Graduate Texts in Physics

Simon Širca
Martin Horvat

Computational Methods for Physicists

Compendium for Students

Springer

Graduate Texts in Physics

Simon Širca
Martin Horvat

Computational Methods in Physics

Compendium for Students

Second Edition

Springer

Preface to the First Edition

This book evolved from short written homework instructions for the course in Computational Physics at the Department of Physics, University of Ljubljana. The feedback received from the students was used to gradually supplement the instructions by oral presentations in the classroom and additional material on the web. The heritage of this course, established and initially taught for a number of years by Prof. Kodre, represented a basis onto which we attempted to span an even richer manifold, and to better elucidate the "exercises" from the mathematical, physical, as well as programming and computational viewpoints. The somewhat spartan instructions thus evolved into a much more general textbook which is intended primarily for third- and fourth-year physics students, and for PhD students as an aid for all courses with a mathematical physics tinge. The book might also appeal to mathematics students. It was one of our local goals to modestly interweave physics and mathematics studies, and this is why the book steers between mathematical rigidity and more profane perspectives of numerical methods, while it tries to preserve the colorful content of the field of mathematical physics.

We were driven by the realization that physics students are often insufficiently prepared to face various obstacles they encounter in numerical solution or modeling of physical problems. Only a handful of them truly know how something can "actually be computed" or how their work can be efficiently controlled and its results can be reliably checked. Everyone can solve the matrix system $Ax = b$, but almost no one has an idea how to estimate the error and relate this estimate to the possible true error. They use explicit integrators of differential equations indiscriminately until they try to look closely at solutions of a problem as simple as $\ddot{x} = -x$. The direness of the situation is compounded by many commercial tools giving a false impression that all problems can be solved by a single keystroke. In parts of the text where basic approaches are discussed, we insist on seemingly ballast numerical details while, on the other hand, we did wish to offer at least some "serious" methods and illustrate them by manageable examples. The book swings back and forth between these extremes: it tries to be neither fully elementary nor encyclopedically complete, but at any rate representative—at least for the first-time reader.

The book is structured exactly with such gradations in mind: additional, "non-compulsory" chapters are marked with stars ★ and can be read by particularly motivated students or used as reference. Similarly, the ⊙ symbols denote simpler tasks in the end-of-chapter problems, while more demanding ones are marked by the symbols ⊕. The main text is peppered with examples terminated by the symbol ◁. The purpose of the appendices is not merely to remove the superfluous contents from the main text, but to enhance the programming efficiency (above all, Appendices B, C, E, J, and K). The sour apples we force our reader to bite are the lack of detailed derivations and references to formulas placed in remote parts of the text, although we tried to design the chapters as self-contained units. This style requires more concentration and consultation with literature on the reader's part, but makes the text more concise. In turn, the book does call for an inspired course tutor. In a typical one-semester course, she may handpick and fine-tune a dozen or so end-of-chapter problems and supply the necessary background, while the students may peruse the book as a convenient point of departure for work.

The end-of-chapter problems should resonate well with the majority of physics students. We scooped up topics from most varied disciplines and tried to embed them into the framework of the book. Chapters are concluded by relatively long lists of references, with the intent that the book will be useful also as a stepping stone for further study and as a decent vademecum.

In spite of all care, mistakes do occur. We shall be grateful to all readers turning our attention to any error they might spot, no matter how relevant. The Errata is maintained at the book's web page http://cmp.books.fmf.uni-lj.si which also contains the data files needed in some of the problems.

We wish to express our gratitude to Prof. Claus Ascheron, Senior Editor at Springer, for his effort in preparation and advancement of this book, as well as to Donatas Akmanavičius and his team for its meticulous production at VTeX.

The original text of the Slovenian edition was scrutinized by two physicists (Profs. Alojz Kodre and Tomaž Prosen) as well as three mathematicians (Associate Professors Emil Žagar, Marjetka Krajnc, and Gašper Jaklič). We thank them; from the navigation between the Scylla and Charybdis of these reviewers, we emerged as better sailors and arrived happily, after years of roaming the stormy seas, to our Ithaca.

Ljubljana, Slovenia

Simon Širca
Martin Horvat

Contents

1.2.2 Rational (Padé) Approximation

Suppose that, on some interval, we wish to effectively approximate a function f possessing a power expansion

$$f(z) = \sum_{k=0}^{\infty} c_k z^k \,. \tag{1.10}$$

A Padé approximation of f is a rational function with a numerator of degree L and denominator of degree M, determined such that its power expansion matches the series (1.10) up to including the power $L + M$. In other words, if we can find a polynomial P_L of degree L and Q_M of degree M such that

$$f(z) = \frac{P_L(z)}{Q_M(z)} + \mathcal{O}(z^{L+M+1}) \,, \qquad Q_M(0) = 1 \,,$$

then

$$[L/M]_f(z) = \frac{P_L(z)}{Q_M(z)} = \frac{a_0 + a_1 z + a_2 z^2 + \cdots + a_L z^L}{b_0 + b_1 z + b_2 z^2 + \cdots + b_M z^M} \,, \qquad b_0 = 1 \,, \tag{1.11}$$

defines a Padé approximation of order (L, M) of the function f. The coefficients a_k and b_k can be determined by equating the approximation $[L/M]_f$ with the power series for f and reading off the coefficients of the same powers of x:

$$\left(b_0 + b_1 z + \cdots + b_M z^M\right)\left(c_0 + c_1 z + \cdots\right) = a_0 + a_1 z + \cdots + a_L z^L + \mathcal{O}(z^{L+M+1}) \,.$$

By comparing the terms with powers $z^{L+1}, z^{L+2}, \ldots, z^{L+M}$ we obtain a system of equations for the coefficients b_k of the denominator Q_M, while by comparing terms with powers $z^0, z^1, \ldots, z^L$ we obtain explicit equations for the coefficients a_k of the numerator P_L. For example, with $L = M = 3$, we get

$$\begin{pmatrix} c_3 & c_2 & c_1 \\ c_4 & c_3 & c_2 \\ c_5 & c_4 & c_3 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} -c_4 \\ -c_5 \\ -c_6 \end{pmatrix} \,, \qquad \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} c_0 & 0 & 0 & 0 \\ c_1 & c_0 & 0 & 0 \\ c_2 & c_1 & c_0 & 0 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \begin{pmatrix} 1 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \,.$$

Since $b_0 = 1$, the first line of the equation on the right tells us that the zeroth coefficient of P_L is equal to the zeroth coefficient of the power expansion of the function, $a_0 = c_0$. The bulk of the work is hidden in the matrix system on the left. Large Padé systems of this type are solved by robust algorithms like Gauss elimination with complete pivoting because, in most cases, we are interested in relatively low degrees L and M and because accuracy takes precedence over speed. In the sense of

properties mentioned in the following, *diagonal Padé approximations*, in which $L = M$, are the most efficient.

Example (Adapted from [15], p. 4.) The function

$$f(z) = \sqrt{\frac{1+z/2}{1+2z}} = \sum_{k=0}^{\infty} c_k z^k = 1 - \frac{3}{4}z + \frac{39}{32}z^2 - \frac{267}{128}z^3 + \frac{7563}{2048}z^4 - \cdots \tag{1.12}$$

has the first two diagonal Padé approximations

$$[1/1]_f(z) = \frac{1+\frac{7}{8}z}{1+\frac{13}{8}z}, \qquad [2/2]_f(z) = \frac{1+\frac{17}{8}z+\frac{61}{64}z^2}{1+\frac{23}{8}z+\frac{121}{64}z^2}.$$

(Compute them!) The comparison of two power expansions and these Padé approximations is shown in Fig. 1.3, revealing a most pleasant property: if some power series converges to a function with a convergence radius ρ, that is, for all $|z| < \rho$ and $0 < \rho < \infty$, then an appropriately chosen Padé approximation converges to f for all $z \in \mathcal{R}$, where the domain $\mathcal{R}$ is larger than the domain defined by $|z| < \rho$. The function f in example (1.12) has the limit $\lim_{z\to\infty} f(z) = 1/2$ and its power series has a convergence radius of only $\rho = 1/2$. On the other hand, both lowest-order diagonal Padé approximations are stable at infinity. Moreover, when the order of the approximation is increased, the correct limit $1/2$ is approached rapidly: $\lim_{z\to\infty}[1/1]_f(z) = 7/13 \approx 0.5385$ and $\lim_{z\to\infty}[2/2]_f(z) = 61/121 \approx 0.5041$. In other words, the Padé approximation tells us something about how the function behaves outside of the convergence radius of its power series, and ensures better asymptotics. ◁

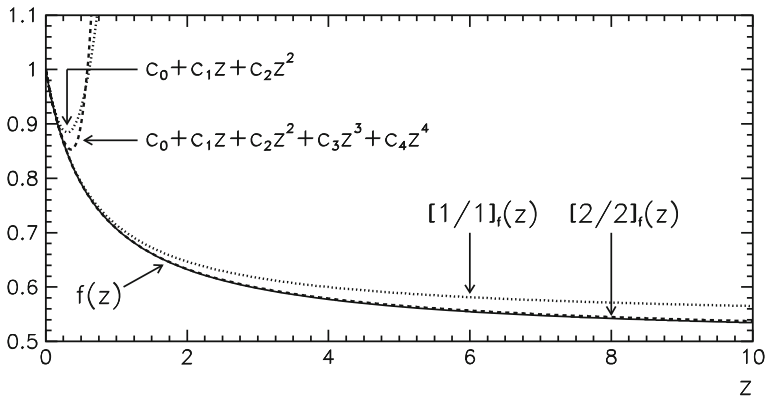


Fig. 1.3 Comparison of power expansions of degree two and four for the function (1.12) and the diagonal Padé approximations $[1/1]_f$ and $[2/2]_f$. The curves for f and $[2/2]_f$ are barely distinguishable in this plot

Substantial work has been done in the minimization of summation errors (see [8, 38–40]). Here we list three most widely used summation methods that do not require more than $\mathcal{O}(n)$ of operations.

Simple recursive summation Assume that we have the values $\{a_k\}_{k=0}^n$ and wish to compute their sum $S = \sum_{k=0}^n a_k$. Most obviously, this can be accomplished by computing $\hat{S} = (\ldots(((a_0 \oplus a_1) \oplus a_2) \oplus a_3)\ldots \oplus a_{n-1}) \oplus a_n$, in a loop

Input: real numbers $a_0, a_1, \ldots, a_n$
$\hat{S} = a_0$;
for $k = 1$ **step** 1 **to** n **do**
  $\hat{S} = \hat{S} + a_k$;
end
Output: $\hat{S}$ is the numerical sum of numbers a_k

The deviation of the numerical sum $\hat{S}$ from the exact sum S strongly depends on how a_k are sorted. If they are unsorted, we have

$$\left|S - \hat{S}\right| \leq \frac{\varepsilon_{\mathrm{M}}}{2} n \sum_{k=0}^{n} |a_k| + \mathcal{O}\left(\varepsilon_{\mathrm{M}}^2\right), \tag{1.57}$$

where ε_{M} is the arithmetic precision (see p. 2) [39]. In simple summation one can therefore expect a loss of up to $\log_{10} n$ significant digits. The estimate for the upper limit of the error (not the error itself) is smallest when the terms are sorted as $|a_k| \leq |a_{k+1}|$. Sorting requires at least $\mathcal{O}(n \log n)$ additional operations.

Kahan's algorithm A much better procedure to sum a series, by which the effect of rounding errors is greatly diminished, was proposed by Kahan [41]:

Input: real numbers $a_0, a_1, \ldots, a_n$
$\hat{S} = a_0$;
$c = 0$;
for $k = 1$ **step** 1 **to** n **do**
  $y = a_k - c$;
  $t = \hat{S} + y$;
  $c = (t - \hat{S}) - y$; // do not omit brackets
  $\hat{S} = t$;
end
Output: $\hat{S}$ is the numerical sum of numbers a_k

Algebraically, the value of c is zero, but in finite arithmetic it represents a large part of the lost precision when summing $t = \hat{S} + y$. It is added to the sum in the next step and by doing this, it compensates the rounding error from the previous step. The deviation of the numerical sum from the exact one satisfies

$$\left|S - \hat{S}\right| \leq \left(\varepsilon_{\mathrm{M}} + \mathcal{O}\left(n\varepsilon_{\mathrm{M}}^2\right)\right) \sum_{k=0}^{n} |a_k| . \tag{1.58}$$

According to (1.58), Kahan's summation is more precise than simple summation for $n\varepsilon_{\mathrm{M}}/2 \leq 1$. In practice, this actually applies to even larger n (see Fig. 1.9). In the implementation of the algorithm we should make sure that the compiler does not simplify it, since the essence of its strength is hidden in the rules of floating-point arithmetic. In C and C++ variables should be declared `volatile`.

