Numerical Analysis 2018-23

Gustav Delius

2025-04-26

Table of contents

1 Introduction

In our digital age, mathematics isn't just an abstract pursuit; it's an essential tool that powers a vast array of applications. From weather forecasting to black hole simulations, from urban planning to medical research, the application of mathematics has become indispensable. Central to this applied force is Numerical Analysis.

Numerical Analysis is the discipline that bridges continuous mathematical theories with their concrete implementation on digital computers. These computers, by design, work with discrete quantities, and translating continuous problems into this discrete realm isn't always straightforward. So why should you want to venture into Numerical Analysis?

1. **Precision and Stability:** Computers, despite their power, can introduce significant errors if mathematical problems are implemented without care. Numerical Analysis offers techniques to ensure we obtain results that are both accurate and stable.

2. **Efficiency:** Real-world applications often demand not just correctness, but efficiency. By grasping the methods of Numerical Analysis, we can design algorithms that are both accurate and resource-efficient.

3. **Broad Applications:** Whether your interest lies in physics, engineering, biology, finance, or many other scientific fields, Numerical Analysis provides the computational tools to tackle complex problems in these areas.

4. **Basis for Modern Technologies:** Core principles of Numerical Analysis are foundational in emerging fields such as artificial intelligence, quantum computing, and data science.

In this module, we'll explore the key techniques, algorithms, and principles of Numerical Analysis that enable us to translate mathematical problems into computational solutions. We'll delve into the challenges that arise in this translation, the strategies to overcome them, and the interaction of theory and practice.

By the end, you won't merely understand the methods of scientific computing; you'll be equipped to apply them efficiently and effectively in diverse scenarios. With a strong foundation in Numerical Analysis, you'll be better prepared to engage with the practical challenges of the modern world.

The main topic of this module is finding *approximate solutions* to various mathematical problems. A typical example for such a problem would be to find $x > 0$ such that $\cos(x) = x$.

There is indeed one and only one such x; this is apparent from a graph (Figure 1.1), and can also be proven rigorously without too much difficulty. However, it seems impossible to find a *closed form* expression for this x: It's not a fraction, a square root of a fraction, an integer multiple of π, etc. For practical purposes, the best we can give is a *numerical approximation* of this x, that is, we can compute it to, say, 10 decimal digits of accuracy.

This situation is pervasive in mathematics and its applications. Many problems of practical importance can be solved only approximately, and for others an approximation is much more efficient to find than the closed-form expression.

In Numerical Analysis, we aim to find *approximation algorithms* for mathematical problems, i.e., schemes that allow us to compute the solution approximately. These algorithms use only elementary operations $(+, -, \times, /)$ but often a long sequence of them, so that in practice they need to be run on computers. This part of the problem—how to implement such algorithms on a computer—is called Scientific Computing.
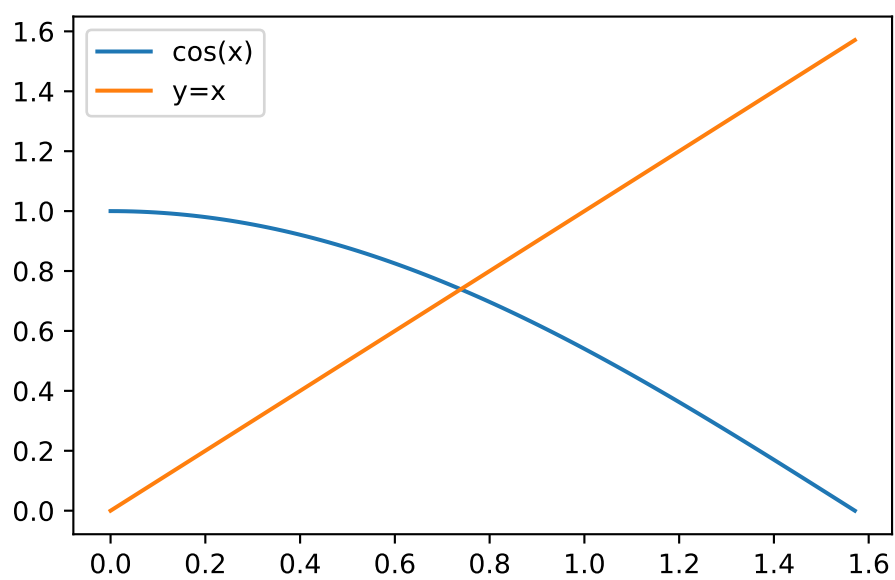
Figure 1.1

2 Errors

As we embark on our exploration of Numerical Analysis, it might be tempting to dive directly into the algorithms, tools, and techniques that form the core of this discipline. Yet, there's a foundational aspect that we must address before we venture further: errors. Just as a builder needs to understand the properties and potential weaknesses of materials before constructing a robust building, a numerical analyst must grasp the nuances of errors in order to craft effective and reliable algorithms.

Why start with errors? Here's why this is the perfect launching pad:

1. **Inherent Limitations of Computers:** Digital computers, by their nature, represent numbers using a finite number of bits. This limitation leads to rounding and truncation errors, which can cascade through calculations and produce misleading results. Understanding these errors equips us to prevent or minimize them.

2. **Preventing Cumulative Mistakes:** Even small errors, when compounded over many iterations or operations, can result in significant discrepancies. Grasping the origins and types of these errors is vital to ensuring our computations remain on track.

3. **Building Robust Algorithms:** A deep understanding of errors allows us to design algorithms that are both accurate and stable. Ignoring errors or misunderstanding them can lead to algorithms that produce wildly inaccurate results, even if they seem correct on the surface.

4. **Gaining Confidence in Results:** When we know where errors come from and how they manifest, we can better judge the reliability of our results. This confidence is crucial when numerical solutions are used for critical applications in engineering, medicine, finance, and more.

Errors are not just a minor consideration; they are central to the field of Numerical Analysis. By addressing them upfront, we lay a solid foundation upon which the rest of our study will be built. This chapter will shed light on the nature of rounding errors, demonstrate how seemingly benign calculations can lead to substantial inaccuracies, and offer insights into mitigating these pitfalls.

So, as we delve into the world of errors, remember: understanding our limitations and challenges is the first step to overcoming them. Let us begin our journey with a clear-eyed view of the obstacles, so we are best prepared to navigate the rich landscape of Numerical Analysis.

As mentioned, our goal is to find *approximate* rather than exact solutions to problems. That is, our results will always contain errors. Errors can arise in several ways:

(a) The input data (e.g., experimental data) may contain errors.

(b) On computers, numbers are represented only to a finite number of digits, thus we encounter *roundoff errors* – when storing numbers in the first place, or in arithmetic operations.

(c) The numerical method as such usually involves *approximation errors* of some kind.

Item a is outside our scope in this course. In this chapter, we will discuss roundoff errors (item b). Approximation errors (item c) will be discussed in later chapters.

2.1 Floating point numbers

We can write any real number in floating point notation

$$x = \pm 0.d_1 d_2 d_3 \ldots d_k d_{k+1} d_{k+2} \cdots \times 10^n. \tag{2.1}$$

The number $\pm 0.d_1 d_2 d_3 \ldots d_k \ldots$ is called the **mantissa** and the power of ten is called the **exponent**. However, a computer can work only with finite set of digits. Therefore, we truncate the mantissa and limit the range of the exponent n to produce a k-digit (decimal) machine number. This can be done in two ways:

- **Chopping** results in
$$x^* = \pm 0.d_1 d_2 \ldots d_k \times 10^n. \tag{2.2}$$

- **Rounding** results in
$$x^* = \pm 0.d_1 d_2 \ldots d_{k-1} d_k^* \times 10^n, \tag{2.3}$$

  where x^* is obtained by adding $0.5 \times 10^{n-k}$ (or, equivalently, $5 \times 10^{n-k-1}$ to x and chopping.

Either machine number is called (decimal) k-digit **floating point representation** of x, and denoted by $fl(x)$.

Example 2.1. The floating point representation of $\pi = 3.14159265\ldots$ is

$$\pi = 0.314159265\ldots \times 10^1. \tag{2.4}$$

The five-digit floating-point form of π is given by

$$fl(\pi) = 0.31415 \times 10^1 = 3.1415 \tag{2.5}$$

if chopping is used. In the case of rounding, we first compute

$$\begin{aligned} \pi + 0.5 \times 10^{1-5} &= 0.314159265\ldots \times 10^1 + 0.000005\ldots \times 10^1 \\ &= 0.314164265\ldots \times 10^1 \end{aligned} \tag{2.6}$$

and then chop to obtain

$$fl(\pi) = 0.31416 \times 10^1 = 3.1416. \tag{2.7}$$

Remark. In this module, we represent numbers in *decimal* (base 10) form. Computers actually use *binary* (base 2) form, but we will usually ignore this difference. Industry standards exist for representing floating point numbers in binary form. For example, a *double-precision floating-point number* according to Binary Floating Point Arithmetic Standard 574–1985 developed by the IEEE (Institute for Electrical and Electronic Engineers) consists of

(a) 1-bit (binary digit) sign indicator,

(b) 11-bit exponent with a base of 2,

(c) 52-bit mantissa.

This provides between 15 and 16 decimal digits of precision and a range of approximately 10^{-308} to 10^{+308} (and similarly -10^{+308} to -10^{-308} for negative numbers). If numbers occurring in calculations have a magnitude of less than the lower limit ($\approx 10^{-308}$), they are set to zero; this is called **underflow**. Numbers greater than the upper limit ($\approx 10^{+308}$) result in **overflow** and computations are halted.

The error resulting from replacing a number by its floating-point form is called **roundoff error**. In order to quantify roundoff errors (and other errors), we introduce:

Definition 2.1. If x^* is an approximation to x, the **absolute error** is

$$E_{abs} := |x - x^*|, \tag{2.8}$$

and the **relative error** (for $x \neq 0$) is

$$E_{rel} := \frac{|x - x^*|}{|x|}. \tag{2.9}$$

With these notions, our roundoff errors can be estimated. If $x^* = fl(x)$ is obtained by k-digit chopping, then (prove it!)

$$\frac{|x - x^*|}{|x|} \leq 10^{-k+1}. \tag{2.10}$$

If $x^* = fl(x)$ is obtained by k-digit rounding, then (prove it!)

$$\frac{|x - x^*|}{|x|} \leq 0.5 \times 10^{-k+1}. \tag{2.11}$$

Definition 2.2. Suppose that a number x^* approximates x and that $x^* \neq x$. We say that x^* approximates x to k **significant digits** (or x^* has k significant digits with respect to x) if

  (i) the floating point representations of x^* and x have the same exponent, say n;

  (ii) the exponent of $|x^* - x|$ is $n - k$ and the first digit of its mantissa is less than or equal to 5.

In other words, x^* approximates x to k significant digits if and only if:

$$0.1 \times 10^{n-k} \leq |x^* - x| < 0.5 \times 10^{n-k}. \tag{2.12}$$

Example 2.2. Suppose that $x = 23.492 = 0.23492 \times 10^2$ and $x^* = 23.489 = 0.23489 \times 10^2$. Then the absolute error is

$$|x^* - x| = 0.00003 \times 10^2 = 0.3 \times 10^{-2}, \tag{2.13}$$

so x^* approximates x to 4 significant digits. The relative error is

$$\frac{|x - x^*|}{|x|} = \frac{0.3 \times 10^{-2}}{0.23492 \times 10^2} \approx 0.1277 \times 10^{-3}. \tag{2.14}$$

It can be shown that if $x^* = fl(x)$ is obtained by k-digit rounding, then x^* has at least k significant digits with respect to x.

Further reading: Section 1.2 of (Burden and Faires 2010).

2.2 Error-generating computations

In each arithmetic computation performed by a computer on floating point numbers, additional round-off error may occur. Let us illustrate this in the example of addition. Let x, y be two real numbers and $x^* = fl(x)$, $y^* = fl(y)$. We denote the computer-performed addition of x^* and y^* as $x^* \oplus y^*$, as opposed to the normal addition $+$. In a simple model, we will assume that $\oplus$ is just normal addition followed by rounding:

$$x^* \oplus y^* = fl(x^* + y^*). \tag{2.15}$$

Rounding is unavoidable in general due to the finite length of the mantissa, and generates additional error. Let us quantify this by estimating the absolute error of $x^* \oplus y^*$, as an approximation for $x + y$. Using the triangle inequality, we obtain

$$\begin{aligned}
|x + y - x^* \oplus y^*| &= |x - x^* + y - y^* + x^* + y^* - fl(x^* + y^*)| \\
&\leq |x - x^*| + |y - y^*| + |x^* + y^* - fl(x^* + y^*)|.
\end{aligned} \tag{2.16}$$

Thus, the total absolute error is the sum of the individual absolute errors for x^* and y^*, plus an extra term arising from rounding.

From this, it might seem that addition is relatively unaffected by rounding—after all, the rounding error occurs in the last digit of the mantissa. However, there are several situations where the rounding error can become very relevant. First, if we are performing a large number of additions (say, several thousands), then every one of these can cause an additional (small) rounding error, and the sum of these may be quite large. Second, the *relative* error can increase significantly in the case of *subtraction*, which is included in the above by considering $y < 0$.

Example 2.3. Consider 5-digit rounding and set

$$x = 1 + \frac{\pi}{1000}, \quad x^* = 1.0031, \quad y = y^* = -1. \tag{2.17}$$

In this case, $x^* \oplus y^* = fl(0.0031) = 3.1 \times 10^{-3}$. The absolute errors of x^* and $x^* \oplus y^*$, as well as the relative error of x^*, are approximately 0.4×10^{-4}. However, the relative error of $x^* \oplus y^*$ is approximately $0.4 \times 10^{-4}/3.1 \times 10^{-3} \approx 10^{-2}$—subtraction has increased it significantly, and $x^* \oplus y^*$ has only 2 valid digits.

This phenomenon is called **cancellation of digits** and occurs whenever two almost equal numbers are subtracted from each other.

Multiplication behaves similar to addition, only that the *relative* error is the sum of the individual relative errors, plus an extra error arising from rounding. Also, multiplying a machine number x^* by a large number (or dividing by a small number) will not much affect the relative error, but significantly increase the *absolute* error.

To summarize, typical sources of significant roundoff error are

(a) performing a large number of individual arithmetic operations (e.g., additions),

(b) subtraction of almost equal numbers,

(c) multiplication by a very large number, or division by a very small number.

Where these occur, it can be worthwhile to reformulate the algebraic expressions suitably in order to circumvent the roundoff problem. Several examples for this can be found in Practical A1.

Roundoff errors occur in virtually all numerical methods that we will consider. However, for reasons of simplicity, we will usually ignore roundoff errors in the theoretical discussion, and mention them only where they become particularly relevant.

Further reading: Section 1.2 of (Burden and Faires 2010).

3 Solving nonlinear equations

Having navigated the intricacies of rounding errors and understanding the nuances they introduce, it is time to delve into one of the cornerstones of Numerical Analysis: solving nonlinear equations, also known as the *root-finding problem*. The challenge of finding the roots of equations that do not lend themselves to simple algebraic solutions has been a persistent one throughout the history of mathematics. Yet, it is exactly these challenges that have spurred the development of a rich array of techniques, many of which are iterative in nature.

Why start with nonlinear equations? The reasons are manifold:

1. **Pervasiveness of Nonlinear Problems:** In the real world, many problems—ranging from physics to engineering to economics—are inherently nonlinear. Understanding how to tackle these equations is foundational for addressing a multitude of real-world scenarios.

2. **Introduction to Iterative Algorithms:** Solving nonlinear equations provides a natural and practical context to introduce iterative methods. These methods, which involve refining solutions progressively to approach the true answer, are central to many areas within Numerical Analysis.

3. **Variety of Techniques:** This chapter introduces a spectrum of methods, from the simple yet effective bisection method to the powerful Newton's method. By studying a range of techniques, students will gain both a broad overview and a deep understanding of the strategies at their disposal.

4. **Laying Groundwork for Convergence:** As the first foray into iterative solutions, this chapter also paves the way for discussing the order of convergence, a crucial concept that measures the efficiency of iterative processes.

Throughout this chapter, we will dissect each method, understanding its mechanics, advantages, drawbacks, and areas of application. Through hands-on examples and explorations, you will gain not just theoretical knowledge, but practical skills that are immediately applicable.

3.1 Defining the problem

Let $f : [a, b] \to \mathbb{R}$ be a continuous function, i.e. $f \in C[a, b]$. The task is to find an $p \in [a, b]$ such that

$$f(p) = 0. \tag{3.1}$$

The number p is called a **root** of the equation or a **zero** of the function f. In our example in the introduction, we had $f(x) = \cos(x) - x$.

The first question to clarify is whether such a root exists. If we assume that f changes sign on $[a, b]$, that is, that $f(a)f(b) < 0$, then we know (by the Intermediate Value Theorem) that f must have a zero in $[a, b]$. However, in general there could be more than one zero in the interval. For our purposes, we will always assume that f has a unique zero $p \in (a, b)$, and we will investigate our algorithms under that assumptions. In practice, one would first need to choose the interval $[a, b]$ suitably so that it contains a unique zero.

3.2 Bisection method

The *bisection method* is a very simple approach to the root finding problem, and one where the approximation error is very easy to control. The idea behind it is as follows: One divides the interval $[a, b]$ into halves, and decides in which subinterval the zero is located (by looking at the sign of $f(x)$ in the midpoint of the interval). Then one repeats the method for this subinterval, and continues this until sufficient precision is reached.

More formally, the bisection method would be described as follows:

1. Set $a_1 = a$, $b_1 = b$, and $x_1 = (a_1 + b_1)/2$.

2. Compute $f(x_1)$.

3. If $f(x_1) = 0$, then $p = x_1$.

4. If $f(x_1) \neq 0$, then we choose a_2 and b_2 as follows:

    1. if $f(x_1) \cdot f(a_1) > 0$ (i.e. $f(x_1)$ and $f(a_1)$ have the same sign), then we know $p \in (x_1, b_1)$, and we set $a_2 = x_1$ and $b_2 = b_1$;

    2. if $f(x_1) \cdot f(a_1) < 0$ (i.e. $f(x_1)$ and $f(a_1)$ have opposite signs), then we know $p \in (a_1, x_1)$, and we set $a_2 = a_1$ and $b_2 = x_1$.

5. We repeat the procedure for the interval $[a_2, b_2]$ and so on until we compute p with a specified accuracy.

We now show that this method actually works as expected: namely, that the sequence (x_n) converges to the root p, and that we can control the error of (x_n) approximating p.

Theorem 3.1. *Suppose that $f \in C[a, b]$ has a unique zero p and that $f(a) \cdot f(b) < 0$. Then the sequence $\{x_n\}$ of the bisection method converges to p, and*

$$|x_n - p| \leq \frac{b - a}{2^n} \quad \text{for all } n \geq 1. \tag{3.2}$$

Proof. For $n \geq 1$, we have by construction

$$a_n \leq p \leq b_n. \tag{3.3}$$

Subtracting x_n from this inequality yields

$$a_n - x_n \leq p - x_n \leq b_n - x_n. \tag{3.4}$$

Recalling that $x_n = (a_n + b_n)/2$, we can rewrite this as

$$-\frac{b_n - a_n}{2} \leq p - x_n \leq \frac{b_n - a_n}{2}, \tag{3.5}$$

or equivalently, since $b_n - a_n = (b - a)/2^{n-1}$,

$$|p - x_n| \leq \frac{1}{2}(b_n - a_n) = \frac{b - a}{2^n}. \tag{3.6}$$

This implies that $x_n \to p$ as $n \to \infty$ and also gives us the proposed error estimate. $\square$

Example 3.1. Let us find an approximation to $\sqrt{2}$ correct to within 10^{-3}. Consider the function $f(x) = x^2 - 2$. One of its zeros is $\sqrt{2}$. If we take $a = 1$ and $b = 2$, then $f(a) = -1 < 0$ and $f(b) = 2 > 0$. In addition, we know that $\sqrt{2}$ is the unique zero of f on the interval $[a, b]$. Now we can apply the bisection method and calculate an approximation to $\sqrt{2}$.

First, we calculate the number of steps necessary to compute $\sqrt{2}$ with accuracy 10^{-3}. This requires to find an integer N such that

$$|x_N - p| \leq \frac{b - a}{2^N} = 2^{-N} < 10^{-3}. \tag{3.7}$$

Since $2^{10} = 1024$, this inequality is satisfied if we take $N = 10$. Table 3.1 shows the results of the bisection method.

Table 3.1: Ten iterations of the bisection method for approximating $\sqrt{2}$

	n	a	b	x	f_x
0	1	1.000000	2.000000	1.500000	0.250000
1	2	1.000000	1.500000	1.250000	-0.437500
2	3	1.250000	1.500000	1.375000	-0.109375
3	4	1.375000	1.500000	1.437500	0.066406
4	5	1.375000	1.437500	1.406250	-0.022461
5	6	1.406250	1.437500	1.421875	0.021729
6	7	1.406250	1.421875	1.414062	-0.000427
7	8	1.414062	1.421875	1.417969	0.010635
8	9	1.414062	1.417969	1.416016	0.005100
9	10	1.414062	1.416016	1.415039	0.002336

Thus, $\sqrt{2} \approx 1.4150$ with absolute error smaller than 10^{-3} (Note that $\sqrt{2} = 1.414213562373095\ldots$, so that the actual error is $E = |x_{10} - \sqrt{2}| = 0.0008255001269048545\ldots$).

This example illustrates that the method converges, although rather slowly—we will come across much faster methods in due course. Also, it appears that the method is unnecessarily inefficient—note that x_7 was a better approximation to $\sqrt{2}$ than x_{10}. On the other hand, the advantage is the simple and clear error estimate.

Finally, let us write the algorithm in a form that is more adapted to a computer implementation. We do not use a specific programming language here, but **pseudocode**. The bisection method is formulated in Algorithm 2.1. This matches our heuristic description above; note however that there are a number of efficiency improvements: We do not need to store the entire sequences $\{a_n\}$, $\{b_n\}$, $\{x_n\}$, but only their "current" values, which we label a, b, x. Also, by storing intermediate results, one tries to evaluate f as few times as possible, which means that the implementation is efficient even if evaluating f is time consuming.

Algorithm 2.1: Bisection method

```
 1 : function Bisection(f, a, b, N)      ♯ function f, interval [a, b], number of steps N
 2 :     Fₐ ← f(a)
 3 :     for k from 1 to N do
 4 :         x ← (a + b)/2;   Fₓ ← f(x)   ♯ computes midpoint and function value there
 5 :         if Fₓ = 0 then               ♯ zero has already been found
 6 :             break
 7 :         else  if Fₓ · Fₐ < 0 then    ♯ zero is in [a, x]
 8 :             b ← x
 9 :         else                         ♯ zero is in [x, b]
10 :             a ← x;   Fₐ ← Fₓ
11 :         end if
12 :     end for
13 :     return x
14 : end function
```

Further reading: Sections 1.3, 2.1 of (Burden and Faires 2010).

3.3 Method of false position

The *method of false position* (or *regula falsi* method) is a slight variation of the bisection method. The idea is to choose a more efficient intermediate point x than just the midpoint of the interval $[a, b]$. Instead, one joins the points $(a, f(a))$ and $(b, f(b))$ with a straight line (the *secant* line) and find the point where this line intersects the x axis. Since the line is given by

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a), \tag{3.8}$$

this intersection point is located at

$$x = a - \frac{f(a)}{f(b) - f(a)}(b - a). \tag{3.9}$$

This method is illustrated in Figure 3.1.
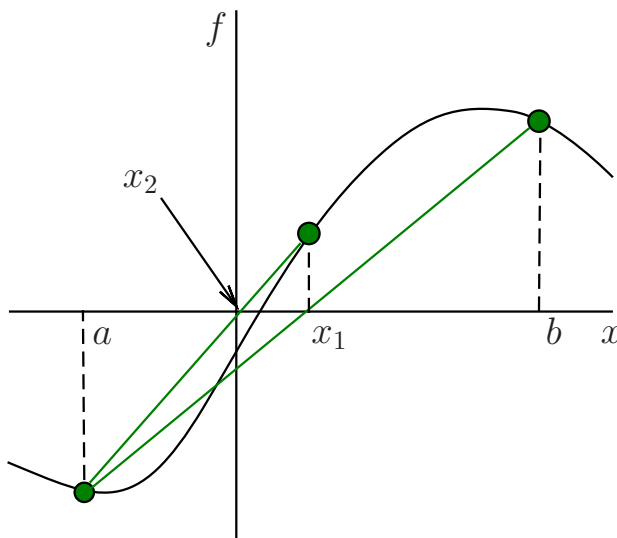
Figure 3.1: Method of false position

The algorithm for the method of false position is just the same as Algorithm 2.1, except that in line 4, the statement $x \leftarrow (a + b)/2$ is replaced by $x \leftarrow a - \frac{f(a)}{f(b)-f(a)}(b - a)$. (Again, one may optimize this to avoid evaluating f too often.)

Example 3.2. Let us return to the problem of finding an approximation to $\sqrt{2}$ correct to within 10^{-3} by finding a zero of the function $f(x) = x^2 - 2$. Again taking $a = 1$ and $b = 2$, we have $f(a) = -1 < 0$ and $f(b) = 2 > 0$, and so $\sqrt{2}$ is the unique zero of f on the interval $[a, b]$. Carrying out 10 steps of the method of false position, we obtain the results in Table 3.3. The desired accuracy is obtained after the fourth step. Moreover, comparing with Table 3.1, it appears that the method of False Position converges much faster than the Bisection method in this example.

Table 3.3: Numerical example for method of false position

k	x_n	$f(x_n)$	$|x_n - \sqrt{2}|$
1	1.333333333	-0.222222223	0.080880229
2	1.400000000	-0.040000000	0.014213562
3	1.411764706	-0.006920415	0.002448856
4	1.413793104	-0.001189059	0.000420458
5	1.414141414	-0.000204061	0.000072148
6	1.414201183	-0.000035014	0.000012379
7	1.414211438	-0.000006009	0.000002124
8	1.414213198	-0.000001031	3.64×10^{-7}
9	1.414213500	-1.76×10^{-7}	6.20×10^{-8}
10	1.414213552	-2.90×10^{-8}	1.00×10^{-8}

The method of false position converges faster than the bisection method in many cases. Unfortunately, it is not guaranteed that the method of false position will always converge faster—there are other examples where it actually turns out to be *slower*. A precise error estimate for the method of false position is hard to find, so we cannot predict how fast the method will converge.

Further reading: Sections 2.3 of (Burden and Faires 2010).

3.4 Fixed point iteration

We will now investigate a different problem that is closely related to root finding: the fixed point problem. Given a function g (of one real argument with real values), we look for a number p such that

$$g(p) = p. \tag{3.10}$$

This p is called a **fixed point** of g.

Any root finding problem $f(x) = 0$ can be reformulated as a fixed point problem, and this can be done in many (in fact, infinitely many) ways. For example, given f, we can define $g(x) := f(x) + x$; then

$$f(x) = 0 \quad \Leftrightarrow \quad g(x) = x. \tag{3.11}$$

Just as well, we could set $g(x) := \lambda f(x) + x$ with any $\lambda \in \mathbb{R} \setminus \{0\}$, and there are many other possibilities.

The heuristic idea for approximating a fixed point of a function g is quite simple. We take an initial approximation x_0 and calculate subsequent approximations using the formula

$$x_n := g(x_{n-1}). \tag{3.12}$$

A graphical representation of this sequence is shown in Figure 3.2.
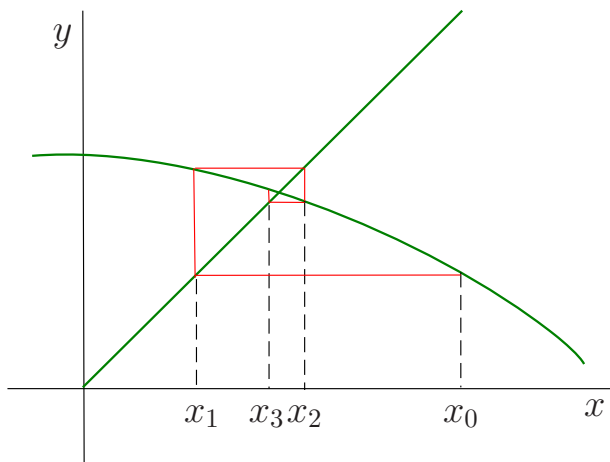
Figure 3.2: Fixed point iteration

Why is this sequence expected to approximate a fixed point? Suppose for a moment that the sequence (x_n) converges to some number p, and that g is continuous. Then

$$p = \lim_{n \to \infty} x_n = \lim_{n \to \infty} g(x_{n-1}) = g\left(\lim_{n \to \infty} x_{n-1} \right) = g(p). \tag{3.13}$$

Thus, *if* the sequence converges, then it converges to a fixed point. However, this resolves the problem only partially. One would like to know:

- Under what conditions does the sequence (x_n) converge?

- How fast is the convergence, i.e., can one obtain an estimate for the approximation error?

The following theorem gives us the answers to those questions. We will revisit this theorem—in a more general case—later.

Theorem 3.2 (Fixed Point Theorem). *Suppose that $g : [a, b] \to [a, b]$ is differentiable, and that there exists $0 < k < 1$ such that*

$$|g'(x)| \le k \quad \text{for all } x \in (a, b). \tag{3.14}$$

Then, g has a unique fixed point $p \in [a, b]$; and for any choice of $x_0 \in [a, b]$, the sequence defined by

$$x_n := g(x_{n-1}) \quad \text{for all } n \ge 1 \tag{3.15}$$

converges to p. The following estimate holds:

$$|p - x_n| \le k^n |p - x_0| \quad \text{for all } n \ge 1. \tag{3.16}$$

Proof. We first show that g has a fixed point p in $[a, b]$. If $g(a) = a$ or $g(b) = b$ then g has a fixed point at an endpoint. If not, then it must be true that $g(a) > a$ and $g(b) < b$. This means that the function $h(x) := g(x) - x$ satisfies

$$h(a) = g(a) - a > 0, \quad h(b) = g(b) - b < 0 \tag{3.17}$$

and since h is continuous on $[a, b]$ the Intermediate Value Theorem guarantees the existence of $p \in (a, b)$ for which $h(p) = 0$, equivalently $g(p) = p$, so that p is a fixed point of g.

To show that the fixed point is unique, suppose that $q \ne p$ is a fixed point of g in $[a, b]$. The Mean Value Theorem implies the existence of a number $\xi \in (\min\{p, q\}, \max\{p, q\}) \subseteq (a, b)$ such that

$$\frac{g(p) - g(q)}{p - q} = g'(\xi). \tag{3.18}$$

Then

$$|p - q| = |g(p) - g(q)| = |(p - q)g'(\xi)| = |p - q||g'(\xi)|$$
$$\leq k|p - q| < |p - q|, \tag{3.19}$$

where the inequalities follow from Eq. 3.14. This is a contradiction, which must have come from the assumption $p \neq q$. Thus $p = q$ and the fixed point is unique.

Since g maps $[a, b]$ onto itself, the sequence $\{x_n\}$ is well defined. For each $n \geq 0$ the Mean Value Theorem gives the existence of a $\xi \in (\min\{x_n, p\}, \max\{x_n, p\}) \subseteq (a, b)$ such that

$$\frac{g(x_n) - g(p)}{x_n - p} = g'(\xi). \tag{3.20}$$

Thus for each $n \geq 1$ by Eq. 3.14, Eq. 3.15

$$|x_n - p| = |g(x_{n-1}) - g(p)| = |(x_{n-1} - p)g'(\xi)|$$
$$= |x_{n-1} - p||g'(\xi)| \leq k|x_{n-1} - p|. \tag{3.21}$$

Applying this inequality inductively, we obtain the error estimate Eq. 3.16. Moreover since $k < 1$ we have

$$\lim_{n \to \infty} |x_n - p| \leq \lim_{n \to \infty} k^n |x_0 - p| = 0, \tag{3.22}$$

which implies that (x_n) converges to p. $\qquad \square$

The following example shows why the conditions of the Theorem 3.2 are important.

Example 3.3. The equation

$$f(x) = x^2 - 2 = 0 \tag{3.23}$$

has a unique root $\sqrt{2}$ in $[1, 2]$. There are many ways of writing this equation in the form $x = g(x)$; we consider two of them:

$$x = g(x) = x - (x^2 - 2), \quad x = h(x) = x - \frac{x^2 - 2}{3}. \tag{3.24}$$

Which of these fixed point problems generate a rapidly converging sequence?

It is easy to see that the condition of the fixed point theorem is not satisfied by the function g on $[1, 2]$: Namely, $g(2) = 0$, so that $g(2) \notin [1, 2]$. On the other hand h satisfies the conditions because

$$h(x) \in [1, 2] \quad \left(\max_{x \in [1,2]} h(x) = \frac{17}{12}, \min_{x \in [1,2]} h(x) = \frac{4}{3} \right) \quad \text{and} \quad |h'(x)| \leq 1/3. \tag{3.25}$$

Thus, the fixed point theorem guarantees that the sequence (x_n) produced by the fixed point iteration procedure $x_n = h(x_{n-1})$ converges to $p = \sqrt{2}$.

Table 3.4: Fixed point iteration converges quickly for h

n	$x_k = g(x_{n-1})$	$|x_n - \sqrt{2}|$
0	1.0	0.414213562
1	2.0	0.585786438
2	0.0	1.414213562
3	2.0	0.585786438
4	0.0	1.414213562
5	2.0	0.585786438
6	0.0	1.414213562
7	2.0	0.585786438
8	0.0	1.414213562

Table 3.5: Fixed point iteration converges quickly for h

n	$x_k = h(x_{n-1})$	$\lvert x_n - \sqrt{2}\rvert$
0	1.0	0.414213562
1	1.333333333	0.080880229
2	1.407407407	0.006806155
3	1.413808871	0.000404691
4	1.414190363	0.000023199
5	1.414212235	0.000001327
6	1.414213486	7.6×10^{-8}
7	1.414213558	4.0×10^{-9}
8	1.414213562	0.0

Table 3.5 shows the sequences generated by fixed point iteration on g and h with start value $x_0 = 1$. It is apparent that the sequence generated by h converges quite fast, whereas the one generated by g does not converge at all. The example is explored further in Practical 2.

The example illustrates that one needs to be careful in rewriting root finding problems as fixed point problems—there are many ways to do so, but not all lead to a good approximation. Note at this point that Theorem 3.2 gives only sufficient conditions for convergence; in practice, convergence might occur even if the conditions are violated.

For implementing the fixed point method as a computer algorithm, there's one more complication to be taken into account: how many steps of the iteration should be taken, i.e., how large should n be chosen, in order to reach the desired precision? For the bisection method, the error estimate Eq. 3.2 allows an easy answer. The estimate in fixed point iteration, Eq. 3.16, turns out to be more difficult to use. While we can certainly estimate

$$\lvert x_0 - p\rvert \le \max\{\lvert x_0 - a\rvert, \lvert x_0 - b\rvert\}, \tag{3.26}$$

the constant k (which influences the speed of convergence significantly) is often difficult to obtain in practice, since it involves estimates on the derivative of g.

Instead, one uses a different **stopping condition** for the algorithm. Since the sequence is expected to converge rapidly, one uses the difference $\lvert x_n - x_{n-1}\rvert$ to measure the precision reached. If this difference is below a specified limit, say τ, the iteration is stopped. Since it is possible that the iteration does *not* converge—see the example above—one would also stop the iteration (with an error message) if a certain number of steps is exceeded, in order to avoid infinite loops. Algorithm 2.2 shows all this combined in pseudocode.

Algorithm 2.2: Fixed point iteration

```
 1 : function FixedPoint(g, x₀, τ, N)          ♯ function g, start point x_0,
 2 :    x ← x₀;  n ← 0                          ♯ tolerance τ, max. num. of
 3 :    loop                                    ♯ iterations N
 4 :       y ← x;  x ← g(x)
 5 :       if |y − x| < τ then                  ♯ Desired tolerance reached
 6 :          break
 7 :       end  if
 8 :          n ← n + 1
 9 :       if n > N then                        ♯ Max. num. of iterations
10 :          exception("Iteration does not converge")  ♯ reached
11 :       end if
12 :    end loop
13 :    return x
14 : end function
```

Further reading: Section 2.2 of (Burden and Faires 2010).

3.5 Newton's method

Newton's method is one of the most effective numerical methods for solving a root-finding problem $f(x) = 0$. To derive this method, we need Taylor's theorem in its simplest form, for a real-valued function of one real variable, with the remainder in Lagrange form.

Theorem 3.3 (Taylor's Theorem in 1d)**.** *Suppose that $f \in C^{k+1}(I)$ for some $I \subset \mathbb{R}$. For each $a \in I$ and $x \in I$, there exists ξ between a and x such that*

$$f(x) = f(a) + (x-a)f'(a) + \cdots + \frac{(x-a)^k}{(k)!}f^{(k)}(a) + \frac{(x-a)^{k+1}}{(k+1)!}f^{(k+1)}(\xi). \qquad (3.27)$$

Here $f^{(k)}(a)$ denotes the kth derivative of f at $x = a$.

The idea behind Newton's method is as follows. We assume that $f \in C^2[a, b]$. Let p be the root of $f(x) = 0$, and let x^* be an (initial) approximation to p such that $|x^* - p|$ is small. The first-order Taylor expansion of $f(x)$ about x^* is

$$f(x) = f(x^*) + (x - x^*)f'(x^*) + \frac{(x - x^*)^2}{2}f''(\xi) \qquad (3.28)$$

with some ξ between x and x^* (so that $\xi \in [a, b]$). Setting $x = p$, we have

$$0 = f(x^*) + (p - x^*)f'(x^*) + \frac{(p - x^*)^2}{2}f''(\xi). \qquad (3.29)$$

Since $|x^* - p|$ is small, we ignore the last term in this formula and obtain

$$0 \approx f(x^*) + (p - x^*)f'(x^*) \quad \Rightarrow \quad p \approx x^* - \frac{f(x^*)}{f'(x^*)}. \qquad (3.30)$$

Of course, p computed using this formula will not be the exact root, but it is natural to expect it to be a better approximation than x^*.

Let us formulate this as an iterative algorithm. Starting with an initial approximation x_0, we define a sequence of approximation values (x_n) by

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}. \qquad (3.31)$$

Note that this is just a fixed point iteration with the function

$$g(x) := x - \frac{f(x)}{f'(x)}. \tag{3.32}$$

Geometrically, at the nth iteration, we consider the tangent to $f(x)$ at $x = x_{n-1}$, and finding its intersection with the x axis—see Figure 3.3.
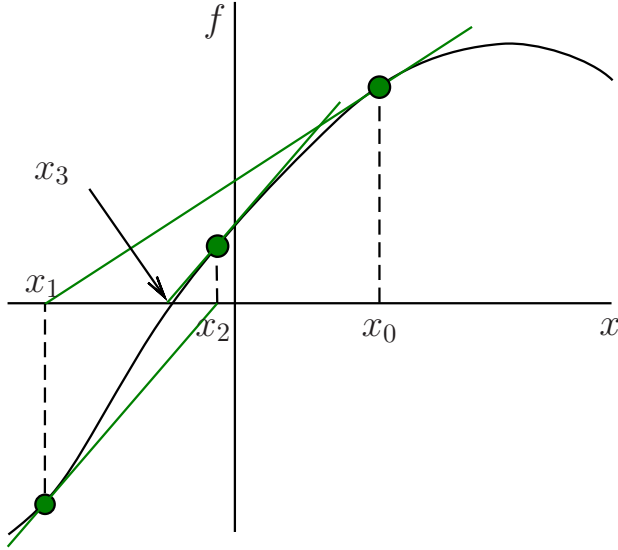
Figure 3.3: Newton's method

So far, this was a heuristical motivation. We would like a mathematical statement that guarantees convergence of $\{x_n\}$ to a root of the equation; this is the content of the following theorem.

Theorem 3.4. *Let $f \in C^2[a,b]$. If $p \in (a,b)$ is such that $f(p) = 0$ and $f'(p) \neq 0$, then there exists a $\delta > 0$ such that the sequence (x_n) generated by Newton's method converges to p for any initial approximation $x_0 \in [p - \delta, p + \delta]$.*

The idea of the proof is to show that for some δ the conditions of the fixed point theorem for function $g(x)$ (given by Eq. 3.32) are satisfied. The proof can be found in the book by (Burden and Faires 2010).

We did not give an explicit error estimate, but it will turn out later that Newton's method, under suitable conditions, converges even much faster than a general fixed point method.

Unfortunately, the conditions of the above theorem are such that it is hard to predict in concrete examples whether Newton's method will produce a converging sequence for a given initial approximation. Sometimes, a combination of the bisection method and Newton's method is used in practice: the first method is employed to obtain a sufficiently good initial approximation for the second one, which then produces a fast and precise approximation.

Example 3.4. For $f(x) = x^2 - 2$, $x \in (0, \infty)$, equation Eq. 3.32 yields the function

$$g(x) = x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - 2}{2x} = \frac{x}{2} + \frac{1}{x}. \tag{3.33}$$

The sequence generated by the formula $x_n = \frac{x_{n-1}}{2} + \frac{1}{x_{n-1}}$ $(n \geq 1)$ converges to the root $\sqrt{2}$ of the equation $f(x) = 0$ for any choice of $x_0 \in (0, \infty)$ (prove it!). Calculations yield the values in Table 3.6, which shows a very fast convergence. (Compare this with the result of fixed point iteration in Table 3.5.)

Table 3.6: Numerical example for Newton's method

n	$x_n = g(x_{n-1})$	$\|x_n - \sqrt{2}\|$
0	*1.0*	*0.414213562*
1	*1.500000000*	*0.085786438*
2	*1.416666667*	*0.002453105*
3	*1.414215686*	*0.000002124*
4	*1.414213562*	*0.0*
5	*1.414213562*	*0.0*

Let us write down the algorithm as a pseudocode. Since Newton's method is a special case of fixed point iteration, this just follows Algorithm 2.2 line by line, with the same stopping conditions. For the sake of completeness, it is shown in Algorithm 2.3.

Algorithm 2.3: Newton's method

```
 1 : function NewtonMethod(f, f′, x₀, τ, N)     ♯ function f, its derivative f′,
 2 :    x ← x₀; n ← 0                            ♯ start point x₀, tolerance τ,
 3 :    loop                                     ♯ max. num. of iterations N
 4 :       v ← f(x)/f′(x); y ← x − v
 5 :       if |y − x| < τ then                   ♯ Desired tolerance reached
 6 :          break
 7 :       end if
 8 :        n ← n + 1; x ← y
 9 :       if n > N then                         ♯ Max. num. of iterations
10 :          exception("Iteration does not converge")  ♯ reached
11 :        end if
12 :    end loop
13 :    return y
14 : end function
```

As a final remark, let us mention that Newton's method works just the same for analytic function on $\mathbb{C}$, rather than real-valued functions on $\mathbb{R}$. The definition of the approximating sequence is identical to Eq. 3.31, where subtraction and division are now read in terms of complex numbers. We will not go into detail here, but you will investigate an example in Practical A3.

Further reading: Section 2.3 of (Burden and Faires 2010).

3.6 Secant method

While Newton's method is often convenient, one problem is that it requires explicit knowledge of the derivative of f. In practice, this may not always be available; e.g., when the function f itself is approximated by a piece of computer code. In this situation, one can use a slightly different method, called the *secant method*.

The idea here is to replace the derivative with a finite difference quotient. By definition of the derivative, we have

$$f'(x_n) = \lim_{x \to x_n} \frac{f(x) - f(x_n)}{x - x_n}. \tag{3.34}$$

Replacing x by x_{n-1}, we obtain

$$f'(x_n) \approx \frac{f(x_{n-1}) - f(x_n)}{x_{n-1} - x_n}. \tag{3.35}$$

Using this approximation, the definition Eq. 3.31 of the approximating sequence is replaced by

$$x_{n+1} := x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}. \tag{3.36}$$

This defines the secant method. In geometrical terms, we have replaced the tangent used in Newton's method (Figure 3.3) by a secant (Figure 3.4).
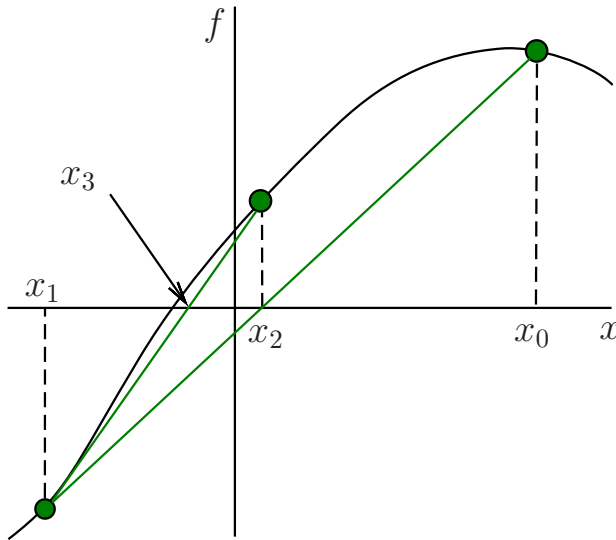
Figure 3.4: Secant method

Note that the secant method is *not* an example of fixed point iteration as discussed in Section 3.4, since the expression for x_{n+1} in equation Eq. 3.36 involves both x_n and x_{n-1}. For the same reason, the method requires *two* initial approximations, x_0 and x_1.

Example 3.5. Applying the secant method to the same example $f(x) = x^2 - 2$ with $x_0 = 1$ and $x_1 = 1.1$, we obtain the data in Table 3.7. We see that the convergence of the method is quite fast, although slower than the convergence of the Newton method applied to the same problem.

Table 3.7: Numerical example for secant method

n	x_n	$|x_n - \sqrt{2}|$
0	1.0	0.414213562
1	1.1	0.314213562
2	1.476190476	0.061976914
3	1.406654344	0.007559218
4	1.414051050	0.000162512
5	1.414213998	4.36×10^{-7}
6	1.414213562	0.0
7	1.414213562	0.0

Further reading: Section 2.3 of (Burden and Faires 2010).

