

NUMERICAL METHODS

Lecture Notes 2014–2015

Willem Hundsdorfer

CWI / Radboud Universiteit Nijmegen

Updates will be made available at:
www.cwi.nl/~willem

Preface

In these notes some basic *numerical methods* will be described. The construction of numerical methods and the study of these methods is called *numerical mathematics*.

Numerical mathematics goes back a long time, as will become clear from the names of famous mathematicians associated with some well-known numerical methods, such as Newton, Euler, Lagrange, Gauss. Today, numerical mathematics is still a very active branch of mathematics. Numerical simulations are more and more used by scientists to study problems from physics, chemistry, biology, medicine, finance, and by engineers to design planes, chips, and many other applications. The combination of numerical mathematics and mathematical modelling with sophisticated computing platforms is often called *computational science*.

In these notes some numerical methods are discussed and analyzed. Increasingly such methods are incorporated into numerical software, in packages like MATLAB and MATHEMATICA, but still the limitations of the various methods have to be understood.

Material: The material for these notes has been primarily taken from the following books

- W. Gautschi, *Numerical Analysis : an Introduction*, Birkhäuser, 1997.
- J. Stoer, R. Bulirsch, *Introduction to Numerical Analysis*, 3th ed., Springer, 2002,

and the lecture notes

- E. Hairer, *Introduction à l'Analyse Numérique*, University of Geneva, 2001.

Also available on the internet, and in Dutch, are the lecture notes

- M.N. Spijker, J.A. van de Griend, *Inleiding tot de Numerieke Wiskunde*, University of Leiden, 2008.

Each section of these notes finishes with a number of exercises. Some of them are programming exercises. The choice of programming language or system is free. It is easiest to choose a system where some standard numerical subroutines (e.g. solving linear systems) and plotting is directly available. Examples of such systems are MATLAB, or OCTAVE/SCILAB, and MATHEMATICA. For programming with PYTHON the extensions NUMPY and MATPLOTLIB can be used.

Subsections and exercises marked with an asterisk * are not required for the examination, and the same applies to the numbered remarks. Those parts are mainly included to make the text self-contained. Most of the exercises having an asterisk elaborate statements appearing in the text but not proven there, and these statements may be taken for granted.

Typing errors: This text will, unfortunately, contain a number of small errors. If you find some, please let me know (willem.hundsdorfer@cwi.nl).

Some notations and Taylor expansions: For given real functions φ_1, φ_2 we will write

$$(0.1) \quad \varphi_1(t) = \varphi_2(t) + \mathcal{O}(t^n) \quad (t \rightarrow 0)$$

if there are $\delta, C > 0$ such that $|\varphi_1(t) - \varphi_2(t)| \leq C|t|^n$ for all $|t| < \delta$.

This notation will often be used in Taylor expansions:

$$(0.2) \quad f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \cdots + \frac{1}{k!}f^{(k)}(x)h^k + \mathcal{O}(h^{k+1})$$

for $h \rightarrow 0$, where the function $f : \mathbb{R} \rightarrow \mathbb{R}$ is assumed to be $k+1$ times continuously differentiable around x . The series $\sum_{j=0}^k \frac{1}{j!} f^{(j)}(x) h^j$ is called a *truncated Taylor series*.

The expansion (0.2) follows from Taylor's theorem with remainder term:

$$(0.3) \quad f(x+h) = \sum_{j=0}^k \frac{1}{j!} f^{(j)}(x) h^j + \int_0^1 \frac{(1-t)^k}{k!} f^{(k+1)}(x+th) dt h^{k+1}.$$

The remainder term can also be expressed with an intermediate value as

$$(0.4) \quad f(x+h) = \sum_{j=0}^k \frac{1}{j!} f^{(j)}(x) h^j + \frac{1}{(k+1)!} f^{(k+1)}(x+\theta h) h^{k+1}$$

with some $\theta \in (0, 1)$. So $\xi = x + \theta h$ is an intermediate point, between x and $x+h$, and for $k=1$ this is just the mean-value theorem, $f(x+h) - f(x) = f'(\xi)h$.

The formulas (0.2) and (0.3) remain valid for functions $f : \mathbb{R} \rightarrow \mathbb{R}^m$ and $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$, $m > 1$, with proper interpretation of the terms $f^{(j)}(x)h^j$ and norms instead of absolute values. Formula (0.4), on the other hand, only holds for real, scalar functions. For example, if $f : \mathbb{R} \rightarrow \mathbb{R}^m$ then (0.4) holds for each component f_i individually, but all these components may have different intermediate points $\xi_i = x + \theta_i h$.

Contents

1	Nonlinear Equations	1
1.1	Bisection	1
1.2	Fixed Point Iteration	2
1.3	Newton's Method	3
1.4	Systems of Equations	6
1.5	Exercises	8
2	Linear Systems	10
2.1	Gaussian Elimination and LU -Decomposition	10
2.2	Positive Definite Matrices and Band Matrices	13
2.3	Overdetermined Systems: the Normal Equations	15
2.4	The Condition Number of a Matrix	16
2.5	Exercises	17
3	Polynomial Interpolation and Approximation	20
3.1	Interpolation Formulas	20
3.2	The Interpolation Errors	21
3.3	Piecewise Polynomials and Splines	25
3.4	Exercises	27
4	Trigonometric Interpolation with DFT and FFT	30
4.1	Fourier Series and Fourier Transforms	30
4.2	Approximation Properties	32
4.3	Trigonometric Interpolation	33
4.4	Fast Fourier Transforms	35
4.5	Exercises	37
5	Numerical Integration	38
5.1	Composite Integration Schemes	38
5.2	Super-convergence and Gauss Quadrature	41
5.3	Practical Error Estimation and Partitioning	45
5.4	Exercises	46
6	Initial Value Problems for ODEs	48
6.1	Runge-Kutta Methods	48
6.2	Consistency	52
6.3	Convergence	55
6.4	Step-size Selection	57
6.5	Exercises	59

7	Stiff Initial Value Problems	61
7.1	Explicit and Implicit Euler Method for Stiff Problems	61
7.2	Stability Regions and Implicit Methods	64
7.3	Example: the θ -Method for Stiff Problems	68
7.4	A Semi-discrete Initial-Boundary Value Problem	70
7.5	Exercises	72
8	Two-Point Boundary Value Problems	73
8.1	Shooting Methods	73
8.2	Sturm-Liouville Problems and Weak Forms	74
8.3	Galerkin Methods and Finite Elements	77
8.4	Exercises	80

1 Nonlinear Equations

A basic numerical problem is: find an x satisfying the nonlinear equation $f(x) = 0$, where f is a given function. Such problems arise very frequently. Sometimes the function f is given in analytical form, but there are also many applications where that is not the case. In fact, evaluation of function values $f(x)$ may require itself some numerical procedure.

Example 1.1 Let

$$f(x) = \int_0^x \varphi(s) ds - \mu,$$

with given $\mu > 0$ and φ a continuous real function. If this function φ is complicated, we will not have an explicit expression for the integral, and a numerical procedure may be needed to compute the value of $f(x)$ for given x . On the other hand, certain properties of f may be known. For instance, if $0 < \alpha \leq \varphi(s) \leq \beta$ (for all $s \geq 0$), then $f(x)$ is monotonically increasing and $\alpha x - \mu \leq f(x) \leq \beta x - \mu$ for $x \geq 0$. Therefore we know that the equation $f(x) = 0$ has a unique solution x_* in $[\mu/\beta, \mu/\alpha]$. The question is now: how can this solution x_* be computed with some prescribed accuracy? \diamond

In this section we will discuss several numerical methods for finding approximations to the solution of $f(x) = 0$, or the related fixed point equation $g(x) = x$. First the methods are considered for scalar equations with one real variable. Extensions to systems are given afterwards.

1.1 Bisection

Suppose $f : \mathbb{R} \rightarrow \mathbb{R}$ is continuous on the interval $[a, b]$. If $f(a)$ and $f(b)$ have opposite signs we know there is an $x_* \in (a, b)$ such that $f(x_*) = 0$. Such x_* is called a zero of f or root of the equation. Let us assume for the moment that x_* is the only root in $[a, b]$.

One of the most simple methods from numerical mathematics is *bisection*, where the interval is repeatedly halved, such that each new interval still contains a root. First we initialize $a_1 = a$, $b_1 = b$, and then for $k = 1, 2, 3, \dots$ we take

$$(1.1) \quad \begin{cases} x_k = \frac{1}{2}(a_k + b_k); \\ \text{if } f(x_k) = 0 \text{ we are done; otherwise} \\ \dots \text{ if } \text{sign}(f(x_k)) = \text{sign}(f(a_k)), \text{ set } a_{k+1} = x_k, b_{k+1} = b_k, \\ \dots \text{ if } \text{sign}(f(x_k)) = \text{sign}(f(b_k)), \text{ set } a_{k+1} = a_k, b_{k+1} = x_k. \end{cases}$$

It is clear that each interval $[a_k, b_k]$ contains a zero of f , and $b_k - a_k = 2^{1-k}(b - a)$, because the interval is halved each time. Consequently

$$(1.2) \quad |x_k - x_*| \leq 2^{-k}(b - a) \quad (k \geq 1).$$

If we want an approximation x_k to x_* with an error less than a given tolerance Tol , which is a desired accuracy, then we can terminate the algorithm as soon as $2^{-k} < Tol/(b - a)$.

Bisection guarantees convergence of x_k towards x_* . However, this convergence is rather slow. For example, if $b - a = 1$ and we want to approximate x_* up to 12 decimal places, then we need in general $k = 40$ steps to achieve this. An other drawback of the bisection method is that it cannot be extended to systems of equations. For the other methods treated in this section such extensions do exist.

Remark 1.2 In theory we could choose Tol arbitrarily small. However, on computers it is standard to use finite precision representations of real numbers. Any non-zero $x \in \mathbb{R}$ can be written as $x = \pm d \cdot 10^m$ with $0.1 \leq d < 1$, integer m and with \pm either plus or minus. The truncated or rounded version is

$$\hat{x} = \pm 0.d_1d_2 \dots d_n \cdot 10^m,$$

where $n \in \mathbb{N}$ is the number of digits, and $d_j \in \{0, 1, \dots, 9\}$, $d_1 \neq 0$. This fixed number n , say $n = 16$, determines the relative precision of the representation.¹

As a result, any arithmetic operation (addition, multiplication, division) will have some small error, called the round-off error. This is due to the fact that a product $\hat{x} \cdot \hat{y}$ or sum $\hat{x} + \hat{y}$ of two rounded numbers \hat{x}, \hat{y} will be rounded again to n significant digits to fit it in the computer memory. \diamond

1.2 Fixed Point Iteration

Let us consider the fixed point problem

$$(1.3) \quad g(x) = x,$$

where $g : \mathbb{R} \rightarrow \mathbb{R}$ is given. Starting with an initial guess x_0 , we can compute successive approximations x_1, x_2, \dots by the iteration

$$(1.4) \quad x_{k+1} = g(x_k) \quad (k = 0, 1, \dots).$$

This is called *fixed point iteration*, or *functional iteration*.

Theorem 1.3 Let $g : [a, b] \rightarrow [a, b]$ be continuously differentiable, and suppose there is a number $\theta < 1$ such that $|g'(x)| \leq \theta$ for all $x \in [a, b]$. Then there is a unique fixed point x_* of g in $[a, b]$. Moreover, for any $x_0 \in [a, b]$ the fixed point iteration converges, and

$$(1.5) \quad |x_k - x_*| \leq \theta |x_{k-1} - x_*| \quad (k \geq 1).$$

¹Actually, computers do not use a decimal, base 10 representation, but base 2 (or a power of 2). A common value for the precision is $2^{-52} \approx 2 \cdot 10^{-16}$.

Proof. Let $f(x) = x - g(x)$. This function is continuous with $f(a) \leq 0$, $f(b) \geq 0$, because $g(a) \geq a$ and $g(b) \leq b$. Hence there is a zero of f in $[a, b]$, and this is a fixed point of g .

According to the mean value theorem, for any pair $x, \tilde{x} \in [a, b]$ there is a point ξ between x and \tilde{x} such that $g(x) - g(\tilde{x}) = g'(\xi)(x - \tilde{x})$. Hence

$$|g(x) - g(\tilde{x})| \leq \theta |x - \tilde{x}|.$$

This shows that a fixed point is unique, and it gives the estimate for the error. \square

Corollary 1.4 Let $g : [a, b] \rightarrow \mathbb{R}$ be continuously differentiable, and suppose that $x_* \in [a, b]$ satisfies $g(x_*) = x_*$ and $|g'(x_*)| < 1$. Then, there is a $\delta > 0$ such that the fixed point iteration converges whenever $|x_0 - x_*| < \delta$.

Proof. Let $\theta \in (|g'(x_*)|, 1)$, and take $\delta > 0$ small enough to have $|g'(x)| \leq \theta$ for all $x \in [x_* - \delta, x_* + \delta]$. For x in this interval we have $|g(x) - g(x_*)| \leq \theta |x - x_*|$, so the interval is mapped into itself. The result now follows by applying Theorem 1.3 with a, b replaced by $\alpha = x_* - \delta$ and $\beta = x_* + \delta$, respectively. \square

From (1.5) we directly obtain the error bound

$$(1.6) \quad |x_k - x_*| \leq \theta^k |x_0 - x_*| \quad (k \geq 1).$$

Since the size of $|x_0 - x_*|$ is not known, this seems of little practical value. However, we also have

$$|x_k - x_*| \leq \theta |x_{k-1} - x_*| \leq \theta (|x_{k-1} - x_k| + |x_k - x_*|),$$

and therefore (1.5) also gives the bound

$$(1.7) \quad |x_k - x_*| \leq \frac{\theta}{1-\theta} |x_k - x_{k-1}| \quad (k \geq 1),$$

which provides a stopping criterion during the iteration process.

1.3 Newton's Method

Suppose $f : \mathbb{R} \rightarrow \mathbb{R}$ is differentiable, and we want to find a root x_* of the equation

$$(1.8) \quad f(x) = 0.$$

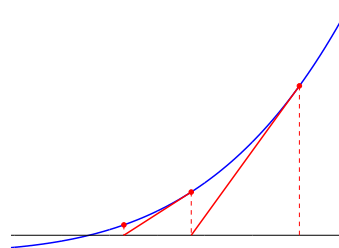
Given an approximation x_k , we can locally linearize the function f by

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k),$$

and then find a new approximation x_{k+1} which is a zero of the linearized function. This procedure leads to the iteration

$$(1.9) \quad x_{k+1} = x_k - \frac{1}{f'(x_k)} f(x_k) \quad (k = 0, 1, 2, \dots).$$

This is known as *Newton's method* or *Newton-Raphson iteration*. In general, the convergence of this iteration is much faster than for the methods of the previous sections. However, convergence is only guaranteed if the initial value x_0 is sufficiently close to x_* .



Theorem 1.5 Let $f : [a, b] \rightarrow \mathbb{R}$ be twice continuously differentiable, and assume that $x_* \in [a, b]$ satisfies $f(x_*) = 0$ and $f'(x_*) \neq 0$. There is a $\delta > 0$ such that the Newton iteration converges whenever $|x_0 - x_*| < \delta$. Moreover, then there is a $\gamma > 0$ such that

$$(1.10) \quad |x_k - x_*| \leq \gamma |x_{k-1} - x_*|^2 \quad (k \geq 1).$$

Proof. For x close to x_* we have $f'(x) \neq 0$. Let $g(x) = x - (f'(x))^{-1}f(x)$. Then x_* is a fixed point of g , and we have $g'(x) = (f'(x))^{-2}f''(x)f(x)$. So, in particular $g'(x_*) = 0$. It follows that for $\delta > 0$ sufficiently small, and $\alpha = x_* - \delta$, $\beta = x_* + \delta$, the interval $[\alpha, \beta]$ is mapped to itself under g . The fixed point iteration for g , which is the same as the Newton iteration for f , thus converges to x_* if our initial value x_0 lies in $[\alpha, \beta]$.

Further we have

$$\begin{aligned} 0 &= f(x_k) + f'(x_k)(x_{k+1} - x_k), \\ 0 &= f(x_*) = f(x_k) + f'(x_k)(x_* - x_k) + \frac{1}{2}f''(\xi_k)(x_* - x_k)^2 \end{aligned}$$

with ξ_k between x_* and x_k (Taylor's theorem). By subtraction it is seen that

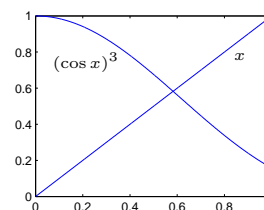
$$x_{k+1} - x_* = \frac{1}{2}(f'(x_k))^{-1}f''(\xi_k)(x_* - x_k)^2,$$

from which the proof follows with $\gamma = \frac{1}{2} \max_{\xi, \eta \in [\alpha, \beta]} |f''(\xi)| / |f'(\eta)|$. \square

Illustration. Consider the equation $f(x) = 0$ with

$$f(x) = x - (\cos x)^3.$$

By drawing the graphs of x and $(\cos x)^3$ it is clear that the equation will have a unique solution in the interval $[0, 1]$.



First we try to solve the equation with Newton's method. Using the starting value $x_0 = 0$ the iterates x_1, x_2, \dots, x_6 , are found to be:

x_0	x_1	x_2	x_3	x_4	x_5	x_6
0.000000	1.000000	0.515084	0.583029	0.582440	0.582440	0.582440

In this table, only the first six decimal digits are presented, and for the digits that are not yet correct a small font is used. After a somewhat hesitant start, with

initial overshoot, the Newton iterates converge very rapidly to the correct value x_* which is computed with fifteen digits accuracy as $x_* = 0.58244007115820$.

For this problem we can also try fixed point iteration. A natural choice for the iteration function is $g(x) = (\cos x)^3$. However, with this choice we have $g'(x) = -3 \sin x (\cos x)^2$, which happens to be larger than one in modulus near x_* . Indeed, the iteration does not converge:

x_0	x_1	x_2	x_3	x_4	x_5	x_6
0.000000	1.000000	0.157728	0.963220	0.186051	0.949115	0.197546

Convergence can be achieved with another fixed point iteration function. Consider $g(x) = x - c f(x)$ with $c \in \mathbb{R}$ still to be determined. It is clear that for any $c \neq 0$ the fixed point of g will be the zero of f . To choose a suitable value of c , note that $f'(x) = 1 + 3 \sin x (\cos x)^2 \in [1, 4]$ if $x \in [0, 1]$. Therefore $g'(x) = 1 - c f'(x)$ will assume values between $1 - c$ and $1 - 4c$. For $c = \frac{2}{5}$ we thus have $|g'(x)| \leq \frac{3}{5}$. With this choice the iteration converges:

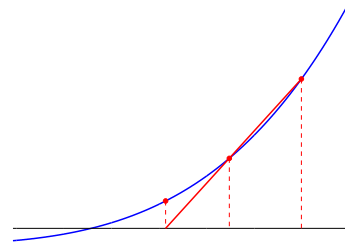
x_0	x_1	x_2	x_3	x_4	x_5	x_6
0.000000	0.400000	0.552554	0.578212	0.581848	0.582357	0.582428

In fact the rate of convergence is faster than might be expected. That is caused by the fact that $|g'(x_*)|$ turns out to be approximately 0.14. The value $\frac{3}{5}$ found above is a rather crude over-estimation.

Remark 1.6 There are many variants of Newton's method, obtained by approximating the derivative $f'(x_k)$. For example, replacing $f'(x_k)$ by the difference quotient $(f(x_k) - f(x_{k-1})) / (x_k - x_{k-1})$ leads to the *secant method*

$$(1.11) \quad x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k).$$

Here x_{k+1} depends on the two previous values x_k and x_{k-1} , so we now need two starting values x_0 and x_1 . Computation of the derivative is here avoided. This can be an advantage, for example if f is not given in closed form, but only a recipe is provided to compute $f(x)$ for given x by some (numerical) procedure.



The convergence of this secant method is in general slower than for Newton's method. If x_{k-1} and x_k are close to x_* it can be shown under the conditions of the previous theorem that

$$|x_{k+1} - x_*| \leq \tilde{\gamma} |x_k - x_*| |x_{k-1} - x_*|,$$

with a positive constant $\tilde{\gamma}$. If $|x_k - x_*|$ is much smaller than $|x_{k-1} - x_*|$ this is obviously not as good as the quadratic bound in (1.10). In fact, in computations it is often observed that $|x_{k+1} - x_*|$ is proportional to $|x_k - x_*|^p$ with order $p = \frac{1}{2}(1 + \sqrt{5}) \approx 1.62$. (This can be proven under suitable technical assumptions.) \diamond

1.4 Systems of Equations

The fixed point iteration and Newton's method can be used to solve systems of nonlinear equations, where we work with vectors $v \in \mathbb{R}^m$ instead of scalars $x \in \mathbb{R}$. For $F : \mathbb{R}^m \rightarrow \mathbb{R}^m$ we will use the notation

$$F(v) = \begin{pmatrix} F_1(v) \\ \vdots \\ F_m(v) \end{pmatrix} \quad \text{for } v = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} \in \mathbb{R}^m.$$

The $m \times m$ Jacobian matrix, containing all partial derivatives $\frac{\partial F_i(v)}{\partial v_j}$, is denoted as

$$F'(v) = \begin{pmatrix} \frac{\partial F_1(v)}{\partial v_1} & \cdots & \frac{\partial F_1(v)}{\partial v_m} \\ \vdots & & \vdots \\ \frac{\partial F_m(v)}{\partial v_1} & \cdots & \frac{\partial F_m(v)}{\partial v_m} \end{pmatrix}.$$

This will also be written more compactly as $v = (v_j)$, $F(v) = (F_j(v)) \in \mathbb{R}^m$ and $F'(v) = (\partial F_i(v)/\partial v_j) \in \mathbb{R}^{m \times m}$. Thus, subindices are used for the components. Of course, a vector itself may have a subindex already. For a sequence of vectors u_0, u_1, u_2, \dots , the j -th component of $u_k \in \mathbb{R}^m$ can be denoted as $(u_k)_j$. It should always be clear from the context whether v_j is a vector itself or a component of a vector v .

Let $F : \mathbb{R}^m \rightarrow \mathbb{R}^m$. We want to find $u_* \in \mathbb{R}^m$ that solves the system of equations

$$(1.12) \quad F(u) = 0.$$

Newton's method for this system reads

$$(1.13) \quad u_{k+1} = u_k - (F'(u_k))^{-1} F(u_k) \quad (k = 0, 1, 2, \dots).$$

This produces vectors $u_k \in \mathbb{R}^m$ that converge – hopefully – towards u_* .

It should be noted that (1.13) should *not* be implemented this way. Computing the inverse of a matrix is very expensive in terms of computing time. Instead, first the *linear system* $F'(u_k)v_k = F(u_k)$ is to be solved, and then $u_{k+1} = u_k - v_k$.

For systems with a large dimension m it can be advantageous not to work with the exact Jacobian matrix $F'(u_k)$, but with an approximation $A(u_k)$. The iteration then becomes

$$(1.14) \quad u_{k+1} = u_k - (A(u_k))^{-1} F(u_k) \quad (k = 0, 1, 2, \dots).$$

In general, the convergence will become slower, but each iteration step in (1.14) may be cheaper than in (1.13).

There are possible choices for such modification. For example, (i) the partial derivatives $\partial F_i(v)/\partial v_j$ in the Jacobian matrix can be replaced by difference quotients $\frac{1}{h}(F_i(v + he_j) - F_i(v))$, where e_j is the j -th unit vector in \mathbb{R}^m (i.e., all components of e_j are 0 except for the j -th component which equals 1). An

other possibility, that can be used if u_0 is known to be close to u_* , is to take (ii) $A(v) = F'(u_0)$, so that the partial derivatives only need to be computed once. This may also reduce the work needed to solve the linear systems in the iteration (because only one LU -decomposition will be needed; see next section).

The fixed point iteration to find an approximate solution for the system $G(u) = u$ reads $u_{k+1} = G(u_k)$. This corresponds to (1.14) with $F(v) = v - G(v)$ and $A(v) = I$, the identity matrix. The result of Theorem 1.3 remains essentially valid: if G maps a closed set $\mathcal{D} \subset \mathbb{R}^m$ into itself and $\|G(u) - G(v)\| \leq \theta \|u - v\|$ for all $u, v \in \mathcal{D}$ with $\theta \in (0, 1)$ in some suitable norm, then the iteration converges to the unique fixed point u_* in \mathcal{D} . This result – and its generalization to Banach spaces – is known as the *contraction mapping theorem*. The contraction property of G can be established by verifying that its partial derivatives are continuous and $\|G'(u)w\| \leq \theta \|w\|$ for all $u \in \mathcal{D}$ and $w \in \mathbb{R}^m$.

In the same way, the result of Theorem 1.5 on convergence of Newton's method generalizes, ensuring *local convergence*, starting sufficiently close to a root u_* . Of course, it would be very desirable to have knowledge on the regions of attraction $\mathcal{D}(u_*)$ consisting of those u_0 for which we have convergence towards u_* . However, these sets are complicated in general.

Example 1.7 As an interesting illustration, consider the Newton iteration $z_{k+1} = z_k + f(z_k)/f'(z_k)$ in the complex plane \mathbb{C} for the function $f(z) = z^3 - 1$, leading to

$$z_{k+1} = \frac{2}{3}z_k + \frac{1}{3}z_k^{-2}.$$

This iteration in \mathbb{C} is the same as for the corresponding function $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ with $F_1(x, y) = \operatorname{Re} f(x + iy)$ and $F_2(x, y) = \operatorname{Im} f(x + iy)$ (see Exercise 1.6).

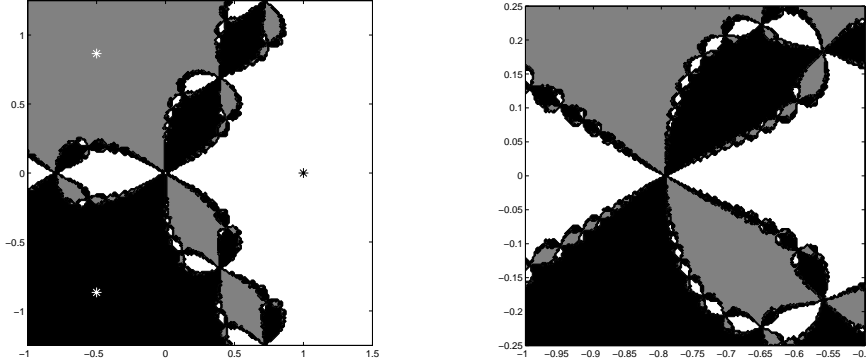


Figure 1.1: Domains of attraction of the Newton iteration. The right panel contains a zoom of the left panel around $z = -0.75$.

The domains of attraction for the three roots $z_* = 1$ and $-\frac{1}{2} \pm \frac{1}{2}i\sqrt{3}$ are shown in Figure 1.1 as white, gray and black regions. The plot in the left panel extends to $-1 \leq x \leq 1.5$, $-1.25 \leq y \leq 1.25$. The right panel, containing an enlargement around $x = -0.75$, $y = 0$, reveals the fractal structure of the sets. \diamond

Remark 1.8 If we want to find all zeros in \mathbb{C} of a polynomial P , the scalar complex version of Newton's method can be used with $f(z) = P(z)$ to find a first zero, say z_1 . Next, we can set $f(z) = Q_1(z) = P(z)/(z - z_1)$, find a zero z_2 of this new function, and continue with $f(z) = Q_2(z) = P(z)/((z - z_1)(z - z_2))$, etc. This technique is called *deflation*.

To avoid inaccuracies, due to the fact that the computed z_j will only be approximations to the genuine roots, it is generally recommended to perform one or two additional Newton iterations with the original polynomial P after having found an approximation with the deflated function Q_j . \diamond

1.5 Exercises

Exercise 1.1. Suppose $|g'(x_*)| > 1$. Is it then possible to have local convergence of the fixed point iteration (1.4)?

Exercise 1.2. In practice, one would like to terminate the Newton iteration if some specified level of accuracy has been achieved, say $|x_k - x_*| \leq Tol$, with given desired accuracy Tol .

Assume $x_* \in [\alpha, \beta]$ and $|f'(x)|^{-1} \leq \gamma$ for all $x \in [\alpha, \beta]$. Show, from the mean-value theorem, that if x_k is in this interval and $|f(x_k)| \leq \varepsilon$, then $|x_k - x_*| \leq \gamma \cdot \varepsilon$.

Exercise 1.3. The equation $x + \log x = 0$ with $x \in [0, 1]$ can be written in the following ways: (i) $x = -\log x$, (ii) $x = e^{-x}$, (iii) $x = \frac{1}{2}(x + e^{-x})$. Each possibility gives rise to a different fixed point iteration. Which one do you prefer? Show that with the last option $|x_k - x_*| < 3^{-k}$ for $k \geq 1$.

Exercise 1.4. To compute $\sqrt[p]{c}$ for a positive number c and integer $p \geq 2$, we consider Newton's method with $x_0 > 0$, giving

$$x_{k+1} = \frac{p-1}{p} x_k + \frac{c}{p} x_k^{1-p} \quad (k = 0, 1, 2, \dots).$$

(a) Show that the sequence $\{x_k\}$ is monotonically decreasing for $k \geq 1$ with limit $\sqrt[p]{c}$.

(b) Consider the convergence for $c = 0$, and compare this with the result of Theorem 1.5.

Exercise 1.5. How does Newton's method read for a linear system of equations $Av = b$.

*Exercise 1.6.** Let the components of $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ be defined by

$$F_1(x, y) = x^3 - 3xy^2 - 1, \quad F_2(x, y) = 3x^2y - y^3.$$

Show that the Newton iteration for this function coincides with the one for the complex map $f(z) = z^3 - 1$ used in the Example 1.7.

Exercise 1.7 (programming). Compute the fixed point of the cosine function $x_* = \cos x_*$ up to 12 decimal places using (i) bisection on $[0, 1]$, (ii) fixed point iteration with $x_0 = 0$, (iii) Newton's method with $x_0 = 0$.

Exercise 1.8 (programming). Let

$$F(v) = \begin{pmatrix} v_1^2 + 4v_1v_2 - 3 \\ v_1 - v_2^2 - 3 \end{pmatrix} \quad \text{for } v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \in \mathbb{R}^2.$$

Compute the $\{u_k\}$ from Newton's method starting with $u_0 = (1, -1)^T$. Discuss possibilities to stop this iteration.

2 Linear Systems

In this section we will discuss some basic numerical methods to solve linear systems of equations

$$(2.1) \quad Av = b,$$

with given matrix $A = (a_{ij}) \in \mathbb{R}^{m \times m}$ and vector $b = (b_i) \in \mathbb{R}^m$. The solution $v \in \mathbb{R}^m$ exists and is unique if and only if the matrix A is nonsingular.

