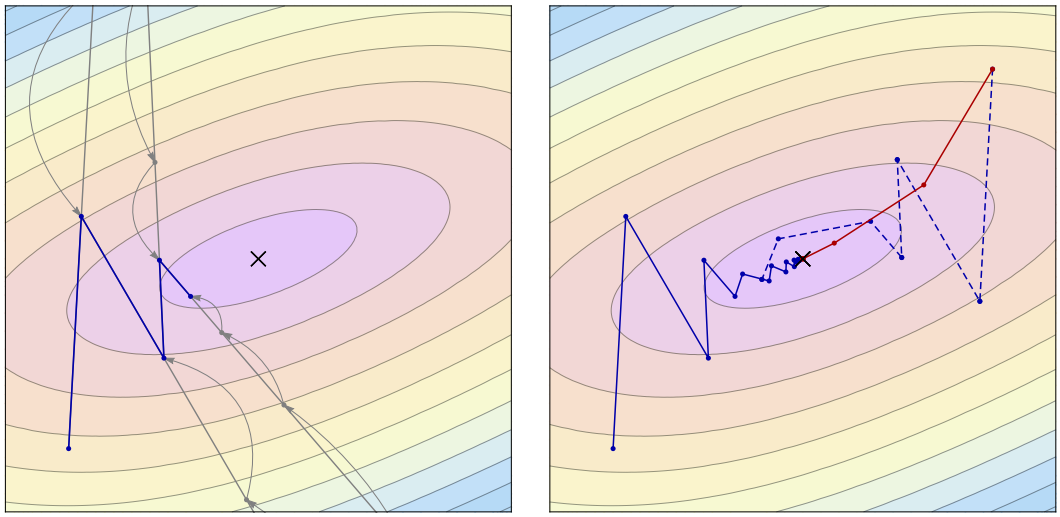


Figure 6.3: *Left figure*: backtracking a direction until improvement in $f(\boldsymbol{x})$ is found. *Right figure*: blue line is progress of gradient descent with backtracking; red line is progress of Newton's method; dashed blue line is conjugate gradient descent (not in this course).

optimized. It should be large enough that iterations make progress. Note that it does *not* cost a great deal if α' is too large, as a few repetitions of $\alpha_0 \mapsto \rho\alpha_0$ will soon find the right magnitude. After the first step, a reasonable guess for α' is such that the first-order approximation to the amount of descent is the same as it was at the last step, so

$$\alpha' = \alpha_{n-1}\frac{\boldsymbol{g_{n-1}^T d_{n-1}}}{\boldsymbol{g_n^T d_n}}. \tag{6.6}$$

This still leaves the very first α' to be supplied by the user.

Finally, we need to define what constitutes a 'good enough' step length. We could simply say that $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}) < f(\boldsymbol{x_n})$ is good enough, but this allows for steps that make infinitesimal progress. Instead, we could adopt the **Armijo rule**, which is

$$f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}) < f(\boldsymbol{x_n}) + \sigma\alpha_n \boldsymbol{g_n^T d_n}$$

where σ is a small constant, often in the range $10^{-4} - 10^{-1}$. This ensures that we make at least a proportion of the progress that the first-order Taylor approximation would expect. It is a consequence of the second-order remainder that the Armijo rule is satisfied for sufficiently small α_n whenever $\boldsymbol{d_n}$ is a descent direction, so the backtracking is guaranteed to terminate.

The backtracking method is depicted in Figure 6.3, as well as the progress of an example of gradient descent that uses it. We should stress that this method of backtracking is far from the best: more sophisticated optimization methods use methods that can also increase α_n as well as shrink it, to meet what are called the **Wolfe conditions** which include both the Armijo rule and another condition on the curvature of $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n})$. You can read about them yourself if you are interested.

We have computed the gradient at each step, so we can terminate when it is close to zero. Since the scale of $\boldsymbol{g_n}$ will depend on the problem, it makes sense to set a relative threshold on $\|\boldsymbol{g_n}\|$, and one simple possibility is to make it relative to (or a combination of absolute and relative to) $\boldsymbol{g_0}$. This method is shown as Algorithm 8, which includes backtracking and choosing the initial step size based on the previous step by (6.6). The user supplies the very first initial size step α_0', along with the backtracking parameters ρ and σ.

Algorithm 8 Gradient descent with backtracking. Replace $\|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$ by another termination condition, if desired.

procedure GRADIENTDESCENT(f, $\boldsymbol{x_0}$, α'_0, ρ, σ, N, tol)
  $\boldsymbol{g_0} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})$
  $\boldsymbol{d_0} \leftarrow -\boldsymbol{g_0}$ ▷ Descent direction
  $f_0 \leftarrow f(\boldsymbol{x_0})$
  $\alpha_0 \leftarrow \alpha'_0$
  $n \leftarrow 0$
  repeat
    $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}$
    $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    while $f_{n+1} \geq f_n + \sigma\alpha_n \boldsymbol{g_n}^T \boldsymbol{d_n}$ **do** ▷ Armijo rule
      $\alpha_n \leftarrow \rho\alpha_n$ ▷ Exponential backtracking
      $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}$
      $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    end while
    $n \leftarrow n + 1$
    $\boldsymbol{g_n} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})$
    $\boldsymbol{d_n} \leftarrow -\boldsymbol{g_n}$ ▷ Next descent direction
    $\alpha_n \leftarrow \alpha_{n-1} \frac{\boldsymbol{g_{n-1}}^T \boldsymbol{d_{n-1}}}{\boldsymbol{g_n}^T \boldsymbol{d_n}}$ ▷ Next initial step size
  until $n = N \vee \|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$
  return $\boldsymbol{x_n}$ ▷ Should raise a warning if $n = N$
end procedure

Example 6.2 The function in Figures 6.2 and 6.3 is

$$f(x,y) = 4(x-2y)^2 + (x+2y)^2 + \tfrac{1}{50}\big(4(x-2y)^4 + (x+2y)^4\big).$$

Algebra shows that its minimum is at $(0,0)$. Various gradient descent methods will use it as a running example in the lectures.

A harder example will be

$$g(x,y) = \log\big(1 + f(x,y)\big)$$

which, unlike f, is non-convex.