Recursive summation of pairs Summation is an associative and commutative operation between real numbers. Algebraically, the order of summation is thus irrelevant, and this fact is exploited by the Linz's procedure [42]. In the first step we sum the consecutive pairs of terms and obtain a new series. In this series we again sum the consecutive pairs and repeat this ($r = \lceil \log_2 n \rceil$)-times, until we are left with only one term, which represents the final sum:

Input: real numbers $a_0, a_1, \ldots, a_{n-1}$, where $n = 2^r$, $r \in \mathbb{N}$
$m = m' = n/2$;
for $k = 0$ **step** 1 **to** $m - 1$ **do**
  $S_{0,k} = a_{2k} + a_{2k+1}$;
end
for $j = 1$ **step** 1 **to** $r - 1$ **do**
  $m' = m'/2$;
  for $k = 0$ **step** 1 **to** $m' - 1$ **do**
    $S_{j,k} = S_{j-1,2k} + S_{j-1,2k+1}$;
  end
end
Output: $\hat{S} = S_{r-1,0}$ is the numerical sum of numbers a_k

Because each term a_k in the sum is touched only r-times, the deviation of the sum $\hat{S}$ from the exact value S is much smaller than in simple recursive summation:

$$\left|S - \hat{S}\right| \leq \frac{\varepsilon_{\mathrm{M}}}{2} \log_2 n \sum_{k=0}^{n-1} |a_k|$$

(compare to (1.57)). For the intermediate sums $S_{j,k}$ we need additional computer memory to store $n/2$ real numbers, which is a bit wasteful compared to the simple and Kahan's summation which require only $\mathcal{O}(1)$ of memory. Linz's algorithm can be improved by compensating the numerical error and selecting the pairs in a more intricate manner. For details, consult [43].

In all three methods we specified the upper bounds for $|S - \hat{S}|$; for a given set of numbers $\{a_k\}$, all methods may be equally precise. In general, we recommend the Linz's method unless pairs can not be formed or this does not make much sense (for example, for relatively short series). On the other hand, the Kahan's method, which is both simple and precise, never fails to enchant (see Fig. 1.9). For very precise summation, we resort to more sophisticated but slower methods like *distillation algorithms* described in [38, 44, 45].

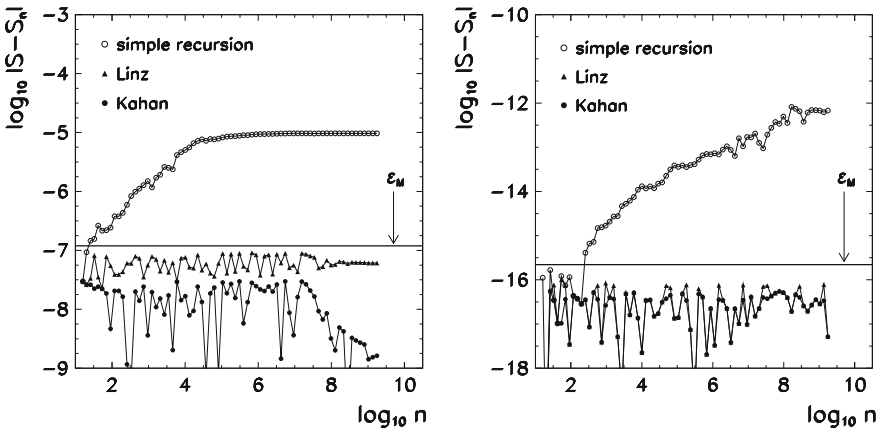


Fig. 1.9 Rounding errors in summing series with many terms. Shown is the absolute error of the numerical partial sums $S_n = \sum_{k=0}^{n} (-1)^k/(k+1)$ with respect to the limiting value $S_\infty = \log 2$. [LEFT] Summation in single-precision arithmetic. [RIGHT] Summation in double-precision arithmetic. The horizontal lines correspond to $\varepsilon_M = 1.19 \cdot 10^{-7}$ (left) and $\varepsilon_M = 2.22 \cdot 10^{-16}$ (right) — see p. 2

1.4.3 Acceleration of Convergence

The convergence of the partial sums $S_n = \sum_{k=0}^{n} a_k$ to the limit $S = \lim_{n\to\infty} S_n$ may be slow. By "slow" we mean its leading behavior to be $|S_n - S| = \mathcal{O}(n^{-p})$ (power) or $|S_n - S| = \mathcal{O}((\log n)^{-p})$ (logarithmic) where $p > 0$ is the convergence order. Slow convergence is not desired since it implies large numerical costs and a potential accumulation of rounding errors.

We speak of "fast" convergence when it is better than "slow" according to the definition given above. Ideally, one would like to have exponential (geometric) convergence $|S_n - S| = \mathcal{O}(a^n)$ where $a \in [0, 1)$. In many cases, convergence can be accelerated by transforming the original series into another series that converges more rapidly. In the following, we describe a few basic approaches. A modern introduction to convergence acceleration with excellent examples and many hints can be found in [46]; for a more detailed review, see [47].

Richardson extrapolation Assume that we already know the order of convergence for a series $S = \sum_{k=0}^{\infty} a_k$ so that for its partial sums $S_n = \sum_{k=0}^{n} a_k$ we have

$$S = S_n + \frac{\alpha}{n^p} + \mathcal{O}(n^{-r}) , \qquad r > p > 0 .$$

We think of the "value of the series" as being the value of the partial sum plus a correction with the known leading-order behavior α/n^p. By transforming

$$T_n^{(1)} = \frac{2^p S_{2n} - S_n}{2^p - 1} = S_{2n} + \frac{S_{2n} - S_n}{2^p - 1}$$

the term α/n^p can be eliminated and the terms $T_n^{(1)}$ give us a better estimate for the sum, for which we obtain $S = T^{(1)} + \mathcal{O}(n^{-r})$. The very same trick can be repeated — until this makes sense — by forming new sequences,

$$T_n^{(2)} = \frac{2^{p+1}T_{2n}^{(1)} - T_n^{(1)}}{2^{p+1}-1}\,, \qquad T_n^{(3)} = \frac{2^{p+2}T_{2n}^{(2)} - T_n^{(2)}}{2^{p+2}-1}\,, \qquad \cdots\,.$$

Richardson's procedure is an example of a *linear extrapolation method* X, in which for partial sums S_n and T_n of two series we have $X(\lambda S_n + \mu T_n) = \lambda X(S_n) + \mu X(T_n)$. It is efficient if the partial sums S_n behave like polynomials in some sequence h_n, that is, $S_n = S + c_1 h_n^{p_1} + c_2 h_n^{p_2} + \cdots$ or $S_{n+1} = S + c_1 h_{n+1}^{p_1} + c_2 h_{n+1}^{p_2} + \cdots$, where the ratio h_{n+1}/h_n is constant. If this condition is not met, linear extrapolation may become inefficient or does not work at all. In such cases we resort to *semi-linear* or *non-linear extrapolation* [46].

Aitken's method Aitken's method is one of the classical and most widely used ways to accelerate the convergence by non-linear extrapolation. Assume that we have a sequence of partial sums S_n with the limit $S = \lim_{n\to\infty} S_n$. We transform the sequence S_n into a new sequence

$$T_n^{(1)} = S_n - \frac{(S_{n+1} - S_n)^2}{S_{n+2} - 2S_{n+1} + S_n}\,, \qquad n = 0, 1, 2, \ldots\,, \tag{1.59}$$

where the fraction should be evaluated exactly in the given form in order to minimize rounding errors. (Check that the transformed sequence (1.59) is identical to the column $\epsilon(n, 2)$ of the Wynn's table (1.13).) We repeat the process by using $T_n^{(1)}$ instead of S_n to form yet another, even more accelerated sequence $T_n^{(2)}$, and proceed thus until it continues to make sense as far as the rounding errors are concerned. Figure 1.10 (left) shows the comparison of convergence speeds for the unaccelerated partial sums S_n and the accelerated sequences $T_n^{(1)}$, $T_n^{(2)}$, and $T_n^{(3)}$.

Aitken's method is optimally suited for acceleration of linearly convergent sequences, for which $\lim_{n\to\infty}(S_{n+1} - S)/(S_n - S) = a$ with $-1 \le a < 1$. Such sequences originate in numerous numerical algorithms based on finite differences. In some cases, we apply Aitken's formula to triplets of partial sums S_{n+p}, S_n, and S_{n-p}, where $p > 1$, because sometimes the geometric convergence of a series only becomes apparent at larger p; see also Sect. 2.3 and [48].

Kummer's acceleration The basic idea of the Kummer's method of summing a convergent series $S = \sum_k a_k$ is to subtract from it another (auxiliary) convergent series $B = \sum_k b_k$ with the known limit B, such that

$$\lim_{k\to\infty} \frac{a_k}{b_k} = \rho \neq 0\,.$$

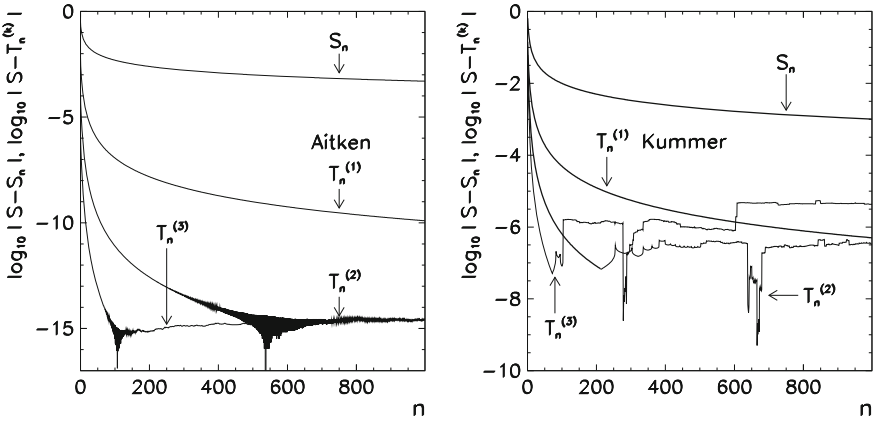


Fig. 1.10 Acceleration of convergence of partial sums. [LEFT] Aitken's method for the sum $S_n = \sum_{k=0}^{n}(-1)^k/(k+1)$ with the limit $S = \lim_{n\to\infty} S_n = \log 2$. Shown is the acceleration of this series with very slow (logarithmic) convergence by three-fold repetition of the Aitken's method. For typical series usually a single step of (1.59) suffices. [RIGHT] Kummer's acceleration of the sums $S_n = \sum_{k=1}^{n} 1/k^2$ with the limit $S = \lim_{n\to\infty} S_n = \pi^2/6$ by using the auxiliary series $\sum_{k=1}^{\infty} 1/(k(k+1))$

Then the original series can be transformed to

$$T = \sum_k a_k = \rho \sum_k b_k + \sum_k (a_k - \rho b_k) = \rho B + \sum_k \left(1 - \rho \frac{b_k}{a_k}\right) a_k \,. \qquad (1.60)$$

The convergence of the series on the right is faster than the convergence of that on the left since $(1 - \rho b_k/a_k)$ tends to zero when $k \to \infty$. An example is the sum $S = \sum_{k=1}^{\infty} 1/k^2 = \pi^2/6$ from which we subtract $B = \sum_{k=1}^{\infty} 1/(k(k+1)) = 1$, thus $\rho = \lim_{k\to\infty} k(k+1)/k^2 = 1$. We use the terms a_k and b_k, as well as ρ and B, in (1.60), and get the transformed partial sum

$$T_n^{(1)} = 1 + \sum_{k=1}^{\infty} \left(1 - \frac{k^2}{k(k+1)}\right) \frac{1}{k^2} \,,$$

which has a faster convergence than the original series. Again, the procedure can be invoked repeatedly (see [49] and Fig. 1.10 (right)).