Overdetermined systems, with more equations than unknowns, will also be briefly considered in this section. Then a least-squares solution can be found by the so-called normal equations.

Finally we will discuss the influence on v of small errors in b . The components of this vector may have been obtained, for instance, by some measurement, with a certain error. Also the rounding of numbers on a computer will lead to small errors in the data.

2.1 Gaussian Elimination and LU -Decomposition

A straightforward –but still popular– method to solve linear equations is Gaussian elimination. The system (2.1), written out, reads

$$(2.2) \quad \begin{array}{rcll} a_{11}v_1 + a_{12}v_2 + \cdots + a_{1m}v_m & = & b_1, \\ a_{21}v_1 + a_{22}v_2 + \cdots + a_{2m}v_m & = & b_2, \\ \vdots & & \vdots \\ a_{m1}v_1 + a_{m2}v_2 + \cdots + a_{mm}v_m & = & b_m. \end{array}$$

Assume $a_{11} \neq 0$. Then we can eliminate v_1 from the last $m - 1$ equations by subtracting from the i -th equation the first equation multiplied by $\lambda_{i1} = a_{i1}/a_{11}$. After this, the system becomes

$$(2.3) \quad \begin{array}{rcl} a_{11}v_1 + a_{12}v_2 + \cdots + a_{1m}v_m & = & b_1, \\ a_{22}^{(2)}v_2 + \cdots + a_{2m}^{(2)}v_m & = & b_2^{(2)}, \\ \vdots & & \vdots \\ a_{m2}^{(2)}v_2 + \cdots + a_{mm}^{(2)}v_m & = & b_m^{(2)}, \end{array}$$

where $a_{ij}^{(2)} = a_{ij} - \lambda_{i1}a_{1j}$ and $b_i^{(2)} = b_i - \lambda_{i1}b_1$. This provides a system of $m-1$ equations for v_2, \dots, v_m , with an extra equation that determines v_1 . If $a_{22}^{(2)} \neq 0$ we can proceed in the same way to obtain a system of $m-2$ equations in the unknowns v_3, \dots, v_m with coefficients $a_{ij}^{(3)} = a_{ij}^{(2)} - \lambda_{i2}a_{2j}^{(2)}$ and $b_i^{(3)} = b_i^{(2)} - \lambda_{i2}b_2^{(2)}$, $\lambda_{i2} = a_{i2}^{(2)}/a_{22}^{(2)}$. Repeating this $m-1$ times we are finally left with a single equation $a_{mm}^{(m)}v_m = b_m^{(m)}$. Then, collecting the first equation from each step gives

$$(2.4) \quad \begin{aligned} a_{11}^{(1)} v_1 + a_{12}^{(1)} v_2 + \cdots + a_{1m}^{(1)} v_m &= b_1^{(1)}, \\ a_{22}^{(2)} v_2 + \cdots + a_{2m}^{(2)} v_m &= b_2^{(2)}, \\ &\vdots \\ a_{mm}^{(m)} v_m &= b_m^{(m)}. \end{aligned}$$

where we denoted $a_{ij}^{(1)} = a_{ij}$ and $b_i^{(1)} = b_i$. This system is now easily solved:

$$(2.5) \quad v_m = \frac{1}{a_{mm}^{(m)}} b_{mm}^{(m)} \quad \text{and} \quad v_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{j>i} a_{ij}^{(i)} v_j \right) \quad (i = m-1, \dots, 2, 1).$$

The elements $a_{kk}^{(k)}$ are called the *pivot elements*. If in the k -th step $a_{kk}^{(k)} = 0$, then there is some other $a_{lk}^{(k)} \neq 0$, because otherwise the first k columns are linearly dependent, and A is singular. Hence we can perform a simple permutation, interchanging the rows k and l , and then continue the procedure. In practice, it will be necessary to perform such a row interchange not only if $a_{kk}^{(k)}$ is precisely zero, but also if it is nearly zero, because division by a very small number can lead to large round-off errors. We then select the index $l \geq k$ such that $a_{lk}^{(k)}$ has the largest modulus. This is called Gaussian elimination with *partial pivoting*. (One can also use *complete pivoting*, where also columns are interchanged, but in practice the partial row pivoting is usually sufficient.)

The permutations are done during the process. Afterwards all these row permutations together are described by a permutation matrix P such that for the permuted system $PAv = Pb$ the Gaussian elimination can proceed without additional row permutations.

Setting $A^{(1)} = PA$ and $b^{(1)} = Pb$ the total process thus can be described formally as

$$[PA, Pb] = [A^{(1)}, b^{(1)}] \rightarrow [A^{(2)}, b^{(2)}] \rightarrow \dots \rightarrow [A^{(m)}, b^{(m)}] = [U, b^{(m)}],$$

where the first $k-1$ columns of $A^{(k)}$ have zero entries below the diagonal, and U is the resulting upper triangular matrix.

In the following, let I stand for the $m \times m$ identity matrix, and let E_{ij} be the $m \times m$ matrix that has entry 1 at the i, j -th position and entries 0 elsewhere. Then $E_{ij}A$ is the matrix with all rows zero, except for the i -th row which is just the j -th row of A . The steps in the Gaussian elimination can be described as

$$A^{(k+1)} = L_k A^{(k)}, \quad b^{(k+1)} = L_k b^{(k)} \quad \text{with} \quad L_k = I - \sum_{j>k} \lambda_{jk} E_{jk}.$$

These matrices L_k have the form

$$L_1 = \begin{pmatrix} 1 & & & & \\ -\lambda_{21} & 1 & & & \\ -\lambda_{31} & & 1 & & \\ \vdots & & & \ddots & \\ -\lambda_{n1} & & & & 1 \end{pmatrix}, \quad L_2 = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & -\lambda_{32} & 1 & & \\ & \vdots & & \ddots & \\ & -\lambda_{n2} & & & 1 \end{pmatrix},$$

etc., with numbers zero on the empty positions.

Theorem 2.1 Suppose A is nonsingular. Then Gaussian elimination gives

$$PA = LU, \quad L = \left(I + \sum_{i>j} \lambda_{ij} E_{ij} \right),$$

with permutation matrix P and upper triangular matrix U .

Proof. It is clear that $U = A^{(n)}$ equals $U = L_{m-1}L_{m-2} \cdots L_1PA$, that is,

$$PA = (L_{m-1}L_{m-2} \cdots L_1)^{-1}U.$$

It remains to show that this inverse equals L .

To see this we can apply the same procedure to L . Multiplication of L by L_1 from the left eliminates the elements of the first column below the diagonal, then multiplication by L_2 from the left clears the elements of the second column below the diagonal, and so on. Therefore $(L_{m-1}L_{m-2} \cdots L_1)L = I$. \square

Such decomposition of PA into a lower triangular L and upper triangular U is called a *LU-decomposition* or *LU-factorization*. Solving $Av = b$ amounts to (i) compute the *LU*-decomposition, and (ii) solve the triangular systems $Lw = Pb$, $Uv = w$.

Saving the factors L and U is useful if we need to solve, one after the other, a number of linear systems $Av = b$ with different vectors b . An example is provided by the modified Newton iteration (1.14) with fixed matrix $A = F'(u_0)$, where we will get a right hand side vector $b = -F(u_{k-1})$ in the k -th iteration step.

Further we note that, for a given permutation, the factors L and U are unique. For, if we have a second decomposition $PA = \tilde{L}\tilde{U}$, then $\tilde{L}^{-1}L = \tilde{U}U^{-1}$. Since $\tilde{L}^{-1}L$ is lower triangular and $\tilde{U}U^{-1}$ is upper triangular, there is a diagonal D such that $L = D\tilde{L}^{-1}$ and $\tilde{U} = DU$. The requirement that the diagonal elements of L and \tilde{L} are 1 specifies $D = I$.

Computational costs. To estimate the computational costs of Gaussian elimination, first note that the computation of $A^{(2)}$ from $A^{(1)} = A$ requires $(m-1)$ divisions and $(m-1)^2$ multiplications and additions. So for large m this takes approximately $(m-1)^2$ operations, where an 'operation' is defined as one multiplication and one addition. Then to find $A^{(3)}$ we need $(m-2)^2$ operations. In total, finding the *LU* decomposition will require approximately $\sum_{j=1}^{m-1} j^2 \sim \int_0^m x^2 dx = \frac{1}{3}m^3$ operations.

The computation of $b^{(2)}$ will take $(m-1)$ operations, and we get $b^{(m)}$ with $(m-1) + \cdots + 2 + 1 = \frac{1}{2}m^2$ operations. Likewise solving the triangular system will take $\frac{1}{2}m^2$ operations and m divisions. For large m this negligible compared to the $\frac{1}{3}m^3$ operations for finding L and U .

Gaussian elimination to solve (2.1) thus takes approximately $\frac{1}{3}m^3$ operations. Estimating the computational costs this way, by counting the number of operations, only gives a rough indication for the actual computation time. Memory handling is not taken into account and computers increasingly possess vector and parallel capacities. Still, it gives an indication.

Round-off errors. It was stated above that pivoting is necessary to avoid round-off errors. To illustrate this we consider the following example.

Example 2.2 Consider

$$A = \begin{pmatrix} 0.005 & 10 \\ 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 10 \\ 2 \end{pmatrix}.$$

Then performing Gauss elimination with exact arithmetic gives

$$\lambda_{21} = \frac{1}{0.005} = 200, \quad a_{22}^{(2)} = 1 - \lambda_{21} \cdot 10 = -1999, \quad b_2^{(2)} = 2 - \lambda_{21} \cdot 10 = -1998,$$

and $v_2 = \frac{1998}{1999}, \quad v_1 = \frac{1}{0.005} \cdot (10 - 10 \cdot v_2) = \frac{2000}{1999}.$

In decimal representation $v_2 = 1.0005\dots$ and $v_1 = 0.99949\dots$.

Now let us simulate floating point arithmetic with three digits accuracy by rounding any real number x to $\tilde{x} = \pm 0.n_1 n_2 n_3 \cdot 10^m$ with integer m and integers n_j between 0 and 9 (and $n_1 \neq 0$). Then we get

$$\tilde{\lambda}_{21} = 200, \quad \tilde{a}_{22}^{(2)} = -2000, \quad \tilde{b}_2^{(2)} = -2000,$$

and $\tilde{v}_2 = 1, \quad \tilde{v}_1 = 0.$

The reason for this bad result is the relation $v_1 = -\frac{10}{0.005}v_2 + \frac{10}{0.005}$. The small error in v_2 will be amplified, due to the fact that the pivot element $a_{11} = 0.005$ is small. If we repeat the process, starting with an interchange of the first and second rows, then this problem will not arise. \diamond

Standard computations are nowadays usually done with 16 digits in floating point representation. In the above example, without pivoting, that would give an accuracy around 10^{-13} . This will deteriorate if we take a_{11} is closer to zero. Moreover, if there are many steps in a process, repeated loss of three digits accuracy easily leads to a bad overall result.

2.2 Positive Definite Matrices and Band Matrices

If $A = (a_{ij})$ is a real $m \times n$ matrix (m rows and n columns) we will write $A \in \mathbb{R}^{m \times n}$. Interchanging rows for columns, and vice versa, gives the *transpose* matrix $A^T = (a_{ji}) \in \mathbb{R}^{n \times m}$. Likewise, the transpose v^T of a (column) vector $v \in \mathbb{R}^m = \mathbb{R}^{m \times 1}$ is the corresponding row vector in $\mathbb{R}^{1 \times m}$.

The real $m \times m$ matrix A is called *symmetric* if $A^T = A$, and a symmetric matrix is called *positive definite* if $v^T A v > 0$ for any nonzero $v \in \mathbb{R}^m$. Positive definite matrices arise in many applications. The next result shows that then the resulting systems can be solved with Gaussian elimination without permutations, and in the LU -decomposition we have $U = DL^T$ with a diagonal matrix D that has positive diagonal entries. Writing $C = \sqrt{D}L^T$ gives $A = C^T C$.

Theorem 2.3 Let A be symmetric and positive definite. Then there is an upper triangular matrix, with positive diagonal entries, such that $A = C^T C$.

Proof. To prove this result we can use induction with respect to the dimension m . The statement of the theorem is obviously true if $m = 1$. Now let $m > 1$ and write

$$A = \begin{pmatrix} \bar{A} & \bar{a} \\ \bar{a}^T & \alpha \end{pmatrix}, \quad C = \begin{pmatrix} \bar{C} & \bar{c} \\ \bar{c}^T & \gamma \end{pmatrix},$$

with scalars α, γ , and with $\bar{A}, \bar{C} \in \mathbb{R}^{\bar{m} \times \bar{m}}$, $\bar{a}, \bar{c} \in \mathbb{R}^{\bar{m}}$, $\bar{m} = m-1$, and zero vector $\bar{o} \in \mathbb{R}^{\bar{m}}$. Since A is positive definite, the same holds for \bar{A} .

Suppose that $\bar{A} = \bar{C}^T \bar{C}$ and \bar{C} is upper triangular with positive diagonal entries (induction assumption). Then $A = C^T C$ iff $\bar{a} = \bar{C}^T \bar{c}$ and $\alpha = \bar{c}^T \bar{c} + \gamma^2$. So, we can compute \bar{c} by solving the lower triangular system $\bar{C}^T \bar{c} = \bar{a}$ and then set $\gamma = \sqrt{\alpha - \bar{c}^T \bar{c}}$. It seems this γ might be purely imaginary. However we have $\det(A) = \det(C^T) \det(C) = (\gamma \det(\bar{C}))^2$, and since the determinant of the positive definite matrix A is positive (it is the product of the eigenvalues and these are all positive) we must have $\gamma^2 > 0$. \square

The decomposition $A = C^T C$ for positive definite A is called a *Cholesky decomposition*. It can be seen from the construction used in the above proof – with the repeated lower triangular systems – that the number of operations in a Cholesky decomposition is roughly $\frac{1}{6}m^3$, which is significantly lower than for a general LU -decomposition.

The $m \times m$ matrix A is called a *band matrix* if there is an integer $k < m$ such that $a_{ij} = 0$ whenever $|i - j| > k$. The number $2k + 1$ is called the *bandwidth* of the matrix. Familiar examples are *diagonal* matrices ($k = 0$) and *tri-diagonal* matrices ($k = 1$).

If Gaussian elimination can be performed without permutations, then the cost is greatly reduced for banded matrices, because L and U will also be banded. In terms of operations, it becomes of the order of $k^2 m$, instead of m^3 for a full matrix. Moreover, and often more importantly, *storage* is greatly reduced if $k \ll m$.

Example 2.4 Let

$$A = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \gamma_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \gamma_m & \alpha_m \end{pmatrix}, \quad L = \begin{pmatrix} 1 & & & \\ \lambda_2 & 1 & & \\ & \ddots & \ddots & \\ & & \lambda_m & 1 \end{pmatrix}, \quad U = \begin{pmatrix} \rho_1 & \sigma_2 & & \\ & \rho_2 & \ddots & \\ & & \ddots & \sigma_m \\ & & & \rho_m \end{pmatrix}.$$

We have $A = LU$ iff $\alpha_1 = \rho_1$ and $\gamma_j = \lambda_j \rho_{j-1}$, $\alpha_j = \lambda_j \sigma_j + \rho_j$, $\beta_j = \sigma_j$ ($2 \leq j \leq m$). Assuming that Gaussian elimination can be applied to A without permutations, we can thus compute the $\lambda_j, \rho_j, \sigma_j$ recursively as $\rho_1 = \alpha_1$,

$$\lambda_j = \gamma_j / \rho_{j-1}, \quad \rho_j = \alpha_j - \lambda_j \beta_j, \quad \sigma_j = \beta_j \quad (2 \leq j \leq m).$$

This requires $m-1$ divisions and $m-1$ multiplications with addition, say $2(m-1)$ operations. Finally, solving $Av = b$ reduces to $Lw = b$, $Uv = w$, and these two bi-diagonal systems together require $3m-2$ operations. In total we therefore just need $5m-4$ operations. Storage of L and U takes $3m-2$ positions. \diamond

2.3 Overdetermined Systems: the Normal Equations

Let $\|u\| = \sqrt{u^T u}$ be the Euclidian norm on \mathbb{R}^m . If A is an $m \times n$ matrix with $m > n$ and $b \in \mathbb{R}^m$, then the system $Av = b$ reads

$$(2.6) \quad \begin{aligned} a_{11}v_1 + a_{12}v_2 + \cdots + a_{1n}v_n &= b_1, \\ a_{21}v_1 + a_{22}v_2 + \cdots + a_{2n}v_n &= b_2, \\ \vdots & \\ a_{m1}v_1 + a_{m2}v_2 + \cdots + a_{mn}v_n &= b_m. \end{aligned}$$

This has in general no solution $v \in \mathbb{R}^n$. But we can find a v such that

$$(2.7) \quad \|Av - b\| = \min_{w \in \mathbb{R}^n} \|Aw - b\|.$$

Such a v is called a *least squares* solution of (2.7).

Let $\varphi(v) = \|Av - b\|^2 = v^T A^T A v - 2b^T A v + b^T b$. Then

$$\frac{1}{h} (\varphi(v + hw) - \varphi(v)) = 2(v^T A^T A - b^T A)w + h \cdot w^T A^T A w,$$

and taking $h \rightarrow 0$ we see that $\varphi'(v) = 2(v^T A^T A - b^T A)$. Consequently, the least squares solution satisfies

$$(2.8) \quad A^T A v = A^T b.$$

These linear equations, with $n \times n$ matrix $A^T A$, are called the *normal equations*.

The matrix $A^T A$ is obviously symmetric. Moreover, it is easily seen that if the columns of A are linearly independent, then $A^T A$ is positive definite, so then the least squares solution is unique and it can be found by a Cholesky decomposition.

Least-squares problems often arise when one wants to fit some simple function to given (measured) data.

Example 2.5 Let points $(x_i, y_i) \in \mathbb{R}^2$ ($1 \leq i \leq m$) be given. We want to find the coefficients of the polynomial $p(x) = \alpha x^2 + \beta x + \gamma$ such that

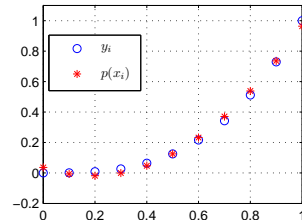
$$\sum_{i=1}^m |p(x_i) - y_i|^2 \quad \text{is minimized.}$$

This is a least squares problem with $a_{ij} = x_i^{j-1}$, $b_i = y_i$ and $v = (\alpha, \beta, \gamma)^T \in \mathbb{R}^3$.

Let us take as a simple illustration

$$x_i = \frac{i-1}{k}, \quad y_i = x_i^3$$

for $i = 1, \dots, k+1$ with $k = 10$. The values $p(x_i)$ of the quadratic polynomial with best least-squares fit is shown in the plot on the right.



A more interesting illustration is provided by the result of Exercise 2.6. ◇

2.4 The Condition Number of a Matrix

To measure the length of a vector we will use a *norm* $\|\cdot\|$ on the vector space \mathbb{R}^m . Recall that a function $\varphi : \mathbb{R}^m \rightarrow \mathbb{R}$ is called a norm if (i) $\varphi(v) \geq 0$ and $\varphi(v) = 0$ only if $v = 0$, (ii) $\varphi(cv) = |c|\varphi(v)$, and (iii) $\varphi(v + w) \leq \varphi(v) + \varphi(w)$, for any $v, w \in \mathbb{R}^m$ and $c \in \mathbb{R}$. Some common norms are

$$(2.9) \quad \begin{cases} \|v\|_1 = \sum_{j=1}^m |v_j|, & \text{the sum norm or } l_1\text{-norm,} \\ \|v\|_2 = \sqrt{\sum_{j=1}^m |v_j|^2}, & \text{the Euclidian norm or } l_2\text{-norm,} \\ \|v\|_\infty = \max_{1 \leq j \leq m} |v_j|, & \text{the maximum norm or } l_\infty\text{-norm,} \end{cases}$$

for $v = (v_j) \in \mathbb{R}^m$. The Euclidian norm of v equals $\|v\|_2 = \sqrt{v^T v}$, and we have the Hölder inequality $\|v\|_2^2 \leq \|v\|_1 \cdot \|v\|_\infty$ (for any $v \in \mathbb{R}^m$).

Given a vector norm $\|\cdot\|$ on \mathbb{R}^m the *induced matrix norm* for $m \times m$ matrices A is defined by

$$(2.10) \quad \|A\| = \max_{v \neq 0} \frac{\|Av\|}{\|v\|},$$

that is, $\|A\|$ is the smallest number α such that $\|Av\| \leq \alpha\|v\|$ for all $v \in \mathbb{R}^m$. The induced matrix norm for the three norms in (2.9) are (see exercise below)

$$(2.11) \quad \begin{cases} \|A\|_1 = \max_{1 \leq j \leq m} \sum_{i=1}^m |a_{ij}|, \\ \|A\|_2 = \sqrt{\text{max. eigenvalue of } A^T A}, \\ \|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^m |a_{ij}|. \end{cases}$$

An important property is $\|AB\| \leq \|A\|\|B\|$ for any two matrices $A, B \in \mathbb{R}^{m \times m}$. For the identity matrix I we have the norm $\|I\| = 1$.

As we will see shortly, the *condition number*

$$(2.12) \quad \text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

indicates how well a linear system $Av = b$ can be solved.

Let us consider, along with the linear system $Av = b$, the approximate system $\tilde{A}\tilde{v} = \tilde{b}$. Suppose that

$$(2.13) \quad \|\tilde{A} - A\| \leq \epsilon_A \|A\|, \quad \|\tilde{b} - b\| \leq \epsilon_b \|b\|.$$

For example, \tilde{A}, \tilde{b} may stand for the computer representations of the exact A and b , with relative errors ϵ_A and ϵ_b . We want to know how much such errors will influence the outcome.

Theorem 2.6 Suppose A is nonsingular, $\kappa = \text{cond}(A)$ and $\epsilon_A \cdot \kappa < 1$. Then

$$\frac{\|\tilde{v} - v\|}{\|v\|} \leq \frac{\kappa}{1 - \epsilon_A \kappa} \cdot (\epsilon_A + \epsilon_b).$$

Proof. Since $\tilde{b} - b = \tilde{A}\tilde{v} - Av = (\tilde{A} - A)\tilde{v} + A(\tilde{v} - v)$, we have

$$\tilde{v} - v = A^{-1} \left(-(\tilde{A} - A)\tilde{v} + (\tilde{b} - b) \right).$$

Taking norms, and using $\|\tilde{v}\| \leq \|v\| + \|\tilde{v} - v\|$ and $\|b\| = \|Av\| \leq \|A\|\|v\|$, it follows that

$$\|\tilde{v} - v\| \leq \|A^{-1}\| \left(\epsilon_A \|A\| (\|v\| + \|\tilde{v} - v\|) + \epsilon_b \|A\| \|v\| \right),$$

from which the proof is now directly obtained. \square

The above result show that solving a linear system where the matrix has a large condition number may lead to a loss in accuracy. Such systems are called *ill-conditioned*. This is not related to the numerical procedure that is used to solve the system. The matrix in Example 2.2 has a moderate condition number in any of the norms (2.9), and in that example we just had to change the computation a bit (by a permutation) to avoid large round-off errors.

Example 2.7 A notorious example for ill-conditioned systems involves the Hilbert matrix $H = (h_{ij})$, $h_{ij} = \int_0^1 x^{i-1} x^{j-1} dx = \frac{1}{i+j-1}$ for $1 \leq i, j \leq m$. The condition number of this matrix quickly becomes large.

In the following table the condition numbers $\text{cond}_\infty(H)$ for the maximum norm are given, together with the errors $\|v - e\|_\infty$ for the problem $Hv = b$ where $b = He$ with $e = (1, \dots, 1)^T$ in \mathbb{R}^m .

m	4	6	8	10	12	14
$\text{cond}_\infty(H)$	$2.84 \cdot 10^4$	$2.91 \cdot 10^7$	$3.39 \cdot 10^{10}$	$3.54 \cdot 10^{13}$	$3.71 \cdot 10^{16}$	$4.08 \cdot 10^{18}$
$\ v - e\ _\infty$	$2.96 \cdot 10^{-13}$	$4.29 \cdot 10^{-10}$	$5.46 \cdot 10^{-7}$	$4.51 \cdot 10^{-4}$	$3.04 \cdot 10^{-1}$	$5.23 \cdot 10^1$

The entries for this table have been computed with MATLAB; a warning was issued for $m \geq 12$ stating that the “Matrix is close to singular or badly scaled. Results may be inaccurate.” It is clear from the table that already the results for lower m are not very accurate. \diamond

2.5 Exercises

Exercise 2.1. This first exercise is intended as a little refresher from your linear algebra course.

(a) A matrix $U \in \mathbb{R}^{m \times m}$ is called *orthogonal* if $U^T U = I$. Clearly U^T is then also orthogonal. For such a matrix, show that $\|Uv\|_2^2 = \|v\|_2^2$ for all $v \in \mathbb{R}^m$ and $\|UA\|_2 = \|AU\|_2$ for any $A \in \mathbb{R}^{m \times m}$.

(b) If A is symmetric, then it is known that $A = U\Lambda U^{-1}$ with real diagonal $\Lambda = \text{diag}(\lambda_i)$ and orthogonal U . Show that for a symmetric matrix we have $\|A\|_2 = \max_i |\lambda_i|$. Also $\|A^{-1}\|_2 = \max_i 1/|\lambda_i|$ if A is nonsingular.

(c) Let P be an $m \times m$ permutation matrix: $Pv = (v_{k_1}, \dots, v_{k_m})^T$ for $v = (v_1, \dots, v_m)^T$, with $k_i \neq k_j$ if $i \neq j$. Show that for all three norms (2.9) and for any $v \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times m}$ we have $\|Pv\| = \|v\|$ and $\|PA\| = \|A\|$.

Exercise 2.2. Prove the expression for the induced matrix norms $\|A\|_1$, $\|A\|_2$ and $\|A\|_\infty$ in (2.11). (Hint: to find $\|A\|_2$, use the fact that $A^T A$ is symmetric.)

Exercise 2.3. Show that for any nonsingular matrix A we have

$$\|A^{-1}\| = \left(\min_{v \neq 0} \frac{\|Av\|}{\|v\|} \right)^{-1}, \quad \text{cond}(A) = \frac{(\max_{\|v\|=1} \|Av\|)}{(\min_{\|v\|=1} \|Av\|)} \geq 1.$$

Exercise 2.4. For the 3×3 matrix

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix},$$

the LU decomposition can be computed by hand. What are the matrices L_1 and L_2 in this process? What happens if we change $a_{33} = 10$ to $a_{33} = 9$?

Exercise 2.5. A matrix A is called *negative definite* if $-A$ is positive definite. Let

$$A = \begin{pmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{m \times m}.$$

(a) Show that A is negative definite. (Hint: write $(Av)_j = (v_{j-1} - v_j) - (v_j - v_{j+1})$ with $v_0 = v_{m+1} = 0$, and then consider $v^T Av = \sum_{j=1}^m v_j (Av)_j$.) As a consequence, we know that $Av = b$ can be solved with $(5m-4)$ operations as in Example 2.4.

(b) The inverse $B = A^{-1}$ can be computed exactly. Consider $Av = e_k$, with $e_k = (0, \dots, 0, 1, 0, \dots, 0)^T$ the k -th unit vector. The solution v will be the k -th column of B , that is, $b_{jk} = v_j$. To compute v , introduce again $v_0 = v_{m+1} = 0$, so that

$$v_{j-1} - 2v_j + v_{j+1} = \begin{cases} 0 & \text{if } j \neq k, \\ 1 & \text{if } j = k. \end{cases}$$

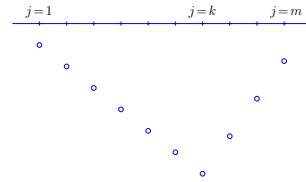
Hence, if $j \neq k$, then v_j is the average of v_{j-1} and v_{j+1} , that is, $v_j = \frac{1}{2}(v_{j-1} + v_{j+1})$. Use this property to find

$$v_j = \frac{j}{k} v_k \quad (j = 1, \dots, k-1),$$

$$v_j = \frac{m+1-j}{m+1-k} v_k \quad (j = k+1, \dots, m),$$

and then show that

$$v_k = -\left(\frac{1}{k} + \frac{1}{m+1-k}\right)^{-1}.$$



(c) Suppose $B = A^{-1}$ has been computed and stored. Then the solution of $Av = b$ can simply be obtained from $v = Bb$. How many operations are required for this matrix-vector product? How does this compare with Example 2.4?

Exercise 2.6 (programming). Let $x_i = (i - 1)/k$ for $i = 1, \dots, k + 1$ with $k = 10$. Let $y_i = f(x_i)$ with a damped oscillating function $f(x) = e^{-2x} \sin(3\pi x)$. We want to find from the normal equations the polynomial of degree $\leq s$ such that $\sum_i |p(x_i) - y_i|^2$ is minimal, with $s = 2, 3, 4, 5$. There is some freedom for doing that.

(a) Let $\phi_j(x) = x^j$ and compute the coefficients α_j such that $p(x) = \sum_{j=0}^s \alpha_j \phi_j(x)$. Compute also the condition number of $B = A^T A$ for the normal equations.

(b) Do the same but now with basis functions $\phi_0(x) = 1$, $\phi_1(x) = 2x - 1$ and

$$\phi_j(x) = \frac{2j-1}{j}(2x-1)\phi_{j-1}(x) - \frac{j-1}{j}\phi_{j-2}(x) \quad \text{for } j \geq 2.$$

(These are the so-called Legendre polynomials, shifted to $[0, 1]$).

3 Polynomial Interpolation and Approximation

Suppose the set of distinct real points x_0, x_1, \dots, x_m in $[a, b]$ is given together with corresponding numbers f_0, f_1, \dots, f_m . We want to find a function P that interpolates these data,

$$(3.1) \quad P(x_i) = f_i \quad (i = 0, 1, \dots, m).$$

In this section we will consider *polynomial interpolation*, where P is a polynomial of degree m or less.

In general, the values f_i may originate from measurements or they can be output of some other numerical procedure. When we want to assess the quality of the interpolation scheme, it will be assumed that the interpolation points are on the graph of a smooth function f , that is, $f_i = f(x_i)$.

We may also try to approximate a given function f by an interpolating polynomial, in which case we are free to choose the nodes x_j . Such polynomial approximations have many applications. For example, the integral $\int_a^b f(x) dx$ can then be approximated by $\int_a^b P(x) dx$.

3.1 Interpolation Formulas

For given nodes x_0, x_1, \dots, x_m , let the polynomials L_i be defined by

$$(3.2) \quad L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_m)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_m)}$$

for $i = 0, 1, \dots, m$.