There are many variants of gradient descent, that are more robust to difficult optimization problems. One is **conjugate gradient descent**, which aims to prevent the sharp 'turns' that are evident in Figure 6.3. This is a phenomenon where each step partly 'undoes' the work of the last. This does not happen with **conjugate directions**, that satisfy

$$\boldsymbol{d_n}^T \mathbf{H} \boldsymbol{d_m} = 0, \quad \text{for} \quad m < n,$$

where $\mathbf{H}$ is a matrix, in our case the Hessian of f at $\boldsymbol{x_n}$. A set of conjugate directions forms an orthonormal basis for the space $\mathbb{R}^d$, but a remarkable fact is that an approximate conjugate direction can be chosen without remembering all the previous directions, or even knowing $\mathbf{H}$! The *Polak-Ribière* version of conjugate gradient descent uses

$$\boldsymbol{d_n} = -\boldsymbol{g_n} + \boldsymbol{d_{n-1}} \frac{(\boldsymbol{g_n} - \boldsymbol{g_{n-1}})^T \boldsymbol{g_n}}{\boldsymbol{g_{n-1}}^T \boldsymbol{g_{n-1}}}$$

but is beyond our syllabus. You can read about it (and alternative versions) in textbooks on optimization. We display the progress of conjugate gradient descent in Figure 6.3. In one sense this method is akin to Newton's method (next section) in that the Hessian of f informs a (sometimes) more direct route towards the local minimum.

Another variant can be applied when the objective function is in the form $l(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} l_i(\boldsymbol{w})$, e.g. arising from N items of data in machine learning where the total

loss function is the average loss on each item. If the complete data are too large to fit into memory, or simply to make each iteration faster, we can calculate a gradient with respect to part of the data only. For example,

$$\boldsymbol{d_n} = -\frac{1}{M-L}\sum_{i=L}^{M}\frac{\mathrm{d}l_i}{\mathrm{d}\boldsymbol{w}}(\boldsymbol{w_n})$$

which is called **minibatch gradient descent** when the selection of data moves through the full set, in batches, as the iterative steps of gradient descent proceed. The smaller the size of the minibatch, the more 'noisy' the gradient, and it sometimes chooses directions that are bad for the full data. When performance is paramount, good minibatch sizes are dictated by memory cache sizes.

An extreme example is **stochastic gradient descent**, which uses a single random item on each step. It calculates the gradient only with respect to that part of the loss function:

$$i \text{ chosen randomly from } \{1,\ldots,N\}, \quad \boldsymbol{d_n} = -\frac{\mathrm{d}l_i}{\mathrm{d}\boldsymbol{w}}(\boldsymbol{w_n}).$$

We should expect to need many more iterations, but each step is much faster. Sometimes the choice of i is not random, but repeats through the entire data set in a shuffled order, re-shuffled for each pass. This ensures that every item of data is visited the same number of times.

6.4 Newton Methods (d dimensions)

Returning to the theme of other numerical algorithms we have seen, let us approximate f by a local quadratic function near $\boldsymbol{x_0}$:

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x}-\boldsymbol{x_0})^T\boldsymbol{g_0} + \tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{x_0})^T\mathbf{H}(f)(\boldsymbol{x_0})(\boldsymbol{x}-\boldsymbol{x_0}).$$

The minimum of the approximation is when

$$\frac{\mathrm{d}\hat{f}}{\mathrm{d}\boldsymbol{x}} = \boldsymbol{g_0} + \mathbf{H}(f)(\boldsymbol{x_0})(\boldsymbol{x}-\boldsymbol{x_0}) = 0 \quad \Leftrightarrow \quad \boldsymbol{x} = \boldsymbol{x_0} - \big(\mathbf{H}(f)(\boldsymbol{x_0})\big)^{-1}\boldsymbol{g_0}.$$

(Of course, approximating f by a local *linear* function is not going to help find a minimum, because linear functions do not have one.)

Iterating this gives rise to **Newton's method**,

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \big(\mathbf{H}(f)(\boldsymbol{x_n})\big)^{-1}\boldsymbol{g_n}. \tag{6.7}$$

Observe that this is a line search method like gradient descent (6.4), but with $\boldsymbol{d_n} = -\big(\mathbf{H}(f)(\boldsymbol{x_n})\big)^{-1}\boldsymbol{g_n}$ (the **Newton direction**) and $\alpha_n = 1$.

A great advantage of Newton-like methods is that there is a natural step size which is independent of the scale of the function f. It is still wise to have the option of backtracking, in case the Newton iteration overshoots the minimum, but it negates the need for the user to supply an initial step size α_0', and means that each step length can be independent of the last.

There are two disadvantages. Clearly, we need to compute the Hessian, d^2 derivatives, and perform $O(d^3)$ work to solve a linear system[1], on each iterative step. This is in contrast to gradient descent, which needs d derivatives for the direction and $O(d)$ work to determine the step length.

The second disadvantage is more subtle: *the Newton direction is not always a descent direction.* According to (6.5), that requires

$$\boldsymbol{g_n}^T\big(\mathbf{H}(f)(\boldsymbol{x_n})\big)^{-1}\boldsymbol{g_n} > 0,$$

which is guaranteed if f is convex (and not stationary at $\boldsymbol{x_n}$), but not guaranteed otherwise. See Figure 6.4 for an example of a non-convex surface on which Newton's method fails. Near a local maximum, Newton's method will in fact do precisely the opposite of our intention and rapidly converge to that local maximum. In other situations it may diverge. We can mitigate this by including backtracking, which will refuse to take a step that increases f, but in that case Newton's method would come to a halt as α_n never meets the Armijo rule when $\boldsymbol{d_n}$ is not a descent direction, and the loop that multiplies by ρ will not terminate. (If we were to swap the sign of $\boldsymbol{d_n}$ in such situations – an ugly hack – we would then need to change the backtracking rule.) Newton's iteration is also ill-defined if $\mathbf{H}(f)(\boldsymbol{x_n})$ is singular.

If f has a well-behaved local minimum $\boldsymbol{x^*}$, $\mathbf{H}(f)(\boldsymbol{x_n})$ should be positive definite near $\boldsymbol{x^*}$, and in such circumstances Newton's method will have quadratic convergence close to the root. Thus it can polish up an approximate local minimum into a highly accurate local minimum in few iterations. But even when f is convex, quadratic convergence is not guaranteed if $\mathbf{H}(f)$ is only semidefinite at $\boldsymbol{x^*}$. This is the optimization analogue of the double root when root-finding.