1.4.4 Alternating Series

In alternating series the sign of the terms flips periodically,

$$S = a_0 - a_1 + a_2 - a_3 + \ldots = \sum_{k=0}^{\infty} (-1)^k a_k \,, \qquad S_n = \sum_{k=0}^{n} (-1)^k a_k \,.$$

In physics such examples can be encountered e.g. in electro-magnetism in problems with oppositely charged particles or currents flowing in opposite directions. An example is the calculation of the electric potential $U(x, y, z)$ of charges of opposite signs lying next to each other at distances a along the x-axis:

$$U(x, y, z) \propto \sum_{k=-\infty}^{\infty} \frac{(-1)^k}{\sqrt{(x + ka)^2 + y^2 + z^2}} .$$

Making a monotonous series alternate For realistic physics problems, the results of series summation may be unpredictable. Simple recursive summation may suffer from large rounding errors. On the other hand, the acceleration of alternating series is typically more efficient than the acceleration of series with exclusively positive (or exclusively negative) terms. A monotonous sequence can be transformed into an alternating one by using the Van Wijngaarden's trick:

$$\sum_{k=0}^{\infty} a_k = \sum_{k=0}^{\infty} (-1)^k b_k , \qquad b_k = \sum_{j=0}^{\infty} 2^j a_{2^j(k+1)-1} .$$

Euler's transformation One of the oldest ways to accelerate the convergence of an alternating sequence by a linear combination of its terms is the *Euler transformation*. We rewrite the original sum $S = \sum_k (-1)^k a_k$ and its partial sum as

$$S = \sum_{k=0}^{\infty} (-1)^k \frac{\Delta^k a_0}{2^{k+1}} , \qquad S_n = \sum_{k=0}^{n} (-1)^k \frac{\Delta^k a_0}{2^{k+1}} , \tag{1.61}$$

where

$$\Delta^k a_0 = (-1)^k \sum_{j=0}^{k} (-1)^j \binom{k}{j} a_j .$$

If there exist $N \in \mathbb{N}$ and $C > 0$ such that $|\Delta^n a_0| \leq C$ for all $n > N$, the series (1.61) converges faster than geometrically with

$$|S - S_n| \leq \frac{C}{2^{n+1}} , \qquad n > N .$$

In practical algorithms, we first form the partial sums

$$s_n^{(0)} = \sum_{k=0}^{n} (-1)^k a_k , \qquad n = 0, 1, \ldots, N - 1 ,$$

and recursively compute the *partial Euler transforms*

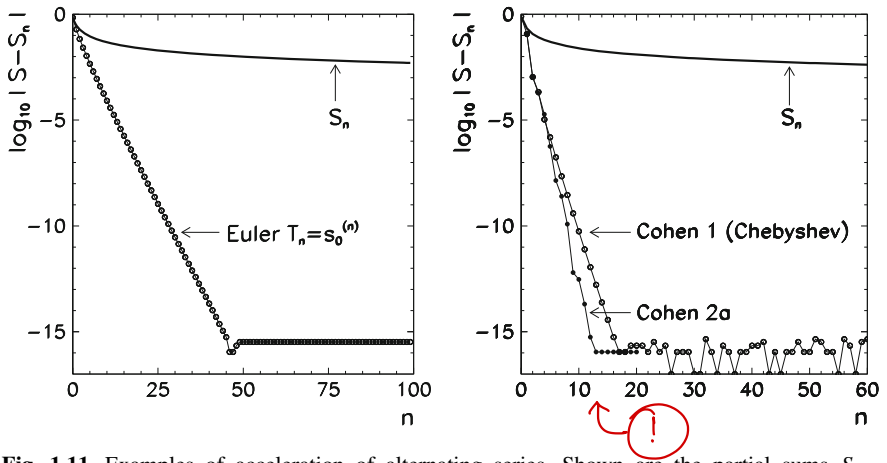


Fig. 1.11 Examples of acceleration of alternating series. Shown are the partial sums $S_n = \sum_{k=0}^{n}(-1)^k a_k$ without and with acceleration. [LEFT] Euler's method (1.62) for $a_k = 1/(k+1)$ (limit $S = \lim_{n\to\infty} S_n = \log 2$). [RIGHT] Cohen–Villegas–Zagier's algorithm 1 from p. 43 by using Chebyshev polynomials (1.66) and algorithm 2a in [51] for $a_k = 1/(2k+1)$ (limit $S = \lim_{n\to\infty} S_n = \pi/4$). To compute the partial sum to ≈ 15 significant digits typically less than $\approx 10 - 20$ terms of the accelerated series are required

$$s_n^{(j+1)} = \frac{1}{2}\left(s_n^{(j)} + s_{n+1}^{(j)}\right) , \qquad j = 0, 1, \ldots . \tag{1.62}$$

The values $T_n \equiv s_0^{(n)}$ represent the improved (accelerated) approximations of the partial sums S_n (Fig. 1.11 (left)). The procedure is numerically demanding, since it requires $\mathcal{O}(n^2)$ operations and $\mathcal{O}(n)$ of memory for a complete transformation of a series with n terms. It turns out that the optimally precise results are obtained not by using the transform (1.62) with $j = N - 1$ and $n = 0$, but with $j = \lceil 2N/3 \rceil$ and $n = \lceil N/3 \rceil$. An efficient implementation is given in [50].

Generalizing the Euler's method Euler's transformation can be generalized by using the theory of measures. In this fresh approach to the summation of alternating series [51] we assume that for a series $\sum_{k=0}^{\infty}(-1)^k a_k$ there exists a positive function w such that the series terms a_k are its moments on the interval [0, 1],

$$a_k = \int_0^1 x^k w(x)\,\mathrm{d}x \, . \tag{1.63}$$

The sum of the series can then be written as

$$S = \sum_{k=0}^{\infty}(-1)^k a_k = \int_0^1 \left(\sum_{k=0}^{\infty}(-1)^k x^k w(x)\right)\mathrm{d}x = \int_0^1 \frac{w(x)}{1+x}\,\mathrm{d}x \, .$$

(In the final summation formula the weight function does not appear.) In the last step, we have used the identity

$$\sum_{k=0}^{n-1}(-1)^k x^k = \frac{1-(-x)^n}{1+x}\,, \qquad |x|<1\,, \tag{1.64}$$

in the limit $n \to \infty$. We now choose a sequence of polynomials $\{P_n\}$, where P_n has a degree n and $P_n(-1) \neq 0$. To the sequence $\{P_n\}$ we assign the numbers

$$S_n = \frac{1}{P_n(-1)} \int_0^1 \frac{P_n(-1) - P_n(x)}{1+x}\, w(x)\,\mathrm{d}x\,.$$

The numbers S_n are linear combinations of the series terms a_k. This can be seen by inserting the expression for a general polynomial $P_n(x) = \sum_{k=0}^{n} p_k(-x)^k$ into the equation for S_n and observe (1.64) and a_k (1.63). We obtain

$$S_n = \frac{1}{d_n}\sum_{k=0}^{n-1}(-1)^k c_k^{(n)} a_k\,, \qquad d_n = \sum_{k=0}^{n} p_k\,, \qquad c_k^{(n)} = \sum_{j=k+1}^{n} p_j\,.$$

The S_n defined in this way represent the partial sums of S that converge to S when n is increased. The difference between the partial sum S_n and the sum S can be constrained as

$$\left|S - S_n\right| \le \frac{1}{|P_n(-1)|}\int_0^1 \frac{|P_n(x)|}{1+x}\, w(x)\,\mathrm{d}x \le \frac{M_n}{|P_n(-1)|}\,|S|\,,$$

where $M_n = \sup_{x\in[0,1]} |P_n(x)|$ is the maximum value of the polynomial P_n on $[0, 1]$. The sufficient condition for the convergence of the partial sums S_n to S is therefore $\lim_{n\to\infty} M_n/P_n(-1) = 0$. The authors of [51] recommend to choose a sequence of polynomials $\{P_n\}$ such that $M_n/P_n(-1)$ converges to zero as quickly as possible. The following three choices are the most fruitful.

The first type of the polynomials P_n that may cross one's mind, is

$$P_n(x) = (1-x)^n = \sum_{k=0}^{n}\binom{n}{k}(-x)^k\,, \qquad P_n(-1) = 2^n\,, \qquad M_n = 1\,.$$

Namely, the corresponding partial sums are

$$S_n = \frac{1}{2^n}\sum_{k=0}^{n-1}(-1)^k c_k^{(n)} a_k\,, \qquad c_k^{(n)} = \sum_{j=k+1}^{n}\binom{n}{j}\,, \tag{1.65}$$

and they are identical to the partial sums of the Euler transform (1.61), except for a different subscripting (the sums (1.61) with subscript n are equal to the sums (1.65) with subscript $n+1$). By this choice we obtain $|S - S_n| \le |S|/2^n$. Faster convergence, $|S - S_n| \le |S|/3^n$, can be obtained by using the polynomials

$$P_n(x) = (1-2x)^n = \sum_{k=0}^{n} 2^k \binom{n}{k} (-x)^k \,, \qquad P_n(-1) = 3^n \,, \qquad M_n = 1 \,.$$

Here the partial sums have the form

$$S_n = \frac{1}{3^n} \sum_{k=0}^{n-1} (-1)^k c_k^{(n)} a_k \,, \qquad c_k^{(n)} = \sum_{j=k+1}^{n} 2^j \binom{n}{j} \,.$$

A third choice is a special family of Chebyshev polynomials, which have other beneficial algebraic properties and are orthogonal. We define these polynomials implicitly by $P_n(\sin^2 t) = \cos(2nt)$ or explicitly by

$$P_n(x) = T_n(1-2x) = \sum_{j=0}^{n} 4^j \frac{n}{n+j} \binom{n+j}{2j} (-x)^j \,,$$

where $T_n(x) = \cos(n \arccos x)$ are the standard Chebyshev polynomials of degree n on $[-1, 1]$. The polynomials of this sequence are computed by using the recurrence $P_{n+1}(x) = 2(1-2x)P_n(x) - P_{n-1}(x)$, which is initiated by $P_0(x) = 1$ and $P_1(x) = 1 - 2x$. For polynomials chosen in this way, one can show that $P_n(-1) = \frac{1}{2}[(3+\sqrt{8})^n + (3-\sqrt{8})^n]$ and $M_n = 1$. The partial sums

$$S_n = \frac{1}{P_n(-1)} \sum_{k=0}^{n-1} (-1)^k c_k^{(n)} a_k \,, \qquad c_k^{(n)} = \sum_{j=k+1}^{n} 4^j \frac{n}{n+j} \binom{n+j}{2j} \,, \tag{1.66}$$

converge to the final sum as

$$\left|S - S_n\right| \le \frac{2|S|}{(3+\sqrt{8})^n} < \frac{2|S|}{5.828^n} \,,$$

so we need to sum only $n \approx 1.31\, D$ terms for a precision of D significant digits! The coefficients $c_k^{(n)}$ and other constants can be computed iteratively and the whole computation of S_n can be implemented in a very compact algorithm [51]

Input: numbers $a_0, a_1, \ldots, a_{n-1}$ of an alternating series $\sum_{k=0}^{n-1} (-1)^k a_k$
$d = (3+\sqrt{8})^n$; $d = (d + 1/d)/2$;
$b = -1$; $c = -d$; $s = 0$;
for $k = 0$ **step** 1 **to** $n-1$ **do**
  $c = b - c$;
  $s = s + c\, a_k$;
  $b = (k+n)(k-n)b/((k+1/2)(k+1))$;
end
Output: partial sum $S_n = s/d$

This algorithm requires $\mathcal{O}(1)$ of memory and $\mathcal{O}(n)$ of CPU. Similar results can be obtained by using other families of orthogonal polynomials; the paper [51] describes further algorithms in which the coefficients of the partial sums can not be generated as easily, but yield even faster convergence. For many types of sequences, these algorithms allow us to achieve convergence rates of $|S - S_n| \leq |S|/7.89^n$, in some cases even the breath-taking $|S - S_n| \leq |S|/17.93^n$. However, they require $\mathcal{O}(n)$ of memory and $\mathcal{O}(n^2)$ of CPU.

1.4.5 Levin's Transformations

Levin's transformations [52] are among the most generally useful, handy, and efficient methods to accelerate the convergence of series by semi-linear extrapolation. We implement them by using divided differences which are computed recursively:

$$\delta^k f_n = \frac{\delta^{k-1} f_{n+1} - \delta^{k-1} f_n}{t_{n+k} - t_n}\,, \qquad \delta^0 f_n = f_n\,,$$

where $t_n = (n + n_0)^{-1}$ and we usually take $n_0 = 0$ or $n_0 = 1$. To compute the extrapolated partial sums we need the partial sums S_n and auxiliary functions ψ which depend on the terms of the sequence and its character (monotonous or alternating). We use the formula

$$S_{k,n} = \delta^k \left(\frac{S_n}{\psi(n)} \right) \left[\delta^k \left(\frac{1}{\psi(n)} \right) \right]^{-1}\,, \qquad k = 1, 2, \ldots\,. \tag{1.67}$$

and take $S_{k,0}$ (with $n_0 = 1$) or $S_{k,1}$ (with $n_0 = 0$) as the extrapolated sum. Levin's transformations differ by the functional forms of ψ. The best known are

$$T:\ \psi(n) = a_n\,, \qquad U:\ \psi(n) = (n + n_0)a_n\,, \qquad W:\ \psi(n) = a_n^2/(a_{n+1} - a_n)\,.$$

The T-transformation is best for alternating series in which the partial sums behave as $S_n \sim r^n$. The U-transformation works well with monotonous sequences for which $S_n \sim n^{-r}$ applies. The W-transformation can be used in either case regardless of the series type, although it is more sensitive to rounding errors than the U- and T-methods. The U-method is recommended [46] as a reliable way to speed up the summation of any series. The U-transformation, including the extrapolation to the limit and providing the remainder estimates, is implemented in the GSL library [53] in the `gsl_sum_levin_u_accel()` function.

Example Let us sum the slowly converging series $S_n = \sum_{k=0}^{n} (-1)^k/(k+1)$ with the limit $S = \lim_{n\to\infty} S_n = \log 2$. We choose the Levin's T-method and $n_0 = 0$, thus $t_n = n^{-1}$ and $\psi(n) = (-1)^n/(n+1)$. By using (1.67) with $n = 1$ we obtain

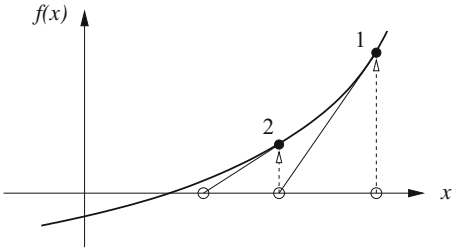


Fig. 2.1 Searching for the approximations of the zero by using the Newton-Raphson's method. At each point of the iteration $(x_k, f(x_k))$ we compute the tangent to the function. The intersection of the tangent with the abscissa is the new approximation of the zero x_{k+1}. See also Fig. 2.2 (right)

The sum $\phi_n(x)$ can be used in the iteration of a method of order $n+1$ like

$$x_{k+1} = \phi_n(x_k) \,,$$

as advertised at the outset (2.2). This iteration operates equally well on the real axis and in the complex plane, and can therefore be used to seek both real and complex roots. (If f is real and we wish to find a complex root, the initial approximation also needs to be complex.) The most important representatives of this family of methods are the Newton-Raphson's method ($n = 1$),

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{2.7}$$

(Fig. 2.1), and the modified Newton-Raphson's method ($n = 2$), with the iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} - \frac{f''(x_k)\,f(x_k)^2}{2\,f'(x_k)^3} \,.$$

Methods of higher orders can be constructed by augmenting the iteration function, but their applicability is limited as the numerical computation, or even the existence of higher derivatives of f, are questionable. Methods of orders greater than two are rarely used.