3.7 Order of convergence

3.7.1 Definition

Definition 3.1. Suppose that $x_n \to p$ as $n \to \infty$ and $x_n \neq p$ for all n. The sequence $\{x_n\}$ is said to have **order of convergence** $\alpha \geq 1$ if there exists a constant $\lambda > 0$ such that

$$\lim_{n \to \infty} \frac{E_{n+1}}{E_n^\alpha} = \lambda. \tag{3.37}$$

Here E_n denotes the absolute error in the nth approximation: $E_n = |x_n - p|$.

If $\alpha = 1, 2, 3, ...$, the convergence is said to be *linear, quadratic, cubic, ...*, respectively. Note that if the convergence is linear, then the positive constant λ that appears in the above definition must be smaller than 1 ($0 < \lambda < 1$), because otherwise the sequence will not converge.

A sequence with a higher order of convergence converges much more rapidly than a sequence with a lower order of convergence. To see this, let us consider the following example:

Example 3.6. Let $\{x_n\}$ and $\{y_n\}$ be sequences converging to zero and let, for $n \geq 0$,

$$|x_{n+1}| = k|x_n| \quad \text{and} \quad |y_{n+1}| = k|y_n|^2, \tag{3.38}$$

where $0 < k < 1$. According to the definition, $\{x_n\}$ is linearly convergent and $\{y_n\}$ is quadratically convergent.

Also, we have

$$\begin{aligned}
|x_n| &= k|x_{n-1}| = k^2|x_{n-2}| = ... = k^n|x_0|, \\
|y_n| &= k|y_{n-1}|^2 = k|k|y_{n-2}|^2|^2 = k^3|y_{n-2}|^4 = k^7|y_{n-3}|^8 = ... = k^{2^n-1}|y_0|^{2^n}.
\end{aligned} \tag{3.39}$$

This illustrates that the quadratic convergence is much faster that the linear convergence.

3.7.2 Order of convergence for the Fixed Point Iteration

Suppose that $g(x)$ satisfies the conditions of the Fixed Point Theorem on interval $[a, b]$, so that the sequence $\{x_n\}$ generated by the formula $x_{n+1} = g(x_n)$ with $x_0 \in [a, b]$ converges to a fixed point p. Then, using the Mean Value Theorem, we obtain

$$\begin{aligned}
E_{n+1} = |x_{n+1} - p| &= |g(x_n) - g(p)| \\
&= |g'(\xi)(x_n - p)| = E_n|g'(\xi_n)|,
\end{aligned} \tag{3.40}$$

where ξ_n is a number between x_n and p. This implies that if $x_n \to p$, then $\xi_n \to p$ as $n \to \infty$. Therefore,

$$\lim_{n \to \infty} \frac{E_{n+1}}{E_n} = |g'(p)|. \tag{3.41}$$

In general, $g'(p) \neq 0$, so that the fixed point iteration produces a linearly convergent sequence.

Can the fixed-point iteration produce convergent sequences with convergence of order 2, 3, etc. ? It turns out that, under certain conditions, this is possible.

We will prove the following

Theorem 3.5. *Let $m > 1$ be an integer, and let $g \in C^m[a, b]$. Suppose that $p \in [a, b]$ is a fixed point of g, and a point $x_0 \in [a, b]$ exists such that the sequence generated by the formula $x_{n+1} = g(x_n)$ converges to p. If $g'(p) = \cdots = g^{(m-1)}(p) = 0$, then $\{x_n\}$ has the order of convergence m.*

Proof. Expanding $g(x_n)$ in Taylor's series at point p, we obtain:

$$
\begin{aligned}
x_{n+1} = g(x_n) = g(p) &+ (x_n - p)g'(p) + ... \\
&+ \frac{(x_n - p)^{m-1}}{(m-1)!}g^{(m-1)}(p) \\
&+ \frac{(x_n - p)^m}{m!}g^{(m)}(\xi_n) \\
= p &+ \frac{(x_n - p)^m}{(m)!}g^{(m)}(\xi_n),
\end{aligned}
\tag{3.42}
$$

where ξ_n is between x_n and p and, therefore, in $[a, b]$ ($x_n \in [a, b]$ at least for sufficiently large n). Then we have

$$
\begin{aligned}
E_{n+1} = |x_{n+1} - p| = |g(x_n) - p| &= \left| \frac{(x_n - p)^m}{(m)!}g^{(m)}(\xi_n) \right| \\
&= E_n^m \frac{|g^{(m)}(\xi_n)|}{m!}.
\end{aligned}
\tag{3.43}
$$

Therefore (using the fact that $\xi_n \to p$),

$$
\lim_{n \to \infty} \frac{E_{n+1}}{E_n^m} = \frac{|g^{(m)}(p)|}{m!},
\tag{3.44}
$$

which means that $\{x_n\}$ has convergence of order m. $\square$

3.7.3 Order of convergence of Newton's method

Newton's method for approximating the root p of the equation $f(x) = 0$ is equivalent to the fixed-point iteration $x_{n+1} = g(x_n)$ with

$$
g(x) = x - \frac{f(x)}{f'(x)}.
\tag{3.45}
$$

Suppose that sequence $\{x_n\}$ converges to p and $f'(p) \neq 0$. We have

$$
g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2} \quad \Rightarrow \quad g'(p) = \frac{f(p)f''(p)}{[f'(p)]^2} = 0.
\tag{3.46}
$$

It follows from the above theorem that the order of convergence of Newton's method is 2 (except in the special case where $g''(p) = 0$).

3.7.4 Order of convergence of the secant method

The situation with the secant method is more complicated (since it cannot be reduced to the fixed point iteration) and requires a separate treatment. The result is that the secant method has order of convergence $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Note that α is known as the *golden ratio*. If you are intrigued to see the golden ratio appear in this context, you can find a proof below. If you are happy to just accept the miracle, you can skip the proof and go on to the Exercises.

Suppose that a sequence $\{x_n\}$, generated by the secant method

$$
x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})},
\tag{3.47}
$$

converges to p. Let

$$
e_n = x_n - p,
\tag{3.48}
$$

so that $E_n = |e_n|$, and we assume that $E_n \ll 1$, which is definitely true for sufficiently large n (since the sequence $\{x_n\}$ is converging to p). Subtracting p from both sides of Eq. 3.47, we obtain

$$e_{n+1} = e_n - \frac{f(p+e_n)(e_n - e_{n-1})}{f(p+e_n) - f(p+e_{n-1})}, \tag{3.49}$$

Expanding $f(p+e_n)$ and $f(p+e_{n-1})$ in Taylor series about p and taking into account that $f(p) = 0$, we find that

$$\begin{aligned}
f(p+e_n) &= e_n f'(p) + \frac{e_n^2}{2} f''(p) + \cdots \\
&= e_n f'(p)(1 + e_n Q) + \cdots, \\
f(p+e_{n-1}) &= e_{n-1} f'(p) + \frac{e_{n-1}^2}{2} f''(p) + \cdots \\
&= e_{n-1} f'(p)(1 + e_{n-1} Q) + \cdots,
\end{aligned} \tag{3.50}$$

where

$$Q = \frac{f''(p)}{2f'(p)}. \tag{3.51}$$

Substitution of Eq. 3.50 into Eq. 3.49 yields

$$\begin{aligned}
e_{n+1} &= e_n - \frac{e_n(e_n - e_{n-1})f'(p)(1 + e_n Q) + \cdots}{f'(p)\left[e_n - e_{n-1} + Q(e_n^2 - e_{n-1}^2) + \cdots\right]} \\
&= e_n \left(1 - \frac{1 + e_n Q + \cdots}{1 + Q(e_n + e_{n-1}) + \cdots}\right).
\end{aligned} \tag{3.52}$$

Since, for small x,

$$\frac{1}{1 + x + \cdots} = 1 - x + \cdots, \tag{3.53}$$

we obtain

$$\begin{aligned}
e_{n+1} &= e_n \left(1 - (1 + e_n Q + \cdots)(1 - Q(e_n + e_{n-1}) + \cdots)\right) \\
&= Q e_n e_{n-1} + \cdots.
\end{aligned} \tag{3.54}$$

Thus, for sufficiently large n, we have

$$e_{n+1} \approx Q e_n e_{n-1}. \tag{3.55}$$

Hence,

$$E_{n+1} \approx |Q| E_n E_{n-1}. \tag{3.56}$$

Now we assume that (for all sufficiently large n)

$$E_{n+1} \approx \lambda E_n^\alpha, \tag{3.57}$$

where λ and α are positive constants. Substituting Eq. 3.57 into Eq. 3.56, we find

$$\lambda E_n^\alpha \approx |Q| E_n E_{n-1} \quad \text{or} \quad \lambda E_n^{\alpha-1} \approx |Q| E_{n-1}. \tag{3.58}$$

Applying Eq. 3.57 one more time (with n replaced by $n-1$), we obtain

$$\lambda \left(\lambda E_{n-1}^\alpha\right)^{\alpha-1} \approx |Q| E_{n-1} \tag{3.59}$$

or, equivalently,

$$\lambda^\alpha E_{n-1}^{\alpha(\alpha-1)} \approx |Q| E_{n-1}. \tag{3.60}$$

The last equation will be satisfied provided that

$$\lambda^\alpha = |Q|, \quad \alpha(\alpha - 1) = 1, \tag{3.61}$$

which requires that

$$\lambda = |Q|^{1/\alpha}, \quad \alpha = (1 + \sqrt{5})/2 \approx 1.62. \tag{3.62}$$

Thus, we have shown that if $\{x_n\}$ is a convergent sequence generated by the secant method, then

$$\lim_{n\to\infty} \frac{E_{n+1}}{E_n^\alpha} = |Q|^{1/\alpha}. \tag{3.63}$$

Thus, the secant method has *superlinear* convergence.

Further reading: Section 2.4 of (Burden and Faires 2010).

3.8 Exercises

3.8.1 Written exercises

Exercise 3.1. Consider the bisection method for finding the zero of

$$f(x) = \sqrt{x} - \cos x \tag{3.64}$$

on $x \in [0, 1]$

a) Using the starting values of $a = 0.5$ and $b = 1.0$, calculate the first 5 steps of the bisection method.

b) How many steps of the bisection method would be need to ensure that the root is accurate to 10 significant figures?

c) Show that the bisection method gives a sequence x_n with an error that converges linearly to zero. Does that imply that the bisection method converges linearly?

Exercise 3.2. Consider the fixed point iteration

$$x_{n+1} = \frac{x_n^3 + 3ax_n}{3x_n^2 + a}, \tag{3.65}$$

where $a > 0$ is given.

a) What does this fixed point iteration approximate?

b) Use the scheme to calculate $\sqrt{23}$ correct to 10 significant digits.

Exercise 3.3. Consider the problem of finding a numerical approximation to $\sqrt{3}$ using the fixed point iteration method with the function

$$g(x) = x + \lambda P(x)(x^2 - 3) \tag{3.66}$$

where $P(x)$ is a general polynomial of degree m.

For the case $P(x) = 1$ and $\lambda = -1/4$, $g(x)$ satisfies the fixed point theorem on the domain $[1, 2]$, i.e.

$$g : [1, 2] \to [1, 2] \tag{3.67}$$

with

$$|g'(x)| < k, 0 < k < 1, x \in [1, 2]. \tag{3.68}$$

a) For $P(x) = 1$ and $\lambda = -1/4$, show that the sequence $\{x_n\}$ defined by $x_{n+1} = g(x_n)$ converges linearly to $\sqrt{3}$.

b) Show that $g(x)$ also satisfies the fixed point theorem on the domain $[1, 2]$ for the case $P(x) = x(x^2 - 5)$ and $\lambda = +1/12$, and thus there exists a unique fixed point defined by $x_{n+1} = g(x_n)$.

c) Given a starting value of $x_0 = 1$, calculate the number n of iterations required to achieve an absolute error of $E_n = 10^{-8}$ for the cases (i) $P(x) = 1$, $\lambda = -1/4$ and (ii) $P(x) = x(x^2 - 5)$, $\lambda = +1/12$.

Exercise 3.4. Find the small root of

$$x^2 - 10^4 x + 2 = 0 \tag{3.69}$$

to at least 17 significant figures, writing out all steps of your calculation.

3.8.2 Programming exercises

Exercise 3.5. In many physics and engineering applications, the Bessel functions $J_n(x)$ are the radial solutions to the Laplacian operator in cylindrical coordinates, i.e. $\nabla^2 f(r, \theta, z) = 0$. Applying boundary conditions to the problem usually results in the need to compute the roots of $J_n(x)$. Here we will use the **Bisection Method** to identify roots of $J_n(x)$ using the following generalized formula

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{k!(n+k)!4^k}.$$ (3.70)

a) The c++ code `bessel_root.c` (or FORTRAN code `bessel_root.f90`) is set up to compute roots of the Bessel function $J_0(x)$. It takes input from the user for the two points a and b that bracket the root and the maximum error ϵ to compute the root to. Alter the code and compute the first three positive roots for $J_1(x)$ to an accuracy of $\epsilon = 1.0 \times 10^{-5}$.

b) You may have noticed when applying the code that unless you choose the initial points a and b to bracket at least one root, the method does not converge. Alter the code to use the **Secant Method*, which does not require the root to be bracketed. Code for the Secant method is provided by `secant.c` (`secant.f90`).

c) For mathematical thought - i.e. there is no "right answer". Instead of using the secant method, we could use Newton's method here. What is a potential downside to using Newtons method in this application?

Exercise 3.6. Consider the problem of finding the root to the function

$$f(x) = \exp(x) - 10x^2$$ (3.71)

a) The C++ code `newton_root.c` (or FORTRAN code `newton_root.f90`) is set up to compute roots of this function using **Newton's method** that we learned in lecture. It takes input from the user on an initial guess for the root a and the maximum error ϵ to compute the root to. Find three roots of the function.

b) Try using a starting value of $a = 3.5$. Why does the method converge to the negative root instead of the two positive roots which are closer?

c) Alter the code and compute the roots for the function $f(x) = x^3 - 1$ to an accuracy of $\epsilon = 1.0 \times 10^{-5}$. Does the code always converge to the same root?

d) **Challenge** Alter the code so that it can identify any real or complex root of the function $f(z) = z^3 - 1$.

Exercise 3.7. Consider the surface that is described by the cylindrical shape of length L and radius given by the general function $s(z)$

The surface area of this cylinder is given by:

$$A = 2\pi \int_0^L s(z)\sqrt{1 + s'(z)^2}dz.$$ (3.72)

We wish to identify the radial function $s(z)$ that minimizes the surface area of the cylinder.

a) Use the Euler-Lagrange equations to find the ODE for $s(z)$ that minimizes the area.

b) Verify that

$$s(z) = \alpha \cosh\left(\frac{z - \beta}{\alpha}\right)$$ (3.73)

is a solution to the ODE.
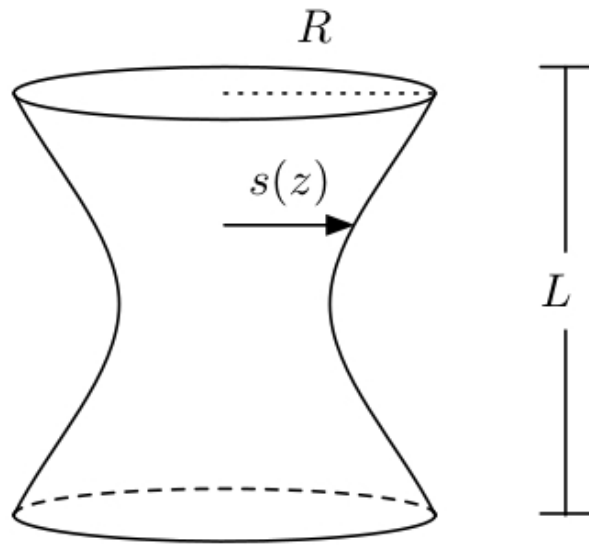
Figure 3.5: A cylinder of length L with surface generated by the radius function $s(z)$

c) Apply the boundary conditions $s(0) = s(L) = R$ to the solution in part (b). Use the two equations that you get from the boundary conditions to obtain a single equation in terms of R, L and a single unknown α.

d) Assuming R and L are known, choose an appropriate root finding technique (bisection,secant, or Newton) and create a root finding code that solves for the unknown α. Using $R = 5.0$ and $L = 6.0$, identify the solution (or solutions) α. Plot the resulting function $s(z)$ that minimizes the surface area of the cylinder.

4 Solving systems of nonlinear equations

So far, we considered the root of one equation in one variable x. In applications, one often meets systems of several equations in several variables.

4.1 Defining the problem

Let $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m$ be given. The root finding problem is that of finding the solution $\mathbf{x} \in \mathbb{R}^m$ such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad \text{or} \quad \begin{cases} f_1(x_1, \dots, x_m) = 0 \\ f_2(x_1, \dots, x_m) = 0 \\ \vdots \\ f_m(x_1, \dots, x_m) = 0 \end{cases} \tag{4.1}$$

This is a system of m (linear or nonlinear) equations for m unknowns $x_1, \dots, x_m$.

Example 4.1. Let $m = 2$ and

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_2 - x_1^2 \end{pmatrix}. \tag{4.2}$$

Then we have the system of two equations

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ x_2 - x_1^2 = 0 \end{cases}. \tag{4.3}$$

This system has a simple geometrical interpretation (see Figure 4.1).
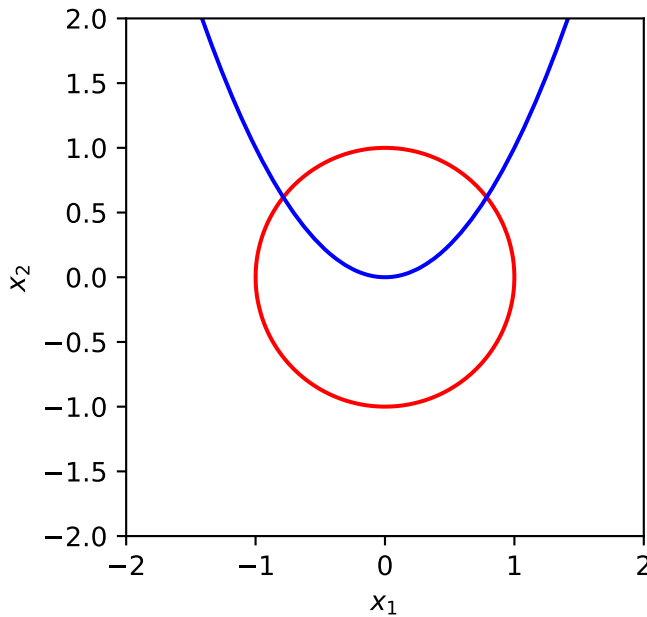
Figure 4.1: The two equations $x_1^2 + x_2^2 - 1 = 0$ and $x_2 - x_1^2 = 0$. The solutions of this system are the intersection points of the parabola with the circle.

Points (x_1, x_2) satisfying the first equation lie on the unit circle, while the second equation gives us the parabola. The solution of this system is represented by the two intersection points of the parabola with the circle in Figure 4.1.

As in the example above, the solutions of a system with m equations and m variables are typically (though not always) discrete points; and there can be more than one solution. In this simple example, one can find the intersection points of the curves explicitly (try it!), and therefore compute the solutions of the system. In general, however, this will not be possible, and we will need approximation methods.

4.2 Vector and matrix norms

Definition 4.1 (Norm of a vector). A function $\| \cdot \| : \mathbb{R}^m \to \mathbb{R}$ is a **vector norm** if it has the following properties:

(i)) $\|\mathbf{x}\| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^m$ and $\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = 0$ (i.e. $\mathbf{x} = (0, 0, ..., 0)^t$),

(ii) $\|\alpha \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^m$,

(iii) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ (the triangle inequality).

Example 4.2. The l_2 and l_∞ norms of the vector $\mathbf{x} = (x_1, x_2, ..., x_m)^t$ are defined by

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^m x_i^2 \right)^{1/2} \quad \text{and} \quad \|\mathbf{x}\|_\infty = \max_{1 \leq i \leq m} |x_i|. \tag{4.4}$$

The l_2 norm is called the Euclidean norm of the vector $\mathbf{x}$. It represents the notion of distance from the origin to the point $\mathbf{x}$ (the length of the straight line joining the points 0 and $\mathbf{x}$).

Definition 4.2. The **distance** between any two points $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$ is defined as the norm of the difference of the vectors: $\|\mathbf{x} - \mathbf{y}\|$.

Definition 4.3. A sequence $\{\mathbf{x}^{(k)}\}$ of vectors in $\mathbb{R}^m$ **converges** to $\mathbf{x}$ with respect to the norm $\| \cdot \|$ if for any $\epsilon > 0$, there exists an integer $N(\epsilon)$ such that

$$\|\mathbf{x}^{(k)} - \mathbf{x}\| < \epsilon \quad \text{for all} \quad k > N(\epsilon). \tag{4.5}$$

In other words, $\mathbf{x}^{(k)} \to \mathbf{x}$ as $k \to \infty$ if $\|\mathbf{x}^{(k)} - \mathbf{x}\| \to 0$ as $k \to \infty$.

Remark. It can be shown that all norms on $\mathbb{R}^m$ are **equivalent** with respect to convergence. This means that if $\| \cdot \|$ and $\| \cdot \|'$ are two norms on $\mathbb{R}^m$ and $\{\mathbf{x}^{(k)}\}$ converges to $\mathbf{x}$ with respect to the norm $\| \cdot \|$, then $\{\mathbf{x}^{(k)}\}$ also converges to $\mathbf{x}$ with respect to the norm $\| \cdot \|'$. (Proof of this fact for the case of l_2 and l_∞ norms follows from the inequality $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{m}\|\mathbf{x}\|_\infty$ which is valid for all $\mathbf{x} \in \mathbb{R}^m$.)

Let $M(m, m)$ be the space of all $m \times m$ real matrices.

Definition 4.4 (Norm of a matrix). A function $\| \cdot \| : M(m, m) \to \mathbb{R}$ is a **matrix norm** if it has the following properties:

(i) $\|A\| \geq 0$ and $\|A\| = 0 \Leftrightarrow A = O$ (where O is the matrix with all zero entries),

(ii) $\|\alpha A\| = |\alpha| \cdot \|A\|$,

(iii) $\|A + B\| \leq \|A\| + \|B\|$ (the triangle inequality),

(iv) $\|AB\| \leq \|A\| \cdot \|B\|$.

where A and B are any $m \times m$ real matrices, and α is any real number.

A **distance** between matrices A and B (with respect to this matrix norm) is $\|A - B\|$.

Definition 4.5 (Natural, or induced, matrix norm)**.** If $\| \cdot \|$ is a vector norm on $\mathbb{R}^m$, then

$$\|A\| = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\| \tag{4.6}$$

is a matrix norm (called *natural, or induced, matrix norm* associated with the vector norm).

In what follows we shall consider only l_∞ matrix norms:

$$\|A\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \|A\mathbf{x}\|_\infty. \tag{4.7}$$

Theorem 4.1. *For any $m \times m$ matrix $A = (a_{ij})$,*

$$\|A\|_\infty = \max_{1 \le i \le m} \left\{ \sum_{j=1}^{m} |a_{ij}| \right\}. \tag{4.8}$$

Example 4.3. Let

$$A = \begin{bmatrix} 1 & 2 & 0 & -1 \\ 1 & 3 & 2 & 0 \\ 0 & -2 & 3 & 2 \\ 1 & 2 & 2 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 2 & 2 & -3 \\ 1 & 3 & 2 \\ -3 & -2 & 2 \end{bmatrix}. \tag{4.9}$$

Then, according to Eq. 4.8, Evidently, $\|A\|_\infty = \max\{4, 6, 7, 6\} = 7$ and $\|B\|_\infty = \max\{7, 6, 7\} = 7$.

Later we will need the following important property of a natural matrix norm.

Theorem 4.2. *For any $\mathbf{x} \neq 0$, any matrix A, and any natural norm $\| \cdot \|$,*

$$\|A\mathbf{x}\| \le \|A\| \cdot \|\mathbf{x}\|. \tag{4.10}$$

Proof. The proof is very easy:

$$\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \left\| A \frac{\mathbf{x}}{\|\mathbf{x}\|} \right\| \le \max_{\|\mathbf{y}\|=1} \|A\mathbf{y}\| = \|A\| \quad \Rightarrow \quad \|A\mathbf{x}\| \le \|A\| \cdot \|\mathbf{x}\|. \tag{4.11}$$

Here we have used the relevant properties of the vector norm and the definition of the natural matrix norm. $\qquad \square$

Remark. For any natural norm, the above theorem implies property (iv) in the definition of the matrix norm. Indeed, for any $\mathbf{x} \neq 0$ we have

$$\|AB\mathbf{x}\| \le \|A\| \|B\mathbf{x}\| \le \|A\| \|B\| \|\mathbf{x}\| \tag{4.12}$$

and

$$\left\| AB \frac{\mathbf{x}}{\|\mathbf{x}\|} \right\| \le \|A\| \|B\|. \tag{4.13}$$

Since the last inequality is valid for all $\mathbf{x} \neq 0$, it implies that

$$\|AB\| = \max_{\|\mathbf{y}\|=1} \|AB\mathbf{y}\| \le \|A\| \|B\|. \tag{4.14}$$

4.3 Fixed point iteration for vectors

We already know that, in the case of functions of one variable, the root finding problem $f(x) = 0$ can be reformulated as a fixed point problem $g(x) = x$. The same is true for systems of nonlinear equations: $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ can be rewritten as $\mathbf{g}(\mathbf{x}) = \mathbf{x}$ in infinitely many ways, for example with $\mathbf{g}(\mathbf{x}) = \mathbf{x} - \lambda \mathbf{f}(\mathbf{x})$ for any nonzero constant λ.

Now we consider the fixed point problem for vectors. Given a function $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^m$, the problem is to find a point $\mathbf{x} \in \mathbb{R}^m$ such that

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} \quad \text{or, equivalently,} \quad \begin{cases} g_1(x_1, \dots, x_m) = x_1 \\ g_2(x_1, \dots, x_m) = x_2 \\ \vdots \\ g_m(x_1, \dots, x_m) = x_m \end{cases} \tag{4.15}$$

We choose an initial approximation, a vector $\mathbf{x}^{(0)}$, and compute the sequence of vectors $\{\mathbf{x}^{(n)}\}$ using the formula

$$\mathbf{x}^{(n+1)} = \mathbf{g}(\mathbf{x}^{(n)}) \quad (n = 0, 1, \dots). \tag{4.16}$$

To investigate the convergence of the fixed point iteration, we need to recall some results from Vector Calculus.

Taylor's series expansion for functions of several variables. We assume that $f : D \to \mathbb{R}$ (where $D \subset \mathbb{R}^m$ is an open and convex region of $\mathbb{R}^m$) is $(n+1)$ times continuously differentiable in D, i.e. $f \in C^{n+1}(D)$. The Taylor's series expansion for f can be obtained from the expansion for a function of one variable.

Let $\mathbf{x}$ and $\mathbf{x}_0$ be two points in D and let $\mathbf{v} = \mathbf{x} - \mathbf{x}_0$, then the function $\phi : [0, 1] \to \mathbb{R}$ defined by

$$\phi(t) = f(\mathbf{x}_0 + t\mathbf{v}) \tag{4.17}$$

is a function of one variable which is $(n+1)$ times continuously differentiable on the interval $[0, 1]$. Note also that $f(\mathbf{x}_0) = \phi(0)$ and $f(\mathbf{x}) = \phi(1)$. Applying Taylor's theorem for functions of one variable to $\phi(t)$, we obtain

$$\phi(t) = \phi(0) + t\phi'(0) + \frac{t^2}{2}\phi''(0) + \dots + \frac{t^n}{n!}\phi^{(n)}(0) + R_n(t) \tag{4.18}$$

where the remainder term $R_n(t)$ can be written in various forms, e.g. in the Lagrange form

$$R_n(t) = \frac{t^{n+1}}{(n+1)!}\phi^{(n+1)}(\xi) \quad \text{for some} \quad \xi \in (0, 1), \tag{4.19}$$

or in the integral form

$$R_n(t) = \frac{1}{n!} \int_0^t \phi^{(n+1)}(s)(t-s)^n \, ds. \tag{4.20}$$

We will see that this integral form of the remainder will be more convenient for us.

The chain rule yields

$$f(\mathbf{x}_0 + t\mathbf{v}) = f(\mathbf{x}_0) + t(\mathbf{v} \cdot \nabla)f(\mathbf{x}_0) + \frac{t^2}{2}(\mathbf{v} \cdot \nabla)^2 f(\mathbf{x}_0) + \dots$$
$$+ \frac{t^n}{n!}(\mathbf{v} \cdot \nabla)^n f(\mathbf{x}_0) + R_n(t) \tag{4.21}$$

{#eq-taylor-multivariable}

where $\mathbf{v} \cdot \nabla = v_1 \partial_{x_1} + v_2 \partial_{x_2} + \cdots + v_m \partial_{x_m}$. Now we set $t = 1$, so that

$$f(\mathbf{x}) = f(\mathbf{x}_0) + (\mathbf{v} \cdot \nabla)f(\mathbf{x}_0) + \frac{1}{2}(\mathbf{v} \cdot \nabla)^2 f(\mathbf{x}_0) + \ldots$$

$$+ \frac{1}{n!}(\mathbf{v} \cdot \nabla)^n f(\mathbf{x}_0) + R_n(1) \quad (4.22)$$

{#eq-taylor-multivariable-final} with the remainder term

$$R_n(1) = \frac{1}{n!} \int\limits_0^1 \phi^{(n+1)}(s)(1-s)^n \, ds. \quad (4.23)$$

In particular, for $n = 0$ we get

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \int\limits_0^1 \sum_{i=1}^m (x_i - x_{0i}) \partial_{x_i} f(\mathbf{x}_0 + s\mathbf{v}) ds \quad (4.24)$$

and for $n = 1$ we get

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \sum_{i=1}^m (x_i - x_{0i}) \partial_{x_i} f(\mathbf{x}_0)$$

$$+ \int\limits_0^1 \sum_{i,j=1}^m (x_i - x_{0i})(x_j - x_{0j}) \partial_{x_i} \partial_{x_j} f(\mathbf{x}_0 + s\mathbf{v})(1-s) ds. \quad (4.25)$$

Now let $\mathbf{g} : D \to \mathbb{R}^m$ and $\mathbf{g} \in C^1(D)$ (i.e. $\mathbf{g}$ is continuously differentiable in D). Applying Eq. 4.24 to each component of $\mathbf{g}$, we obtain the formula

$$g_i(\mathbf{x}) = g_i(\mathbf{y}) + \int\limits_0^1 \sum_{k=1}^m (x_k - y_k) \partial_{x_k} g_i(\mathbf{y} + s(\mathbf{x} - \mathbf{y})) ds, \quad (4.26)$$

which is valid for any $\mathbf{x}, \mathbf{y} \in D$. If we introduce the Jacobian matrix

$$\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \quad (4.27)$$

whose (i,j) component is $\partial g_i / \partial x_j$, the above formula can be written in a nicer form:

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{y}) + \int\limits_0^1 \mathbf{J}(\mathbf{y} + s(\mathbf{x} - \mathbf{y}))(\mathbf{x} - \mathbf{y}) ds. \quad (4.28)$$

It can be shown (see for example Ortega 1972, chap. 8) that for any continuous function $\mathbf{G} : [a, b] \to \mathbb{R}^m$ and any vector norm,

$$\left\| \int\limits_a^b \mathbf{G}(s) ds \right\| \leq \int\limits_a^b \|\mathbf{G}(s)\| \, ds. \quad (4.29)$$

Now we are ready to (partially) prove the following theorem.

Theorem 4.3. *Let $\mathbf{g} : D \to D$ be continuously differentiable, where $D \subset \mathbb{R}^m$ is a closed and convex region. If there exists $0 < k < 1$ such that*

$$\left\| \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \right\| \leq k \quad \text{for all } \mathbf{x} \in D, \quad (4.30)$$

then $\mathbf{g}$ has a unique fixed point $\mathbf{p}$ in D and the sequence $\{\mathbf{x}^{(n)}\}$ generated be the formula

$$\mathbf{x}^{(n+1)} = \mathbf{g}(\mathbf{x}^{(n)}) \quad (n = 0, 1, \ldots) \quad (4.31)$$

converges to $\mathbf{p}$ for any $\mathbf{x}^{(0)} \in D$, and the following estimate holds:

$$\|\mathbf{x}^{(n)} - \mathbf{p}\| \leq k^n \|\mathbf{x}^{(0)} - \mathbf{p}\|. \quad (4.32)$$

Proof. **(partial)** We will only partially prove this theorem. Namely, we will prove that $\mathbf{g}$ is a contraction mapping. Then the *contraction mapping theorem* (see, e.g. Wait 1979, chap. 5) will guarantee the existence and uniqueness of the fixed point, as well as convergence of the fixed point iteration. We will also prove the above error estimate.

To show that $\mathbf{g}$ is a contraction mapping (i.e there is a number $k \in (0,1)$ such that $\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| \leq k \|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in D$), we will use Eq. 4.28. It follows from Eq. 4.28 that

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| = \left\| \int_0^1 \mathbf{J}(\mathbf{y} + s(\mathbf{x} - \mathbf{y}))(\mathbf{x} - \mathbf{y}) ds \right\|. \tag{4.33}$$

Now, with the help of Eq. 4.28, we obtain

$$
\begin{aligned}
\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| &\leq \int_0^1 \|\mathbf{J}(\mathbf{y} + s(\mathbf{x} - \mathbf{y}))(\mathbf{x} - \mathbf{y})\| \, ds \\
&\leq \int_0^1 \|\mathbf{J}(\mathbf{y} + s(\mathbf{x} - \mathbf{y}))\| \, \|\mathbf{x} - \mathbf{y}\| ds \\
&\leq \sup_{\mathbf{x} \in D} \|\mathbf{J}(\mathbf{x})\| \int_0^1 \|\mathbf{x} - \mathbf{y}\| ds = \sup_{\mathbf{x} \in D} \|\mathbf{J}(\mathbf{x})\| \, \|\mathbf{x} - \mathbf{y}\| \\
&\leq k \|\mathbf{x} - \mathbf{y}\|.
\end{aligned}
\tag{4.34}
$$

Here we have used the property of the natural matrix norm that we have proved earlier: $\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|$. The above proves the fact that $\mathbf{g}$ is a contraction mapping.

The proof of the error estimate is exactly the same as in the case of functions of one variable: if $\mathbf{p}$ is a fixed point, then

$$
\begin{aligned}
\|\mathbf{x}^{(n)} - \mathbf{p}\| &= \|\mathbf{g}(\mathbf{x}^{(n-1)}) - \mathbf{g}(\mathbf{p})\| \\
&\leq k\|\mathbf{x}^{(n-1)} - \mathbf{p}\| = k\|\mathbf{g}(\mathbf{x}^{(n-2)}) - \mathbf{g}(\mathbf{p})\| \\
&\leq k^2\|\mathbf{x}^{(n-2)} - \mathbf{p}\| = k\|\mathbf{g}(\mathbf{x}^{(n-3)}) - \mathbf{g}(\mathbf{p})\| \\
&\vdots \\
&\leq k^n\|\mathbf{x}^{(0)} - \mathbf{p}\|.
\end{aligned}
\tag{4.35}
$$

$\square$

Example 4.4. Let us consider the function $\mathbf{g}$ defined on $D = [0,1] \times [0,1] \subset \mathbb{R}^2$ by

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} \cos x_2 \\ \frac{3}{4} \sin x_1 \end{pmatrix}. \tag{4.36}$$

This is well-defined: For any $\mathbf{x} \in [0,1] \times [0,1]$, we know that $\mathbf{g}(\mathbf{x}) \in [0,1] \times [0,1]$ due to the well-known bounds on the trigonometric functions. We can compute the Jacobian matrix of $\mathbf{g}$:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \begin{pmatrix} 0 & -\sin x_2 \\ \frac{3}{4} \cos x_1 & 0 \end{pmatrix} \tag{4.37}$$

Therefore,

$$\left\| \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \right\| \leq \max \left\{ \sin(1), \frac{3}{4} \right\} \leq 0.85, \tag{4.38}$$

so that the conditions of Theorem 4.3 are fulfilled with $k = 0.85$. The fixed point iteration will therefore converge for any choice of $\mathbf{x}^{(0)} \in [0,1] \times [0,1]$, although possibly not very fast, since 0.85 is not very much smaller than 1. See Table 4.2 for two examples of iteration sequences.

n	$x_1^{(n)}$	$x_2^{(n)}$
0	1.0	1.0
1	0.5403023059	0.6311032386
2	0.8073770495	0.3857964440
3	0.9264990269	0.5418571235
4	0.8567523888	0.5996415238
5	0.8255379728	0.5667897802
6	0.8436289716	0.5511845408
7	0.8519047798	0.5602953112
8	0.8470982088	0.5644021231
9	0.8449085622	0.5620215836
10	0.8461795432	0.5609328142

Table 4.2: Numerical example for fixed point iteration

n	$x_1^{(n)}$	$x_2^{(n)}$
0	0.8	0.6
1	0.8253356149	0.5380170682
2	0.8587264910	0.5510816060
3	0.8519586819	0.5677582977
4	0.8431085532	0.5644287452
5	0.8448943215	0.5600357720
6	0.8472361089	0.5609257244
7	0.8467630160	0.5620900623
8	0.8461430490	0.5618550885
9	0.8462682563	0.5615469756
10	0.8464323655	0.5616092188

4.4 Newton's method for systems of equations

A generalisation of Newton's method to systems of non-linear equations can be done in a straightforward manner.

Consider a given function $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m$. We are looking for the solution, $\mathbf{x} \in \mathbb{R}^m$, of the system of equations:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad \text{or} \quad \begin{cases} f_1(x_1, ..., x_m) = 0 \\ f_2(x_1, ..., x_m) = 0 \\ \vdots \\ f_m(x_1, ..., x_m) = 0 \end{cases} \tag{4.39}$$

The idea of Newton's method for systems of equations is the same as for one equation. First we expand each component of $\mathbf{f}$ in Taylor series

$$f_i(\mathbf{x}) = f_i(\mathbf{x}_0) + \sum_j \frac{\partial f_i(\mathbf{x}_0)}{\partial x_j}(x_j - x_{0j}) + ... \quad (i = 1, ..., m). \tag{4.40}$$

This can be written as

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + J(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + ... \tag{4.41}$$

where $J(\mathbf{x}_0)$ is the Jacobian matrix for function $\mathbf{f}$ at point $\mathbf{x}_0$.

Let $\mathbf{p} \in \mathbb{R}^m$ be a solution of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ and $\mathbf{x}^*$ be a sufficiently good approximation to $\mathbf{p}$, so that $\|\mathbf{p} - \mathbf{x}^*\|$ is small. Then we have

$$\mathbf{0} = \mathbf{f}(\mathbf{p}) \approx \mathbf{f}(\mathbf{x}^*) + J(\mathbf{x}^*)(\mathbf{p} - \mathbf{x}^*). \tag{4.42}$$

If the matrix $J(\mathbf{x}^*)$ is invertible, this can be rewritten as

$$\mathbf{p} \approx \mathbf{x}^* - J^{-1}(\mathbf{x}^*)\mathbf{f}(\mathbf{x}^*). \tag{4.43}$$

This formula will not give us the exact solution $\mathbf{p}$ because we have ignored higher order terms in the Taylor expansion. Nevertheless, it is natural to expect that it will give us a better approximation than $\mathbf{x}^*$.

So, we will consider the following iterative process:

- We choose an initial approximation $\mathbf{x}^{(0)}$.

- We compute the sequence of approximations, $\{\mathbf{x}^{(0)}\}$, using the formula

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - J^{-1}(\mathbf{x}^{(n)})\mathbf{f}(\mathbf{x}^{(n)}). \tag{4.44}$$

This is *Newton's method for systems of equations.*

Remark. If the number of equations is large (i.e. m is large), it is computationally very expensive to find $J^{-1}(\mathbf{x}^{(n)})$. Therefore, in practice, Eq. 4.44 is replaced by the following equivalent procedure: first we solve the system of linear equations

$$J(\mathbf{x}^{(n)})\mathbf{v}^{(n)} = -\mathbf{f}(\mathbf{x}^{(n)}) \tag{4.45}$$

for $\mathbf{v}^{(n)}$, and then we compute $\mathbf{x}^{(n+1)}$:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \mathbf{v}^{(n)}. \tag{4.46}$$

Newton's method for systems of equations produces quadratically converging sequences, as follows from the following theorem:

Theorem 4.4. *Let $D \subset \mathbb{R}^m$ be open and convex, and let $\mathbf{f} \in C^2(D)$. If $\mathbf{p} \in D$ is such that $\mathbf{f}(\mathbf{p}) = 0$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{p})$ is invertible, then there exists $\delta > 0$ such that the sequence $\{\mathbf{x}^{(n)}\}$ defined in Eq. 4.44 converges to $\mathbf{p}$ for any initial approximation $\mathbf{x}^{(0)}$ with $\|\mathbf{p} - \mathbf{x}^{(0)}\| < \delta$. Moreover, if $\mathbf{f} \in C^3(D)$, then there exists $K > 0$ such that*

$$\|\mathbf{x}^{(n)} - \mathbf{p}\| \leq K\|\mathbf{x}^{(n-1)} - \mathbf{p}\|^2 \quad for \ all \quad n \geq 1. \tag{4.47}$$

We will not prove this theorem here. A proof of a more general theorem can be found in Kelley (1995).

The estimate Eq. 4.47 guarantees that convergence is very fast if $\mathbf{x}^{(0)}$ is chosen close enough to the solution $\mathbf{p}$. In practical applications, as few as 4 or 5 steps of Newton iteration are often sufficient to reach the desired accuracy.

Example 4.5. Consider the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ with

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_2 - x_1^2 \end{pmatrix}. \tag{4.48}$$

This system has two solutions corresponding to the two intersection points of the unit circle and the parabola shown in Figure 4.1. These two solutions can be computed analytically:

$$\mathbf{p}_1 = \begin{pmatrix} \sqrt{(\sqrt{5}-1)/2} \\ (\sqrt{5}-1)/2 \end{pmatrix} \approx \begin{pmatrix} 0.7861513773 \\ 0.618033988 \end{pmatrix} \tag{4.49}$$

and

$$\mathbf{p}_2 = \begin{pmatrix} -\sqrt{(\sqrt{5}-1)/2} \\ (\sqrt{5}-1)/2 \end{pmatrix} \approx \begin{pmatrix} -0.7861513773 \\ 0.618033988 \end{pmatrix}. \tag{4.50}$$

The Jacobian of $\mathbf{f}$ is

$$J(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{pmatrix} 2x_1 & 2x_2 \\ -2x_1 & 1 \end{pmatrix}, \tag{4.51}$$

so $\mathbf{f}$ is continuously differentiable. It is also straightforward to check that the second and third order partial derivatives of $\mathbf{f}$ are continuous (do this!). The inverse of the Jacobian is

$$J^{-1}(\mathbf{x}) = \begin{pmatrix} \dfrac{1}{2x_1(2x_2+1)} & -\dfrac{x_2}{x_1(2x_2+1)} \\ \dfrac{1}{2x_2+1} & \dfrac{1}{2x_2+1} \end{pmatrix}. \tag{4.52}$$

This matrix is not well-defined for all $\mathbf{x}$ (it has singularities for $x_1 = 0$ and any x_2 and for $x_2 = -1/2$ and any x_1), but it is defined at the roots we seek (check this!), so Newton's method is convergent by Theorem 4.4, provided that we choose initial values close enough to those roots.

We choose $\mathbf{x}^{(0)} = (0.5, 0.5)^t$ and $\mathbf{x}^{(0)} = (-0.5, 0.5)^t$ as the initial approximations for the two solutions, and then compute sequences of approximations using Eq. 4.45 and Eq. 4.46. Table 4.3 demonstrates the rapid convergence of Newton's method to the two solutions.

Table 4.3: Numerical example for Newton's method

n	$x_1^{(n)}$	$x_2^{(n)}$	$\|\mathbf{x}^{(n)} - \mathbf{p}_1\|$
0	0.5	0.5	
1	0.87500000	0.62500000	0.08884862
2	0.79067460	0.61805556	0.00452323
3	0.78616432	0.61803399	0.00001294
4	0.78615138	0.61803399	$7.499 \cdot 10^{-10}$

n	$x_1^{(n)}$	$x_2^{(n)}$	$\|\mathbf{x}^{(n)} - \mathbf{p}_2\|$
0	-0.5	0.5	
1	-0.87500000	0.62500000	0.08884862
2	-0.79067460	0.61805556	0.00452323
3	-0.78616432	0.61803399	0.00001294
4	-0.78615138	0.61803399	$7.499 \cdot 10^{-10}$

Further reading: Section 10.2 of (Burden and Faires 2010).

5 Iterative techniques for solving systems of linear equations

Consider the system of linear equations

$$A\mathbf{x} = \mathbf{b} \tag{5.1}$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{5.2}$$

for a vector $\mathbf{x}$ of unknowns. An iterative method for a linear system starts with an initial approximation $\mathbf{x}^{(0)}$ and generates a sequence of vectors $\{\mathbf{x}^{(k)}\}$ that converges to the solution $\mathbf{x}$. In constructing an iterative method, we first convert the system $A\mathbf{x} = \mathbf{b}$ to the form

$$\mathbf{x} = T\mathbf{x} + \mathbf{c} \tag{5.3}$$

where T is a fixed matrix and $\mathbf{c}$ is a fixed vector. Then, we compute a sequence of approximations using the formula

$$\mathbf{x}^{(k+1)} = T\mathbf{x}^{(k)} + \mathbf{c} \quad \text{for} \quad k = 0, 1, 2, ... \tag{5.4}$$

Note that this can be viewed as the fixed point iteration for function $\mathbf{g}(\mathbf{x}) = T\mathbf{x}^{(k)} + \mathbf{c}$.