Theorem 3.1 Assume the nodes x_0, x_1, \dots, x_m are distinct. Then the unique interpolation polynomial of degree $\leq m$ is given by

$$(3.3) \quad P(x) = \sum_{i=0}^m L_i(x) f_i.$$

Proof. The polynomial L_i has value 1 in x_i and value 0 in the other nodes x_j , $j \neq i$. It is therefore clear that P provides an interpolation polynomial of degree m at most.

Moreover, it is the only interpolation polynomial of degree $\leq m$. To see this, suppose Q is another one. Then $R = P - Q$ is a polynomial of degree $\leq m$ with $m + 1$ zeros x_j , $j = 0, \dots, m$, which implies that R is identically equal to zero. \square

Formula (3.3) is called the *Lagrange interpolation formula*. In this form it is not directly suited to evaluate the polynomials numerically. It can be rewritten in various ways to make it more suitable for computations. Of course, in view of the uniqueness, the formulas are mathematically equivalent.

Usually one wants to find $P(x)$ for a number of different values x . A popular way for multiple output values is the *Newton divided difference formula*, where

the interpolating polynomial is computed recursively (somewhat related to Exercise 3.2). This is extensively treated in the book of Stoer & Bulirsch, for example.

Another way of rewriting the Lagrange formula leads to the so-called *barycentric formulas*. For this, we introduce the polynomial

$$(3.4) \quad M(x) = (x - x_0)(x - x_1) \cdots (x - x_m),$$

and the factors

$$(3.5) \quad w_i = \frac{1}{M'(x_i)} = \frac{1}{\prod_{j \neq i} (x_i - x_j)} \quad (i = 0, 1, \dots, m).$$

Then we have $L_i(x) = M(x) w_i (x - x_i)^{-1}$, and therefore

$$(3.6) \quad P(x) = M(x) \sum_{i=0}^m \frac{w_i}{x - x_i} f_i.$$

This can be further rewritten in the form of a weighted average of the f_i values; see Exercise 3.1.

Computing the factors w_0, w_1, \dots, w_m requires order m^2 operations, and after that we can evaluate $P(x)$ for any given x with additional order m operations. This is much better than with direct use of (3.3).

3.2 The Interpolation Errors

To derive an expression for the error made by interpolation, we assume that the distinct nodes x_j are in the interval $[a, b]$, but also that the points (x_j, f_j) are on the graph of a smooth function f . Let the polynomial M be as in (3.4).

Theorem 3.2 Suppose $f : [a, b] \rightarrow \mathbb{R}$ is $(m+1)$ times differentiable, and let P be the polynomial of degree $\leq m$ that passes through $(x_j, f(x_j))$ for $j = 0, 1, \dots, m$. Then for any $x \in [a, b]$ there is a $\xi \in [a, b]$ such that

$$f(x) - P(x) = \frac{1}{(m+1)!} M(x) f^{(m+1)}(\xi).$$

Proof. Suppose $x \neq x_j$ for $j = 0, 1, \dots, m$; otherwise there is nothing to prove. For this fixed x , let $\kappa \in \mathbb{R}$ be such that $f(x) - P(x) = \kappa M(x)$, and consider

$$Q(y) = f(y) - P(y) - \kappa M(y) \quad (\text{for } y \in [a, b]).$$

We have $Q(y) = 0$ if $y = x$ or $y = x_j$, $j = 0, \dots, m$. Therefore, Q has at least $m+2$ zeroes in $[a, b]$. Between any two zeroes of Q there is a zero of Q' (Rolle's theorem), so we know that Q' has at least $m+1$ zeroes in $[a, b]$. Subsequently, Q'' has at least m zeroes in $[a, b]$, and so on. Finally, $Q^{(m+1)}$ has at least one zero in $[a, b]$, which we call ξ . But then $0 = Q^{(m+1)}(\xi) = f^{(m+1)}(\xi) - (m+1)! \kappa$, that is, $\kappa = \frac{1}{(m+1)!} f^{(m+1)}(\xi)$. \square

Corollary 3.3 Under the assumptions of the previous theorem, we have

$$|f(x) - P(x)| \leq \frac{1}{(m+1)!} \sup_{\xi \in [a,b]} |f^{(m+1)}(\xi)| \cdot |M(x)|. \quad \square$$

Illustration. Suppose $x_0 = \min_i x_i$ and $x_m = \max_i x_i$. We can use the interpolation formulas outside $[x_0, x_m]$, but this is not recommended. It will usually lead to large errors, due to the fact that the function values $|M(x)|$ quickly become very large outside the interval $[x_0, x_m]$.

An illustration with $m = 10$ is given in the left panel of Figure 3.1 for $f(x) = \cos(5x - 1)$ with equally spaced points $x_j = -1 + 2j/m \in [-1, 1]$. Then $\frac{1}{(m+1)!} |f^{(m+1)}(\xi)| \leq \frac{1}{(m+1)!} 5^{m+1}$, which is small for large m . Indeed with increasing m , the interpolation error becomes very small on $[-1, 1]$. However outside the interpolation interval we see that the error on the left becomes large right away.

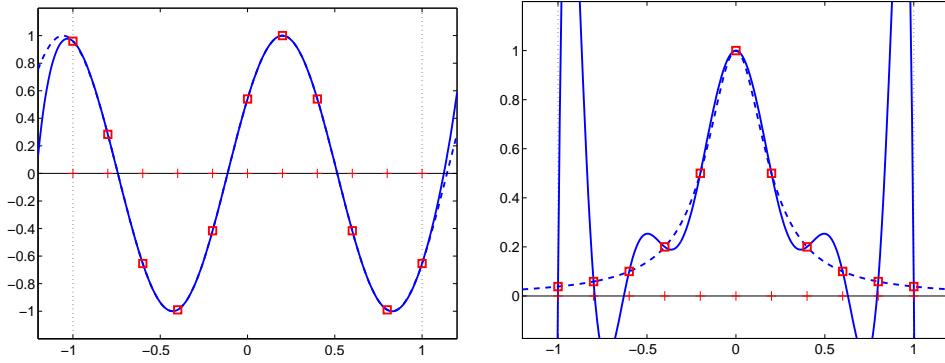


Figure 3.1: Interpolation with polynomials of degree $m = 10$, equidistant nodes $x_i = -1 + 0.2i$ in $[-1, 1]$ with $f_i = f(x_i)$, where $f(x) = \cos(5x - 1)$ (left panel) and $f(x) = 1/(1 + 25x^2)$ (right panel). The graph of f is given by the dashed line, the interpolating polynomial is the solid line.

A more serious problem arises when $\frac{1}{(m+1)!} |f^{(m+1)}|$ is not small. Then the error can exhibit large oscillations towards the endpoints of the interpolation interval. This is called the *Runge phenomenon*. An illustration is presented in the right panel of Figure 3.1 for $f(x) = (1 + 25x^2)^{-1}$ with the same grid spacing and $m = 10$ as before. In the middle of the interpolation interval the result is not too bad, and this would in fact become better with larger m but then also the oscillations near the endpoints become worse.

Chebyshev nodes. To some extent the bad result in Figure 3.1 is due to the equal spacing of the nodes x_j used for that computation. A natural question is how to choose these nodes in $[a, b]$ such that $\max_{[a,b]} |M(x)|$ is minimized. To answer this question, let us first take $[a, b] = [-1, 1]$; later it can be transformed back to arbitrary intervals.

Consider the *Chebyshev polynomials*, defined for $n = 0, 1, 2, \dots$ by

$$(3.7) \quad T_n(x) = \cos(n \arccos(x)) \quad (-1 \leq x \leq 1).$$

It is not obvious that these are polynomials. However it is easy to see (by setting $x = \cos \theta$ and using $\cos((n+1)\theta) + \cos((n-1)\theta) = 2 \cos(\theta) \cos(n\theta)$) that we have the recursion

$$(3.8) \quad T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \quad (n \geq 2).$$

This makes it clear that T_n is indeed a polynomial of degree n , and the factor for the leading power x^n is 2^{n-1} . Note also that formula (3.8) can be used for all $x \in \mathbb{R}$. Moreover, if $n > 0$, we have $|T_n(x)| = 1$ for $n+1$ distinct points in $[-1, 1]$:

$$(3.9a) \quad T_n(\eta_k) = (-1)^k \quad \text{for} \quad \eta_k = \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, \dots, n,$$

and all roots of T_n are in the interval $(-1, 1)$:

$$(3.9b) \quad T_n(\xi_k) = 0 \quad \text{for} \quad \xi_k = \cos\left(\left(k + \frac{1}{2}\right)\frac{\pi}{n}\right), \quad k = 0, 1, \dots, n-1.$$

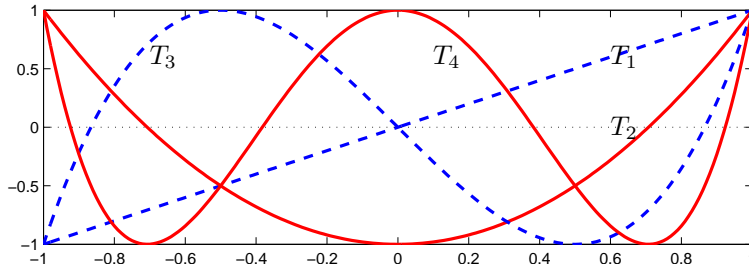


Figure 3.2: Chebyshev polynomials T_1, T_2, T_3, T_4 on $[-1, 1]$.

Lemma 3.4 We have $|T_n(x)| \leq 1$ for all $x \in [-1, 1]$. Moreover, if S_n is another polynomial of degree n with factor 2^{n-1} for the leading power x^n , then $\max_{x \in [-1, 1]} |S_n(x)| \geq 1$.

Proof. The property $\max_{[-1, 1]} |T_n(x)| \leq 1$ follows directly from (3.7). Now suppose S_n has the same leading coefficient and $\max_{[-1, 1]} |S_n(x)| < 1$. Then $R(x) = S_n(x) - T_n(x)$ is a polynomial of degree $n-1$, because the highest powers cancel. We know that $T_n(x)$ oscillates between the values -1 and 1 . It follows from (3.9a) that R has a zero on each interval (η_k, η_{k-1}) , $k = 1, 2, \dots, n$. Therefore R has at least n zeroes on $[-1, 1]$. But this implies $R \equiv 0$, which contradicts the assumption $\max_{[-1, 1]} |S_n(x)| < 1$. \square

Remark 3.5 By counting the zeros a bit more carefully, it is also easy to show that $\max_{x \in [-1, 1]} |S_n(x)| > 1$ if $S_n \neq T_n$. \diamond

Now, returning to our problem, we see that if we want to make $\max_{[-1,1]} |M(x)|$ minimal, then the choice is $M(x) = 2^{-m}T_{m+1}(x)$. This means that the nodes x_j should coincide with the zeroes of T_{m+1} , that is, $x_j = \cos((j + \frac{1}{2})\pi/(m + 1))$.

For an arbitrary interval $[a, b]$ we can use the translation $x \mapsto \frac{1}{2}(a+b) + \frac{1}{2}(b-a)x$ to map $[-1, 1]$ onto $[a, b]$. It follows that $\max_{[a,b]} |M(x)|$ is minimal for the nodes

$$(3.10) \quad x_j = \frac{1}{2}(a + b) + \frac{1}{2}(b - a) \cos\left((j + \frac{1}{2})\frac{\pi}{m+1}\right) \quad (j = 0, 1, \dots, m).$$

Illustration. The results for these Chebyshev nodes on $[-1, 1]$ with $m = 10$ are shown in Figure 3.3, with the lay-out as before for the equally spaced nodes. The very large oscillations at the endpoints for $f(x) = 1/(1 + 25x^2)$ have become less severe, but it is clear that the interpolation is there still inaccurate. However, with these Chebyshev nodes the interpolation will improve quickly if we increase m , in contrast to the case with equidistant nodes.

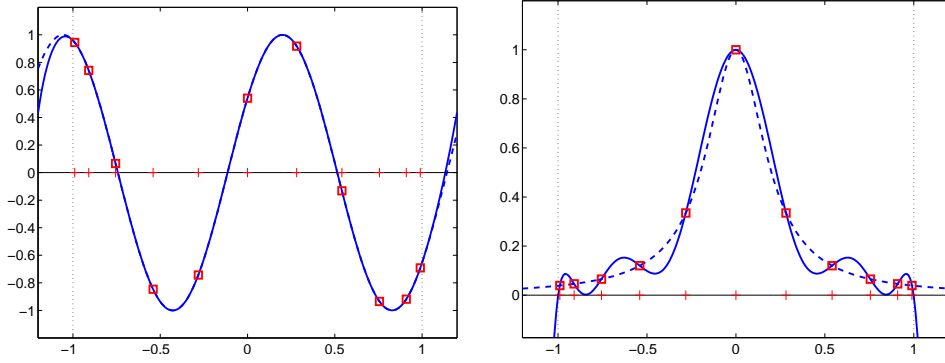


Figure 3.3: Interpolation with polynomials of degree $m = 10$, Chebyshev nodes in $[-1, 1]$ with $f_i = f(x_i)$, where $f(x) = \cos(5x - 1)$ (left panel) and $f(x) = 1/(1 + 25x^2)$ (right panel). This is to be compared with Figure 3.1.

For the smooth function $f(x) = \cos(5x - 1)$ the errors are not visible inside the interpolation interval, but they are actually much smaller for the Chebyshev points (maximal error $7.09 \cdot 10^{-4}$ on $[-1, 1]$) than for the uniformly distributed points (maximal error $6.74 \cdot 10^{-3}$).

Although the result for $f(x) = 1/(1 + 25x^2)$ is now much better than for the uniformly distributed points, there are obviously still relatively large errors with $m = 10$. Increasing m leads rapidly to small errors, as shown in Table 3.1.

Table 3.1: Maximal error $\max_{x \in [-1,1]} |f(x) - P(x)|$ for increasing m with the Chebyshev nodes and $f(x) = 1/(1 + 25x^2)$.

m	10	20	30	40	50	60
Max. err.	$1.09 \cdot 10^{-1}$	$1.53 \cdot 10^{-2}$	$2.04 \cdot 10^{-3}$	$2.86 \cdot 10^{-4}$	$3.96 \cdot 10^{-5}$	$5.18 \cdot 10^{-6}$

Remark 3.6 It can be shown (but this is quite difficult) that the interpolation polynomials with Chebyshev nodes will converge for increasing m to $f(x)$, uniformly on $[a, b]$, provided that f is *continuously differentiable*.

A famous theorem of Weierstrass states that any *continuous* function f can be approximated by polynomials with arbitrary high accuracy, uniformly on the interval $[a, b]$. A constructive proof of this result can be given using the Bernstein polynomials $B_n(x) = \sum_{j=0}^n \binom{n}{j} x^j (1-x)^{n-j} f(\frac{j}{n})$ with interval $[a, b] = [0, 1]$.

These polynomials $B_n(x)$ will converge uniformly to $f(x)$ for increasing n , but the convergence is very, very slow. From a practical numerical point of view these Bernstein polynomials are therefore not suited to approximate functions. \diamond

3.3 Piecewise Polynomials and Splines

Assume from now on $a = x_0 < x_1 < \dots < x_m = b$, and let $h_i = x_i - x_{i-1}$ for $i = 1, 2, \dots, m$. To avoid the oscillatory behaviour that can be encountered with high-order interpolation, we can consider interpolation on each sub-interval $[x_{i-1}, x_i]$ with a low order polynomial P_i .

The most simple interpolation of this kind is piecewise linear interpolation,

$$(3.11) \quad P_i(x) = \frac{x - x_{i-1}}{h_i} f_i - \frac{x - x_i}{h_i} f_{i-1} \quad \text{for } x_{i-1} \leq x \leq x_i.$$

From Corollary 3.3 it follows that if $f_i = f(x_i)$ with a function f that is twice differentiable, then

$$(3.12) \quad |f(x) - P_i(x)| \leq \frac{1}{8} h_i^2 \sup_{x_{i-1} \leq \xi \leq x_i} |f''(\xi)| \quad \text{for all } x \in [x_{i-1}, x_i].$$

To increase the accuracy we can take P_i to be a polynomial of higher degree, for example, of degree 3 passing through (x_j, f_j) , $j = i-2, i-1, i, i+1$. However, at the nodes we will then have merely continuity of our interpolant. To get a good looking output, for example for plotting, more smoothness is often required.

A very popular choice is to take the interpolant as a *cubic spline* function S , where S is a polynomial of degree ≤ 3 on each sub-interval $[x_{i-1}, x_i]$ and S is twice continuously differentiable on $[a, b]$.

Consider a piecewise cubic polynomial given on $[x_{i-1}, x_i]$ by $S(x) = S_i(x)$, with

$$(3.13) \quad S_i(x) = f_i + u_i(x - x_i) + v_i(x - x_i)^2 + w_i(x - x_i)^3,$$

for $i = 1, 2, \dots, m$. Here it is already ensured that $S_i(x_i) = f_i$. The remaining conditions are

$$(3.14) \quad \begin{cases} S_i(x_{i-1}) = f_{i-1} & (i = 1, 2, \dots, m), \\ S'_{i-1}(x_{i-1}) = S'_i(x_{i-1}), \quad S''_{i-1}(x_{i-1}) = S''_i(x_{i-1}) & (i = 2, 3, \dots, m). \end{cases}$$

This gives $3m-2$ conditions for the free parameters u_i, v_i, w_i . Therefore, two extra conditions are needed. Common choices are

$$(3.15a) \quad S''(a) = S''(b) = 0 \quad \text{the so-called } \textit{natural} \text{ or } \textit{free} \text{ splines},$$

$$(3.15b) \quad S'(a) = f'_0, \quad S'(b) = f'_m \quad \text{the so-called } \textit{clamped} \text{ splines},$$

where for the last choice we need of course the additional data f'_0, f'_m at the endpoints. It will be shown below that this leads to a tri-diagonal system, which is easily solved.

Illustration. One of the motivations to look at splines was the oscillating behaviour of the high-order interpolants for the function $f(x) = 1/(1 + 25x^2)$ on $[-1, 1]$. The right panel of Figure 3.4 shows the cubic natural spline interpolating this function at the equally spaced points $x_i = -1 + 0.2i$, $i = 0, 1, \dots, m = 10$. The results are now satisfactory, in particular when compared the the previous results with the interpolants of degree 10. The spline produces indeed a ‘good looking’ interpolant.

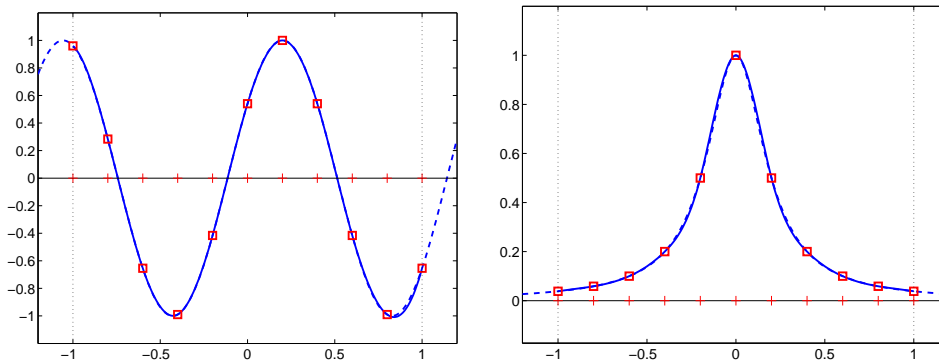


Figure 3.4: Spline interpolation with nodes $x_i = -1 + 0.2i$ in $[-1, 1]$ for $f_i = f(x_i)$, with $f(x) = \cos(5x - 1)$ (left panel) and $f(x) = 1/(1 + 25x^2)$ (right panel). As before, the graph of f is given by the dashed line, the interpolating polynomial is the solid line.

For the smooth function $f(x) = \cos(5x - 1)$ the maximal error on $[-1, 1]$ of the interpolating natural spline is $2.72 \cdot 10^{-2}$; for the clamped spline this error is somewhat smaller, $3.08 \cdot 10^{-3}$. Here the high-order polynomial interpolation with Chebyshev points is more accurate. The same happens with the function $f(x) = 1/(1 + 25x^2)$ if the number of points m gets larger. The maximal errors with increasing m are given in Table 3.2, which is to be compared with Table 3.1.

Table 3.2: Maximal error $\max_{x \in [-1, 1]} |f(x) - P(x)|$ for increasing m with the free splines and $f(x) = 1/(1 + 25x^2)$.

m	10	20	30	40	50	60
Max. err.	$2.20 \cdot 10^{-2}$	$3.20 \cdot 10^{-3}$	$8.24 \cdot 10^{-4}$	$2.77 \cdot 10^{-4}$	$1.11 \cdot 10^{-4}$	$5.25 \cdot 10^{-5}$

Construction. As already mentioned above, construction of a spline merely requires solving a tri-diagonal linear system. This is shown here for a natural spline; for a clamped spline it is similar. Denote $h_i = x_i - x_{i-1}$. Elaboration of

(3.13), (3.14) gives

$$\begin{aligned} f_i - h_i u_i + h_i^2 v_i - h_i^3 w_i &= f_{i-1} & (i = 1, \dots, m), \\ u_i - 2h_i v_i + 3h_i^2 w_i &= u_{i-1} & (i = 2, \dots, m), \\ v_i - 3h_i w_i &= v_{i-1} & (i = 2, \dots, m). \end{aligned}$$

Moreover we have for the natural spline $v_1 - 3h_1 w_1 = 0$ and $v_m = 0$. It is convenient to introduce $v_0 = 0$. Then

$$\begin{aligned} h_i w_i &= \frac{1}{3}(v_i - v_{i-1}) & (i = 1, \dots, m), \\ h_i u_i &= f_i - f_{i-1} + \frac{1}{3}h_i^2(2v_i + v_{i-1}) & (i = 1, \dots, m), \end{aligned}$$

and then for the v_1, v_2, \dots, v_{m-1} it is finally found that

$$(3.16) \quad \frac{1}{3}h_i v_{i-1} + \frac{2}{3}(h_i + h_{i+1})v_i + \frac{1}{3}h_{i+1}v_{i+1} = b_i \quad (i = 1, \dots, m-1),$$

with right-hand side terms $b_i = (h_i^{-1}(f_{i-1} - f_i) - h_{i+1}^{-1}(f_i - f_{i+1}))$. This is a linear system for the v_i with a symmetric positive-definite tri-diagonal matrix, that can be solved very quickly.

Properties.* Splines have interesting mathematical properties. Here we only mention two such properties, without proof. For a given function f , let S be a cubic spline that interpolates $(x_i, f(x_i))$ ($0 \leq i \leq m$).

First, if f is four times differentiable, and S is the clamped spline with $S'(a) = f'(a)$ and $S'(b) = f'(b)$, then the interpolation error satisfies

$$(3.17) \quad \max_{x \in [a, b]} |f(x) - S(x)| \leq \frac{5}{384}h^4 \max_{\xi \in [a, b]} |f^{(4)}(\xi)|$$

where $h = \max_i h_i$. A similar $\mathcal{O}(h^4)$ error bound holds for the natural spline in the interior of the domain.

Secondly, splines have an interesting minimization property with respect to the integral $\int_a^b |f''(x)|^2 dx$, which can be viewed as a measure for the total bending of f on $[a, b]$. If f is two times continuously differentiable, and the interpolating spline is such that $S''(a)(f'(a) - S'(a)) = S''(b)(f'(b) - S'(b))$ – which holds for both the natural and the clamped spline – then

$$(3.18) \quad \int_a^b |S''(x)|^2 dx \leq \int_a^b |f''(x)|^2 dx.$$

3.4 Exercises

Exercise 3.1. Show that $\sum_{i=0}^m L_i(x) \equiv 1$ for the Lagrange polynomials (3.2). Hint: what happens if all $f_i = 1$ in (3.3)?

Next, show that formula (3.6) can be written as

$$(3.19) \quad P(x) = \frac{\sum_{i=0}^m Q_i(x) f_i}{\sum_{i=0}^m Q_i(x)}, \quad Q_i(x) = \frac{w_i}{x - x_i} \quad (i = 0, 1, \dots, m).$$

Exercise 3.2. Suppose $P_{0,\dots,m-1}(x)$ interpolates $(x_0, f_0), \dots, (x_{m-1}, f_{m-1})$, and $P_{1,\dots,m}(x)$ interpolates $(x_1, f_1), \dots, (x_m, f_m)$. Show that

$$P_{0,1,\dots,m}(x) = \frac{1}{x_m - x_0} \left((x_m - x) P_{0,\dots,m-1}(x) + (x - x_0) P_{1,\dots,m}(x) \right)$$

then passes through all data points (x_j, f_j) , $j = 0, 1, \dots, m$.

Exercise 3.3. Suppose f is twice continuously differentiable, and let P be the linear interpolant with nodes x_0 and $x_1 = x_0 + h$. Show that

$$|f(x) - P(x)| \leq \frac{1}{8} h^2 \max_{\xi \in [x_0, x_1]} |f''(\xi)| \quad (\text{for all } x \in [x_0, x_1]),$$

and

$$\left| \int_{x_0}^{x_1} f(x) dx - \int_{x_0}^{x_1} P(x) dx \right| \leq \frac{1}{12} h^3 \max_{\xi \in [x_0, x_1]} |f''(\xi)|.$$

Exercise 3.4. We can modify the proof of Theorem 3.2 to obtain an expression for the error of the first derivative at the nodes. Show that for each node x_j there is a $\xi_j \in [a, b]$ such that

$$f'(x_j) - P'(x_j) = \frac{1}{(m+1)!} M'(x_j) f^{(m+1)}(\xi_j).$$

Hint: consider $Q(y) = f(y) - P(y) - \kappa M(y)$ with $\kappa = \frac{1}{M'(x_j)} (f'(x_j) - P'(x_j))$.

Exercise 3.5 (Product formulas). Let x_i and y_j be given for $0 \leq i, j \leq m$ together with values f_{ij} in \mathbb{R} . We want a polynomial $P(x, y)$ of degree $\leq m$ in x and y that interpolates these data:

$$P(x_i, y_j) = f_{ij} \quad (0 \leq i, j \leq m).$$

(a) Derive the two-dimensional linear interpolation formula, $m = 1$, from the one-dimensional formulas.

(b) Discuss generalization to $m > 1$.

*Exercise 3.6.** Consider $[a, b] = [-1, 1]$ with equidistant nodes $x_j = a + jh$, where $h = (b-a)/m = 2/m$, and let $M(x) = (x - x_0)(x - x_1) \cdots (x - x_m)$. Show that

$$\frac{1}{4} (m-1)! h^{m+1} \leq \max_{x \in [a, b]} |M(x)| \leq \frac{1}{4} m! h^{m+1}.$$

Hint: to derive the upper bound, first consider $x \in (x_0, x_1)$, and to derive the lower bound consider $x = \frac{1}{2}(x_0 + x_1)$.

Note: the factorial term $m!$ can be well approximated by Stirling's formula $m! \sim \sqrt{2\pi m} (m/e)^m$. Using this, the size of the upper bound becomes more clear for large m :

$$\max_{x \in [a, b]} |M(x)| \lesssim \frac{1}{2} \sqrt{\frac{2\pi}{m}} \left(\frac{2}{e}\right)^m.$$

*Exercise 3.7.** Consider the interval $[a, b] = [-1, 1]$. Show that for the Chebyshev nodes (3.10) we have

$$\max_{x \in [a, b]} |M(x)| \leq 2^{-m}.$$

Exercise 3.8. Suppose the data f_j are obtained from measurements which have a relative error ε or less. This means that, instead of the exact values f_j , we have to work with perturbed values $\tilde{f}_j = (1 + \varepsilon_j) f_j$ with $|\varepsilon_j| \leq \varepsilon$. This leads to a perturbed interpolant $\tilde{P}(x)$, with $\tilde{P}(x_j) = \tilde{f}_j$ for $j = 1, 2, \dots, m$. Show that

$$|\tilde{P}(x) - P(x)| \leq \varepsilon \max_{0 \leq i \leq m} |f_i| \cdot \Lambda_m, \quad \Lambda_m = \max_{x \in [a, b]} \sum_{i=0}^m |L_i(x)|.$$

Note: the value Λ_m is called the *Lebesgue constant* for the nodes x_0, x_1, \dots, x_m . It has been studied extensively. It is known that for equidistant nodes, Λ_m grows exponentially with m . For the Chebyshev nodes, Λ_m behaves much nicer: it grows only as $\log(m)$. So also in this respect there is a clear advantage for the Chebyshev nodes over the equidistant nodes.

Exercise 3.9 (programming). Write a program for the interpolation problem (3.1) with a polynomial of degree $\leq m$.

- (a) Use equidistant nodes and verify Figure 3.1. What happens if m is increased?
- (b) Use Chebyshev nodes and verify Figure 3.3. What happens if m is increased?
- (c) Do the same for the natural cubic splines and Figure 3.4.

Exercise 3.10 (programming). Splines can be used to draw smooth curves in \mathbb{R}^2 . Suppose points $(x_j, y_j) \in \mathbb{R}^2$ are given for $j = 0, 1, \dots, m$. Consider $t_j = j/m$. We can compute the natural splines $X(t)$ interpolating (t_j, x_j) and $Y(t)$ interpolating (t_j, y_j) . Plot $Y(t)$ versus $X(t)$ for $\{x_j\} = \{-1, 0, 1, 0, -0.5, 0.5\}$ and $\{y_j\} = \{0, 1, 0.5, 0, 1.5, 1.5\}$.

4 Trigonometric Interpolation with DFT and FFT

In this section² we will study interpolation and approximation of periodic functions $f : \mathbb{R} \rightarrow \mathbb{R}$. It will be assumed for convenience, but without loss of generality, that the period is 2π , that is, $f(x + 2\pi) = f(x)$ for all $x \in \mathbb{R}$. The approximations will consist of trigonometric polynomials

$$(4.1a) \quad \varphi(x) = a_0 + 2 \sum_{k=1}^{m-1} \left(a_k \cos(kx) + b_k \sin(kx) \right) + a_m \cos(mx),$$

with coefficients

$$(4.1b) \quad a_k = \frac{1}{2m} \sum_{j=0}^{2m-1} f(x_j) \cos(kx_j), \quad b_k = \frac{1}{2m} \sum_{j=0}^{2m-1} f(x_j) \sin(kx_j),$$

where $x_j = \pi j/m$. We will see that for smooth functions, this interpolation formula gives a very accurate approximation to f already for rather modest values of m .

Instead of using expressions with $\cos(kx)$ and $\sin(kx)$ terms as in (4.1), it is easier to work with complex exponentials e^{ikx} , $i = \sqrt{-1}$, and it is then also natural to let $f : \mathbb{R} \rightarrow \mathbb{C}$.