The Newton-Raphson's method with the iteration (2.7) is the most simple, generally useful and explored method from the Newton's family. The search for the zero by using this formula is illustrated in Fig. 2.1. Due to this characteristic graphical interpretation this procedure is also known as the *tangent method*.

It is clear from this construction that the Newton-Raphson's iteration has a quadratic convergence if the derivative $f'(x)$ exists in some open neighborhood S of ξ and is continuous and non-zero ($f'(x) \neq 0\ \forall S$). This can be confirmed by expanding the iteration formula around ξ in powers of the difference $e_k = x_k - \xi$. We get

$$e_{k+1} \approx \frac{f''(\zeta)}{2f'(x_k)} e_k^2 , \qquad \zeta = \zeta(x_k) \in S .$$

The method therefore has quadratic convergence when $\max_{x \in S} |f''(x)| \neq 0$. The speed of convergence is essentially driven by the ratio of the second and the first derivative evaluated at the zero of f. If the zero is multiple, the Newton-Raphson's method slows down and becomes only linearly convergent. In such cases, quadratic convergence can be restored by writing the iteration (2.7) as

$$x_{k+1} = x_k - m \frac{f(x_k)}{f'(x_k)} ,$$

where m is the order of the zero of f. If the order of ξ is unknown and there is a chance that the zero is multiple, one should not try to solve $f(x) = 0$ but rather

$$u(x) = 0 , \qquad u(x) = \frac{f(x)}{f'(x)} .$$

When u is inserted in (2.7), we obtain an iteration of the form

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k) - f(x_k) f''(x_k)/f'(x_k)} ,$$

which also accommodates multiple zeros.

Under certain assumptions about f and the initial value x_0, the Newton-Raphson sequence is locally convergent, but it does not converge globally (for arbitrary initial values); see Example below. For some classes of f Newton's method can be tuned to become globally convergent [4]. Sufficient conditions for convergence are specified by Ostrowski theorems (Appendix D).

Example (Adapted from [5]) Kepler's equation

$$E = M + \varepsilon \sin E \tag{2.8}$$

connects the parameters of the planet's orbital motion around the Sun: eccentric anomaly E, average anomaly M, and eccentricity $\varepsilon = \sqrt{1 - b^2/a^2}$ (Fig. 2.2 (left)). The time dependence of M is given by $M(t) = 2\pi(t - t_\mathrm{p})/T$, where T is the period and t_p is the time of the passage through the perihelion (the point on the elliptic orbit closest to the Sun). Various forms of analytic solutions exist [1].

Equation (2.8) can be solved by simple iteration, $E_{k+1} = M + \varepsilon \sin E_k$, which converges very slowly. It is therefore much better if the equation is rewritten in the Newton form (2.7) with the function $f(E) = E - M - \varepsilon \sin E$. We get

$$E_{k+1} = \frac{M - \varepsilon\,[E_k \cos E_k - \sin E_k]}{1 - \varepsilon \cos E_k} , \qquad k = 0, 1, 2, \ldots .$$

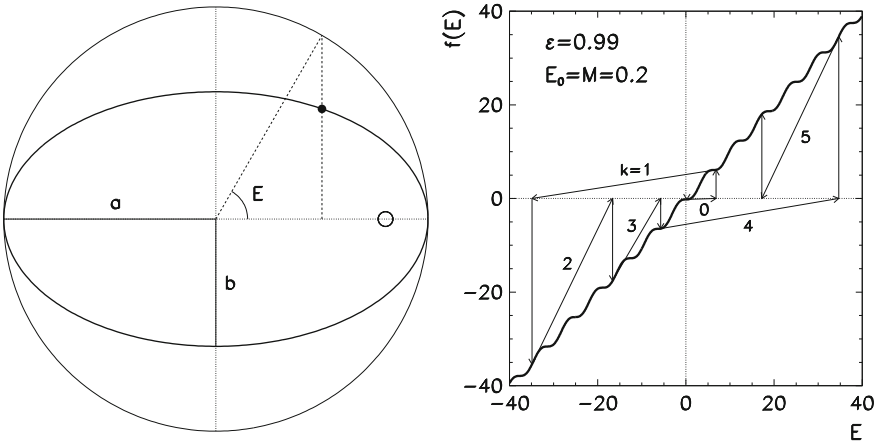


Fig. 2.2 Solving Kepler's equation by Newton's method. [LEFT] The definition of the anomaly E. [RIGHT] Poor convergence of the approximations E_k with a strongly eccentric orbit and bad initial approximation E_0. The function $f(E) = E - M - \varepsilon \sin E$ is drawn

With a prudent choice of the initial approximation the sequence $\{E_k\}$ converges very rapidly: for $\varepsilon = 0.95$, $M = 245°$, and the initial approximation $E_0 = 215°$, we obtain the solution $E \approx 3.740501878977461$ to machine precision in just three steps; the simple iteration would require 131 steps for the same precision.

For small eccentricities, $\varepsilon \ll 1$, Newton's method is rather insensitive to the initial approximation, as in such cases $E = M + \varepsilon \sin E \approx M$, and the simple guess $E_0 = M$ does the job. For large ε, however, the sequence of the approximations $\{E_k\}$ with a poorly chosen initial value E_0 may diverge. Assume that we always set $E_0 = M$ initially. If $(\varepsilon, M) = (0.991, 0.13\pi)$ or $(0.993, 0.13\pi)$, the sequence $\{E_k\}$ converges, but for the values $(0.992, 0.13\pi)$ it diverges! Similar behavior can be observed near $(\varepsilon, M) = (0.99, 0.2)$ (Fig. 2.2 (right)). The reason for the divergence is the bad choice of E_0 combined with the tricky form of the function $f(E) = E - M - \varepsilon \sin E$ with almost horizontal parts where the tangent to f is "thrown" far away (see the jumps at $k = 1$ and $k = 4$). A good approximation for any ε, which can be used to avoid the pitfalls seen above, is $E_0 = \pi$ [5]. ◁

Example We use Newton's method to compute the complex roots of $z^3 - 1 = 0$. The analytic solutions are 1, $\mathrm{e}^{\mathrm{i}2\pi/3}$, and $\mathrm{e}^{-\mathrm{i}2\pi/3}$. We choose the initial approximation $z_0 = x_0 + \mathrm{i}\, y_0 \in [-2, 2] \times [-2, 2]$ and observe the value of z at the end of the iteration $z_{k+1} = z_k - f(z_k)/f'(z_k)$, where $f(z) = z^3 - 1$. Figure 2.3 (left) shows the phase of the final value of z, while Fig. 2.3 (right) shows the number of steps needed to achieve convergence to machine precision. ◁

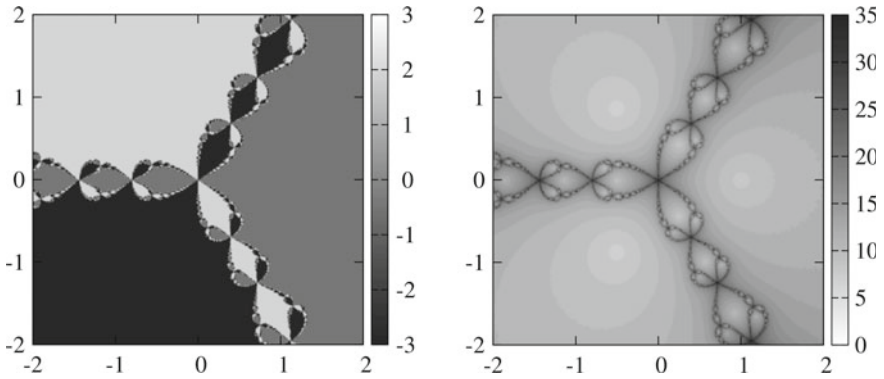


Fig. 2.3 Solving $z^3 - 1 = 0$ by Newton's method. [LEFT] The phase of the final value z as a function of the initial approximation specified by $\mathrm{Re}\, z$ and $\mathrm{Im}\, z$ on the abscissa and the ordinate. The three shaded areas are the *basins of attraction* corresponding to the phases $0, 2\pi/3 \approx 2.094395$, and $-2\pi/3 \approx -2.094395$ to which the individual initial approximation is "attracted" (see [6] and Sect. 2.5.8). [RIGHT] The number of iterations needed to achieve convergence, as a function of the initial approximation

2.1.3 The Secant Method and Its Relatives

Together with bisection, the secant method and its variants are the oldest known iterative procedures. From the pairs $(x_{k-1},\ f(x_{k-1}))$ and $(x_k,\ f(x_k))$ with oppositely signed $f(x_{k-1})$ and $f(x_k)$ we determine the straight line intersecting the abscissa at x_{k+1}, and we compute the value of f at this very point, $f(x_{k+1})$. The process is repeated on the subinterval on which the function's values at its boundary points are oppositely signed, until convergence is achieved.

This method of finding the zero (regula falsi, see below) is just a special case in the family of methods based on the Newton-Raphson's iteration where the exact derivative of the function is replaced by the approximate one:

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

When this approximation is inserted in (2.7), we obtain the iteration

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})}\,. \tag{2.9}$$

To compute x_{k+1} we need the information from *two* previous points, hence this method can not be written in the one-point form $x_{k+1} = \phi(x_k)$. The iteration (2.9) can be used in various ways which yield the secant method, the method of false position (regula falsi), and the chord method. Among these, only the secant method

2.5.7 The Jenkins–Traub Method

The most efficient "sure-fire" and "black-box" method currently on the market, which is implemented in major software packages like MATHEMATICA, MATLAB or the IMSL library, is the *three-stage Jenkins-Traub method* [28]. It is well suited for the computation of zeros of real and complex polynomials and has more than quadratic order, but it is too complicated to be discussed at the level of this textbook. Even though the method is globally convergent, it is reported to become unreliable for degrees higher than $n \approx 50$ [29].

2.5.8 The Hubbard–Schleicher–Sutherland Method

Recall the example in Fig. 2.3 where the sensitivity of the Newton's method to initial conditions — its lack of global convergence — has been observed. In an exciting new development [30] it has been proven that it is possible to find *all* roots of a degree-n complex polynomial by Newton's method without recourse to deflation, starting from a set of $\mathcal{O}(n \log^2 n)$ initial points ($\mathcal{O}(n)$ if all roots are real) that can be constructed explicitly. The method works for degrees on the order of thousands or even millions [31], a task not unheard of in modern computer algebra systems and geometric modeling. In the following we quote the authors' main result in recipe form.

We construct a set of initial points I_n whose main feature is that at least one of them is in the basin of attraction of every root. It turns out that the points of this set lie along one or more concentric circles in the complex plane, the outermost of which has the radius $r(1+\sqrt{2})$, where r is an estimate for the radius of the disc containing all roots, estimated one way or another or calculated, say, by using (2.19) or (2.20). The number of required circles is

$$s = \lceil 0.26632 \log n \rceil \,,$$

and we need to place

$$N = \lceil 8.32547\, n \log n \rceil$$

points on each circle. For $n \le 42$, a single circle is needed ($s = 1$). For $n \le 1825$, $n \le 78015$ and $n \le 3\,333\,550$ one needs 2, 3 and 4 circles, respectively. The points along the circles are evenly spaced in angle. We set

$$\rho(\nu) = r\left(1+\sqrt{2}\right)\left(1-\frac{1}{n}\right)^{(2\nu-1)/4s} \,, \qquad \theta_j = \frac{2\pi j}{N} \,,$$

for $\nu = 1, 2, \ldots, s$ and $j = 0, 1, \ldots, N-1$. The initial grid then consists of sN points $\rho(\nu)\exp(\mathrm{i}\theta_j)$. An example of such a grid required to solve

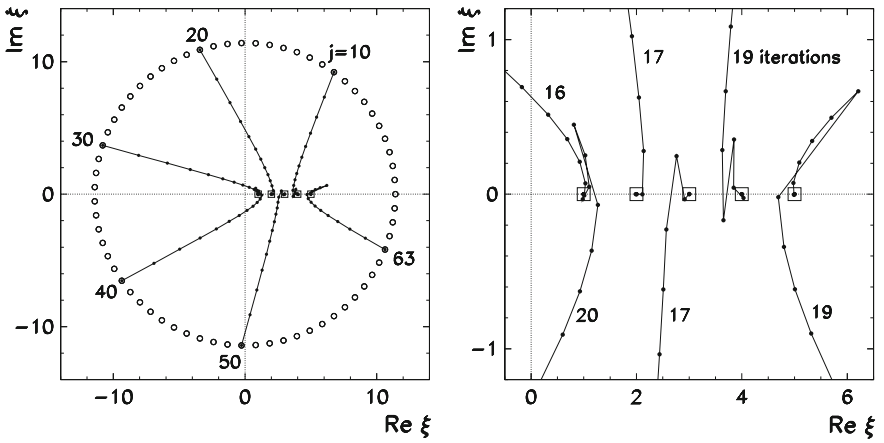


Fig. 2.11 Solving a fifth-degree polynomial equation (2.34) by using the Hubbard-Schleicher-Sutherland method. [LEFT] The initial grid I_5 consisting of $N = 67$ points along a single circle ($s = 1$) with a radius of ≈ 11.4 and some Newton "trajectories" starting from the points with angular indices $j = 10, 20, 30, 40, 50$ and 63, converging on the zeros at $\mathrm{Re}\,\xi = 4, 2, 1, 1, 3$ and 5, respectively. [RIGHT] Zoom-in of the region containing the zeros. The numbers indicate the required number of Newton iterations terminating at a tolerance below 10^{-6}

$$p(x) = \prod_{k=1}^{5}(x-k) = -120 + 274\,x - 225\,x^2 + 85\,x^3 - 15\,x^4 + x^5 = 0 \quad (2.34)$$

($n = 5, s = 1, N = 67$) is shown in Fig. 2.11 (left).