Let us say a few words about the convergence of sequences of vectors generated by formula Eq. 5.4 (a more detailed analysis for concrete methods will be given later). Subtracting Eq. 5.3 from Eq. 5.4, we obtain

$$\mathbf{x}^{(k+1)} - \mathbf{x} = T(\mathbf{x}^{(k)} - \mathbf{x}) \tag{5.5}$$

from which it follows that

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}\| = \|T(\mathbf{x}^{(k)} - \mathbf{x})\| \le \|T\|, \|\mathbf{x}^{(k)} - \mathbf{x}\|, \tag{5.6}$$

where we have used the property of the natural matrix norm, Theorem 4.2. It follows from Eq. 5.6 and the contraction mapping theorem that the sequence will converge linearly provided that $\|T\| \| < 1$.

5.1 The Jacobi method

Example 5.1. Consider the system

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 7, \\ x_1 + 8x_2 + 3x_3 &= -4, \\ -2x_1 - x_2 + 10x_3 &= 9, \end{aligned} \tag{5.7}$$

Its unique solution is $\mathbf{x} = (1, -1, 1)^t$. First, we convert the system to the form $\mathbf{x} = T\mathbf{x} + \mathbf{c}$ by solving the first equation for x_1, the second for x_2 and the third for x_3:

$$\begin{aligned} x_1 &= -\frac{2}{10}x_2 + \frac{1}{10}x_3 + \frac{7}{10}, \\ x_2 &= -\frac{1}{8}x_1 - \frac{3}{8}x_3 - \frac{1}{2}, \\ x_3 &= \frac{2}{10}x_1 + \frac{1}{10}x_2 + \frac{9}{10}, \end{aligned} \tag{5.8}$$

Thus, $\mathbf{x} = T\mathbf{x} + \mathbf{c}$ with

$$T = \begin{pmatrix} 0 & -\frac{2}{10} & \frac{1}{10} \\ -\frac{1}{8} & 0 & -\frac{3}{8} \\ \frac{2}{10} & \frac{1}{10} & 0 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} \frac{7}{10} \\ -\frac{1}{2} \\ \frac{9}{10} \end{pmatrix}. \tag{5.9}$$

Let the initial approximation be $\mathbf{x}^{(0)} = (0,0,0)^t$. Then,

$$\begin{aligned} x_1^{(1)} &= -\frac{2}{10}x_2^{(0)} + \frac{1}{10}x_3^{(0)} + \frac{7}{10} = \frac{7}{10}, \\ x_2^{(1)} &= -\frac{1}{8}x_1^{(0)} - \frac{3}{8}x_3^{(0)} - \frac{1}{2} = -\frac{1}{2}, \\ x_3^{(1)} &= \frac{2}{10}x_1^{(0)} + \frac{1}{10}x_2^{(0)} + \frac{9}{10} = \frac{9}{10}. \end{aligned} \tag{5.10}$$

and

$$\begin{aligned} x_1^{(2)} &= -\frac{2}{10}x_2^{(1)} + \frac{1}{10}x_3^{(1)} + \frac{7}{10} = 0.89, \\ x_2^{(2)} &= -\frac{1}{8}x_1^{(1)} - \frac{3}{8}x_3^{(1)} - \frac{1}{2} = -0.925, \\ x_3^{(2)} &= \frac{2}{10}x_1^{(1)} + \frac{1}{10}x_2^{(1)} + \frac{9}{10} = 0.99. \end{aligned} \tag{5.11}$$

One can see that just two iterations give a fairly good approximation to the solution $\mathbf{x} = (1, -1, 1)^t$. Further calculations yield:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0	0.0	0.0
1	0.7	-0.5	0.9
2	0.89	-0.925	0.99
3	0.984	-0.9825	0.9855
4	0.99505	-0.9925625	0.99855
5	0.9983675	-0.9988375	0.99975375
6	0.999742875	-0.9997035938	0.9997897500

The method we have used is called the *Jacobi iterative method.* In general, it consists of solving the ith equation of the system $A\mathbf{x} = \mathbf{b}$ for x_i to obtain

$$x_i = \sum_{j=1, j\neq i}^{n} \left(-\frac{a_{ij}x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}} \quad \text{for} \quad i = 1, 2, ..., n \tag{5.12}$$

and calculating $\mathbf{x}^{(k+1)}$ from $\mathbf{x}^{(k)}$ for $k \geq 0$ by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(\sum_{j=1, j\neq i}^{n} \left(-a_{ij}x_j^{(k)} \right) + b_i \right) \quad \text{for} \quad i = 1, 2, ..., n. \tag{5.13}$$

Let us write the Jacobi method in matrix form $\mathbf{x}^{(k+1)} = T\mathbf{x}^{(k)} + \mathbf{c}$. To do this, we decompose the matrix A in the form $A = D + L + U$, where D is the diagonal part of A, L is the strictly lower-triangular part of A, and U is the strictly upper-triangular part of A:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & ... & 0 & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & ... & ... & 0 \\ a_{12} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n2} & ... & a_{n,n-1} & 0 \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & ... & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & ... & ... & 0 \end{pmatrix} \tag{5.14}$$

Now we have

$$(D + L + U)\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad D\mathbf{x} = -(L + U)\mathbf{x} + \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x} = -D^{-1}(L + U)^{\mathbf{x}} + D{-}1\mathbf{b}. \tag{5.15}$$

Thus, we have converted the initial system $A\mathbf{x} = \mathbf{b}$ to the form $\mathbf{x} = T_J\mathbf{x} + \mathbf{c}_J$ with $T_J = -D^{-1}(L+U)$ and $\mathbf{c}_J = D^{-1}\mathbf{b}$, so that the Jacobi iterative method has the form

$$\mathbf{x}^{(k+1)} = T_J^{\mathbf{x}}(k) + \mathbf{c}_J. \tag{5.16}$$

The Jacobi method requires that $a_{ii} \neq 0$ for each $i = 1, 2, ..., n$. Therefore, if one of the elements a_{ii} is zero, we need to reorder the equation so that $a_{ii} \neq 0$ for each $i = 1, 2, ..., n$.

5.2 The Gauss-Seidel method

How to improve the Jacobi method? In the Example 5.1, we used the iteration procedure

$$
\begin{aligned}
x_1^{(k+1)} &= -\frac{2}{10}x_2^{(k)} + \frac{1}{10}x_3^{(k)} + \frac{7}{10}, \\
x_2^{(k+1)} &= -\frac{1}{8}x_1^{(k)} - \frac{3}{8}x_3^{(k)} - \frac{1}{2}, \\
x_3^{(k+1)} &= \frac{2}{10}x_1^{(k)} + \frac{1}{10}x_2^{(k)} + \frac{9}{10}.
\end{aligned}
\tag{5.17}
$$

When we calculate $x_2^{(k+1)}$, we use $x_1^{(k)}$ and $x_3^{(k)}$, which have been calculated at the previous step. We may notice however that this stage we already know $x_1^{(k+1)}$ which is assumed to be a better approximation to x_1. It is natural therefore to replace $x_1^{(k)}$ in the second equation by $x_1^{(k+1)}$. Similarly, we may replace $x_1^{(k)}$ and $x_2^{(k)}$ in the third equation by $x_1^{(k+1)}$ and $x_2^{(k+1)}$. This yields the formula

$$
\begin{aligned}
x_1^{(k+1)} &= -\frac{2}{10}x_2^{(k)} + \frac{1}{10}x_3^{(k)} + \frac{7}{10}, \\
x_2^{(k+1)} &= -\frac{1}{8}x_1^{(k+1)} - \frac{3}{8}x_3^{(k)} - \frac{1}{2}, \\
x_3^{(k+1)} &= \frac{2}{10}x_1^{(k+1)} + \frac{1}{10}x_2^{(k+1)} + \frac{9}{10}.
\end{aligned}
\tag{5.18}
$$

Eq. 5.18 with the initial approximation $\mathbf{x}^{(0)} = (0,0,0)^t$ generate the sequence

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0	0.0	0.0
1	0.7	-0.5875	0.9812500000
2	0.915625	-0.9824218750	0.9848828125
3	0.9949726562	-0.9937026368	0.9996242675
4	0.9987029542	-0.9996969695	0.9997708938
5	0.9999164833	-0.9999036455	0.9999929322
6	0.9999800223	-0.9999948524	0.9999965193

This modification of the Jacobi technique is called the *Gauss-Seidel iterative method*. Comparing $\mathbf{x}^{(6)}$ obtained using the Jacobi and Gauss-Seidel methods, we see that the Gauss-Seidel method produces a better approximation to the exact solution $\mathbf{x} = (1, -1, 1)^t$.

The general formula for the Gauss-Seidel method is (cf. Eq. 5.13)

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i \right) \tag{5.19}$$

for $i = 1, ..., n$. Let us rewrite the Gauss-Seidel method in matrix form. To do this, we multiply both sides of Eq. 5.19 by a_{ii} and collect all terms of the kth iteration. This yields

$$a_{i1}x_1^{(k+1)} + a_{i2}x_2^{(k+1)} + ... + a_{ii}x_i^{(k+1)} = -a_{i,i+1}x_{i+1}^{(k)} - ... - a_{in}x_n^{(k)} + b_i \tag{5.20}$$

for $i = 1, ..., n$, or, equivalently,

$$
\begin{aligned}
a_{11}x_1^{(k+1)} &= -a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - ... - a_{1n}x_n^{(k)} + b_1 \\
a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} &= \qquad\quad -a_{23}x_3^{(k)} - ... - a_{2n}x_n^{(k)} + b_2 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\
a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + ... + a_{n,n}x_n^{(k+1)} &= \qquad\qquad\qquad\qquad\qquad\qquad b_n.
\end{aligned}
\tag{5.21}
$$

Therefore,

$$
(D+L)\mathbf{x}^{(k+1)} = -U\mathbf{x}^{(k)} + \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x}^{(k+1)} = -(D+L)^{-1}U\mathbf{x}^{(k)} + (D+L)^{-1}\mathbf{b}
\tag{5.22}
$$

for $k = 1, 2, ...$. Introducing the notation $T_G = -(D+L)^{-1}U$ and $\mathbf{c}_G = (D+L)^{-1}\mathbf{b}$, we rewrite the Gauss-Seidel method in the form

$$
\mathbf{x}^{(k+1)} = T_G\mathbf{x}^{(k)} + \mathbf{c}_G.
\tag{5.23}
$$

Note that a lower-triangular matrix is nonsingular iff its diagonal elements are nonzero. In particular, $D+L$ is nonsingular (i.e. its inverse exists) iff $a_{ii} \neq 0$ for each $i = 1, 2, ..., n$.

5.3 The convergence of iterative techniques.

We will use the following notation

$$
E_i^{(k)} = |x_i^{(k)} - x_i|
\tag{5.24}
$$

and

$$
E^{(k)} = \|\mathbf{x}^{(k)} - \mathbf{x}\|_\infty = \max\{E_1^{(k)}, ..., E_n^{(k)}\}.
\tag{5.25}
$$

Then $\mathbf{x}^{(k)} \to \mathbf{x}$ is equivalent to $E^{(k)} \to 0$ as $k \to \infty$. We assume that $a_{ii} \neq 0$ for all i (otherwise both methods will not work).

For the Jacobi method we have

$$
x_i = \frac{1}{a_{ii}}\left(\sum_{j=1, j \neq i}^{n}(-a_{ij}x_j) + b_i\right)
\tag{5.26}
$$

and

$$
x_i^{(k+1)} = \frac{1}{a_{ii}}\left(\sum_{j=1, j \neq i}^{n}(-a_{ij}x_j^{(k)}) + b_i\right) \quad \text{for } i = 1, 2, ..., n
\tag{5.27}
$$

Subtracting Eq. 5.26 from Eq. 5.27 and using the triangle inequality, we obtain

$$
\begin{aligned}
E_i^{(k+1)} &= \frac{1}{|a_{ii}|}\left|\sum_{j=1, j \neq i}^{n} a_{ij}\left(x_j^{(k)} - x_j\right)\right| \leq \frac{1}{|a_{ii}|}\sum_{j=1, j \neq i}^{n}|a_{ij}|\left|x_j^{(k)} - x_j\right|, \\
&= \sum_{j=1, j \neq i}^{n}\frac{|a_{ij}|}{|a_{ii}|}E_j^{(k)} \leq \sum_{j=1, j \neq i}^{n}\frac{|a_{ij}|}{|a_{ii}|}E^{(k)} \leq \mu E^{(k)},
\end{aligned}
\tag{5.28}
$$

where

$$
\mu = \max_i \sum_{j=1, j \neq i}^{n}\frac{|a_{ij}|}{|a_{ii}|}.
\tag{5.29}
$$

Therefore,

$$
E^{(k+1)} \leq \mu, E^{(k)},
\tag{5.30}
$$

and if $\mu < 1$, then $\mathbf{x}^{(k)}$ converges to $\mathbf{x}$ strongly linearly, with rate of convergence μ.

Note that μ depends only on the coefficient matrix A. Note also that the terms in Eq. 8.34 are the absolute values of the entries of T_J, summed in each row, so that $\mu = \|T_J\|_\infty$. In the example, we considered in the previous chapter,

$$A = \begin{pmatrix} 10 & 2 & -1 \\ 1 & 8 & 3 \\ -2 & -1 & 10 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 7 \\ -4 \\ 9 \end{pmatrix}, \qquad T_J = \begin{pmatrix} 0 & -\frac{2}{10} & \frac{1}{10} \\ -\frac{1}{8} & 0 & -\frac{3}{8} \\ \frac{2}{10} & \frac{1}{10} & 0 \end{pmatrix}. \tag{5.31}$$

So, it follows from Eq. 8.34 that in our example $\mu = 1/2$. In general, the condition $\mu < 1$ is equivalent to

$$\sum_{j=1, j \neq i}^{n} |a_{ij}| < |a_{ii}| \quad \text{for all} \quad i. \tag{5.32}$$

Matrices A with this property are said to be *strictly diagonally dominant*. It follows that every strictly diagonally dominant square matrix is invertible. In fact it is not difficult to prove this directly, using elementary linear algebra.

For the Gauss-Seidel scheme the situation is slightly more complicated. Here we have

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} + b_i \right) \tag{5.33}$$

for $i = 1, 2, \dots, n$. Subtracting Eq. 5.26 from Eq. 5.33, we obtain

$$\begin{aligned} E_i^{(k+1)} &= \frac{1}{|a_{ii}|} \left| \sum_{j=1}^{i-1} a_{ij} \left(x_j^{(k+1)} - x_j \right) + \sum_{j=i+1}^{n} a_{ij} \left(x_j^{(k)} - x_j \right) \right| \\ &\leq \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}| \left| x_j^{(k+1)} - x_j \right| + \sum_{j=i+1}^{n} |a_{ij}| \left| x_j^{(k)} - x_j \right| \right) \\ &= \sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} E_j^{(k+1)} + \sum_{j=i+1}^{n} \frac{|a_{ij}|}{|a_{ii}|} E_j^{(k)} \leq \alpha_i E^{(k+1)} + \beta_i E^{(k)}, \end{aligned} \tag{5.34}$$

where

$$\begin{aligned} \alpha_i &= \sum j = 1^{i-1} \frac{|a_{ij}|}{|a_{ii}|}, \\ \beta_i &= \sum j = i + 1^{n} \frac{|a_{ij}|}{|a_{ii}|}, \\ \alpha_1 &= \beta_n = 0. \end{aligned} \tag{5.35}$$

Evidently, $\mu = \max_i(\alpha_i + \beta_i)$. From our analysis of the Jacobi method it is appropriate to assume $\mu < 1$, so that $\alpha_i + \beta_i < 1$ for all i and also $1 - \alpha_i > 0$ for all i. Suppose now that $E^{(k+1)} = E_m^{(k+1)}$ for some m (recalling that $E^{(k+1)}$ is the maximum of the $E_i^{(k+1)}$). Then we have

$$E^{(k+1)} = E_m^{(k+1)} \leq \alpha_m E^{(k+1)} + \beta_m E^{(k)}. \tag{5.36}$$

Hence,

$$E^{(k+1)} \leq \frac{\beta_m}{1 - \alpha_m} E^{(k)} \leq \eta\, E^{(k)}, \tag{5.37}$$

where

$$\eta = \max_i \frac{\beta_i}{1 - \alpha_i}. \tag{5.38}$$

Finally, we note that for each i we have

$$\alpha_i + \beta_i - \frac{\beta_i}{1 - \alpha_i} = \frac{\alpha_i[1 - (\alpha_i + \beta_i)]}{1 - \alpha_i} \geq \frac{\alpha_i[1 - \mu]}{1 - \alpha_i} \geq 0. \tag{5.39}$$

This implies that $\mu \geq \eta$. Indeed,

$$\alpha_i + \beta_i \geq \frac{\beta_i}{1 - \alpha_i} \quad \Rightarrow \quad \mu = \max i(\alpha_i + \beta_i) \geq \frac{\beta_i}{1 - \alpha_i} \quad \Rightarrow \quad \mu \geq \max_i \frac{\beta_i}{1 - \alpha_i} = \eta. \tag{5.40}$$

So, the Gauss-Seidel imethod converges strongly linearly with rate η, which is at least as fast as that of the Jacobi method. Although straightforward, computation of η is more complicated than μ. In our example we have

i	α_i	β_i	$1 - \alpha_i$	$\beta_i/(1 - \alpha_i)$
1	0	3/10	1	3/10
2	1/8	3/8	7/8	3/7
3	3/10	0	7/10	0

Therefore $\eta = 3/7 \approx 0.43$ (compared with $\mu = 0.5$).

Thus, we have proved the following:

Theorem 5.1. *If A is strictly diagonally dominant, then for any $\mathbf{x}^{(0)} \in \mathbb{R}^n$, both the Jacobi and Gauss-Seidel methods generate sequences $\{\mathbf{x}^{(k)}\}$ that converge to the unique solution of $A\mathbf{x} = \mathbf{b}$.*

Note that matrix A in our example is strictly diagonally dominant, so it is not unexpected that both Jacobi and Gauss-Seidel iterations converge to the solution of the system.

Remark. The above theorem gives us a sufficient condition for convergence of sequences generated by both methods. However, this condition is not necessary. If it is not satisfied, the Jacobi and/or Gauss-Seidel methods may still produce converging sequences. So, if A is not strictly diagonally dominant, we cannot predict whether the Jacobi method or Gauss-Seidel method will converge.

6 Approximation and interpolation

In many situations, it is useful to approximate a function f with a "simpler" function with desirable properties. For example, if the antiderivative of a function f is not known, then computing an integral $\int_a^b f(x)\,dx$ might be difficult. However, if we approximate f with a function P with known antiderivative, then $\int_a^b f(x)\,dx \approx \int_a^b P(x)\,dx$ can be computed easily. This is useful in numerical integration, which we will consider in the next chapter.

In this chapter we consider the case where the values of a function f are given only at at certain discrete points $x = x_i$, that is, f is given in the form of a table. For example, $f(x_i)$ might be the result of an experimental measurement or of numerical approximations. In this case, we would like to connect the points $(x_i, f(x_i))$ with a simple, reasonably smooth curve; this is called **interpolation**.

6.1 Polynomial interpolation

The "simple" functions that we will consider here are *polynomials*, that is, functions of the form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0, \tag{6.1}$$

where $a_0, ..., a_n$ are real numbers, with $a_n \neq 0$ is the *leading order coefficient* and where n is a nonnegative integer, called the *degree* of P and denoted $\deg P$.

The problem that we want to consider can be stated as follows:

Given distinct points $x_0, x_1, ..., x_n$ (not necessarily in increasing magnitude) in the domain of a function f, find a polynomial P with $\deg P \leq n$ such that

$$P(x_0) = f(x_0), \quad P(x_1) = f(x_1), \quad ... \quad P(x_n) = f(x_n). \tag{6.2}$$

Any such polynomial P is called an *interpolating polynomial* for f.

Polynomials are suitable for approximation and interpolation because of the following important result.

Theorem 6.1 (Weierstrass Approximation Theorem). *If $f : [a, b] \to \mathbb{R}$ is continuous on $[a, b]$, then for any $\epsilon > 0$, there is a polynomial $P(x)$ such that*

$$|f(x) - P(x)| < \epsilon \quad for \quad all \quad x \in [a, b]. \tag{6.3}$$

In other words, any function, continuous on the closed interval, can be uniformly approximated by a polynomial.

6.1.1 The Lagrange interpolating polynomial

We will now discuss a method of finding an interpolating polynomial. Let us first consider the simplest case, $n = 1$. Suppose we know the value of a function f at two points x_0, x_1. To interpolate the values of f by a first-degree polynomial means to determine a polynomial P of degree 1 (i.e., a straight line) that passes through two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. This polynomial has the form

$$P(x) = a_0 + a_1 x. \tag{6.4}$$

The conditions $P(x_0) = f(x_0)$ and $P(x_1) = f(x_1)$ give us the following system of linear equations for a_0 and a_1:

$$a_0 + a_1 x_0 = f(x_0), \quad a_0 + a_1 x_1 = f(x_1). \tag{6.5}$$

This system is solved easily for a_0 and a_1, and we obtain

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \quad a_0 = \frac{x_0 f(x_1) - x_1 f(x_0)}{x_1 - x_0}. \tag{6.6}$$

Thus, we have

$$P(x) = \frac{x_0 f(x_1) - x_1 f(x_0)}{x_1 - x_0} + \frac{f(x_1) - f(x_0)}{x_1 - x_0} x. \tag{6.7}$$

Let us now rewrite this polynomial in a slightly different form:

$$P(x) = f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0} \tag{6.8}$$

If we introduce functions

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}, \tag{6.9}$$

then we can write Eq. 6.7 as the polynomial P can be written as

$$P(x) = f(x_0)L_0(x) + f(x_1)L_1(x). \tag{6.10}$$

Note that functions $L_0(x)$ and $L_1(x)$ have the property that

$$L_0(x_0) = 1 = L_1(x_1), \quad L_0(x_1) = 0 = L_1(x_0). \tag{6.11}$$

This property ensures that $P(x)$, given by Eq. 6.7, satisfies the required conditions $P(x_0) = f(x_0)$ and $P(x_1) = f(x_1)$. Indeed,

$$\begin{aligned} P(x_0) &= f(x_0)L_0(x_0) + f(x_1)L_1(x_0) = f(x_0) \cdot 1 + f(x_1) \cdot 0 = f(x_0), \\ P(x_1) &= f(x_0)L_0(x_1) + f(x_1)L_1(x_1) = f(x_0) \cdot 0 + f(x_1) \cdot 1 = f(x_1). \end{aligned} \tag{6.12}$$

Let us now consider the general case. We construct a polynomial of degree at most n that passes through the $(n + 1)$ points $(x_0, f(x_0))$, $(x_1, f(x_1))$, ..., $(x_n, f(x_n))$. As a first step, we need a generalization of the functions L_0 and L_1 above; namely, we are looking for polynomials $L_0, \dots, L_n$ of degree n such that

$$L_k(x_i) = \delta_{ik} = \begin{cases} 0 & \text{if } i \neq k, \\ 1 & \text{if } i = k. \end{cases} \tag{6.13}$$

(The definition of these L_k also depends on $x_0, \dots, x_n$, and in particular on n. For ease of reading, we do not indicate this dependence explicitly.)

A short computation shows that polynomials with this property are given by

$$L_k(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i}. \tag{6.14}$$

We then set

$$P(x) = \sum_{j=0}^{n} f(x_j)L_j(x). \tag{6.15}$$

This is indeed a suitable interpolating polynomial. In fact it is the only possible interpolating polynomial, as the following theorem shows.

Theorem 6.2 (Lagrange interpolating polynomial). *Suppose that $x_0, x_1, \dots, x_n \in \mathbb{R}$ are distinct numbers in the domain of a function f. There exists a unique polynomial P with $\deg P \leq n$ such that*

$$f(x_k) = P(x_k) \quad \text{for all } k = 0, 1, \dots, n. \tag{6.16}$$

This polynomial, called the nth Lagrange interpolating polynomial, is given by Eq. 6.15, where the functions* $L_k(x)$ are given by Eq. 6.14.

Proof. It is evident from Eq. 6.14–Eq. 6.15 that $\deg P \leq n$. Moreover, by Eq. 6.13, we have for $k = 0, \dots, n$

$$P(x_k) = \sum_{j=0}^{n} f(x_j) L_j(x_k) = \sum_{j=0}^{n} f(x_j) \delta_{jk} = f(x_k). \tag{6.17}$$

The only remaining point is uniqueness. Suppose that P and $\hat{P}$ are two interpolating polynomials with degree at most n. Then

$$Q(x) := P(x) - \hat{P}(x) \tag{6.18}$$

is another polynomial, and $\deg Q \leq n$. However, since

$$P(x_k) = f(x_k) = \hat{P}(x_k) \quad \text{for all } k = 0, \dots, n, \tag{6.19}$$

we know that Q has $n + 1$ zeros, namely $x_0, \dots, x_n$. This contravenes the Fundamental Theorem of Algebra, and so the only possibility is $Q = 0$, whence $P = \hat{P}$. $\square$

Example 6.1. The values of a function f are given in the table:

k	x_k	$f(x_k)$
0	-1	-1
1	1	3
2	2	8

Let us construct the Lagrange interpolating polynomial of degree 2 for this data. From Eq. 6.14 we have

$$\begin{aligned}
L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1)(x - 2)}{(-1 - 1)(-1 - 2)} = \tfrac{1}{6}(x^2 - 3x + 2), \\
L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x + 1)(x - 2)}{(1 + 1)(1 - 2)} = -\tfrac{1}{2}(x^2 - x - 2), \\
L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x + 1)(x - 1)}{(2 + 1)(2 - 1)} = \tfrac{1}{3}(x^2 - 1).
\end{aligned} \tag{6.20}$$

Hence

$$\begin{aligned}
P(x) &= \sum_{j=0}^{2} f(x_j) L_j(x) \\
&= -1 \cdot \tfrac{1}{6}(x^2 - 3x + 2) - 3 \cdot \tfrac{1}{2}(x^2 - x - 2) + 8 \cdot \tfrac{1}{3}(x^2 - 1) \\
&= x^2 + 2x.
\end{aligned} \tag{6.21}$$

Above we have constructed the polynomial P to interpolate the values of a function f at the points $x_0, \dots, x_n$. But is P also a good approximation to f between these points? The theorem below gives an answer.

Theorem 6.3 (Error term for interpolating polynomials). *Suppose that $x_0, \dots, x_n$ are distinct numbers in the interval $[a, b]$ and that $f \in C^{n+1}[a, b]$. Then, for each $x \in [a, b]$, there exists a number $\xi \in (a, b)$ such that*

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - x_k), \tag{6.22}$$

where P is the nth interpolating polynomial.

Proof. We use *Rolle's theorem* in this proof. Rolle's theorem states that, if $h \in C^1[c, d]$ with $h(c) = h(d) = 0$, then there exists $x \in (c, d)$ such that $h'(x) = 0$.

If $x = x_k$ for some k, then $f(x_k) = P(x_k)$ and Eq. 6.22 holds for any choice of ξ. Therefore for the rest of this proof we assume that

$$x \neq x_k \quad \text{for all } k = 0, \dots, n. \tag{6.23}$$

Define the function

$$g(t) := f(t) - P(t) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{t - x_i}{x - x_i} \quad \text{for } t \in [a, b].$$

{eq-error-function-zeros} Since $f \in C^{n+1}[a, b]$, $P \in C^{\infty}[a, b]$ and in view of Eq. 6.23, it follows that $g \in C^{n+1}[a, b]$. Applying g at $t = x$ and at $t = x_k$ for $k = 0, \ldots, n$, we obtain

$$g(x) = f(x) - P(x) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{x - x_i}{x - x_i} = 0,$$

$$g(x_k) = f(x_k) - P(x_k) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{x_k - x_i}{x - x_i} = 0. \tag{6.24}$$

Thus $g \in C^{n+1}[a, b]$ and g has $n + 2$ distinct zeros at $x, x_0, x_1, \ldots, x_n$. Applying Rolle's theorem to each of the $n + 1$ subintervals between these zeros, it follows that the derivative $g' \in C^{n}[a, b]$ has at least $n + 1$ zeros in $[a, b]$. Again applying Rolle's theorem, the second derivative $g'' \in C^{n-1}[a, b]$ has at least n zeros in $[a, b]$. Applying the same argument to successive derivatives of g, it follows finally that the $(n + 1)^{\text{th}}$ derivative $g^{(n+1)}$ has at least one zero, which we call ξ. This means that

$$0 = g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - P^{(n+1)}(\xi) - (f(x) - P(x)) \frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} \frac{t - x_i}{x - x_i} \bigg|_{t=\xi}. \tag{6.25}$$

Considering each of the elements of this equation in turn, we have $P^{(n+1)}(\xi) = 0$ as $\deg P \leq n$. Also, we have

$$\frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} \frac{t - x_i}{x - x_i} = \frac{d^{n+1}}{dt^{n+1}} \frac{t^{n+1}}{\prod_{i=0}^{n}(x - x_i)} = \frac{(n + 1)!}{\prod_{i=0}^{n}(x - x_i)}. \tag{6.26}$$

Substituting this into Eq. 6.25, we obtain the equation

$$0 = f^{(n+1)}(\xi) - (f(x) - P(x)) \frac{(n + 1)!}{\prod_{i=0}^{n}(x - x_i)}, \tag{6.27}$$

which is equivalent to Eq. 6.25. □

Example 6.2. Let

$$f(x) = \frac{1}{x}. \tag{6.28}$$

The interpolating polynomial determined by the values of f at the points $x_0 = 2.0$, $x_1 = 2.5$ and $x_2 = 4$ is given by

$$P(x) = \tfrac{1}{20}x^2 - \tfrac{17}{40}x + \tfrac{23}{20}. \tag{6.29}$$

Let us obtain the theoretical bound for the error of approximation of $f(3)$ by $P(3)$. We have

$$|f'''(\xi)| = \left| -\frac{6}{\xi^4} \right| \leq \frac{3}{8} \quad \text{for all } \xi \in [2, 4]. \tag{6.30}$$

Therefore,

$$|f(3) - P(3)| \leq \max_{\xi \in [2,4]} \left| \frac{f'''(\xi)}{3!}(3 - 2)(3 - 2.5)(3 - 4) \right| \leq \frac{1}{32} = 0.03125. \tag{6.31}$$

The actual error of this approximation is

$$E = |f(3) - P(3)| = \frac{1}{3} - 0.325 = 0.008333\ldots, \tag{6.32}$$

which is smaller than our theoretical bound (as one would expect).

Now let us evaluate the error involved in approximating f by P on the whole interval $[2, 4]$. Again using Eq. 6.30, we have

$$|f(x) - P(x)| \leq \max_{\xi \in [2,4]} \left| \frac{f'''(\xi)}{3!} (x-2)(x-2.5)(x-4) \right|$$
$$\leq \frac{1}{16} |\phi(x)| \leq \frac{1}{16} \max_{x \in [2,4]} |\phi(x)|, \tag{6.33}$$

where
$$\phi(x) := (x-2)(x-2.5)(x-4) \quad \text{for } x \in [2, 4]. \tag{6.34}$$

The function ϕ has a maximum at $x' := \frac{17}{6} - \frac{\sqrt{13}}{6} \approx 2.232$, with $\phi(x') \approx 0.110$, and a minimum at $x'' := \frac{17}{6} + \frac{\sqrt{13}}{6} \approx 3.434$, where $\phi(x'') \approx -0.758$. By Eq. 6.33 we obtain

$$|f(x) - P(x)| \leq \frac{|\phi(x'')|}{16} \approx 0.048. \tag{6.35}$$

Finally, let us add one more point to our data, say, $x_3 = 3.5$. Then the polynomial interpolating the values of f at the four points $x_0 = 2$, $x_1 = 2.5$, $x_2 = 4$ and $x_3 = 3.5$ is given by

$$P(x) = -\frac{1}{70}x^3 + \frac{6}{35}x^2 - \frac{211}{280}x + \frac{201}{140}. \tag{6.36}$$

In this case, $P(3) = \frac{93}{280}$ and the actual error is

$$|f(3) - P(3)| = \left| \frac{1}{3} - \frac{93}{280} \right| = \frac{1}{840} \approx 0.0012. \tag{6.37}$$

This is almost 8 times smaller than the error of the interpolating polynomial based on the original three points.

Further reading: Section 3.1 of (Burden and Faires 2010).

6.1.2 Divided differences

As we saw in Theorem 6.2, the interpolating polynomial (of minimal degree) for a function at distinct points $x_0, x_1, \ldots, x_n$ is unique. However, it can be rewritten in many different ways. The Lagrange form Eq. 6.15 may not always be the optimal one for numerical purposes, since computing its value requires a large number of multiplications. Here we present an alternative form of the same polynomial, known as the *Newton form*.

Let us illustrate the idea again in the case of a linear polynomial, interpolating a function f between two points x_0 and x_1. It seems natural to write this straight line in the form

$$f(x_0) + m(x - x_0) \tag{6.38}$$

with slope $m = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$. We thus arrive at

$$P(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0). \tag{6.39}$$

Indeed, one checks that $P(x_0) = f(x_0)$, $P(x_1) = f(x_1)$, and so this is the unique interpolating polynomial between these points. The slope $\frac{f(x_1) - f(x_0)}{x_1 - x_0}$, which looks a bit like the derivative of f, is called the 1$^{\text{st}}$ divided difference and we write

$$f[x_0, x_1] := \frac{f(x_1) - f(x_0)}{x_1 - x_0}. \tag{6.40}$$

How does this generalize to higher-order polynomials? It turns out that the second-order interpolating polynomial through the points x_0, x_1, x_2 is given by

$$P(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)+$$
$$+ \underbrace{\frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}}_{=:f[x_0,x_1,x_2]}(x - x_0)(x - x_1). \tag{6.41}$$

(It is clear that $P(x_0) = f(x_0)$, $P(x_1) = f(x_1)$, and some computation yields $P(x_2) = f(x_2)$.) The term $f[x_0, x_1, x_2]$ is called the 2$^{\text{nd}}$ divided difference and reminds one of the second derivative.

Let us define these concepts more generally. We introduce the 0$^{\text{th}}$ divided difference as

$$f[x_i] := f(x_i), \tag{6.42}$$

and then define the kth divided difference recursively as

$$f[x_i, x_{i+1}, \dots, x_{i+k}] := \frac{f[x_{i+1}, x_{i+2}, \dots, x_{i+k}] - f[x_i, x_{i+1}, \dots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{6.43}$$

This agrees with the examples above.

We now claim that all interpolating polynomials can be written in terms of divided differences, in generalization of equations Eq. 6.39 and Eq. 6.41.

Theorem 6.4. *Let P be the nth order interpolating polynomial for a function f at the points $x_0, \dots, x_n$. It holds that*

$$P(x) = \sum_{k=0}^{n} f[x_0, x_1, \dots, x_k] \prod_{0 \le j < k} (x - x_j). \tag{6.44}$$

This relation is known as *Newton's divided-difference formula*. Note that an empty product is defined to be equal to 1, so in the $k = 0$ term in Eq. 6.44 the factor $\prod_{0 \le j < 0}(x - x_0) = 1$, so the first term in the sum is $f[x_0]$.

Proof. We use induction on n. For $n = 0$, we have $P(x) = f(x_0) = f[x_0]$ and Eq. 6.44 holds. Now suppose that Eq. 6.44 is already known for $n - 1$ in place of n. Denote by $\hat{P}$ the interpolating polynomial through $x_0, \dots, x_{n-1}$, and with $\check{P}$ the interpolating polynomial through $x_1, \dots, x_n$. By the induction hypothesis,

$$\hat{P}(x) = \sum_{k=0}^{n-1} f[x_0, x_1, \dots, x_k] \prod_{0 \le j < k} (x - x_j),$$
$$\check{P}(x) = \sum_{k=1}^{n} f[x_1, x_2, \dots, x_k] \prod_{1 \le j < k} (x - x_j). \tag{6.45}$$

Let P be the interpolating polynomial for f at $x_0, \dots, x_n$. We first prove that

$$P(x) = \frac{(x - x_0)\check{P}(x) - (x - x_n)\hat{P}(x)}{x_n - x_0}. \tag{6.46}$$

Indeed, one verifies the equality at $x = x_0$, and $x = x_1, \dots, x_{n-1}$, and at $x = x_n$ by a short computation in each case. Then Eq. 6.46 holds in generality due to the uniqueness of the interpolating polynomial (established in Theorem 6.2).

Now we compute the leading order coefficient c_n of P, i.e., the constant c_n such that $P(x) = c_n x^n +$ lower order terms. From Eq. 6.45, Eq. 6.46 one can directly read off that

$$c_n = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0} = f[x_0, \dots, x_n]. \tag{6.47}$$

Finally, let us define

$$Q(x) := P(x) - f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}). \tag{6.48}$$

This Q is a polynomial of at most order $n - 1$, since the leading coefficients of the two summands cancel. Also, $Q(x_i) = P(x_i) = f_i$ for all $0 \le i < n$. Uniqueness of the interpolating polynomial Theorem 6.2 implies $Q(x) = \hat{P}(x)$. This yields

$$
\begin{aligned}
P(x) &= \hat{P}(x) + f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}) \\
&= \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] \prod_{0 \le j < k} (x - x_j)
\end{aligned}
\tag{6.49}
$$

by Eq. 6.45, which completes the proof. $\square$

Remark. Note that nowhere in the proof we had to use that the interpolation points x_i had to be arranged in increasing order.

Writing the interpolating polynomial in terms of divided differences has several advantages:

- The polynomial requires fewer algebraic operations to evaluate. In fact, one might rewrite it as

$$P(x) = f(x_0) + (x - x_0)(f[x_0, x_1] + (x - x_1)(f[x_0, x_1, x_2] + \ldots \tag{6.50}$$

- If extra precision is required, it is easy to add an extra interpolation point x_{n+1} without recomputing the lower-order divided differences.

- We can see from Eq. 6.44 that, if the kth divided difference is constant, this means that the degree of the interpolating polynomial is k (because higher divided differences are all zero), so that we do not need to use all the data in the table ($k + 1$ points will be enough).

- The divided differences can be computed easily (by hand or with a computer) in a simple scheme, as shown in the next example.

Example 6.3. Suppose that the values of a function f at 7 points are as in Table 6.2. The remaining columns of that table illustrates how the divided differences are calulated.

Table 6.2: *Numerical example for divided difference method*

x	$f[x]$	$f[,]$	$f[,,]$	$f[,,,]$	$f[,,,,]$
-1	-2				
		3			
0	1		0		
		3		1	
1	4		3		0
		9		1	
2	13		6		0
		21		1	
3	34		9		0
		39		1	
4	73		12		
		63			
5	136				

The third divided difference is constant, so the interpolating polynomial is cubic. We obtain

$$
\begin{aligned}
P(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\
&= -2 + 3(x + 1) + 0(x + 1)x + (x + 1)x(x - 1) = x^3 + 2x + 1.
\end{aligned}
\tag{6.51}
$$

Further reading: Section 3.3 of (Burden and Faires 2010).

6.2 Cubic spline interpolation

In the previous sections we considered the approximation of arbitrary functions on closed intervals using a single polynomial. However, this does not always lead to satisfactory approximations because high-degree polynomials can oscillate erratically, and the error bounds can become large if the derivatives of the approximated functions are not bounded. An alternative approach is to divide the approximation interval into a collection of subintervals and construct a (generally) different approximating polynomial on each subinterval. This is called *piecewise-polynomial approximation*.

In this section we consider a function f defined at the points $x_0, \dots, x_n$. In contrast to the previous sections, we assume here that

$$x_0 < x_1 < \cdots < x_n. \tag{6.52}$$

The simplest such piecewise-polynomial approximation is *linear interpolation*, which consists of joining the data points $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ by a series of straight lines, i.e. the interpolating function S satisfies

$$S(x) = \begin{cases} f(x_0) + \frac{f(x_1)-f(x_0)}{x_1-x_0}(x - x_0) & \text{for } x \in [x_0, x_1], \\ f(x_1) + \frac{f(x_2)-f(x_1)}{x_2-x_1}(x - x_1) & \text{for } x \in [x_1, x_2], \\ \vdots \\ f(x_{n-1}) + \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}(x - x_{n-1}) & \text{for } x \in [x_{n-1}, x_n]. \end{cases} \tag{6.53}$$

Linear interpolation is simple, but it has the disadvantage that the interpolating function S is generally not differentiable at the interpolation points $x_1, \dots, x_{n-1}$.

The most common piecewise-polynomial approximation uses cubic polynomials between each successive pair of nodes and is called *cubic spline interpolation*. We will discuss this here only in the context of approximating functions, but splines more generally can approximate curves in the plane or in higher dimensions. This is useful for example for applications in digital art. For a very good introduction to splines in this context, see this YouTube video.

Definition 6.1 (Cubic spline interpolant)**.** A cubic spline interpolant S for a function f with known values at points $x_0 < x_1 < \cdots < x_n$ is a twice continuously differentiable function on $[x_0, x_n]$ (i.e. $S \in C^2[x_0, x_n]$) such that it is equal to a cubic polynomial, denoted S_k, on the interval $[x_{k-1}, x_k]$ for each $k = 1, \dots, n$ and $S(x_k) = f(x_k)$ for $k = 0, \dots, n$.

This definition implies that

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{6.54}$$

for $x \in [x_{i-1}, x_i]$ ($i = 1, 2, \dots, n$) and that S_i satisfy the following requirements:

$$S(x_i) = f_i \text{ for } i = 0, 1, \dots, n; \tag{6.55}$$

$$S_i(x_i) = S_{i+1}(x_i) \text{ for } i = 1, 2, \dots, n-1; \tag{6.56}$$

$$S_i'(x_i) = S_{i+1}'(x_i) \text{ for } i = 1, 2, \dots, n-1; \tag{6.57}$$

$$S_i''(x_i) = S_{i+1}''(x_i) \text{ for } i = 1, 2, \dots, n-1. \tag{6.58}$$

It is not obvious *a priori* that such an interpolant exists, or, if so, that it is unique. We have $4n$ unknown coefficients a_i, b_i, c_i, d_i ($i = 1, 2, \dots, n$). Condition 6.55 gives $n + 1$ equations. Conditions 6.56, 6.57, 6.58 give $3(n-1)$ equations. Thus, we have $n + 1 + 3(n - 1) = 4n - 2$ equations for $4n$ unknowns. So, we need to specify two more conditions to define $S(x)$ uniquely. Common choices are:

- $S''(x_0) = S''(x_n) = 0$ (natural cubic spline);
- $S'(x_0) = f'(x_0)$, $S'(x_n) = f'(x_n)$ (clamped cubic spline).

Natural cubic spline. Let $h_i = x_i - x_{i-1}$. Condition 6.55 gives

$$a_i = f_i \quad \text{for} \quad i = 1, 2, \dots, n \tag{6.59}$$

and

$$a_1 - b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = f_0. \tag{6.60}$$

Condition 6.56 gives

$$a_i = a_{i+1} - b_{i+1} h_{i+1} + c_{i+1} h_{i+1}^2 - d_{i+1} h_{i+1}^3 \quad \text{for} \quad i = 1, 2, \dots, n-1. \tag{6.61}$$

Condition 6.57 gives

$$b_i = b_{i+1} - 2c_{i+1} h_{i+1} + 3d_{i+1} h_{i+1}^2 \quad \text{for} \quad i = 1, 2, \dots, n-1. \tag{6.62}$$

Condition 6.58 gives

$$2c_i = 2c_- 6d_{i+1} h_{i+1} \quad \text{for} \quad i = 1, 2, \dots, n-1. \tag{6.63}$$

It follows from Eq. 6.63 that

$$d_i = \frac{c_i - c_{i-1}}{3h_i} \quad \text{for} \quad i = 2, \dots, n. \tag{6.64}$$

From the (endpoint) conditions $S''(x_0) = S''(x_n) = 0$ (corresponding to the natural cubic spline), it follows that

$$d_1 = \frac{c_1}{3h_1} \quad \text{and} \quad c_n = 0. \tag{6.65}$$

It follows from Eq. 6.61 that

$$b_{i+1} = \frac{a_{i+1} - a_i + c_{i+1} h_{i+1}^2 - d_{i+1} h_{i+1}^3}{h_{i+1}} \quad \text{for} \quad i = 1, 2, \dots, n-1.$$

and

$$b_i = \frac{a_i - a_{i-1} + c_i h_i^2 - d_i h_i^3}{h_i} \quad \text{for} \quad i = 2, 3, \dots, n, \tag{6.66}$$

so that

$$b_{i+1} - b_i = \frac{a_{i+1} - a_i}{h_{i+1}} - \frac{a_i - a_{i-1}}{h_i} + c_{i+1} h_{i+1} - c_i h_i - d_{i+1} h_{i+1}^2 + d_i h_i^2$$

for $i = 2, \dots, n-1$. Substituting this into Eq. 6.62 and using Eq. 6.59 and Eq. 6.64 we find that

$$\begin{aligned} h_i c_{i-1} + 2(h_i + h_{i+1})c_i + h_{i+1} c_{i+1} &= 3\left(\frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \right) \\ &= 3(f[x_i, x_{i+1}] - f[x_{i-1}, x_i]) \\ &= 3(h_i + h_{i+1}) f[x_{i-1}, x_i, x_{i+1}] \end{aligned} \tag{6.67}$$

for $i = 2, \dots, n-1$. Thus, we have obtained $n-2$ linear equations for $n-1$ unknowns $c_1, \dots, c_{n-1}$ (we already know that $c_n = 0$). One more equation is obtained as follows. First, the condition $S(x_0) = S_1(x_0) = f_0$ and Eq. 6.65 yield

$$b_1 = \frac{f_1 - f_0}{h_1} + \frac{2}{3} h_1 c_1. \tag{6.68}$$

On substituting this into Eq. 6.62 for $i = 1$ and using Eq. 6.66 for $i = 2$, we obtain

$$2(h_1 + h_2)c_1 + h_2 c_2 = 3\left(\frac{f_2 - f_1}{h_2} - \frac{f_1 - f_0}{h_1} \right) = 3(h_1 + h_2) f(x_0, x_1, x_2). \tag{6.69}$$

Eq. 6.67, Eq. 6.69 can be written as

$$
\begin{pmatrix}
A_1 & h_2 & 0 & \dots & & \dots & 0 \\
h_2 & A_2 & h_3 & & & & \vdots \\
0 & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & & 0 \\
\vdots & & & h_{n-2} & A_{n-2} & h_{n-1} \\
0 & \dots & & \dots & 0 & h_{n-1} & A_{n-1}
\end{pmatrix}
\begin{pmatrix}
c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1}
\end{pmatrix}
= 3
\begin{pmatrix}
F_1 \\ F_2 \\ \vdots \\ \vdots \\ \vdots \\ F_{n-2} \\ F_{n-1}
\end{pmatrix},
\tag{6.70}
$$

where

$$
A_i = 2(h_i + h_{i+1}) \quad \text{and} \quad F_i = (h_i + h_{i+1})f(x_{i-1}, x_i, x_{i+1})
\tag{6.71}
$$

for $i = 1, \dots, n-1$ The matrix of this system is symmetric and tridiagonal. Moreover, it is strictly diagonally dominant. So, it can be solved numerically using both Gaussian elimination and iterative techniques.