To derive (4.1), we will first consider discrete Fourier transforms (DFT), which is an important subject on its own with many applications; for example in signal analysis and data compression. There are very efficient implementations of these discrete Fourier transforms, known as fast Fourier transforms (FFT), which will be briefly discussed later.

4.1 Fourier Series and Fourier Transforms

Let $f : \mathbb{R} \rightarrow \mathbb{C}$ be a 2π -periodic function. The *Fourier transform* of f is the sequence $\{\dots, c_{-1}, c_0, c_1, c_2, \dots\}$ given by

$$(4.2) \quad c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx \quad (k \in \mathbb{Z}),$$

where \mathbb{Z} stands for the set of all integers. Conversely, under mild smoothness assumptions, f can be represented by the *Fourier series*

$$(4.3) \quad f(x) = \sum_{k \in \mathbb{Z}} c_k e^{ikx} \quad (x \in \mathbb{R}).$$

For the main estimates in this section it will be assumed that the function f is piecewise continuously differentiable. This is sufficient to justify (4.3).

The problem that will be addressed in this section is the following. Suppose that f is only known on the set of discrete, equidistant points

$$(4.4) \quad x_j = \frac{2\pi j}{n}, \quad j = 0, 1, \dots, n.$$

²The presentation in this section is mainly based on the lecture notes of E. Hairer. The other references listed in the Preface cover this material only partially.

Then we can consider

$$(4.5) \quad v_k = \frac{1}{n} \sum_{j=0}^{n-1} f(x_j) e^{-ikx_j} \quad (k \in \mathbb{Z})$$

to approximate the integral in (4.2). We will see that with these coefficients v_k we can construct an interpolant for f on the points x_0, \dots, x_n . This interpolant will be a complex trigonometric polynomial, consisting of linear combinations of powers of e^{ix} .

Discrete Fourier Transforms. We denote by Π_n the set of complex sequences $u = \{\dots, u_{-1}, u_0, u_1, u_2, \dots\}$ that are n -periodic, that is, $u_k \in \mathbb{C}$ and $u_k = u_{k+n}$ for all $k \in \mathbb{Z}$. The *discrete Fourier transform* (DFT) is the linear map \mathcal{F}_n defined for any $u \in \Pi_n$ by:

$$(4.6) \quad v = \mathcal{F}_n u \quad \text{if} \quad v_k = \frac{1}{n} \sum_{j=0}^{n-1} u_j e^{-ikx_j} \quad (k \in \mathbb{Z}).$$

Lemma 4.1 The discrete Fourier transform \mathcal{F}_n maps Π_n into itself. This linear mapping is invertible, and its inverse is given by:

$$(4.7) \quad u = \mathcal{F}_n^{-1} v \quad \text{if} \quad u_k = \sum_{j=0}^{n-1} v_j e^{ikx_j} \quad (k \in \mathbb{Z}).$$

Proof. Introducing $\omega = e^{2\pi i/n}$, we can write $e^{ikx_j} = \omega^{kj}$. Since $\omega^n = e^{2\pi i} = 1$, we have $\omega^{nj} = 1$ for any $j \in \mathbb{Z}$. Hence

$$v_{k+n} = \frac{1}{n} \sum_{j=0}^{n-1} u_j \omega^{-(k+n)j} = \frac{1}{n} \sum_{j=0}^{n-1} u_j \omega^{-kj} = v_k,$$

which shows that $v \in \Pi_n$.

Now let \mathcal{F}_n^{-1} be defined as in (4.7). To show that this is the inverse of \mathcal{F}_n , consider

$$\begin{aligned} (\mathcal{F}_n^{-1} \mathcal{F}_n u)_l &= \sum_{k=0}^{n-1} (\mathcal{F}_n u)_k \omega^{kl} = \frac{1}{n} \sum_{k=0}^{n-1} \sum_{j=0}^{n-1} u_j \omega^{-kj} \omega^{kl} \\ &= \frac{1}{n} \sum_{j=0}^{n-1} u_j \left(\sum_{k=0}^{n-1} \omega^{k(l-j)} \right) = u_l. \end{aligned}$$

Here the last equality follows from

$$\sum_{k=0}^{n-1} \omega^{kd} = \sum_{k=0}^{n-1} (\omega^d)^k = \begin{cases} n & \text{if } d \equiv 0 \pmod{n}, \\ \frac{\omega^{nd}-1}{\omega^d-1} = 0 & \text{otherwise,} \end{cases}$$

due to the fact that $\omega^d = 1$ if $d \equiv 0 \pmod{n}$. □

Since any $u \in \Pi_n$ is determined by n components, we can identify Π_n with \mathbb{C}^n . Among the other properties of the discrete Fourier transform, we mention the discrete Parseval identity: $\sum_{j=0}^{n-1} |v_j|^2 = \frac{1}{n} \sum_{j=0}^{n-1} |u_j|^2$.

4.2 Approximation Properties

For a 2π -periodic function $f : \mathbb{R} \rightarrow \mathbb{C}$, we consider the Fourier coefficients c_k from (4.2), together with the coefficients v_k from the discrete Fourier transform (4.5). We want to know how well v_k approximates c_k .

Lemma 4.2 Suppose the series $\sum_{k \in \mathbb{Z}} c_k$ is absolutely convergent. Then

$$(4.8) \quad v_k - c_k = \sum_{j \neq 0} c_{k+jn}.$$

Proof. As before, let $\omega = e^{2\pi i/n}$. Inserting (4.3) into (4.5) gives

$$v_k = \frac{1}{n} \sum_{j=0}^{n-1} \left(\sum_{l \in \mathbb{Z}} c_l \omega^{lj} \right) \omega^{-kj} = \sum_{l \in \mathbb{Z}} c_l \left(\frac{1}{n} \sum_{j=0}^{n-1} \omega^{(l-k)j} \right).$$

Here changing the order of the summation is justified because of the absolute convergence of the Fourier series. Since $\frac{1}{n} \sum_{j=0}^{n-1} \omega^{(l-k)j}$ equals 1 if $l = k \pmod{n}$ and 0 otherwise, the result (4.8) follows. \square

For a 2π -periodic function $f : \mathbb{R} \rightarrow \mathbb{C}$ we will write $f \in E^0$ if there is a finite partitioning $0 = \theta_0 < \theta_1 < \theta_2 < \dots < \theta_s = 2\pi$ of the interval $[0, 2\pi]$ such that the derivative f' is continuous and bounded on each sub-interval (θ_{j-1}, θ_j) , $j = 1, 2, \dots, s$. So, discontinuities of f are allowed at the points θ_j , but the left and right limits $\lim_{x \uparrow \theta_j} f(x)$ and $\lim_{x \downarrow \theta_{j-1}} f(x)$ exist. For $p \geq 1$ we will write $f \in E^p$ if $f^{(p)} \in E^0$ and $f^{(p-1)}$ is continuous on \mathbb{R} . We may then take $\int_0^{2\pi} f^{(p)}(x) e^{-ikx} dx = \sum_{j=1}^s \int_{\theta_{j-1}}^{\theta_j} f^{(p)}(x) e^{-ikx} dx$.

Lemma 4.3 Suppose $f \in E^p$. Then there is a $\gamma > 0$ such that the Fourier coefficients c_k satisfy

$$(4.9) \quad |c_k| \leq \gamma |k|^{-(p+1)} \quad (\text{for all } k \in \mathbb{Z}).$$

Proof. Using periodicity of $f^{(j)}(x) e^{-ikx}$ for $j < p$, it follows by repeated partial integration that

$$\begin{aligned} c_k &= \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi ik} \int_0^{2\pi} f'(x) e^{-ikx} dx = \dots \\ &= \frac{1}{2\pi (ik)^p} \int_0^{2\pi} f^{(p)}(x) e^{-ikx} dx = \frac{-1}{2\pi (ik)^{p+1}} \int_0^{2\pi} f^{(p)}(x) d(e^{-ikx}). \end{aligned}$$

On each sub-interval (θ_{j-1}, θ_j) , $1 \leq j \leq s+1$, we have

$$\int_{\theta_{j-1}}^{\theta_j} f^{(p)}(x) d(e^{-ikx}) = \lim_{\varepsilon \downarrow 0} \left(f^{(p)}(x) e^{-ikx} \Big|_{\theta_{j-1}+\varepsilon}^{\theta_j-\varepsilon} \right) + \int_{\theta_{j-1}}^{\theta_j} f^{(p+1)}(x) e^{-ikx} dx,$$

and this is well-defined. The bound (4.9) thus follows with a constant $\gamma > 0$ given by $\gamma = \frac{1}{2\pi} \sum_{j=1}^s \left| \int_{\theta_{j-1}}^{\theta_j} f^{(p)}(x) d(e^{-ikx}) \right|$. \square

Corollary 4.4 Suppose $f \in E^p$ with $p \geq 1$. Then there is an $\alpha > 0$ such that

$$(4.10) \quad |v_k - c_k| \leq \alpha n^{-(p+1)} \quad \text{for } |k| \leq \frac{1}{2}n.$$

Proof. We have $|v_k - c_k| \leq \sum_{j \neq 0} |c_{k+jn}| \leq \gamma \sum_{j \neq 0} |k + jn|^{-(p+1)}$. Moreover, for $|k| \leq \frac{1}{2}n$ it holds that $|k + jn| \geq (|j| - \frac{1}{2})n$. The upper bounds (4.10) are therefore obtained with $\alpha = 2\gamma \sum_{j \geq 1} (j - \frac{1}{2})^{-(p+1)}$. \square

From (4.9) we see that the c_k rapidly tend to zero as $k \rightarrow \infty$, whereas the sequence $\{v_k\}$ is n -periodic. Therefore, for large $|k|$, say $|k| \geq n$, v_k will not be a good approximation to c_k . On the other hand, if $|k| \leq \frac{1}{2}n$ it is in general very good, as expressed by (4.10).

Remark 4.5 In particular, for $k = 0$ we find from (4.10) that

$$(4.11) \quad \left| h \sum_{j=0}^{n-1} f(x_j) - \int_0^{2\pi} f(x) dx \right| \leq \frac{1}{(2\pi)^p} \alpha h^{p+1},$$

where $h = 2\pi/n$. When discussing numerical quadrature in the next section we will see that this result is actually quite amazing. \diamond

4.3 Trigonometric Interpolation

To construct an interpolant for f , using the discrete values $f(x_j)$ with $x_j = 2\pi j/n$ for $j = 0, 1, \dots, n-1$, we will use the coefficients v_k from the discrete Fourier transform (4.5). It will be convenient to take n even and let $m = \frac{1}{2}n$. Then the trigonometric polynomial

$$(4.12) \quad \varphi(x) = \frac{1}{2} v_{-m} e^{-imx} + \sum_{|k| < m} v_k e^{ikx} + \frac{1}{2} v_m e^{imx}$$

satisfies $\varphi(x_j) = f(x_j)$ for $j = 0, 1, \dots, n-1$. By considering $u \in \Pi_n$ with $u_j = f(x_j)$, this interpolation property follows from the observation that $\{v_k e^{ikx_j}\}$ is an n -periodic sequence, and therefore

$$\varphi(x_j) = \sum_{k=0}^{n-1} v_k e^{ikx_j} = (\mathcal{F}_n^{-1} v)_j = (\mathcal{F}_n^{-1} \mathcal{F}_n u)_j = u_j.$$

Our previous estimates now give the following result on the interpolation errors for trigonometric polynomials:

Theorem 4.6 Suppose $f : \mathbb{R} \rightarrow \mathbb{C}$ is 2π -periodic and its Fourier series is absolutely convergent. Then the trigonometric polynomial (4.12) satisfies

$$|f(x) - \varphi(x)| \leq |c_{-m}| + |c_m| + 2 \sum_{|k| > m} |c_k| \quad (\text{for all } x \in \mathbb{R}).$$

Proof. Subtraction of (4.12) from (4.3) gives

$$\begin{aligned} f(x) - \varphi(x) &= \left(\frac{1}{2}c_{-m}e^{-imx} + \sum_{|k|>m} c_k e^{ikx} + \frac{1}{2}c_m e^{imx} \right) \\ &+ \left(\frac{1}{2}(c_{-m} - v_{-m})e^{-imx} + \sum_{|k|<m} (c_k - v_k) e^{ikx} + \frac{1}{2}(c_m - v_m) e^{imx} \right). \end{aligned}$$

The proof now follows from (4.8) and the triangle inequality. \square

Corollary 4.7 Assume $f \in E^p$ with $p \geq 1$. Then the error with trigonometric interpolation satisfies $\max_{x \in \mathbb{R}} |f(x) - \varphi(x)| \leq C m^{-p}$ with $C > 0$ independent of m .

Proof. Since $p \geq 1$, the estimate (4.9) shows that the Fourier series of f will converge absolutely, and we obtain

$$|f(x) - p(x)| \leq 2\gamma m^{-(p+1)} + 4\gamma \sum_{k>m} k^{-(p+1)}$$

Since $\sum_{k>m} k^{-(p+1)} \leq \int_m^\infty t^{-(p+1)} dt = \frac{1}{p} m^{-p}$, the error bound of the corollary follows with $C = 2\gamma + 4\gamma/p$. \square

For many functions an even better bound exists: it is known that if f can be extended to a function of the complex variable $z = x + iy$ which is analytic in the strip $-\gamma \leq y \leq \gamma$, then the c_k can be bounded in modulus as $\beta e^{-\gamma|k|}$, which leads to the exponential bound $C e^{-\gamma m}$ for the interpolation errors.

For real 2π -periodic functions $f : \mathbb{R} \rightarrow \mathbb{R}$ we can rewrite the the trigonometric interpolant (4.12) in terms of cosine and sine functions. Since v_{-k} and v_k are then complex conjugate according to (4.5), and $v_{-m} = v_{-m+n} = v_m$, we obtain by setting $v_k = a_k + ib_k$ the formulas (4.1) with $m = \frac{1}{2}n$. This is the usual form of trigonometric interpolation formulas for real functions found in the literature. Similar formulas can also be derived for n odd.

Illustration. Trigonometric interpolation of smooth functions is very accurate. To illustrate the behaviour for a non-smooth example, let us consider $f(x) = 1 + x/\pi - x^2/\pi^2$ (for $-\pi < x \leq \pi$). This function is of course arbitrarily often differentiable on $(-\pi, \pi]$, but when it is extended to the whole real line by periodic continuation, $f(x + 2\pi) = f(x)$, discontinuities are created at $x = \pm\pi, \pm3\pi, \dots$, resulting in $f \in E^0$.

The trigonometric interpolants with $m = 4$ and $m = 8$ are shown in the left panel of Figure 4.1. Oscillations arise at the points $x = \pm\pi$; this is known as the *Gibbs phenomenon*. These oscillations will persist for larger m but they do become more and more localized around the discontinuities.

If we take $f(x) = 1 - x^2/\pi^2$ (for $-\pi < x \leq \pi$), then the periodic continuation is continuous but not differentiable at $x = \pm\pi, \pm3\pi, \dots$, resulting in $f \in E^1$. Already with $m = 8$ a rather accurate approximation is found, as can be seen in

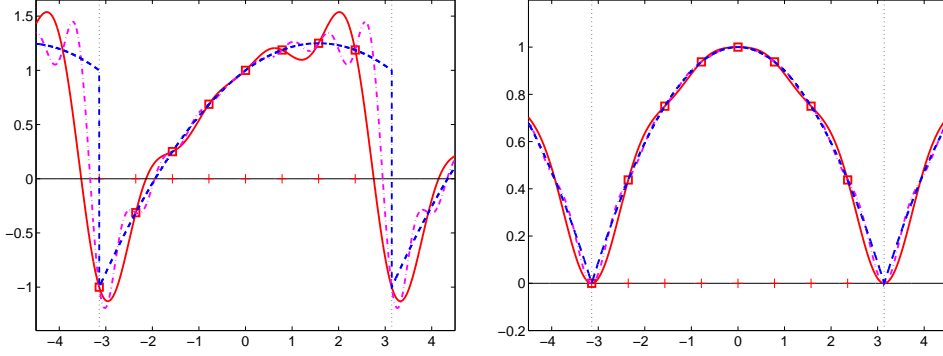


Figure 4.1: Interpolation with trigonometric polynomials with $m = 4$ and $m = 8$ for the functions $f(x) = 1 + x/\pi - x^2/\pi^2$ (left panel) and $f(x) = 1 - x^2/\pi^2$ (right panel). The graph of f is given by the dashed lines. The solid lines correspond with $m = 4$ and the dash-dotted lines with $m = 8$.

the right panel of Figure 4.1. The individual lines in this plot are not so easy to distinguish anymore, but that just illustrates the accuracy.

Needless to say, the above functions f are only intended as numerical illustration. In practice, we will not try to interpolate simple polynomials.

4.4 Fast Fourier Transforms

The discrete Fourier transform $v = \mathcal{F}_n u$ arises in many applications. Direct computation of all components v_0, v_1, \dots, v_{n-1} in (4.6) requires n^2 multiplications and additions. This can be done better. With the *fast Fourier transform* algorithm (FFT) it can be done in $n \log_2 n$ operations if n is a power of 2. The same applies to the computation of the inverse $\mathcal{F}_n^{-1} v$. Since discrete Fourier transforms arise so often, this is a very important practical result.

In the following, it will be more natural to work with $n\mathcal{F}_n$ instead of \mathcal{F}_n . We will use the notation $\omega_n = e^{2\pi i/n}$, and it is assumed that n is even, $n = 2m$.

The key to FFT is the realization that one transform $2m\mathcal{F}_{2m}$ of dimension $2m$ can be computed by doing two transforms $m\mathcal{F}_m$ of dimension m . More specifically: suppose $u = \{u_j\}$ and $v = \{v_j\}$ are in Π_m , and define

$$(4.13) \quad y = \{\dots, u_0, v_0, u_1, v_1, \dots, u_m, v_m, \dots\} \in \Pi_{2m}.$$

Then we have, for $k = 0, 1, \dots, m$,

$$(4.14) \quad \begin{cases} 2m(\mathcal{F}_{2m}y)_k &= m(\mathcal{F}_m u)_k + \omega_{2m}^{-k} m(\mathcal{F}_m v)_k \\ 2m(\mathcal{F}_{2m}y)_{m+k} &= m(\mathcal{F}_m u)_k - \omega_{2m}^{-k} m(\mathcal{F}_m v)_k. \end{cases}$$

To prove the first relation, note that

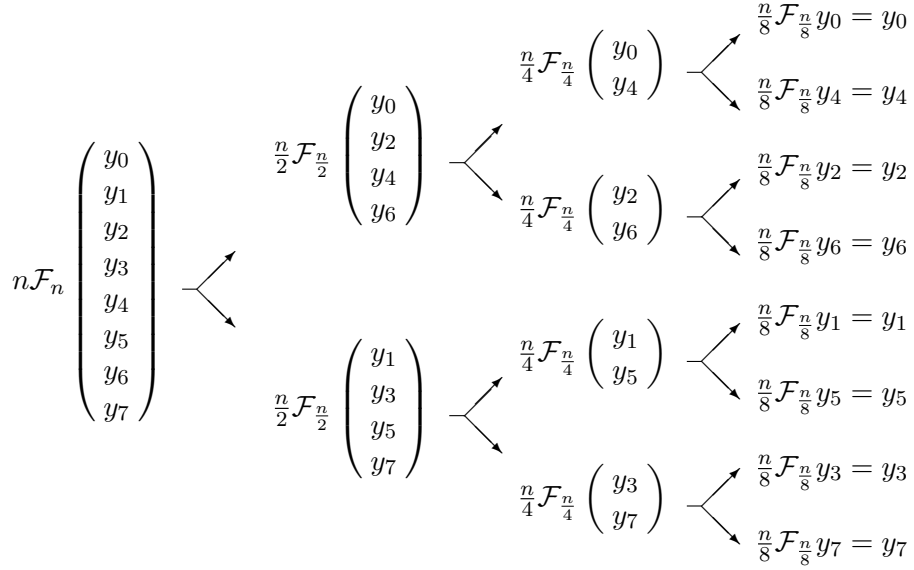
$$2m(\mathcal{F}_{2m}y)_k = \sum_{j=0}^{2m-1} y_j \omega_{2m}^{-jk} = \sum_{l=0}^{m-1} y_{2l} \omega_{2m}^{-2lk} + \sum_{l=0}^{m-1} y_{2l+1} \omega_{2m}^{-(2l+1)k}.$$

Using $y_{2l} = u_l$, $y_{2l+1} = v_l$ and $\omega_{2m}^{-2lk} = \omega_m^{-lk}$, $\omega_{2m}^{-(2l+1)k} = \omega_m^{-lk} \cdot \omega_{2m}^{-k}$, we thus obtain

$$2m (\mathcal{F}_{2m} y)_k = \sum_{l=0}^{m-1} u_l \omega_m^{-lk} + \omega_{2m}^{-k} \sum_{l=0}^{m-1} v_l \omega_m^{-lk} = m (\mathcal{F}_m u)_k + \omega_{2m}^{-k} m (\mathcal{F}_m v)_k$$

The second relation follows in the same way, using $\omega_{2m}^m = -1$.

The relations (4.14) can be applied recursively. If $n = 2^j$ this leads in j steps to the trivial problems of the form $\mathcal{F}_1 w = w$. For example, with $n = 8 = 2^3$ we get schematically:



The computation is now done by going in this scheme from right to left. We first have to perform a permutation so that the ordering on the right is obtained. This can be done easily with binary representation of the indices, using the so-called 'bit-reversal' trick:

index left		binary		reversed		index right
0	→	(0,0,0)	→	(0,0,0)	→	0
1	→	(0,0,1)	→	(1,0,0)	→	4
2	→	(0,1,0)	→	(0,1,0)	→	2
3	→	(0,1,1)	→	(1,1,0)	→	6
4	→	(1,0,0)	→	(0,0,1)	→	1
5	→	(1,0,1)	→	(1,0,1)	→	5
6	→	(1,1,0)	→	(0,1,1)	→	3
7	→	(1,1,1)	→	(1,1,1)	→	7

Using these ideas, the discrete Fourier transform \mathcal{F}_n can be performed with just $n \log_2 n$ complex multiplications and $\frac{1}{2} n \log_2 n$ complex additions if n is a power of 2. For the inverse \mathcal{F}_n^{-1} this is the same, of course.

Similar ideas can be used if n is not a power of 2, by decomposing n in its prime factors. The FFT procedures then do become somewhat less effective. If $n = 2^j \cdot p$ with prime $p > 2$, then after j steps of 'halving' we are left with j problems of size p . Standard FFT subroutines to compute \mathcal{F}_n and \mathcal{F}_n^{-1} are available on many computer systems for n arbitrary. These work most efficiently if n is a power of 2 ($\sim n \log_2 n$ operations), and least efficiently if n is a prime ($\sim n^2$ operations).

4.5 Exercises

Exercise 4.1 (fast convolution evaluations). For $u, v \in \Pi_n$, the *convolution* $u * v$ is defined by $(u * v)_k = \sum_{j=0}^{n-1} u_{k-j} v_j$ (for $k \in \mathbb{Z}$). Show that $u * v \in \Pi_n$ and

$$\mathcal{F}_n(u * v) = n \mathcal{F}_n u \cdot \mathcal{F}_n v$$

where the multiplication on the right is component-wise. How can we compute a convolution $u * v$ efficiently for large n ?

Exercise 4.2. Let $f(x) = 1 - x^2/\pi^2$ for $x \in (-\pi, \pi]$. We extend this function to \mathbb{R} by periodic continuation, $f(x + 2\pi) = f(x)$. Compute the Fourier coefficients c_k of this periodic function. How fast will the trigonometric interpolant converge towards f for increasing m (c.f. Figure 4.1)?

Exercise 4.3. In signal analysis, a 2π -periodic function f represents a *periodic signal*, and the values $f(x_k)$, $k = 0, 1, \dots, n-1$, are called *samples* of the signal. The signal is said to have maximal frequency N if its Fourier coefficients c_k are zero for $|k| > N$. The *sampling theorem* states that for an exact reconstruction of such a signal from its samples, at least $n = 2N$ samples are needed. Explain this result.

*Exercise 4.4.** The trigonometric polynomial (4.12) interpolates the data (x_j, f_j) with $f_j = f(x_j)$, $j = 0, 1, \dots, n-1$. Show that

$$\varphi(x) = \sum_{j=0}^{n-1} f_j \psi(x - x_j)$$

where $\psi(z) = \frac{1}{n} \sin(\frac{1}{2}nz) \cot(\frac{1}{2}z)$.

Exercise 4.5 (programming). The basic idea for FFT was already present in a work of Gauss, who wanted to predict the orbit of the asteroid Pallas from some observations. The data used by Gauss consisted of 12 points (θ_j, f_j) with angles θ_j in degrees:

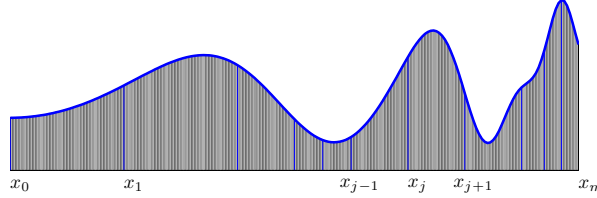
θ_j	0	30	60	90	120	150	180	210	240	270	300	330
f_j	408	89	-66	10	338	807	1238	1511	1583	1462	1183	804

Interpolate these data by a trigonometric polynomial (4.1) with $m = 6$ and $x_j = 2\pi\theta_j/360$. Plot the data points and the interpolant. [Gauss did these calculations by hand. If you don't like programming, you can try to do the same.]

5 Numerical Integration

5.1 Composite Integration Schemes

In this section we discuss *numerical quadrature*, that is, numerical approximation of integrals $I = \int_a^b f(x) dx$ with given function $f : [a, b] \rightarrow \mathbb{R}$. This function may exhibit different behaviour on different parts of the integration interval, so then it is natural to break down the integration into these parts.



Let us assume that we have some suitable partitioning

$$x_0 = a < x_1 < x_2 < \dots < x_m = b,$$

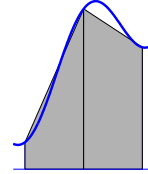
which divides the integration interval $[a, b]$ into m sub-intervals $[x_{k-1}, x_k]$. Then, of course, $I = \sum_{k=1}^m I_k$ with

$$(5.1) \quad I_k = \int_{x_{k-1}}^{x_k} f(x) dx.$$

It remains to find suitable approximations \tilde{I}_k for the sub-integrals I_k . Let us first consider some simple examples, where we denote $h_k = x_k - x_{k-1}$.

On each sub-interval we can approximate $f(x)$ by a linear interpolation polynomial

$$f(x) \approx \frac{x_k - x}{h_k} f(x_{k-1}) + \frac{x - x_{k-1}}{h_k} f(x_k).$$

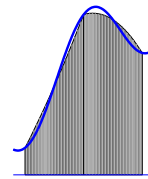


Integrating this linear interpolant leads to the *trapezoidal rule*

$$(5.2) \quad \tilde{I}_k = \frac{1}{2} h_k (f(x_{k-1}) + f(x_k)).$$

Better accuracy can be obtained by using a quadratic interpolant on $[x_{k-1}, x_k]$ with nodes x_{k-1}, x_k and $x_{k-1/2} = \frac{1}{2}(x_{k-1} + x_k)$. A little calculation shows that this leads to the *Simpson rule*

$$(5.3) \quad \tilde{I}_k = \frac{1}{6} h_k (f(x_{k-1}) + 4f(x_{k-1/2}) + f(x_k)).$$



For the approximation of the integral $I = \sum_{k=1}^m I_k$ application of such a quadrature rule leads to a composite integration scheme $\tilde{I} = \sum_{k=1}^m \tilde{I}_k$. For example, if we have $h_j = h$ for all j , then the trapezoidal rule gives

$$(5.4) \quad \tilde{I} = \frac{1}{2} h f(x_0) + h f(x_1) + h f(x_2) + \dots + h f(x_{m-1}) + \frac{1}{2} h f(x_m),$$

and Simpson's rule leads to

$$(5.5) \quad \begin{aligned} \tilde{I} = & \frac{1}{6}hf(x_0) + \frac{2}{3}hf(x_{\frac{1}{2}}) + \frac{1}{3}hf(x_1) + \frac{2}{3}hf(x_{\frac{3}{2}}) + \dots \\ & \dots + \frac{1}{3}hf(x_{m-1}) + \frac{2}{3}hf(x_{m-\frac{1}{2}}) + \frac{1}{6}hf(x_m). \end{aligned}$$

Illustration. As a simple illustration of the performance of the composite formulas we compute approximations to

$$I = \int_0^{1/2} \sin(\pi x) dx = \frac{1}{\pi},$$

with a uniform partitioning $x_j = jh$, $h = 1/(2m)$. Along with the results \tilde{I}_T for the trapezoidal rule (5.4) and \tilde{I}_S for the Simpson rule (5.4), we also consider the result of the simple Riemann sum

$$(5.6) \quad \tilde{I}_R = hf(x_0) + hf(x_1) + \dots + hf(x_{m-1}).$$

The performance of these composite formulas is given in the following table, and it is obvious that the Riemann sum gives larger errors than the trapezoidal rule, whereas the composite Simpson rule gives the most accurate results, by far, in this example.

Table 5.1: Errors $|I - \tilde{I}|$ for the composite Riemann sum \tilde{I}_R , the composite trapezoidal rule \tilde{I}_T and the composite Simpson rule \tilde{I}_S , with $I = \int_0^{1/2} \sin(\pi x) dx$.

m	1	2	4	8	16
$ I - \tilde{I}_R $	$3.2 \cdot 10^{-1}$	$1.4 \cdot 10^{-1}$	$6.6 \cdot 10^{-2}$	$3.2 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$
$ I - \tilde{I}_T $	$6.8 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$	$4.1 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$	$2.6 \cdot 10^{-4}$
$ I - \tilde{I}_S $	$7.2 \cdot 10^{-4}$	$4.2 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$	$1.6 \cdot 10^{-7}$	$1.0 \cdot 10^{-8}$

General Quadrature Formulas. To simplify the notation let us first look at the integral

$$(5.7) \quad J = \int_0^h \varphi(x) dx.$$

Later this can be transformed back to I_k by setting $\varphi(x) = f(x_{k-1} + x)$. We approximate J by the general *quadrature formula* – also called *quadrature method* –

$$(5.8) \quad \tilde{J} = h \sum_{i=1}^s b_i \varphi(c_i h)$$

with $b_i, c_i \in \mathbb{R}$ and with $s \geq 1$ an integer. These b_i and c_i are called the *weights* and *nodes* c_i of the quadrature formula. For example, with the trapezoidal rule we

have $s = 2$, $b_1 = b_2 = \frac{1}{2}$, $c_1 = 0$, $c_2 = 1$, and Simpson's rule has $s = 3$, $b_1 = b_3 = \frac{1}{6}$, $b_2 = \frac{4}{6}$, $c_1 = 0$, $c_2 = \frac{1}{2}$, $c_3 = 1$.