We then start the Newton's iteration from each point $z_0 \in I_n$ and make it stop when $|z_i - z_{i-1}| < \varepsilon/n$ or after at most

$$K = \lceil n \log(r/\varepsilon) \rceil$$

iterations, where ε is the desired precision of the root. (Typically $\varepsilon \approx 10^{-6}$ is taken, with an option for later root refinement.) This condition guarantees that there is at least one root ξ_j for which $|z_i - \xi_j| < \varepsilon$. If the root ξ_j approximated by z_i is different from all previous roots, store it as a valid result, otherwise discard its "trajectory" entirely. If Newton's method has been applied K times to z_0 without converging to a root, save the value z_K in a set I_n^1 for possible future use. In addition, if $|z_i| > r(1+\sqrt{2})$ for any $i > 1$ (a kind of "runaway" z), store z_i in I_n^1 as well. If all points in I_n have been tried and the number of located roots is less than n, repeat the whole procedure by taking I_n^1 as the new initial set and place any non-convergent points in the next auxiliary set I_n^2. Continue until all n roots are found.

The number of iterations required to find all roots with a precision of ε does not exceed $\mathcal{O}(n^2 \log^4 n + n \log|\log\varepsilon|)$. The number of initial points can be reduced by increasing r, but starting from large radii implies that more Newton steps need

Chapter 3
Matrix Methods

For physicist's needs, numerical linear algebra is so comprehensively covered by classic textbooks [1, 2] that another detailed description of the algorithms here would be pointless. To a larger extent than in other chapters we wish to merely set up road-signs between approaches to solving systems of linear equations, least-squares problems, and eigenvalue problems. Only Sect. 3.9 on random matrices conveys a somewhat different tone. Above all, we shall pay attention to classes of matrices for which analytic and numerical tools are particularly thoroughly developed and efficient.

To numerically solve problems in linear algebra, never write your own routines! This advice applies even to seemingly straightforward operations like matrix multiplication, and even more so to solving systems of equations $Ax = b$ or eigenvalue problems $Ax = \lambda x$. For these tasks we almost invariably use specialized libraries (see, for example, [3] and Appendix J).

3.1 Basic Operations

3.1.1 Matrix Multiplication

Classical multiplication of $n \times n$ matrices in the form

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} , \qquad i, j = 1, 2, \ldots, n , \tag{3.1}$$

where the $A_{ik} B_{kj}$ multiplication is nested into three loops over i, j, and k, requires $F = 2n^3$ multiplications or additions and $M = n^3 + 3n^2$ memory accesses. The optimal ratio is $F/M \approx n/2$ [1], so this method of multiplication is not optimal.

S. Širca and M. Horvat, *Computational Methods in Physics*,
Graduate Texts in Physics, https://doi.org/10.1007/978-3-319-78619-3_3

(Note that M may depend on processor architecture, peculiarities of the programming language, implementation, and compiler.) In standard libraries BLAS3 or LAPACK (see Appendix J) multiplication is realized in block form that also requires $F = 2n^3$ arithmetic operations, but has a better asymptotic ratio F/M for large n. For all forms of direct multiplication in floating-point arithmetic we have the estimate

$$\mathrm{fl}(AB) = AB + E\,, \qquad |E| \le n\frac{\varepsilon_\mathrm{M}}{2}|A|\,|B| + \mathcal{O}\big(\varepsilon_\mathrm{M}^2\big)\,, \tag{3.2}$$

where $|A|$ means $(|A|)_{ij} = |A_{ij}|$. We also have

$$\|\mathrm{fl}(AB) - AB\|_1 \le n\frac{\varepsilon_\mathrm{M}}{2}\|A\|_1\|B\|_1 + \mathcal{O}\big(\varepsilon_\mathrm{M}^2\big)\,. \tag{3.3}$$

Faster algorithms exist, e.g. Strassen's [4] that splits the matrices A and B into smaller blocks which are then recursively multiplied and summed. The asymptotic cost of the algorithm is $4.70\,n^{2.81}$, so it really starts to soar when used with matrices of dimensions n in the hundreds or thousands. In some cases, Strassen's method may exhibit instabilities, but robust versions are contained in the fast version BLAS3 [5] and we should use them if speed is our absolute priority. For optimally fast multiplication we need to understand the connection between hardware and software; see Fig. 3.1 and [6]. We can see in the anatomically detailed paper [7] just how deep this knowledge should be in order to design the best algorithms. For Strassen's multiplication the estimate (3.3) still applies while (3.2) is *almost always* true (see [2] and Chap. 23 in [8]).

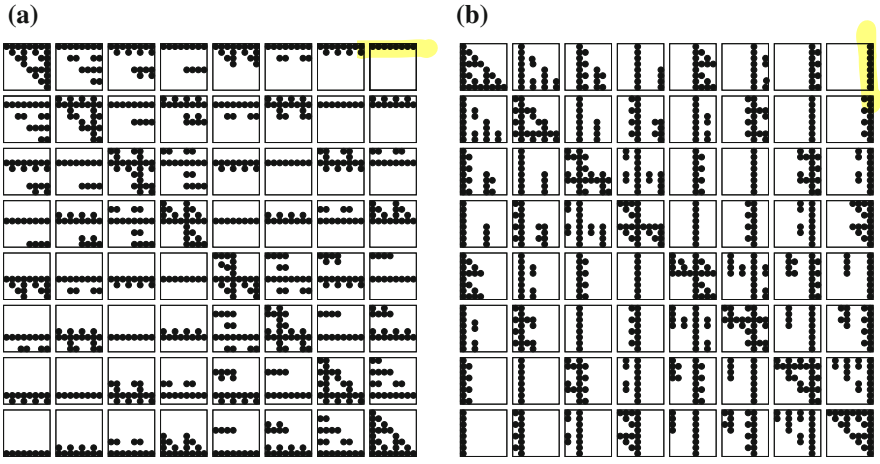


Fig. 3.1 Strassen's multiplication of 8×8 matrices. The diagrams show the elements of A and B which are accessed in memory in order to form the elements of C. For example, to get the C_{18} element (upper right corners) we need A_{1k} $(1 \le k \le 8)$ and B_{k8} $(1 \le k \le 8)$ as in classical multiplication (3.1), but a different pattern for other elements

The currently lowest asymptotic cost is claimed by the Coppersmith-Winograd algorithm [9] which is of order $\mathcal{O}(n^{2.38})$, but the error constant in front of $n^{2.38}$ is so large that the algorithm becomes useful only at very large n. A standard version is described in [10] and the best adaptive implementation in [11]. The theoretical lower limit of the numerical cost for any matrix multiplication algorithm is, of course, $2n^2$, since each element of A and B needs to be accessed at least once. See also [12–14].

3.1.2 *Computing the Determinant*

The determinant of a matrix A or its permutation PA should never be computed with the school-book method via sub-determinants. Rather, we use LU decomposition (3.7) and read off the determinant from the diagonal elements of U:

$$\det(PA) = (-1)^{\det(P)} \prod_{i=1}^{n} U_{ii} \,, \qquad \det(P) = \text{order of permutation} \,.$$

3.2 Systems of Linear Equations

This section deals with methods to solve systems of linear equations $Ax = b$, where A is a $n \times n$ matrix and b and x are vectors of dimension n. The routines from good linear algebra libraries (e.g. LAPACK) allow us to solve the system for r right-hand sides simultaneously (we are then solving $AX = B$ where B and X are $n \times r$ matrices). Solving $Ax = b$ is equivalent to "computing the inverse of A" although A^{-1} almost never needs to be explicitly known or computed. Table 3.1 summarizes some of the most widely used routines.

3.2.1 *Analysis of Errors*

Numerical errors in solving the problem $Ax = b$ can be analyzed in two ways. One way is to observe the sensitivity of the solution x to small perturbations of A or b. If the vector $\widehat{x}$ solves the perturbed equation $(A + \delta A)\widehat{x} = b + \delta b$ where $\|\delta A\| \le \varepsilon \|A\|$, $\|\delta b\| \le \varepsilon \|b\|$, and $\varepsilon \|A^{-1}\| \|A\| < 1$, the deviation $x - \widehat{x}$ from the exact solution can be bounded as

$$\frac{\|x - \widehat{x}\|}{\|\widehat{x}\|} \le \frac{\varepsilon}{1 - \varepsilon \|A^{-1}\| \|A\|} \left\{ \|A^{-1}\| \|A\| + \frac{\|A^{-1}\| \|b\|}{\|x\|} \right\} \le \frac{2\varepsilon\kappa(A)}{1 - \varepsilon\kappa(A)} \,, \quad (3.4)$$

and

$$\mathcal{K}_m(A^\dagger, w_1) = \text{span}\left\{w_1, A^\dagger w_1, (A^\dagger)^2 w_1, \ldots, (A^\dagger)^{m-1} w_1\right\} .$$

The resulting basis vectors are biorthogonal, $w_i^{\mathrm{T}} v_j = \delta_{i,j}$ $(1 \le i, j \le m)$, while (3.37) is replaced by an asymmetric transformation of the form

$$W_m^\dagger A V_m = H_m ,$$

where V_m and W_m are constructed from the v_i's and w_i's, respectively. The Hessenberg matrix H_m is tridiagonal but in general no longer real and symmetric.

3.7.3 *Preconditioning and Filtering*

Just as in the case of sparse linear solvers, the efficiency of iterative projection algorithms for solving large eigenvalue problems can be greatly improved by preconditioning. In addition, their convergence can be accelerated by a technique called polynomial filtering: the idea behind it is to pre-process the initial vectors or the initial Krylov subspace in order to reduce their components in the unwanted parts of the spectrum relative to those in the wanted parts. For details see Chaps. 7 and 8 of [68].

3.8 Pseudospectra of Matrices ⋆

The concept of pseudospectra is useful in the analysis of specific mathematical problems and physical systems in which the standard eigenvalue methods may fall short of completely explaining their known or presumed properties and behavior. Most often this can be witnessed in systems represented by matrices which are highly *non-normal*. Normal matrices are diagonalizable, i.e. they can be decomposed by (3.27) and hence possess a complete set of orthogonal eigenvectors such that $X^{-1} = X^{\mathrm{T}}$ or $X^{-1} = X^\dagger$, and the corresponding set of eigenvalues, the spectrum $\sigma(A)$. *Non-normality*, on the other hand, refers to the fact that the eigenvectors of A, if they exist, are *far from orthogonal*: this translates into the statement that A is far from normal when $\|X\|\|X^{-1}\| \gg 1$ in some matrix norm. In this section we assume $\|\cdot\| = \|\cdot\|_2$.

3.8.1 *Definition of Pseudospectrum*

For arbitrary (usually small) $\varepsilon > 0$, the ε-pseudospectrum $\sigma_\varepsilon(A)$ of $A \in \mathbb{C}^{n\times n}$ is the set of points $z \in \mathbb{C}$ for which

$$\|(A - zI)^{-1}\| > \varepsilon^{-1} , \tag{3.39}$$

i.e. the norm of the resolvent of A at z exceeds some (usually large) value. Note that the proper spectrum implies $z \in \sigma(A) \Leftrightarrow \|(A - zI)^{-1}\| = \infty$. Indeed, for normal matrices $\|(A - zI)^{-1}\|$ is always large when z is near an eigenvalue. What makes non-normal matrices so special is the fact that this norm can be large even when z is far away from the spectrum, with important consequences. For instance, the transient (non-asymptotic) behavior of a non-normal system may be driven by the pseudospectrum, not the genuine eigenvalues alone. An example will be shown in the discussion of stability of stiff differential equations in Sect. 7.10, see also Fig. 11.8 (left). Contrary to common experience, the eigenvalues may also fail to predict the correct asymptotic and resonance behavior of such systems when variable coefficients, non-linearities or complicated force terms (right-hand sides) are present. A convincing motivation for pseudospectra and the proof of their all-pervading importance in wide areas of physics is offered by the landmark (and almost inexhaustible) monograph [69].