When $c_1, \dots, c_n$ are known, coefficients $d_1 \dots, d_n$ and $b_1 \dots, b_n$ can be determined using Eq. 6.64, Eq. 6.65, Eq. 6.66, Eq. 6.68, Eq. 6.69.

Example 6.4. Let us compute the natural spline for the function f given in Table 6.3.

Table 6.3: Numerical example for spline interpolation

k	x_k	$f(x_k)$
0	0.0	
1	0.5	
2	1.0	-1
3	1.5	
4	2.0	

From Eq. 6.59, we obtain

$$
a_1 = 0, \quad a_2 = -1, \quad a_3 = 0, \quad a_4 = 1.
\tag{6.72}
$$

Also, we have

$$
F_1 = 0, \quad F_2 = 12, \quad F_3 = 0.
\tag{6.73}
$$

We then need to solve the system

$$
\begin{pmatrix}
2 & \frac{1}{2} & 0 \\
\frac{1}{2} & 2 & \frac{1}{2} \\
0 & \frac{1}{2} & 2
\end{pmatrix}
\begin{pmatrix}
c_1 \\ c_2 \\ c_3
\end{pmatrix}
=
\begin{pmatrix}
0 \\ 12 \\ 0
\end{pmatrix},
\tag{6.74}
$$

which gives

$$
c_1 = -\frac{12}{7}, \quad c_2 = \frac{48}{7}, \quad c_3 = -\frac{12}{7}.
\tag{6.75}
$$

We also have $c_4 = 0$.

Substituting these into Eq. 6.64–Eq. 6.66, we find

$$
\begin{aligned}
b_1 &= -\frac{18}{7}, \quad b_2 = 0, \quad b_3 = \frac{18}{7}, \quad b_4 = \frac{12}{7}, \\
d_1 &= -\frac{8}{7}, \quad d_2 = \frac{40}{7}, \quad d_3 = -\frac{40}{7}, \quad d_4 = \frac{8}{7}.
\end{aligned}
\tag{6.76}
$$

which allows us to write the spline interpolant as

$$
S(x) =
\begin{cases}
-(8/7)x^3 - (12/7)x + 1 & \text{if } x \in [0, 0.5), \\
(40/7)x^3 - (72/7)x^2 + (24/7)x + 1/7 & \text{if } x \in [0.5, 1), \\
-(40/7)x^3 + 24x^2 - (216/7)x + 81/7 & \text{if } x \in [1, 1.5), \\
(8/7)x^3 - (48/7)x^2 + (108/7)x - 81/7 & \text{if } x \in [1.5, 2].
\end{cases}
\tag{6.77}
$$

Further reading: Section 3.5 of (Burden and Faires 2010).

7 Numerical integration

We now discuss approximation algorithms for evaluating definite integrals. For many functions of practical relevance, their antiderivative is not explicitly known, so that integrating them is not possible in explicit terms. For example, an integral like

$$\int_0^2 \exp(-x^2)dx \tag{7.1}$$

can be evaluated only by numerical approximation.

The basic method for approximating an integral of a function $f(x)$ is called *numerical quadrature* and uses a formula of the form

$$\int_a^b f(x)dx \approx \sum_{i=0}^n c_i f(x_i), \tag{7.2}$$

where $x_0, \ldots, x_n$ are *nodes* and $c_0, \ldots, c_n$ are *weights*. The nodes are chosen in the interval $[a, b]$ and the weights are chosen so that the formula is exact for polynomials of degree n. The integral of a general function f is then approximated by the integral of the interpolating polynomial P_n of degree n through the nodes $x_0, \ldots, x_n$ of f. In other words, the integral is replaced with a discrete sum of function values of f with certain numerical coefficients.

For a simple example, consider $\int_a^b f(x)dx$ for a nonnegative function f; illustrated in Figure 7.1. The integral $\int_a^b f(x)dx$ corresponds to the area under the graph of f in the interval $[a, b]$. This region can be approximated by a trapezium through the points $(a, 0)$, $(a, f(a))$, $(b, f(b))$, $(b, 0)$. Computing the area of the trapezium, we obtain

$$\int_a^b f(x)dx \approx (b-a)\frac{(f(a) + f(b))}{2}. \tag{7.3}$$

This is the so-called *Trapezium rule*.

We motivated the Trapezium rule geometrically, but two question remain open at this point:

(a) How large is the approximation error?

(b) How can the approximation be improved for additional precision?

We will answer these questions by generalizing the Trapezium rule as follows. First, we select a collection of distinct points $x_0, x_1, \ldots, x_n$ from the interval $[a, b]$. Then, in generalization of the straight line that interpolated the function f in the Trapezium method, we construct the Lagrange interpolating polynomial (see Theorem 6.3) through $x_0, \ldots, x_n$ to obtain

$$f(x) = \sum_{i=0}^n f(x_i)L_i(x) + R(x), \tag{7.4}$$

where

$$R(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\prod_{i=0}^n (x - x_i) \tag{7.5}$$

and $\xi(x) \in [a, b]$ for each x. Integrating this formula over $[a, b]$, we obtain

$$\int_a^b f(x)dx = \sum_{i=0}^n c_i f(x_i) + E(f) \tag{7.6}$$
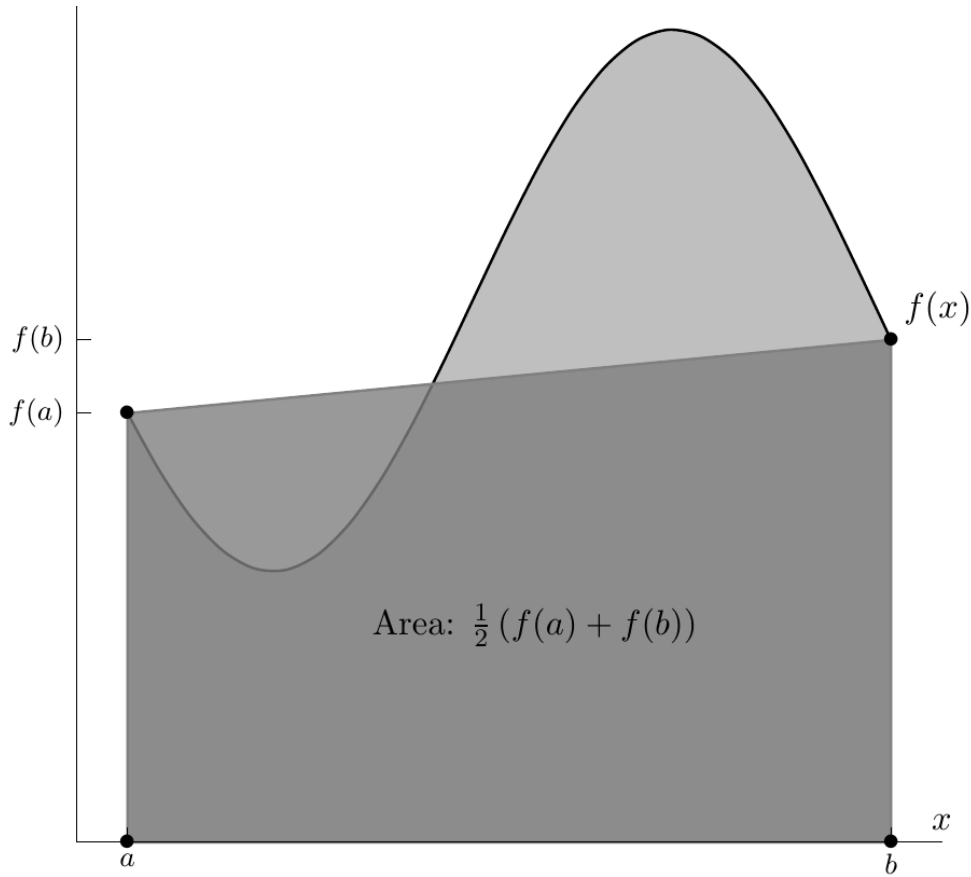
Figure 7.1: Trapezium rule

where

$$c_i = \int_a^b L_i(x)dx, \quad E(f) = \int_a^b R(x)\, dx. \tag{7.7}$$

The idea is that it is desirable for the error term $E(f)$ to be small—more on that below.

It is usual to choose the nodes $x_0, \dots, x_n$ equally spaced in the interval $[a, b]$, in other words, with $h = \frac{1}{n}(b - a)$, one sets

$$x_i := a + ih \quad \text{for } i = 0, \dots, n. \tag{7.8}$$

The construction above then yields the so called **closed Newton-Cotes formulae** for integration. Let us consider the cases of $n = 1$ (which corresponds to the Trapezium rule as above) and $n = 2$ in more detail.

7.1 Trapezium rule

Let us now put $n = 1$, that is, we use interpolation by a straight line. Following the recipe from above, we have $h := b - a$, the nodes are $x_0 = a$ and $x_1 = b$, and we have the Lagrange polynomials

$$
\begin{aligned}
L_0(x) &= \frac{x - x_1}{x_0 - x_1} = -\frac{1}{h}(x - b), \\
L_1(x) &= \frac{x - x_0}{x_1 - x_0} = \frac{1}{h}(x - a).
\end{aligned}
\tag{7.9}
$$

The constants c_0, c_1 are then

$$
\begin{aligned}
c_0 &= \int_a^b L_0(x)dx = -\frac{1}{h}\int_a^b (x-b)dx \\
&= -\frac{1}{2h}\left[(x-b)^2\right]_{x=a}^b = \frac{h}{2}, \\
c_1 &= \int_a^b L_1(x)dx = \frac{1}{h}\int_a^b (x-a)dx \\
&= \frac{1}{2h}\left[(x-a)^2\right]_{x=a}^b = \frac{h}{2}.
\end{aligned}
\tag{7.10}
$$

The approximation formula Eq. 7.6 then gives

$$
\int_a^b f(x)dx \approx \frac{h}{2}(f(a)+f(b)),
\tag{7.11}
$$

which is exactly the Trapezium rule.

Remark. It can be shown by direct calculation that the Trapezium rule produces an exact result for $f(x) = 1$, $f(x) = x$ and, hence for any polynomial of degree 0 and 1. (Prove it!)

Let us now consider the error term $E(f)$. We assume that $f \in C^2[a, b]$ and employ the formula for the error of the interpolation polynomial

$$
f(x) = P(x) + \frac{f''(\xi(x))}{2!}(x-x_0)(x-x_1).
\tag{7.12}
$$

Since $f \in C^2[a, b]$, we have

$$
m \le f''(x) \le M \quad \text{for all} \quad x \in [a, b],
\tag{7.13}
$$

where

$$
m = \min_{x \in [a,b]} f''(x), \quad M = \max_{x \in [a,b]} f''(x).
\tag{7.14}
$$

Let

$$
B(x) = -\frac{(x-x_0)(x-x_1)}{2}.
\tag{7.15}
$$

Evidently, $B(x) \ge 0$ for all $x \in [a, b]$. Therefore, it follows from Eq. 7.12 that

$$
m\,B(x) \le P(x) - f(x) \le M\,B(x).
\tag{7.16}
$$

Integration of these inequalities yields

$$
m\int_{x_0}^{x_1} B(x)dx \le \int_{x_0}^{x_1} (P(x)-f(x))\,dx \le M\int_{x_0}^{x_1} B(x)dx.
\tag{7.17}
$$

Since

$$
\begin{aligned}
\int_{x_0}^{x_1} B(x)dx &= \frac{1}{2}\int_{x_0}^{x_1} (x-x_0)(x_1-x)dx \\
&= \frac{1}{2}\int_0^1 h\,t\,h(1-t)\,h\,dt \\
&= \frac{h^3}{2}\left(\frac{t^2}{2}-\frac{t^3}{3}\right)\Bigg|_{t=0}^{t=1} = \frac{h^3}{12}.
\end{aligned}
\tag{7.18}
$$

(here we have introduced new variable of integration t by the formula $x = x_0 + th$) Eq. 7.17 can we rewritten as

$$
m \le \frac{12}{h^3}\int_{x_0}^{x_1} (P(x)-f(x))\,dx \le M.
\tag{7.19}
$$

Since $f''(x)$ is continuous in $[a, b]$, the intermediate value theorem says that there exists $\xi \in [a, b]$ such that

$$f''(\xi) = \frac{12}{h^3} \int_{x_0}^{x_1} (P(x) - f(x))\, dx. \tag{7.20}$$

This and the expression for the integral of $P(x)$ give us the *Trapezium rule* with error term:

$$\int_{x_0}^{x_1} f(x) dx = \frac{h}{2} \left(f(x_0) + f(x_1) \right) - \frac{h^3}{12} f''(\xi). \tag{7.21}$$

We see that the error term can be controlled by an estimate of the second derivative of f. Also, it will be small if h is small. Evidently, for a fixed interval $[a, b]$ we cannot vary h to reduce the error; we will return to this problem later.

7.2 Simpson's rule

Now let us consider the case $n = 2$, which should yield a better approximation. We have $h = \frac{1}{2}(b-a)$, and our nodes are $x_0 = a$, $x_1 = a + h$, $x_2 = a + 2h = b$. We first recall the polynomials L_j:

$$\begin{aligned}
L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{1}{2h^2}(x - (a + h))(x - b), \\
L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = -\frac{1}{h^2}(x - a)(x - b), \\
L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{1}{2h^2}(x - a)(x - (a + h)).
\end{aligned} \tag{7.22}$$

Their integrals are, with the same substitution $x = a + ht$ as before (but now $0 \le t \le 2$),

$$\begin{aligned}
c_0 &= \int_a^b L_0(x)\, dx = \frac{h}{2} \int_0^2 (t-1)(t-2) dt = \frac{h}{2}\frac{2}{3} = \frac{h}{3}, \\
c_1 &= \int_a^b L_1(x)\, dx = -h \int_0^2 t(t-2) dt = -h\left(-\frac{4}{3}\right) = \frac{4h}{3}, \\
c_2 &= \int_a^b L_2(x)\, dx = \frac{h}{2} \int_0^2 t(t-1) dt = \frac{h}{2}\frac{2}{3} = \frac{h}{3}.
\end{aligned} \tag{7.23}$$

This means that our approximation formula is

$$\int_a^b f(x) dx \approx \frac{h}{3} \left(f(a) + 4f(a + h) + f(b) \right). \tag{7.24}$$

This is known as *Simpson's rule*, which is graphically represented in Figure 7.2.

Error term for Simpson's rule. Let us recall the Trapezium rule with error term:

$$\int_{x_0}^{x_1} f(x) dx = \frac{h}{2} \left[f(x_0) + f(x_1) \right] - \frac{h^3}{12} f''(\xi). \tag{7.25}$$

It has been obtained by integrating the linear interpolating polynomial with nodes x_0 and $x_1 = x_0 + h$. Therefore, it must produce an exact result for any polynomial of degree 0 and 1. Indeed, the error term in Eq. 11.59 vanishes for polynomials of degree 0 and 1.

Simpson's rule has been obtained by integrating the quadratic interpolating polynomial with nodes x_0, $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$. So, it must produce an exact result for polynomials of degree 0, 1 and 2. Will it generate a nonzero error for cubic polynomials? To examine this, it suffices to consider $f(x) = x^3$. By analogy with the Trapezium rule we assume that Simpson's rule with error term has the form

$$\int_{x_0}^{x_2} f(x) dx = \frac{h}{3} \left[f(x_0) + 4f(x_1) + f(x_2) \right] + C f'''(\xi) \tag{7.26}$$
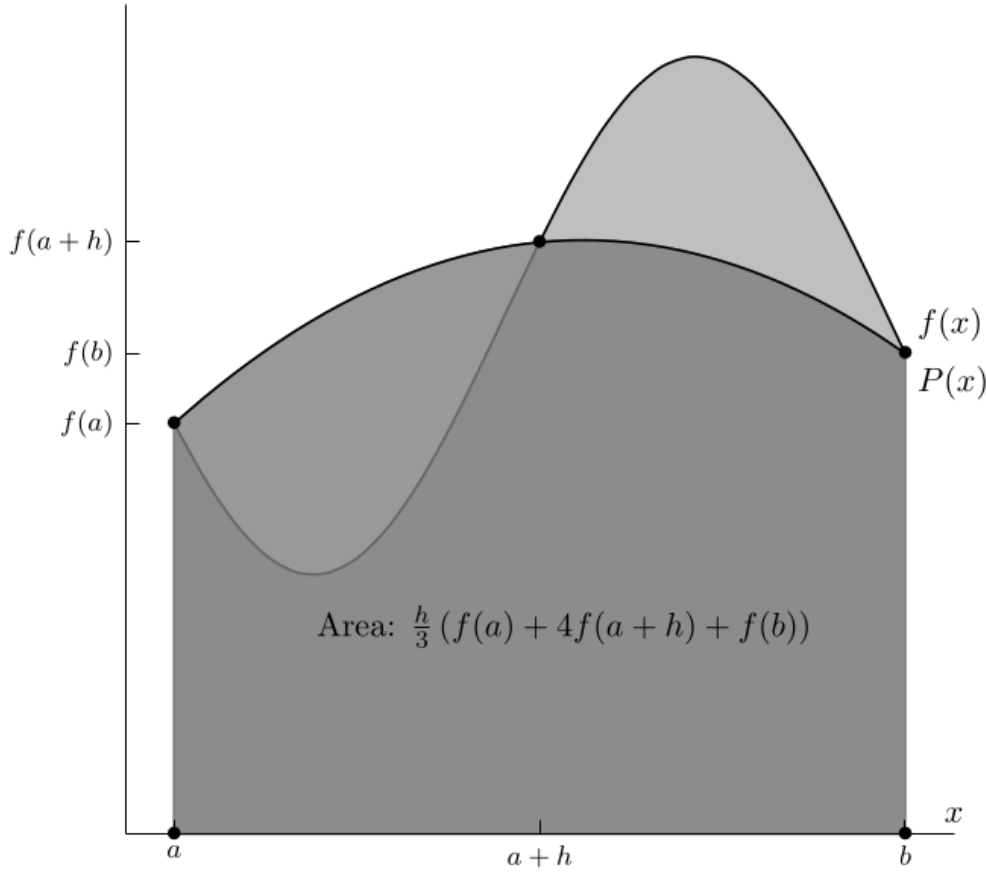
Figure 7.2: Simpson's rule

for some constant C and some $\xi \in [x_0, x_2]$. Substituting $f(x) = x^3$ in Eq. 7.26, we obtain

$$\int_{x_0}^{x_2} x^3 dx = \frac{h}{3} \left[x_0^3 + 4x_1^3 + x_2^3 \right] + 6C. \tag{7.27}$$

The integral on the left hand side of Eq. 11.69 can be written as

$$
\begin{aligned}
(\text{ l.h.s. }) &= \frac{x_2^4}{4} - \frac{x_0^4}{4} = \frac{(x_0 + 2h)^4}{4} - \frac{x_0^4}{4} \\
&= 2hx_0^3 + 6h^2 x_0^2 + 8h^3 x_0 + 4h^4.
\end{aligned} \tag{7.28}
$$

For the right hand side of Eq. 11.69, we have

$$
\begin{aligned}
(\text{ r.h.s. }) &= \frac{h}{3} \left[x_0^3 + 4(x_0 + h)^3 + (x_0 + 2h)^3 \right] + 6C \\
&= \frac{h}{3} \Big[x_0^3 + 4 \left(x_0^3 + 3x_0^2 h + 3x_0 h^2 + h^3 \right) \\
&\qquad + x_0^3 + 6x_0^2 h + 12x_0 h^2 + 8h^3 \Big] + 6C \\
&= \frac{h}{3} \left[6x_0^3 + 18x_0^2 h + 24x_0 h^2 + 12h^3 \right] + 6C \\
&= 2hx_0^3 + 6h^2 x_0^2 + 8h^3 x_0 + 4h^4 + 6C.
\end{aligned} \tag{7.29}
$$

If follows from Eq. 7.28 and Eq. 7.29 that Eq. 11.69 simplifies to $0 = 6C$, so that $C = 0$. This means that Simpson's rule is exact for polynomials of degree 3 (an unexpected result!).

This also means that our assumption about the error term for Simpson's rule is wrong. So, we make another assumption, namely:

$$\int_{x_0}^{x_2} f(x) dx = \frac{h}{3} \left[f(x_0) + 4f(x_1) + f(x_2) \right] + Cf^{(4)}(\xi). \tag{7.30}$$

To find C, we substitute $f(x) = x^4$ in Eq. 7.30. This yields the equation

$$\int_{x_0}^{x_2} x^4 dx = \frac{h}{3}\left[x_0^4 + 4x_1^4 + x_2^4\right] + 24C. \tag{7.31}$$

The left hand side of Eq. 7.31 can be written as

$$(\text{l.h.s. }) = \frac{x_2^5}{5} - \frac{x_0^5}{5} = \frac{(x_0 + 2h)^5}{5} - \frac{x_0^4}{4} \tag{7.32}$$
$$= 2hx_0^4 + 8h^2 x_0^3 + 16h^3 x_0^2 + 16h^4 x_0 + \frac{32}{5}h^5.$$

For the right hand side of Eq. 7.31, we have

$$(\text{r.h.s. }) = \frac{h}{3}\left[x_0^4 + 4(x_0 + h)^4 + (x_0 + 2h)^4\right] + 24C$$
$$= \frac{h}{3}\Big[x_0^3 + 4\left(x_0^4 + 4x_0^3 h + 6x_0^2 h^2 + 4x_0 h^3 + h^4\right)$$
$$+ x_0^4 + 8x_0^3 h + 24x_0^2 h^2 + 32x_0 h^3 + 16h^4\Big] + 24C \tag{7.33}$$
$$= \frac{h}{3}\left[6x_0^4 + 24x_0^3 h + 48x_0^2 h^2 + 48x_0 h^3 + 20h^4\right] + 24C$$
$$= 2hx_0^4 + 8h^2 x_0^3 + 16h^3 x_0^2 + 16h^4 x_0 + \frac{20}{3}h^5 + 24C.$$

Eq. 7.31–Eq. 7.33 imply that

$$\frac{32}{5}h^5 = \frac{20}{3}h^5 + 24C. \tag{7.34}$$

so that

$$C = -\frac{h^5}{90}. \tag{7.35}$$

Thus, Simpson's rule with error term is given by

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90}f^{(4)}(\xi) \tag{7.36}$$

for some $\xi \in [x_0, x_2]$. Note that the above argument is not a proof of the existence of such ξ. Alternative derivations of formula Eq. 7.36 can be found in textbooks on Numerical Analysis (e.g. (Burden and Faires 2010)).

7.3 Higher-order Newton-Cotes formulae

We have now seen that the Trapezium and Simpson's rules are examples of *Newton-Cotes formulae*, which are obtained by integrating of an interpolating polynomial for interpolation points $x_0, \dots, x_n$. The Trapezium rule and Simpson's rule correspond to $n = 1$ and $n = 2$ respectively. For $n = 3$, we have **Simpson's three-eighth rule**

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] - \frac{3h^5}{80}f^{(4)}(\xi), \tag{7.37}$$

where $h = (x_3 - x_0)/3$ and $\xi \in [x_0, x_3]$. The error of Simpson's three-eighth rule is proportional to h^5 with a coefficient that is larger than that in the error term of Simpson's rule, so this method seems less efficient than Simpson's rule.

For $n = 4$ we have the formula

$$\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right]$$
$$- \frac{8h^7}{945}f^{(6)}(\xi). \tag{7.38}$$

where $h = (x_4 - x_0)/4$ and $\xi \in [x_0, x_4]$.

7.4 Composite numerical integration

The Newton-Cotes formulas are generally unsuitable for large integration intervals. This would require high-degree formulas. An alternative to this is a *piecewise* approach to numerical integration that uses the low-order Newton-Cotes formulas such as the Trapezium and Simpson's rules.

Let us apply Simpson's formula to approximate the integral $\int_0^4 x^4 dx$. We have

$$I = \int_0^4 x^4 dx \approx \frac{2}{3} \left[0 + 4 \cdot 2^4 + 4^4 \right] = \frac{2}{3}(64 + 256) = 213.333,$$

$$E = |I - 213.333| = |204.8 - 213.333| = 8.533.$$

(7.39)

To apply a piecewise technique to this problem, we divide $[0, 4]$ into two subintervals $[0, 2]$ and $[2, 4]$ and use Simpson's rule twice with $h = 1$:

$$I = \int_0^4 x^4 dx \approx \frac{1}{3} \left[0 + 4 \cdot 1^4 + 2^4 \right]$$

$$+ \frac{1}{3} \left[2^4 + 4 \cdot 3^4 + 4^4 \right] = 205.333,$$

$$E = |204.8 - 205.333| = 0.533.$$

(7.40)

To reduce the error, we can proceed further by subdividing the intervals $[0, 2]$ and $[2, 4]$ and use Simpson's rule with $h = 1/2$.

To generalize this procedure, we choose an *even* integer n and divide the interval $[a, b]$ into n subintervals. Then we apply Simpson's rule to each consecutive pair of subintervals. With $h = (b - a)/n$ and $x_j = a + jh$ for $j = 0, 1, ..., n$, we have

$$\int_a^b f(x)dx = \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)dx$$

$$= \sum_{j=1}^{n/2} \left\{ \frac{h}{3} \left[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j}) \right] - \frac{h^5}{90} f^{(4)}(\xi_j) \right\}$$

(7.41)

for some ξ_j between x_{2j-2} and x_{2j}, provided that $f \in C^4[a, b]$.

One can see that for each $j = 1, 2, ..., (n/2) - 1$, the number $f(x_{2j})$ appears in the term corresponding to the interval $[x_{2j-2}, x_{2j}]$ and also in the term corresponding to the interval $[x_{2j}, x_{2j+2}]$. Taking this into account, we obtain

$$\int_a^b f(x)dx = \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)dx$$

$$= \frac{h}{3} \left\{ f(x_0) + 2 \sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4 \sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n) \right\} + E.$$

(7.42)

The error of this approximation is

$$E = -\frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j)$$

(7.43)

where $x_{2j-2} < \xi_j < x_{2j}$ for each $j = 1, 2, ..., n/2$. If $f \in C^4[a, b]$, then (according to the Extreme Value Theorem) $f^{(4)}(x)$ attains its maximum and minimum values in $[a, b]$. Let

$$m = \min_{x \in [a,b]} f^{(4)}(x), \quad M = \max_{x \in [a,b]} f^{(4)}(x).$$

(7.44)

Then we have

$$m \le f^{(4)}(\xi_j) \le M \quad \text{for each} \quad j = 1, 2, ..., n/2.$$

(7.45)

Therefore,

$$\frac{n}{2}m \le \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le \frac{n}{2}M, \tag{7.46}$$

and

$$m \le \frac{2}{n}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le M. \tag{7.47}$$

By the Intermediate Value Theorem, there is a $\mu \in (a,b)$ such that

$$f^{(4)}(\mu) = \frac{2}{n}\sum_{j=1}^{n/2} f^{(4)}(\xi_j). \tag{7.48}$$

Hence,

$$E = -\frac{h^5}{180}nf^{(4)}(\mu) = -\frac{b-a}{180}h^4 f^{(4)}(\mu). \tag{7.49}$$

Thus, we have proved the following theorem.

Theorem 7.1 (Composite Simpson's rule). *Let $f \in C^4[a,b]$, n be even, $h = (b-a)/n$, and $x_j = a+jh$ for $j = 0, 1, ..., n$. There exist a number $\mu \in (a,b)$ such that*

$$\int_a^b f(x)dx = \frac{h}{3}\left\{ f(x_0) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right\} \tag{7.50}$$
$$- \frac{b-a}{180}h^4 f^{(4)}(\mu).$$

Similarly, one can prove analogous theorems for Composite Trapezium rule.

Theorem 7.2 (Composite Trapezium rule). *Let $f \in C^2[a,b]$, $h = (b-a)/n$, and $x_j = a + jh$ for $j = 0, 1, ..., n$. There exist a number $\mu \in (a,b)$ such that*

$$\int_a^b f(x)dx = \frac{h}{2}\left\{ f(x_0) + 2\sum_{j=1}^{n-1} f(x_j) + f(x_n)\right\} - \frac{b-a}{12}h^2 f''(\mu). \tag{7.51}$$

Example 7.1. Apply the composite Simpson rule to compute

$$I = \int_0^2 e^x dx \tag{7.52}$$

with absolute error less than 10^{-2}.

Solution. First we need to determine the number of subintervals* n in the composite Simpson rule that would ensure that absolute error less than 10^{-2}. It follows from Eq. 7.50 that

$$E = \frac{b-a}{180}h^4|f^{(4)}(\mu)| \le \frac{b-a}{180}h^4 M, \tag{7.53}$$

where M is the upper bound for $|f^{(4)}(x)|$ in $[0,2]$. Evidently, $M = e^2$. Therefore, we require that

$$\frac{b-a}{180}h^4 e^2 < 10^{-2} \quad \Leftrightarrow \quad \frac{(b-a)^5}{180n^4}e^2 < 10^{-2} \quad \Leftrightarrow \quad n^4 > \frac{100(b-a)^5}{180}e^2. \tag{7.54}$$

Substituting $a = 0$ and $b = 2$, we find that

$$n^4 > \frac{5 \cdot 32e^2}{9} = 131.3609973, \quad \text{so that} \quad n \ge 4 \ (n^4 = 256). \tag{7.55}$$

Applying the composite Simpson rule with $n = 4$, we obtain the approximation

$$\tilde{I} = 6.391210187. \tag{7.56}$$

The actual error is

$$|I - \tilde{I}| = 0.002154088. \tag{7.57}$$

So it is indeed less that 10^{-2}.

All the Composite Newton-Cotes techniques are stable with respect to roundoff error. Consider, for example, the Composite Simpson rule with n subintervals applied to a function $f(x)$ on $[a, b]$. We assume that $f(x_i)$ is approximated by $\tilde{f}(x_i)$ with the roundoff error e_i. Then, the accumulated error in the Composite Simpson rule is

$$
\begin{aligned}
E(h) &= \left| \frac{h}{3} \left(e_0 + 2 \sum_{j=1}^{(n/2)-1} e_{2j} + 4 \sum_{j=1}^{n/2} e_{2j-1} + e_n \right) \right| \\
&\leq \frac{h}{3} \left(|e_0| + 2 \sum_{j=1}^{(n/2)-1} |e_{2j}| + 4 \sum_{j=1}^{n/2} |e_{2j-1}| + |e_n| \right)
\end{aligned}
\tag{7.58}
$$

If the roundoff errors are uniformly bounded by ε, then

$$E(h) \leq \frac{h}{3} \left(\varepsilon + 2 \left(\frac{n}{2} - 1 \right) \varepsilon + 4 \frac{n}{2} \varepsilon + \varepsilon \right) = nh\varepsilon = (b-a)\varepsilon. \tag{7.59}$$

Thus, the bound for the accumulated error is independent of h, so that h can be taken as small as we wish.

8 Numerical differentiation

Often we need to calculate the derivative of a function whose values are given only for a finite set of points. So, we need a formula which would approximate the derivatives of our function at these points and which would use only the values of the function at these points.

Recall that, by definition, the derivative of the function $f(x)$ at x_0 is

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{8.1}$$

It is natural to expect that if h is sufficiently small, then

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}. \tag{8.2}$$

This formula approximates the derivative using values of f at two points x_0 and $x_0 + h$.

What is a general method of constructing approximate formulas for the derivative? There are many ways of doing that. For example, given a set of distinct points $x_0, \ldots, x_n$ and values of f at these points, we can construct the interpolating polynomial and then compute the derivatives of this polynomial at each $x_0, \ldots, x_n$. This method, although possible, is less transparent and instructive than a method based of Taylor's polynomials (series) which we will discuss below.

Let $f \in C^{(n+1)}(I)$ where I is an open interval. Then we have the Taylor formula (of order n):

$$
\begin{aligned}
f(x) =& f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \ldots \\
&+ \frac{(x - x_0)^n}{n!} f^{(n)}(x_0) + \frac{(x - x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi)
\end{aligned} \tag{8.3}
$$

for any $x, x_0 \in I$ and for some ξ between x and x_0 (ξ depends on x and x_0). If we use the notation $h = x - x_0$, then Eq. 8.3 takes the form

$$
\begin{aligned}
f(x_0 + h) =& f(x_0) + h f'(x_0) + \frac{h^2}{2!} f''(x_0) + \ldots \\
&+ \frac{h^n}{n!} f^{(n)}(x_0) + \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\xi)
\end{aligned} \tag{8.4}
$$

with ξ between x_0 and $x_0 + h$ (ξ can also be written as $\xi = x_0 + \theta h$ where $0 < \theta < 1$).

Definition 8.1. If function $g(h)$ has the property that

$$|g(h)| \le K|h^p|, \quad K, p > 0, \tag{8.5}$$

for all h in some open interval containing 0, we write

$$g(h) = O(h^p) \tag{8.6}$$

as $h \to 0$. This is pronounced "g is big-oh of h^p". We sometimes say that $g(h)$ converges to 0 as fast as h^p.

For example, function $f(x) = \sin(x)/x - 1$ converges to 0 as fast as x^2 converges to zero (as $x \to 0$). To show this, it suffices to consider the third Taylor polynomial for $\sin(x)$:

$$\sin(x) = x - \frac{x^3}{3!} \cos(\xi) \tag{8.7}$$

where ξ is some number between 0 and x. We have

$$\left| \frac{\sin x}{x} - 1 \right| = \frac{|x|^2}{3!} |\cos(\xi)| \leq \frac{x^2}{3!} = \frac{x^2}{6} \quad \Rightarrow \quad \frac{\sin x}{x} - 1 = O(x^2). \tag{8.8}$$

Here we used the fact that $|\cos x| \leq 1$ for all $x \in \mathbb{R}$.

Properties of $O(h^n)$:

1. $O(h^n) + O(h^m) = O(h^l)$ for $n, m > 0$ and $l = \min\{n, m\}$.

2. $O(h^n)O(h^m) = O(h^{n+m})$ for $n, m > 0$.

3. $h^m O(h^n) = O(h^{n+m})$ for $n > 0$ and $n + m > 0$.

It follows from Eq. 8.4 that

$$|f(x_0 + h) - T_n(h)| = \left| \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\xi) \right| \tag{8.9}$$

where T_n is the nth Taylor polynomial:

$$T_n(h) = f(x_0) + h f'(x_0) + \frac{h^2}{2!} f''(x_0) + \cdots + \frac{h^n}{n!} f^{(n)}(x_0). \tag{8.10}$$

From our assumption that $f \in C^{(n+1)}(I)$ and $x, x_0 \in I$, it follows that there exists a closed interval $[a, b]$ such that $[a, b] \subset I$ and $x, x_0 \in [a, b]$. Therefore, $f^{(n+1)}(x)$ attains its maximum and minimum values in $[a, b]$, so that $|f^{(n+1)}(x)| \leq M$ for some M. Thus, we have

$$|f(x_0 + h) - T_n(h)| \leq \left| \frac{h^{n+1}}{(n+1)!} M \right| = \frac{M}{(n+1)!} |h^{n+1}|, \tag{8.11}$$

which means that

$$f(x_0 + h) - T_n(h) = O\left(h^{n+1}\right) \tag{8.12}$$

or, equivalently,

$$f(x_0 + h) = T_n(h) + O\left(h^{n+1}\right). \tag{8.13}$$

8.1 Two-point forward and backward formulas for f'

Let us derive the formula for the derivative based on points x_0 and $x_0 + h$ ($h > 0$). If we put $n = 1$ in Eq. 8.4, we obtain

$$f(x_0 + h) = f(x_0) + h f'(x_0) + \frac{h^2}{2!} f''(\xi_1) \tag{8.14}$$

where $x_0 < \xi_1 < x_0 + h$. Solving this for $f'(x_0)$, we find that

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{h}{2!} f''(\xi_1). \tag{8.15}$$

This is called the *forward-difference formula* for $f'(x_0)$. Similarly, one can obtain the *backward-difference formula* for $f'(x_0)$[1]:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \frac{h}{2!} f''(\xi_2) \tag{8.16}$$

where $x_0 - h < \xi_2 < x_0$ ($h > 0$).

---

[1]In fact, it can be obtained from Eq. 8.15 simply by changing h to $-h$.

8.2 Three-point formulas for f'

Now let us derive an approximation formula for $f'(x_0)$ based on points $x_0 - h$, x_0 and $x_0 + h$ ($h > 0$). The Taylor formula Eq. 8.4 with $n = 2$ implies that

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f'''(\xi^+), \quad x_0 < \xi^+ < x_0 + h,$$

$$f(x_0 - h) = f(x_0) - hf'(x_0) + \frac{h^2}{2!}f''(x_0) - \frac{h^3}{3!}f'''(\xi^-), \quad x_0 - h < \xi^- < x_0. \tag{8.17}$$

Subtracting the second equation from the first one, we obtain

$$f(x_0 + h) - f(x_0 - h) = 2hf'(x_0) + \frac{h^3}{3!}\left(f'''(\xi^+) + f'''(\xi^-)\right). \tag{8.18}$$

Solving this for $f'(x_0)$ yields

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{3!}\frac{1}{2}\left(f'''(\xi^+) + f'''(\xi^-)\right). \tag{8.19}$$

Since f''' is continuous, by the Intermediate Value Theorem there exists ξ between ξ^- and ξ^+ such that

$$f'''(\xi) = \frac{1}{2}\left(f'''(\xi^+) + f'''(\xi^-)\right). \tag{8.20}$$

Therefore,

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{6}f'''(\xi) \tag{8.21}$$

where $x_0 - h < \xi < x_0 + h$. This is the *central difference* formula for $f'(x_0)$. Note that the error of the central difference approximation is $O(h^2)$ as $h \to 0$. Thus, we have a second order approximation for $f'(x_0)$.

There are two more three-point formulas for $f'(x_0)$:

$$f'(x_0) = \frac{1}{2h}\left[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)\right] + O(h^2)$$

$$f'(x_0) = \frac{1}{2h}\left[f(x_0 - 2h) - 4f(x_0 - h) + 3f(x_0)\right] + O(h^2) \tag{8.22}$$

The first formula uses points x_0, $x_0 + h$ and $x_0 + 2h$ and is called the *three-point forward difference formula* for $f'(x_0)$. The second formula is called the *three-point backward difference formula* and uses points $x_0 - 2h$, $x_0 - h$ and x_0. Note that the second equation can be obtained from the first by simply replacing h with $-h$, so, in fact, these two represent only one formula.

Example 8.1. Prove Eq. 8.22 assuming that $f \in C^3(I)$ where I is some open interval containing x_0.

Solution. First we choose a sufficiently small* h, so that $[x_0, x_0 + 2h] \subset I$. Then $f'''(x)$ is bounded for all $x \in [x_0, x_0 + 2h]$ and we can write

$$f(x_0 + h) = f(x_0) + h\,f'(x_0) + \frac{h^2}{2}\,f''(x_0) + O(h^3),$$

$$f(x_0 + 2h) = f(x_0) + 2h\,f'(x_0) + \frac{(2h)^2}{2}\,f''(x_0) + O(h^3) \tag{8.23}$$

$$= f(x_0) + 2h\,f'(x_0) + 2h^2\,f''(x_0) + O(h^3).$$

These and Eq. 8.22 yield

$$E = f'(x_0) - \frac{1}{2h}\left[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)\right]$$

$$= f'(x_0) - \frac{1}{2h}\left[-3f(x_0) + 4\left(f(x_0) + h\,f'(x_0) + \frac{h^2}{2}f''(x_0)\right)\right.$$

$$\left. - f(x_0) - 2hf'(x_0) - 2h^2 f''(x_0) + O(h^3)\right] \tag{8.24}$$

$$= O(h^2).$$

8.3 Higher derivatives

Taylor series can also be used to derive formulas for approximating higher derivatives of a function given at a finite set of points.

Let us consider an example. First, we expand f in a third Taylor polynomial about x_0 and evaluate f at $x = x_0 - h$ and $x = x_0 + h$. Then

$$
\begin{aligned}
f(x_0 + h) &= f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + \frac{1}{6}f'''(x_0)h^3 + \frac{1}{24}f^{(4)}(\xi^+)h^4, \\
f(x_0 - h) &= f(x_0) - f'(x_0)h + \frac{1}{2}f''(x_0)h^2 - \frac{1}{6}f'''(x_0)h^3 + \frac{1}{24}f^{(4)}(\xi^-)h^4,
\end{aligned}
\tag{8.25}
$$

where $x_0 - h < \xi^- < x_0 < \xi^+ < x_0 + h$. Adding these equations, we obtain

$$
f''(x_0) = \frac{1}{h^2}[f(x_0 - h) - 2f(x_0) + f(x_0 + h)] - \frac{h^2}{24}[f^{(4)}(\xi^+) + f^{(4)}(\xi^-)].
\tag{8.26}
$$

Assuming that $f^{(4)}$ is continuous on $[x_0 - h, x_0 + h]$, we can rewrite this equation in a simpler form. Since $[f^{(4)}(\xi^+) + f^{(4)}(\xi^-)]/2$ is between $f^{(4)}(\xi^+)$ and $f^{(4)}(\xi^-)$, the Intermediate Value theorem implies that there exists a number ξ between ξ^+ and ξ^- such that

$$
f^{(4)}(\xi) = \frac{1}{2}[f^{(4)}(\xi^+) + f^{(4)}(\xi^-)].
\tag{8.27}
$$

Therefore,

$$
f''(x_0) = \frac{1}{h^2}[f(x_0 - h) - 2f(x_0) + f(x_0 + h)] - \frac{h^2}{12}f^{(4)}(\xi).
\tag{8.28}
$$

where $x_0 - h < \xi < x_0 + h$. This is called the *central difference formula* for $f''(x_0)$ and it uses values of f at three points. Its truncation error is $O(h^2)$. This is one of the most popular finite difference formulas in Numerical Analysis.

Finite difference formulas for higher derivatives can be derived in a similar manner.

8.4 The effect of Roundoff Errors

Consider the central difference formula:

$$
f'(x_0) = \frac{1}{2h}[f(x_0 + h) - f(x_0 - h)] - \frac{h^2}{6}f^{(3)}(\xi).
\tag{8.29}
$$

Suppose that the values of $f(x_0 - h)$ and $f(x_0 + h)$ are computed with roundoff errors e^- and e^+, respectively, i.e.

$$
f(x_0 - h) = f^- + e^- \quad \text{and} \quad f(x_0 + h) = f^+ + e^+.
\tag{8.30}
$$

Here $\tilde{f}^{\pm}$ are the computed valued. Substitution of these in the central difference formula yields

$$
f'(x_0) = \frac{1}{2h}[f^+ + e^+ - f^- - e^-] - \frac{h^2}{6}f^{(3)}(\xi).
\tag{8.31}
$$

The total error in the approximation is

$$
f'(x_0) - \frac{f^+ - f^-}{2h} = \frac{e^+ - e^-}{2h} - \frac{h^2}{6}f^{(3)}(\xi).
\tag{8.32}
$$

The total error has a part due to roundoff error and a part due to truncation error. Suppose that the roundoff errors are bounded by some number $\varepsilon > 0$ (this is always true in practice), i.e.

$$
|e^{\pm}| \leq \varepsilon,
\tag{8.33}
$$

and that the third derivative of f is bounded by $M > 0$ for all $x \in [x_0 - h, x_0 + h]$. Then we have

$$\left| f'(x_0) - \frac{f^+ - f^-}{2h} \right| \leq \frac{\varepsilon}{h} + \frac{h^2}{6} M. \tag{8.34}$$

One can see that, to reduce the truncation error we must reduce h. But as h is reduced, the roundoff error ε/h grows.

To determine the optimal value of h (for which the total error is the smallest one), we consider the function

$$E(h) = \frac{\varepsilon}{h} + \frac{h^2}{6} M. \tag{8.35}$$

which is the upper bound for the total error as a function of h. Since $E'(h) = -\varepsilon/h^2 + hM/3 = 0$ at $h = h^* = (3\varepsilon/M)^{1/3}$, the function $E(h)$ attains its minimum value at

$$h = \left(\frac{3\varepsilon}{M} \right)^{1/3}, \tag{8.36}$$

and this is the optimal value of h. The corresponding minimum error is

$$E_{min} = E(h^*) = \frac{1}{2} \left(9M\varepsilon^2 \right)^{1/3}. \tag{8.37}$$

Unfortunately, in practice, we cannot compute an optimal h to use in approximating the derivative, because usually we do not know the third derivative of the function. But we must be aware that reducing the step size will not always improve the approximation.

Similar analysis can be done for other finite difference formulas for the derivative, and in all cases it leads to similar conclusions.