We will say that the quadrature formula has *order* p if $\tilde{J} = J$ whenever φ is a polynomial of degree $\leq p-1$. To express the order conditions in a simple algebraic way, we use the convention $c_i^0 = 1$, also if $c_i = 0$.

Theorem 5.1 The quadrature formula (5.8) has order p if and only if

$$(5.9) \quad \sum_{i=1}^s b_i c_i^{j-1} = \frac{1}{j} \quad \text{for } j = 1, 2, \dots, p.$$

The proof of this result easily follows by considering $\varphi(x) = x^{j-1}$ for $j = 1, \dots, p$ to show necessity of (5.9), and then $\varphi(x) = \sum_{j=0}^{p-1} \alpha_j x^j$ to show sufficiency.

For a method of order p we denote

$$(5.10) \quad C_j = \frac{1}{(j-1)!} \left(\frac{1}{j} - \sum_{i=1}^s b_i c_i^{j-1} \right) \quad (j \geq p+1).$$

If the method has order p , but not $p+1$, then $C_{p+1} \neq 0$ is called the *error constant* of the method.

If $p = s$ and the nodes c_i are distinct, the relation (5.9) can be viewed as an $s \times s$ linear system for the weights b_i ,

$$(5.11) \quad \begin{pmatrix} 1 & 1 & \dots & 1 \\ c_1 & c_2 & \dots & c_s \\ \vdots & \vdots & & \vdots \\ c_1^{s-1} & c_2^{s-1} & \dots & c_s^{s-1} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \vdots \\ \frac{1}{s} \end{pmatrix}.$$

The matrix for this linear system is a so-called Vandermonde matrix, and such a matrix is nonsingular if the c_i are distinct; see Exercise 5.2. As a consequence, for given distinct nodes c_i we can always find corresponding unique weights b_i such that we have order $p = s$ (at least). A natural question is whether the order can be larger than s .

Example 5.2 The trapezoidal rule has order $p = s = 2$ and $C_3 = -\frac{1}{12}$. The fact that we have order two is not very surprising: we used linear interpolation to construct it. A little more surprising is the fact that Simpson's rule has order $p = s + 1 = 4$ and $C_5 = -\frac{1}{2880}$. The underlying reason for having order $s + 1$ will be explained in the next section. \diamond

The Error for Composite Integration. To derive an error bound for the composite integration scheme, we first take a look at the error for $J = \int_0^h \varphi(x) dx$. It will be assumed that φ is a smooth function so that all derivatives appearing in the analysis exist and are bounded. We will use the Taylor expansion

$$\varphi(h) = \varphi(0) + \varphi'(0)h + \dots + \frac{1}{r!}\varphi^{(r)}(0)h^r + R_{r+1}.$$

The remainder term R_{r+1} can be bounded as $|R_{r+1}| \leq \frac{1}{(r+1)!} h^{r+1} \max_{[0,h]} |\varphi^{(r+1)}|$. Omitting for the moment this remainder term, or setting $r = \infty$, it follows that

$$\begin{aligned} J - \tilde{J} &= h \left(\int_0^1 \varphi(th) dt - \sum_{i=1}^s b_i \varphi(c_i h) \right) \\ &= \sum_{q \geq 0} \frac{1}{q!} h^{q+1} \left(\int_0^1 t^q dt - \sum_{i=1}^s b_i c_i^q \right) \varphi^{(q)}(0) \\ &= C_{p+1} h^{p+1} \varphi^{(p)}(0) + C_{p+2} h^{p+2} \varphi^{(p+1)}(0) + \dots \end{aligned}$$

This series can be terminated with an $\mathcal{O}(h^{r+2})$ remainder term if φ is $r+1$ times differentiable.

Now, let us consider the error $I - \tilde{I} = \sum_{k=1}^m (I_k - \tilde{I}_k)$ for the composite integration scheme. Then we directly obtain

$$(5.12) \quad |I - \tilde{I}| \leq \sum_{k=1}^m \left(|C_{p+1}| h_k^{p+1} \max_{x \in [x_{k-1}, x_k]} |f^{(p)}(x)| + \mathcal{O}(h_k^{p+2}) \right).$$

In terms of the maximal mesh-width $h = \max_k h_k$, this can be estimated by

$$(5.13) \quad |I - \tilde{I}| \leq (b-a) |C_{p+1}| h^p \max_{x \in [a,b]} |f^{(p)}(x)| + \mathcal{O}(h^{p+1}).$$

This shows *convergence with order p* upon refinement of the partitioning, $h \rightarrow 0$, under the assumption that the relevant derivatives of f are bounded by some moderate constant.

5.2 Super-convergence and Gauss Quadrature

As we already saw, for given distinct nodes c_i it is always possible to select the weights b_i such that the order is $p = s$. This does not exclude the possibility of having a larger order. Methods with $p > s$ are often called *super-convergent*.

A quadrature formula is said to be *symmetric* if $b_i = b_{s+1-i}$ and $c_i = 1 - c_{s+1-i}$ for $i = 1, 2, \dots, s$. Having symmetry is quite natural; it means that $\int_0^h \varphi(x) dx$ is approximated the same way as the mirror integral $\int_0^h \varphi(h-x) dx$, and in general there is no reason to have a directional preference.

When studying the order of a quadrature formula it is convenient to take $h = 1$. The factor h is just a scaling in (5.7) and (5.8). So we consider in this section

$$(5.14) \quad J = \int_0^1 \varphi(x) dx, \quad \tilde{J} = \sum_{i=1}^s b_i \varphi(c_i),$$

and we wonder for what degree polynomials we will have $J = \tilde{J}$.

Theorem 5.3 A symmetric quadrature formula has an even order p : if the formula is exact for any polynomial of degree $\leq 2k-2$ then it also exact for polynomials of degree $2k-1$.

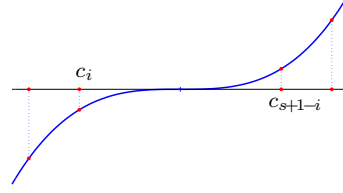
Proof. Any polynomial φ on $[0, 1]$ of degree $2k - 1$ can be written as

$$\varphi(x) = \alpha \left(x - \frac{1}{2}\right)^{2k-1} + R(x)$$

with R a polynomial of degree $\leq 2k - 2$ and $\alpha \in \mathbb{R}$ the leading coefficient of φ . If the order of the quadrature formula is $2k - 1$ at least, then R is integrated exactly.

If the formula is symmetrical, then also

$\int_0^1 (x - \frac{1}{2})^{2k-1} dx = 0$ will be integrated exactly by the quadrature rule, because each term $b_i(c_i - \frac{1}{2})^{2k-1}$ in the quadrature formula is cancelled by its counterpart $b_{s+1-i}(c_{s+1-i} - \frac{1}{2})^{2k-1} = b_i((1 - c_i) - \frac{1}{2})^{2k-1} = -b_i(c_i - \frac{1}{2})^{2k-1}$.



□

This explains why the Simpson rule has order four: it clearly has order three, at least, because it is based on quadratic interpolation, but the method is also symmetric.

As we will see shortly, orders larger than $s + 1$ are possible with suitable choices of the nodes.

Theorem 5.4 Suppose the quadrature formula (5.8) has order $p \geq s$, and let

$$(5.15) \quad M(x) = (x - c_1)(x - c_2) \cdots (x - c_s).$$

Then we have order $p \geq s + q$ if and only if

$$(5.16) \quad \int_0^1 M(x)Q(x) dx = 0 \quad \text{for any polynomial } Q \text{ of degree } \leq q - 1.$$

Proof. If φ is a polynomial of degree $\leq s + q - 1$, we can write it as

$$\varphi(x) = M(x)Q(x) + R(x)$$

with $\deg(R) \leq s - 1$ and $\deg(Q) \leq q - 1$. This decomposition of φ (which is polynomial division with remainder) follows by taking R to interpolate φ at the points c_1, \dots, c_s . Then $\deg(\varphi - R) \leq s + q - 1$ and since $\varphi(c_i) - R(c_i) = 0$, the polynomial $\varphi(x) - R(x)$ will be of the form $M(x)Q(x)$ with $\deg(Q) \leq q - 1$.

From the decomposition of φ we see that the exact integral and numerical approximation satisfy

$$\begin{aligned} \int_0^1 \varphi(x) dx &= \int_0^1 M(x)Q(x) dx + \int_0^1 R(x) dx, \\ \sum_{i=1}^s b_i \varphi(c_i) &= \sum_{i=1}^s b_i M(c_i)Q(c_i) + \sum_{i=1}^s b_i R(c_i) = \sum_{i=1}^s b_i R(c_i). \end{aligned}$$

Since the quadrature formula has order s at least, it is exact for R , that is $\int_0^1 R(x) dx = \sum_{i=1}^s b_i R(c_i)$. Therefore the quadrature formula will also be exact for φ if $\int_0^1 M(x)Q(x) dx = 0$ holds. So we see that (5.16) implies $p \geq s + q$.

On the other hand, if $p \geq s + q$ and $\deg(Q) \leq q - 1$, then $\deg(M \cdot Q) \leq s + q - 1$ and therefore $\int_0^1 M(x)Q(x) dx = \sum_i b_i M(c_i)Q(c_i) = 0$. □

A direct consequence of this lemma is that the order of a quadrature method can not exceed $2s$ (because if it were larger than $2s$ we would have $\int_0^1 M(x)^2 dx = 0$ according to (5.16), which is not possible).

The Gauss methods. The condition (5.16) is equivalent to

$$(5.17) \quad \int_0^1 M(x) x^{j-1} dx = 0 \quad \text{for } j = 1, \dots, q.$$

This constitutes a system of q equations for c_1, \dots, c_s . As we will see this can be solved with $q = s$, leading to the *Gauss quadrature methods* of order $2s$. Note that we need to consider here only the choice of the nodes c_i , because the weights b_i are determined by (5.11) once the c_i have been chosen.

If we take $s = 1$, it is directly seen that the method has order 2 iff $c_1 = \frac{1}{2}$ with weight $b_1 = 1$. This is the one-stage Gauss method, better known as the *midpoint rule*. It is somewhat related to the trapezoidal rule, but the error constant of the midpoint rule is smaller in absolute value.

The nodes and weights of the first three Gauss methods are given by Table 5.2. As we will see shortly, the nodes c_i of the Gauss methods are the zeros of Legendre polynomials. For the higher order methods these coefficients c_i and corresponding b_i can be found in tables or computed numerically.

Table 5.2: Coefficients and error constants for the Gauss methods with $s = 1, 2, 3$.

	p	(b_1, \dots, b_s)	(c_1, \dots, c_s)	C_{p+1}
$s = 1$	2	1	$\frac{1}{2}$	$\frac{1}{24}$
$s = 2$	4	$(\frac{1}{2}, \frac{1}{2})$	$(\frac{1}{2} - \frac{\sqrt{3}}{6}, \frac{1}{2} + \frac{\sqrt{3}}{6})$	$\frac{1}{4320}$
$s = 3$	6	$(\frac{5}{18}, \frac{8}{18}, \frac{5}{18})$	$(\frac{1}{2} - \frac{\sqrt{15}}{10}, \frac{1}{2}, \frac{1}{2} + \frac{\sqrt{15}}{10})$	$\approx 4.96 \cdot 10^{-7}$

In view of the high accuracy of the Gauss methods with large s it is tempting to apply them directly on the interval $[a, b]$ without any partitioning. However, usually the size of the higher derivatives appearing in error bounds are difficult to estimate, and then it is not clear a-priori how large s should be taken to get an error below a given tolerance Tol . Moreover, if the behaviour of f is changing over the integration interval, for instance smooth in one region but rapidly varying in other regions, then partitioning of the interval will be needed anyway to get an efficient integration.

Legendre polynomials.* To obtain the coefficients of the Gauss methods we consider the *Legendre polynomials* on the interval $[-1, 1]$. The Legendre polynomial P_n of degree n is given by the formula

$$(5.18) \quad P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n \quad (n = 0, 1, 2, \dots),$$

where the multiplying factor is chosen such that $P_n(1) = 1$. The important property for us is

$$(5.19) \quad \int_{-1}^1 P_n(x) P_j(x) dx = 0 \quad (j = 0, 1, \dots, n-1).$$

This can be shown, by a somewhat tedious calculation, using $(j+1)$ times integration by parts (each time with differentiation for P_j), and using the fact that the k -th derivative of $(x^2 - 1)^n$ is zero at $x = \pm 1$ when $k < n$.

The property (5.19) implies that $\{P_0, P_1, \dots, P_n\}$ form a basis for the polynomials of degree n , and we have $\int_{-1}^1 P_n(x) Q(x) dx = 0$ whenever Q is a polynomial of degree less than n .

The Legendre polynomials satisfy recursively $P_0(x) = 1$, $P_1(x) = x$ and

$$(5.20) \quad P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x) \quad (n \geq 2),$$

see Exercise 5.7. The first few polynomials are displayed in Figure 5.1.

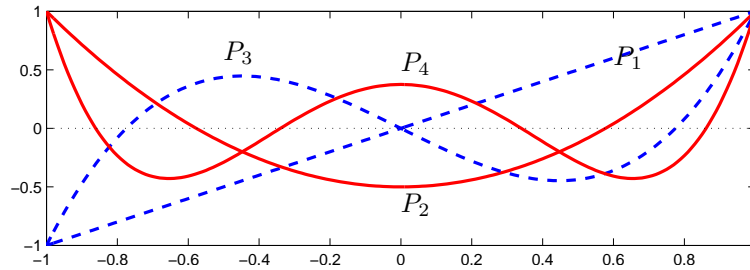


Figure 5.1: Legendre polynomials P_1, P_2, P_3, P_4 on $[-1, 1]$.

Finally, we mention the property that all roots of the polynomials P_n are in $(-1, 1)$. To see this, let r_1, \dots, r_k be the points in $(-1, 1)$ where P_n changes sign, and let $Q(x) = \prod_{i=1}^k (x - r_i)$. Then it is clear that $P_n(x)Q(x)$ does not change sign on $[-1, 1]$, but if $k < n$ we also know that $\int_{-1}^1 P_n(x)Q(x) dx = 0$. It thus follows that we must have $k = n$, and hence P_n has n distinct roots in $(-1, 1)$.

Returning to our quadrature methods, we see that if we take $M(x)$ to be a multiple of $P_s(2x - 1)$ then (5.16) is satisfied with $q = s$. Hence the Gauss methods of order $2s$ are such that the nodes c_i are the zeros of $P_s(2x - 1)$, and the weights are then defined by (5.11).

Remark 5.5 For some applications it is natural to have either $c_1 = 0$ or $c_s = 1$. This leads to the *Radau quadrature* methods with order $2s - 1$. If both $c_1 = 0$ and $c_s = 1$ are prescribed, then an order $2s - 2$ can be reached, and this is called *Lobatto quadrature*.

Other popular methods are obtained by basing the nodes on the zeros or extrema of Chebyshev polynomials. The resulting methods are known as *Clenshaw-Curtis formulas*. They only have order s , but are still very accurate due to very small error constants.

Finally we mention that for the integration of a *periodic function* over one period, the most simple choice $c_i = (i - 1)/s$, $b_i = 1/s$ (equidistant nodes and equal weights) is optimal. This surprising result is explained by (4.11). The composite trapezoidal rule (5.4) has the same optimal form, due to $f(x_0) = f(x_m)$ by periodicity. \diamond

5.3 Practical Error Estimation and Partitioning

To find an approximation \tilde{I} of the integral $I = \int_a^b f(x) dx$ we will usually employ a partitioning of $[a, b]$ in sub-intervals $[x_{k-1}, x_k]$. As before, let $h_k = x_k - x_{k-1}$. Suppose that *Tol* is a given tolerance. If we can ensure that

$$|I_k - \tilde{I}_k| \leq h_k \cdot \text{Tol} \quad (k = 1, \dots, m),$$

then the total error will be bounded by

$$|I - \tilde{I}| \leq \sum_{k=1}^m |I_k - \tilde{I}_k| \leq (b - a) \cdot \text{Tol}.$$

Embedded methods. To find upper bounds for $|I_k - \tilde{I}_k|$ in a strict mathematical sense is often difficult or impossible in practical problems. Instead, we therefore look for heuristic bounds that are easy to compute.

Suppose that along with our method $(b_i, c_i)_{i=1}^s$ of order p we consider a second method $(\hat{b}_i, \hat{c}_i)_{i=1}^{\hat{s}}$ of order $\hat{p} < p$. Then the difference between the two quadrature methods

$$\hat{E}_k = h_k \sum_{i=1}^s b_i f(x_{k-1} + c_i h_k) - h_k \sum_{i=1}^{\hat{s}} \hat{b}_i f(x_{k-1} + \hat{c}_i h_k)$$

can be used to get a heuristic upper bound for $E_k = I_k - \tilde{I}_k$ on the sub-interval $[x_{k-1}, x_k]$, because we know $|E_k| = \mathcal{O}(h_k^{p+1})$, $|\hat{E}_k| = \mathcal{O}(h_k^{\hat{p}+1})$, and therefore $|E_k| \leq |\hat{E}_k|$ as $h_k \rightarrow 0$. So, although this heuristic bound is valid only if h_k is ‘sufficiently small’, we shall simply use $|\hat{E}_k|$ as an error estimator without questioning how small ‘sufficiently small’ is.

To compute $|\hat{E}_k|$ cheaply it is advantageous to have the same nodes in the two methods, $\hat{c}_i = c_i$, because then no new f -evaluations are needed for the error estimator. Of course we should have $\hat{b}_i \neq b_i$ for at least one index i . Such a combination is called an *embedded method*.

Example 5.6 For the trapezoidal rule $\tilde{I}_k = \frac{1}{2}h_k(f(x_{k-1}) + f(x_k))$ we can use the simple method

$$(5.21) \quad \hat{I}_k = h_k f(x_{k-1})$$

of order $\hat{p} = 1$ to estimate the error.

An embedded method for Simpson's rule $\tilde{I}_k = \frac{1}{6}h_k(f(x_{k-1}) + 4f(x_{k-\frac{1}{2}}) + f(x_k))$ is obtained by using either the trapezoidal rule or the midpoint rule (one-stage Gauss method) as low-order method, $\hat{p} = 2$. \diamond

Remark 5.7 In general, $|\hat{E}_k|$ will be a rather poor estimate for the error $|E_k| = |I_k - \tilde{I}_k|$ over the sub-interval $[x_{k-1}, x_k]$. Actually, it usually is a good estimate for the error of the secondary method $(\tilde{b}_i, \tilde{c}_i)_{i=1}^{\tilde{s}}$, but that makes it too pessimistic for our original, primary method. A possible remedy for this is to consider a third method $(\check{b}_i, \check{c}_i)_{i=1}^{\check{s}}$ with order $\check{p} < \hat{p}$, and

$$\check{E}_k = h_k \sum_{i=1}^s b_i f(x_{k-1} + c_i h_k) - h_k \sum_{i=1}^{\check{s}} \check{b}_i f(x_{k-1} + \check{c}_i h_k)$$

and then use, with $q = (p - \hat{p})/(\hat{p} - \check{p})$,

$$\bar{E}_k = \hat{E}_k \cdot (\hat{E}_k / \check{E}_k)^q \quad (\sim h_k^{\hat{p}+1} (h_k^{\hat{p}+1} / h_k^{\check{p}+1})^q = h_k^{p+1})$$

as a better estimate for the error. \diamond

A simple automatic partitioning. If we accept one of the above estimates $|\hat{E}_k|$ or $|\bar{E}_k|$ as an error estimate, then this can be used in a variety of ways to find a partitioning of the integration interval $[a, b]$.

A very simple procedure is as follows: start with the whole interval, apply the quadrature method, and estimate the error. If the estimated error is larger than $(b - a)Tol$ then divide the interval in two and repeat the procedure on the two sub-intervals. Continuing this way, we eventually end up with a partitioning for which the estimated error is less than the acceptable error level.

This procedure is in fact a bit too simple. For example, with $f(x) = \sin(2\pi)^2$ it might be concluded that the integral $\int_0^1 f(x) dx$ is zero if the initial estimate only uses function values at $x = 0, \frac{1}{2}, 1$. The initial partitioning should somewhat match the variations in f .

There are other partitioning strategies. In the next section we will come back to this issue in connection with step-size selection for ODE solvers.

5.4 Exercises

Exercise 5.1. Derive the trapezoidal rule (5.2) and Simpson's rule (5.3) from the corresponding Lagrange interpolation formulas. (It is easiest to first consider the interval $J = [0, h]$ instead of $I_k = [x_{k-1}, x_k]$.)

Exercise 5.2. Assume $c_i \neq c_j$ if $i \neq j$. Show that the Vandermonde matrix (5.11) is non-singular, by considering the transpose matrix and the interpolating polynomial $P(x) = \sum_{j=1}^s a_j x^{j-1}$ such that $P(c_j) = f_j$ ($1 \leq j \leq s$).

Exercise 5.3. For the general quadrature formula (5.8): assume that the method is obtained by interpolation through $(c_i h, \varphi(c_i h))$, $i = 1, \dots, s$ with a polynomial of degree $\leq s - 1$, and subsequently we solve the integral for this polynomial interpolant exactly. Show that the order of this method will be s (at least), and

$$b_i = \int_0^1 L_i(x) dx, \quad L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^s \frac{x - c_j}{c_i - c_j}.$$

Exercise 5.4. Suppose the quadrature formula has order $p \geq s$, and $c_i = 1 - c_{s+1-i}$ for $i = 1, \dots, s$ (symmetry of the nodes). Show that then $b_i = b_{s+1-i}$ for all i (symmetry of the weights), using the result of the previous exercise.

Exercise 5.5. Derive the coefficients of the Gauss method with $s = 2$ from (5.17).

Exercise 5.6 (2D integration). Consider a two-dimensional integral

$$I = \int_0^1 \int_0^1 g(x, y) dy dx.$$

Give a composite quadrature formula for this integral based on the trapezoidal rule with equidistant nodes $x_i = ih$ and $y_j = jh$ in the x - and y -direction, with $h = 1/m$. (Hint: Define $f(x) = \int_0^1 g(x, y) dy$.)

*Exercise 5.7.** Consider the Legendre polynomials P_n on $[-1, 1]$.

(a) Show that $P_n(-x) = (-1)^n P_n(x)$, and therefore $P_n(-1) = (-1)^n$.

(b) Write $xP_{n-1}(x)$ as a linear combination $\sum_{j=0}^n \alpha_j P_j(x)$. Show that $\alpha_j = 0$ if $j \leq n - 2$. The remaining coefficients $\alpha_n, \alpha_{n-1}, \alpha_{n-2}$ can be found by considering $x = 1$, $x = -1$ as well as the coefficients in front of the highest powers. Show that (5.20) is the result.

Exercise 5.8. Consider embedded methods with Simpson's rule ($p = 4$) as primary method and with $\hat{p} = 2$. Which methods are suited as secondary methods? Explain why it is not possible to get $\hat{p} = 3$ if we want to avoid new function evaluations.

Exercise 5.9 (programming). Approximate the integral $I = \int_0^5 (\sin x^2) e^{-x/3} dx$ with the composite formulas (5.4) and (5.5). To get a feeling for the number of points needed to get an accuracy of 10^{-3} , experiment with increasing m , keeping track which decimal digits change.

Exercise 5.10 (programming). Write a program for solving $I = \int_a^b f(x) dx$ with automatic partitioning using Simpson's rule as primary method and the trapezoidal rule as secondary method (for error estimation). To test the program, apply it with $a = 0$, $b = \frac{\pi}{2}$ and $f(x) = e^{2x} \cos x$, for which we know that $I = \frac{1}{5}(e^\pi - 2)$.

6 Initial Value Problems for ODEs

In this section we will discuss some numerical methods for the solution of systems of ordinary differential equations (ODEs)

$$\begin{cases} u'_1(t) = f_1(t, u_1(t), u_2(t), \dots, u_m(t)), \\ u'_2(t) = f_2(t, u_1(t), u_2(t), \dots, u_m(t)), \\ \vdots \\ u'_m(t) = f_m(t, u_1(t), u_2(t), \dots, u_m(t)), \end{cases}$$

with time $t \in [0, T]$. In vector notation, this will be written as $u'(t) = f(t, u(t))$ with unknown $u(t) = (u_i(t)) \in \mathbb{R}^m$ and given function $f : \mathbb{R} \times \mathbb{R}^m \rightarrow \mathbb{R}^m$. It will be assumed that $u(t)$ is specified at initial time $t = 0$. We then have an *initial value problem* for a system of ODEs,

$$(6.1) \quad u'(t) = f(t, u(t)), \quad u(0) = u_0,$$

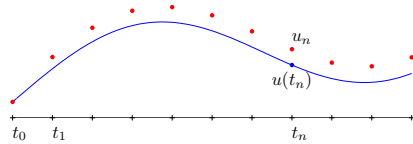
with given initial value $u_0 \in \mathbb{R}^m$.

Let $\|\cdot\|$ be a vector norm on \mathbb{R}^m . The function f is said to satisfy a *Lipschitz condition* if

$$(6.2) \quad \|f(t, \tilde{v}) - f(t, v)\| \leq L \|\tilde{v} - v\| \quad \text{for all } t \in [0, T] \text{ and } \tilde{v}, v \in \mathbb{R}^m,$$

with $L > 0$ called the Lipschitz constant. If this holds and f is continuous in t , then it is known (Picard's theorem) that for any given $u_0 \in \mathbb{R}^m$ the initial value problem (6.1) has a unique solution on $[0, T]$. Moreover, if f is q times differentiable, then the solution u will be $q+1$ times differentiable on $[0, T]$. If the Lipschitz condition is only valid for \tilde{v}, v on some bounded subset $\mathcal{D} \subset \mathbb{R}^m$, then existence and uniqueness of solutions is guaranteed locally. For the theoretical results in this section it will be assumed for convenience that the global Lipschitz condition (6.2) is satisfied.

In the following, we consider numerical approximations $u_n \approx u(t_n)$ at the points $t_n = n\tau$, $n = 0, 1, 2, \dots$, with $\tau > 0$ being the *step-size*. For the moment this step-size τ is assumed to be constant; variable steps will be discussed later.



6.1 Runge-Kutta Methods

We will first consider some simple methods, and then we will see that they all fall in the class of Runge-Kutta methods. Note that the solution of the initial value problem (6.1) satisfies

$$(6.3) \quad u(t_n) = u(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(t, u(t)) dt,$$

for $n = 1, 2, \dots, N$ with number of steps N such that $N\tau = T$. This will provide a connection with the quadrature methods of the previous section.

Euler's Method. The most simple numerical method for solving the initial value problem is *Euler's method*

$$(6.4) \quad u_n = u_{n-1} + \tau f(t_{n-1}, u_{n-1}).$$

It will be shown later that Euler's method does convergence to the exact solution on any bounded time interval $[0, T]$ if f satisfies a Lipschitz condition. However, the error will be proportional with τ only (convergence with order one).

Trapezoidal Rule. Applying the trapezoidal quadrature rule (5.2) to (6.3) gives the following ODE method, called the *implicit trapezoidal rule*,

$$(6.5) \quad u_n = u_{n-1} + \frac{1}{2}\tau f(t_{n-1}, u_{n-1}) + \frac{1}{2}\tau f(t_n, u_n).$$

This method is called implicit because to find the new approximation u_n we need to solve a system of equations.

Such an implicit system is avoided if we first compute a predictor u_n^* by Euler's method and substitute this in the right-hand side. The resulting method becomes

$$(6.6) \quad \begin{aligned} u_n^* &= u_{n-1} + \tau f(t_{n-1}, u_{n-1}), \\ u_n &= u_{n-1} + \frac{1}{2}\tau f(t_{n-1}, u_{n-1}) + \frac{1}{2}\tau f(t_n, u_n^*). \end{aligned}$$

This method is known as the *explicit trapezoidal rule* or the *modified Euler method*.

As we will see below, both (6.5) and (6.6) are convergent with an error proportional to τ^2 (convergence with order two). Of course, computing the new approximation u_n from (6.6) is much easier than from its implicit counterpart (6.5). Nevertheless, we will see later that there is an important class of initial value problems – the so called stiff problems – for which (6.5) is to be preferred.

In this section we will mainly discuss explicit methods, but the theoretical results on convergence are also valid for implicit methods.

Illustration. We consider the following initial value problem with two components, $u(t) = (v(t), w(t))^T$ in \mathbb{R}^2 ,

$$(6.7) \quad \begin{cases} v'(t) = (1 - w(t)) v(t), & v(0) = 0.1, \\ w'(t) = (-1 + 1.2 v(t)) w(t), & w(0) = 1, \end{cases}$$

describing population densities in a simple predator-prey (Lotka-Volterra) model. The time runs from $t = 0$ till $t = T$ with end-point $T = 15$. The solution is known to be periodic in time, even though the solution itself and its period are not known. But we can compute an accurate approximation by any of the above methods by choosing a very small time step.

Figure 6.1 contains plots of an accurate approximation. The left panel of the figure shows $v(t)$ and $w(t)$ as function of $t \in [0, 15]$. The right panel gives a plot of the trajectory $(v(t), w(t))$ in the (v, w) -plane, the so-called phase plane. The periodic nature is then more clear.

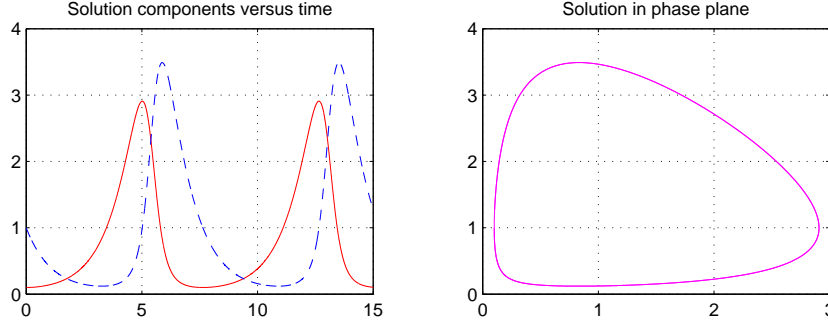


Figure 6.1: Solution for the Lotka-Volterra problem (6.7). Left panel: components $v(t)$ (solid line) and $w(t)$ (dashed line) as function of time t . Right panel: trajectory $(v(t), w(t))_{t \in [0, T]}$ in the phase plane.

Now we try to compute the solution with the Euler method (6.4) and the explicit trapezoidal rule (6.6), first using $N = 100$ steps and then repeatedly doubling the number of steps. This gives for both methods the error $\|u(t_N) - u_N\|_2$ at the end point $t_N = T$ as function of the step-size $\tau = T/N$ (for $u(t_N)$ a very accurate reference value was computed numerically). The result is presented in Table 6.1.