Since the eigenvalues may be extremely sensitive to perturbations (see Sect. 3.6.1), an equivalent definition of the pseudospectrum can be based on what happens to the eigenvalues of A when its matrix elements are changed by a small amount: the pseudospectrum $\sigma_\varepsilon(A)$ of A is the set of $z \in \mathbb{C}$ such that

$$z \in \sigma(A + E) , \tag{3.40}$$

where $E \in \mathbb{C}^{n\times n}$ is any small perturbation with $\|E\| < \varepsilon$. In other words, if one could generate all perturbations of A with $\|E\| < \varepsilon$, they would fill the area enclosed by the ε-boundary of the pseudospectrum.

Example (Adapted from [69].) Consider the tridiagonal non-symmetric Toeplitz matrix

$$A = \begin{pmatrix} 0 & 1 & & & \\ \frac{1}{4} & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{4} & 0 & 1 \\ & & & \frac{1}{4} & 0 \end{pmatrix} . \tag{3.41}$$

It can be symmetrized by $B = DAD^{-1}$, where $D = \mathrm{diag}\,(2, 4, \ldots, 2^n)$, yielding

$$B = \begin{pmatrix} 0 & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{2} & 0 & \frac{1}{2} \\ & & & \frac{1}{2} & 0 \end{pmatrix} . \tag{3.42}$$

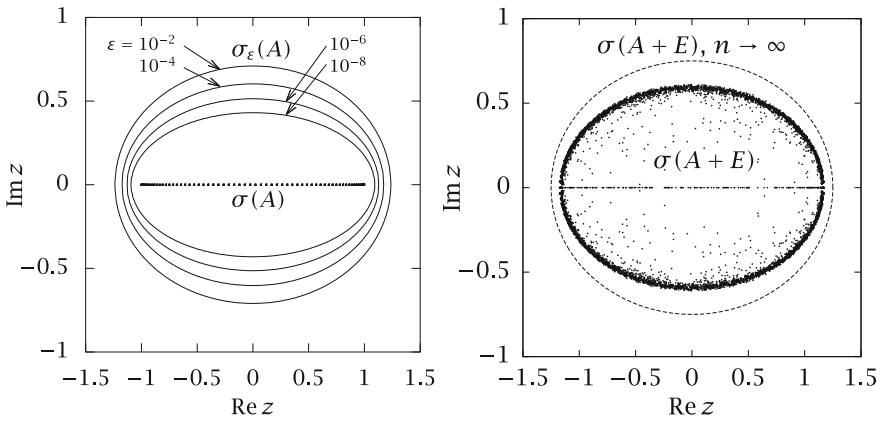


Fig. 3.9 [LEFT] The spectrum $\sigma(A)$ (dots on the real axis) and the pseudospectra $\sigma_\varepsilon(A)$ (ellipses) of the matrix (3.41) of dimension $n = 64$. [RIGHT] Superposition of eigenvalues of 100 matrices $A + E$, where E is a random matrix with $\|E\| = 0.001$

The similarity transformation preserves the eigenvalues, hence the spectra of A and B are identical, consisting of n distinct numbers on the real axis:

$$\lambda_i(A) = \lambda_i(B) = \cos\frac{i\pi}{n+1}\,, \qquad 1 \le i \le n \tag{3.43}$$

(see (A.7)). The pseudospectra of A, however, reside far from the real axis. Figure 3.9 (left) shows the boundaries of $\sigma_\varepsilon(A)$ with $n = 64$ for several ε. We see that even at distances from the spectrum on the order of 1 the norm of the resolvent remains very large.

Figure 3.9 (right) illustrates the pseudospectrum of A according to the alternative definition (3.40). The plot shows a superposition of 6400 pseudoeigenvalues at $\varepsilon = 0.001$ generated by calculating the eigenvalues of 100 matrices $A + E$, where E is a random matrix whose elements are independent normal random deviates. The matrix E has been normalized such that $\|E\| = 0.001$.

Figure 3.9 is just a visualization of the pseudospectrum. But what does it *mean* for the behavior of A and B? Suppose that we wish to predict the norms of the powers A^k and B^k for various k. The classical approach to this problem is to resort to eigenvalues: first diagonalize A via $X^{-1}AX = \Lambda$, then, since taking the power of a diagonal matrix is trivial, calculate

$$\Lambda^k = \left(X^{-1}AX\right)^k = X^{-1}AAA\cdots AX = X^{-1}A^kX\,,$$

and, finally, "un-diagonalize" Λ^k to obtain $A^k = X\Lambda^kX^{-1}$, and analogously for B^k. Furthermore, by (3.43), the spectral radii (the maximum absolute eigenvalues) of A and B are equal, $\rho(A) = \rho(B) = \cos(\pi/(n+1))$. Since these quantities are smaller than 1, by the procedure outlined above both A and B are power-bounded as

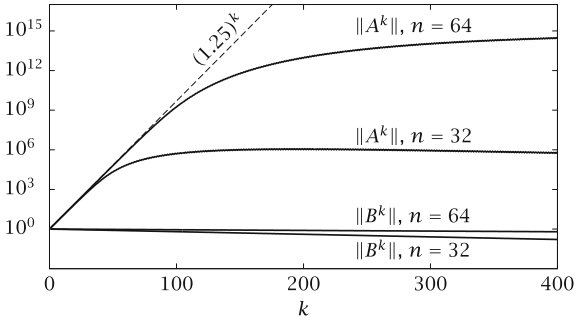


Fig. 3.10 The norms of the powers A^k and B^k of matrices (3.41) and (3.42) as functions of k, for $n = 32$ and $n = 64$

$\|A^k\| \leq C_A$ and $\|B^k\| \leq C_B$ for all $k \geq 0$, with $\|A^k\| \to 0$ and $\|B^k\| \to 0$ as $k \to \infty$. Figure 3.10 confirms these expectations, yet it is obvious that the behavior of the powers of the non-normal matrix A and of the symmetric (and therefore normal) matrix B generated from it are completely different — even though their spectra are identical.

The pseudospectra help to elucidate the most interesting "transient" region where $k < n$. Here $\|A^k\|$ grows exponentially, with the slope in the log-linear plot given by $(1.25)^k$. The value 1.25 corresponds to the rightmost point of the dashed ellipse in Fig. 3.9 (right), i.e. to the absolutely maximum pseudoeigenvalue. It is precisely this pseudospectral radius — rather than the eigenvalues — that drives the behavior of $\|A^k\|$ for moderate k. ◁

3.8.2 *Pseudospectra of Linear Operators*

The concept of matrix pseudospectra can be generalized to linear operators acting in infinitely dimensional spaces, for instance, in complete normed (Banach) spaces. Assuming that the inverse of such an operator A is bounded, its ε-pseudospectrum $\sigma_\varepsilon(A)$ can still be defined as the set of points $z \in \mathbb{C}$ such that

$$\|(A - z)^{-1}\| > \varepsilon^{-1}$$

by analogy to (3.39), but $\|\cdot\|$ is now an operator norm (see Appendix A.4). The most common operators of this kind are differential or integral operators. The simplest example is the derivative operator $A = \mathrm{d}/\mathrm{d}x$ acting in the space of functions $u \in L^2(0, 1)$,

$$Au = u' = \frac{\mathrm{d}u}{\mathrm{d}x}\,, \tag{3.44}$$

and subject to the boundary condition $u(1) = 0$. The eigenfunctions of this differential operator would have to have the form $u(x) = \mathrm{e}^{zx}$ for some $z \in \mathbb{C}$, but the boundary condition forbids any solution of this kind. Hence the spectrum of A is

approximation of the continuous Fourier transform of a periodic function f on the interval $[0, 2\pi]$:

$$\frac{1}{2\pi}\int_0^{2\pi} f(x)\,\mathrm{e}^{-\mathrm{i}kx}\,\mathrm{d}x = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)\,\mathrm{e}^{-\mathrm{i}2\pi jk/N} + \mathcal{R}_N\,,$$

where $\mathcal{R}_N = -f''(\xi)(2\pi)^2/(12N^2)$ and $\xi \in [0, 2\pi]$.

4.2.3 Aliasing

The coefficients of the discrete Fourier transform (4.11) and the coefficients of the exact expansion (4.7) are related by

$$\widetilde{f}_k = \widehat{f}_k + \sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} \widehat{f}_{k+Nm}\,, \qquad k = -N/2, \ldots, N/2-1\,. \tag{4.16}$$

By using (4.9) and (4.13) this can be written as $I_N f = S_N f + R_N f$. The remainder

$$R_N f = I_N f - S_N f = \sum_{k=-N/2}^{N/2-1} \left(\sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} \widehat{f}_{k+Nm} \right) \phi_k \tag{4.17}$$

is called the *aliasing error* and it measures the difference between the interpolation polynomial and the truncated Fourier series. Aliasing implies that the Fourier component with the wave-number $(k + Nm)$ behaves like the component with the wave-number k. Because the basis functions are periodic,

$$\phi_{k+Nm}(x_j) = \phi_k(x_j)\,, \qquad m \neq 0\,,$$

such components are indistinguishable (Fig. 4.4). In other words, on a discrete mesh, the kth Fourier component of the interpolant $I_N f$ depends not only on the kth component of f, but also on the higher components mimicking the kth.

The aliasing error is orthogonal to the truncation error $f - S_N f$, thus

$$\|f - I_N f\|^2 = \|f - S_N f\|^2 + \|R_N f\|^2\,.$$

The interpolation error is therefore always larger than the truncation error.

Example Knowing how to control aliasing has important practical consequences. The signals on standard audio compact discs are sampled at the frequency of $\nu_\mathrm{s} = 44.1\,\mathrm{kHz}$. The critical frequency (4.6) is then $\nu_\mathrm{c} = \nu_\mathrm{s}/2 = 22.05\,\mathrm{kHz}$. Such fine sampling prevents aliasing in the audible part of the spectrum and there is no distortion

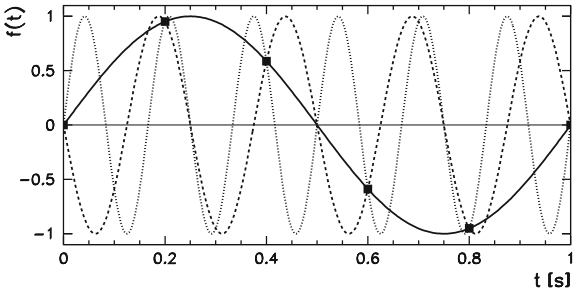


Fig. 4.4 A periodic signal $\sin(2\pi\nu t)$ with the frequency $\nu = 1\,\mathrm{s}^{-1}$ (full line) is sampled at $\nu_\mathrm{s} = 5\,\mathrm{s}^{-1}$ (squares): the critical frequency (4.6) is $\nu_\mathrm{c} = \omega_\mathrm{c}/(2\pi) = 2.5\,\mathrm{s}^{-1}$. We obtain exactly the same function values by sampling the functions $\sin(2\pi(\nu - \nu_\mathrm{s})t)$ (dashed line) or $\sin(2\pi(\nu + \nu_\mathrm{s})t)$ (dotted line). At this sampling rate, aliasing causes all three signals to be represented with the frequency ν in the discrete Fourier spectrum

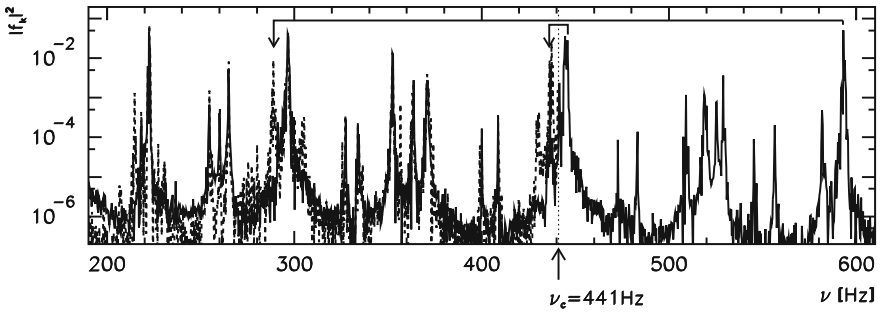


Fig. 4.5 Frequency spectrum of the concluding chord of the Toccata and Fugue for organ in *d*-minor, BWV 565, of J. S. Bach. Full curve: sampling at 44.1 kHz. Dashed line: sampling at a smaller frequency 882 Hz causes aliasing. The arrows indicate the peak at 593 Hz which is mirrored across the critical frequency $\nu_\mathrm{c} = 441$ Hz onto the frequency $(441 - (593 - 441))$ Hz $= 289$ Hz, and the peak at 446 Hz which maps onto 436 Hz

of the signal. Had we wished, however, to compress the signal, we should not do this by simply decreasing the sampling frequency, since this would map the high-frequency components into the low-frequency part of the spectrum and the signal would be distorted. Instead, we should use a filter to remove the high-frequency part of the spectrum and only then down-sample. Figure 4.5 shows the appearance of aliasing in the frequency spectrum of an acoustic signal sampled at 44.1 and 882 Hz without filtering. ◁