9 A direct method for solving tridiagonal linear systems

Consider the following system of linear equations

$$A\mathbf{x} = \mathbf{F}, \tag{9.1}$$

where $\mathbf{F} \in \mathbb{R}^n$ is a given vector, $\mathbf{x} \in \mathbb{R}^n$ is the vector of unknowns and A is a given $n \times n$ tridiagonal matrix, i.e.

$$A = \begin{pmatrix}
D_1 & U_1 & 0 & \cdots & \cdots & \cdots & 0 \\
L_2 & D_2 & U_2 & \ddots & & & \vdots \\
0 & L_3 & D_3 & U_3 & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & \ddots & \ddots & D_{n-1} & U_{n-1} \\
0 & \cdots & \cdots & \cdots & 0 & L_n & D_n
\end{pmatrix}. \tag{9.2}$$

A system like this has appeared when we discussed spline interpolation (see Eq. 6.70). Similar linear systems also arise in finite-difference methods for solving differential equations. Of course, the solution of this system can be approximated using iterative techniques such as Jacobi or Gauss-Seidel methods. However, there are much more efficient direct methods for solving such systems. All direct methods are equivalent to Gaussian elimination applied to the above trigiagonal system. One of these, called the double-sweep method, is described below.

The above system of linear equations can be written as

$$\begin{aligned}
D_1 x_i + U_1 x_2 &= F_1, \\
L_i x_{i-1} + D_i x_i + U_i x_{i+1} &= F_i \quad \text{for} \quad i = 2, \dots, n-1, \\
L_n x_{n-1} + D_n x_n &= F_n.
\end{aligned} \tag{9.3}$$

It is convenient to introduce

$$x_0 = 0 \quad \text{and} \quad x_{n+1} = 0. \tag{9.4}$$

Then Eq. 9.3 can be rewritten as

$$L_i x_{i-1} + D_i x_i + U_i x_{i+1} = F_i \quad \text{for} \quad i = 1, \dots, n. \tag{9.5}$$

To solve Eq. 9.5, we will seek α_i and β_i such that

$$x_{i-1} = \alpha_i x_i + \beta_i \quad \text{for} \quad i = 1, 2, \dots, n+1. \tag{9.6}$$

Substitution of Eq. 9.6 into Eq. 9.5 yields

$$(\alpha_i L_i + D_i) x_i + U_i x_{i+1} + \beta_i L_i - F_i = 0 \quad \text{for} \quad i = 1, \dots, n. \tag{9.7}$$

From Eq. 9.6, we also have

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1} \quad \text{for} \quad i = 0, 1, \dots, n. \tag{9.8}$$

Substituting this into Eq. 9.7, we find that

$$[(\alpha_i L_i + D_i)\alpha_{i+1} + U_i] x_{i+1} + [(\alpha_i L_i + D_i)\beta_{i+1} + \beta_i L_i - F_i] = 0 \quad \text{for} \quad i = 1, \dots, n. \tag{9.9}$$

The last equation is satisfied if the two expressions in the square brackets are both zero. This leads to the following recursive formulae:

$$\alpha_{i+1} = -\frac{U_i}{D_i + \alpha_i L_i}, \quad \beta_{i+1} = -\frac{\beta_i L_i - F_i}{D_i + \alpha_i L_i}, \quad \text{for} \quad i = 1, \dots, n. \tag{9.10}$$

Now if α_1 and β_1 are known, then α_i and β_i for $i = 2, 3, \dots, n+1$ can be computed from Eq. 9.10. α_1 and β_1 can be determined from Eq. 9.6 and the fact that $x_0 = 0$. Indeed,

$$x_0 = \alpha_1 x_1 + \beta_1 \quad \text{and} \quad x_0 = 0 \quad \Rightarrow \quad \alpha_1 x_1 + \beta_1 = 0. \tag{9.11}$$

To satisfy the last equation, we choose $\alpha_1 = 0$ and $\beta_1 = 0$. Once we know all α_i and β_i, we compute $x_n, x_{n-1}, \dots, x_1$ using formula Eq. 9.6.

Formulae Eq. 9.6 and Eq. 9.10 will work provided that the coefficients L_i, U_i and D_i are such that $D_i + \alpha_i L_i \neq 0$ for $i = 1, \dots, n$. For tridiagonal systems that arise in finite-difference methods for differential equations, the coefficients L_i, U_i and D_i usually satisfy the inequalities

$$L_i, U_i > 0, \quad D_i < 0, \quad -D_i \geq L_i + U_i. \tag{9.12}$$

It can be shown that these restrictions on L_i, U_i and D_i are sufficient for the double-sweep method to work.

The following function implements this method in Python:

Let us look at a simple example of a tri-diagonal system:

```
Solution: [-1. -1. -1. -1. -1.]
```

Let's check that x does indeed satisfy Eq. 9.1. First we need to construct the full matrix A.

```
array([[-2.,  1.,  0.,  0.,  0.],
       [ 1., -2.,  1.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  1., -2.,  1.],
       [ 0.,  0.,  0.,  1., -2.]])
```

Now we can use Python's matrix multiplication opeator @ to calculate $A\mathbf{x}$:

```
array([ 1.00000000e+00, -2.22044605e-16,  0.00000000e+00,  0.00000000e+00,
        1.00000000e+00])
```

This indeed agrees with $\mathbf{F}$ up to rounding errors.

10 Initial Value Problems

10.1 Introduction

The topic of this and the next chapter is to find *approximate solutions* to *ordinary differential equations*.

Let us briefly recall what an ordinary differential equation (ODE) is. A rather arbitrarily chosen example for an ODE (here, of second order) is

$$y''(x) + 4y'(x) + \sqrt[3]{y(x)} + \cos(x) = 0. \tag{10.1}$$

Equations like this are normally satisfied by many functions $y(x)$: the problem has many solutions. In order to specify a uniquely solvable problem, one needs to fix *initial values*, i.e., the value of y and its first derivative at some point, say, at $x = 0$:

$$y''(x) + 4y'(x) + \sqrt[3]{y(x)} + \cos(x) = 0, \quad y(0) = 1, \ y'(0) = -2. \tag{10.2}$$

This is a so-called *initial-value problem* (IVP). Another variant is to specify the value of $y(x)$, but not of its derivative, at two different points:

$$y''(x) + 4y'(x) + \sqrt[3]{y(x)} + \cos(x) = 0, \quad y(0) = 2, \ y(1) = 1. \tag{10.3}$$

This is called a *boundary value problem* (BVP).

Both IVPs and BVPs have a unique solution (under certain mathematical conditions). However, while one can show on abstract grounds that these solutions exist, it is often not practicable to find an explicit expression for them. The best one can hope for is to approximate the solution numerically. This is exactly our topic: to find approximation algorithms for the solutions of IVPs (this chapter) and BVPs (in Chapter 11).

10.2 Euler's Method

For studying initial value problems (IVPs) for ordinary differential equations, let us start with the simplest known approximation scheme: *Euler's method*. We will focus on direct computations here, and defer more in-depth discussions of the mathematical foundations to later.

We consider an IVP for a first-order equation, of the form

$$y'(x) = f(x, y(x)), \quad a \le x \le b, \quad y(a) = \alpha. \tag{10.4}$$

Here a, b and α are some real numbers, and f is a function $\mathbb{R}^2 \to \mathbb{R}$. (For a concrete example, see Example 10.1 below.) Under certain conditions on f — to be discussed in Section 10.3 — one knows that the problem has a unique solution; that is, that there is exactly one function $y(x)$ defined on $[a, b]$ which satisfies Eq. 10.4. We often call $y(x)$ the *exact solution* of the IVP. However, it is often difficult or even impossible to find an explicit expression for $y(x)$. Our aim here is to approximate the solution numerically.
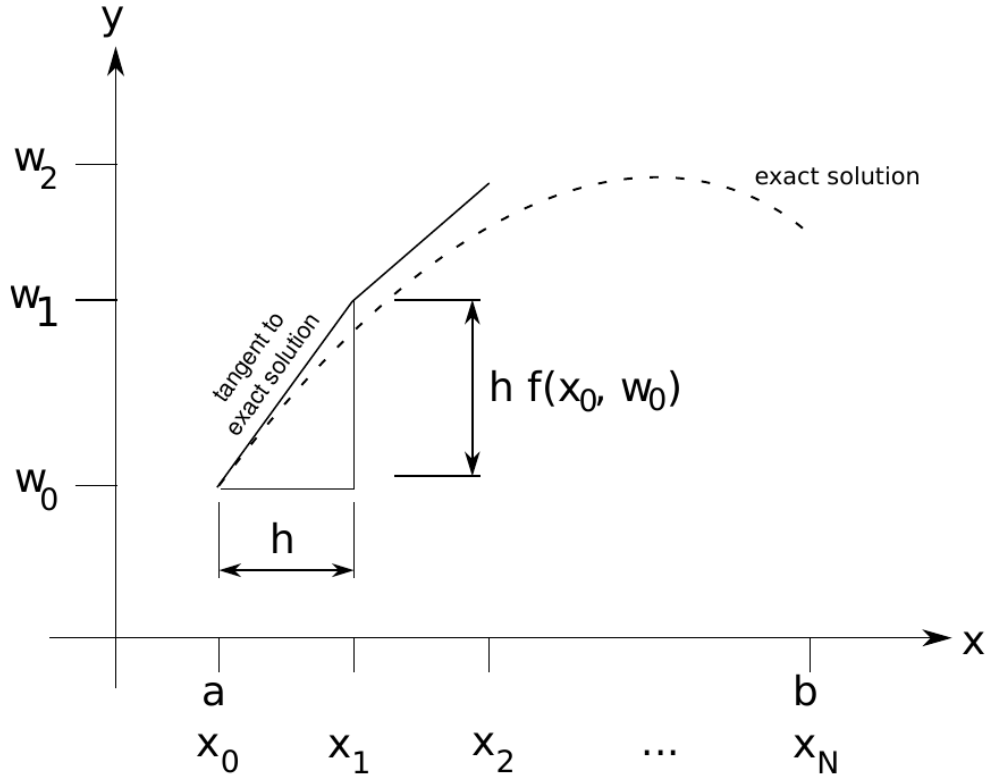
Figure 10.1: Euler's method

10.2.1 The method

The idea behind Euler's method is as follows (see Figure 10.1 for a visualization): We pick $N \in \mathbb{N}$, and divide the interval $[a, b]$ into N subintervals of equal length, $h = (b - a)/N$. The end points of these intervals are

$$x_i = a + ih, \quad i = 0, \dots, N. \tag{10.5}$$

Terminology: N is called the **number of steps** and h is called the **step size**. The points $x_0, \dots, x_N \in [a, b]$ are referred to as **mesh points**.

We will approximate the exact solution y only at the mesh points x_i; that is, we are looking for numbers $w_0, \dots, w_N$ such that $y(x_i) \approx w_i$.

The first approximation w_0 is easy to choose: We know that y satisfies the initial condition, $y(x_0) = y(a) = \alpha$. Thus we set $w_0 := \alpha$; this is even exact.

For finding $w_1 \approx y(x_1)$, we use linear approximation:

$$
\begin{aligned}
y(x_1) = y(x_0 + h) &\approx y(x_0) + hy'(x_0) \\
&\stackrel{(*)}{=} y(x_0) + hf(x_0, y(x_0)) = w_0 + hf(x_0, w_0).
\end{aligned}
\tag{10.6}
$$

For the equality $(*)$, we have used that y satisfies the ODE Eq. 10.4.

In the same way, we can find an approximation for $y(x_2)$:

$$
\begin{aligned}
y(x_2) = y(x_1 + h) &\approx y(x_1) + hy'(x_1) \\
&\stackrel{(*)}{=} y(x_1) + hf(x_1, y(x_1)) \stackrel{(\circ)}{\approx} w_1 + hf(x_1, w_1).
\end{aligned}
\tag{10.7}
$$

Our approximation value is $w_2 := w_1 + hf(x_1, w_1)$. Note that we have used another approximation step $(\circ)$, using that $y(x_1) \approx w_1$ and that f is sufficiently smooth; we will come back to this point in Section 10.2.2. We can continue the scheme for w_3, w_4, etc.:

$$w_0 := \alpha, \tag{10.8}$$

$$w_1 := w_0 + hf(x_0, w_0), \tag{10.9}$$

$$w_2 := w_1 + hf(x_1, w_1), \tag{10.10}$$

$$\vdots$$

$$w_{i+1} := w_i + hf(x_i, w_i). \tag{10.11}$$

So the w_i are defined recursively. Eq. 10.11 is called the **difference equation** of Euler's method.

Example 10.1. Let us consider the IVP

$$y'(x) = y(x) - x^2 + 1, \quad 0 \le x \le 1, \quad y(0) = \frac{1}{2}. \tag{10.12}$$

In this case, the exact solution can be found using an integrating factor. It is

$$y(x) = (x+1)^2 - \frac{1}{2}e^x. \tag{10.13}$$

This will allow us to compare the approximation with the exact solution. It is generally a good idea to test one's numerical method on an example where one already knows the exact solution.

We choose $N = 10$ steps, i.e., $h = 1/10$. Here we shall only compute the first two steps. By Eq. 10.8, we certainly have

$$x_0 = 0, \quad w_0 = \frac{1}{2}. \tag{10.14}$$

This allows us to compute the approximation w_1 at the mesh point $x_1 = 1/10$, namely, by Eq. 10.9,

$$\begin{aligned} w_1 &= w_0 + hf(x_0, w_0) = w_0 + h(w_0 - x_0^2 + 1) \\ &= \frac{1}{2} + \frac{1}{10}\left(\frac{1}{2} - 0 + 1\right) = \frac{1}{2} + \frac{3}{20} \\ &= \frac{13}{20}. \end{aligned} \tag{10.15}$$

Continuing the iteration to $x_2 = 2/10$, we have by Eq. 10.10 that

$$\begin{aligned} w_2 &= w_1 + hf(x_1, w_1) = w_1 + h(w_1 - x_1^2 + 1) \\ &= \frac{13}{20} + \frac{1}{10}\left(\frac{13}{20} - \left(\frac{1}{10}\right)^2 + 1\right) = \frac{13}{20} + \frac{1}{10} \cdot \frac{174}{100} \\ &= \frac{407}{500}. \end{aligned} \tag{10.16}$$

Continuing further, we would obtain the values shown in Table 10.1. The values of the exact solution Eq. 10.13, evaluated to 9 decimals, are added for comparison. The last column, the error bound, will be discussed shortly.

Table 10.1: Approximation values and errors for Euler's Method

i	x_i	w_i	$y(x_i)$	$|y(x_i) - w_i|$	error bound
0	0	0.5	0.5	0	0
1	0.1	0.65	0.657414541	0.007414541	0.0078878189
2	0.2	0.814	0.829298621	0.015298621	0.0166052069
3	0.3	0.9914	1.015070596	0.023670596	0.0262394106
4	0.4	1.18154	1.214087651	0.032547651	0.0368868524
5	0.5	1.383694	1.425639364	0.041945364	0.0486540953
6	0.6	1.5970634	1.648940600	0.051877200	0.0616589100
7	0.7	1.82076974	1.883123646	0.062353906	0.0760314530
8	0.8	2.053846714	2.127229536	0.073382822	0.0919155696

i	x_i	w_i	$y(x_i)$	$|y(x_i) - w_i|$	error bound
9	0.9	2.295231385	2.380198444	0.084967059	0.1094702333
10	1.0	2.543754524	2.640859086	0.097104562	0.1288711371

10.2.2 Error bounds

Denote the error at step i of Euler's method by

$$\varepsilon_i = y(x_i) - w_i. \tag{10.17}$$

Clearly $\varepsilon_0 = 0$. We can represent ε_1 as follows: assuming the second derivative y'' exists on (a, b), we can use Taylor's Theorem to give

$$y(x_1) = y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2 y''(\xi_1)}{2} \tag{10.18}$$

for some $\xi_1 \in (x_0, x_1)$. Now, y is a solution to the IVP so $y(x_0) = y(a) = \alpha = w_0$ and $y'(x_0) = f(x_0, y(x_0)) = f(x_0, \alpha) = f(x_0, w_0)$ and, by definition of w_1,

$$y(x_0) + hy'(x_0) = w_0 + hf(x_0, w_0) = w_1. \tag{10.19}$$

Substituting this gives

$$y(x_1) = w_1 + \frac{h^2 y''(\xi_1)}{2}. \tag{10.20}$$

So the error after one step can be written as

$$\varepsilon_1 = y(x_1) - w_1 = \frac{h^2 y''(\xi_1)}{2}. \tag{10.21}$$

Although we do not know what $y''(\xi_1)$ is, this does show how the error depends on the step length: it behaves like a multiple of h^2.

We can now make an optimistic guess: if every step contributes a multiple of h^2, then the total error at the end, after N steps, will be some multiple of $Nh^2 = (Nh)h = (b-a)h$. As $h \to 0$, this tends to zero (whatever the unknown constants are) so the approximate solution converges to the exact solution. It turns out that this is basically correct, although much more careful reasoning is needed, as we shall see when we look at the error after the second step.

We can try to analyse the second step in the same way as the first step: use Taylor's Theorem to write

$$y(x_2) = y(x_1 + h) = y(x_1) + hy'(x_1) + \frac{h^2 y''(\xi_2)}{2} \tag{10.22}$$

and use the fact that y is a solution of the DE to substitute $y'(x_1) = f(x_1, y(x_1))$:

$$y(x_2) = y(x_1) + hf(x_1, y(x_1)) + \frac{h^2 y''(\xi_2)}{2}. \tag{10.23}$$

At this point, things look different: in the first step, we had $y(x_0) = y(a) = \alpha$, but here we do not have $y(x_1) = w_1$: the first step starts at $(x_0, w_0) = (a, \alpha)$, which lies exactly on the solution curve, but the second step starts at (x_1, w_1), which does not lie on the solution curve. However, w_1 is an approximation to $y(x_1)$, and we have an expression for the error, namely ε_1. Substituting $y(x_1) = w_1 + \varepsilon_1$ gives

$$y(x_2) = w_1 + \varepsilon_1 + hf(x_1, w_1 + \varepsilon_1) + \frac{h^2 y''(\xi_2)}{2}. \tag{10.24}$$

Now, $w_1 + hf(x_1, w_1 + \varepsilon_1)$ is very similar to the formula for w_2: the only difference is that it has $w_1 + \varepsilon_1$, instead of w_1. As we did for for $y(x_1)$, we can think of this as being an approximation plus an error:

$$\underbrace{f(x_1, w_1 + \varepsilon_1)}_{\text{exact}} = \underbrace{f(x_1, w_1)}_{\text{approx}} + \underbrace{[f(x_1, w_1 + \varepsilon_1) - f(x_1, w_1)]}_{\text{error}} \tag{10.25}$$

leading to

$$y(x_2) = \varepsilon_1 + \underbrace{w_1 + hf(x_1, w_1)}_{=w_2} + h[f(x_1, w_1 + \varepsilon_1) - f(x_1, w_1)]\frac{h^2 y''(\xi_2)}{2}, \tag{10.26}$$

which, as intended, forces w_2 to appear:

$$y(x_2) = \varepsilon_1 + w_2 + h[f(x_1, w_1 + \varepsilon_1) - f(x_1, w_1)] + \frac{h^2 y''(\xi_2)}{2}. \tag{10.27}$$

Subtracting w_2 from both sides gives

$$\varepsilon_2 = \varepsilon_1 + h[f(x_1, w_1 + \varepsilon_1) - f(x_1, w_1)] + \frac{h^2 y''(\xi_2)}{2}. \tag{10.28}$$

This describes the error at step 2 as the sum of three terms which can be thought of as:

- ε_1, the error carried forward from step 1;

- $h[f(x_1, w_1 + \varepsilon_1) - f(x_1, w_1)]$, the error caused by starting at (x_1, w_1), which is not on the solution curve;

- $h^2 y''(\xi_2)/2$, the error built into Euler's method by the approximation $y(x + h) \approx y(x) + hy'(x)$.

This analysis works for every step: we have

$$\varepsilon_{i+1} = \varepsilon_i + h[f(x_i, w_i + \varepsilon_i) - f(x_i, w_i)] + \frac{h^2 y''(\xi_{i+1})}{2}, \tag{10.29}$$

where $\xi_{i+1} \in (x_i, x_{i+1})$. This is a recurrence relation for ε_i, and $\varepsilon_0 = 0$. Now, we need to ask how large the different contributions can be. For the truncation error inherent in Euler's method, we simply assume that there is a constant M such that $|y''(x)| \leq M$ for all $x \in [a, b]$. For the error associated with starting away from the solution curve, we use Taylor's Theorem yet again: assuming f is differentiable in the second variable, we can write

$$f(x_i, w_i + \varepsilon_i) - f(x_i, w_i) = \varepsilon_i \frac{\partial}{\partial z} f(x_i, z)\Big|_{z=\eta_i} \tag{10.30}$$

for some $\eta_i \in (w_i, w_i + \varepsilon_i)$. We now make the assumption that there is a constant L such that

$$\left|\frac{\partial}{\partial z} f(x, z)\right| \leq L \tag{10.31}$$

for all $x \in [a, b]$ and all z. This leads to

$$|h[f(x_i, w_i + \varepsilon_i) - f(x_i, w_i)]| = h|f(x_i, w_i + \varepsilon_i) - f(x_i, w_i)| \leq hL|\varepsilon_i|. \tag{10.32}$$

Using the triangle inequality on the formula for ε_{i+1}, we have

$$\begin{aligned} |\varepsilon_{i+1}| &\leq |\varepsilon_i| + Lh|\varepsilon_i| + \frac{Mh^2}{2} = (1 + Lh)|\varepsilon_i| + \frac{Mh^2}{2} \\ &= (1 + Lh)|\varepsilon_i| + h\tau(h), \end{aligned} \tag{10.33}$$

where $\tau(h) = Mh/2$ (this is just a convenient abbreviation, which makes the final answer look tidy). We can apply this estimate repeatedly, starting off with $\varepsilon_0 = 0$, to find an estimate for the error after any number of steps.

$$\begin{aligned} |\varepsilon_0| &= 0 \\ |\varepsilon_1| &\leq h\tau(h) \\ |\varepsilon_2| &\leq (1 + Lh)|\varepsilon_1| + h\tau(h) \\ &\leq [(1 + Lh) + 1]h\tau(h) \\ |\varepsilon_3| &\leq (1 + Lh)|\varepsilon_2| + h\tau(h) \\ &\leq [(1 + Lh)^2 + (1 + Lh) + 1]h\tau(h) \\ &\vdots \\ |\varepsilon_n| &\leq [(1 + Lh)^{n-1} + (1 + Lh)^{n-2} + \cdots + 1]h\tau(h). \end{aligned} \tag{10.34}$$

This final formula can be proved by induction. This is a geometric sum, so we can use the standard formula to give for $1 \leq n \leq N$

$$|\varepsilon_n| \leq h\tau(h) \sum_{i=0}^{n-1} (1 + Lh)^i = h\tau(h) \frac{(1 + Lh)^n - 1}{Lh} \tag{10.35}$$

and the h terms in the numerator and denominator cancel (this is why the abbreviated the Taylor's Theorem estimate as $h\tau(h)$). The dependency on n is awkward here, so we use the fact that $1 + x \leq e^x$ for all x to replace $1 + Lh$ by e^{Lh}:

$$|\varepsilon_n| \leq \frac{\tau(h)}{L} (e^{Lhn} - 1)'' \tag{10.36}$$

Now, hn is the distance from $x_0 = a$ to x_n, so

$$|\varepsilon_n| \leq \frac{\tau(h)}{L} (e^{L(x_n - a)} - 1). \tag{10.37}$$

The RHS increases exponentially as x_n increases, with the maximum value at $x_n = b$, so we can finally conclude that

$$|\varepsilon_N| \leq \frac{\tau(h)}{L} (e^{L(x_n - a)} - 1) \leq \frac{\tau(h)}{L} (e^{L(b - a)} - 1). \tag{10.38}$$

In all cases, we see that the error is bounded above by a multiple of $\tau(h)$, which is itself a multiple of h (because $\tau(h) = Mh/2$). The multiplier depends on the width of the interval on which we solve the equation but, crucially, if we fix the interval then the multiplier does not change as we decrease the step size. The error therefore tends to zero as $h \to 0$ (equivalently, as $N \to \infty$), so the approximate solution converges to the exact solution.

Example 10.2 (Example for error bounds)**.** Let us reconsider Example 10.1, with $f(x,y) = y - x^2 + 1$, and compute the error bounds. We have $\partial f/\partial y = 1$, so we have $L = 1$ – see Eq. 10.31. For the constant M, we use the fact that we know the exact solution: $y(x) = (x + 1)^2 - e^x/2$. This gives us $y''(x) = 2 - e^x/2$, which, on the interval $[0, 1]$, is bounded by $3/2$ (its value at 0). So $M = 3/2$. Inserting, this gives us

$$|y(x_i) - w_i| \leq \frac{3}{40} (e^{(x_i - a)} - 1). \tag{10.39}$$

These bounds are included in Table 10.1. You can see that the actual errors are indeed below the bounds, but not very much, particularly for small x.

Of course, using the exact solution in the error bounds can be considered "cheating" to some extent, since the exact solution of the ODE is in general unknown. There are methods for obtaining estimates for the constant M even if $y(x)$ is not explicitly known, but we will not discuss them at this point.

With the Euler method, we have an approximation algorithm for the generic initial value problem Eq. 10.4, which works for any sufficiently smooth f. We have found an explicit estimate Eq. 10.38 for the approximation error, which in particular shows that the approximation values converge to the exact solution, $w_i \to y(x_i)$ as $h \to 0$.

However, the Euler method as presented here has two main shortcomings:

First, the convergence is rather slow - only of order $O(h)$. One would need to choose the step size h very small in order to arrive at a useful approximation. Decreasing h means, first of all, an increase in computation time. But also, small values of h make the difference equation Eq. 10.11 prone to roundoff errors; limits in floating point precision limit the usable range for h. (For more details on the influence of roundoff errors on the approximation result, see for example (Burden and Faires 2010 Theorem 5.10).)

Second, we have formulated the method for a *first-order* ODE. Most applications, however, use higher-order ODEs (usually second-order), systems of first-order ODEs, or indeed a combination of both. Our approximation methods needs to be generalized to these cases in order to be useful in practice.

Next we will introduce the generalizations needed for handling higher-order ODEs and systems of ODEs. In most textbooks, you will find this generalization only in later chapters, for example in Sec. 5.9 of (Burden and Faires 2010), or not at all. However, here I take the viewpoint that, while treating systems of ODEs is not really difficult, it is so important that it should be introduced right in the beginning! As we shall see, this can be boiled down to almost no more than a little change in notation.

10.2.3 Systems of ODEs

A generic initial value problem for a system of m first-order ODEs would look as follows:

$$
\begin{aligned}
y_1{}'(x) &= f_1(x, y_1(x), \dots, y_m(x)), \\
&\;\;\vdots \\
y_m{}'(x) &= f_m(x, y_1(x), \dots, y_m(x)),
\end{aligned}
\tag{10.40}
$$

for x in some interval $[a, b]$, with initial values

$$
y_1(a) = \alpha_1, \dots, y_m(a) = \alpha_m.
\tag{10.41}
$$

Here a, b, $\alpha_1, \dots, \alpha_m$ are given constants, and $f_1, \dots, f_m$ are functions from $\mathbb{R} \times \mathbb{R}^m$ to $\mathbb{R}$. The exact solution of the system would consist of functions $y_1, \dots, y_m : [a, b] \to \mathbb{R}$.

The idea of handling these ODE systems mainly involves rewriting them in a convenient way. To that end, let us introduce the vectors

$$
\begin{aligned}
\alpha &= (\alpha_1, \dots, \alpha_m), \\
\mathbf{y} &= (y_1, \dots, y_m),
\end{aligned}
\tag{10.42}
$$

and the vector-valued function $\mathbf{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ given by

$$
\mathbf{f}(x, \mathbf{y}) = (f_1(x, y_1, \dots, y_m), \dots, f_m(x, y_1, \dots, y_m)).
\tag{10.43}
$$

With these, the IVP Eq. 10.40 for the ODE system reads

$$
\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \quad a \le x \le b, \quad \mathbf{y}(a) = \alpha.
\tag{10.44}
$$

Note the formal similarity with the analogue Eq. 10.4 for a single ODE! Our exact solution $\mathbf{y}$ is now a function from $[a, b]$ to $\mathbb{R}^m$.

The idea in generalizing our approximation methods to systems of ODEs is based on this formal similarity as well. For example, the difference equation for the "Euler method for ODE systems" reads

$$
\begin{aligned}
\mathbf{w}_0 &:= \alpha, \\
\mathbf{w}_{i+1} &:= \mathbf{w}_i + h \, \mathbf{f}(x_i, \mathbf{w}_i) \quad (i = 0, \dots, N-1).
\end{aligned}
\tag{10.45}
$$

The approximation values $\mathbf{w}_i$ are now vectors in $\mathbb{R}^m$. The error estimates obtained in Section 10.2.2 carry over very directly to the case of ODE systems; more on this will follow in later sections.

10.2.4 Second-order ODEs

In applications, one often meets second-order ODEs. Initial value problems for them can be defined as follows:

$$
y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \; y'(a) = \alpha'.
\tag{10.46}
$$

Note that we need to specify an additional initial value $\alpha' \in \mathbb{R}$ for the first derivative.

Fortunately, these can be rewritten into an equivalent system of first-order ODEs, so that they are — for our purposes — not really different from what was discussed above. Namely, we substitute y and y' with the components of a 2-vector $\mathbf{u}$: We set $u_1 = y$, $u_2 = y'$.

More formally, this works as follows. Given a solution $y(x)$ of Eq. 10.46, we set

$$\mathbf{g}(x, \mathbf{u}) := (u_2, f(x, u_1, u_2)).$$ (10.47)

It is then easy to check that $\mathbf{u}(x)$ is a solution of the IVP

$$\mathbf{u}' = \mathbf{g}(x, \mathbf{u}), \quad a \le x \le b, \quad \mathbf{u}(a) = \beta.$$ (10.48)

On the other hand, given a solution $\mathbf{u}(x)$ of Eq. 10.48, we set $y(x) := u_1(x)$, $\alpha := \beta_1$, $\alpha' := \beta_2$ and obtain a solution of Eq. 10.46. In this sense, Eq. 10.46 and Eq. 10.48 are equivalent; and for the purpose of developing numerical methods, it is sufficient if we consider the first-order system Eq. 10.48.

Example 10.3 (Example of a second-order ODE)**.** Let us illustrate the substitution process in an example. Consider the IVP for a second-order ODE,

$$\begin{aligned} y''(x) &= y'(x)\cos(x) + 2y(x), \quad 0 \le x \le 1, \\ y(0) &= 2,\, y'(0) = -3. \end{aligned}$$ (10.49)

So, $f(x, y, y') = y'\cos(x) + 2y$ in the present case. Using the rules in Eq. 10.47, we obtain an equivalent IVP for a system of two first-order equations:

$$\mathbf{u}'(x) = \begin{pmatrix} u_2(x) \\ u_2(x)\cos(x) + 2u_1(x) \end{pmatrix}, \quad 0 \le x \le 1, \quad \mathbf{u}(0) = \begin{pmatrix} 2 \\ -3 \end{pmatrix}.$$ (10.50)

Once we have found an (approximate) solution for this system, we would set $y(x) := u_1(x)$ and obtain an (approximate) solution of Eq. 10.49.

With similar methods, one can rewrite third-order, fourth-order, etc. ODEs as systems of first-order ODEs. Equally, *systems* of higher-order ODEs can be transformed into systems of first-order ODEs by appropriate substitutions. For example, systems of second-order ODEs are very common in Newtonian Mechanics: N particles moving in three-dimensional space are modelled using a system of $3N$ coupled second-order ODEs, which can be transformed into a system of $6N$ coupled first-order ODEs.

Thus, the most general case of IVP that we need to consider is given by Eq. 10.44. In the following, we will formulate all our approximation methods for this case.

10.3 Fundamentals

We will now take a step back, and revisit the theory of initial value problems for ODEs and their numerical treatment from a more general perspective. In doing so, we will always consider initial value problems for systems of (first-order) ODEs, in the form Eq. 10.44. This means that we will make extensive use of techniques from Vector Calculus.

10.3.1 Vectors and matrices

In order to describe numerical approximations of vectors, we will need a notion of distance between two vectors. Throughout this part of the course, we do not use the Euclidean norm for this purpose, but the *maximum norm* or ℓ_∞ *norm*, which is given by

$$\|\mathbf{v}\|_\infty := \max_{1 \le i \le m} |v_i|.$$ (10.51)

From now on, we will just write $\|\mathbf{v}\|$ instead of $\|\mathbf{v}\|_\infty$.

We also need a corresponding norm for matrices $\mathbf{A} \in \mathcal{M}(m, m)$.

10.3.2 Calculus in several variables

We will also need to work with functions that depend on vectors, and with vector-valued functions. For a review of the techniques of Calculus in several variables, see for example (Weir, Thomas, and Hass 2010 Ch. 14) or (Stewart 1991 Ch. 15). We give a very brief review here, in our notation.

Let f be a function from $\mathbb{R}^m$ to $\mathbb{R}$ (a function of m variables). Instead of derivatives of functions of a single variable, one can consider *partial derivatives* of f, denoted as

$$\frac{\partial f}{\partial x_j}(\mathbf{x}) = \frac{d}{dt} f(x_1, \ldots, x_j + t, \ldots, x_m)\Big|_{t=0}, \tag{10.52}$$

and higher order partial derivatives accordingly. If $\mathbf{f}$ is a vector-valued function, that is, $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^k$, then all partial derivatives are vector-valued as well (each component is differentiated). All first derivatives of such a function can conveniently be combined into an $k \times m$ matrix,

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f^{(k)}}{\partial x_1} & \cdots & \frac{\partial f^{(k)}}{\partial x_m} \end{pmatrix} \tag{10.53}$$

The multi-dimensional chain rule can be expressed quite easily in this formalism: if $\mathbf{f}$ and $\mathbf{g}$ are vector-valued mappings such that $\mathbf{g} \circ \mathbf{f} : \mathbf{x} \mapsto \mathbf{g}(\mathbf{f}(\mathbf{x}))$ is defined (i.e. the range of $\mathbf{f}$ matches the domain of $\mathbf{g}$) then the derivative of $\mathbf{g} \circ \mathbf{f}$ is given by

$$\frac{\partial(\mathbf{g} \circ \mathbf{f})}{\partial \mathbf{x}} = \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{x}}. \tag{10.54}$$

where $\cdot$ represents matrix multiplication. Higher-order derivatives can be treated in a similar, though somewhat more complicated matrix formalism.

Also, a generalization of Taylor's theorem holds for functions of several variables. This is summarized in Appendix A.

10.3.3 ODEs

We now treat initial value problems for ODEs in our vector formalism. In a first reading, it is always a useful exercise to reduce our statements to the case of one ODE ($m = 1$), in which case the computations become more elementary. In this case, we can simply replace $\mathbf{y}$ with a scalar function y, the initial value vector α with a number α, the norm $\|\cdot\|$ with the absolute value $|\cdot|$, and so forth.

Let us first define formally what we mean by a solution of an initial value problem.

Definition 10.1. Let $m \in \mathbb{N}$, $a < b \in \mathbb{R}$, $\alpha \in \mathbb{R}^m$, and $\mathbf{f} : [a, b] \times \mathbb{R}^m \to \mathbb{R}^m$. We say that a function $\mathbf{y} \in \mathcal{C}^1([a, b], \mathbb{R}^m)$ is a **solution of the initial value problem (IVP)**

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad a \leq x \leq b, \quad \mathbf{y}(a) = \alpha \tag{10.55}$$

if for all $x \in [a, b]$,

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)) \quad \text{and} \quad \mathbf{y}(a) = \alpha. \tag{10.56}$$

Our first question is when such an IVP has a unique solution (so that we can reasonably search for a numerical approximation of it). The key condition involved here is the so-called *Lipschitz condition*.

Definition 10.2. We say that $\mathbf{f} : [a, b] \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies a **Lipschitz condition** if there is a constant $L > 0$ such that for all $x \in [a, b]$ and $\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^m$,

$$\|\mathbf{f}(x, \mathbf{y}) - \mathbf{f}(x, \hat{\mathbf{y}})\| \leq L\|\mathbf{y} - \hat{\mathbf{y}}\|. \tag{10.57}$$

The constant L above is called a **Lipschitz constant**.

The Lipschitz condition is in a sense an intermediate concept between continuity and differentiability. Often, we can use a simple criterion to check it — in fact, we already have, in the definition of the constant L in the error analysis of Euler's method in one variable.

Lemma 10.1. *If $\partial\mathbf{f}/\partial\mathbf{y}$ exists and is bounded, then $\mathbf{f}$ satisfies a Lipschitz condition with Lipschitz constant*

$$L = \sup\left\{\left\|\frac{\partial\mathbf{f}}{\partial\mathbf{y}}(x,\mathbf{y})\right\| : x \in [a,b], \mathbf{y} \in \mathbb{R}^m\right\}. \tag{10.58}$$

Note here that $\partial\mathbf{f}/\partial\mathbf{y}$ is an $m \times m$ *matrix*! Again, you may first want to consider the case $m = 1$.

Proof. Given $x \in [a,b]$, $\mathbf{y},\hat{\mathbf{y}} \in \mathbb{R}^m$, and $j \in \{1,\dots,m\}$, consider the function $g : [0,1] \to \mathbb{R}$ defined by $g(t) = f_j((1-t)\hat{\mathbf{y}} + t\mathbf{y})$, so $g(0) = \hat{\mathbf{y}}$ and $g(1) = \mathbf{y}$, and apply the scalar MVT to give $\mathbf{y} - \hat{\mathbf{y}} = g(1) - g(0) = g'(t_0)$ for some $t_0 \in (0,1)$; now, using the chain rule,

$$\begin{aligned}
g'(t) &= \frac{\partial f_j}{\partial\mathbf{y}}(x,(1-t)\hat{\mathbf{y}} + t\mathbf{y}) \cdot \frac{d}{dt}((1-t)\hat{\mathbf{y}} + t\mathbf{y}) \\
&= \frac{\partial f_j}{\partial\mathbf{y}}(x,(1-t)\hat{\mathbf{y}} + t\mathbf{y}) \cdot (\mathbf{y} - \hat{\mathbf{y}}).
\end{aligned} \tag{10.59}$$

Combining these gives us

$$f_j(x,\mathbf{y}) - f_j(x,\hat{\mathbf{y}}) = \frac{\partial f_j}{\partial\mathbf{y}}(x,\eta) \cdot (\mathbf{y} - \hat{\mathbf{y}}) \tag{10.60}$$

where $\eta = (1-t_0)\hat{\mathbf{y}} + t_0\mathbf{y}$. Using the property of the vector norm of the scalar product (see Definition 4.1) we find

$$|f_j(x,\mathbf{y}) - f_j(x,\hat{\mathbf{y}})| \leq \left\|\frac{\partial f_j}{\partial\mathbf{y}}(x,\eta)\right\| \|\mathbf{y} - \hat{\mathbf{y}}\| \leq L\|\mathbf{y} - \hat{\mathbf{y}}\| \tag{10.61}$$

with L as in Eq. 10.58. Taking the maximum over j implies the result. $\qquad\square$

Why do we consider this kind of condition? Because the Lipschitz condition is the essential ingredient for the well-definedness of the initial value problem. Namely, one has:

Theorem 10.1. *Let $\mathbf{f} : [a,b] \times \mathbb{R}^m \to \mathbb{R}^m$ be continuous and satisfy a Lipschitz condition. Then, for any $\alpha \in \mathbb{R}^m$, the initial value problem Eq. 10.55 has a unique solution $\mathbf{y}$.*

This theorem is given here without proof. (It is one of the central theorems in the theory of ordinary differential equations. See (Burden and Faires 2010 Theorem 5.17) for references.)

The Lipschitz condition is not only relevant for the abstract existence of a solution. Recalling Eqs. Eq. 10.30–Eq. 10.31, we see that it also enters our error estimates for numeric approximations. We will analyse this in more detail later.

Let us discuss a few examples for $m = 1$, on the interval $[a,b] = [0,2]$.

Example 10.4 (Unique solution)**.** Let us recall the example Eq. 10.12, with $f(x,y) = y - x^2 + 1$. This function is certainly continuous in both variables, and differentiable in y. We have $\partial f(x,y)/\partial y = 1$; in particular, the derivative is bounded. By Lemma 10.1, we have a Lipschitz constant $L = 1$ and thus, by Theorem 10.1, a unique solution of the IVP. In fact, for the initial condition $y(0) = 1/2$, the solution is given in Eq. 10.13.

Example 10.5 (No solution)**.** Consider $f(x,y) = y^2 + 1$. Again, the function is continuous and differentiable. However, $\partial f(x,y)/\partial y = 2y$ is unbounded, so Lemma 10.1 cannot be applied. Actually, we can explicitly see that f does *not* satisfy a Lipschitz condition. Namely, if it did, then the quotient

$$\frac{|f(x,y) - f(x,\hat{y})|}{|y - \hat{y}|} \quad \text{(for } x \in [0,2],\ y \neq \hat{y} \in \mathbb{R}) \tag{10.62}$$

would be bounded (by the Lipschitz constant L). However, in our case, set $\hat{y} = 0$; then

$$\frac{|f(x,y) - f(x,\hat{y})|}{|y - \hat{y}|} = \frac{|y^2 + 1 - 1|}{|y|} = |y| \tag{10.63}$$

which is *not* bounded. So f cannot satisfy a Lipschitz condition.

Hence, for the corresponding initial value problem,

$$y' = y^2 + 1, \quad 0 \le x \le 2, \quad y(0) = 0, \tag{10.64}$$

the existence and uniqueness result Theorem 10.1 does not apply. In fact, a solution for small x is given by

$$y(x) = \tan(x), \tag{10.65}$$

but this solution does *not* exist for all $x \in [0,2]$.

The problem here is that $\partial f(x,y)/\partial y$ is unbounded as y grows large. One can still control IVPs of this kind, using so-called *local Lipschitz conditions*, where the estimate Eq. 10.57 is required to hold only for $(x, \mathbf{y}), (x, \hat{\mathbf{y}}) \in \mathcal{D}$ with $\mathcal{D}$ any fixed compact set, and the Lipschitz constant L is allowed to depend on $\mathcal{D}$. Our numerical methods will still be applicable in this case, as long as we do not approach the possible singularities of the solution too closely. However, the formalism becomes much more complicated, and we do not treat these cases explicitly here.

Example 10.6 (Non-unique solution)**.** Consider $f(x,y) = \sqrt[3]{y}$. While f is continuous, the partial derivative $\partial f(x,y)/\partial y$ does *not exist* at $y = 0$, so again, Lemma 10.1 cannot be applied. Indeed, set $\hat{y} = 0$, then

$$\frac{|f(x,y) - f(x,\hat{y})|}{|y - \hat{y}|} = \frac{|y|^{1/3}}{|y|} = |y|^{-2/3} \tag{10.66}$$

which is unbounded near $y = 0$. So f does not satisfy a Lipschitz condition, and we are not guaranteed a unique solution of the IVP

$$y' = \sqrt[3]{y}, \quad 0 \le x \le 2, \quad y(0) = 0. \tag{10.67}$$

In fact, this IVP has at least *three* solutions:

$$y(x) = \left(\frac{2x}{3}\right)^{3/2}, \quad y(x) = -\left(\frac{2x}{3}\right)^{3/2}, \quad y(x) = 0. \tag{10.68}$$

This case is, in a sense, worse than the other examples above. Running our numerical methods for the IVP Eq. 10.67 is likely to give unpredictable results; even a small rounding error might cause us to "switch" between the different solutions of the IVP.

10.4 One-Step Difference Methods

We now return to the initial value problem Eq. 10.55:

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \quad a \le x \le b, \quad \mathbf{y}(a) = \alpha. \tag{10.69}$$

As in Euler's method, we use equally-spaced mesh points $x_0, \ldots, x_N$ in the interval $[a, b]$, with a uniform step size $h = (b-a)/N$, so $x_i = a + ih$, and we seek approximate values $\mathbf{w}_i \in \mathbb{R}^m$, with $\mathbf{w}_i \approx \mathbf{y}(x_i)$.