[1] As in the last chapter, when we write $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \big(\mathbf{H}(f)(\boldsymbol{x_n})\big)^{-1}\boldsymbol{g_n}$, for better efficiency we really mean solve $\big(\mathbf{H}(f)(\boldsymbol{x_n})\big)\boldsymbol{d_n} = -\boldsymbol{g_n}$ for $\boldsymbol{d_n}$ and set $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \boldsymbol{d_n}$.

In an attempt to mitigate the risks of Newton's method, there are methods (beyond this course) to modify the Hessian. A simple trick is to add $\lambda\mathbf{I}$ to $\mathbf{H}(f)$, where λ is just big enough to make the Hessian positive definite. Ideally λ should be slightly greater than the magnitude of the most negative eigenvalue of $\mathbf{H}(f)$, but finding eigenvalues is expensive, so some methods simply try bigger and bigger λ until the **Cholesky algorithm** (which factors $\mathbf{H}$ as $\mathbf{L}\mathbf{L}^T$, and makes for an efficient calculation of $\boldsymbol{d_n}$) succeeds.

A version of Newton's method for minimization, which includes backtracking but excludes Hessian modification, is Algorithm 9.

6.4.1 Connections between Optimization and Root-Finding

The astute reader will have drawn a connection between Newton's method for optimization (6.7) and Newton's method for root-finding (5.6). The former (omitting backtracking) is precisely the latter applied to $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$: by equation (1.3), the Hessian of f is precisely the Jacobian of $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$. This makes sense, as zero derivative is a necessary condition for a local optimum. Why did this chapter not simply say: 'Chapter 6. Algorithms for Optimization. Apply the methods from Chapter 5 to $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$. End of chapter.'?

The failure modes of Newton's method for optimization demonstrate why. Zero derivative is a necessary condition for a local minimum, but not a sufficient condition, and Newton's method without backtracking will just as happily converge to a saddle point or a maximum as to a minimum. For a strongly convex function (where all eigenvalues of the Hessian are strictly positive everywhere) with continuous second derivatives, there is only one global minimum stationary point and Newton's method is globally quadratically convergent. But even so it is not making full use of the structure of a minimization problem. Minimization ought to be *easier* than root-finding for a stationary point, because we have an unambiguous definition of the next iterative step being 'better' than the last, and because a Hessian has structure that an arbitrary Jacobian does not. For one thing, it is symmetric assuming, as we have throughout this section, that the second derivatives are all continuous.

We could also ask the converse question: since all $\boldsymbol{x}$ with $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ are solutions to

$$\arg\min_{\boldsymbol{x}} \|\boldsymbol{f}(\boldsymbol{x})\|, \tag{6.8}$$

why not use minimization algorithms to solve root-finding problems? (To avoid the square root and preserve differentiability, we are more likely to minimize $\|\boldsymbol{f}(\boldsymbol{x})\|^2$.) It is not a

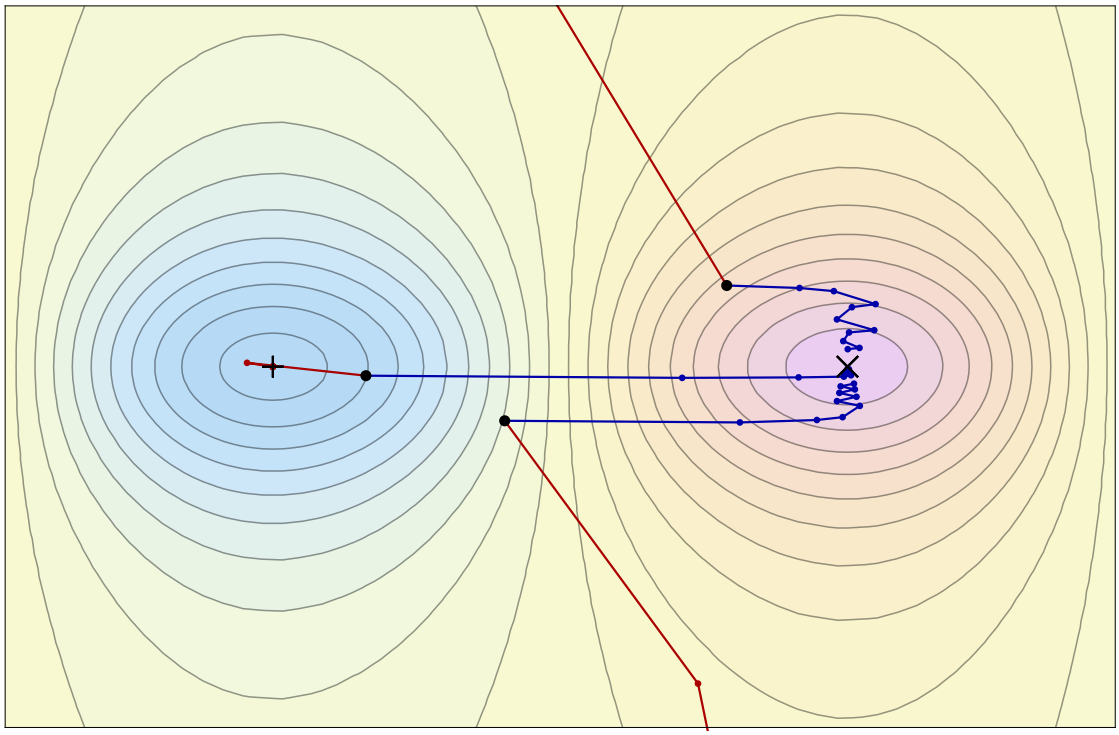


Figure 6.4: Newton's method (without backtracking) failing on a non-convex surface. The minimum is marked with $\times$, and maximum with $+$. Optimization has been started from three different points: the *blue steps* are gradient descent, the *red steps* are Newton's method starting from the same points. In all cases gradient descent eventually finds the local minimum; Newton's method diverges in two cases and converges to the local maximum in the other. If backtracking were added, Newton's method would indefinitely backtrack as it has chosen a non-descent direction.