Table 6.1: Errors for the Lotka-Volterra problem (6.7) at time $T = 15$, with number of steps $N = 100, 200, 400, \dots$ and $\tau = T/N$.

N	100	200	400	800	1600	3200
Err. (6.4)	1.78	4.12	$9.87 \cdot 10^{-1}$	$3.64 \cdot 10^{-1}$	$1.59 \cdot 10^{-1}$	$7.49 \cdot 10^{-2}$
Err. (6.6)	$1.19 \cdot 10^{-2}$	$5.30 \cdot 10^{-3}$	$1.60 \cdot 10^{-3}$	$4.34 \cdot 10^{-4}$	$1.13 \cdot 10^{-4}$	$2.88 \cdot 10^{-5}$

The results with Euler's method are very inaccurate for $N \leq 400$. If more steps are taken, we see that the error halves (approximately) when the number of steps is doubled. It seems that the error is proportional to $\tau \sim 1/N$ for $\tau > 0$ sufficiently small.

The results with the explicit trapezoidal rule (6.6) are much better. The error now appears to be proportional to τ^2 if $\tau > 0$ is sufficiently small. Even though this method requires per step twice as much work as the forward Euler method, this extra effort per step clearly pays off. As we will see later, in Table 6.3, the results can still be considerably improved using higher-order methods.

Runge-Kutta methods. The above methods, and many others, fall in the class of the (general) *Runge-Kutta methods*, where we work successively towards the approximation u_n at the new time level t_n by way of intermediate approximations $v_{n,1}, v_{n,2}, \dots, v_{n,s}$, starting from u_{n-1} . A step with a Runge-Kutta method is

given by

$$(6.8a) \quad v_{n,i} = u_{n-1} + \tau \sum_{j=1}^s a_{ij} f(t_{n-1} + c_j \tau, v_{n,j}) \quad (i = 1, 2, \dots, s),$$

$$(6.8b) \quad u_n = u_{n-1} + \tau \sum_{i=1}^s b_i f(t_{n-1} + c_i \tau, v_{n,i}).$$

The method is specified by the coefficients a_{ij}, b_i, c_i , ($1 \leq i, j \leq s$) where the b_i are called the weights, the c_i are the nodes, and s is the number of stages. The nodes are usually assumed to be in $[0, 1]$, and they are related to the coefficients a_{ij} by

$$(6.9) \quad c_i = \sum_{j=1}^s a_{ij}.$$

The above methods all fit in this framework. Several other examples will be given later. Runge-Kutta methods can be conveniently represented by the tableau of coefficients.

c_1	a_{11}	\cdots	a_{1s}
\vdots	\vdots		\vdots
c_s	a_{s1}	\cdots	a_{ss}
	b_1	\cdots	b_s

The Runge-Kutta method (6.8) method is called *explicit* if $a_{ij} = 0$ whenever $j \geq i$, and otherwise it is called *implicit*. With an explicit method we get an explicit expression for $v_{n,i}$ in terms of already computed u_{n-1} and $v_{n,j}$, $j < i$. If the method is implicit it will be tacitly assumed that the implicit relations are uniquely solvable (which can be proven for sufficiently small τ if (6.2) holds). Finally we mention that the intermediate vectors $v_{n,i}$ are approximations to $u(t_{n-1} + c_i \tau)$, but in general less accurate than the approximation u_n to $u(t_n)$.

Example 6.1 An explicit two-stage Runge-Kutta method can be represented by the tableau

c_1	0	0
c_2	a_{21}	0
	b_1	b_2

with $c_1 = 0$ and $c_2 = a_{21}$. The choice of the coefficients a_{21}, b_1, b_2 determines the method. For autonomous differential equations, $u'(t) = f(u(t))$, the method reads

$$\begin{cases} v_1 = u_{n-1}, \\ v_2 = u_{n-1} + \tau a_{21} f(v_1), \\ u_n = u_{n-1} + \tau b_1 f(v_1) + \tau b_2 f(v_2), \end{cases}$$

with intermediate vectors v_1 and v_2 changing from step to step – which can be emphasized, by giving v_1, v_2 an extra subindex n , as in (6.8). The method can also be written, more compactly, as

$$(6.10) \quad u_n = u_{n-1} + \tau b_1 f(u_{n-1}) + \tau b_2 f(u_{n-1} + \tau a_{21} f(u_{n-1})).$$

If $a_{21} = 1$, $b_1 = b_2 = \frac{1}{2}$ we retrieve the explicit trapezoidal rule (6.6). Another popular choice is $a_{21} = \frac{1}{2}$, $b_1 = 0$, $b_2 = 1$. \diamond

6.2 Consistency

To compute approximations $u_n \approx u(t_n)$ over the whole interval $[0, T]$ we can take N steps with step-size $\tau = T/N$. In each of these steps, bringing us from a point t_{n-1} to t_n , there will be some (small) error in approximating the exact solution $u(t)$. Such an error per step is called a local error. The error at the end-point $u(t_N) - u_N$, which is called a global error, is build-up by N of these local errors. The global errors are studied in the next section. Here we will derive bounds for the local errors.

To get an upper bound on this accumulated global error we need to have a bound for the local errors, but we also need to estimate how these local errors will affect the final result. In this section we will study bounds for the local errors.

Consider a Runge-Kutta method, written in a compact way as

$$(6.11) \quad u_n = u_{n-1} + \tau \Phi_\tau(t_{n-1}, u_{n-1}),$$

with an increment function Φ_τ . See for instance, formula (6.10) for explicit two-stage methods. For an implicit method this increment function will be defined implicitly.

If we insert exact solution values into this formula, we get

$$(6.12) \quad u(t_n) = u(t_{n-1}) + \tau \Phi_\tau(t_{n-1}, u(t_{n-1})) + \tau d_n,$$

with a residual τd_n called the *local error*. The method is said to be *consistent of order p* for the solution u if $d_n = \mathcal{O}(\tau^p)$ as $\tau \rightarrow 0$, uniformly for $t_n \in [0, T]$. Below we will discuss the order of consistency for specific Runge-Kutta methods in some detail.

The local error $\tau d_n = \mathcal{O}(\tau^{p+1})$ can be interpreted as the error introduced in one step: it is the difference between the exact solution at time t_n and the numerical approximation that would have been obtained if we had started with $u_{n-1} = u(t_{n-1})$ at time t_{n-1} .

Example 6.2 The increment function for Euler's method is simply given by $\Phi_\tau(t, v) = f(t, v)$. The local errors are

$$\tau d_n = u(t_n) - u(t_{n-1}) - \tau f(t_{n-1}, u(t_{n-1})).$$

By Taylor expansion of $u(t_n) = u(t_{n-1} + \tau)$ we get

$$u(t_n) = u(t_{n-1}) + \tau u'(t_{n-1}) + \frac{1}{2} \tau^2 u''(t_{n-1}) + \mathcal{O}(\tau^3).$$

Since $f(t_{n-1}, u(t_{n-1})) = u'(t_{n-1})$, it follows that

$$d_n = \frac{1}{2} \tau u''(t_{n-1}) + \mathcal{O}(\tau^2),$$

assuming that f is twice continuously differentiable. This gives a bound $\|d_n\| \leq C\tau$ for $\tau \in (0, \tau_*]$ for τ_* sufficiently small, with $C = \max_{[0, T]} \|u''(t)\|$. The method is therefore consistent of order $p = 1$. \diamond

Order Conditions. To obtain a Runge-Kutta method with order of consistency p for arbitrary, smooth solutions u – simply called a *method of order p* – there are a number of algebraic conditions on the coefficients of the method that must be satisfied. When deriving these so-called order conditions it will always be tacitly assumed that f is $p+1$ times differentiable. The way to derive the order conditions is by making Taylor expansions in powers of τ of one step of the numerical method and the exact solution.

Example 6.3 (Conditions for order one and two) Without loss of generality, we can study the local error in the first step, setting $n = 1$, that is, we study $\tau d_1 = u(t_1) - u_1$ starting with $u(0) = u_0$. Moreover, it is convenient to restrict ourselves to autonomous differential equations $u'(t) = f(u(t))$, where f does not depend explicitly on t . Actually, this does not lead to a loss of generality, as will be seen in Exercise 6.4.

We want to find the order p such that $u(t_1) - u_1 = \mathcal{O}(\tau^{p+1})$, and we consider here the conditions for having $p = 1$ and $p = 2$. For this we will make a Taylor series development of $u(t_1)$ and u_1 in powers of τ .

The first two derivatives of u at $t = 0$ are found (chain rule) to be

$$u'(0) = f_0, \quad u''(0) = f'_0 \cdot f_0,$$

where $f_0 = f(u_0)$ and $f'_0 = f'(u_0)$. Hence for the exact solution we obtain the Taylor expansion

$$(6.13) \quad u(t_1) = u_0 + \tau f_0 + \frac{1}{2}\tau^2 f'_0 \cdot f_0 + \mathcal{O}(\tau^3).$$

For the first Runge-Kutta step (6.8) we have $v_i = u_0 + \tau \sum_j a_{ij} f(v_j)$ s, with summations in this example ranging from 1 to s . This gives

$$f(v_i) = f(u_0) + f'(u_0) \cdot \tau \sum_j a_{ij} f(v_j) + \mathcal{O}(\tau^2).$$

Since $v_j = u_0 + \mathcal{O}(\tau)$, $f(v_j) = f(u_0) + \mathcal{O}(\tau)$ and $\sum_j a_{ij} = c_i$, it follows that

$$f(v_i) = f_0 + \tau c_i f'_0 \cdot f_0 + \mathcal{O}(\tau^2).$$

Finally, from $u_1 = u_0 + \tau \sum_i b_i f(v_i)$ we now obtain

$$(6.14) \quad u_1 = u_0 + \left(\sum_i b_i \right) \tau f_0 + \left(\sum_i b_i c_i \right) \tau^2 f'_0 \cdot f_0 + \mathcal{O}(\tau^3).$$

Comparing (6.13) with (6.14), it is seen that we will have order $p = 1$ iff $\sum_i b_i = 1$, and we have $p = 2$ iff the conditions $\sum_i b_i = 1$ and $\sum_i b_i c_i = \frac{1}{2}$ are both satisfied. \diamond

Table 6.2: Order conditions of Runge-Kutta methods for $p = 1, 2, 3, 4$, with the summations from 1 to s , and $c_i = \sum_j a_{ij}$.

order p	order conditions	
1	$\sum_i b_i = 1$	
2	$\sum_i b_i c_i = 1/2$	
3	$\sum_i b_i c_i^2 = 1/3$	$\sum_{i,j} b_i a_{ij} c_j = 1/6$
4	$\sum_i b_i c_i^3 = 1/4$	$\sum_{i,j} b_i c_i a_{ij} c_j = 1/8$
	$\sum_{i,j} b_i a_{ij} c_j^2 = 1/12$	$\sum_{i,j,k} b_i a_{ij} a_{jk} c_k = 1/24$

The conditions on the coefficients of a Runge-Kutta method (6.8) for having order $p = 1$ up to $p = 4$ are listed in Table 6.2. These conditions can be derived by comparing Taylor expansions of $u(t_1)$ and u_1 in powers of τ , as in the previous example. For increasing order p , the derivation of these order conditions does become rather technical.

The order conditions in this table are cumulative. So, for order $p = 4$ there are in total 8 conditions on the coefficients. The number of order conditions quickly increases for higher orders. For example, to have $p = 5$ we get 9 new conditions (giving in total 17 conditions); then, for order $p = 6$ there are 20 additional conditions (in total 37 conditions). Nevertheless, many high-order methods have been constructed which are used in popular codes.

For explicit methods it can be shown (see Corollary 7.3 in the next section) that $p \leq s$. Moreover, it is known that $p = s$ can be achieved with explicit methods only for s up to four.

Example 6.4 Typical, well-known examples of explicit methods of order three and four are given by the following tableaus:

$$(6.15) \quad \text{a) } \begin{array}{c|cc} 0 & & \\ 1/3 & 1/3 & \\ 2/3 & 0 & 2/3 \\ \hline & 1/4 & 0 & 3/4 \end{array} \quad \text{b) } \begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1/2 & 0 & 1/2 & \\ 1 & 0 & 0 & 1 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Method (6.15.a) with $p = s = 3$ is Heun's third-order method. Method (6.15.b) with $p = s = 4$ is known as the *classical Runge-Kutta method*. It can be seen as a generalization of the Simpson quadrature rule. \diamond

Illustration. Consider again the Lotka-Volterra model (6.7), but now solved with the methods (6.15.a) and (6.15.b) with varying number of steps N and $\tau = T/N$. The errors $\|u(t_N) - u_N\|_2$ at the end point $t_N = T$ are given in Table 6.3. These

errors are to be compared with those in Table 6.1 for Euler's method and the explicit trapezoidal rule. We see that, in particular for the fourth-order classical Runge-Kutta method (6.15.b), there is a spectacular improvement in the results compared to the low-order methods (6.4) and (6.6).

Table 6.3: Errors for the Lotka-Volterra problem (6.7) at time $T = 15$, with number of steps $N = 100, 200, 400, \dots$ and $\tau = T/N$.

N	100	200	400	800	1600	3200
Err. (6.15.a)	$6.8 \cdot 10^{-3}$	$8.2 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.3 \cdot 10^{-5}$	$1.6 \cdot 10^{-6}$	$2.0 \cdot 10^{-7}$
Err. (6.15.b)	$9.7 \cdot 10^{-5}$	$8.7 \cdot 10^{-6}$	$6.3 \cdot 10^{-7}$	$4.2 \cdot 10^{-8}$	$2.7 \cdot 10^{-9}$	$1.7 \cdot 10^{-10}$

6.3 Convergence

As starting point for deriving convergence results we consider again

$$(6.16a) \quad u_n = u_{n-1} + \tau \Phi_\tau(t_{n-1}, u_{n-1}),$$

$$(6.16b) \quad u(t_n) = u(t_{n-1}) + \tau \Phi_\tau(t_{n-1}, u(t_{n-1})) + \tau d_n,$$

for $n = 1, 2, \dots, N$, $n\tau = T$. The local errors τd_n provide a measure for the error of the method, but we are actually interested in the *global error* $e_n = u(t_n) - u_n$. The method (6.11) is called *convergent of order p* for the solution u if $e_n = \mathcal{O}(\tau^p)$ as $\tau \rightarrow 0$, uniformly for $t_n \in [0, T]$.

These global errors can be related to the local errors under the assumption of a Lipschitz condition for the increment function. In the following result we also allow for an error $e_0 = u(0) - u_0$ in the initial condition.

Theorem 6.5 Assume there are $\tau_*, L_* > 0$ such that

$$(6.17) \quad \|\Phi_\tau(t, \tilde{v}) - \Phi_\tau(t, v)\| \leq L_* \|\tilde{v} - v\|$$

for all $t = t_n \in [0, T]$, $\tilde{v}, v \in \mathbb{R}^m$ and $\tau \in (0, \tau_*]$. Then, for such τ and t_n , the global error satisfies

$$(6.18) \quad \|e_n\| \leq e^{L_* t_n} \|e_0\| + \frac{1}{L_*} (e^{L_* t_n} - 1) \cdot \max_{1 \leq j \leq n} \|d_j\|.$$

Proof. Subtraction of (6.11) from (6.12) gives

$$(6.19) \quad \|e_n\| \leq (1 + L_* \tau) \|e_{n-1}\| + \tau \|d_n\| \quad (n = 1, 2, \dots, N).$$

Setting $\delta = \frac{1}{L_*} \max_{1 \leq j \leq n} \|d_j\|$, it follows that $(\|e_n\| + \delta) \leq (1 + L_* \tau)(\|e_{n-1}\| + \delta)$, and hence

$$(\|e_n\| + \delta) \leq (1 + L_* \tau)^n (\|e_0\| + \delta).$$

Using the fact that $1 + L_* \tau \leq \exp(L_* \tau)$ now directly leads to (6.18). \square

Remark 6.6 The crucial inequality in this proof is (6.19), showing that the error e_n at time level t_n can be bounded in terms of the error e_{n-1} at the previous time level, multiplied by a factor $\rho = 1 + L_*\tau$, together with the new error τd_n introduced in this step. If we apply this inequality recursively, it follows that

$$\|e_n\| \leq (1 + L_*\tau)^n \|e_0\| + \sum_{j=1}^n (1 + L_*\tau)^{n-j} \tau \|d_j\|,$$

which gives a more detailed estimate for the global error. \diamond

It is clear from the above theorem that convergence of order p for the solution u will hold provided (i) the method is consistent of order p , and (ii) the Lipschitz condition (6.17) holds for the method. This Lipschitz condition is easy to establish for any Runge-Kutta method.

Lemma 6.7 Suppose the Lipschitz condition (6.2) holds for the function f . Consider a Runge-Kutta method (6.8), and let $\alpha = \max_i \sum_j |a_{ij}|$, $\beta = \sum_j |b_j|$. Then the Lipschitz condition (6.17) for the method holds with

$$L_* = \frac{\beta L}{1 - \alpha L \tau_*}, \quad 0 < \tau_* < \frac{1}{\alpha L}.$$

Proof. Consider one step of the Runge-Kutta method (6.8) with intermediate vectors v_i , $i = 1, \dots, s$. The increment is

$$\Phi_\tau(t_{n-1}, u_{n-1}) = \frac{1}{\tau}(u_n - u_{n-1}).$$

Along with this, also consider a step starting from a \tilde{u}_{n-1} with corresponding intermediate vectors \tilde{v}_i , and let $\Delta v_i = v_i - \tilde{v}_i$ and $\Delta u_n = u_n - \tilde{u}_n$. Then

$$\|\Delta v_i\| \leq \|\Delta u_{n-1}\| + L\tau \sum_j |a_{ij}| \|\Delta v_j\| \leq \|\Delta u_{n-1}\| + \alpha L\tau \max_j \|\Delta v_j\|,$$

which gives $(1 - \alpha L\tau) \max_i \|\Delta v_i\| \leq \|\Delta u_{n-1}\|$. Hence, if $\tau \in (0, \tau_*]$ with $\tau_* > 0$ such that $1 - \alpha L\tau_* > 0$, then

$$\|\Delta u_n - \Delta u_{n-1}\| \leq L\tau \sum_j |b_j| \cdot \max_i \|\Delta v_i\| \leq \frac{\beta L\tau}{1 - \alpha L\tau} \|\Delta u_{n-1}\|,$$

from which the result follows. \square

We thus see that for any Runge-Kutta method, consistency of order p implies convergence of order p , under the assumption of the Lipschitz condition (6.2) for the function f .

6.4 Step-size Selection

To solve an initial value problem numerically, we can take a method and apply it on $[0, T]$ with constant step-sizes $\tau = T/N$. Repeating the computation with an other step-size, for instance with 2τ , gives a fair indication of the error.

However, to get an efficient scheme it is important to adapt the step-sizes to match the variations in the solution. This is similar to the case of quadrature formulas, but with ODEs we have in general no idea in advance how the solution will behave, or where it will exhibit rapid variations. In the following, let Tol be a tolerance specified by the user. We will try to select the stepsizes such that the (estimated) local errors are bounded by this number Tol .

Consider an attempted step from t_{n-1} to $t_n = t_{n-1} + \tau_n$ with step-size τ_n . Suppose the method has order p and we have an available estimate \hat{E}_n for the norm of the local error,

$$\hat{E}_n = \gamma_n \tau_n^{\hat{p}+1} + \mathcal{O}(\tau_n^{\hat{p}+2}),$$

with $\hat{p} \approx p$ and some unknown constant $\gamma_n > 0$. Here $\hat{p} = p$ if \hat{E}_n is an estimate of the genuine local error of the method, but often the estimate \hat{E}_n is quite rough and \hat{p} may be less than p . For example, the estimate may be obtained by comparing u_n with the result \hat{u}_n obtained from a lower-order method, say of order $\hat{p} = p - 1$, embedded in the primary method, similar as for quadrature methods.

Having the estimate \hat{E}_n two cases can occur: $\hat{E}_n > Tol$ or $\hat{E}_n \leq Tol$. In the first case we decide to reject this step and to redo it with a smaller step-size $\tau_n = \tau_{new}$, where we aim at $\hat{E}_{new} = Tol$. In the second case the step is accepted, and we continue the integration with $\tau_{n+1} = \tau_{new}$ for the new step from t_n to t_{n+1} , again aiming at $\hat{E}_{new} = Tol$.

The constant γ_n is not known, but we will have approximately $\hat{E}_n = \gamma_n \tau_n^{\hat{p}+1}$ and $\hat{E}_{new} = \gamma_n \tau_{new}^{\hat{p}+1}$. If we require $\hat{E}_{new} = Tol$, we can eliminate γ_n to arrive at

$$(6.20) \quad \tau_{new} = r \tau_n, \quad r = (Tol / \hat{E}_n)^{1/(\hat{p}+1)}.$$

Because rough estimates are used, the expression for the new step-size found in most codes has the form

$$(6.21) \quad \tau_{new} = \min(r_{max}, \max(r_{min}, \vartheta r)) \cdot \tau_n,$$

where r_{max} and r_{min} are a maximal and minimal growth ration, and $\vartheta < 1$ serves to make the estimate conservative so as to avoid repeated rejections. Typical values are $\vartheta \in [0.7, 0.9]$, $r_{min} \in [0.1, 0.5]$ and $r_{max} \in [1.5, 10]$.

Example 6.8 ($p = 2$, $\hat{p} = 1$). As an example we will provide the explicit trapezoidal rule (6.6) with a simple step-size control and illustrate the resulting solver. Since the explicit Euler result $u_n^* = u_{n-1} + \tau f(t_{n-1}, u_{n-1})$ is available already in a step with (6.6), we can use

$$(6.22) \quad \hat{E}_n = \|u_n - u_n^*\|$$

as an estimator for the norm of the local error. Notice that for $u_{n-1} = u(t_{n-1})$ this estimator satisfies

$$\hat{E}_n = \frac{1}{2}\tau^2 \|u''(t_{n-1})\| + \mathcal{O}(\tau^3).$$

This provides an accurate estimator for the local error of Euler's method, but we will use this nevertheless for the second-order method, where we expect it to give an upper bound. Then, in formula (6.20) we have $\hat{p} = 1$. By choosing an appropriate norm for computing \hat{E}_n and by making a choice for the parameters in (6.21) the step-size control can now be used.

As an illustration, we apply this method with error estimator to the problem

$$(6.23) \quad \begin{cases} v'(t) = 1 + v(t)^2 w(t) - 4v(t), & v(0) = 1.01, \\ w'(t) = 3v(t) - v(t)^2 w(t), & w(0) = 3, \end{cases}$$

with end time $T = 20$. It is a simplified chemical model with two chemical species. With these initial values the variation of the solution is small at first. Around time $t = 8$ and $t = 15$ there are large variations, and the step-size should then become smaller.

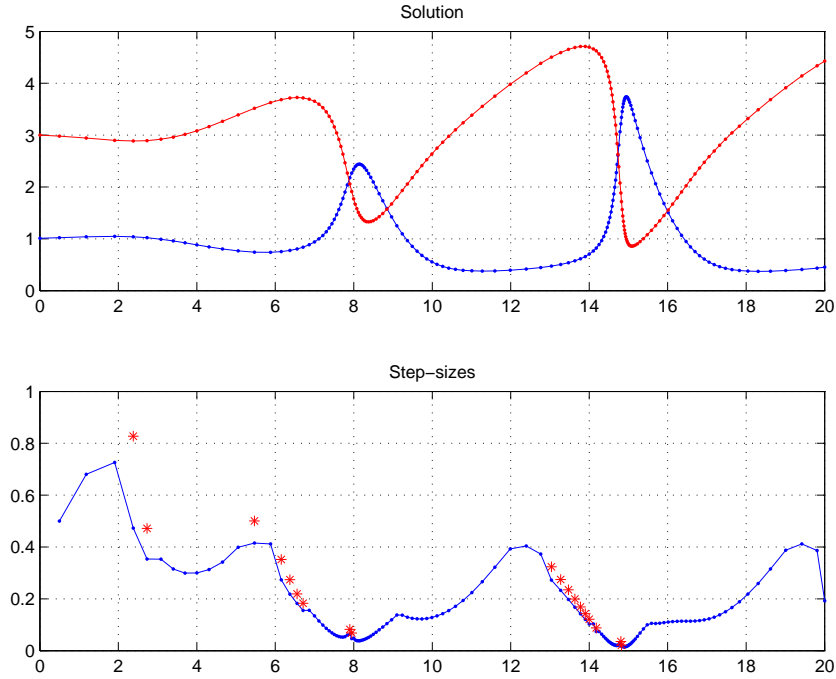


Figure 6.2: Variable step-sizes for (6.23). Top panel: the two solution components versus time. Bottom panel: accepted and rejected (*) step-sizes.

The results are shown in Figure 6.2, with plots of the solution components $v_n \approx v(t_n)$, $w_n \approx w(t_n)$ versus t_n , and also the step-sizes τ_n versus t_n . These results have been obtained with $Tol = 10^{-2}$, $\vartheta = 0.85$, $r_{min} = 0.5$, $r_{max} =$

1.5 and the maximum norm on \mathbb{R}^2 . As we see in the figure, there are quite a number of rejections, so here it might be better to take ϑ slightly smaller, say 0.8. More importantly, for smaller tolerances a large number of steps will be taken. Better efficiency will then be obtained using methods with a higher order. (See Exercise 6.5.) \diamond

It should be stressed that Tol is just an aim for the local errors; it is *not* a guaranteed upper bound for the global errors. When an ODE code (no matter how sophisticated) is used for a new class of problems some representative problems should first be tried with different values of Tol to get a feeling of the actual global accuracy.

Remark 6.9 A starting step-size for the first step on scale with the initial solution variation must be prescribed. Sophisticated ODE codes do this automatically. It can be based on the norms of $\|u_0\|$ and $\|f(t_0, u_0)\|$, by requiring that an Euler step $u_0 + \tau f(t_0, u_0)$ does not differ more than a certain percentage from u_0 .

Further we note that there are many variants for the step-size selection. For instance, we may look at relative errors. Also, instead of $\hat{E}_n \lesssim Tol$ one can aim at having $\hat{E}_n \lesssim (\tau_n/T) Tol$, which is called error control per unit step. That would be similar to the strategy described in the previous section for quadrature methods with automatic partitioning. \diamond

6.5 Exercises

Exercise 6.1. For special differential equations $u'(t) = f(t)$, the initial value problem reduces to a quadrature problem. What does the Runge-Kutta method (6.8) give for such a problem? What are the order conditions for this class of special problems?

Exercise 6.2. Show that there is a one-parameter class of explicit Runge-Kutta methods with $p = s = 2$. What will be the result of one step with these methods for the scalar linear differential equation $u'(t) = \lambda u(t)$? Show that order $p > 2$ is not possible if $s = 2$, by considering this scalar linear equation.

Exercise 6.3. Consider the method, with parameter $\theta \in [0, 1]$, given by the formula

$$u_n = u_{n-1} + (1 - \theta)\tau f(t_{n-1}, u_{n-1}) + \theta\tau f(t_n, u_n).$$

It is known as the θ -method. Special cases are Euler's method ($\theta = 0$) and the implicit trapezoidal rule ($\theta = \frac{1}{2}$). Insertion of exact solution values in this formula gives

$$u(t_n) = u(t_{n-1}) + (1 - \theta)\tau u'(t_{n-1}) + \theta\tau u'(t_n) + \tau\delta_n,$$

with a residual term $\tau\delta_n$.

(a) Show that this θ -method can be written in the form of a general Runge-Kutta method with $s = 2$.

(b) Due to the special form of the θ -method, convergence can be demonstrated in a rather direct way. For this, derive a Taylor expansion for the residuals δ_n , assuming $u(t)$ to be sufficiently smooth.

(c) Show convergence of the θ -method under assumption (6.2). What is the order of the method? (Hint: find L_* such that $(1 - \theta\tau L)^{-1}(1 + (1 - \theta)\tau L) \leq 1 + \tau L_*$ for $\theta\tau L \leq \frac{1}{2}$.)

Exercise 6.4. Consider a solution $u(t)$ of non-autonomous problem (6.1) in \mathbb{R}^m , i.e., with the function $f(t, v)$ depending explicitly on t .

(a) Let $\bar{u}(t) = (t, u(t)^T)^T$. Show that \bar{u} is the solution of an autonomous initial value problem $\bar{u}'(t) = \bar{f}(\bar{u}(t))$, $\bar{u}(0) = \bar{u}_0$ in \mathbb{R}^{m+1} .

(b) Show that any Runge-Kutta method of order $p \geq 1$, satisfying (6.9), produces the same results for these two formulations of the problem.

Exercise 6.5 (programming). Write a program for the problem (6.7) with the Euler method (6.4), the explicit trapezoidal rule (6.6) and the fourth-order Runge-Kutta method (6.15.b), using fixed step-sizes $\tau = T/N$. When plotting the results in the phase-plane with Euler's method for moderate values of N you will see that the solutions spiral outward, instead of staying close to the periodic exact solution. Can you explain this behaviour by looking at the vector field for this differential equation in the phase-plane?

Exercise 6.6 (programming). The program with variable step-sizes made for Figure 6.2 can be found on the web-site for this course. Modify this program by using method (6.15.a) with $p = s = 3$. For this, find an embedded method with coefficients a_{ij}, \hat{b}_i, c_i and $\hat{p} = 2$, by taking, for example, $\hat{b}_2 = \hat{b}_3$.

7 Stiff Initial Value Problems

In practice, initial value problems (6.1) with high dimension m appear very often. Usually such problems are stiff. In this section the concept of stiffness will be discussed. As we will see, for stiff problems numerical methods should have suitable stability properties, and implicit Runge-Kutta methods (6.8) then become important.

7.1 Explicit and Implicit Euler Method for Stiff Problems

Euler's method (6.4) is often called the *forward* or *explicit Euler method*. This to distinguish it from the *backward* or *implicit Euler method*, which reads

$$(7.1) \quad u_n = u_{n-1} + \tau f(t_n, u_n).$$

This is also a method of order one, so it seems to have no advantage over its forward counterpart (6.4). There is an obvious disadvantage: being implicit, it takes more work per step. Nevertheless, there is an important class of initial value problems where (7.1) is to be preferred over (6.4).