Euler's method was based on the Taylor expansion

$$\underbrace{y(x+h)}_{\text{exact}} = \underbrace{y(x) + hy'(x)}_{\text{approx}} + \underbrace{\frac{h^2}{2}y''(\xi)}_{\text{error}}. \tag{10.70}$$

Any other formula of this type can be used to define a similar method and, if the error term is smaller than the one in Euler's method, should lead to smaller errors in the approximate solution. In complete generality, and for vectors instead of scalars, we consider a formula

$$\mathbf{y}(x+h) = \mathbf{y}(x) + h\phi(x, \mathbf{y}(x), h) + \text{error} \tag{10.71}$$

All the higher-order Taylor approximations fit in to this framework, as do the important Runge-Kutta methods described below. This leads to the general single-step method

$$\begin{aligned} \mathbf{w}_0 &:= \ , \\ \mathbf{w}_{i+1} &:= \mathbf{w}_i + h\phi(x_i, \mathbf{w}_i, h). \end{aligned} \tag{10.72}$$

Introducing some notation for the error, we write

$$\mathbf{y}(x_{i+1}) = \mathbf{y}(x_i) + h\phi(x_i, \mathbf{y}(x_i), h) + h\tau_{i+1}(h) \tag{10.73}$$

where $\tau_{i+1}(h)$, the *local truncation error*, is defined by

$$\tau_{i+1}(h) = \frac{1}{h} \left[\mathbf{y}(x_{i+1}) - \mathbf{y}(x_i) - h\phi(x_i, \mathbf{y}(x_i), h) \right] \tag{10.74}$$

In terms of the numerical method, this is the error that would occur at step $i+1$ if we started on the exact solution curve, divided by the step size. We assume we have an upper bound $\tau(h)$ such that, for some $h_0 > 0$,

$$\|\tau_i(h)\| \le \tau(h) \tag{10.75}$$

for all $i \in \{0, \ldots, N\}$ and all $h \in (0, h_0]$. For the method to be useful, we must have $\tau(h) \to 0$ as $h \to 0^+$; in Euler's method, $\tau(h) = Mh/2$ but in other methods $\tau(h)$ is much smaller than this: typically $\tau(h) = O(h^n)$ as $h \to 0^+$ for some $n > 1$. We also assume the existence of a Lipschitz constant L such that

$$\|\phi(x, \mathbf{y}, h) - \phi(x, \hat{\mathbf{y}}, h)\| \le L\|\mathbf{y} - \hat{\mathbf{y}}\| \tag{10.76}$$

for all $x \in [a, b]$, for all $\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^m$ and all $h \in (0, h_0]$ (this replaces the second use of Taylor's Theorem in the analysis of Euler's method; looking back at that calculation, the Lipschitz constant is all that was needed). We conclude the setup by defining the actual error at step i by

$$\varepsilon_i = \mathbf{y}(x_i) - \mathbf{w}_i. \tag{10.77}$$

Under these hypotheses, we have:

Theorem 10.2. *Consider an initial value problem as in Eq. 10.55, which we assume to have a solution* $\mathbf{y}$. *For this IVP, consider the one-step difference method described by Eq. 10.72. Then,*

$$\|\mathbf{y}(x_i) - \mathbf{w}_i\| \le \frac{\tau(h)}{L} \left(e^{L(x_i - a)} - 1 \right) \quad \text{for all } h \in (0, h_0], \ i \in \{0, \ldots, N\}. \tag{10.78}$$

Proof. We establish this in much the same way as we did the corresponding result for Euler's method. Starting with the Taylor-like formula for $\mathbf{y}(x_{i+1})$ in terms of $\mathbf{y}(x_i)$, ϕ and τ_{i+1}, we have:

$$\begin{aligned} \mathbf{y}(x_{i+1}) &= \mathbf{y}(x_i + h) \\ &= \mathbf{y}(x_i) + h\phi(x_i, \mathbf{y}(x_i), h) + h\tau_{i+1}(h) \\ &= \varepsilon_i + \underbrace{\mathbf{w}_i + h\phi(x_i, \mathbf{w}_i, h)}_{=\mathbf{w}_{i+1}} + h[\phi(x_i, \mathbf{y}(x_i), h) - \phi(x_i, \mathbf{w}_i, h)] + h\tau_{i+1}(h) \\ &= \varepsilon_i + \mathbf{w}_{i+1} + h[\phi(x_i, \mathbf{y}(x_i), h) - \phi(x_i, \mathbf{w}_i, h)] + h\tau_{i+1}(h) \end{aligned} \tag{10.79}$$

so, subtracting $\mathbf{w}_{i+1}$ from both sides,

$$\varepsilon_{i+1} = \varepsilon_i + h[\phi(x_i, \mathbf{y}(x_i), h) - \phi(x_i, \mathbf{w}_i, h)] + h\tau_{i+1}(h) \tag{10.80}$$

Exactly as for Euler's method, we estimate using the triangle inequality

$$\|\varepsilon_{i+1}\| \leq \|\varepsilon_i\| + h\|\phi(x_i, \mathbf{y}(x_i), h) - \phi(x_i, \mathbf{w}_i, h)\| + \|\tau_{i+1}(h)\|$$
$$\leq (1 + hL)\|\varepsilon_i\| + h\tau(h)$$

(10.81)

using the Lipschitz hypothesis on ϕ and the upper bound τ for the local truncation errors. This is exactly the same formula as for Euler, so we can read off the same answer:

$$\|\varepsilon_n\| \leq \frac{\tau(h)}{L}(e^{L(x_n - a)} - 1) \leq \frac{\tau(h)}{L}(e^{L(b-a)} - 1)$$

(10.82)

$\square$

The main conclusion is that if we leave the integration width fixed and decrease the step size, then the error at each point is bounded above by a fixed multiple of $\tau(h)$: local error determines global error.

Thus, also in the general case, if the local truncation order is of order $O(h^n)$, then the the global error is of the same order. In the following, we will see a variety of one-step methods which aim at achieving a low local truncation error. We will then only need to prove an estimate for τ_i as defined in Eq. 10.74, and the behaviour of the global error with h will be under control.

10.5 Taylor Methods

We will now see the first example of one-step difference methods (beyond Euler's method), namely the so-called *Taylor Methods*.

The idea here is as follows. In Euler's Method, we approximated the exact solution $y(x)$ with a linear function, as in Eq. 10.23. In other words, we used a first-order Taylor expansion. For a better approximation, we can use an n-th order Taylor expansion, now for a vector-valued $\mathbf{y}(x)$:

$$\mathbf{y}(x_{i+1}) = \mathbf{y}(x_i + h) = \mathbf{y}(x_i) + h\frac{d\mathbf{y}}{dx}(x_i) + \frac{h^2}{2}\frac{d^2\mathbf{y}}{dx^2}(x_i) + \cdots$$
$$+ \frac{h^n}{n!}\frac{d^n\mathbf{y}}{dx^n}(x_i) + \mathbf{R}_i,$$

(10.83)

where, according to Theorem A.3, the remainder term $\mathbf{R}_i$ is bounded by

$$\|\mathbf{R}_i\| \leq \frac{h^{n+1}}{(n+1)!} \sup_{x \in [x_i, x_{i+1}]} \left\|\frac{d^{n+1}\mathbf{y}}{dx^{n+1}}(x)\right\|.$$

(10.84)

Following our approach in the Euler method, we now need to replace the unknown exact solution $\mathbf{y}(x)$ in the right-hand side of Eq. 10.83 with expressions in the function $\mathbf{f}$. Since $\mathbf{y}(x)$ solves the ODE, we can certainly write

$$\frac{d\mathbf{y}}{dx}(x) = \mathbf{f}(x, \mathbf{y}(x)),$$

(10.85)

To obtain the second derivative, we need to differentiate this, which requires the use of the chain rule At this point, we *need* to use the chain rule in several variables: Even if $m = 1$, the function f depends on two variables, x and y. One of the functions we need to consider (the outermost function) is the mapping from $\mathbb{R}^{m+1}$ to $\mathbb{R}^m$

$$(x, y_1, \ldots, y_m) \mapsto \mathbf{f}(x, \mathbf{y})$$

(10.86)

The derivative of this is the $m \times (m+1)$ matrix given by

$$\left[\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x}, \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathbf{y}}\right],$$

(10.87)

which represents the $m \times 1$ column vector $\partial \mathbf{f}(x, \mathbf{y})/\partial x$ being concatenated horizontally with the $m \times m$ matrix $\partial \mathbf{f}(x, \mathbf{y})/\partial \mathbf{y}$. The other function we need to consider (the innermost function) is the mapping from $\mathbb{R}$ to $\mathbb{R}^{m+1}$

$$x \mapsto [x, \mathbf{y}(x)] := (x, y_1(x), \dots, y_m(x)) \tag{10.88}$$

whose derivative is the $m + 1$ dimensional column vector

$$\begin{bmatrix} 1 \\ \mathbf{y}'(x) \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{f}(x, \mathbf{y}(x)) \end{bmatrix} \tag{10.89}$$

where we have substituted for $\mathbf{y}'(x)$ using the differential equation. Here we have stacked the scalar 1 on top of the m-dimensional column vector $\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x))$.

Now, the derivative of the RHS of Eq. 10.85 is the product of the $m \times (m + 1)$ matrix Eq. 10.87 and the $m + 1$ dimensional vector Eq. 10.89. This product has the general form

$$[A, B] \cdot \begin{bmatrix} C \\ D \end{bmatrix} = A \cdot C + B \cdot D \tag{10.90}$$

(which works for any block matrices, provided all the sizes are compatible, i.e. provided $A \cdot C + B \cdot D$ makes sense). We need to substitute

$$A = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x}; \qquad B = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathbf{y}}; \qquad C = 1; \qquad D = \mathbf{f}(x, \mathbf{y}(x)) \tag{10.91}$$

which gives us

$$\frac{d^2 \mathbf{y}}{dx^2}(x) = \underbrace{\frac{\partial \mathbf{f}}{\partial x}(x, \mathbf{y}(x)) + \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \cdot \mathbf{f}(x, \mathbf{y}(x))}_{=: \frac{d\mathbf{f}}{dx}(x, \mathbf{y}(x))}. \tag{10.92}$$

We call the right-hand side the *total derivative* of $\mathbf{f}$.

For higher derivatives, we can apply an analogous argument: We define recursively,

$$\begin{aligned} \frac{d^0 \mathbf{f}(x, \mathbf{y})}{dx^0} &:= \mathbf{f}(x, \mathbf{y}), \\ \frac{d^{j+1} \mathbf{f}(x, \mathbf{y})}{dx^{j+1}} &:= \frac{\partial}{\partial x} \frac{d^j \mathbf{f}(x, \mathbf{y})}{dx^j} + \left(\frac{\partial}{\partial \mathbf{y}} \frac{d^j \mathbf{f}(x, \mathbf{y})}{dx^j} \right) \cdot \mathbf{f}(x, \mathbf{y}). \end{aligned} \tag{10.93}$$

Then we know that, when evaluated on the exact solution, we have

$$\frac{d^j \mathbf{f}}{dx^j}(x, \mathbf{y}(x)) = \frac{d^{j+1} \mathbf{y}}{dx^{j+1}}(x, \mathbf{y}). \tag{10.94}$$

Hence, Eq. 10.83 rewrites to

$$\mathbf{y}(x_{i+1}) = \mathbf{y}(x_i) + h f(x_i, \mathbf{y}(x_i)) + \underbrace{\frac{h^2}{2} \frac{d\mathbf{f}}{dx}(x_i, \mathbf{y}(x_i)) + \cdots + \frac{h^n}{n!} \frac{d^{n-1}\mathbf{f}}{dx^{n-1}}(x_i, \mathbf{y}(x_i))}_{=: \mathbf{h}\phi(x, \mathbf{y}(x_i), h) =: \mathbf{hT}_n(x, \mathbf{y}(x_i), h)} + \mathbf{R}_i. \tag{10.95}$$

This motivates us to define the *Taylor method of order n* by the difference equation

$$\begin{aligned} \mathbf{w}_0 &:= \alpha, \\ \mathbf{w}_{i+1} &:= \mathbf{w}_i + h\mathbf{f}(x_i, \mathbf{w}_i) + \frac{h^2}{2} \frac{d\mathbf{f}}{dx}(x_i, \mathbf{w}_i) + \dots + \frac{h^n}{n!} \frac{d^{n-1}\mathbf{f}}{dx^{n-1}}(x_i, \mathbf{w}_i) \\ &= \mathbf{w}_i + h\mathbf{T}_n(x_i, \mathbf{w}_i, h), \end{aligned} \tag{10.96}$$

where

$$\mathbf{T}_n(x, \mathbf{y}, h) := \sum_{j=0}^{n-1} \frac{h^j}{(j+1)!} \frac{d^j \mathbf{f}}{dx^j}(x, \mathbf{y}). \tag{10.97}$$

We have thus defined a one-step difference method with function $\phi = \mathbf{T}_n$. According to Theorem 10.2, its global error is determined by its local truncation error, and by a Lipschitz constant for $\mathbf{T}_n$ (which we do not consider in detail here).

Theorem 10.3. *The local truncation error of the Taylor method of order n satisfies the bound*

$$\|\tau_i(h)\| \leq \frac{h^n}{(n+1)!} \sup_{x \in [a,b]} \left\|\frac{\mathrm{d}^{n+1}\mathbf{y}}{\mathrm{d}x^{n+1}}(x)\right\| =: \tau(h), \tag{10.98}$$

supposing that the exact solution $\mathbf{y}(x)$ *has* $(n+1)$ *continuous derivatives.*

Proof. By definition of the local truncation error in Eq. 10.74 and using Eq. 10.97, we have

$$\begin{aligned}
\|\tau_{i+1}(h)\| &= \frac{1}{h}\|\mathbf{y}(x_{i+1}) - \mathbf{y}(x_i) - h\,\mathbf{T}_n(x_i, \mathbf{y}(x_i), h)\| \\
&= \frac{1}{h}\Bigg\|\mathbf{y}(x_{i+1}) - \mathbf{y}(x_i) - \sum_{j=0}^{n-1}\frac{h^{j+1}}{(j+1)!}\frac{\mathrm{d}^j\mathbf{f}}{\mathrm{d}x^j}(x_i, \mathbf{y}(x_i))\Bigg\|.
\end{aligned} \tag{10.99}$$

However, comparing with Eq. 10.95, this means

$$\|\tau_{i+1}(h)\| = \frac{1}{h}\|\mathbf{R}_i\|, \tag{10.100}$$

and the proposed statement now follows from Eq. 10.84. $\qquad\square$

So, for any n, we get an approximation method of order $O(h^n)$. The Taylor method of order 1 coincides with Euler's method. Thus, as a side result, we have proved that Euler's method converges in the case of ODE systems.

Example 10.7. Let us once again consider the example IVP [1] from Eq. 10.12,

$$y'(x) = y(x) - x^2 + 1, \quad 0 \leq x \leq 1, \quad y(0) = \frac{1}{2}. \tag{10.101}$$

Thus $f(x,y) = y - x^2 + 1$. To set up the Taylor methods, say of order up to 4, we need to compute the first 3 total derivatives of f. In our example, this yields:

$$\begin{aligned}
\frac{\mathrm{d}f(x,y)}{\mathrm{d}x} &= \frac{\partial f(x,y)}{\partial x} + \frac{\partial f(x,y)}{\partial y}f(x,y) \\
&= (-2x) + 1\cdot(y - x^2 + 1) = y - x^2 - 2x + 1; \\
\frac{\mathrm{d}^2 f(x,y)}{\mathrm{d}x^2} &= \frac{\partial}{\partial x}\frac{\mathrm{d}f(x,y)}{\mathrm{d}x} + \frac{\partial}{\partial y}\frac{\mathrm{d}f(x,y)}{\mathrm{d}x}f(x,y) \\
&= (-2x - 2) + 1\cdot(y - x^2 + 1) = y - x^2 - 2x - 1; \\
\frac{\mathrm{d}^3 f(x,y)}{\mathrm{d}x^3} &= \frac{\partial}{\partial x}\frac{\mathrm{d}^2 f(x,y)}{\mathrm{d}x^2} + \frac{\partial}{\partial y}\frac{\mathrm{d}^2(x,y)f}{\mathrm{d}x^2}f(x,y) \\
&= (-2x - 2) + 1\cdot(y - x^2 + 1) = y - x^2 - 2x - 1.
\end{aligned} \tag{10.102}$$

In this simple example, we can now see that *all* higher-order total derivatives are equal to $\mathrm{d}^2 f(x,y)/\mathrm{d}x^2$; this will not normally be true.

Our second-order Taylor function T_2 is then

$$\begin{aligned}
T_2(x,y,h) &= f(x,y) + \frac{h}{2}\frac{\mathrm{d}f(x,y)}{\mathrm{d}x} \\
&= (y - x^2 + 1) + \frac{h}{2}(y - x^2 - 2x + 1) \\
&= (1 + \frac{h}{2})(y - x^2 + 1) - hx.
\end{aligned} \tag{10.103}$$

---

[1]The same example is considered in (Burden and Faires 2010, sec. 5.3).

The second-order Taylor method therefore reads,

$$
\begin{aligned}
w_0 :=& \frac{1}{2}, \\
w_{i+1} :=& w_i + hT_2(x_i, w_i, h) \\
=& w_i + (h + \frac{h^2}{2})(w_i - x_i^2 + 1) - h^2 x_i.
\end{aligned}
\tag{10.104}
$$

Analogously, one finds after a bit of computation that

$$
\begin{aligned}
T_4(x, y, h) =& f(x, y) + \frac{h}{2}\frac{\mathrm{d}\,f(x, y)}{\mathrm{d}x} + \frac{h^2}{6}\frac{\mathrm{d}^2 f(x, y)}{\mathrm{d}x^2} + \frac{h^3}{24}\frac{\mathrm{d}^3 f(x, y)}{\mathrm{d}x^3} \\
=& \left(1 + \frac{h}{2} + \frac{h^2}{6} + \frac{h^3}{24}\right)(y - x^2) \\
& + \left(1 + \frac{h}{3} + \frac{h^2}{12}\right)hx \\
& + \left(1 + \frac{h}{2} - \frac{h^2}{6} - \frac{h^3}{24}\right).
\end{aligned}
\tag{10.105}
$$

10.5.1 Advantages and disadvantages

We have seen that the error of the order n Taylor method is $O(h^n)$. This is much improved over the $O(h^1)$ of Euler's method. Thus, in most situations, Taylor methods will give a much more accurate result.

However, the error estimate depends on the supremum of the $(n+1)$-th derivative of the exact solution, and on the Lipschitz constant for T_n. We do not know a priori that these are small. Usually, this does not pose a problem in practice; we will however see some counterexamples in Section 10.9.

Taylor methods can be constructed for any order n, in a straightforward and unique way, by computing derivatives of the function f.

Still, Taylor methods are not so frequently used in practice. The main reason for this is that they require us to compute the total derivatives of the function f explicitly. In practice, this is too cumbersome to be done by hand. It can be achieved with symbolic differentiation algorithms, using computer algebra packages such as Maple. However, one would like to avoid this extra complexity. Also, it may not always be feasible to compute the derivatives explicitly: Suppose that, in a computer program, the function f is given as a "black box" procedure that computes (or rather approximates) the function values $f(x, y)$ numerically; how would we gain access to the derivatives of f?

10.6 Runge-Kutta Methods

Runge-Kutta methods are another example of one-step difference methods; they are very relevant in practice. They arise as modifications of the Taylor methods discussed in the previous section. Their main advantage is that they do *not* require us to compute derivatives of the function f explicitly.

10.6.1 Motivation: The Modified Euler Method

The idea of Runge-Kutta methods is to avoid computing the total derivatives $d^n f/dx^n$. Rather, these are replaced with *finite difference quotients.*

Recall that, if g is a twice differentiable function, then we can use Taylor's Theorem to write

$$
g(x + h) = g(x) + hg'(x) + O(h^2)
\tag{10.106}
$$

and rearrange to give

$$g'(x) = \frac{g(x+h) - g(x)}{h} + O(h). \tag{10.107}$$

The fraction on the r.h.s. is called a finite difference quotient. We also remark that, by the MVT (or Taylor's Theorem)

$$g(x+h) = g(x) + O(h). \tag{10.108}$$

We can replace h by $O(h^k)$ here to give

$$g(x + O(h^k)) = g(x) + O(O(h^k)) = g(x) + O(h^k). \tag{10.109}$$

We start with the Taylor method of order two, here for $m = 1$, given by the function

$$T_2(x, y, h) = f(x, y) + \frac{h}{2}\frac{\mathrm{d}f}{\mathrm{d}x}(x, y). \tag{10.110}$$

We use a difference quotient to approximate the total derivative, evaluated on the exact solution $y(x)$:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x, y(x)) = \frac{f(x+h, y(x+h)) - f(x, y(x))}{h} + O(h). \tag{10.111}$$

Inside this expression, for the term $y(x+h)$, we use a Taylor approximation:

$$y(x+h) = y(x) + hy'(x) + O(h^2) = y(x) + hf(x, y(x)) + O(h^2). \tag{10.112}$$

Using Eq. 10.109 we obtain

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x, y(x)) = \frac{1}{h}\Big(f(x+h, y(x) + hf(x, y(x))) - f(x, y(x))\Big) + O(h). \tag{10.113}$$

Inserting into Eq. 10.110, we have

$$T_2(x, y(x), h) = \frac{1}{2}f(x, y(x)) + \frac{1}{2}f(x+h, y(x) + hf(x, y(x))) + O(h^2). \tag{10.114}$$

Thus, if instead of the function T_2, we use the function

$$\phi(x, y, h) = \frac{1}{2}f(x, y) + \frac{1}{2}f(x+h, y + hf(x, y)) \tag{10.115}$$

for our one-step method, we will incur an additional local truncation error of order $O(h^2)$. However, this is not "much worse" than the Taylor method, which already has an error of order $O(h^2)$.

The one-step difference method corresponding to ϕ is called the *Modified Euler method*. Its difference equation, $w_{i+1} := w_i + h\phi(x_i, w_i, h)$, can be rewritten in the following cleaned-up form:

$$\begin{aligned}
w_0 &:= \alpha, \\
k_{i,1} &:= hf(x_i, w_i), \\
k_{i,2} &:= hf(x_i + h, w_i + k_{i,1}), \\
w_{i+1} &:= w_i + \frac{1}{2}k_{i,1} + \frac{1}{2}k_{i,2}.
\end{aligned} \tag{10.116}$$

This shows in particular that only *two* evaluations of f are needed in each step of the method. That is important to know, since the evaluation of f is usually the time-consuming part when the method is implemented on a computer.

We have seen now, roughly, that the Modified Euler method works, and is of order $O(h^2)$. A more complete proof will follow below. Let us first have a look at other methods of the same kind.

10.6.2 General Runge-Kutta methods

A general Runge-Kutta method is defined by a *Butcher tableau*:

$$
\begin{array}{c|ccccc}
a_1 & & & & & \\
a_2 & b_{21} & & & & \\
a_3 & b_{31} & b_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
a_s & b_{s1} & b_{s2} & \dots & b_{s,s-1} & \\
\hline
 & c_1 & c_2 & \dots & c_{s-1} & c_s
\end{array}
\tag{10.117}
$$

where a_i, b_{ij}, c_i are real numbers. The tableau is a shorthand notation for the associated difference equation of a one-step method:

$$
\begin{aligned}
\mathbf{w}_0 &:= \alpha; \\
\mathbf{k}_1 &:= h\mathbf{f}(x_i + a_1 h, \mathbf{w}_i + 0), \\
\mathbf{k}_2 &:= h\mathbf{f}(x_i + a_2 h, \mathbf{w}_i + b_{21}\mathbf{k}_1), \\
\mathbf{k}_3 &:= h\mathbf{f}(x_i + a_3 h, \mathbf{w}_i + b_{31}\mathbf{k}_1 + b_{32}\mathbf{k}_2), \\
&\ \ \vdots \\
\mathbf{k}_s &:= h\mathbf{f}(x_i + a_s h, \mathbf{w}_i + \sum_{j=1}^{s-1} b_{sj}\mathbf{k}_j); \\
\mathbf{w}_{i+1} &:= \mathbf{w}_i + \sum_{j=1}^{s} c_j \mathbf{k}_j
\end{aligned}
\tag{10.118}
$$

(The $\mathbf{k}_j$ depend in addition on the step i, but they are regarded as "intermediate results", and we do not denote this dependence explicitly.) Our Modified Euler method is an example of such a scheme, namely with the tableau

$$
\begin{array}{c|cc}
0 & & \\
1 & 1 & \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}
\tag{10.119}
$$

We do not discuss here in general how these tableaux are obtained, or how to prove in general of what order their local truncation error is. However, here are some more examples.

The *Midpoint method* is another method of order $O(h^2)$. It is given by the tableau

$$
\begin{array}{c|cc}
0 & & \\
\frac{1}{2} & \frac{1}{2} & \\
\hline
 & 0 & 1
\end{array}
\tag{10.120}
$$

and its difference equation can be written explicitly as

$$
\mathbf{w}_0 := \alpha, \quad \mathbf{w}_{i+1} = \mathbf{w}_i + h\,\mathbf{f}\!\left(x_i + \frac{h}{2}, \mathbf{w}_i + \frac{h}{2}\mathbf{f}(x_i, \mathbf{w}_i)\right).
\tag{10.121}
$$

There are more methods of order $O(h^2)$; for example, *Heun's method*: [2]

$$
\begin{array}{c|cc}
0 & & \\
\frac{2}{3} & \frac{2}{3} & \\
\hline
 & \frac{1}{4} & \frac{3}{4}
\end{array}
\tag{10.122}
$$

The explicit form of the difference equation is

$$
\mathbf{w}_0 := \alpha, \quad \mathbf{w}_{i+1} = \mathbf{w}_i + \frac{h}{4}\Big(\mathbf{f}(x_i, \mathbf{w}_i) + 3\mathbf{f}(x_i + \frac{2}{3}h, \mathbf{w}_i + \frac{2}{3}h\,\mathbf{f}(x_i, \mathbf{w}_i))\Big).
\tag{10.123}
$$

---

[2]The names of methods may vary in the literature; it is best to compare the tableaux when reading different sources!

Finally, let us mention the "classical" Runge-Kutta method [3] of order $O(h^4)$. This method is widely used in practice. Its Butcher tableau is

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
& \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6}
\end{array}
\tag{10.124}
$$

and it is *not* very useful to write its difference equation in one line! (In general, when used in programming, working with the intermediate results $\mathbf{k}_i$ is a good way of organizing the code.)

10.6.3 Local truncation error for second-order methods

The general idea of proving a rigorous error estimate for a Runge-Kutta method is comparing it to the Taylor method of order n, with n appropriately chosen. Given the function $\phi_{\mathrm{RK}}(x, y, h)$ that defines the method (as a one-step difference method), one would try to prove that

$$
\|\,_{\mathrm{RK}}(x, \mathbf{y}(x), h) - \mathbf{T}_n(x, \mathbf{y}(x), h)\| \leq c h^n
\tag{10.125}
$$

with a constant $c > 0$. This proof will usually involve a Taylor expansion of the function $\mathbf{f}$ within ϕ_{RK}, and the constant c will depend on estimates for the function $\mathbf{f}$ and its derivatives. Once Eq. 10.125 is known, it follows from the definition of the local truncation error Eq. 10.74 and the triangle inequality that

$$
\tau_{\mathrm{RK}}(h) \leq \tau_{\mathrm{Taylor}\text{-}n}(h) + c\, h^n.
\tag{10.126}
$$

With the estimate for $\tau_{\mathrm{Taylor}\text{-}n}(h)$ (the truncation error of the n-th order Taylor method) known from Theorem 10.3 this yields

$$
\tau_{\mathrm{RK}}(h) \leq c'\, h^n.
\tag{10.127}
$$

with another constant $c' > 0$. Theorem 10.2 then tells us that the global error of the Runge-Kutta method is of order $O(h^n)$.

However, for most higher-order methods, such as the classical Runge-Kutta method Eq. 10.124 of order $O(h^4)$, the proof of Eq. 10.125 is rather tedious, since the computation becomes quite lengthy.

Here, we will give the proof only for $O(h^2)$ methods with a Butcher tableau of the form

$$
\begin{array}{c|cc}
0 & & \\
c & c & \\
\hline
& (1-d) & d
\end{array}
\tag{10.128}
$$

where $c, d \in (0, 1]$, $cd = \frac{1}{2}$. This includes the Modified Euler method ($c = 1$, $d = 1/2$), the Midpoint method ($c = 1/2$, $d = 1$), and Heun's method ($c = 2/3$, $d = 3/4$).

Theorem 10.4. *Consider a Runge-Kutta method of the form Eq. 10.128. Suppose that the function* $\mathbf{f}$ *and all its derivatives up to order* 2 *are bounded.* [4] *Then, there exists a constant C such that the local truncation error is bounded by*

$$
\|\tau_i(h)\| \leq Ch^2.
\tag{10.129}
$$

---

[3]If you hear someone speaking of "the Runge-Kutta method", they are probably referring to this $O(h^4)$ method.

[4]This is not necessarily a realistic assumption - but one that simplifies the proof. One can do a similar argument using local bounds on $\mathbf{f}$ and its derivatives in a suitable neighbourhood of the exact solution; however, this introduces formal complications which we want to avoid here.

Proof. As explained above, we need to prove Eq. 10.125 in our case. The function ϕ is here given by

$$\phi(x, \mathbf{y}, h) = (1 - d)\,\mathbf{f}(x, \mathbf{y}) + d\,\underbrace{\mathbf{f}(x + ch, \mathbf{y} + ch\mathbf{f}(x, \mathbf{y}))}_{=:\mathbf{k}}, \tag{10.130}$$

and we need to compare it to the Taylor method of order 2, given by

$$\mathbf{T}_2(x, y, h) = \mathbf{f}(x, \mathbf{y}) + \frac{h}{2}\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}x}(x, \mathbf{y}). \tag{10.131}$$

We first take the term $\mathbf{k}$ from Eq. 10.130, and use a first-order Taylor expansion of the function $\mathbf{f}$ around the point $(x, \mathbf{y})$, in the form of Theorem A.5. This gives

$$\begin{aligned}
\mathbf{k} &= \mathbf{f}(x, \mathbf{y}) + \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial(x, y^{(1)}, \dots, y^{(m)})} \cdot \begin{pmatrix} ch \\ ch\, f^{(1)}(x, \mathbf{y}) \\ \vdots \\ ch\, f^{(m)}(x, \mathbf{y}) \end{pmatrix} + \mathbf{R} \\
&= \mathbf{f}(x, \mathbf{y}) + ch\frac{\partial \mathbf{f}}{\partial x} + ch\frac{\partial \mathbf{f}}{\partial \mathbf{y}} \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{R} \\
&= \mathbf{f}(x, \mathbf{y}) + ch\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}x}(x, \mathbf{y}) + \mathbf{R}.
\end{aligned} \tag{10.132}$$

We assume that all second partial derivatives of $\mathbf{f}$ (by x or y_j) are bounded by a constant M. According to Theorem A.5, the Taylor remainder term $\mathbf{R}$ is then bounded by

$$\|\mathbf{R}\| \leq \frac{1}{2}(ch)^2 \|(1, f_1, \dots, f_m)\|^2 \cdot (m + 1)^2 M. \tag{10.133}$$

After possibly modifying the constant M, we can assume that $\|f_j\| \leq M$ as well, and that $M \geq 1$. This gives

$$\|\mathbf{R}\| \leq \frac{(m + 1)^2 M^3 c^2}{2} h^2. \tag{10.134}$$

Inserting Eq. 10.132 into Eq. 10.130 yields

$$\begin{aligned}
\phi(x, \mathbf{y}, h) &= (1 - d)\mathbf{f}(x, \mathbf{y}) + d\mathbf{k} \\
&= \mathbf{f}(x, \mathbf{y}) + cdh\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}x}(x, y) + d\mathbf{R} \\
&= \mathbf{T}_2(x, y, h) + d\mathbf{R},
\end{aligned} \tag{10.135}$$

since $cd = 1/2$. Consequently,

$$\|\phi(x, \mathbf{y}, h) - \mathbf{T}_2(x, \mathbf{y}, h)\| \leq \frac{(m + 1)^2 M^3 c}{4} h^2, \tag{10.136}$$

which completes the proof. $\qquad\square$

10.6.4 Advantages and disadvantages

Runge-Kutta methods are very widely used in practice. While they are slightly less accurate than Taylor methods, they can be chosen of the same order $O(h^n)$, and in this sense they are "as good" as Taylor methods. However, they have the advantage that it is not necessary to compute the total derivatives of $\mathbf{f}$ explicitly. They are rather straightforward to implement on a computer, given the Butcher tableau of a method. Particularly, the "classical" Runge-Kutta method is a good choice where one needs an algorithm of reasonable accuracy which is easy to implement.

However, the error bounds of Runge-Kutta methods are sensitive to the higher-order derivatives of $\mathbf{f}$. (They share this problem with Taylor methods.) In some cases, to be discussed in Section 10.9, this leads to problems.

10.7 Error Control, Runge-Kutta-Fehlberg Method

10.7.1 Error control

We have so far discussed how the approximation error changes with the step size h, but we have not really explained how to choose h in practice.

In principle, given an IVP, one might first try to prove rigorous error estimates, like in the example in Section 10.2, but with h left open. Then one can choose h according to the maximum error one wants to allow. However, these estimates are very hard (if not impossible) to compute in realistic examples.

Instead, one would like to have an approximation algorithm that automatically computes an estimate for the error, at least roughly, and that chooses the step size h accordingly. This technique is known as *error control*.

The idea is roughly as follows. Let us use *two* approximation methods, with approximation values $\mathbf{w}_i$ and $\tilde{\mathbf{w}}_i$. Suppose that $\tilde{\mathbf{w}}_i$ is much more exact than $\mathbf{w}_i$, i.e., that the absolute error of $\tilde{\mathbf{w}}_i$ is much smaller than that of $\mathbf{w}_i$. (For example, the second approximation method might be of higher order in h.) Then we have

$$\|\mathbf{y}(x_i) - \mathbf{w}_i\| \le \underbrace{\|\mathbf{y}(x_i) - \tilde{\mathbf{w}}_i\|}_{\text{negligible}} + \|\tilde{\mathbf{w}}_i - \mathbf{w}_i\| \approx \|\tilde{\mathbf{w}}_i - \mathbf{w}_i\|. \tag{10.137}$$

However, the right-hand side can be computed without knowing the exact solution $\mathbf{y}(x)$.

To make this more concrete, let us say that $\mathbf{w}_i$ and $\tilde{\mathbf{w}}_i$ are computed by two one-step difference methods, with defining functions ϕ and $\tilde{\phi}$, respectively. Let us assume that ϕ yields a local truncation error $\|\tau_i(h)\|$ of order $O(h^n)$, while the local truncation error $\|\tilde{\tau}_i(h)\|$ of $\tilde{\phi}$ is of order $O(h^{n+1})$.

Further, we fix i and suppose that $\mathbf{y}(x_i) = \mathbf{w}_i = \tilde{\mathbf{w}}_i$, that is, we start form an exact value in the previous step. [5] For given step size h, we want to estimate the local truncation error $\|\tau_{i+1}(h)\|$ of the first method, supposing that $\tilde{\tau}_{i+1}$ is negligible compared with it.

$$\|\tau_{i+1}(h)\| = \frac{1}{h}\|\mathbf{y}(x_{i+1}) - \mathbf{y}(x_i) - h\phi(x_i, \mathbf{w}_i, h)\|$$

$$= \frac{1}{h}\|\underbrace{(\mathbf{y}(x_{i+1}) - \mathbf{y}(x_i) - h\tilde{\phi}(x_i, \mathbf{w}_i, h))}_{\sim\, h\|\tilde{\tau}_{i+1}\|,\ \text{negligible}} + h\tilde{\phi}(x_i, \mathbf{w}_i, h) - h\phi(x_i, \mathbf{w}_i, h)\| \tag{10.138}$$

$$\approx \|\tilde{\phi}(x_i, \mathbf{w}_i, h) - \phi(x_i, \mathbf{w}_i, h)\|.$$

Thus we can determine the local truncation error approximately by evaluating the functions ϕ and $\tilde{\phi}$.

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
\text{(a)} & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & \\
\text{(b)} & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array} \tag{10.139}$$

However, what we actually want is to choose the step size h so that $\|\tau_{i+1}(h)\| \le T$ with some given "tolerance" value T. To that end, let us assume that $\|\tau_{i+1}(h)\|$ is not only of order $O(h^n)$, but in fact (roughly) proportional to h^n:

$$\|\tau_{i+1}(h)\| \approx K\,h^n \quad \text{for some } K > 0. \tag{10.140}$$

---

[5]We will not deal with any "global" error behaviour here.

We proceed as follows. We start with some fixed step size h and compute the related function values:

$$|\tau_{i+1}(h)| \approx K\,h^n \approx \|\tilde{\phi}(x_i, \mathbf{w}_i, h) - \phi(x_i, \mathbf{w}_i, h)\|. \tag{10.141}$$

Now we want to adjust h by a factor $q > 0$, such that $\|\tau_{i+1}(qh)\| \approx T$. By our assumption Eq. 10.140,

$$\|\tau_{i+1}(qh)\| \approx K\,(qh)^n \approx T. \tag{10.142}$$

Dividing Eq. 10.142 by Eq. 10.141 yields

$$q^n \approx \frac{T}{\|\tilde{\phi}(x_i, \mathbf{w}_i, h) - \phi(x_i, \mathbf{w}_i, h)\|}, \tag{10.143}$$

thus we should choose

$$q = \left(\frac{T}{\|\tilde{\phi}(x_i, \mathbf{w}_i, h) - \phi(x_i, \mathbf{w}_i, h)\|}\right)^{1/n} = \left(\frac{hT}{\|\tilde{\mathbf{w}}_{i+1} - \mathbf{w}_{i+1}\|}\right)^{1/n}, \tag{10.144}$$

where it is understood that $\mathbf{w}_{i+1}, \tilde{\mathbf{w}}_{i+1}$ are computed with $\phi, \tilde{\phi}$ using the step size h.

10.7.2 The Runge-Kutta-Fehlberg method

A popular implementation of the above principle is the *Runge-Kutta-Fehlberg* method. It uses an order-4 Runge-Kutta method for ϕ and an order-5 Runge-Kutta method for $\tilde{\phi}$. The Butcher tableaux for both methods is **?@eq-butcherrkf**.

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
\textbf{(a)} & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & \\
\textbf{(b)} & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}
$$

The special feature of this algorithm is that the two approximation methods *share* most parts of their coefficient schemes; more precisely, the auxiliary values $\mathbf{k}_1, \ldots, \mathbf{k}_6$ are the same in both approximation methods. Only the "final lines" in the Butcher tableaux are different: for computing $\mathbf{w}_{i+1}$, one uses the line marked (a), while for $\tilde{\mathbf{w}}_{i+1}$, the line marked (b) is used. This arrangement makes the algorithm rather efficient: the function $\mathbf{f}$ needs to be evaluated only 6 times per step. Compare this with the 4 function evaluations needed for the classical Runge-Kutta method of order 4 *without* error control. The additional overhead caused by the error control method is noticeable, but not too large.

Algorithm 10.1 shows the Runge-Kutta-Fehlberg method in pseudocode. It works roughly by the principles discussed above, but with some modification in detail. In lines 4–8, the next approximation value $\mathbf{w}_{i+1}$ is computed; however, this happens only if the estimate for the local truncation error, computed according to Eq. 10.138, is below the specified tolerance T. Then, in lines 9–22, the step size h for the next step is computed. In slight deviation from Eq. 10.144, one chooses a more conservative factor,

$$q = \left(\frac{hT}{2\|\tilde{\mathbf{w}}_{i+1} - \mathbf{w}_{i+1}\|}\right)^{1/4} \tag{10.145}$$

(note $n = 4$). Also, the factor q is limited to the interval $[0.1, 4]$ (lines 10–16), in order to avoid possible instabilities by rapid changes in the step size. Further, the step size is never increased beyond a given value $h_{\max}$ (line 17); and if it decreases below a minimum step size $h_{\min}$, we terminate with an error message (line 21) – otherwise we might run into problems with limited floating point precision. Some extra handling is needed for the last step of the approximation, in order to ensure that we end up exactly at the right-hand boundary of the interval $[a, b]$ (line 19).

Algorithm 10.1 The Runge-Kutta-Fehlberg method

1: **function** RUNGEKUTTAFEHLBERG$(a, b, \alpha, T, h_{min}, h_{max})$
2:        $\to$ tolerance T; minimal/maximal step size h_{min}, h_{max}
3:    $x := a, w := \alpha, h := h_{max}$
4:    **repeat**
5:       Compute $k_1, \dots, k_6$ from Butcher tableau with step size h
6:       $R := h^{-1} \| \sum_{j=1}^{6} (c_j - \tilde{c}_j) k_j \|$ $\to$ Estimate local truncation error
7:       **if** $R \leq T$ **then**
8:          $x := x + h, w := w + \sum_{j=1}^{5} c_j k_j$ $\to$ accept approximation
9:       **end if**
10:          $\to$ Now choose the new step size
11:       $q := (T/2R)^{1/4}$ $\to$ factor to multiply h with
12:       **if** $q \leq 0.1$ **then**
13:          $h := 0.1h$ $\to$ do not decrease too much
14:       **else if** $q \geq 4$ **then**
15:          $h := 4h$ $\to$ do not increase too much
16:       **else**
17:          $h := h \cdot q$
18:       **end if**
19:       **if** $h \geq h_{max}$ **then**
20:          $h := h_{max}$ $\to$ do not exceed max step size
21:       **end if**
22:       **if** $x + h > b$ **then**
23:          $h := b - x$ $\to$ next step would exceed interval
24:       **else if** $h < h_{min}$ **then**
25:          Error: minimum step size exceeded
26:       **end if**
27:    **until** $x \geq b$
28:    **return** w
29: **end function**

10.8 Multi-Step Methods

N.B.: This section is intended as a brief overview, and its material is not examinable. More information can be found, e.g., in (Burden and Faires 2010, sec. 5.6).

Multi-step methods are an altervative way of approximating the solution of the initial value problem

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad a \leq x \leq b, \quad \mathbf{y}(a) = \alpha. \tag{10.146}$$

Again, they are generalizations of Euler's method, and we use the same division of $[a, b]$ into equally spaced mesh points $x_0, \dots, x_n$. But rather than using several evaluations of the function $\mathbf{f}$ in each approximation step, like Runge-Kutta methods do, we make use of the values of $\mathbf{f}$ that have already been computed at previous mesh points.

A general *k-step multi-step method* is given by a difference equation

$$
\begin{aligned}
\mathbf{w}_0 &:= \ , \\
\mathbf{w}_1 &:= \alpha_1, \quad \dots \quad , \ \mathbf{w}_{k-1} := \alpha_{k-1}, \\
\mathbf{w}_{i+1} &:= \alpha_{k-1}\mathbf{w}_i + \alpha_{k-2}\mathbf{w}_{i-1} + \cdots + \alpha_0 \mathbf{w}_{i-k+1} \\
&\quad + h\left(b_k \mathbf{f}(x_{i+1}, \mathbf{w}_{i+1}) + b_{k-1}\mathbf{f}(x_i, \mathbf{w}_i) + \cdots + b_0 \mathbf{f}(x_{i-k+1}, \mathbf{w}_{i-k+1}) \right).
\end{aligned}
\tag{10.147}
$$

Here $a_0 \dots a_{k-1}, b_0 \dots b_k \in \mathbb{R}$ are constants that define the method. $\alpha_1 \dots \alpha_{k-1} \in \mathbb{R}^m$ are certain starting values that are needed for the difference equation to work; we will discuss later how to obtain them. If $b_k = 0$, the method is called *explicit*; otherwise, it is called *implicit*. In the latter case, $\mathbf{w}_{i+1}$ appears

on both sides of the difference equation, and is hence defined only implicitly; we will discuss later how such methods can be used.

Like for one-step methods, one defines the *local truncation error* of the multistep method Eq. 10.147 by

$$\tau_{i+1} := \frac{1}{h}\left[\mathbf{y}(x_{i+1}) - \sum_{j=0}^{k-1} a_{k-1-j}\mathbf{y}(x_{i-j}) - h\sum_{j=0}^{k} b_{k-j}\mathbf{f}(x_{i+1-j}, \mathbf{y}(x_{i-j+1}))\right]. \tag{10.148}$$

Again, the local truncation error determines the global error, i.e., an analogue of Theorem 10.2 for multi-step methods holds. However, we do not discuss this here.

Examples of k-step multistep methods are the so-called Adams-Bashforth and Adams-Moulton methods. They are derived from rewriting the differential equation as an integral equation,

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}(x)) \quad \Leftrightarrow \quad \mathbf{y}(x_{i+1}) = \mathbf{y}(x_i) + \int_{x_i}^{x_{i+1}} \mathbf{f}(x, \mathbf{y}(x))\,dx, \tag{10.149}$$

and then approximating the components $\hat{f}_j$ on the right hand side with Lagrange interpolating polynomials. The integral can be done explicitly, and one obtains the difference equation of a multi-step method; see (Burden and Faires 2010, sec. 5.6) for details. The number k of steps depends on the number of interpolation points chosen. Adams-Bashforth methods (**?@tbl-abmethod**) are k-step *explicit* methods of error order $O(h^k)$ and Adams-Moulton methods (**?@tbl-ammethod**) are k-step *implicit* methods of error order $O(h^{k+1})$,

Notation: $\mathbf{f}_j := \mathbf{f}(x_j, \mathbf{w}_j)$.

As with Taylor and Runge-Kutta methods, the estimates for the local truncation error depend on higher-order total derivatives of $\mathbf{f}$.

In order to apply these multi-step methods in examples, we need additional start values $\mathbf{w}_1 = \alpha_1, \dots, \mathbf{w}_{k-1} = \alpha_{k-1}$. These are normally obtained by using one-step methods of the same error order. For example, for the four-step Adams-Bashforth method, one might use the classical Runge-Kutta method of order $O(h^4)$.

Notation: $\mathbf{f}_j := \mathbf{f}(x_j, \mathbf{w}_j)$.

10.8.1 Predictor-corrector methods

Implicit multi-step methods, like the Adams-Moulton methods shown in **?@tbl-ammethod**, involve the term $\mathbf{f}(x_{i+1}, \mathbf{w}_{i+1})$ on the right-hand side of the difference equation. This means that $\mathbf{w}_{i+1}$ is defined only implicitly; it is unclear at first how these methods should be used in practice.

One option would be to solve the difference equation for $\mathbf{w}_{i+1}$, either symbolically (for simple cases of $\mathbf{f}$) or numerically. This is useful only in very specific cases; we will come back to this approach in Section 10.9.

The other, and indeed frequently used, option is to combine them with explicit multi-step methods into so-called *predictor-corrector methods*. These work as follows.

Suppose that $\mathbf{w}_0, \dots, \mathbf{w}_i$ are already known. These are inserted into the difference equation of an explicit method, the *(predictor)*, which gives an approximation value $\mathbf{w}_{i+1}^{(p)}$. This value is then inserted into the r.h.s. of the difference equation of an implicit method, the *corrector*, in order to obtain $\mathbf{w}_{i+1}$.

To illustrate this, let us consider a frequently used case: the Adams-Bashforth 4-step method as the predictor, combined with the Adams-Moulton 3-step method as the corrector. In each step of the method, we compute the "predicted value" by the explicit method,

$$\mathbf{w}_{i+1}^{(p)} = \mathbf{w}_i + \frac{h}{24}(55\mathbf{f}_i - 59\mathbf{f}_{i-1} + 37\mathbf{f}_{i-2} - 9\mathbf{f}_{i-3}), \tag{10.150}$$

and afterwards the "corrected value" using the implicit method:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \frac{h}{24}(9\mathbf{f}(x_{i+1}, \mathbf{w}_{i+1}^{(p)}) + 19\mathbf{f}_i - 5\mathbf{f}_{i-1} + \mathbf{f}_{i-2}). \tag{10.151}$$

Here $\mathbf{f}_j := \mathbf{f}(x_j, \mathbf{w}_j)$. Both the predictor and the corrector are of order $O(h^4)$, and so is the entire method, as it turns out. However, the combined predictor-corrector method is more precise than each of the parts alone.