Algorithm 9 Newton's method for optimization, with backtracking. Replace $\|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$ by another termination condition, if desired.

procedure NEWTONOPTIMIZATION(f, $\boldsymbol{x_0}$, ρ, σ, N, tol)
  $\boldsymbol{g_0} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})$
  Solve for $\boldsymbol{d_0}$: $\big(\mathbf{H}(f)(\boldsymbol{x_0})\big)\boldsymbol{d_0} = -\boldsymbol{g_0}$ ▷ Raise an error if no solutions!
  $f_0 \leftarrow f(\boldsymbol{x_0})$
  $n \leftarrow 0$
  repeat
    (optional: if $\boldsymbol{g_n}^T\boldsymbol{d_n} > 0$, consider switching the sign of $\boldsymbol{d_n}$)
    $\alpha_n \leftarrow 1$
    $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n\boldsymbol{d_n}$
    $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    while $f_{n+1} \geq f_n + \sigma\alpha_n\boldsymbol{g_n}^T\boldsymbol{d_n}$ **do** ▷ Armijo rule
      $\alpha_n \leftarrow \rho\alpha_n$ ▷ Exponential backtracking
      $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n\boldsymbol{d_n}$
      $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    end while
    $n \leftarrow n + 1$
    $\boldsymbol{g_n} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})$
    Solve for $\boldsymbol{d_n}$: $\big(\mathbf{H}(f)(\boldsymbol{x_n})\big)\boldsymbol{d_n} = -\boldsymbol{g_n}$ ▷ Raise an error if no solutions!
  until $n = N \vee \|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$
  return $\boldsymbol{x_n}$ ▷ Should raise a warning if $n = N$
end procedure

terrible idea to use a few steps of a minimization algorithm for (6.8) to find a starting point for the methods of Chapter 5. But $\|\boldsymbol{f}(\boldsymbol{x})\|^2$ is rarely a convex function, and optimization algorithms applied to it may struggle to avoid local minima that are not zeros of $\boldsymbol{f}$. A case where $\|\boldsymbol{f}(\boldsymbol{x})\|^2$ *is* convex arises when $\boldsymbol{f}$ is linear, and indeed iterative optimization methods can usefully be applied to the problem $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$. Gradient descent and conjugate gradient descent are commonly used for large linear systems where Gaussian elimination is infeasible, or when the system is sparse (most coefficients are zero).

6.5 Quasi-Newton Methods (d dimensions)

As with root-finding, there are secant-style methods for optimization that do not compute derivatives. We can take the opportunity to include three advantages over Newton's method: the replacement of derivatives (in this case the Hessian) by an approximation that builds through the iterations; the removal of $O(d^3)$ work in solving a linear system; remedying the problems caused by Hessians that are not positive definite.

There are many analogues of Broyden's method, and a full explanation of their derivation is beyond this course. But we can present the most widely-used method. It was discovered independently (and justified by slightly different arguments) by Broyden, Fletcher, Goldfarb, and Shanno, and hence is known as **BFGS**.

First, recall the Jacobian update for Broyden's method:

$$\hat{\mathbf{J}}_{\boldsymbol{n}} = \hat{\mathbf{J}}_{\boldsymbol{n-1}} + \frac{\boldsymbol{\Delta y} - \hat{\mathbf{J}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}}{\|\boldsymbol{\Delta x}\|^2}\boldsymbol{\Delta x}^T. \qquad \triangle$$

(In our case we should replace $\hat{\mathbf{J}}_{\boldsymbol{n}}$ by $\hat{\mathbf{H}}_{\boldsymbol{n}}$ and $\boldsymbol{\Delta y}$ by $\boldsymbol{\Delta g} = \boldsymbol{g_n} - \boldsymbol{g_{n-1}}$, since we are searching for a zero derivative rather than a zero of the function.) This should not be used to update a Hessian, because it does not give a symmetric matrix. Instead, BFGS uses a rank-2 update that preserves symmetry, while satisfying a secant equation and minimizing a suitably-measured change in the inverse Hessian from the previous step (all details omitted). The approximate Hessian at step n is given by:

$$\hat{\mathbf{H}}_{\boldsymbol{n}} = \hat{\mathbf{H}}_{\boldsymbol{n-1}} + \frac{\boldsymbol{\Delta g}\boldsymbol{\Delta g}^T}{\boldsymbol{\Delta g}^T\boldsymbol{\Delta x}} - \frac{\hat{\mathbf{H}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}\boldsymbol{\Delta x}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}}{\boldsymbol{\Delta x}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}}. \qquad (6.9) \qquad \triangle$$

Furthermore, $\hat{\mathbf{H}}_{\boldsymbol{n}}$ will be positive definite as long as $\hat{\mathbf{H}}_{\boldsymbol{n-1}}$ is positive definite and the so-called **curvature condition** is satisfied:

$$\boldsymbol{\Delta x}^T\boldsymbol{\Delta g} > 0. \qquad (6.10) \qquad \triangle$$

Unfortunately, although this condition is satisfied for convex f, for non-convex f it can fail for small step lengths. There are a number of ways to proceed:

(i) use a more complicated step-length selection algorithm that expands as well as contracts the step size so as to enforce the **Wolfe conditions**, which ensure (6.10);
(ii) if (6.10) fails, simply skip the Hessian update and set $\hat{\mathbf{H}}_{\boldsymbol{n}} = \hat{\mathbf{H}}_{\boldsymbol{n-1}}$;
(iii) if (6.10) fails, *damp* the Hessian update with $\hat{\mathbf{H}}_{\boldsymbol{n}}$ some weighted average of (6.9) and $\hat{\mathbf{H}}_{\boldsymbol{n-1}}$;
(iv) if (6.10) fails, recognize that the approximate Hessian is not looking good, and we may be in a non-convex region of f. Reset to $\hat{\mathbf{H}}_{\boldsymbol{n}} = \mathbf{I}$, so that we use the ultra-safe gradient descent for the next step, and can start re-approximating the Hessian when we are hopefully in a convex region of f.

For simplicty and safety we choose the last option.

After computing the approximate Hessian, BFGS proceeds to use a quasi-Newton direction

$$\boldsymbol{d_n} = -\hat{\mathbf{H}}_{\boldsymbol{n}}^{-1}\boldsymbol{g_n}$$

and the rest of the algorithm works as Newton's method. The termination can also be similar, for example stopping when the magnitude of $\boldsymbol{g_n}$ is small relative to the initial gradient $\boldsymbol{g_0}$, or some other condition.