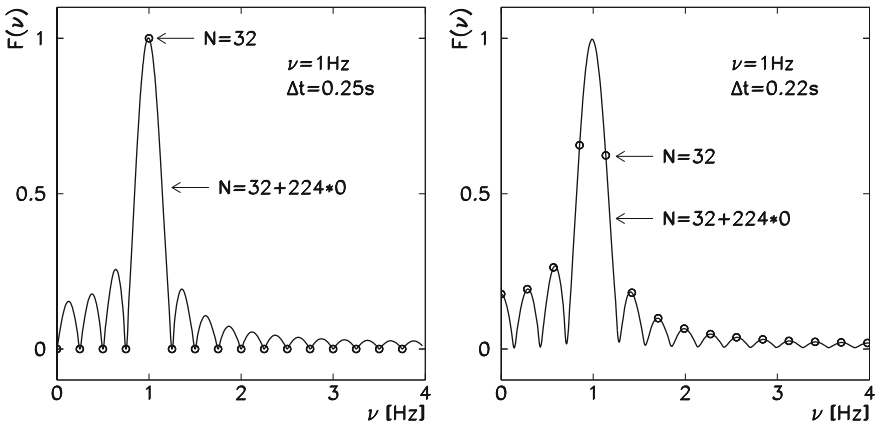


Fig. 4.6 Leakage in the frequency spectrum in the discrete Fourier transform of a sine wave with the frequency 1 Hz. [LEFT] Sampling at $N = 32$ points 0.25 s apart encompasses precisely four complete waves. The only non-zero component of the transform is the one corresponding to the frequency of 1 Hz. [RIGHT] Sampling at $N = 32$ points 0.22 s apart covers only 3.52 waves. Many non-zero frequency components appear. The curves connect the discrete transform of the same signals, except that the 32 original samples of the signal are followed by 224 zeros (total of 256 points). Adding zeros in the temporal domain is known as *zero padding* and improves the resolution in the frequency domain. In the limit $N \to \infty$ we approach the continuous Fourier transform

4.2.4 *Leakage*

The discrete Fourier transformation of realistic signals of course involves only finitely many values. From an infinite sequence we pick (multiply by one) only a sample of length N, whereas the remaining values are dropped (multiplied by zero). Due to this restriction, known as *windowing*, the frequency spectrum exhibits the *leakage* phenomenon shown in Fig. 4.6 (adapted from [4]). To some extent, leakage can be controlled by using more sophisticated *window functions* that engage a larger portion of the signal and smoothly fade out instead of crude multiplication of the signal by one and the remainder by zero. The advantages and weaknesses of some classical window functions are discussed in [5].

4.2.5 *Fast Discrete Fourier Transformation (FFT)*

The discrete Fourier transformation (4.11) is a mapping between the vector spaces of dimensions N, $\mathcal{F}_N : \mathbb{C}^N \to \mathbb{C}^N$. Let us rewrite it in a more transparent form,

$$F = \mathcal{F}_N[f]\,, \qquad F_k = \frac{1}{N}\sum_{j=0}^{N-1} f_j\, \mathrm{e}^{-2\pi \mathrm{i} jk/N}\,, \tag{4.18}$$

where we have denoted $f = \{f_j\}_{j=0}^{N-1}$ and $F = \{F_k\}_{k=0}^{N-1}$. Note that the indices j and k run symmetrically, both from 0 to $N-1$. The inverse transformation is

$$f = \mathcal{F}_N^{-1}[F]\,, \qquad f_j = \sum_{k=0}^{N-1} F_k\, \mathrm{e}^{2\pi \mathrm{i} jk/N}\,.$$

Then (4.18) can be written as

$$F_k = \frac{1}{N} \sum_{j=0}^{N-1} W_N^{kj} f_j\,, \qquad W_N = \mathrm{e}^{-2\pi \mathrm{i}/N}\,. \tag{4.19}$$

To evaluate the DFT by computing this sum we need $\mathcal{O}(N^2)$ operations. But precisely the same result can be achieved with far fewer operations by using the Cooley-Tukey algorithm. Let N be divisible by m. Then the sum can be split into m partial sums, and each of them runs over the elements f_j of the array f with the same modulus of the index j mod m:

$$F_k = \frac{1}{N} \sum_{l=0}^{m-1} W_N^{kl} \sum_{j=0}^{N/m-1} W_{N/m}^{kj} f_{mj+l}\,.$$

Let us denote by $f^{(l)} = \{f_{mj+l}\}_{j=0}^{N/m-1}$ the components of the array f which have the same modulus of the index with respect to m. We have thus recast the transform of the original array f of dimension N as a sum of m transforms of the shorter arrays $f^{(l)}$ of dimension N/m. This can be written symbolically as

$$\mathcal{F}_N[f]_k = \frac{1}{N} \sum_{l=0}^{m-1} W_N^{kl} \left(\frac{N}{m} \mathcal{F}_{N/m}[f^{(l)}] \right)_k\,.$$

This is a recursive computation of the DFT for the array f that follows the idea of *divide-and-conquer* algorithms. The array f for which the DFT should be computed is gradually broken down into sub-arrays, thus reducing the amount of necessary work. The optimal factorization is $N = 2^p$ ($p \in \mathbb{N}$) in which at each step the array is split into two sub-arrays containing elements of the original array with even and odd indices, respectively. This method requires $\mathcal{O}(N \log_2 N)$ operations for the full DFT instead of $\mathcal{O}(N^2)$ by direct summation, lending it the name *Fast Fourier Transformation*. Similar speeds are attainable by factorizing N to primes, e.g.

$$N = 2^{p_1} 3^{p_2} 5^{p_3} 7^{p_4}\,, \qquad p_i \in \mathbb{N}\,,$$

which is supported by all modern FFT libraries. Good implementations of the FFT are complicated, as the factorization should be carefully matched to the addressing of the arrays. The most famous library is the multiple-award winning FFTW3 (Fastest

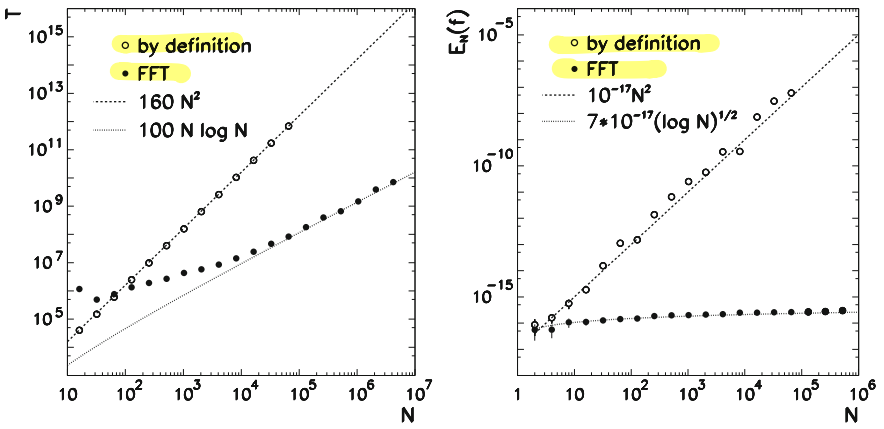


Fig. 4.7 [LEFT] The numerical cost (number of CPU cycles T) of the computation of the DFT by the basic definition (4.19) and by using the FFT, as a function of the sample size N. [RIGHT] The average measure of deviation $E_N(h)$ for the computation of the DFT by definition and by using the FFT

Fourier Transform in the West) [6]; see also [7, 8]. A comparison of the CPU cost of the standard DFT and FFT is illustrated in Fig. 4.7 (left).

Example Due to the fewer operations, FFT is not only essentially faster than the naive DFT; it is also more precise, as can be confirmed by a numerical experiment. We form the array $f = \{f_0, f_1, \ldots, f_{N-1}\}$ of random complex numbers. We apply the DFT to compute the transform of f, to which we apply the inverse DFT. Finally, we compute the deviation of the resulting array from the original array:

$$\Delta f = (\mathcal{F}_N^{-1} \circ \mathcal{F}_N) f - f \ .$$

In arithmetic with precision ε, we get $\Delta f \neq 0$. We define the average deviation as $E_N(f) = \langle \|\Delta f\|_2 / \|f\|_2 \rangle$, where the average $\langle \cdot \rangle$ is over a large set of random arrays. The results are shown in Fig. 4.7 (right). When the DFT is computed by (4.18), we get $E_N(f) \sim \mathcal{O}(\varepsilon N^2)$, while the FFT gives $E_N(f) \sim \mathcal{O}(\varepsilon \sqrt{\log N})$ [9]. In short, the FFT algorithm is unbeatable! All decent numerical libraries support the computation of the DFT by FFT algorithms (see Appendix J). ◁

4.2.6 *Multiplication of Polynomials by Using the FFT*

Multiplication of polynomials is one of the tasks in computing with power bases, e.g. in expansions in powers of the perturbation parameters in classical and quantum mechanics. The multiplication of $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{m} b_i x^i$ in the form

$$q(x)p(x) = \sum_{i=0}^{n+m} c_i x^i \,, \qquad c_i = \sum_{k=0}^{i} a_k b_{i-k} \,,$$

requires $\mathcal{O}((n+1)(m+1))$ operations to determine the coefficients c_i. If the number of terms is large ($n, m \gg 1$) this process is slow and prone to rounding errors. A faster and a more precise way is offered by the FFT. We form two arrays of length $N = m + n + 1$. The coefficients of the polynomials p and q are stored at the beginning of these arrays while the remaining elements are set to zero:

$$A = \{A_i\}_{i=0}^{N-1} = \{a_0, \ldots, a_n, \underbrace{0, \ldots, 0}_{m}\} \,, \qquad B = \{B_i\}_{i=0}^{N-1} = \{b_0, \ldots, b_m, \underbrace{0, \ldots, 0}_{n}\} \,.$$

The coefficients of the product are given by the convolution

$$c_i = \sum_{k=0}^{N-1} A_k B_{i-k} \,, \qquad i = 0, 1, \ldots, N-1 \,,$$

where we assume periodic boundary conditions, $A_k = A_{N+k}$, $B_k = B_{N+k}$. The convolution is then evaluated by first computing the FFT of the arrays A and B,

$$\hat{A} = \{\hat{A}_i\}_{i=0}^{N-1} = \mathcal{F}_N[A] \,, \qquad \hat{B} = \{\hat{B}_i\}_{i=0}^{N-1} = \mathcal{F}_N[B] \,,$$

multiplying the transforms component-wise into a new array $\hat{C} = \{\hat{A}_i \hat{B}_i\}_{i=0}^{N-1}$, and finally compute its inverse FFT,

$$C = \{C_i\}_{i=0}^{N-1} = N\mathcal{F}_N^{-1}[\hat{C}] \,.$$

This procedure has a numerical cost of $\mathcal{O}(N \log_2 N)$ which, for $n, m \gg 1$, is much smaller than the cost of directly computing the sums of the products.

4.2.7 Power Spectral Density

The Fourier transformation can be seen as a decomposition of a signal to a linear combination of the functions $A_\omega e^{i\omega x}$. The quantity $|A_\omega|^2$ is the *signal power* at the given frequency ω. If we are dealing with real signals, we are mostly interested in the total power at the absolute value of the frequency, $|A_\omega|^2 + |A_{-\omega}|^2$ for $\omega \geq 0$.

For a continuous signal f with the Fourier transform (4.1), we define the *double-sided power spectral density* (PSD) as

$$S(\omega) = |F(\omega)|^2 \,, \qquad \omega \in \mathbb{R} \,,$$

while the single-sided power spectral density is

$$S(\omega) = |F(-\omega)|^2 + |F(\omega)|^2 \ , \qquad \omega \in \mathbb{R}_+ \ .$$

Often, the double-sided spectral density of a signal is defined via the single-sided density in which the power of a component with the frequency ω is equal to the power of the component with the frequency $-\omega$, and then $S(\omega) = 2|F(\omega)|^2$.