Algorithm 10.2 Adams fourth-order predictor-corrector

1: **function** ADAMSPREDICTORCORRECTOR(a, b, α, N)
2:        $\rightarrow$ endpoints $a \leq b$; initial condition α; number of steps N
3:    $h := (b-a)/N$, $x_0 := a$, $w_0 := \alpha$, $f_0 := f(x_0, w_0)$.
4:    **for** $i = 1, 2, 3$ **do** $\rightarrow$ prepare first steps
5:       Compute w_i using Runge-Kutta fourth order
6:       $x_i := x_{i-1} + h$; $f_i := f(x_i, w_i)$
7:    **end for**
8:    **for** $i = 4, \dots, N$ **do** $\rightarrow$ main approximation
9:       $x := a + ih$
10:       $w := w_3 + \frac{h}{24}(55f_3 - 59f_2 + 37f_1 - 9f_0)$ $\rightarrow$ predict
11:       $w := w_3 + \frac{h}{24}(9f(x, w) + 19f_3 - 5f_2 + f_1)$ $\rightarrow$ correct
12:       **for** j = 0,1,2 **do**
13:          $x_j := x_{j+1}$; $w_j := w_{j+1}$; $f_j := f_{j+1}$;
14:       **end for**
15:       $x_3 := x$; $w_3 := w$; $f_3 := f(x, w)$;
16:    **end for**
17:
18:    **return** w
19: **end function**

Algorithm 10.2 shows this predictor-corrector method in pseudocode. Lines 3–6 use the classical Runge-Kutta method to set up the required initial values, while lines 7–15 implement the actual multistep method. The scheme is quite efficient, since in each step of the method, only two evaluations of the function $\mathbf{f}$ are used: in line 10 and in line 14 of the algorithm. All other values $\mathbf{f}_j$ are already known from the previous steps.

10.8.2 Advantages and disadvantages

Multistep methods, in particular Adams-Bashforth and Adams-Moulton methods, can be formulated for any error order $O(h^n)$. However, independent of n, they involve only one evaluation of $\mathbf{f}$ per step (or two for predictor-corrector methods). Compare this with Runge-Kutta methods, where the number of evaluations needed in each step grows approximately like n. Therefore, in particular at high orders, multistep methods tend to be faster than Runge-Kutta methods.

We can naturally combine an explicit and an implicit method of same error order into a predictor-corrector method, which further improves precision. (It should be noted that a similar approach is possible with implicit Runge-Kutta methods, which we have not discussed here.)

However, multistep methods have the disadvantage that they are more difficult to implement. In particular, an additional one-step method is needed in order to compute the extra start values needed for the method.

Like for Taylor and Runge-Kutta methods, the error estimates for Adams-Bashforth and Adams-Moulton methods depends on higher-order derivatives of $\mathbf{f}$, which are not always small. We will discuss this in Section 10.9.

Similar to the Runge-Kutta-Fehlberg algorithm described in Section 10.7, we can modify Adams predictor-corrector methods so that they include an automatic choice of the step size. The difference

between predicted and corrected value gives a natural indication of the local truncation error. However, the implementation of such a method is not really straightforward, and we do not discuss it in detail here; see for example (Burden and Faires 2010, sec. 5.7). One of the difficulties is that, each time when we change the step size h, also the initial values α_j need to be computed again. This can make changing the step size rather costly. It is also possible to generalize multistep methods so that they work with variable *order*. Again, we do not discuss that here.

10.9 Stiff equations

In this section, we will discuss a certain class of initial value problems, called *stiff equations*, which give rise to particular problems when applying numerical methods to them.

Let us consider the following IVP as an example:

$$y'(x) = -20y(x) + 10\cos(2x), \quad 0 \le x \le 3, \quad y(0) = 1. \tag{10.152}$$

This IVP has the exact solution

$$y(x) = \underbrace{\frac{50}{101}\cos(2x) + \frac{5}{101}\sin(2x)}_{\text{steady-state solution}} + \underbrace{\frac{51}{101}\exp(-20x)}_{\text{transient solution}}. \tag{10.153}$$

Note here that one part of the solution, the *transient solution*, decays very rapidly as x grows. For large x, only the remainder - the *steady-state solution* - contributes to $y(x)$.

When applying our approximation methods developed so far to this IVP, one notices the following behaviour: Below a certain "critical step size", our methods give a reasonable approximation of the exact solution, as expected. However, above this step size, the approximation is completely unreliable, and usually grows very rapidly – although the exact solution is bounded by 1. This applies to all methods developed so far, and going to higher-order methods – such as classical Runge-Kutta – does not help.

The reason for this becomes apparent when computing the derivatives of the transient solution:

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n}\exp(-20x) = (-20)^n\exp(-20x). \tag{10.154}$$

This expression can be very large, in particular for high n. Since our error bounds all depended on the supremum of higher-order derivatives of $y(x)$, this explains the large approximation error.

ODEs that show this behaviour are called *stiff equations*. An exact definition of this term is hard to give. Roughly speaking, their crucial property is that they have a transient part of the solution, which decays very rapidly. Looking at the ODE Eq. 10.152 directly, this might be seen from the large negative factor in front of y. Of course, in realistic examples, the exact solution of the problem will *not* be known, the ODE may look more complicated, and one will need to deal with a system of equations rather than with a single equation, so that it is much harder to deduce the "stiff" behaviour from the ODE directly. In applications, it is often apparent from the context whether a problem is stiff or not. Stiff equations take their name from problems in mechanics: a system with "stiff" springs, i.e., with large spring constants and/or strong friction, is typically described by a stiff ODE.

10.9.1 The test equation

In order to understand the problem better, and to decide which numerical methods best to apply, we need a simple but comprehensive test case for our numerical methods. To that end, we apply our methods to the *test equation*

$$y'(x) = \lambda y(x), \quad 0 \le x, \quad y(0) = 1, \tag{10.155}$$

where $\lambda \in \mathbb{C}$ is a constant, with $\operatorname{Re} \lambda < 0$. The exact solution is

$$y(x) = \exp(\lambda x). \tag{10.156}$$

This solution has *only* a transient part, which makes it ideal for our purposes.

(We are now dealing with an ODE for a *complex*-valued function y. This does not really need a generalization of our previous methods: We can always split y into a two-vector $(\operatorname{Re} y, \operatorname{Im} y)$, and thus rewrite our equations as a system of two first-order ODEs with real values. However, the complex-valued notation is convenient here. The nonzero imaginary part of λ allows us to include oscillatory solutions as well, with almost no added effort.)

Let us apply the Euler method to the test equation. The difference equation gives

$$w_{i+1} = w_i + h\, f(x_i, w_i) = w_i + \lambda h w_i = (1 + \lambda h) w_i. \tag{10.157}$$

With $w_0 = 1$, it is then clear that for all $i \in \mathbb{N}$,

$$w_i = (1 + \lambda h)^i. \tag{10.158}$$

Thus, if $|1 + \lambda h| > 1$, the approximation grows exponentially for large i, and is therefore "grossly wrong" as the exact solution vanishes for large x. If $|1 + \lambda h| < 1$, then $w_i \to 0$, which at least roughly resembles the behaviour of the exact solution.

For our other (explicit) approximation methods, one finds in generalization of Eq. 10.158,

$$w_i = \big(Q(\lambda h)\big)^i \tag{10.159}$$

with some function Q (for explicit methods, Q is in fact a polynomial). The approximation values grow or decay if $|Q(\lambda h)| > 1$ or $|Q(\lambda h)| < 1$, respectively.

This motivates us to define the *region of stability* of an approximation method:

$$\mathcal{R} := \big\{ \mu \in \mathbb{C} \,\big|\, |Q(\mu)| < 1 \big\}. \tag{10.160}$$

In order for the approximation to be "reasonable", i.e. to ensure $w_i \to 0$ as $i \to \infty$, we then need to choose our step size h so that $h\lambda \in \mathcal{R}$.

Specifically for Euler's method, we have $Q(\mu) = 1 + \mu$, and $\mathcal{R}$ is a disc with center -1 and radius 1. For higher-order Taylor methods, the regions of stability can be computed similarly; they are shown in Figure 10.2. For all examples of Runge-Kutta methods discussed in Section 10.6, the region agrees with that of the Taylor method of the same order. [6]

However, from this picture, it appears that *all* of our methods are vulnerable to the problems posed by stiff equations, as suggested also by the example above. For having a stable method for a large range of step sizes, we need to look at a different approach. It turns out that certain *implicit* multi-step methods provide an advantage here.

Let us, in particular, consider the *Implicit Trapezoidal method*, which is given by the difference equation

$$\begin{aligned} \mathbf{w}_0 &:= \alpha, \\ \mathbf{w}_{i+1} &= \mathbf{w}_i + \frac{h}{2}(\mathbf{f}(x_{i+1}, \mathbf{w}_{i+1}) + \mathbf{f}(x_i, \mathbf{w}_i)). \end{aligned} \tag{10.161}$$

Applying it to the test equation Eq. 10.155, we obtain

$$w_{i+1} = w_i + \frac{h}{2}(\lambda w_{i+1} + \lambda w_i) = (1 + \frac{\lambda h}{2})w_i + \frac{\lambda h}{2} w_{i+1}. \tag{10.162}$$

We can solve this for w_{i+1}:

$$w_{i+1} = \frac{2 + \lambda h}{2 - \lambda h} w_i. \tag{10.163}$$

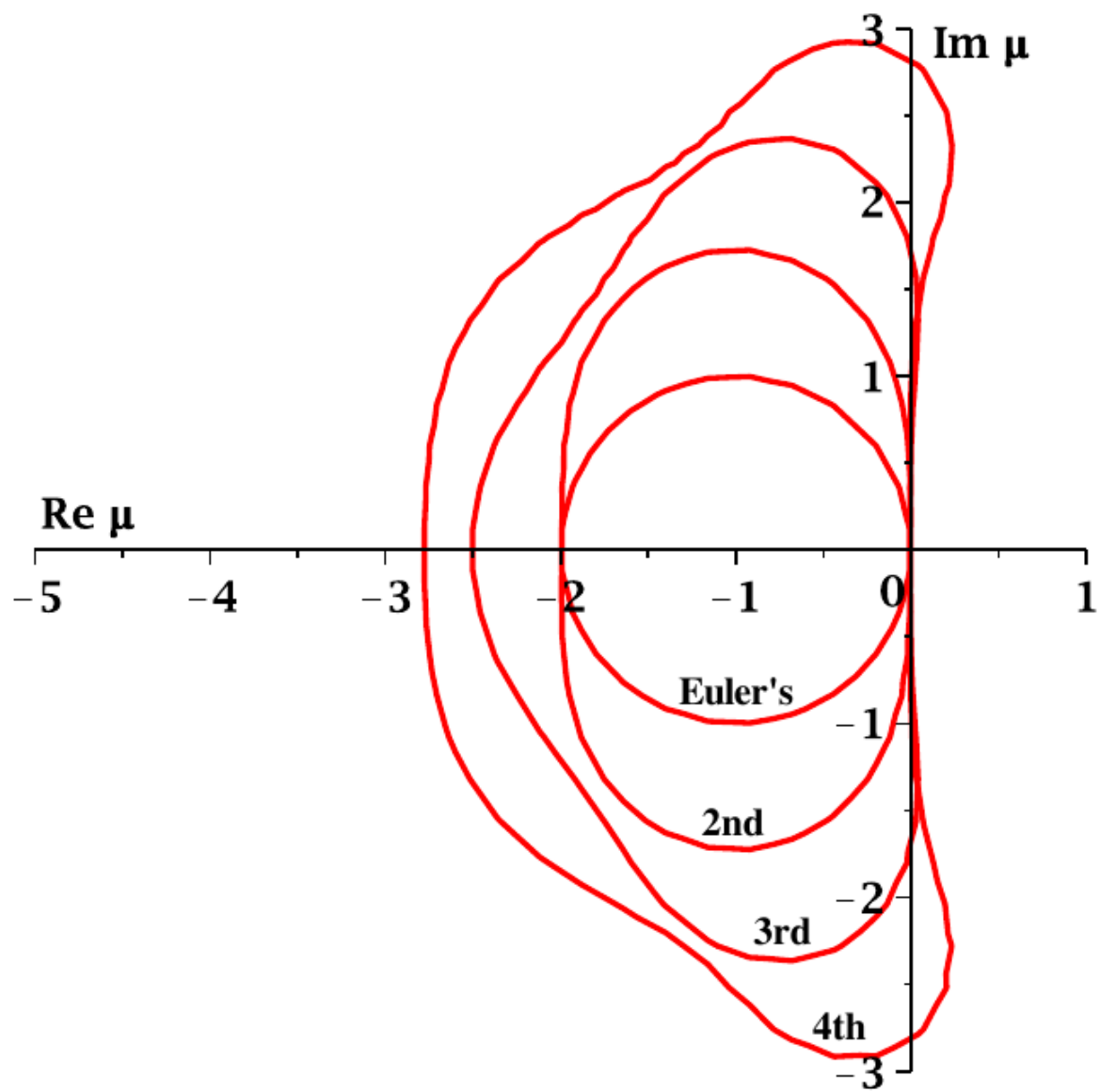[6]This is true for all of our examples, but not for all Runge-Kutta methods in general.

Figure 10.2: Region of stability for Taylor methods

Thus, the result is again of the form Eq. 10.159, with

$$Q(\mu) = \frac{2+\mu}{2-\mu}. \tag{10.164}$$

Since

$$|Q(\mu)| < 1 \quad \Leftrightarrow \quad |2+\mu|^2 < |2-\mu|^2$$
$$\Leftrightarrow \quad 4 + 4\operatorname{Re}\mu + |\mu|^2 < 4 - 4\operatorname{Re}\mu + |\mu|^2 \quad \Leftrightarrow \quad \operatorname{Re}\mu < 0, \tag{10.165}$$

the region of stability $\mathcal{R}$ is just the left half plane. This is the maximum we could hope for: the implicit trapezoidal method gives reasonable results on stiff problems for all step sizes.

Methods of this kind, i.e., whose region of stability $\mathcal{R}$ contains the entire left half plane, are called *absolutely stable* or *A-stable*.

One can apply the above stability analysis to other implicit methods as well, with some added complication – namely, when solving the difference equation as in Eq. 10.163, we may in general find several solutions. Figure 10.3 show the resulting regions of stability in some examples. Not all implicit methods are A-stable. In fact, there are no A-stable multistep methods of order $O(h^3)$ or above. Sometimes, one uses a class of implicit multistep methods known as *backward differentiation methods (BDM)*, which exist for arbitrary error order and come at least close to A-stability; we do not discuss them here.
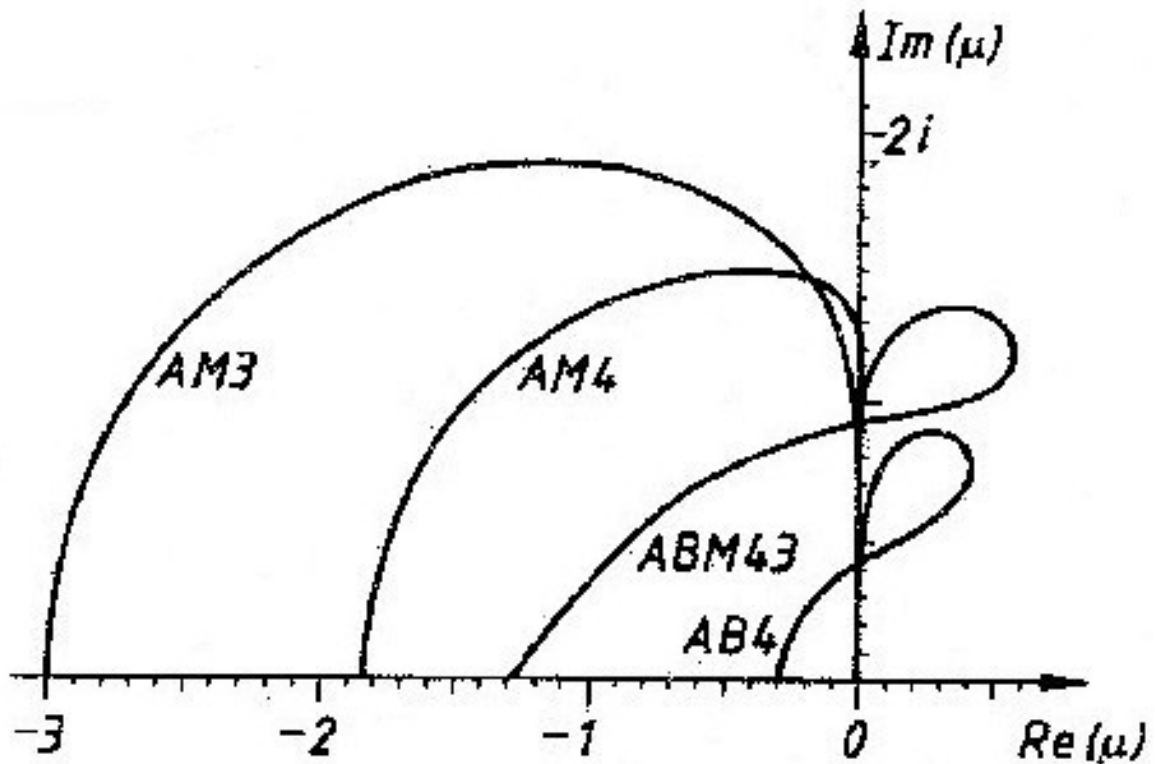

Figure 10.3: Region of stability for Adams-Bashforth and Adams-Moulton methods, AB4: Adams-Bashforth 4-step; AM3/AM4: Adams-Moulton 3-step/4-step; ABM43: Adams-Bashforth-Moulton predictor-corrector, 4/3-step. Graphics taken from

10.9.2 Implementing the Implicit Trapezoidal Method

Our analysis so far suggests that we should use the Implicit Trapezoidal method for the numerical treatment of stiff ODEs. However, since the difference equation of the method defines w_{i+1} only implicitly, this poses conceptual problems.

In the extremely simple case of the test equation Eq. 10.155, we were able to solve the difference equation explicitly for w_{i+1}; see Eq. 10.163. We can generalize this to the case of a *linear* ODE:

$$y'(x) = \ell(x)y(x) + g(x), \quad a \leq x \leq b, \quad y(a) = \alpha. \tag{10.166}$$

Here ℓ, g are continuous functions from $[a, b]$ to $\mathbb{R}$. In this case, the difference equation Eq. 10.161 of the implicit trapezoidal method reads,

$$w_{i+1} = w_i + \frac{h}{2}\Big(\ell(x_{i+1})w_{i+1} + g(x_{i+1}) + \ell(x_i)w_i + g(x_i)\Big). \tag{10.167}$$

Much like in Eq. 10.163, this can be solved for w_{i+1}:

$$w_{i+1}\Big(1 - \frac{h}{2}\ell(x_{i+1})\Big) = \Big(1 + \frac{h}{2}\ell(x_i)\Big) + \frac{h}{2}g(x_{i+1}) + \frac{h}{2}g(x_i)$$
$$\Rightarrow \quad w_{i+1} = \frac{2 + h\ell(x_i)}{2 - h\ell(x_{i+1})}w_i + h\frac{g(x_{i+1}) + g(x_i)}{2 - h\ell(x_{i+1})}. \tag{10.168}$$

Thus, for linear ODEs as in Eq. 10.166, the implicit trapezoidal method can be applied with an explicit difference equation.

For a nonlinear ODE, it will generally not be possible to solve the difference equation symbolically. However, we can try to solve it *numerically*. As an approach for numerically solving nonlinear equations, we shall use Newton Iteration.

As a reminder: [7] Let $F : \mathbb{R} \to \mathbb{R}$ be twice differentiable; suppose we are looking for a solution x of the equation

$$F(x) = 0. \tag{10.169}$$

This solution is found with Newton's method as follows. Starting with x_0 sufficiently close to a solution, one recursively defines the sequence

$$x_k := x_{k-1} - \frac{F(x_{k-1})}{F'(x_{k-1})}. \tag{10.170}$$

Then $x_k \to x_\infty$ with $F(x_\infty) = 0$.

When implementing this on a computer, one cannot pass to the limit $k \to \infty$, but needs to stop the iteration when a sufficient precision is reached. In practice, one usually takes the magnitude of the difference term on the r.h.s. of Eq. 10.170 as an indicator: The iteration is stopped when $|F(x_{k-1})/F'(x_{k-1})| < T$, where $T > 0$ is the desired tolerance level.

In our situation of the implicit trapezoidal method, we need to solve the equation

$$w_{i+1} = w_i + \frac{h}{2}(f(x_i, w_i) + f(x_{i+1}, w_{i+1})). \tag{10.171}$$

for w_{i+1}, where x_i, x_{i+1}, w_i are given numbers. In other words, we are looking for zeros of the function

$$F(\hat{w}) = \hat{w} - w_i - \frac{h}{2}(f(x_i, w_i) + f(x_{i+1}, \hat{w})). \tag{10.172}$$

Following Eq. 10.170, the following sequence should converge to the solution w_{i+1}:

$$\hat{w}_0 := w_i,$$
$$\hat{w}_k := \hat{w}_{k-1} - \underbrace{\frac{\hat{w}_{k-1} - w_i - \frac{h}{2}\big(f(x_i, w_i) + f(x_{i+1}, \hat{w}_k)\big)}{1 - \frac{h}{2}\frac{\partial f}{\partial y}(x_{i+1}, \hat{w}_{k-1})}}_{=:v}. \tag{10.173}$$

We would run this iteration until $|v| < T$.

---

[7]For more details on Newton's method for solving (single) nonlinear equations, see Section 3.5, or (Burden and Faires 2010, sec. 2.3).

Both in the case of linear and nonlinear equations, we have so far discussed a single ODE only. For applications in practice, we would need to generalize the methods to systems of ODEs. This is indeed possible, but requires a bit of thought.

In the linear case Eq. 10.166, one would consider a matrix-valued function ℓ and a vector-valued g. The difference equation can then still be solved symbolically, like in Eq. 10.168; however, the division by the factor $(2 - h\ell(x_{i+1}))$ needs to be replaced by a multiplication with an inverse matrix, or equivalently, by solving a system of linear equations. Likewise, in the nonlinear case, we can handle ODE systems but would need the multi-dimensional version of Newton's method (or, in other words, Newton's method for systems of nonlinear equations). We shall not discuss the details of this generalization here.

11 Boundary Value Problems

11.1 Fundamentals

In this last part of the course, we will consider *boundary problems (BVPs)* for ODEs, specifically second-order ODEs. Here, in contrast to initial value problems, one does not specify the value of the function y *and its derivative* at the left-hand side of the interval in question. Rather, one fixes the value of $y(x)$ at both the left- and the right-hand side. The BVP we will consider throughout is

$$y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta, \tag{11.1}$$

where $f : [a, b] \times \mathbb{R}^2 \to \mathbb{R}$ and $\alpha, \beta \in \mathbb{R}$.

Boundary value problems for first-order ODEs are not meaningful, so we need to consider at least ODEs of second order. One can as well formulate BVPs for higher-order ODEs, for systems of ODEs, or by specifying the derivative y' at the boundary rather than the function y itself. We will however not treat these cases here.

Our first question is whether a unique solution of Eq. 11.1 exists, before approximating it numerically. This question turns out to be more delicate than with IVPs. We first consider a few examples.

Example 11.1. Let us consider the very simple BVP

$$y'' = y, \quad 0 \le x \le 1, \quad y(0) = y(1) = 1. \tag{11.2}$$

The ODE $y'' = y$ itself (disregarding the boundary conditions for a moment) clearly has the two solutions $y(x) = e^x$ and $y(x) = e^{-x}$. From the theory of linear ODEs, one knows then that the general solution of the ODE is

$$y(x) = ce^x + de^{-x}, \quad c, d \in \mathbb{R}. \tag{11.3}$$

That is, every solution of $y'' = y$ has this form. Now inserting the boundary conditions, we have

$$1 = ce^0 + de^0 = c + d \quad \Leftrightarrow \quad d = 1 - c, \tag{11.4}$$

and

$$1 = ce^1 + de^{-1} = ce + \frac{d}{e} \quad \Leftrightarrow \quad 1 = ce + \frac{1-c}{e} = c\left(e - \frac{1}{e}\right) + \frac{1}{e}. \tag{11.5}$$

From Eq. 11.5, we find that the boundary conditions are satisfied if and only if

$$1 - \frac{1}{e} = c\left(e - \frac{1}{e}\right) \quad \Leftrightarrow \quad c = \frac{e - 1}{e^2 - 1} = \frac{1}{e + 1}, \tag{11.6}$$

of which we compute $d = 1 - c = \frac{e}{e+1}$. That means, the function

$$y(x) = \frac{e^x}{e + 1} + \frac{e^{1-x}}{e + 1} \tag{11.7}$$

is a solution of the BVP Eq. 11.2, and by our computation, it is the *only* solution of the BVP. In short, Eq. 11.2 has a unique solution.

Example 11.2. Now let us consider a slightly modified BVP,

$$y'' = -y, \quad 0 \le x \le 2\pi, \quad y(0) = y(2\pi) = 1. \tag{11.8}$$

Again, from the theory of linear ODEs, one knows that the general solution of the ODE (disregarding the boundary conditions) is

$$y(x) = c\cos(x) + d\sin(x), \quad c, d \in \mathbb{R}. \tag{11.9}$$

The boundary conditions demand that

$$1 = y(0) = c\cos(0) + d\sin(0) = c \Leftrightarrow c = 1, \tag{11.10}$$

and likewise

$$1 = y(2\pi) = c \Leftrightarrow c = 1. \tag{11.11}$$

However, the choice of d does not affect the boundary conditions! In fact, any function of the form

$$y(x) = \cos(x) + d\sin(x), \quad d \in \mathbb{R} \tag{11.12}$$

solves the BVP Eq. 11.8. This BVP has *many* solutions.

Example 11.3. Finally, consider another slight modification of our example:

$$y'' = -y, \quad 0 \le x \le 2\pi, \quad y(0) = 0, \ y(2\pi) = 1. \tag{11.13}$$

Again, the general solution of the ODE is given by Eq. 11.9. However, now our boundary conditions demand that

$$0 = y(0) = c, \quad \text{and } 1 = y(2\pi) = c, \tag{11.14}$$

which gives us a contradiction. In other words, the BVP Eq. 11.13 has *no solution*.

11.1.1 A criterion

We need a criterion that guarantees that a BVP has a unique solution. As is clear from the above examples, the Lipschitz condition which we previously considered does not suffice. The following theorem (Burden and Faires 2010 Theorem 11.1), which we quote here without proof, gives a sufficient (not a necessary) criterion.

Theorem 11.1. *Suppose that $f : [a,b] \times \mathbb{R}^2 \to \mathbb{R}$ is continuous, that the partial derivatives[1] $\partial f(x,y,y')/\partial y$ and $\partial f(x,y,y')/\partial y'$ exist and are continuous, and that**

1. $\dfrac{\partial f}{\partial y}(x,y,y') > 0$ for all $x \in [a,b]$, $y, y' \in \mathbb{R}$,

2. $\left| \dfrac{\partial f}{\partial y'}(x,y,y') \right| \le M$ for all $x \in [a,b]$, $y, y' \in \mathbb{R}$,

with a constant $M > 0$. Then, the boundary value problem

$$y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta \tag{11.15}$$

has a unique solution for any $\alpha, \beta \in \mathbb{R}$.

Briefly speaking, in order to obtain a unique solution, it suffices that $\partial f/\partial y$ is *positive* and $\partial f/\partial y'$ is *bounded*. The former condition was violated in our examples Eq. 11.8 and Eq. 11.13.

---

[1]*To explain the notation: These are the partial derivatives of f by its second and third argument, respectively.*

11.1.2 Linear ODEs

We will often consider the simplified, but still very relevant, case of boundary value problems for *linear* ODEs. By this, we mean a BVP of the form

$$y''(x) = p(x)y'(x) + q(x)y(x) + r(x), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta, \qquad (11.16)$$

where $p, q, r : [a, b] \to \mathbb{R}$. A uniqueness criterion for these can be given as follows.

Theorem 11.2. *Suppose that $p, q, r \in \mathcal{C}^0([a, b])$, and that $q(x) > 0$ for all $x \in [a, b]$. Then, the boundary value problem Eq. 11.16 has a unique solution for any $\alpha, \beta \in \mathbb{R}$.*

Proof. This follows immediately from Theorem 11.1. In the present case,

$$\begin{aligned}
f(x, y, y') &= p(x)y' + q(x)y + r(x), \\
\frac{\partial f(x, y, y')}{\partial y} &= q(x), \\
\frac{\partial f(x, y, y')}{\partial y'} &= p(x),
\end{aligned} \qquad (11.17)$$

and continuous functions on a bounded interval are always bounded. $\qquad \square$

11.1.3 Approximation methods

In the following sections, we will discuss three very different approximation methods for the solutions of BVPs:

- the *Shooting Method*, which makes use of the techniques for IVPs which we discussed in Chapter 10,

- the *Finite Difference Method*, which approximates the derivatives of the ODE with difference quotients,

- the *Rayleigh-Ritz Method*, which reformulates the BVP as a minimization problem of a certain integral.

The Finite Difference Method and the Rayleigh-Ritz method are particularly important: first, because they are adapted to the nature of boundary value problems; second, because they have generalizations to the approximation theory of partial differential equations.

11.2 The Shooting Method

As the first approximation method for the solutions of BVPs, we will discuss the *Shooting Method*. It works by transforming the boundary value problem into an initial value problem, and then using our known approximation methods for BVPs.

11.2.1 The idea

We will consider the BVP

$$y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta. \tag{11.18}$$

For applying our approximation methods for IVPs to the differential equation, we are missing one initial value, namely for $y'(a)$.

Let us, for the moment, just pick *some* fixed $\gamma \in \mathbb{R}$, and then consider the initial value problem

$$y_\gamma''(x) = f(x, y_\gamma(x), y_\gamma'(x)), \quad a \le x \le b, \quad y_\gamma(a) = \alpha, \ y_\gamma'(a) = \gamma. \tag{11.19}$$

(The solution $y_\gamma(x)$ will depend on our choice of γ, and we indicate this with the γ subscript.) We can solve the IVP Eq. 11.19 with one of our known approximation methods. This will yield an approximation value $w_N \approx y_\gamma(b)$. Now we can compare whether our choice of γ was reasonable: Namely, we can see whether $w_N \approx \beta$, or how far w_N is away from β. Depending on this result, we adjust our value γ, and run the approximation method for Eq. 11.19 again. We repeat this until $w_N \approx \beta$ to a reasonable precision; then the w_j will approximate $y(x_j)$, where y is the solution of the BVP Eq. 11.18.

11.2.2 Systematic Approach

The open point is how to choose the value γ. Of course, we want a systematic (i.e., algorithmic) way to find the optimum value.

To formalize this, let us consider the map

$$g : \mathbb{R} \to \mathbb{R}, \ \gamma \mapsto y_\gamma(b), \tag{11.20}$$

where $y_\gamma(x)$ is the exact solution of the IVP Eq. 11.19. We already have methods to compute g approximately (by using approximation methods for IVPs). We need to find γ such that $y_\gamma(x) = \beta$, i.e., such that

$$g(\gamma) = \beta. \tag{11.21}$$

In other words, we need to find an (approximate) solution to the equation Eq. 11.21. We will consider two cases:

In the simpler case, g is a *linear* function, i.e., $g(\gamma) = m\gamma + c$. In this case, Eq. 11.21 can be solved explicitly (once m and c are known), namely

$$\gamma = \frac{\beta - c}{m}. \tag{11.22}$$

It remains to determine the constants m and c.

More generally, g can be a *nonlinear* function, and in this case we need to solve Eq. 11.21 numerically. We will use Newton's method to that end. That is, we consider a sequence γ_k, recursively defined by

$$\gamma_k := \gamma_{k-1} - \frac{g(\gamma_{k-1}) - \beta}{g'(\gamma_{k-1})}, \tag{11.23}$$

which is expected to converge to the desired value γ. To that end, it remains to compute the derivative g' of the map g.

11.2.3 Linear Shooting

Let us first consider the linear case. Not very surprisingly, this arises when the underlying ODE is linear. That is, let us assume that our BVP is of the form

$$y''(x) = p(x)y'(x) + q(x)y(x) + r(x), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta, \tag{11.24}$$

with p, q, r being continuous functions on $[a, b]$.

It turns out to be convenient to consider *two* related IVPs, neither of which involves the value γ:

$$y_0''(x) = p(x)y_0'(x) + q(x)y_0(x) + r(x), \quad a \le x \le b, \quad y_0(a) = \alpha, \quad y_0'(a) = 0; \tag{11.25}$$

$$z'' = p(x)z'(x) + q(x)z(x), \quad a \le x \le b, \quad z(a) = 0, \quad z'(a) = 1. \tag{11.26}$$

Using their exact solutions $y_0(x)$ and $z(x)$, we can define

$$y_\gamma(x) := y_0(x) + \gamma z(x); \tag{11.27}$$

and by adding Eq. 11.25 and Eq. 11.26, we see that this y_γ is the (unique) solution of the IVP Eq. 11.19.

The solutions $y_0(x)$ and $z(x)$ can be found (approximately) by our known methods from Chapter 10. We can then compute the correct value of γ for our boundary value problem, as indicated in Eq. 11.22:

$$\gamma = \frac{\beta - y_0(b)}{z(b)}. \tag{11.28}$$

Re-inserting this value γ into Eq. 11.27, we have found the (approximate) solution of the BVP.

Algorithm 11.1 The Linear Shooting method with Runge-Kutta approximation

---

1: **function** LINEARSHOOTING(a, b, α, β, N)
2: $\qquad \to$ boundary values α, β; number of Runge-Kutta steps N
3: $\quad F := (x, \mathbf{u}) \mapsto (u^{(2)}, p(x)u^{(2)} + q(x)u^{(1)} + r(x)) \qquad \to$ Rewrite and solve IVP for y_0
4: $\quad \mathbf{y} := \text{RUNGEKUTTA}(F, a, b, (\alpha, 0), N) \qquad \to \mathbf{y} = (\mathbf{y}_0, \dots, \mathbf{y}_N)$, each $\mathbf{y}_j$ is a 2-vector
5: $\quad G := (x, \mathbf{u}) \mapsto (u^{(2)}, p(x)u^{(2)} + q(x)u^{(1)}) \qquad \to$ Rewrite and solve IVP for z
6: $\quad \mathbf{z} := \text{RUNGEKUTTA}(G, a, b, (0, 1), N) \qquad \to \mathbf{z} = (\mathbf{z}_0, \dots, \mathbf{z}_N)$, each $\mathbf{z}_j$ is a 2-vector
7: $\quad \gamma := \dfrac{\beta - y_N^{(1)}}{z_N^{(1)}} \qquad \to$ Determine correct initial value
8: $\quad$ **for** $i = 0, \dots, N$ **do**
9: $\qquad w_i := y_i^{(1)} + \gamma z_i^{(1)} \qquad \to$ compute solution of BVP
10: $\quad$ **end for**
11: $\quad$ **return** $(w_0, \dots, w_N)$
12: **end function**

Algorithm 11.1 summarizes the Linear Shooting Method. We first rewrite the IVP Eq. 11.25 for y_0 as a system of two first-order ODEs, and approximate its solution (lines 3–4). For concreteness' sake, we choose the classical Runge-Kutta method for the approximation. We do likewise with the IVP Eq. 11.26 for z, in lines 5–6. Now we can compute the correct value of γ (line 7), and combine the approximations for y_0 and z into the approximation of the BVP solution (lines 8–10).

11.2.4 Nonlinear Shooting

Now let us turn to the nonlinear case, i.e., to a generic BVP of the form

$$y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta. \tag{11.29}$$

As said before, we will use Newton's method for finding the correct value of γ. That means, we need to compute the sequence (γ_k) given by

$$\gamma_k := \gamma_{k-1} - \frac{g(\gamma_{k-1}) - \beta}{g'(\gamma_{k-1})}. \tag{11.30}$$

To that end, we need to know how to compute $g(\gamma)$ and $g'(\gamma)$ numerically.

Here finding $g(\gamma)$ is not difficult: We have $g(\gamma) = y_\gamma(b)$, where y_γ is the solution of the IVP

$$y_\gamma''(x) = f(x, y_\gamma(x), y_\gamma'(x)), \quad a \le x \le b, \quad y_\gamma(a) = \alpha, \ y_\gamma'(a) = \gamma. \tag{11.31}$$

We can approximate this solution numerically with any of our known methods – say, classical Runge-Kutta – and thus obtain an approximation of $g(\gamma)$.

However, for its derivative g', the right approach is much less obvious. We need to compute

$$g'(\gamma) = \frac{\partial}{\partial \gamma} y_\gamma(x)\Big|_{x=b}. \tag{11.32}$$

In order to find this value, we differentiate Eq. 11.31 by γ, using the chain rule.

$$\frac{\partial y_\gamma'}{\partial \gamma}(a) = 1. \tag{11.33}$$

This does not determine $\partial y_\gamma / \partial \gamma$ directly. However, it gives us a second-order IVP from which the derivative can be determined.

To make this more concrete, let us use the substitution

$$u^{(1)} = y_\gamma, \quad u^{(2)} = y_\gamma', \quad u^{(3)} = \frac{\partial y_\gamma}{\partial \gamma}, \quad u^{(4)} = \frac{\partial y_\gamma'}{\partial \gamma}. \tag{11.34}$$

We can then rewrite Eq. 11.31 *and* Eq. 11.33 into an IVP for 4 first-order ODEs:

$$\mathbf{u}' = \begin{pmatrix} u^{(2)} \\ f(x, u^{(1)}, u^{(2)}) \\ u^{(4)} \\ \frac{\partial f}{\partial y}(x, u^{(1)}, u^{(2)})\, u^{(3)} + \frac{\partial f}{\partial y'}(x, u^{(1)}, u^{(2)})\, u^{(4)} \end{pmatrix}, \quad a \le x \le b, \quad \mathbf{u}(a) = \begin{pmatrix} \alpha \\ \gamma \\ 0 \\ 1 \end{pmatrix}. \tag{11.35}$$

From this IVP, which can be treated, e.g., with the classical Runge-Kutta method, we can now read off

$$g(\gamma) = u^{(1)}(b), \quad g'(\gamma) = u^{(3)}(b). \tag{11.36}$$

This finally allow us to compute the Newton step Eq. 11.30.

Let us summarize the Nonlinear Shooting Method in Algorithm 11.2. Its structure is of the now well-known Newton iteration type. In each iteration of the loop, we solve the IVP Eq. 11.35 with the classical Runge-Kutta method. We then pick the approximation values at the rightmost mesh point and compute the next γ value from them. The usual methods of breaking the Newton loop are in place - either after sufficient precision is reached, or (with an error) after too many steps have been taken.

11.3 The Finite Difference Method

The *Finite Difference Method (FDM)* is another, very different method of approximating the solution of the BVP

$$y''(x) = f(x, y(x), y'(x)), \quad a \le x \le b, \quad y(a) = \alpha, \ y(b) = \beta. \tag{11.37}$$

This method does not refer back to our approximation methods for IVPs, but is adapted directly to BVPs.

Algorithm 11.2 The Nonlinear Shooting method with Runge-Kutta approximation

---

1: **function** $\textsc{NonlinearShooting}(a, b, \alpha, \beta, \gamma_0, N, T, M)$
2:       $\to$ boundary values α, β; start value γ_0; number of Runge-Kutta steps N;
3:       $\to$ tolerance T; maximum number of Newton iterations M
4:     $F := (x, \mathbf{u}) \mapsto \left(u^{(2)}, f(x, u^{(1)}, u^{(2)}), u^{(4)}, \frac{\partial f}{\partial y}(x, u^{(1)}, u^{(2)})\, u^{(3)} + \frac{\partial f}{\partial y'}(x, u^{(1)}, u^{(2)})\, u^{(4)}\right)$
5:     $\gamma := \gamma_0$; $k := 0$
6:     **while** $k \leq M$ **do**
7:         $\mathbf{w} := \textsc{RungeKutta}(F, a, b, (\alpha, \gamma, 0, 1), N)$
8:             $\to$ here $\mathbf{w} = (\mathbf{w}_0, \ldots, \mathbf{w}_N)$, each $\mathbf{w}_j$ is a 4-vector
9:         $v := \dfrac{w_N^{(1)} - \beta}{w_N^{(3)}}$; $\gamma := \gamma - v$
10:        **if** $|v| < T$ **then**
11:            **break**
12:        **end if**
13:        $k := k + 1$
14:     **end while**
15:     **return** $(w_0^{(1)}, \ldots, w_N^{(1)})$
16: **end function**

11.3.1 The idea

Similar to before, we divide our interval with equally spaced mesh points, $x_0, \ldots, x_{N+1}$, where $x_0 = a$ and $x_{N+1} = b$.[2] As before, we want to approximate $y(x_i)$ with a value w_i ($i = 1, \ldots, N$).

The main idea of the FDM is to approximate the derivatives $y'(x_i)$ and $y''(x_i)$ with difference quotients ("finite differences"). For example, we could use

$$y'(x_i) \approx \frac{w_{i+1} - w_i}{h}, \tag{11.38}$$

but we will see an even better choice later. A similar approximation will need to be found for $y''(x_i)$.

In this way, the ODE is transformed into a system of equations for $w_1, \ldots, w_N$, and the boundary conditions are expressed by $w_0 = \alpha$, $w_{N+1} = \beta$.

11.3.2 Centred difference formulas

As a first step, we will derive an improved version of the difference quotient approximation Eq. 11.38, in which the error term was of order $O(h)$. Let us assume that the solution $y(x)$ of the BVP has three continuous derivatives. We write down second-order Taylor expansions of $y(x_{i+1})$ and $y(x_{i-1})$, cf. Theorem A.1:

$$4y(x_{i+1}) \;=\; y(x_i + h) = y(x_i) + h y'(x_i) + \frac{h^2}{2} y''(x_i) + \frac{h^3}{6} y'''(\xi_+); \tag{11.39}$$

$$y(x_{i-1}) = y(x_i - h) = y(x_i) - h y'(x_i) + \frac{h^2}{2} y''(x_i) - \frac{h^3}{6} y'''(\xi_-) \tag{11.40}$$

with some $\xi_+ \in [x_i, x_{i+1}]$ and $\xi_- \in [x_{i-1}, x_i]$. Subtracting Eq. 11.40 from Eq. 11.39, we obtain

$$y(x_{i+1}) - y(x_{i-1}) = 2h y'(x_i) + \frac{h^3}{6} \underbrace{\left(y'''(\xi_+) + y'''(\xi_-)\right)}_{2 y'''(\xi)}. \tag{11.41}$$

Here $\xi \in [x_{i-1}, x_{i+1}]$ is some point obtained from the intermediate value theorem. Solving for $y'(x_i)$, we find

---

[2]Note the change of convention: Our step size is now $h = (b - a)/(N + 1)$, and we have $N + 2$ (not $N + 1$) mesh points, of which N are inside the interval (a, b). This convention, which we shall use from now on, is very useful for boundary value problems: the function y needs to be determined at $x_1, \ldots, x_N$ but is already known at x_0 and x_{N+1}.

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h} - \frac{h^2}{6} y'''(\xi). \tag{11.42}$$

This is known as a *centred difference formula*. Note that the "error term" here is of order $O(h^2)$ which is better than the $O(h)$ one obtains in Eq. 11.38.

We can obtain a similar formula for $y''(x_i)$ as well, using a third-order Taylor expansion of y. See (Burden and Faires 2010, sec. 4.1, Eq. (4.9)) for details. The result is

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} - \frac{h^2}{12} y''''(\xi) \tag{11.43}$$

with some $\xi \in [x_{i-1}, x_{i+1}]$.

11.3.3 Recipe for the Finite Difference Method

We can now formulate an outline of the FDM in our context. Starting from a boundary value problem of the form Eq. 11.37, we consider the N equations

$$y''(x_i) = f(x_i, y(x_i), y'(x_i)), \quad i = 1, \dots, N. \tag{11.44}$$

In order to obtain an approximate solution, we manipulate the equations as follows:

- Replace all occurrences of $y(x_i)$ with w_i.
- Replace all occurrences of $y'(x_i)$ with $\dfrac{w_{i+1} - w_{i-1}}{2h}$; cf. Eq. 11.42.
- Replace all occurrences of $y''(x_i)$ with $\dfrac{w_{i+1} - 2w_i + w_{i-1}}{h^2}$; cf. Eq. 11.43.
- Set $w_0 = \alpha$, $w_{N+1} = \beta$.

This will leave us with a system of N equations for the N variables $w_1, \dots, w_N$. This equation system can either be linear, in which case we can apply algorithms for solving linear equation systems (see Chapter 5); or it can be nonlinear, in which case we need Newton's method in several variables (see Chapter 4) to approximate the solution.

The error order of this approximation scheme is determined by the remainder term in the centred difference formulas, Eqs. Eq. 11.42 and Eq. 11.43; namely, the method is of order $O(h^2)$. However, we will not prove this here.

11.3.4 The Linear Finite Difference Method

Let us first apply the recipe to a *linear* ODE, that is, to a boundary value problem of the form

$$y''(x) = p(x)y'(x) + q(x)y(x) + r(x), \quad a \le x \le b, \quad y(a) = \alpha,\ y(b) = \beta. \tag{11.45}$$

In this case, Eq. Eq. 11.44 reads

$$y''(x_i) = p(x_i)y'(x_i) + q(x_i)y(x_i) + r(x_i), \quad i = 1, \dots, N. \tag{11.46}$$

By the substitution rules mentioned above, and with the abbreviations $p_i := p(x_i)$, $q_i := q(x_i)$, $r_i := r(x_i)$, we get:

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} = p_i \frac{w_{i+1} - w_{i-1}}{2h} + q_i w_i + r_i, \quad i = 1, \dots, N. \tag{11.47}$$

Multiplying with h^2, and bringing all w_j to the left-hand side, we can rewrite this as

$$w_{i+1} - 2w_i + w_{i-1} - \frac{h}{2}p_i w_{i+1} + \frac{h}{2}p_i w_{i-1} - h^2 q_i w_i = h^2 r_i, \quad (11.48)$$

or

$$\left(-1 + \frac{h}{2}p_i\right)w_{i+1} + (2 + h^2 q_i)w_i + \left(-1 - \frac{h}{2}p_i\right)w_{i-1} = -h^2 r_i \quad (11.49)$$

for $i = 1, \dots, N$.