Example 7.1 As an illustration, consider the simple linear problem

$$(7.2) \quad \begin{aligned} v'(t) &= \lambda_1 v(t) - \lambda_2 w(t), & v(0) &= v_0, \\ w'(t) &= \lambda_2 w(t), & w(0) &= w_0, \end{aligned}$$

with time interval $t \in [0, 1]$, $v_0 = 1$, $w_0 = 0.1$ and $\lambda_1 = -1$, $\lambda_2 = -(1 + \kappa)$, $\kappa \gg 1$. The differential equation is of the form $u'(t) = Au(t)$ with a 2×2 matrix A and λ_1, λ_2 are the eigenvalues of A . The exact solution is given by

$$(7.3) \quad v(t) = e^{\lambda_1 t} \left(v_0 + \frac{\lambda_2}{\lambda_2 - \lambda_1} w_0 \right) - \frac{\lambda_2}{\lambda_2 - \lambda_1} e^{\lambda_2 t} w_0, \quad w(t) = e^{\lambda_2 t} w_0.$$

Since $\lambda_2 \ll -1$, the terms with $e^{\lambda_2 t}$ become negligible after a short while, but we will see that having $\lambda_2 \ll -1$ will cause difficulties for explicit methods.

In Figure 7.1 the numerical approximations v_n for the first solution component are plotted versus t_n for the explicit Euler method with step-size $\tau = \frac{1}{50}$ and with increasing κ . It is clear from these pictures that the explicit Euler method cannot be used with this step-size if κ gets large.

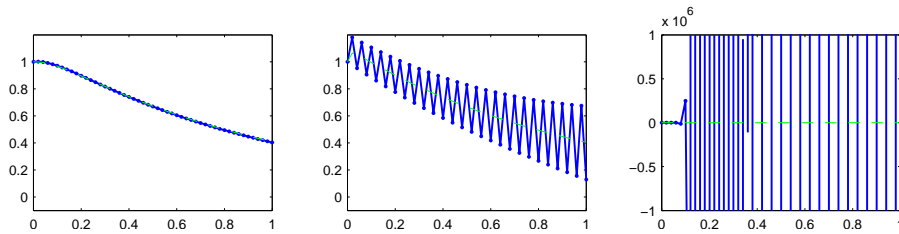


Figure 7.1: Results v_n for the explicit Euler method with $\tau = 1/50$ and $\kappa = 10$ (left), $\kappa = 100$ (middle), $\kappa = 1000$ (right). The exact solution is indicated with dashed lines.

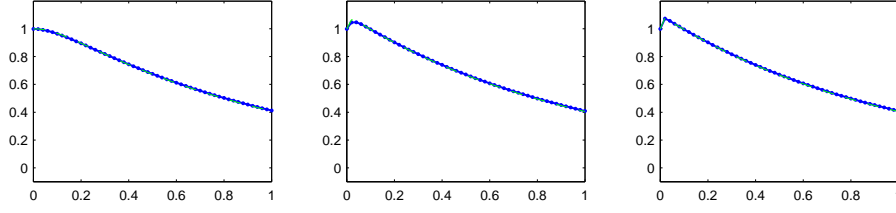


Figure 7.2: Results v_n for the implicit Euler method with $\tau = 1/50$ and $\kappa = 10$ (left), $\kappa = 100$ (middle), $\kappa = 1000$ (right). The exact solution is indicated by the dashed lines.

The behaviour of the implicit Euler method is very different. Figure 7.2 contains the same plots, again with increasing κ , but here the numerical approximations stay close to the exact solution no matter how large κ becomes.

We note that the solution is smooth after a very short while, and the solution is also not very sensitive for perturbations in the initial values v_0 and w_0 . On the other hand, for the Lipschitz constant it can be shown that $L \geq \max_j |\lambda_j|$, so L becomes large for increasing κ . \diamond

To understand the different behaviour of the explicit and implicit Euler method observed in this example, let us consider a linear problem

$$(7.4) \quad u'(t) = A u(t), \quad u(0) = u_0,$$

with constant matrix $A \in \mathbb{R}^{m \times m}$. The Lipschitz constant (6.2) for this problem equals $L = \|A\|$. Assume the matrix A is diagonalizable,

$$(7.5) \quad A = V \Lambda V^{-1}, \quad \Lambda = \text{diag}(\lambda_i).$$

Here the $\lambda_i \in \mathbb{C}$ are the eigenvalues of A and the i -th column of $V \in \mathbb{C}^{m \times m}$ contains the corresponding eigenvector. Then, for $w(t) = V^{-1}u(t)$ we have $w'(t) = \Lambda w(t)$, and hence $w(t) = \exp(t\Lambda)w(0)$ where $\exp(t\Lambda) = \text{diag}(\exp(t\lambda_i))$. In terms of our original u we thus get

$$(7.6) \quad u(t) = V \exp(t\Lambda) V^{-1} u_0.$$

It is clear that $u(t)$ stays bounded for all $t > 0$ with arbitrary initial value $u_0 \in \mathbb{R}^m$ if and only if

$$(7.7) \quad \text{Re } \lambda_i \leq 0 \quad \text{for all } 1 \leq i \leq m.$$

Application of the explicit Euler method to (7.4) gives a different picture. We then have $u_n = (I + \tau A)u_{n-1}$, and since $I + \tau A = V(I + \tau \Lambda)V^{-1}$ this leads to

$$(7.8) \quad u_n = V(I + \tau \Lambda)^n V^{-1} u_0.$$

To have boundedness of u_n uniformly for $n \geq 0$ with arbitrary initial value $u_0 \in \mathbb{R}^m$ we now obtain the condition

$$(7.9) \quad |1 + \tau \lambda_i| \leq 1 \quad \text{for all } 1 \leq i \leq m.$$

On the other hand, for the implicit Euler method we get $u_n = (I - \tau A)^{-1} u_{n-1}$, which gives

$$(7.10) \quad u_n = V(I - \tau \Lambda)^{-n} V^{-1} u_0.$$

To have boundedness of u_n uniformly for $n \geq 0$ with arbitrary initial value $u_0 \in \mathbb{R}^m$ we now have the condition

$$(7.11) \quad |1 - \tau \lambda_i|^{-1} \leq 1 \quad \text{for all } 1 \leq i \leq m.$$

Returning to our example, the matrix A for (7.2) has eigenvalues $\lambda_1 = -1$ and $\lambda_2 = -L$, where $L = 1 + \kappa$ is the Lipschitz constant in the maximum norm. Therefore, the boundedness condition (7.9) for the explicit Euler method reads $\tau L \leq 2$, and if this is not satisfied we can get exponential growth of the solutions, $\|u_n\| \sim \rho^n$, with a growth factor $\rho = |\tau L - 1|$. In Figure 7.1 this is $\rho = 1.02$ for the middle panel, and $\rho \approx 19$ for the right panel. So, even though the large negative eigenvalue λ_2 hardly influences the solution, it causes numerical difficulties with the explicit Euler method. In contrast to this, for the implicit Euler method the boundedness condition (7.9) is clearly satisfied for any step-size $\tau > 0$.

The above considerations can also be applied to study the error propagation in the numerical methods; this will be discussed below in Section 7.3.

Stiff problems. Consider a general initial value problem $u'(t) = f(t, u(t))$, $u(0) = u_0$, on a time interval $[0, T]$. Along with the solution u starting from u_0 we also consider a perturbed initial value \tilde{u}_0 with corresponding solution \tilde{u} . The sensitivity of u with respect to this initial perturbation is measured by

$$(7.12) \quad M = \max_{t \in [0, T]} \frac{\|u(t) - \tilde{u}(t)\|}{\|u_0 - \tilde{u}_0\|}.$$

If f is continuously differentiable with Lipschitz constant L , we know that the explicit Euler approximations u_n, \tilde{u}_n will converge to $u(t_n)$ and $\tilde{u}(t_n)$, respectively, as $\tau \rightarrow 0$, and also $\|u_n - \tilde{u}_n\| \leq (1 + \tau L)^n \|u_0 - \tilde{u}_0\|$. It follows that we always have the upper bound $M \leq \exp(LT)$, but M may be much smaller.

There are many problems with a smooth solution and moderate sensitivity factors M for arbitrary initial perturbations, but with a large value LT . Such problems are called *stiff*. Problem (7.1) is an example.

Stiffness is not a precisely defined mathematical concept, because no quantification is given for ‘large’ or ‘moderate’. Instead it is an operational concept, indicating the class of problems for which implicit methods can perform (much) better than explicit methods. For any explicit method a moderate value LT is needed to have a favourable error growth in the numerical procedure, whereas with a suitable implicit method this numerical error growth will in general not depend on LT but on the moderate quantity M .

A large percentage of initial value problems arising in practice is stiff, in particular for problems with large dimension m . In general, large problems contain many different time scales. A process with short time scale can be associated with

$e^{\lambda t}$ where $\operatorname{Re} \lambda \ll 0$. Because the short time scale effects quickly disappear, the overall behaviour of the solution is determined by the long time scales. However, when we apply a method like explicit Euler to such a problem, the step-size has to be adjusted to these short time scales, even after their effect has died out in the exact solution, because otherwise errors will be greatly amplified in each step.

7.2 Stability Regions and Implicit Methods

To find methods suitable for stiff problems we will first look again at the simple test equation

$$(7.13) \quad u'(t) = \lambda u(t)$$

with $\lambda \in \mathbb{C}$. Application of a Runge-Kutta method to the test equation will give

$$(7.14) \quad u_n = R(\tau\lambda) u_{n-1}$$

with a function R determined by the coefficients of the method. This function R is called the *stability function* of the method. As we will see shortly, for explicit methods it is a polynomial. For implicit methods it will be a rational function. The set

$$(7.15) \quad \mathcal{S} = \{z \in \mathbb{C} : |R(z)| \leq 1\}$$

is called the region of absolute stability or simply the *stability region* of the method. If the whole left half-plane $\{z \in \mathbb{C} : \operatorname{Re} z \leq 0\}$ is contained in \mathcal{S} , the method is called *A-stable*.

Some properties of stability functions are summarized in the following result.

Theorem 7.2 If the Runge-Kutta method has order $p \geq 1$, then

$$R(z) = e^z + \mathcal{O}(z^{p+1}) \quad \text{as } z \rightarrow 0.$$

Moreover, if the method is explicit with s stages, then R is a polynomial of degree $\leq s$, and the method cannot be *A-stable*.

Proof. Consider fixed λ and denote $z = \lambda\tau$. The local error in the first step is

$$\tau d_1 = u(t_1) - u_1 = (e^z - R(z)) u_0.$$

The first statement thus follows from the requirement $\|\tau d_1\| = \mathcal{O}(\tau^{p+1})$ as $\tau \rightarrow 0$.

The Runge-Kutta method applied to (7.13) reads

$$v_{n,i} = u_{n-1} + z \sum_{j=1}^s \alpha_{ij} v_{n,j}, \quad u_n = u_{n-1} + z \sum_{i=1}^s b_i v_{n,i}.$$

If the method is explicit it is seen by induction that $v_{n,i}$ equals u_{n-1} multiplied by a polynomial of degree $i-1$ in z , and therefore in the final stage u_n equals u_{n-1} multiplied by $R(z)$, with a polynomial R of degree s , at most. Since the method

has order $p \geq 1$, this R is not constant. Consequently $|R(z)| > 1$ when $|z|$ gets large, so the method cannot be A -stable. \square

Explicit methods. As we saw in Theorem 7.2, the stability function of an explicit s -stage method is a polynomial of degree s or less. It is therefore not possible to satisfy $R(z) = e^z + \mathcal{O}(z^{p+1})$ for $z \rightarrow 0$ with an order $p > s$. As a consequence, we have the following result:

Corollary 7.3 For an explicit Runge-Kutta method with s stages, the order p cannot be larger than s . \square

For explicit methods with $p = s$ the stability function is given by

$$(7.16) \quad R(z) = 1 + z + \frac{1}{2!}z^2 + \cdots + \frac{1}{s!}z^s.$$

We saw already in the previous section that having $p = s$ is possible for explicit methods with s up to four. The stability regions of these methods are displayed in Figure 7.3.

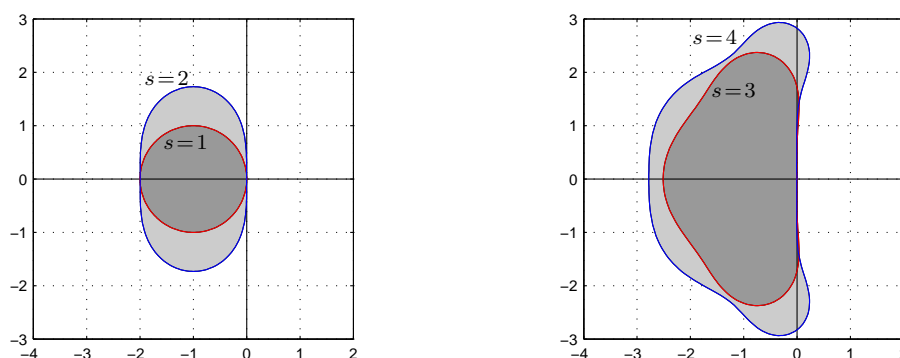


Figure 7.3: Stability regions \mathcal{S} for the stability functions (7.16) of degree $s = 1, 2, 3, 4$.

If we apply one of these explicit method to (7.13) with $\operatorname{Re}\lambda < 0$, then the step-size τ should be chosen such that $\tau\lambda \in \mathcal{S}$. Otherwise the numerical approximations u_n will exhibit an exponential growth (due to $|R(\tau\lambda)| > 1$) whereas the exact solution values $u(t_n)$ tend to zero for increasing n . If the modulus of λ is very large, the requirement $\tau\lambda \in \mathcal{S}$ will lead to a very small step-size τ , and this will make the method inefficient. The same happens for linear and nonlinear systems of ODEs (with λ an eigenvalue of A , and A a Jacobian matrix of f). We will therefore look at implicit methods, for which such severe step-size restrictions can be avoided.

Implicit methods. Implicit methods can be A -stable. Examples are the implicit Euler method (7.1) with

$$(7.17) \quad R(z) = \frac{1}{1 - z},$$

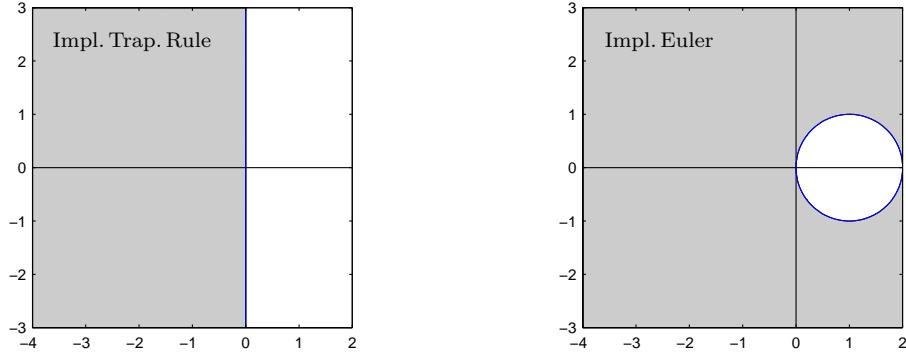


Figure 7.4: Stability regions \mathcal{S} (shaded areas) for the implicit trapezoidal rule (left) and the implicit Euler method (right).

and the implicit trapezoidal rule (6.5) with

$$(7.18) \quad R(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}.$$

The corresponding stability regions are shown in Figure 7.4. Note that the stability region of the implicit trapezoidal rule is precisely the left-half plane which nicely agrees with the set $z \in \mathbb{C}$ for which $|e^z| \leq 1$. On the other hand, for the exponential function we also have $|e^z| \rightarrow 0$ if $z \rightarrow -\infty$. This property is mimicked by the implicit Euler method but not by the trapezoidal rule.

Another simple implicit method is *implicit midpoint rule*, given by

$$(7.19) \quad u_n = u_{n-1} + \tau f(t_{n-\frac{1}{2}}, \frac{1}{2}u_{n-1} + \frac{1}{2}u_n).$$

Its stability function is (7.18), the same as for the implicit trapezoidal rule. This implicit midpoint rule is the first member of a larger class of methods, the Gauss methods, which will be derived below using the idea of collocation.

Collocation methods.* Several interesting classes of implicit methods can be obtained by using *collocation*, where we are looking for a polynomial of degree s such that the differential equation is satisfied at s discrete points. This is closely related to numerical quadrature and polynomial interpolation.

The collocation method is described here for the first step, taking us from $t_0 = 0$ to $t_1 = \tau$. For given distinct nodes c_1, c_2, \dots, c_s , the collocation polynomial v of degree s is defined by

$$(7.20) \quad v(0) = u_0, \quad v'(c_i\tau) = f(c_i\tau, v(c_i\tau)) \quad (i = 1, \dots, s),$$

and the new approximation $u_1 \approx u(t_1)$ is then taken as $u_1 = v(\tau)$. This applies to systems as well as for scalar problems. (For the arguments in the following proof it is easiest to think first about scalar problems.)

Proposition 7.4 Let $L_i(t) = \prod_{j \neq i} (t - c_j) / (c_i - c_j)$ for $i = 1, \dots, s$. Then the collocation method given by (7.20) with $u_1 = v(\tau)$ is equivalent to a Runge-Kutta method (6.8) with coefficients

$$a_{ij} = \int_0^{c_i} L_j(t) dt, \quad b_j = \int_0^1 L_j(t) dt.$$

Proof. Setting $f_i = f(c_i\tau, v(c_i\tau)) = v'(c_i\tau)$, we have according to the Lagrange interpolation formula

$$v'(\theta\tau) = \sum_{j=1}^s L_j(\theta) f_j.$$

Using $v(0) = u_0$, integration thus gives

$$v(c_i\tau) = u_0 + \tau \int_0^{c_i} v'(\theta\tau) d\theta = u_0 + \tau \sum_{j=1}^s \left(\int_0^{c_i} L_j(\theta) d\theta \right) f_j,$$

which gives the Runge-Kutta formula (6.8) with $v_i = v(c_i\tau)$. \square

If the nodes c_1, \dots, c_s are chosen as in the Gauss quadrature rule (zeros of shifted Legendre polynomials), then the resulting Runge-Kutta methods are called *Gauss methods*, or *Gauss-Legendre methods*. These methods turn out to have order $2s$ (not proven here), and they are A -stable, as will be shown next.

Proposition 7.5 For any number of stages s , the Gauss method is A -stable.

Proof. In this proof we will use the fact that Gauss quadrature is exact for polynomials of degree $2s - 1$ or less. It follows that all weights b_i in Gauss quadrature are positive, because

$$b_i = \sum_{j=1}^s b_j L_i(c_j)^2 = \int_0^1 L_i(\theta)^2 d\theta > 0.$$

Application of the collocation method to our complex test problem $u'(t) = \lambda u(t)$ with $\text{Re}\lambda \leq 0$, leads for $\mu(t) = |v(t)|^2 = \overline{v(t)} \cdot v(t)$ to

$$\mu'(c_i\tau) = 2 \text{Re} \left(\overline{v(c_i\tau)} v'(c_i\tau) \right) = 2 \text{Re}\lambda \cdot \mu(c_i\tau) \leq 0.$$

But since μ' is a polynomial of degree $2s - 1$ at most, it follows that

$$\mu(\tau) = \mu(0) + \int_0^\tau \mu'(t) dt = \mu(0) + \tau \sum_{i=1}^m b_i \mu'(c_i\tau) \leq \mu(0),$$

showing that $|u_1| \leq |u_0|$. \square

Relevance of the test equation. We started out in (6.1) with general initial value problems, probably of high dimension, nonlinear and with difficult solutions. But the concepts of stability region and A -stability are related to the simple scalar test equation $u'(t) = \lambda u(t)$, for which we know already the solution $u(t) = e^{\lambda t} u(0)$.

It turns out that the linear scalar stability concepts are very relevant for general nonlinear systems. If we want to know how numerical solutions for a general ODE system $u'(t) = f(t, u(t))$ react to perturbations, a first step is to consider the local influence of small perturbations. But that can be studied for a linearized

problem $u'(t) = Au(t)$. Then if A is diagonalizable we are led to the test problem $u'(t) = \lambda u(t)$, with λ eigenvalues of A .

There many theoretical results in the opposite direction: starting from assumptions on the numerical method for the test equation (e.g. A -stability) one can show stability and convergence results for interesting classes of linear and nonlinear systems. For such results we refer to the specialized books given at the end of these notes.

7.3 Example: the θ -Method for Stiff Problems

To discuss some important issues for stiff problems, we will consider here a simple class of methods. For the general class of Runge-Kutta methods such a discussion would become too technical.

In this section we consider the following methods, with parameter $\theta \in [0, 1]$,

$$(7.21) \quad u_n = u_{n-1} + (1 - \theta)\tau f(t_{n-1}, u_{n-1}) + \theta\tau f(t_n, u_n).$$

This contains the explicit Euler method ($\theta = 0$), the implicit Euler method ($\theta = 1$) and the implicit trapezoidal rule ($\theta = \frac{1}{2}$) as special cases. With unspecified θ , the method (7.21) is simply known as the ' θ -method'. The stability function of the θ -method is $R(z) = (1 - \theta z)^{-1}(1 + (1 - \theta)z)$ and the method is A -stable for $\theta \geq \frac{1}{2}$ (see Exercise 7.3).

Implementation aspects. To discuss implementation of the implicit methods, with $\theta > 0$, we can consider autonomous problems, where $f(t, v) = f(v)$. Then, in the step (7.21), starting from a known u_{n-1} , the vector $u_n \in \mathbb{R}^m$ is the solution of the system $w = g(w)$ with

$$g(w) = u_{n-1} + (1 - \theta)\tau f(u_{n-1}) + \theta\tau f(w).$$

If f satisfies the Lipschitz condition (6.2), then we have

$$\|g(\tilde{w}) - g(w)\| \leq \theta\tau L \cdot \|\tilde{w} - w\|.$$

Therefore the system could be solved by fixed point iteration provided that $\theta\tau L < 1$. However, we want to use the implicit methods for stiff problems with $\tau L \gg 1$. This can be done, but instead of fixed point iteration we need a Newton type iteration. Note that u_{n-1} will in general already be a reasonably good first approximation to u_n . It is common practice to apply a modified Newton iteration where the Jacobian matrix is held fixed during the iteration. Assuming we have a tolerance Tol used for step-size selection, a simple computational scheme for the n -th step becomes:

$$\left\{ \begin{array}{l} \text{Compute } A_n = f'(u_{n-1}), \text{ set } w^0 = u_{n-1}. \text{ Then iterate, for } k = 1, 2, \dots, \\ w^k = w^{k-1} - (I - \theta\tau A_n)^{-1} (w^{k-1} - u_{n-1} - (1 - \theta)\tau f(u_{n-1}) - \theta\tau f(w^{k-1})), \\ \text{until the displacement } \|w^k - w^{k-1}\| \text{ is less than a fraction (say } \frac{1}{10}) \text{ of } Tol. \end{array} \right.$$

Here the stopping criterion aims at solving the system of equations with an error less than the (estimated) local error due to discretization. There are many variants on this simple scheme.

On top of this, one has to decide how the *linear algebraic equations* with matrix $I - \theta\tau A_n$ are to be solved in each Newton iteration. For ‘small’ problems this can be done with *LU*-decomposition, but for ‘big’ problems this is often done iteratively. So then we get another iteration, within the Newton iteration! It will be clear that writing a good code for (a class of) stiff problems can be a challenging task.

Convergence and error propagation for linear problems. In the above we only looked at the behaviour of the modulus or norm of the solutions, but the same can be done for differences of solutions with perturbations. Along with the (unperturbed) θ -method (7.21) we can consider

$$(7.22) \quad \tilde{u}_n = \tilde{u}_{n-1} + (1 - \theta)\tau f(t_{n-1}, \tilde{u}_{n-1}) + \theta\tau f(t_n, \tilde{u}_n) + \tau\delta_n.$$

These residuals $\tau\delta_n$ may arise from various error sources, for instance round-off errors or errors due to not solving the implicit relations exactly. We can also describe discretization errors in this way, by putting $\tilde{u}_n = u(t_n)$, leading to convergence results similar to Exercise 6.3 but now without Lipschitz conditions on f . This will be described here for linear problems.

Consider the θ -method applied to a linear inhomogeneous system

$$u'(t) = Au(t) + g(t),$$

with diagonalizable matrix $A = V\Lambda V^{-1} \in \mathbb{R}^{m \times m}$, $\Lambda = \text{diag}(\lambda_k)$, such that $\text{Re } \lambda_k \leq 0$ for all $1 \leq k \leq m$. If no bound for the moduli $|\lambda_k|$ is specified this problem can be arbitrarily stiff, but for the *A*-stable methods, $\theta \geq \frac{1}{2}$, we know that $|R(\tau\lambda_k)| \leq 1$ for all k and $\tau > 0$. As in (7.8), this gives

$$(7.23) \quad \|R(\tau A)^n\| \leq \kappa, \quad \|R(\tau A)^{n-j}(I - \theta\tau A)^{-1}\| \leq \kappa,$$

where $\kappa = \|V\|\|V^{-1}\|$, and $R(Z) = (I + (1 - \theta)Z)^{-1}(I - \theta Z)$ is the stability function with matrix argument $Z \in \mathbb{R}^{m \times m}$.

Let $\varepsilon_n = \tilde{u}_n - u_n$. Then subtraction of (7.21) from (7.22) gives

$$(7.24) \quad \varepsilon_n = R(\tau A)\varepsilon_{n-1} + (I - \theta\tau A)^{-1}\tau\delta_n.$$

Recursive application of this relation leads to

$$(7.25) \quad \varepsilon_n = R(\tau A)^n\varepsilon_0 + \sum_{j=1}^n R(\tau A)^{n-j}(I - \theta\tau A)^{-1}\tau\delta_j.$$

If $\theta \geq \frac{1}{2}$, then (7.23) gives

$$(7.26) \quad \|\varepsilon_n\| \leq \kappa\|\varepsilon_0\| + \kappa\tau \sum_{j=1}^n \|\delta_j\|.$$

If we apply this with $\tilde{u}_n = u(t_n)$, a convergence result is obtained for $\theta \geq \frac{1}{2}$, where the upper bound for the global errors will depend only on the condition number κ , on t_n and bounds for the derivatives of the exact solution, but *not* on the Lipschitz condition or the size of the moduli of the eigenvalues of A . Compare this with Exercise 6.3.

For the explicit Euler method, on the other hand, we *do* need a bound on the $|\lambda_k|$, and the stepsize τ should be such that $|1 + \tau\lambda_k| \leq 1$ for all eigenvalues λ_k . Otherwise all errors made during the process can be amplified with a large constant, leading to a fast exponential growth of errors.

7.4 A Semi-discrete Initial-Boundary Value Problem

Many stiff initial value problems for ODEs have their origin in partial differential equations (PDEs). As an illustration we consider the ODE system

$$(7.27) \quad u'_j(t) = \frac{\gamma}{h^2} (u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)) + \kappa u_j(t) (1 - u_j(t)),$$

with component index $j = 1, 2, \dots, m$, and with $u_0(t) = u_{m+1}(t) = 0$. Here $\gamma = \frac{1}{10}$, $\kappa = 10$ and $h = \frac{1}{m+1}$. Moreover, initial values $u_j(0)$ will be prescribed for all components.

This problem is obtained from the partial differential equation

$$(7.28) \quad \frac{\partial}{\partial t} w(x, t) = \gamma \frac{\partial^2}{\partial x^2} w(x, t) + \kappa w(x, t) (1 - w(x, t)),$$

where $w(x, t)$ stands for a concentration of a biological species that varies over space and time. This combines the simple model $w' = \kappa w(1 - w)$ for population growth with spatial diffusion. If $w(x, 0) \in (0, 1)$ in a given point x at time $t = 0$, then the ODE part will initiate a growth towards $w = 1$, but at the same time the diffusion will cause a flow from regions with high w towards regions with lower w . We consider equation (7.28) for $t \in [0, T]$ and $0 < x < 1$. Together with the initial condition $w(x, 0) = w_0(x)$, we also impose the boundary conditions $w(0, t) = 0$, $w(1, t) = 0$,³ giving an *initial-boundary value problem* for a PDE.

Now suppose we impose a spatial grid $x_j = jh$, $j = 1, \dots, m$, with h the mesh-width in space, and use the approximate spatial derivatives

$$\frac{\partial^2}{\partial x^2} w(x, t) \approx \frac{1}{h^2} (w(x - h, t) - 2w(x, t) + w(x + h, t))$$

at the grid points x_j . Then we obtain the ODE system (7.27) where the components $u_j(t)$ approximate the PDE solution at the grid points, $u_j(t) \approx w(x_j, t)$. It can be shown (beyond scope of these notes) that the error in this approximation will be $\mathcal{O}(h^2)$ provided w is smooth. So we will take m large, giving $h > 0$ small.

Figure 7.5 shows the solution of (7.27) at various times, starting with the initial profile $u_j(0) = w(x_j, 0) = \frac{1}{10} \exp(-100(x_j - \frac{1}{4})^2) + \frac{1}{4} \exp(-100(x_j - \frac{3}{4})^2)$ and $m = 100$.

³Actually, these boundary conditions are not very realistic for a biological model, but they are taken here for simplicity.

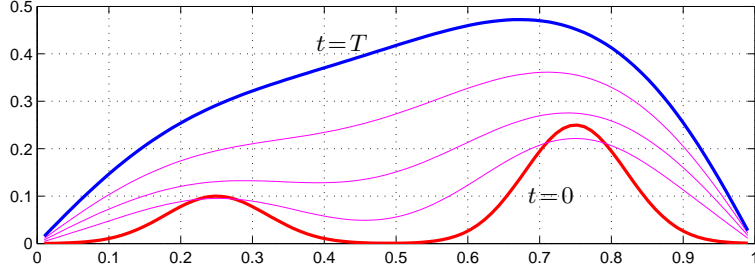


Figure 7.5: Time evolution for (7.27) with $m = 100$: solutions versus x at the start $t = 0$ and final time $t = T = \frac{1}{4}$ (fat lines), and intermediate $t = \frac{1}{16}, \frac{2}{16}, \frac{3}{16}$ (thin lines).

The resulting ODE system is often called a *semi-discrete system* because space has been discretized but time is still continuous. We can write the system in vector form as $u'(t) = Au(t) + g(u(t))$, with matrix $A \in \mathbb{R}^{m \times m}$ and nonlinear function g defined by

$$(7.29) \quad A = \frac{\gamma}{h^2} \begin{pmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}, \quad g(v) = \kappa \begin{pmatrix} v_1(1-v_1) \\ v_2(1-v_2) \\ \vdots \\ v_m(1-v_m) \end{pmatrix}$$

for $v = (v_j) \in \mathbb{R}^m$. We already saw (Exercise 2.5) that this tri-diagonal matrix A is negative definite. Therefore, $A = V\Lambda V^{-1}$ where V is orthogonal and $\Lambda = \text{diag}(\lambda_i)$ contains the eigenvalues $\lambda_i < 0$. The eigenvalues of A are known to be in the interval $[-4\gamma/h^2, 0]$, with the most negative ones close to $-4\gamma/h^2$.