To get BFGS started we need to supply an initial Hessian $\hat{\mathbf{H}}_{\mathbf{0}}$. The identity matrix is an easy-to-justify choice, because then the first step of BFGS is simply gradient descent. As the iterations proceed, $\hat{\mathbf{H}}_{\boldsymbol{n}}$ should approach the true Hessian if f is well-behaved. And since $\hat{\mathbf{H}}_{\boldsymbol{n}}$ is constructed to be positive definite (even for non-convex f!), BFGS should not be attracted by local maxima or saddle points, nor should the system $\hat{\mathbf{H}}_{\boldsymbol{n}}\boldsymbol{d_n} = -\boldsymbol{g_n}$ be singular.

A simple version of BFGS is displayed as Algorithm 10, but a further improvement is to track the inverse Hessian directly using the Sherman-Morrison formula,

$$\hat{\mathbf{H}}_{\boldsymbol{n}}^{-1} = \hat{\mathbf{H}}_{\boldsymbol{n-1}}^{-1} + \frac{(\boldsymbol{\Delta x}^T\boldsymbol{\Delta g} + \boldsymbol{\Delta g}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}^{-1}\boldsymbol{\Delta g})\boldsymbol{\Delta x}\boldsymbol{\Delta x}^T}{(\boldsymbol{\Delta x}^T\boldsymbol{\Delta g})^2} - \frac{\hat{\mathbf{H}}_{\boldsymbol{n-1}}^{-1}\boldsymbol{\Delta g}\boldsymbol{\Delta x}^T + \boldsymbol{\Delta x}\boldsymbol{\Delta g}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}^{-1}}{\boldsymbol{\Delta x}^T\boldsymbol{\Delta g}},$$

△ thus avoiding any $O(d^3)$ computations. This is completely analogous to the case of Broyden's method that you examined on a problem sheet.

Error analysis of BFGS is well beyond our level. Under certain conditions on f it has been proved globally convergent (that is, always eventually converges to a local minimum no matter what the starting point $\boldsymbol{x_0}$). It can be shown that convergence is superlinear but not in general quadratic. In practice it outperforms Newton's method both for reliability and speed. The limited-memory version of BFGS known as **L-BFGS**, which avoids storing any matrices in full, is popular in machine learning applications of high dimension, but the simpler and quicker methods like gradient descent are also useful in practice.

Final Example The lectures will conclude with a long example on steganography and steganalysis, reviewing convexity, Lagrange multipliers, numerical root-finding, and numerical optimization.

Algorithm 10 A simple version of the BFGS method. Replace $\|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$ by another termination condition, if desired. Replace $\hat{\mathbf{H}}_{\boldsymbol{n}} \leftarrow \mathbf{I}$ by alternative actions when the curvature condition fails, if desired.

procedure BFGS($\boldsymbol{f}$, $\boldsymbol{x_0}$, ρ, σ, N, tol)
  $\hat{\mathbf{H}}_{\mathbf{0}} \leftarrow \mathbf{I}$
  $\boldsymbol{g_0} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0})$
  $\boldsymbol{d_0} \leftarrow -\boldsymbol{g_0}$ ▷ First step by gradient descent
  $f_0 \leftarrow f(\boldsymbol{x_0})$
  $n \leftarrow 0$
  repeat
    $\alpha_n \leftarrow 1$
    $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}$
    $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    while $f_{n+1} \geq f_n + \sigma\alpha_n \boldsymbol{g_n}^T \boldsymbol{d_n}$ **do** ▷ Armijo rule
      $\alpha_n \leftarrow \rho\alpha_n$ ▷ Exponential backtracking
      $\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}$
      $f_{n+1} \leftarrow f(\boldsymbol{x_{n+1}})$
    end while
    $n \leftarrow n + 1$
    $\boldsymbol{g_n} \leftarrow \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})$
    $\boldsymbol{\Delta x} \leftarrow \boldsymbol{x_n} - \boldsymbol{x_{n-1}}$
    $\boldsymbol{\Delta g} \leftarrow \boldsymbol{g_n} - \boldsymbol{g_{n-1}}$
    if $\boldsymbol{\Delta x}^T \boldsymbol{\Delta g} > 0$ **then** ▷ Curvature condition
      $\hat{\mathbf{H}}_{\boldsymbol{n}} \leftarrow \hat{\mathbf{H}}_{\boldsymbol{n-1}} + \frac{\boldsymbol{\Delta g}\boldsymbol{\Delta g}^T}{\boldsymbol{\Delta g}^T\boldsymbol{\Delta x}} - \frac{\hat{\mathbf{H}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}\boldsymbol{\Delta x}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}}{\boldsymbol{\Delta x}^T\hat{\mathbf{H}}_{\boldsymbol{n-1}}\boldsymbol{\Delta x}}$ ▷ Update Hessian
    else
      $\hat{\mathbf{H}}_{\boldsymbol{n}} \leftarrow \mathbf{I}$ ▷ Curvature condition failed, reset Hessian
    end if
    Solve for $\boldsymbol{d_n}$: $\hat{\mathbf{H}}_{\boldsymbol{n}}\boldsymbol{d_n} = -\boldsymbol{g_n}$ ▷ Will always have a solution, since $\hat{\mathbf{H}}_{\boldsymbol{n}}$ is positive definite by construction
  until $n = N \vee \|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$
  return $\boldsymbol{x_n}$ ▷ Should raise a warning if $n = N$
end procedure

6.6 Bonus Mathematics: Linear Convergence of Gradient Descent

This is not part of the syllabus.

We prove that gradient descent in d dimensions has linear convergence, for the easiest case of so-called *strongly convex Lipschitz* functions. Define $D = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| < c\}$ to be a region around a local minimum $\boldsymbol{x}^*$, and within it let f satisfy the following condition: there are constants $0 < \underline{\lambda} \leq \overline{\lambda}$ such that all eigenvalues of $\mathbf{H}(f)(\boldsymbol{x})$ are between $\underline{\lambda}$ and $\overline{\lambda}$. The upper bound on the eigenvalues is equivalent to the condition in the challenge exercise on tutorial sheet 4. The lower bound ensures that f is convex on D, but the fact that eigenvalues are bounded away from zero ensures *strong* convexity: a stronger hypothesis than that on the tutorial sheet, which allows us to prove good convergence. It turns out that f having not too much curvature (the upper bound on eigenvalues), but not too little (the lower bound) is the sweet spot to guarantee linear convergence of gradient descent.