Note here that w_{i-1} may be w_0 (if $i = 1$) and w_{i+1} may be w_{N+1} (if $i = N$). So not all w_j are variables, some need to be replaced with the boundary values α and β. Keeping this in mind, we can rewrite the equation system Eq. 11.49 in matrix form:

$$\underbrace{\begin{pmatrix} 2 + h^2 q_1 & -1 + \frac{h}{2}p_1 & 0 & \cdots & 0 \\ -1 - \frac{h}{2}p_2 & 2 + h^2 q_2 & -1 + \frac{h}{2}p_2 & 0 & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & -1 - \frac{h}{2}p_{N-1} & 2 + h^2 q_{N-1} & -1 + \frac{h}{2}p_{N-1} \\ 0 & \cdots & 0 & -1 - \frac{h}{2}p_N & 2 + h^2 q_N \end{pmatrix}}_{=:\mathbf{A}}\mathbf{w}$$
$$= \underbrace{\begin{pmatrix} -h^2 r_1 + (1 + \frac{h}{2}p_1)\alpha \\ -h^2 r_2 \\ \vdots \\ -h^2 r_{N-1} \\ -h^2 r_N + (1 - \frac{h}{2}p_N)\beta \end{pmatrix}}_{=:\mathbf{b}}. \quad (11.50)$$

The matrix $\mathbf{A}$ is tridiagonal. Therefore, we can use fast algorithms (e.g. Crout factorization) in order to solve the linear equation system, and obtain the vector $\mathbf{w} = (w_1, \dots, w_N)$ in $O(N)$ time. This directly gives us the required approximation of the solution $y(x)$.

11.3.5 The Nonlinear Finite Difference Method

Let us now discuss the case of a completely generic, possibly nonlinear ODE. As before, we set out from the N equations in Eq. 11.44. Again we substitute the function y and its derivatives with w_i and their finite differences. However, in absence of more information about the function f, we end up with just

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} = f\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right), \quad i = 1, \dots, N. \quad (11.51)$$

We can rewrite this in a more convenient form:

$$0 = \underbrace{-w_{i+1} + 2w_i - w_{i-1} + h^2 f\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right)}_{=:F^{(i)}(\mathbf{w})} \quad (11.52)$$

for $i = 1, \dots, N$. Our task is now to solve the nonlinear equation system $\mathbf{F}(\mathbf{w}) = 0$ for $\mathbf{w} = (w_1, \dots, w_N)$, where it is understood that $w_0 = \alpha$ and $w_{N+1} = \beta$.

We will do this with the N-dimensional version of Newton's method, as discussed in Chapter 4. To that end, we need to compute the Jacobian matrix of $\mathbf{F}$; that is, we need to compute all the partial derivatives $\partial F^{(i)}/\partial w^{(j)}$. Fortunately, most of these turn out to vanish. We compute from Eq. 11.52,

$$\frac{\partial F^{(i)}}{\partial w^{(i+1)}} = -1 + \frac{h}{2}\frac{\partial f}{\partial y'}\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right) \quad \text{for } i = 1, \dots, N-1, \quad (11.53)$$

$$\frac{\partial F^{(i)}}{\partial w^{(i-1)}} = -1 - \frac{h}{2}\frac{\partial f}{\partial y'}\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right) \quad \text{for } i = 2, \dots, N, \quad (11.54)$$

$$\frac{\partial F^{(i)}}{\partial w^{(i)}} = 2 + h^2 \frac{\partial f}{\partial y}\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right) \quad \text{for } i = 1, \dots, N. \tag{11.55}$$

All other partial derivatives are 0. That is, the Jacobian matrix $\mathbf{J} = \partial \mathbf{F} / \partial \mathbf{w}$ is tridiagonal. This allows us to solve the linear system involved in each step of the Newton iteration very quickly.

For doing the Newton iteration, we need a sensible start value for the vector $\mathbf{w}$. A common choice is to place the points $(x_i, w^{(i)})$ on a straight line from (a, α) to (b, β):

$$w_0^{(i)} = \alpha + ih \frac{\beta - \alpha}{b - a}. \tag{11.56}$$

Algorithm 11.3 The Nonlinear Finite Difference method

1: **function** NONLINEARFINITEDIFFERENCE$(a, b, \alpha, \beta, N, M)$
2:           $\rightarrow$ boundary values α, β; number of mesh points N; max. Newton iterations M
3:     $h := (b - a)/(N + 1)$; $k := 1$
4:     $w^{(i)} := \alpha + ih \frac{\beta - \alpha}{b - a}$ $(i = 1, \dots, N)$ $\rightarrow$ Set start value for $\mathbf{w}$
5:     **while** $k \leq M$ **do**
6:         Compute $\mathbf{F}(\mathbf{w})$ and $\mathbf{J}(\mathbf{w})$
7:         Solve $\mathbf{J}(\mathbf{w})\mathbf{v} = \mathbf{F}(\mathbf{w})$ for $\mathbf{v}$ using e.g. Crout factorization
8:         $\mathbf{w} := \mathbf{w} - \mathbf{v}$
9:         **if** $|\mathbf{v}| < T$ **then**
10:             **break**
11:         **end if**
12:         $k := k + 1$
13:     **end while**
14:     **return w**
15: **end function**

All that's left to do is to assemble the algorithm. This is done in Algorithm 11.3. The pattern follows our usual approach to Newton's method.

In each step of the Newton iteration, we need to solve one linear equation system. We can use e.g. the Crout factorization algorithm ((Burden and Faires 2010) Algorithm 6.7) to this end — this makes use of the fact that the Jacobian is tridiagonal, and allows every Newton step to run in only $O(N)$ time.

11.4 The Rayleigh-Ritz Method

The third, and radically different, approximation method for BVPs that we will discuss is called the *Rayleigh-Ritz method.*

11.4.1 Motivation

We start with a brief motivation originating in physics, or in engineering. Consider a beam of length L, fixed at both end points, which is deflected under its own weight. Denote with $y(x)$, $0 \leq x \leq L$, the deflection of the beam at point x. There are two complementary approaches of finding $y(x)$.

a) One may think of the problem in terms of a *local equilibrium of forces.* Several types of forces act at each point x of the beam - the gravitational force (downwards), but also forces relating to the elasticity of the beam, which pull the deflected beam upwards. In the rest position of the beam, the sum of these forces will be 0 at each point. This leads us to an ODE of the form $y''(x) = f(x, y(x), y'(x))$ (we do not specify the function f further here). The ends of the beam are fixed, so we are enforcing $y(0) = y(L) = 0$. Thus, we are dealing with a boundary value problem for an ODE.

b) Particularly in physics, one alternatively thinks of the situation as an *energy minimization problem.* The total energy of the beam will consist of its energy in the gravitational field, and of the energy

associated with elasticity. We can associate with each point of the beam an *energy density*, λ, which depends on the deflection y. It will roughly have the form

$$\lambda = cy'(x)^2 - dy(x) + \dots \tag{11.57}$$

with constants c and d, where the term proportional to $(y'(x))^2$ comes from elasticity, and the term proportional to y from gravitation. (We are not interested in the form of λ in detail here). The rest position of the beam is now found by the requirement that the *total energy* of the beam is minimal. This total energy is found by integrating λ over the length of the beam:

$$E = \int_0^L \lambda(y(x), y'(x)) \, \mathrm{d}x \tag{11.58}$$

So we are left with the problem of finding the function $y(x)$ that minimizes the integral Eq. 11.58, while satisfying the boundary conditions $y(0) = y(L) = 0$.

This example – which is added here only for illustration – raises an interesting mathematical question: Can we, under more general conditions, rewrite a boundary value problem for an ODE equivalently as a minimization problem for an integral expression? If yes, we can try to approximate the integral expression numerically (rather than the BVP), which may offer new options for numerical methods.

11.4.2 Equivalence of BVPs with minimization problems

In this section, we will specialize to a boundary value problems of the following form (a so-called *Sturm-Liouville problem*):

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}y}{\mathrm{d}x}(x)\right) + q(x)y(x) = f(x), \quad 0 \le x \le 1, \quad y(0) = y(1) = 0. \tag{11.59}$$

Here p, q and f are sufficiently smooth functions from $[0,1]$ to $\mathbb{R}$; for details see below. We can reformulate this BVP as a minimization problem for an integral expression: The solution $y(x)$ minimizes the integral

$$I[\phi] = \int_0^1 \left(p(x)(\phi'(x))^2 + q(x)\phi(x)^2 - 2f(x)\phi(x)\right)dx; \tag{11.60}$$

over all functions ϕ that satisfy the boundary conditions that is, if ϕ is any function satisfying the boundary conditions, then one has $I[y] \le I[\phi]$.

More precisely, let $\mathcal{C}_0^2[0,1]$ denote the space of $\mathcal{C}^2$ functions on $[0,1]$ that vanish at the endpoints of the interval. One has:

Theorem 11.3. *Let $p \in \mathcal{C}^1[0,1]$, and $q, f \in \mathcal{C}[0,1]$. Further, suppose that there exists a constant $\delta > 0$ such that*

$$\forall x \in [0,1]: \quad p(x) \ge \delta, \ q(x) \ge 0. \tag{11.61}$$

Then, for any function $y \in \mathcal{C}_0^2[0,1]$, the following conditions are equivalent.

   (i) y is the unique solution of the boundary value problem in Eq. 11.59.

   (ii) y is the unique function in $\mathcal{C}_0^2[0,1]$ which minimizes the integral $I[y]$ in Eq. 11.60.

Proof. Proof. Here we prove only (ii) $\implies$ (i), neglecting the aspect of uniqueness. For a full proof, see (Burden and Faires 2010 Theorem 11.4) and references quoted there.

If y minimizes the integral $I[y]$, then for any fixed $u \in \mathcal{C}_0^2[0,1]$, the function

$$\epsilon \mapsto I[y + \epsilon u] \tag{11.62}$$

has a minimum at $\epsilon = 0$. Therefore, its derivative at $\epsilon = 0$ must vanish:

$$0 = \frac{\mathrm{d}}{\mathrm{d}\epsilon} I[y + \epsilon u]\Big|_{\epsilon=0}$$

$$= \frac{\mathrm{d}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} \int_0^1 \Big(p(x)(y'(x) + \epsilon u'(x))^2 + q(x)(y(x) + \epsilon u(x))^2 - 2(y(x) + \epsilon u(x))f(x)\Big) dx$$

$$= \int_0^1 \Big(2p(x)(y'(x) + \epsilon u'(x))u'(x) + 2q(x)(y(x) + \epsilon u(x))u(x) - 2u(x)f(x)\Big)\Big|_{\epsilon=0} dx$$

$$= 2\int_0^1 \Big(p(x)y'(x)u'(x) + q(x)y(x)u(x) - u(x)f(x)\Big) dx. \tag{11.63}$$

We integrate by parts in the first term of the integral, $(p(x)y'(x))u'(x)$, noting that the boundary terms vanish since $u(0) = u(1) = 0$. This gives

$$0 = \int_0^1 \Big(-(p(x)y'(x))' + q(x)y(x) - f(x)\Big)u(x)\, dx \tag{11.64}$$

Since this holds for all $u \in \mathcal{C}_0^2[0,1]$, and since all functions under the integral sign are continuous, we can conclude that

$$0 = -(py')' + qy - f \tag{11.65}$$

on the entire interval $[0,1]$, which is what we wanted to show. $\qquad\square$

11.4.3 Approximating the integral

The Rayleigh-Ritz method makes use of the equivalence stated in Theorem 11.3. The idea is to approximate the function that minimizes the integral $I[\phi]$, rather than approximating the boundary value problem directly. That is, we try to find functions ϕ that make $I[\phi]$ as small as possible.

Of course, we cannot try *all* functions in the infinite dimensional vector space $\mathcal{C}_0^2[0,1]$. Rather we choose a set of N linearly independent functions ("basis functions"[3]) ϕ_i with $\phi_i(0) = \phi_i(1) = 0$, and use an arbitrary linear combination

$$\phi(x) = \sum_{i=1}^N c_i \phi_i(x) \tag{11.66}$$

of these functions as our trial function. The coefficients c_i are to minimize the integral. There is a large freedom of choice for the functions ϕ_j, and this can be exploited to adapt the approximation method to our needs. We leave the choice of ϕ_j open for the moment, and will fix them later on.

Substituting the trial function Eq. 11.66 into $I[\phi]$ in Eq. 11.60 gives

$$I[\phi] = \int_0^1 \left(p(x)\left(\sum_{i=1}^N c_i \phi_i'(x)\right)^2 + q(x)\left(\sum_{i=1}^N c_i \phi_i(x)\right)^2 - 2f(x)\sum_{i=1}^N c_i \phi_i(x)\right) dx. \tag{11.67}$$

We now minimize over the real parameters c_i. They will be fixed by the necessary requirement for having an extremum, namely, that all partial derivatives vanish,

$$0 = \frac{\partial I[\phi]}{\partial c_j} = \int_0^1 \left(2p(x)\sum_{i=1}^N c_i \phi_i'(x)\phi_j'(x) + 2q(x)\sum_{i=1}^N c_i \phi_i(x)\phi_j(x) - 2f(x)\phi_j(x)\right) dx \tag{11.68}$$

for all $j = 1, \ldots, N$. This is a linear system of the form

$$\sum_{i=1}^N A^{(j,i)}c^{(i)} = b^{(j)}, \quad j = 1, \ldots, N, \tag{11.69}$$

---

[3]While these are usually called "basis functions", e.g. in (Burden and Faires 2010), they are of course not a basis of the space $\mathcal{C}_0^2[0,1]$.

where

$$A^{(j,i)} = \int_0^1 \Big(p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x) \Big) dx, \qquad (11.70)$$

$$b^{(j)} = \int_0^1 f(x)\phi_j(x)dx. \qquad (11.71)$$

Once the basis functions ϕ_j are chosen, we can compute the components of $\mathbf{A}$ and $\mathbf{b}$, solve the linear system Eq. 11.69 for $\mathbf{c}$, and then obtain the approximation function ϕ from Eq. 11.66.

11.4.4 Choosing the basis functions

The question is now how to choose the functions ϕ_j so that we get a reasonable numerical approximation of the solution $y(x)$, which is furthermore fast to compute. Our requirements are:

- The ϕ_j need to satisfy the boundary conditions: $\phi_j(0) = \phi_j(1) = 0$.

- They should be sufficiently smooth. Note that, for purposes of numerical approximation, it is not be necessary to require two continuous derivatives (as for the ODE solution). Since we deal only with integral expressions in ϕ_j and ϕ_j', it should suffice if ϕ_j has one derivative that exists almost everywhere and is piecewise continuous, or at least integrable.

- They should be designed so that their linear combinations can approximate a wide range of functions. (This may seem a bit vague; but e.g. choosing all the ϕ_j with support in the interval $[0, \frac{1}{2}]$ would clearly be a bad idea.)

- They should be simple enough so that the integrals for $A^{(i,j)}$ and $b^{(j)}$ in Eq. 11.70 and Eq. 11.71 can reasonably be evaluated - if possible, explicitly.

- If possible, not many of them should have overlapping support, so that the matrix $\mathbf{A}$ is sparse, i.e., many of its entries are zero. This will make the linear equation system in Eq. Eq. 11.69 fast to solve.

A simple choice of basis functions, satisfying the above requirements, are the piecewise linear functions

$$\phi_i(x) = \begin{cases} 0 & \text{if } 0 \le x \le x_{i-1}; \\ \frac{1}{h_{i-1}}(x - x_{i-1}) & \text{if } x_{i-1} < x \le x_i; \\ \frac{1}{h_i}(x_{i+1} - x) & \text{if } x_i < x \le x_{i+1}; \\ 0 & \text{if } x_{i+1} < x \le 1 \end{cases} \qquad (11.72)$$

for some conveniently chosen mesh points

$$0 = x_0 < x_1 < x_2 < \cdots < x_N < x_{N+1} = 1. \qquad (11.73)$$

The h_i are the distances between neighbouring mesh points, $h_i = x_{i+1} - x_i$; note that the mesh points need not be equally spaced. These basis functions ϕ_i have simple piecewise constant derivatives,

$$\phi_i'(x) = \begin{cases} 0 & \text{if } 0 < x < x_{i-1}, \\ \frac{1}{h_{i-1}} & \text{if } x_{i-1} < x < x_i, \\ -\frac{1}{h_i} & \text{if } x_i < x < x_{i+1}, \\ 0 & \text{if } x_{i+1} < x < 1. \end{cases} \qquad (11.74)$$

A particular simplifying feature of this set of basis functions is that only neighbouring functions have any overlap, i.e.,

$$\phi_i(x)\phi_j(x) = 0 \quad \text{and} \quad \phi_i'(x)\phi_j'(x) = 0 \quad \text{unless } j \in \{i-1, i, i+1\}. \qquad (11.75)$$

This implies that the matrix $\mathbf{A}$ is tridiagonal. To calculate the entries of $\mathbf{A}$ and $\mathbf{b}$ in Eq. 11.70 and Eq. 11.71, the following integrals need to be evaluated:

$$
\begin{aligned}
Q_{1,i} &= \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i)q(x)dx, \\
Q_{2,i} &= \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 q(x)dx, \\
Q_{3,i} &= \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 q(x)dx, \\
Q_{4,i} &= \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)dx, \\
Q_{5,i} &= \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1})f(x)dx, \\
Q_{6,i} &= \frac{1}{h_i} \int_{x_i}^{x_{i+1}} (x_{i+1} - x)f(x)dx.
\end{aligned}
\tag{11.76}
$$

Then

$$
\begin{aligned}
A^{(i,i)} &= Q_{4,i} + Q_{4,i+1} + Q_{2,i} + Q_{3,i}, & i &= 1, \ldots, N, \\
A^{(i,i+1)} &= -Q_{4,i+1} + Q_{1,i}, & i &= 1, \ldots, N-1, \\
A^{(i,i-1)} &= -Q_{4,i} + Q_{1,i-1}, & i &= 2, \ldots, N, \\
b^{(i)} &= Q_{5,i} + Q_{6,i}, & i &= 1, \ldots, N.
\end{aligned}
\tag{11.77}
$$

One way to evaluate the $6N$ integrals is to approximate the functions $q(x)$, $p(x)$, and $f(x)$ by their linear interpolating polynomials. That is, we write

$$
q(x) = \sum_{i=0}^{N+1} q(x_i)\phi_i(x) + O(h^2),
\tag{11.78}
$$

and similarly for p and f, where the ϕ_i are as given above for $i = 1, \ldots, N$, and

$$
\begin{aligned}
\phi_0(x) &= \begin{cases} \frac{x_1 - x}{x_1} & \text{if } 0 \le x \le x_1, \\ 0 & \text{elsewhere}; \end{cases} \\
\phi_{N+1}(x) &= \begin{cases} \frac{x - x_N}{1 - x_N} & \text{if } x_N \le x \le 1, \\ 0 & \text{elsewhere}. \end{cases}
\end{aligned}
\tag{11.79}
$$

The integrals are now trivial to evaluate, for example

$$
\begin{aligned}
Q_{1,i} &= \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i)q(x)dx \\
&\approx \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i)\left(q(x_i)\frac{x_{i+1} - x}{h_i} + q(x_{i+1})\frac{x - x_i}{h_i}\right)dx \\
&= \frac{h_i}{12}\left(q(x_i) + q(x_{i+1})\right).
\end{aligned}
\tag{11.80}
$$

Similarly we find[4]

$$
\begin{aligned}
Q_{2,i} &\approx \frac{h_{i-1}}{12}\left(3q(x_i) + q(x_{i-1})\right), \\
Q_{3,i} &\approx \frac{h_i}{12}\left(3q(x_i) + q(x_{i+1})\right), \\
Q_{4,i} &\approx \frac{1}{2h_{i-1}}\left(p(x_i) + p(x_{i-1})\right), \\
Q_{5,i} &\approx \frac{h_{i-1}}{6}\left(2f(x_i) + f(x_{i-1})\right), \\
Q_{6,i} &\approx \frac{h_i}{6}\left(2f(x_i) + f(x_{i+1})\right).
\end{aligned}
\tag{11.81}
$$

---

[4]Note that Burden/Faires (Burden and Faires 2010) reports the formula for $Q_{4,i}$ incorrectly.

After this calculation, we can now simply solve the linear system Eq. 11.69 for the coefficients $\mathbf{c} = (c_i)$ in the trial function in order to find the approximation

$$y(x) \approx \sum_{i=1}^{N} c_i \phi_i(x). \tag{11.82}$$

12 Partial Differential Equations

12.1 Partial differential equations: Overview

We will now consider approximation methods for *partial* differential equations (PDEs). An example for a PDE – familiar from the first-year Calculus course – is the heat equation:

$$\frac{\partial^2}{\partial x^2}u(x,t) = \kappa\frac{\partial}{\partial t}u(x,t) \tag{12.1}$$

with some constant $\kappa > 0$. PDEs like the above are defined on a certain region in the x-t-plane (or analogously in more than 2 dimensions), and are always complemented by some kind of boundary conditions on the boundary of that region.

We will try to generalize our methods for boundary value problems of ODEs to the case of PDEs. The BVP methods we have discussed are:

- the Shooting method,
- the Finite Difference method,
- the Rayleigh-Ritz method.

Unfortunately, there is no obvious generalization of the Shooting method to PDEs; the concept of transforming a boundary value problem into an initial value problem does not work in this context.

However, the Finite Difference method can be generalized to PDEs. To that end, we would first need to define a suitable notion of mesh points. Instead of dividing an interval into equally spaced subintervals, we now need to divide a region in two variables (say, a rectangle) with an equally spaced *grid* of mesh points. Then, as before, one can replace the value of u at mesh points with an approximation value, and the derivatives of u with finite difference quotients, and finally solve a (linear) equation system to obtain the approximation values numerically. We will discuss this more in detail in later sections.

The Finite Difference method has limitations when the region in question is not as simple as a rectangle, but is irregularly shaped. In this case, it may not be possible to divide it reasonably with an equally spaced grid, or the mesh points at the edge of the grid may not be located exactly on the boundary (which poses problems when interpreting the boundary values). In these cases, generalizations of the Rayleigh-Ritz method can successfully be used, the so-called *Finite Element methods*. Note that in the Rayleigh-Ritz method, there was a large freedom of choosing the basis functions ϕ_j, and even when restricting to piecewise linear functions, the mesh points x_i did not need to be equally spaced. Likewise, in the multi-dimensional generalizations, one can exploit this freedom to adapt the choice of mesh points to a (possibly irregular) boundary. Finite Element methods are the most advanced numerical methods for solving PDEs, and we will not discuss them in detail here; see, e.g., (Burden and Faires 2010 Ch. 12.4) for an introduction.

12.2 Elliptic PDEs

In this section, we will generalize the Finite Difference method to a very specific PDE, namely the Poisson equation. This equation for a function u of two variables x and y has the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

$$\text{on } R := \{(x, y) : a \leq x \leq b, c \leq y \leq d\},$$

$$u(x, y) = g(x, y) \text{ for } x \in \partial R.$$

$$(12.2)$$

Here f and g are given functions of two variables, and ∂R denotes the boundary of R, that is, the four line segments

$$x = a, c \leq y \leq d; \quad x = b, c \leq y \leq d;$$

$$a \leq x \leq b, y = c; \quad a \leq x \leq b, y = d.$$

$$(12.3)$$

The equation Eq. 12.2 is, in the case $f = 0$, also known as the *Laplace equation*. It is a typical example of the larger class of *elliptic* differential equations, and the methods we will discuss here apply to other elliptic equations as well.

We will try to set up a Finite Difference method to approximate the solution of Eq. 12.2, following our recipe from Section 11.3. To that end, we first have to specify what our mesh points are. Instead of partitioning the interval $[a, b]$, they will now need to partition the rectangle R. To this end, we choose *two* numbers of steps, N and M, associated with the x and y direction respectively, and two corresponding step sizes

$$h := \frac{b - a}{N + 1}, \quad k := \frac{d - c}{M + 1}.$$

We then define $N \times M$ mesh points (x_i, y_j) as

$$(x_i, y_j) = (a + ih, c + jk), \quad 1 \leq i \leq N, \ 1 \leq j \leq M.$$

These lie on a rectangular grid within the rectangle R.

Our next step is to approximate the relevant derivatives of u at the mesh points. We use the centred difference formula Eq. 11.43 twice, once in x direction (at fixed y) and once in y direction (at fixed x). This gives

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} - \frac{h^2}{12} \frac{\partial^4 u}{\partial x^4}(\xi_i, y_j),$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} - \frac{k^2}{12} \frac{\partial^4 u}{\partial y^4}(x_i, \eta_j),$$

$$(12.4)$$

where ξ_i and η_j are some unknown points in the intervals.

Again following our recipe, we will approximate the solution at mesh points $u(x_i, y_j)$ with approximation values $w_{i,j}$. In the PDE Eq. 12.2, we then replace $u(x_i, y_j)$ with $w_{i,j}$ and the derivatives of u with the expressions Eq. 12.4, but leaving away the remainder terms of order $O(h^2) + O(k^2)$. This yields the following equations for the $w_{i,j}$:

$$\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} = f(x_i, y_j)$$

for $1 \leq i \leq N$, $1 \leq j \leq M$. After multiplying with $-h^2$,

$$2\left(\left(\tfrac{h}{k}\right)^2 + 1\right)w_{i,j} - w_{i+1,j} - w_{i-1,j} - \left(\tfrac{h}{k}\right)^2 w_{i,j+1} - \left(\tfrac{h}{k}\right)^2 w_{i,j-1} = -h^2 f(x_i, y_j). \quad (12.5)$$

The boundary conditions are then expressed by setting the values of $w_{i,j}$ for $i = 0$, $j = 0$, $i = N + 1$ or $j = M + 1$ to the required boundary values:

$$w_{0,j} = g(x_0, y_j), \quad w_{N+1,j} = g(x_{N+1}, y_j) \quad \text{for } 1 \leq j \leq M,$$

$$w_{i,0} = g(x_i, y_0), \quad w_{i,M+1} = g(x_j, y_{M+1}) \quad \text{for } 1 \leq i \leq N.$$

$$(12.6)$$

Together, Eq. 12.5 and Eq. 12.6 form a linear equation system that can be solved to obtain approximation values $w_{i,j}$. To that end, it may be useful to renumber the "double indices" i, j with a single

index, setting, e.g., $\ell := N(i-1) + j$. The index k then runs from 1 to NM, and we may rewrite the equation system as $\mathbf{Aw} = \mathbf{b}$ with a $NM \times NM$ matrix $\mathbf{A}$ and a vector $\mathbf{b} \in \mathbb{R}^{NM}$.

As in the one-dimensional case, the matrix $\mathbf{A}$ is sparse, that is, many of its entries are known to be zero. (It is not tridiagonal, however.) This makes the system $\mathbf{Aw} = \mathbf{b}$ fast to solve. Nevertheless, it is evident that the number of mesh points (and hence of vector dimensions) can become rather large quite easily, which sets practical limits to the accuracy of this and other approximation methods for PDEs.

12.3 Parabolic PDEs

As a second class of PDEs to be treated with the Finite Difference method, we consider *parabolic PDEs*. A typical example – and the only one we will consider – is the one-dimensional *heat equation* (also known as *diffusion equation*). This equation for a function u of two variables x and t has the form[1]

$$
\begin{aligned}
&\alpha^2 \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \quad \text{on } R := \{(x,t) : 0 \le x \le L, 0 \le t\}, \\
&u(x,0) = g(x) \text{ for } 0 < x < L, \\
&u(0,t) = u(L,t) = 0 \text{ for all } t > 0.
\end{aligned}
\tag{12.7}
$$

The boundary of the region R consists of *three* pieces here. One also refers to the condition $u(x,0) = g(x)$ as *initial condition*; as we will see, it has some similarities to initial conditions for ODEs.

In applications, $u(x,t)$ might have the interpretation of a local temperature – say, in a homogeneous wall of width L – depending on the spatial position x and on time t. The initial condition is the temperature in the wall at time 0, and the remaining boundary conditions represent the temperature of the environment, depending on time t.

Again, we will set up a Finite Difference method to approximate the solution of Eq. 12.7, following our recipe from Section 11.3. As a first step, we restrict the region R to a rectangle, introducing the condition $0 \le t \le T$ (with some fixed $T > 0$). Then, as for the Laplace equation, we introduce two numbers of steps, N and M, associated with the x and t direction respectively, and two corresponding step sizes

$$
h := \frac{L}{N+1}, \quad k := \frac{T}{M}
$$

(note the slightly different convention for k from before). Once more, we define mesh points (x_i, t_j) as

$$
(x_i, t_j) = (ih, jk), \quad 0 \le i \le N+1, \ 0 \le j \le M.
$$

The next step is to approximate the derivatives of u, and here the very specific properties of the equation Eq. 12.7 show up. In the variable x, we once more use the centred difference formula Eq. 11.43, leading to

$$
\frac{\partial^2 u}{\partial x^2} u(x_i, t_j) = \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j)}{h^2} + O(h^2).
\tag{12.8}
$$

However, for the derivative by t, we cannot follow the same approach. The problem here is that the centred difference formula for the first derivative would involve $u(x_i, t_{j+1})$ and $u(x_i, t_{j-1})$, but we have only *one* boundary (or initial) condition to fix the value if the mesh point is on the boundary of R, and this would lead to an under-determined linear equation system later. As a way out, we use the (much simpler) *forward difference formula*,

$$
\frac{\partial u}{\partial t} u(x_i, t_j) = \frac{u(x_i, t_{j+1}) - u(x_i, t_j)}{k} + O(k).
\tag{12.9}
$$

---

[1] In the form presented here, the equation is actually explicitly solvable in terms of a Fourier series. One might ask therefore why numerical approximation methods are necessary. The answer is that the boundary conditions $u(0,t) = u(L,t) = 0$ chosen here are rather simplistic, which was done to simplify the discussion. But the numerical method can be generalized to more intricate boundary conditions where an explicit solution is no longer feasible.

Further following our recipe, we insert the finite difference formulas Eq. 12.8 and Eq. 12.9 into the PDE Eq. 12.7, then replace $u(x_i, y_j)$ with approximation values $w_{i,j}$ and leave away the remainder terms of order $O(h^2) + O(k)$. This yields the following equations for the $w_{i,j}$:

$$\alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = \frac{w_{i,j+1} - w_{i,j}}{k}, \quad 1 \le i \le N,\ 0 \le j < M.$$

or, setting $\lambda := \alpha^2 k / h^2$,

$$w_{i,j+1} = (1 - 2\lambda)w_{i,j} + \lambda(w_{i+1,j} + w_{i-1,j}) \tag{12.10}$$

where $w_{0,j} = w_{N+1,j} = 0$ for all j (boundary condition) and $w_{i,0} = g(x_i)$ for all i (initial condition).

An interesting point is that the linear equation system Eq. 12.10 is extremely easy to solve: Namely, inserting the known values $w_{i,0}$ into the right-hand side gives us $w_{i,1}$ for all i; again inserting these into the right hand side gives us $w_{i,2}$ for all i; and so forth. We can rewrite this procedure in matrix form: Setting

$$\mathbf{w}_j := (w_{1,j}, \dots, w_{N,j}),$$

$$\mathbf{A}_+ := \begin{pmatrix} (1-2\lambda) & \lambda & 0 & \cdots & & 0 \\ \lambda & (1-2\lambda) & \lambda & 0 & \cdots & 0 \\ 0 & \lambda & (1-2\lambda) & \lambda & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \lambda \\ 0 & \cdots & & 0 & \lambda & (1-2\lambda) \end{pmatrix} \quad \text{(an } N \times N \text{ matrix)},$$

we obtain the relation

$$\mathbf{w}_{j+1} = \mathbf{A}_+ \mathbf{w}_j.$$

Thus, starting with the initial value $\mathbf{w}_0$, the approximation can be obtained by an iterated matrix multiplication. This approximation method is called the *Forward Difference method.*

The Forward Difference method is very simple to apply, but it has a major disadvantage: it becomes unstable if the the step size k is not chosen very small. This phenomenon is closely related to the one we saw for stiff equations in Section 10.9. We will skip a more in-depth analysis here; see (Burden and Faires 2010, sec. 12.2).

We do, however, want to present a solution to the stability problem here, which is similar to the one found for stiff equations: we use an "implicit method" for approximation, the *Backward Difference method.* To that end, instead of the forward difference formula Eq. 12.9, we use the backward difference formula

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j) - u(x_i, t_{j-1})}{k} + O(k). \tag{12.11}$$

Leaving all other construction steps the same, we arrive at another method of order $O(h^2 + k)$ where the equation system Eq. 12.10 is now replaced by

$$w_{i,j-1} = (1 + 2\lambda)w_{i,j} - \lambda(w_{i+1,j} + w_{i-1,j}). \tag{12.12}$$

Again, we rewrite this in matrix form: with

$$\mathbf{w}_j := (w_{1,j}, \dots, w_{N,j}),$$

$$\mathbf{A}_- := \begin{pmatrix} (1+2\lambda) & -\lambda & 0 & \cdots & & 0 \\ -\lambda & (1+2\lambda) & -\lambda & 0 & \cdots & 0 \\ 0 & -\lambda & (1+2\lambda) & -\lambda & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -\lambda \\ 0 & \cdots & & 0 & -\lambda & (1+2\lambda), \end{pmatrix},$$

we can rewrite Eq. 12.12 as

$$\mathbf{w}_{j-1} = \mathbf{A}_- \mathbf{w}_j.$$

This does no longer explicitly give $\mathbf{w}_j$ from $\mathbf{w}_{j-1}$. However, this is only a matter of matrix inversion (or, equivalently, solving linear equation systems) as we clearly have

$$\mathbf{w}_j = \mathbf{A}_-^{-1}\mathbf{w}_{j-1}.$$

Starting from $\mathbf{w}_0$, this again allows us to compute all approximation values by iterative matrix multiplication (or iteratively solving linear equation systems). Since the matrix $\mathbf{A}$ is sparse (tridiagonal), this can be done very efficiently. Thus the Backward Difference method is only slightly more complex than the Forward Difference method. It does, however, not suffer from stability problems.

We can reach a unified view of the two schemes by considering the $N \times N$ tridiagonal matrix

$$B = \begin{pmatrix} 2 & -1 & 0 & & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{pmatrix}$$

Applied to a vector $y \in \mathbb{R}^N$, it gives

$$-2y_1 + y_2, \ldots, y_{i+1} - 2y_i + y_{i+1}, \ldots, y_{N-1} - 2y_N$$

Apart from a factor of $1/h^2$, this is exactly the symmetric difference formula for the second derivative, applied the sequence of values $0, y_1, y_2, \ldots, y_N, 0$.

Consider Euler's method, with step length k:

$$\mathbf{w}_{j+1} = \mathbf{w}_j + k(\text{derivative})$$

In our problem, the time derivative is α^2 times the second space derivative, which can be calculated using the matrix B. Once the scale factors are taken into account, Euler's method leads us to

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \lambda B\mathbf{w}_j = (I + \lambda B)\mathbf{w}_j = A_+\mathbf{w}_j$$

which is exactly the forward difference method above. The backward difference method is given by

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \lambda B\mathbf{w}_{j+1}$$

in which we think of the one-sided difference quotient as an approximation for the derivative at the right-hand endpoint, not the left-hand endpoint. This rearranges to

$$\mathbf{w}_j = (I - \lambda B)\mathbf{w}_{j+1} = A_-\mathbf{w}_{j+1},$$

which is exactly the backward difference method above. As a single-step method for ODEs, this is known as the backward Euler or implicit Euler method.

Finally, analogously to the implicit trapezoidal method, we can use the average of the steps from the forward and backward difference method to give the Crank-Nicolson method.

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \frac{1}{2}\left(\lambda B\mathbf{w}_j + \lambda B\mathbf{w}_{j+1}\right)$$

Rearranging this gives us

$$(I - \lambda B/2)\mathbf{w}_{j+1} = (I + \lambda B/2)\mathbf{w}_j$$

Like the backward difference method, this gives us a tridiagonal system of equations to solve to get from $\mathbf{w}_j$ to $\mathbf{w}_{j+1}$, and can be used to find $\mathbf{w}_j$ for any j.

It turns out that the error terms for the forward and backward difference methods have the form $Ck + O(k^2)$ and $-Ck + O(k^2)$. Taking the average cancels the $\pm Ck$ terms and leaves an error of order $O(k^2)$; in combination with the space variable, we have $O(h^2) + O(k^2)$ for the whole method, as compared with $O(h^2) + O(k)$ for the forward and backward difference methods. Like the implicit trapezoidal method, the Crank-Nicolson method is absolutely stable.

A Taylor's theorem

In many of the approximations and error estimates that we make throughout this course, Taylor's theorem plays an important role. Since the theorem can be formulated in various ways, in particular, with different forms of the remainder term, it will be recalled in this appendix, with the conventions we use.

We first repeat the theorem in its simplest form, for a real-valued function of one real variable, with the remainder in Lagrange form (Apostol 1969, sec. 7.7).

Theorem A.1. *Let $I \subset \mathbb{R}$ be an interval, and $f \in \mathcal{C}^{k+1}(I, \mathbb{R})$. For each $a \in I$ and $x \in I$, there exists $\xi \in [a, x]$ such that*

$$f(x) = \sum_{j=0}^{k} \frac{1}{j!} \frac{d^j f}{dx^j}(a)\,(x-a)^j + \frac{1}{(k+1)!} \frac{d^{k+1} f}{dx^{k+1}}(\xi)\,(x-a)^{k+1}. \tag{A.1}$$

For our purposes, we need generalizations of Taylor's theorem both to functions of several variables and to vector-valued functions. Let us formulate a full generalization to functions $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^n$, even if we do *not* actually need it in this generality.

For this, we need some notation. A tuple of m nonnegative integers, $\mathbf{j} = (j^{(1)}, \ldots, j^{(m)}) \in \mathbb{N}_0^m$, is called a *multi-index*. We use the following shorthand notation:

$$\begin{aligned}
|\mathbf{j}| &= j^{(1)} + \ldots + j^{(m)} && \text{(the \textit{length} of the multi-index)}, \\
\mathbf{j}! &= j^{(1)}! \cdot \ldots \cdot j^{(m)}!, \\
\mathbf{x}^{\mathbf{j}} &= (x^{(1)})^{j^{(1)}} \cdot \ldots \cdot (x^{(m)})^{j^{(m)}} && \text{for } \mathbf{x} \in \mathbb{R}^m, \\
\frac{\partial^{|\mathbf{j}|} \mathbf{f}}{\partial \mathbf{x}^{\mathbf{j}}} &= \frac{\partial^{|\mathbf{j}|} \mathbf{f}}{(\partial x^{(1)})^{j^{(1)}} \cdots (\partial x^{(m)})^{j^{(m)}}}.
\end{aligned} \tag{A.2}$$

Taylor's theorem can then be formulated as follows.

Theorem A.2. *Let $D \subset \mathbb{R}^m$ be convex, and let $\mathbf{f} \in \mathcal{C}^{k+1}(D, \mathbb{R}^n)$. For each $\mathbf{a} \in D$, there exists $\mathbf{R_a} : D \to \mathbb{R}^n$ such that*

$$\mathbf{f}(\mathbf{x}) = \sum_{|\mathbf{j}| \leq k} \frac{1}{\mathbf{j}!} \frac{\partial^{|\mathbf{j}|} \mathbf{f}}{\partial \mathbf{x}^{\mathbf{j}}}(\mathbf{a})\,(\mathbf{x} - \mathbf{a})^{\mathbf{j}} + \mathbf{R_a}(\mathbf{x}) \tag{A.3}$$

and

$$\|\mathbf{R_a}(\mathbf{x})\| \leq \|\mathbf{x} - \mathbf{a}\|^{k+1} \sum_{|\mathbf{j}| = k+1} \frac{1}{\mathbf{j}!} \sup_{\mathbf{x}' \in D} \left\| \frac{\partial^{|\mathbf{j}|} \mathbf{f}}{\partial \mathbf{x}^{\mathbf{j}}}(\mathbf{x}') \right\| \tag{A.4}$$

for all $\mathbf{x} \in D$.

(See (Devinatz 1968, sec. 7.4) for a proof in the case $n = 1$ and with an explicit remainder term. The above version then follows by estimating the remainder with its supremum, and taking the maximum over the components of $\mathbf{f}$. It is possible to obtain an explicit form of the remainder for $n > 1$ as well, although not in Lagrange form; but the formula is slightly complicated, and we will not need it.)

We are interested in the following special cases. First, there is the case where $m = 1$, i.e., $\mathbf{f}$ depends only on one variable. Then the multi-index $\mathbf{j}$ is just a number $j \in \mathbb{N}_0$, the partial derivatives are ordinary derivatives, and we obtain:

Theorem A.3. *Let I be an interval, and let $\mathbf{f} \in \mathcal{C}^{k+1}(I, \mathbb{R}^n)$. For each $a \in I$, there exists $\mathbf{R}_a : I \to \mathbb{R}^n$ such that*

$$\mathbf{f}(x) = \sum_{j=0}^{k} \frac{1}{j!} \frac{d^j \mathbf{f}}{dx^j}(a)\,(x-a)^j + \mathbf{R}_a(x) \tag{A.5}$$

and

$$\|\mathbf{R}_a(x)\| \leq \frac{|x-a|^{k+1}}{(k+1)!} \sup_{x' \in I} \left\|\frac{d^j \mathbf{f}}{dx^j}(x')\right\| \tag{A.6}$$

for all $x \in I$.

On other occasions, we will need the function $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^n$ in full generality, but use the Taylor expansion only up to order $k \leq 1$. In this case, the relevant derivatives of $\mathbf{f}$ can be written in an easier way: Those with $|\mathbf{j}| = 1$ are just single derivatives $\partial \mathbf{f}/\partial x^{(p)}$ with p ranging from 1 to m, and they can conveniently be combined into a matrix $\partial \mathbf{f}/\partial \mathbf{x}$. We have

$$\sum_{|\mathbf{j}|=1} \frac{1}{\mathbf{j}!} \frac{\partial^{|\mathbf{j}|}\mathbf{f}}{\partial \mathbf{x}^{\mathbf{j}}}(\mathbf{a})\,(\mathbf{x}-\mathbf{a})^{\mathbf{j}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{a}) \cdot (\mathbf{x}-\mathbf{a}), \tag{A.7}$$

reading the r.h.s. as a matrix product. The derivatives with $|\mathbf{j}| = 2$ are of the form $\partial^2 \mathbf{f}/\partial x^{(p)}\partial x^{(q)}$, where both cases $p = q$ and $p \neq q$ occur. Working out the numerical prefactors, one obtains the following special cases for order $k = 0$ and $k = 1$ repectively.

Theorem A.4. *Let $D \subset \mathbb{R}^m$ be convex, and let $\mathbf{f} \in \mathcal{C}^1(D, \mathbb{R}^n)$. For each $\mathbf{a} \in D$, there exists $\mathbf{R}_{\mathbf{a}} : D \to \mathbb{R}^n$ such that*

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{a}) + \mathbf{R}_{\mathbf{a}}(\mathbf{x}) \tag{A.8}$$

and

$$\|\mathbf{R}_{\mathbf{a}}(\mathbf{x})\| \leq \|\mathbf{x} - \mathbf{a}\| \sup_{\mathbf{x}' \in D} \left\|\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}')\right\| \tag{A.9}$$

for all $\mathbf{x} \in D$.

Theorem A.5. *Let $D \subset \mathbb{R}^m$ be convex, and let $\mathbf{f} \in \mathcal{C}^2(D, \mathbb{R}^n)$. For each $\mathbf{a} \in D$, there exists $\mathbf{R}_{\mathbf{a}} : D \to \mathbb{R}^n$ such that*

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{a}) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{a}) \cdot (\mathbf{x}-\mathbf{a}) + \mathbf{R}_{\mathbf{a}}(\mathbf{x}) \tag{A.10}$$

and

$$\|\mathbf{R}_{\mathbf{a}}(\mathbf{x})\| \leq \frac{1}{2}\|\mathbf{x}-\mathbf{a}\|^2 \sum_{p,q=1}^{m} \sup_{\mathbf{x}' \in D} \left\|\frac{\partial^2 \mathbf{f}}{\partial x^{(p)}\partial x^{(q)}}(\mathbf{x}')\right\| \tag{A.11}$$

for all $\mathbf{x} \in D$.

References

Apostol, T. M. 1969. *Calculus, Vol. 1*. 2nd ed. Wiley.

Burden, Richard L., and J. Douglas Faires. 2010. *Numerical Analysis*. 9th ed. Brooks Cole.

Devinatz, A. 1968. *Advanced Calculus*. Holt, Rinehart; Winston.

Kelley, C. T. 1995. *Iterative Methods for Linear and Nonlinear Equations*. Frontiers in Applied Mathematics. Society for Industrial; Applied Mathematics (SIAM).

Ortega, James M. 1972. *Numerical Analysis; a Second Course*. Computer Science and Applied Mathematics. New York: Academic Press. https://yorsearch.york.ac.uk/permalink/f/1d5jm03/44YORK_ALMA_DS21223457930001381.

Stewart, J. 1991. *Calculus*. Brooks/Cole.

Wait, R. A. 1979. *The Numerical Solution of Algebraic Equations*. Wiley-Interscience Publication. John Wiley. https://yorsearch.york.ac.uk/permalink/f/1d5jm03/44YORK_ALMA_DS21219298850001381.

Weir, M. D., G. B. Thomas, and J. Hass. 2010. *Thomas' Calculus*. Addison-Wesley.