This initial value problem with large m can be efficiently solved with suitable implicit methods. Then the step-size needs only to be adjusted to the smoothness to keep the local errors small. On the other hand, when using an explicit method like the Euler's method (6.4) or the explicit trapezoidal rule (6.6), very small step-sizes are necessary if the mesh-width h becomes small. The requirement that the segment $[-4\gamma/h^2, 0]$ of the negative real axis fits in the stability regions (left panel of Figure 7.3) leads for these explicit methods to the time step restriction

$$(7.30) \quad \frac{\gamma\tau}{h^2} \leq \frac{1}{2}.$$

The nonlinear term $g(u)$ only contains moderate contributions, and therefore the stability restriction (7.30) will give a fair estimate of the possible time step-sizes. To find accurate approximations to the PDE the mesh-width h must be small, and therefore the explicit methods are not suited for this problem.

7.5 Exercises

Exercise 7.1. Show that the implicit midpoint rule (7.19) can be written as a Runge-Kutta method:

$$\begin{aligned}u_{n-\frac{1}{2}} &= u_{n-1} + \frac{1}{2}\tau f(t_{n-\frac{1}{2}}, u_{n-\frac{1}{2}}), \\u_n &= u_{n-1} + \tau f(t_{n-\frac{1}{2}}, u_{n-\frac{1}{2}}).\end{aligned}$$

Show that the collocation procedure (7.20) with $s = 1$ and $c_1 = \frac{1}{2}$ produces this method.

Exercise 7.2. Which Runge-Kutta method is produced by the collocation procedure (7.20) with $s = 2$ and $c_1 = 0, c_2 = 1$?

Exercise 7.3. Consider the θ -method (7.21) with parameter $\theta \geq 0$. This method was also studied in Exercise 6.3.

- (a) Determine the stability region of the method. For which values of θ is the method A -stable?
- (b) If we apply such an A -stable method to the linear system $u'(t) = Au(t)$ with matrix A given by (7.29), with orthogonal V and all eigenvalues real and negative, show that then $\|u_n\|_2 \leq \|u_{n-1}\|_2$ in the Euclidian norm. (Remember that for an orthogonal V we have $\|Vu\|_2 = \|u\|_2$.)

Exercise 7.4. Consider an autonomous differential equation $u'(t) = f(u(t))$. Show that if we apply only one Newton iteration to the θ -method (7.21) with initial iteration guess u_{n-1} and Jacobian $A_n \approx f'(u_{n-1})$, the result will be

$$(7.31) \quad u_n = u_{n-1} + (I - \theta\tau A_n)^{-1}\tau f(u_{n-1}).$$

Determine the order of consistency of the method with $A_n = f'(u_{n-1})$, by inserting exact solution values as in (6.11), (6.12). Determine the stability region of this linearly implicit method.

Exercise 7.5 (programming). The above linearly implicit method (7.31) with $\theta = \frac{1}{2}$ was used to produce Figure 7.5. The program can be found on the web-site for this course.

- (a) Modify this program such that the explicit Euler method will be used as time stepping method. Verify the stability bound (7.30) experimentally.
- (b) Do the same for the explicit trapezoidal rule (6.6).

8 Two-Point Boundary Value Problems

In this section we will study boundary value problems for ordinary differential equations, also known as two-point boundary value problems. First we will briefly outline the use of initial value problem techniques via the "shooting" method. Although this procedure can be formulated for general two-point boundary value problems, the actual implementation is not straightforward, and theory is complicated.

In the main part of this section we will therefore consider a more restricted class of problems, the so-called Sturm-Liouville problems. For such problems we will study Galerkin methods, leading to simple *finite element* methods. These methods are very important because of the possible extensions to boundary value problems for partial differential equations.

8.1 Shooting Methods

A general form of a two-point boundary value problem on an interval $[a, b]$ is

$$(8.1) \quad u'(t) = f(t, u(t)) \ , \quad \varphi(u(a), u(b)) = 0 \ ,$$

with given functions $f : [a, b] \times \mathbb{R}^m \rightarrow \mathbb{R}^m$ and $\varphi : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}^m$. We will consider arbitrary $m \geq 1$, but to make the essential points of the following procedure more transparent it can first be assumed that $m = 1$.

Along with the boundary value problem (8.1), we also consider the initial value problem

$$(8.2) \quad v'(t) = f(t, v(t)) \ , \quad v(a) = \xi \ ,$$

where $\xi \in \mathbb{R}^m$ is still undetermined. To make the explicit dependence on ξ more clear, we will denote the solution of this initial value problem as $v(t) = v(t; \xi)$. Now, if we define $F(\xi) = \varphi(\xi, v(b; \xi))$, then it is seen that u is a solution of the boundary value problem iff

$$(8.3) \quad u(t) = v(t; \xi_*) \ , \quad F(\xi_*) = 0 \ .$$

This means that we have reduced the problem to a system of equations $F(\xi) = 0$ in \mathbb{R}^m , where the function values $F(\xi)$ can be computed by solving numerically the initial value problem (8.2).

One can try to solve this system of equations by a functional iteration, for instance $\xi_{k+1} = \xi_k - F(\xi_k)$ for $k = 0, 1, 2, \dots$, starting from a guess ξ_0 . However, this will only work for restricted classes of boundary value problems, because $G(\xi) = \xi - F(\xi)$ then needs to be a contraction.

Better convergence of the sequence ξ_k is obtained in general by a Newton iteration $\xi_{k+1} = \xi_k - (F'(\xi_k))^{-1}F(\xi_k)$. But then we need to be able to compute not only the function values $F(\xi)$ but also the derivative $F'(\xi)$.

This is possible. Introducing the $m \times m$ Jacobian matrices of partial derivatives

$$A(t, v) = \frac{\partial f(t, v)}{\partial v} \ , \quad B_1(\xi, v) = \frac{\partial \varphi(\xi, v)}{\partial \xi} \ , \quad B_2(\xi, v) = \frac{\partial \varphi(\xi, v)}{\partial v} \ , \quad V(t; \xi) = \frac{\partial v(t; \xi)}{\partial \xi} \ ,$$

we obtain by the chain rule

$$(8.4) \quad F'(\xi) = B_1(\xi, v(b; \xi)) + B_2(\xi, v(b; \xi)) \cdot V(b; \xi),$$

and finding $V(t) = V(t; \xi)$ amounts to solving (numerically)

$$(8.5) \quad V'(t) = A(t, v(t; \xi)) \cdot V(t), \quad V(a) = I.$$

This differential equation for the matrix $V(t) \in \mathbb{R}^{m \times m}$ is equivalent to m differential equations in \mathbb{R}^m for the columns of $V(t)$.

The above procedure is called *shooting*: with the shooting parameter ξ we aim at the 'target' $\varphi(\xi, v(b; \xi)) = 0$. Turning these ideas into a working computer code requires resolving a number of technical issues. Theoretical statements on convergence are quite difficult in general. That is not surprising since statements about existence and uniqueness of solutions of the boundary value problem (8.1) are already much more complicated than for initial value problems.

8.2 Sturm-Liouville Problems and Weak Forms

In practice, boundary value problems often appear for scalar second-order differential equations $w'' = g(t, w, w')$ with boundary conditions $w(a) = \alpha$, $w(b) = \beta$. By rewriting this differential equation to a system of two first-order differential equations, it is seen that we are still formally in the framework of (8.1). In such second-order problems the independent variable is often a space coordinate, and it will therefore be denoted by x instead of t .

In the remainder of this section we will restrict ourselves to two-point boundary value problems of the form

$$(8.6a) \quad -(p(x) w'(x))' + q(x) w(x) = r(x),$$

$$(8.6b) \quad w(a) = 0, \quad w(b) = 0,$$

where $p, q, r : [a, b] \rightarrow \mathbb{R}$ are given, q and r are continuous and p is continuously differentiable. Linear boundary value problems of this type are called *Sturm-Liouville problems*. We call w a solution if it is twice differentiable, (8.6a) is satisfied for all $x \in [a, b]$, and $w(a) = w(b) = 0$. It is known (theory of differential equations) that (8.6) has a unique solution under the assumption

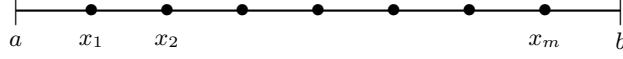
$$(8.7) \quad p(x) \geq p_0 > 0, \quad q(x) \geq 0 \quad (\text{for all } x \in [a, b]).$$

With p strictly positive, we can also write $w'' = \frac{1}{p}(qw - p'w' - r)$, and therefore the second derivative w'' of the solution will be continuous.

A finite difference method. Before going to some (mathematically) more advanced methods to solve (8.6), we first describe briefly a simple finite difference method on uniform grids. It will be based on the fact that if v is a smooth function, then we can approximate the derivative in a point x by a finite difference quotient:

$$v'(x) = \frac{1}{h} \left(v(x + \frac{1}{2}h) - v(x - \frac{1}{2}h) \right) + \mathcal{O}(h^2).$$

Using this, we can make a 'discrete version' of the boundary value problem (8.6). We will use a uniform grid on the interval $[a, b]$ consisting of the grid points $x_i = a + ih$ for $i = 0, 1, \dots, m+1$, where $h = (b - a)/(m + 1)$ is called the mesh width, or grid distance. This grid is called uniform because the distance between the points is the same, $x_{i+1} - x_i = h$.



Let $x_{i+1/2} = \frac{1}{2}(x_i + x_{i+1})$. Setting $v(x) = p(x)w'(x)$ we now first approximate

$$v'(x_i) \quad \text{by} \quad \frac{1}{h}(v(x_{i+1/2}) - v(x_{i-1/2})) \quad \text{for} \quad i = 1, 2, \dots, m,$$

and then we approximate the terms

$$v(x_{i+1/2}) \quad \text{by} \quad \frac{1}{h}p(x_{i+1/2})(w(x_{i+1}) - w(x_i)) \quad \text{for} \quad i = 0, 1, \dots, m.$$

In both of these approximation steps we will make $\mathcal{O}(h^2)$ errors. Applying these steps to (8.6) leads to a linear system for approximations $w_i \approx w(x_i)$. Let $p_{i\pm 1/2} = p(x_{i\pm 1/2})$ and $q_i = q(x_i)$, $r_i = r(x_i)$. The resulting tri-diagonal linear system for the approximations w_i is

$$(8.8) \quad \frac{1}{h^2} \left(-p_{i-1/2} w_{i-1} + (p_{i-1/2} + p_{i+1/2}) w_i - p_{i+1/2} w_{i+1} \right) + q_i w_i = r_i$$

for $i = 1, 2, \dots, m$, with $w_0 = w_{m+1} = 0$.

It can be shown, under suitable assumptions, that we will have an error $\max_i |w(x_i) - w_i| = \mathcal{O}(h^2)$. Related results can be found in Sect. 7.3 of the book of Gautschi (1997) listed in the preface of these notes.

This finite difference method, whereby (8.6) is replaced by (8.8), is conceptually easy. However, generalizations to non-uniform grids are not so easy. In practical applications non-uniform grids are often crucial to get an efficient numerical scheme, just as for quadrature problems and initial value problems. For this we will consider so-called finite element methods, which need some mathematical preparations.

Weak forms. Before introducing the numerical methods, we will first rewrite the problem (8.6) in a more abstract way with operators on function spaces. Let $C^k[a, b]$ stand for the set of real functions on $[a, b]$ that are k times continuously differentiable, and let $C_0^k[a, b] = \{v \in C^k[a, b] : v(a) = v(b) = 0\}$. The operator $L : C_0^2[a, b] \rightarrow C^0[a, b]$ is defined by

$$(Lw)(x) = - (p(x)w'(x))' + q(x)w(x).$$

Then (8.6) can be written compactly as: $Lw = r$ with $w \in C_0^2[a, b]$.

On any of these function spaces we can consider the standard inner-product and norm:

$$(u, v) = \int_a^b u v \, dx, \quad \|u\| = \sqrt{(u, u)},$$

where $\int u v dx$ stands for $\int u(x)v(x) dx$. It will also be convenient to introduce

$$[u, v] = \int_a^b (p u' v' + q u v) dx, \quad \|u\|_* = \sqrt{[u, u]}.$$

This bilinear form $[u, v]$ is defined for $u, v \in C^1[a, b]$. It is also an inner-product. The norm $\|u\|_*$ is often called the *energy norm* for our boundary value problem.

Let $v \in C_0^1[a, b]$. Multiplication of (8.6a) by $v(x)$ and integration by parts over $[a, b]$ gives

$$\int_a^b (-(p w')' v + q w v) dx = \int_a^b (p w' v' + q w v) dx = \int_a^b r v dx.$$

We therefore have the relation

$$(8.9) \quad (Lw, v) = [w, v] = (r, v) \quad \text{for any } v \in C_0^1[a, b].$$

If $w \in C_0^2[a, b]$ we can also follow these arguments backwards to arrive again at (8.6). So, at first sight, little seems to have been gained by this abstract reformulation. However, the bilinear form is also defined for a function w which is merely differentiable, and this will create the possibility to allow, for example, some discontinuities in the source function r .

More general, let $H^1[a, b]$ be the set of functions $v \in C^0[a, b]$ for which there is a finite partitioning $a = \theta_0 < \theta_1 < \dots < \theta_n = b$ such that the derivative v' is continuous and bounded on each sub-interval (θ_{j-1}, θ_j) . Then the integral over v' still exists:

$$\int_a^b v' dx = \sum_{j=1}^n \int_{\theta_{j-1}}^{\theta_j} v' dx.$$

We also define $H_0^1[a, b] = \{v \in H^1[a, b] : v(a) = v(b) = 0\}$.

Instead of (8.6) or (8.9), we will look for a function $w \in H_0^1[a, b]$ such that

$$(8.10) \quad [w, v] = (r, v) \quad \text{for any } v \in H_0^1[a, b].$$

This is called the *weak form* of the boundary value problem (8.6), and the function $w \in H_0^1[a, b]$ that satisfies (8.10) is called a *weak solution*.

Any solution w of (8.6) will also be a weak solution. Conversely, it can be shown that if w is a weak solution and $w \in C_0^2[a, b]$, then it will also be a solution of (8.6). As mentioned before, we can have weak solutions that are not in $C_0^2[a, b]$ when p', q or r are allowed to be discontinuous in some points. In the following we will retain our assumptions $q, r \in C^0[a, b]$ and $p \in C^1[a, b]$; the additional freedom offered by the weak form will be used in the construction of numerical methods.

Both bilinear forms (u, v) and $[u, v]$ are inner products on $H_0^1[a, b]$, and we therefore have the Cauchy-Schwarz inequalities

$$(8.11) \quad (u, v) \leq \|u\| \|v\|, \quad [u, v] \leq \|u\|_* \|v\|_* \quad (\text{for all } u, v \in H_0^1[a, b]),$$

see e.g. Exercise 8.2. To get some more insight in the energy norm, and its relation to the standard norm $\|\cdot\|$, the following lemma will be useful.

Lemma 8.1 Assume (8.7). Let $p_1 = \max_{x \in [a,b]} |p(x)|$, $q_1 = \max_{x \in [a,b]} |q(x)|$, and denote $\gamma_0 = \sqrt{p_0}$, $\gamma_1 = \sqrt{p_1 + (b-a)^2 q_1}$. Then

$$\frac{\gamma_0}{b-a} \|v\| \leq \gamma_0 \|v'\| \leq \|v\|_* \leq \gamma_1 \|v'\| \quad \text{for all } v \in H_0^1[a, b].$$

Proof. Since $v(x) = \int_a^x v'(y) dy$, we obtain by the Schwarz inequality

$$v(x)^2 = \left(\int_a^x 1 \cdot v'(y) dy \right)^2 \leq \left(\int_a^x (1)^2 dy \right) \cdot \left(\int_a^x (v')^2 dy \right) \leq (b-a) \|v'\|^2.$$

Hence

$$(8.12) \quad \|v\| \leq (b-a) \cdot \|v'\|.$$

Further we have $\|v\|_*^2 \geq \int_a^b p v' v' dx \geq p_0 \cdot \|v'\|^2$, from which the second inequality follows. Finally, it is easily seen that $\|v\|_*^2 \leq p_1 \|v'\|^2 + q_1 \|v\|^2$, which gives the third inequality. \square

Remark 8.2 Using $w'' = \frac{1}{p}(qw - p'w' - r)$, it follows from the previous lemma that $\|w''\| \leq \kappa_1 \|w\|_* + \kappa_2 \|r\|$ with constants $\kappa_1, \kappa_2 > 0$ determined by p, q . Since $\|w\|_*^2 = [w, w] = (r, w) \leq \|r\| \|w\|$, it also follows that

$$(8.13) \quad \|w''\| \leq \kappa_0 \|r\|$$

with constant $\kappa_0 > 0$ determined by the given functions p and q . \diamond

8.3 Galerkin Methods and Finite Elements

To obtain numerical approximations we will use the weak form (8.10) of the Sturm-Liouville problem (8.6). It is assumed throughout this section that (8.7) holds with $q, r \in C^0[a, b]$ and $p \in C^1[a, b]$. This ensures that the solution w is twice continuously differentiable.

Let V_h be a finite dimensional subspace of $H_0^1[a, b]$, with basis $\phi_1, \phi_2, \dots, \phi_m$. In the *Galerkin method* we look for an approximation $w_h \in V_h$ such that

$$(8.14) \quad [w_h, v_h] = (r, v_h) \quad \text{for any } v_h \in V_h.$$

The choice of V_h and its basis functions ϕ_j determines the numerical method. If we write our approximate solution in terms of these basis functions as $w_h = \sum_{j=1}^m \xi_j \phi_j$, that is,

$$w_h(x) = \sum_{j=1}^m \xi_j \phi_j(x),$$

then (8.14) is equivalent to finding $\xi = (\xi_j) \in \mathbb{R}^m$ from the linear system

$$(8.15) \quad S \xi = \eta$$

with matrix $S = (s_{ij}) \in \mathbb{R}^{m \times m}$, $s_{ij} = [\phi_i, \phi_j]$, and $\eta = (\eta_i) \in \mathbb{R}^m$, $\eta_i = (r, \phi_i)$. The matrix S is called the stiffness matrix; it is non-singular, even positive definite (see Exercise 8.3). Therefore, the Galerkin approximation w_h is well-defined. A simple choice for V_h and its basis is discussed below.

The following result gives an error estimate for the Galerkin approximation in the energy norm:

Theorem 8.3 The error $w - w_h$ of the Galerkin method satisfies

$$(8.16) \quad \|w - w_h\|_* = \min_{v_h \in V_h} \|w - v_h\|_*.$$

Proof. Since $V_h \subset H_0^1[a, b]$, it is seen by (8.10), (8.14) that we have the orthogonality relation

$$(8.17) \quad [w - w_h, u_h] = 0 \quad \text{for any } u_h \in V_h.$$

By considering arbitrary $v_h \in V_h$, and applying (8.17) with $u_h = w_h - v_h$, it is seen that

$$\|w - w_h\|_*^2 = [w - w_h, w - w_h] = [w - w_h, w - v_h] \leq \|w - w_h\|_* \|w - v_h\|_*.$$

Consequently, $\|w - w_h\|_* \leq \inf_{v_h \in V_h} \|w - v_h\|_*$, but since the approximation w_h itself belongs to V_h the estimate (8.16) follows. \square

We see from this theorem that, in the energy norm, the error $w - w_h$ between the exact solution w and numerical approximation w_h is determined by how well w can be approximated in the space V_h .

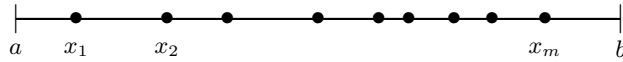
By combining Theorem 8.3 with Lemma 8.1 we obtain the following estimate:

Corollary 8.4 The derivative of the error $w - w_h$ of the Galerkin method satisfies

$$(8.18) \quad \|w' - w_h'\| \leq \frac{\gamma_1}{\gamma_0} \cdot \inf_{v_h \in V_h} \|w' - v_h'\|.$$

Piecewise linear approximations. If the space V_h is spanned by basis functions with a small support, then $\phi_i(x)\phi_j(x)$ will be identically equal to zero for most indices i, j . With suitable ordering, the stiffness matrix is then a band matrix with small band width, which makes the linear system (8.15) easy to solve.

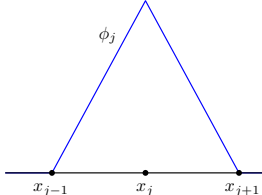
We introduce the grid points $x_0 = a < x_1 < x_2 < \dots < x_m < x_{m+1} = b$ with mesh widths $h_j = x_j - x_{j-1}$. Further, let $h = \max_{1 \leq j \leq m+1} h_j$ be the maximal mesh width.



For our approximation space we now take

$$(8.19) \quad V_h = \{v_h \in H_0^1[a, b] : v_h \text{ is linear on each interval } [x_{j-1}, x_j]\}.$$

On this space of piecewise linear functions, a basis is provided by the so-called *hat-functions*, defined as

$$\phi_j(x) = \begin{cases} 0 & \text{if } x < x_{j-1} \text{ or } x > x_{j+1}, \\ \frac{x-x_{j-1}}{x_j-x_{j-1}} & \text{if } x_{j-1} \leq x \leq x_j, \\ \frac{x-x_{j+1}}{x_j-x_{j+1}} & \text{if } x_j \leq x \leq x_{j+1}, \end{cases}$$


for $j = 1, 2, \dots, m$. With this basis it is obvious that $\phi_i(x) \cdot \phi_j(x) \equiv 0$ if $|i - j| \geq 2$. The stiffness matrix S is therefore tri-diagonal, which makes the system (8.15) easy to solve.

Useful upper bounds for the errors in (8.16) and (8.18) are obtained by considering the piecewise linear interpolant v_h of w , defined on each sub-interval $[x_{j-1}, x_j]$ by

$$(8.20) \quad v_h(x) = w(x_{j-1}) + \frac{x - x_{j-1}}{x_j - x_{j-1}} (w(x_j) - w(x_{j-1})) .$$

This piecewise linear interpolant does not necessarily minimize the right-hand sides in (8.16) and (8.18), but it does give useful upper bounds. It can be shown (Exercise 8.6) that $\|w' - v'_h\| \leq h\|w''\|$. Corollary 8.4 thus gives

$$(8.21) \quad \|w' - w'_h\| \leq C' h \cdot \|w''\|$$

with a constant $C' > 0$. Since $\|w - w_h\| \leq (b-a)\|w' - w'_h\|$, as shown in Lemma 8.1, we also obtain an $\mathcal{O}(h)$ estimate for the error $\|w - w_h\|$. However, this is not optimal. It will be shown below that

$$(8.22) \quad \|w - w_h\| \leq C h^2 \cdot \|w''\|$$

with a constant $C > 0$, but the derivation of this result is more technical.

A refined error bound.* To prove the error bound (8.22), consider the following auxiliary problem: find $u \in V = H_0^1[a, b]$ such that

$$(8.23) \quad [u, v] = (w - w_h, v) \quad \text{for any } v \in V .$$

Here the error $w - w_h$ plays the role of a source term. Therefore, just as we had (8.13) for the original problem, we now get $\|u''\| \leq \kappa_0\|w - w_h\|$. Taking $v = w - w_h$ in (8.23), it follows from (8.17) that

$$\|w - w_h\|^2 = [u, w - w_h] = [u - u_h, w - w_h] \leq \|u - u_h\|_* \|w - w_h\|_*$$

for arbitrary $u_h \in V_h$. To get a useful upper bound, we can choose u_h as the linear interpolant of u . Using estimates for the interpolation errors (Exercise 8.6), it then follows that $\|u - u_h\|_* \leq \gamma_1\|u' - u'_h\| \leq \gamma_1 h\|u''\|$, but this also implies $\|u - u_h\|_* \leq \kappa_0 \gamma_1 h\|w - w_h\|$. Consequently

$$\|w - w_h\| \leq \kappa_0 \gamma_1 h \cdot \|w - w_h\|_* .$$

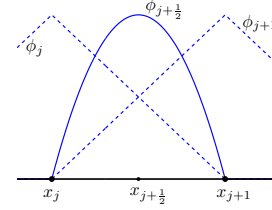
Since we already had the bounds $\|w - w_h\|_* \leq \gamma_1 \|w' - w'_h\| \leq \gamma_1 C' h \|w''\|$, the refined estimate (8.22) is obtained.

Higher-order methods.* To construct higher-order methods, the space V_h can be chosen to consist of piecewise polynomials. For example, with quadratic polynomials, we take V_h to be the set of functions $v_h \in H_0^1[a, b]$ such that v_h is a polynomial of degree 2 or less on each sub-interval $[x_{j-1}, x_j]$.

To obtain a suitable basis for this space V_h , we can start with the hat functions ϕ_j , $j = 1, 2, \dots, m$, and then add the piecewise quadratics

$$\phi_{j+1/2}(x) = \begin{cases} 4(x-x_j)(x_{j+1}-x) & \text{for } x \in [x_j, x_{j+1}], \\ 0 & \text{otherwise,} \end{cases}$$

$j = 0, 1, \dots, m$. Then the set of functions $\psi_j = \phi_{2j}$, $j = 1, 2, \dots, n$, provides a basis with $n = 2m + 1$.



These ideas extend to higher orders, taking V_h as the set of piecewise polynomials with degree $\leq q$ on each sub-interval $[x_j, x_{j+1}]$. In the same way as for the piecewise linear approximations, it can then be shown that the error will satisfy bounds $\|w' - w'_h\| = \mathcal{O}(h^q)$ and $\|w - w_h\| = \mathcal{O}(h^{q+1})$, under suitable smoothness assumptions on the solution w .

Methods of this type, using Galerkin approximations with piecewise polynomials, are known as *finite element methods*. The term 'elements' here refers to the sub-intervals $[x_{j-1}, x_j]$ with the local basis functions. Finite element methods are in particular important for boundary value problems for partial differential equations, with spatial variable in \mathbb{R}^d , for which (8.6) is an example with $d = 1$,

8.4 Exercises

Exercise 8.1. Consider the two-point boundary value problem $u'(t) = \lambda u(t)$ with $u(0)^2 + u(1)^2 = 1$. Find exact solutions using the concept of 'shooting'. What can you say about uniqueness of solutions?

Exercise 8.2. Suppose p, q, r are constant in (8.6), the grid is uniform, and we use the simple finite element method obtained from the Galerkin method with piecewise linear functions. Compute the stiffness matrix S and the inhomogeneous term η in (8.15). Compare this with the finite difference system (8.8).

Exercise 8.3. Let $\phi_1, \phi_2, \dots, \phi_m$ be a basis for the approximation space V_h in the Galerkin method.

(a) Show that (8.14) is equivalent to the linear system (8.15).

(b) Show that the stiffness matrix $S = ([\phi_i, \phi_j])$ in this linear system (8.15) is positive definite, i.e., $\xi^T S \xi > 0$ for all nonzero $\xi \in \mathbb{R}^m$.

*Exercise 8.4.** Consider the set of functions $V = H_0^1[a, b]$.

(a) Show that V is a linear vector space. Also show that for $u, v \in V$ the (point-wise) product function $y(x) = u(x)v(x)$ belongs to V .

(b) Show that the bilinear form $[u, v]$ is an inner-product on V ; i.e., for any $u, v, w \in V$ and $\alpha, \beta \in \mathbb{R}$ we have: (i) $[u, v] = [v, u]$, (ii) $[\alpha u + \beta v, w] = \alpha[u, w] + \beta[v, w]$, (iii) $[u, u] \geq 0$ with equality only if $u = 0$.

(c) Prove the Cauchy-Schwarz inequality: $[u, v] \leq \|u\|_* \|v\|_*$ for any pair $u, v \in V$. Hint: consider $\|\alpha u + \beta v\|_*^2 = \alpha^2 \|u\|_*^2 + 2\alpha\beta[u, v] + \beta^2 \|v\|_*^2$ with $\alpha = -[u, v]/\|u\|_*$ and $\beta = \|u\|_*$.

Exercise 8.5. Let $F(u) = [u, u] - 2(r, u)$ for $u \in V = H_0^1[a, b]$, and consider a finite dimensional subspace $V_h \subset V$. Show that the Galerkin approximation $w_h \in V_h$ minimizes the functional $F(u)$ over the space V_h . Hint: consider $F(u + tv)$ with variable $t \in \mathbb{R}$. Note: minimization of the functional $F(u)$ over V_h is called the *variational form* or *Ritz form* of (8.14).

*Exercise 8.6.** Consider the linear interpolant v_h of w as given by (8.20), and let $h_j = x_j - x_{j-1}$, $h = \max h_j$. For a real function u on $[a, b]$, denote by u_{I_j} the restriction of u to the sub-interval $I_j = [x_{j-1}, x_j]$:

$$u_{I_j}(x) = \begin{cases} u(x) & \text{if } x \in [x_{j-1}, x_j], \\ 0 & \text{if } x \notin [x_{j-1}, x_j]. \end{cases}$$

(a) Show that $\|(w - v_h)_{I_j}\| \leq h_j^2 \|w''_{I_j}\|$ and $\|(w' - v'_h)_{I_j}\| \leq h_j \|w''_{I_j}\|$.

(b) Show that $\|w - v_h\| \leq h^2 \|w''\|$ and $\|w' - v'_h\| \leq h \|w''\|$.

Hints: part (a) is not straightforward; the main steps are

$$(s_1) \quad \int_{x_{j-1}}^{x_j} |w - v_h|^2 dx = h_j^2 \cdot \int_{x_{j-1}}^{x_j} |w' - v'_h|^2 dx,$$

$$(s_2) \quad \int_{x_{j-1}}^{x_j} |w' - v'_h|^2 dx = \int_{x_{j-1}}^{x_j} (w'' - v''_h)(w - v_h) dx,$$

$$(s_3) \quad \int_{x_{j-1}}^{x_j} |w' - v'_h|^2 dx \leq \left(\int_{x_{j-1}}^{x_j} |w''|^2 dx \right)^{1/2} \left(\int_{x_{j-1}}^{x_j} |w - v_h|^2 dx \right)^{1/2}.$$

Here (s_1) is similar to (8.12); in (s_2) integration by parts is used; and for (s_3) the Schwarz inequality is again applied.

Exercise 8.7 (programming). Write a program for piecewise linear Galerkin approximations on a uniform grid to solve $w'' + w = r$ with $[a, b] = [0, \pi]$, $r(x) = (\sin x - 2 \cos x)e^{-x}$ and $w(x) = e^{-x} \sin x$. It is easiest here to calculate the integrals $[\phi_i, \phi_j]$ for the stiffness matrix by hand. Then for (r, ϕ_j) we can use trapezoidal or Simpson quadrature, as in the formulas (5.2) and (5.3).

Test your program with these choices. To measure the error it is easiest to take the Euclidian norm of $e \in \mathbb{R}^m$ with components $e_j = w(x_j) - w_h(x_j)$. Explain why the use of the trapezoidal rule would not be suited for the integrals $[\phi_i, \phi_j]$.