First, a lemma:

Lemma 6.3 (i) For such f, and $\boldsymbol{u}, \boldsymbol{v} \in D$,

$$(\boldsymbol{u}-\boldsymbol{v})^T \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{v}) + \tfrac{\underline{\lambda}}{2}\|\boldsymbol{u}-\boldsymbol{v}\|^2 \overset{(a)}{\leq} f(\boldsymbol{u}) - f(\boldsymbol{v}) \overset{(b)}{\leq} (\boldsymbol{u}-\boldsymbol{v})^T \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{v}) + \tfrac{\overline{\lambda}}{2}\|\boldsymbol{u}-\boldsymbol{v}\|^2.$$

(ii) For $\boldsymbol{x} \in D$,

$$\left\|\tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x})\right\|^2 \leq 2\overline{\lambda}\big(f(\boldsymbol{x}) - f(\boldsymbol{x}^*)\big).$$

Proof (i) The Taylor expansion of f at $\boldsymbol{v}$, with second-order remainder, is

$$f(\boldsymbol{u}) = f(\boldsymbol{v}) + (\boldsymbol{u}-\boldsymbol{v})^T \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{v}) + \tfrac{1}{2}(\boldsymbol{u}-\boldsymbol{v})^T \mathbf{H}(f)(\boldsymbol{w})(\boldsymbol{u}-\boldsymbol{v})$$

for some $\boldsymbol{w}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$. The result follows by rearranging and using $\underline{\lambda}\|\boldsymbol{x}\|^2 \leq \boldsymbol{x}^T\mathbf{A}\boldsymbol{x} \leq \overline{\lambda}\|\boldsymbol{x}\|^2$ when $\underline{\lambda}$ and $\overline{\lambda}$ are lower and upper bounds on the eigenvalues of symmetric $\mathbf{A}$.

(ii)

$$
\begin{aligned}
& f(\boldsymbol{x}^*) - f(\boldsymbol{x}) \\
& \quad \leq f\Big(\boldsymbol{x} - \tfrac{1}{\overline{\lambda}} \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x})\Big) - f(\boldsymbol{x}) && \big\{\, \boldsymbol{x}^* \text{ is the minimum} \,\big\} \\
& \quad \leq -\tfrac{1}{\overline{\lambda}} \big\| \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x}) \big\|^2 + \tfrac{\overline{\lambda}}{2} \big\| -\tfrac{1}{\overline{\lambda}} \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x}) \big\|^2 && \big\{\, \text{(i)}(b), \text{ with } \boldsymbol{u} = \boldsymbol{x} - \tfrac{1}{\overline{\lambda}} \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x}) \text{ and } \boldsymbol{v} = \boldsymbol{x} \,\big\} \\
& \quad = \tfrac{1}{2\overline{\lambda}} \big\| \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x}) \big\|^2, && \big\{\, \text{algebra} \,\big\}
\end{aligned}
$$

and the result follows by multiplying by $-2\overline{\lambda}$. ■

Theorem 6.4 As long as $0 < \alpha \leq 1/\overline{\lambda}$,

$$\frac{\big\| \boldsymbol{x_n} - \alpha \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}) - \boldsymbol{x}^* \big\|}{\|\boldsymbol{x_n} - \boldsymbol{x}^*\|} \leq \sqrt{1 - \alpha\underline{\lambda}}.$$

and so the gradient descent method with step length α converges at least linearly to $\boldsymbol{x}^*$.

Proof

$$
\begin{aligned}
& \big\| \boldsymbol{x_n} - \alpha \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}) - \boldsymbol{x}^* \big\|^2 \\
& = \|\boldsymbol{x_n} - \boldsymbol{x}^*\|^2 + \alpha^2 \big\| \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}) \big\|^2 + 2\alpha(\boldsymbol{x}^* - \boldsymbol{x_n})^T \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}) \\
& \qquad \big\{\, \|\boldsymbol{u} - \boldsymbol{v}\|^2 = \|\boldsymbol{u}\|^2 + \|\boldsymbol{v}\|^2 - 2\boldsymbol{u}^T\boldsymbol{v} \,\big\} \\
& \leq \|\boldsymbol{x_n} - \boldsymbol{x}^*\|^2 + 2\alpha^2\overline{\lambda}\big(f(\boldsymbol{x_n}) - f(\boldsymbol{x}^*)\big) + 2\alpha(\boldsymbol{x}^* - \boldsymbol{x_n})^T \tfrac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n}) \\
& \qquad \big\{\, \text{Lemma 6.3(ii)} \,\big\} \\
& \leq \|\boldsymbol{x_n} - \boldsymbol{x}^*\|^2 + 2\alpha^2\overline{\lambda}\big(f(\boldsymbol{x_n}) - f(\boldsymbol{x}^*)\big) + 2\alpha\Big(f(\boldsymbol{x}^*) - f(\boldsymbol{x_n}) - \tfrac{\underline{\lambda}}{2}\|\boldsymbol{x}^* - \boldsymbol{x_n}\|^2\Big) \\
& \qquad \big\{\, \text{Lemma 6.3(i)}(a) \,\big\} \\
& = \|\boldsymbol{x_n} - \boldsymbol{x}^*\|^2(1 - \alpha\underline{\lambda}) + 2\alpha(\alpha\overline{\lambda} - 1)\big(f(\boldsymbol{x_n}) - f(\boldsymbol{x}^*)\big) \\
& \qquad \big\{\, \text{algebra} \,\big\} \\
& \leq \|\boldsymbol{x_n} - \boldsymbol{x}^*\|^2(1 - \alpha\underline{\lambda}). \\
& \qquad \big\{\, f(\boldsymbol{x_n}) \geq f(\boldsymbol{x}^*) \text{ and } \alpha\overline{\lambda} - 1 \leq 0 \,\big\}
\end{aligned}
$$

The stated inequality follows by dividing and taking square roots.

Since $\underline{\lambda} \leq \overline{\lambda}$, $0 < \alpha < 1/\overline{\lambda}$ implies that $0 < \sqrt{1 - \alpha\underline{\lambda}} < 1$, and this establishes linear convergence. ■

Observe that the ratio of successive errors is smallest when $\overline{\lambda}$ is close to $\underline{\lambda}$, and $\alpha = 1/\overline{\lambda}$. The ratio $\overline{\lambda}/\underline{\lambda}$ is known as the **condition number** of the Hessian, and gradient descent will generally converge more rapidly when the objective Hessians have smaller condition numbers. Condition numbers are also important in the study of numerical stability of algorithms.

Practice Questions

Write programs to do the iterations. If you try to do them by hand, you are crazy.

6.1 Find a starting bracket for the following problems. Approximately how many steps will it take for golden section search to terminate with $|c_n - a_n| < 10^{-6}$?

(i) $\arg\min_x e^{x-1} + e^{100-100x}$, (ii) $\arg\min_x \frac{(x-1)\sin x}{x^2+1}$, (iii) $\arg\min_x \frac{\cos x - 1}{x^2}$.

Do you anticipate any difficulties with numerical precision, in any of these problems?

6.2 Compute the direction of steepest descent, as well as the Newton method direction (but no step sizes), for the first iterative step of solving the following. $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{x_i}$ are constant vectors, y_i are constant scalars, and λ is a small positive constant.

(i) $\arg\min_{(x,y)} \cos(x+y) + \sin(x-y)$ at $(x_0, y_0) = \left(\frac{\pi}{4}, \frac{\pi}{2}\right)$,

(ii) $\arg\min_{(x,y)} e^{2x-y} + e^{2y-x}$ at arbitrary (x_0, y_0),

(iii) $\arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{a}\|\|\boldsymbol{x}-\boldsymbol{a}\|^2 + \frac{1}{2}\|\boldsymbol{b}\|\|\boldsymbol{x}-\boldsymbol{b}\|^2 + \frac{1}{2}\|\boldsymbol{c}\|\|\boldsymbol{x}-\boldsymbol{c}\|^2$ at $\boldsymbol{x_0} = \boldsymbol{0}$,

(iv) $\arg\min_{\boldsymbol{w}} \lambda \boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^N (\boldsymbol{x_i}^T\boldsymbol{w} - y_i)^2$ at arbitrary $\boldsymbol{w_0}$.

The Newton direction in (ii) is independent of x_0 and y_0: how can this be?

6.3 In the previous question, for which cases is the Newton direction a descent direction?

6.4 We are performing gradient descent on the function $f(x,y) = \frac{1}{2}\left((x+y)^4 + (x-y)^4\right)$ at $(x,y) = (2,1)$. Compute the steepest descent direction and, starting with step size 1, perform backtracking with $\rho = \frac{1}{2}$ until the Armijo rule is satisfied.

6.5 'Find the point on the surface $z = e^{x+\frac{1}{2}y}$ that is closest to the origin.' Express this as an unconstrained minimization problem in x and y.

Starting at $(0,0)$, perform 5 steps of Newton's method (with a fixed step size of 1 and no backtracking). Repeat using the BFGS method (similarly with a fixed step size of 1) and compare the BFGS estimated Hessian with the true Hessian used in Newton's method, at each step.

Answers to Chapter 6 Practice Questions

6.1 The number of steps depends on the bracket. If it is in the form (a, b, c) with $b-a : c-b$ in ratio $\phi : 1-\phi$ (or $1-\phi : \phi$) then the interval reduces by ϕ at each step, taking $\lceil -\log_\phi \frac{c-b}{10^{-6}} \rceil$ to reduce to width 10^{-6}. If the initial bracket does not have these ratios, it may take a few more steps. Possible brackets with the golden ratio are:
(i) $(0, 2\phi, 2)$; will take 31 steps. With a sensible starting bracket, overflow can be avoided.
(ii) $(-6, -6\phi, 0)$; will take 33 steps. With different brackets a local, but not global, minimum may be found.
(iii) $(-1, 2\phi-1, 1)$; would take 31 steps BUT, because of catastrophic cancellation in the numerator, we cannot hope to locate the minimum (numerically) any better than $O(\sqrt[4]{\epsilon})$. The roundoff error for computing $f(x)$ will be $O(\epsilon/x^2)$, which outweighs the true difference $f(x) - f(0) \sim \frac{x^2}{24}$ if x is any less than this. With double precision we cannot expect better than about 10^{-4} error (my implementation converges to ≈ -0.00012).

6.2 (i) Gradient descent direction is $(0, \sqrt{2})$, Newton direction is $(0, 1)$.
(ii) Gradient descent direction is $(-2e^{2x_0-y_0} + e^{2y_0-x_0}, e^{2x_0-y_0} - 2e^{2y_0-x_0})$, Newton direction is $(-1, -1)$; it continues in the same direction at every iteration because *the function has no minimum.* Check this before wasting time on an iterative method!
(iii) Gradient descent direction is $\boldsymbol{a}\|\boldsymbol{a}\| + \boldsymbol{b}\|\boldsymbol{b}\| + \boldsymbol{c}\|\boldsymbol{c}\|$, Newton direction is $\frac{\boldsymbol{a}\|\boldsymbol{a}\|+\boldsymbol{b}\|\boldsymbol{b}\|+\boldsymbol{c}\|\boldsymbol{c}\|}{\|\boldsymbol{a}\|+\|\boldsymbol{b}\|+\|\boldsymbol{c}\|}$.
(iv) Gradient descent direction is $-2\lambda\boldsymbol{w_0} - 2\sum_i (\boldsymbol{x_i}^T\boldsymbol{w_0} - y_i)\boldsymbol{x_i}$, Newton direction is $-\big(\lambda\mathbf{I} + \sum_i \boldsymbol{x_i}\boldsymbol{x_i}^T\big)^{-1}\big(\lambda\boldsymbol{w_0} + \sum_i (\boldsymbol{x_i}^T\boldsymbol{w_0} - y_i)\boldsymbol{x_i}\big)$.

6.3 All of them. It is a descent direction if $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}^T \mathbf{H}(f)^{-1} \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} > 0$, a sufficient condition for which is that the Hessian is positive definite. In the case of (iv), note that all $\boldsymbol{a_i}\boldsymbol{a_i}^T$ are positive semidefinite, so $\lambda\mathbf{I} + \sum_i \boldsymbol{a_i}\boldsymbol{a_i}^T$ is positive definite.

6.4 $\boldsymbol{d} = -\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(2,1) = (-56, -52)$. Compare $f(2,1) = 41$ against:

$$\begin{array}{llll} f(\boldsymbol{x}+\boldsymbol{d}) = f(-54,-51) = 60775353, & f(\boldsymbol{x}+\frac{1}{8}\boldsymbol{d}) = f(-5,-5\frac{1}{2}) \approx 6078, \\ f(\boldsymbol{x}+\frac{1}{2}\boldsymbol{d}) = f(-26,-25) = 3382601, & f(\boldsymbol{x}+\frac{1}{16}\boldsymbol{d}) = f(-1\frac{1}{2},-2\frac{1}{4}) \approx 99, \\ f(\boldsymbol{x}+\frac{1}{4}\boldsymbol{d}) = f(-12,-12) = 165888, & f(\boldsymbol{x}+\frac{1}{32}\boldsymbol{d}) = f(\frac{1}{4},-\frac{5}{8}) \approx 0.303, \end{array}$$

smaller than $f(\boldsymbol{x}) - 10^{-1}\frac{1}{32}\boldsymbol{d}^T\boldsymbol{d} = 41 - 18\frac{1}{4}$. Backtracking was certainly needed.

6.5 $\arg\min_{(x,y)} x^2 + y^2 + e^{2x+y}$. The gradient is $(2x + 2e^{2x+y},\ 2y + e^{2x+y})$ and the inverse Hessian is $\frac{1}{10e^{2x+y}+4}\begin{pmatrix} e^{2x+y}+2 & -2e^{2x+y} \\ -2e^{2x+y} & 4e^{2x+y}+2 \end{pmatrix}$. The Newton and BFGS iterations proceed:

Iteration	0	1	2	3	4	5
$\mathbf{H}(f)(\boldsymbol{x_n})$	$\begin{pmatrix} 6 & 2 \\ 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 3.958 & 0.979 \\ 0.979 & 2.490 \end{pmatrix}$	$\begin{pmatrix} 3.557 & 0.779 \\ 0.779 & 2.389 \end{pmatrix}$	$\begin{pmatrix} 3.534 & 0.767 \\ 0.767 & 2.383 \end{pmatrix}$	$\begin{pmatrix} 3.534 & 0.767 \\ 0.767 & 2.383 \end{pmatrix}$	$\begin{pmatrix} 3.534 & 0.767 \\ 0.767 & 2.383 \end{pmatrix}$
Newton $\boldsymbol{x_n}$	$(0.000, 0.000)$	$(-0.286, -0.143)$	$(-0.377, -0.189)$	$(-0.383, -0.192)$	$(-0.383, -0.192)$	$(-0.383, -0.192)$
BFGS $\hat{\mathbf{H}}_n$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 2.595 & 0.797 \\ 0.797 & 1.399 \end{pmatrix}$	$\begin{pmatrix} 2.018 & 0.509 \\ 0.509 & 1.254 \end{pmatrix}$	$\begin{pmatrix} 3.136 & 1.068 \\ 1.068 & 1.534 \end{pmatrix}$	$\begin{pmatrix} 3.582 & 1.291 \\ 1.291 & 1.645 \end{pmatrix}$	$\begin{pmatrix} 3.286 & 1.143 \\ 1.143 & 1.572 \end{pmatrix}$
BFGS $\boldsymbol{x_n}$	$(0.000, 0.000)$	$(-2.000, -1.000)$	$(-0.668, -0.334)$	$(-0.246, -0.123)$	$(-0.407, -0.203)$	$(-0.385, -0.193)$

Note that BFGS has already almost converged to the same answer as Newton's method, even though the estimated Hessian has not (yet) converged to the true Hessian.

Appendix A

Advice on Revision

A few comments about what you can most productively learn. Please note that the final form of exam questions is up to the Prelims (or, for *Mathematics for CS&Philosophy*, Finals) Examiners. However, I will advocate the following to them as well.

First, feel no need to learn the details of the examples. They were to exemplify parts of the syllabus, and are not on the syllabus themselves.

Second, feel no need to learn the exact implementations that I have given in these notes. Certainly learn the mathematical formulae for the iterations, or for numerical integrals, or for approximations. Understand how the algorithms can be implemented with good time complexity, but do not memorize how to juggle the variables in tightly-written code.

Memorizing Formulae

Revision should be about expanding your understanding, and I do not think it productive for you to memorize too many formulae. You should certainly know the rules for differentiation and the statements of theorems or lemmas. But throughout the notes I have marked some formulae with the symbol $\triangle$. If an exam question were to use the corresponding technique or formula, I would advise the examiners to give it as a hint (possibly amongst other hints, or in a simplified form). $\triangle$

To be clear: if a formula is marked with △, you *should* understand it, and you might be asked to demonstrate that you do, but you should not have to memorize it. In some cases I have indicated, underneath △, what part of the formula you *are* expected to learn.

Topics Not On Syllabus

Some topics were mentioned in the lecture notes as a starting point for further reading, but are not on the syllabus:

- The proofs in the 'bonus mathematics' sections: proofs of Taylor's theorem, the method of Lagrange multipliers, the error bound for Trapezium or Simpson's rule, acceleration by Aitken's method, quadratic convergence of Newton's method in more than one dimension, convergence of gradient descent.
- Formal (limit) definition of continuity, derivative, or Jacobian.
- The statement of Taylor's theorem in $n > 1$ dimensions that uses partial derivatives, beyond the second-order term; instead, learn the simplified version Lemma 1.14.
- The Karush-Kuhn-Tucker conditions for constrained optimization. You *should* know, however, that the Lagrange multiplier(s) must be non-negative for inequality constraints, and understand why.
- Boole's rule.
- The following Monte Carlo integration methods: stratified sampling, importance sampling, and control variates. Know that they exist and broadly how they work, but you do not need to know any of the details.
- The formal definition of forward and backward error (as discussed on page 78). But you *should* know the various options for termination criteria of iterative methods.
- Aitken's Δ^2 process.
- Halley's method, Muller's method, IQI, Brent's methods for root-finding and optimization. You should know that these methods exist and roughly what they do, but no details are expected. On the other hand, it would not be unreasonable for an exam question to *give* you a method like these, and then ask you to analyze/use/comment on it.

- Lemma 5.6. If you needed to know this, or a similar result, it would be given to you.
- The Nelder-Mead method.
- The Wolfe conditions.
- Hessian or Jacobian modification in Newton's method.
- Conjugate gradient descent.
- Minibatch or stochastic gradient descent.
- Details of how Broyden's method and BFGS are derived, or the formulae for Jacobian and Hessian update.

How To Do Well

The best way to become proficient with a tool is to use it.

Index