For discrete data $\{f_j\}$ with the transform (4.18) the discrete double-sided power spectral distribution $\{S_k\}$ is defined analogously to the continuous case,

$$S_k = |F_k|^2 \ , \qquad k = 0, 1, \ldots, N-1 \ .$$

The quantity S_k measures the power of the signal at the frequency $2\pi k/N$, while S_{N-k} corresponds to the frequency $-2\pi k/N$, where $k = 0, 1, \ldots, N/2 - 1$. For the single-sided distribution we sum over the powers of negative and positive frequencies. For odd N, the single-sided distribution $\{S_k\}$ is defined as

$$\begin{aligned} S_0 &= |F_0|^2 \ , \\ 2S_k &= |F_k|^2 + |F_{N-k}|^2, \qquad k = 1, 2, \ldots, (N-1)/2 \ , \end{aligned}$$

while for even N it is given by

$$\begin{aligned} S_0 &= |F_0|^2 \ , \\ 2S_k &= |F_k|^2 + |F_{N-k}|^2 \ , \qquad k = 1, 2, \ldots, N/2 \ , \\ S_{N/2} &= |F_{N/2}|^2 \ . \end{aligned}$$

In the discrete case, Parseval's equality applies in the form

$$\frac{1}{N} \sum_{j=0}^{N-1} |f_j|^2 = \sum_{k=0}^{N-1} |F_k|^2 \ .$$

4.2.8 *Sparse FFT*

In many applications including audio and video compression, magnetic resonance imaging, radio astronomy and global positioning systems the overwhelming majority of the Fourier coefficients of a signal are small or equal to zero. While the standard FFT is insensitive to this sparseness of the spectrum and grinds away at its immutable $\mathcal{O}(N \log N)$, a "sparse FFT" should be able to exploit precisely this feature. In other words, given a N-dimensional vector (set of complex values) f with the Fourier transform F, a sparse FFT algorithm should produce an approximation $\widehat{F}$ to F that satisfies

$$\|F - \widehat{F}\|_2 \leq C \min_{k-\text{sparse } G} \|F - G\|_2 \ ,$$

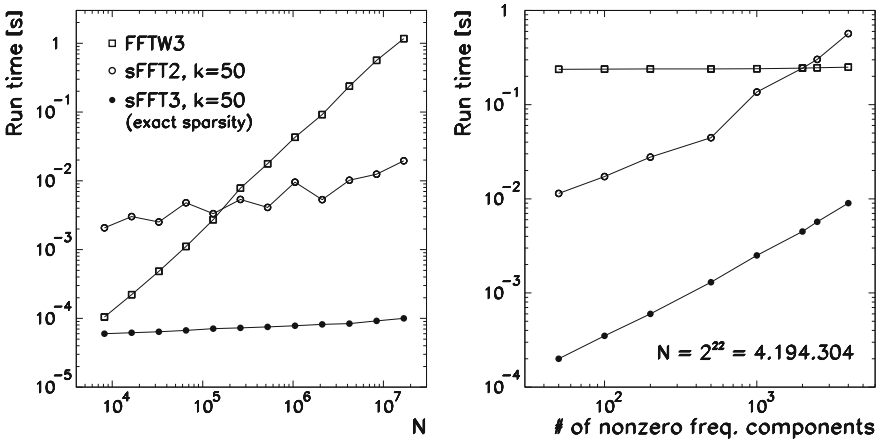


Fig. 4.8 [LEFT] Average times needed to compute the discrete Fourier transforms of length-N signals with sparse ($k = 50$) spectra by using FFTW3 [6], sFFT2 and sFFT3. [RIGHT] Average computation times as functions of the number of non-zero frequency components at fixed signal size N. (Notation as in the left panel.)

where C is a constant and the minimization runs over k-sparse transforms G, i. e. transforms with k non-zero frequency components. Transforms possessing at most k non-zero components are called *exactly sparse*; if the other $N - k$ components still represent a non-zero, yet relatively much smaller fraction of the signal power, the transform is said to be *approximately sparse* — the latter being the usual case due to the presence of noise.

Sparse Fourier transformations have a long history, but in an interesting recent development, the authors have shown that such transforms can be obtained at a computational cost of $\mathcal{O}(\log N\sqrt{Nk\log N})$ or $\mathcal{O}(k\log N\log(N/k))$ in the approximately sparse case, depending on the details of the implementation (see [10–12], sFFT2 library), and as low as $\mathcal{O}(k\log N)$ in the exactly sparse case [12]. Taking architecture-specific details into account, the performance can be further improved (see [13, 14], sFFT3 library, exactly sparse only). Figure 4.8 shows the average computation times for the evaluations of the sparse FFT by using the FFTW3, sFFT2 and sFFT3 libraries.

4.2.9 *Non-uniform (Non-equispaced) FFT*

Unevenly spaced time series are ubiquitous in science, in particular in astrophysics and seismology. There the signals come at times which are beyond our control, as in earthquakes. It may also happen that the detectors are inoperational for a fraction of the time, or that certain data values are flagged as suspicious and must be discarded. An example of such a signal is given in Fig. 4.9. Simplistic approaches to dealing

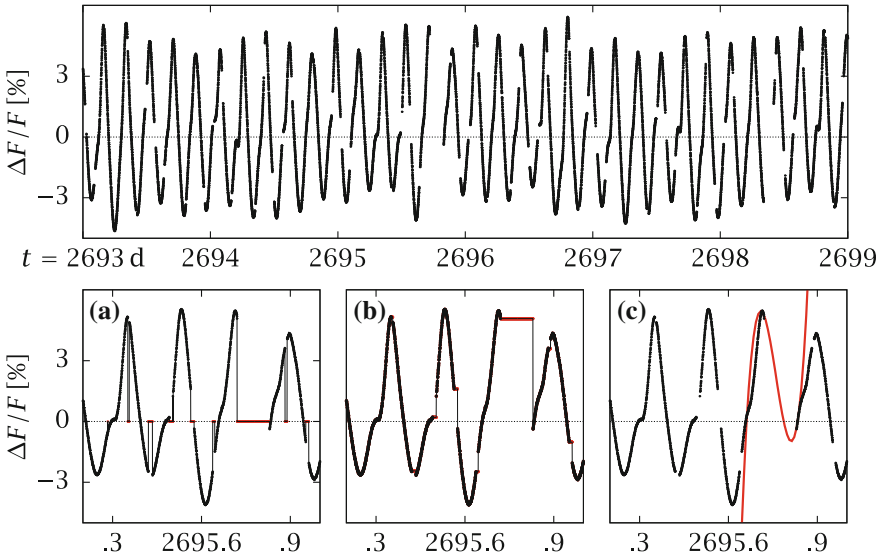


Fig. 4.9 [TOP] A part of the measured light curve (normalized flux F) of the variable star HD 180642 acquired by the CoRoT satellite [15]. [BOTTOM] Naive (and inappropriate) strategies (plotted in red) to cope with the missing data: (**a**) setting them to zero, (**b**) "clamping" to the last known value, (**c**) a dangerous step into the dark by cubic interpolation

with the missing data problem—illustrated in the bottom panel—do not work. Large gaps in the data (case (a), for instance) tend to amplify the low-frequency portion of the power spectrum, corresponding to wavelengths on the order of the gap size.

The most popular and reliable spectral analysis method for unevenly sampled data $\{f_i\}_{i=0}^{N-1}$ acquired at arbitrary times $\{t_i\}_{i=0}^{N-1}$ is to construct the Lomb-Scargle power spectrum (periodogram) [16]

$$P_{\mathrm{LS}}(\omega) = \frac{1}{2\sigma^2}\left\{\frac{\left[\sum_i (f_i - \overline{f})\cos\omega(t_i-\tau)\right]^2}{\sum_i \cos^2\omega(t_i-\tau)} + \frac{\left[\sum_i (f_i - \overline{f})\sin\omega(t_i-\tau)\right]^2}{\sum_i \sin^2\omega(t_i-\tau)}\right\}, \tag{4.20}$$

where

$$\overline{f} = \frac{1}{N}\sum_{i=0}^{N-1} f_i\,, \qquad \sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}\left(f_i - \overline{f}\right)^2,$$

are the signal mean and variance, respectively, and the parameter τ is defined by

$$\tan 2\omega\tau = \frac{\sum_i \sin 2\omega t_i}{\sum_i \cos 2\omega t_i}\,.$$

The offset τ makes $P_{\mathrm{LS}}(\omega)$ independent of shifting all the t_i's by a constant. Moreover, introducing τ in this manner causes the linear least-squares fit of the data to the model $f(t) = A\cos\omega t + B\sin\omega t$ to yield precisely (4.20).

As in the naive DFT, the numerical cost of calculating $P_{\mathrm{LS}}(\omega)$ by direct evaluation of the sums is $\mathcal{O}(N_\omega N)$, where N_ω is the number of desired frequencies in the spectrum. This obstacle can be overcome by a much faster procedure. Defining

$$\begin{aligned} C_f &= \textstyle\sum_i \big(f_i - \overline{f}\big)\cos\omega t_i\ , & S_f &= \textstyle\sum_i \big(f_i - \overline{f}\big)\sin\omega t_i\ , \\ C_2 &= \textstyle\sum_i \cos 2\omega t_i\ , & S_2 &= \textstyle\sum_i \sin 2\omega t_i\ , \end{aligned} \tag{4.21}$$

it holds that

$$\begin{aligned} \textstyle\sum_i \big(f_i - \overline{f}\big)\cos\omega(t_i-\tau) &= C_f\cos\omega\tau + S_f\sin\omega\tau\ , \\ \textstyle\sum_i \big(f_i - \overline{f}\big)\sin\omega(t_i-\tau) &= S_f\cos\omega\tau - C_f\sin\omega\tau\ , \\ \textstyle\sum_i \cos^2\omega(t_i-\tau) &= \frac{N}{2} + \frac{C_2}{2}\cos 2\omega\tau + \frac{S_2}{2}\sin 2\omega\tau\ , \\ \textstyle\sum_i \sin^2\omega(t_i-\tau) &= \frac{N}{2} - \frac{C_2}{2}\cos 2\omega\tau - \frac{S_2}{2}\sin 2\omega\tau\ . \end{aligned} \tag{4.22}$$

Note that if the t_i's were evenly spaced, one could calculate C_f, S_f, C_2 and S_2 by two complex FFTs, plug the results into (4.22) and use these to compute (4.20). The key question, therefore, is how to evaluate the sums of (4.21) for arbitrarily spaced data. This is accomplished by fast non-uniform FFT algorithms (NUFFT) with typical numerical costs of $\mathcal{O}(N\log N)$ or $\mathcal{O}(N\log N + |\log\varepsilon|N)$, where ε is the desired numerical precision.

Several types of NUFFT are available, depending on whether the sampling in the configuration space (time, coordinate) or in the transform space (frequency)—or both—are non-uniform, since the forward and the inverse transformation happen to be not simply the two sides of the same coin as in the standard FFT. An introduction to various classes of algorithms is given in [17, 18], see also [19–21]. Most methods rely on some sort of local rearrangement of the data from an irregular grid onto a regular one, in conjunction with the classic FFT to evaluate the intermediate sums. In the classic `fasper` routine of the NR3E library based on [22] this rearrangement is accomplished by inverse interpolation. In an alternative approach, trigonometric polynomials $p(x) = \sum_k p_k \exp(-2\pi\mathrm{i}kx)$ are approximated by linear combinations of translated window functions φ that are well localized in both configuration and transform spaces; the core of the algorithm is again the transformation of Fourier integrals of φ (calculated analytically) by means of the FFT. This is at the heart of NFFT3 [23], presently the most comprehensive and fastest NUFFT package on the market based on the FFTW3 library and elaborated in more detail in [24–26]. For NFFT3 at work in the astrophysical example mentioned above, see [27].

the description of the long tails. A detailed discussion of the variants of the Hilbert transform with emphasis on signal processing applications can be found in [44, 56, 57] and in the monumental work [58].

4.6 Continuous Wavelet Transformation ⋆

We may think of the *wavelet transformation* of a signal [59–62] as an extension of its Fourier analysis, through which not only the strengths of the signal's frequency components are determined, but also the times at which these components occur. The classic example in Fig. 4.19 illustrates this basic idea for the signal $\sin(t^2)$ whose frequency linearly increases with time. By using the wavelet transformation we can also locate changes in the signal that are not immediately apparent from its temporal behavior alone (Fig. 4.20).

The *continuous wavelet transform* (CWT) of the function f is defined as

$$L_\psi[f](s,t) = \frac{1}{\sqrt{c_\psi s}} \int_{-\infty}^{\infty} f(\tau)\, \psi^*\left(\frac{\tau - t}{s}\right) \mathrm{d}\tau\,, \qquad t \in \mathbb{R}\,, \quad s \neq 0\,,$$

where t is the time at which a feature of scale s is observed in the function f, and c_ψ is the normalization constant. The function ψ, whose properties are given in the

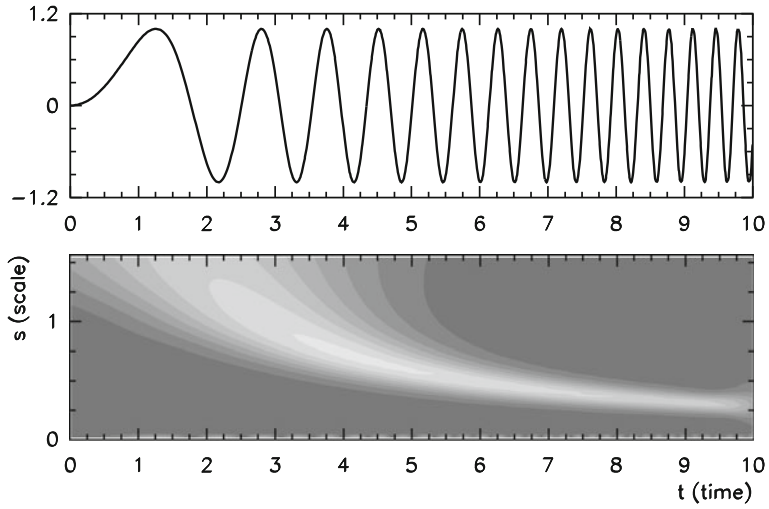


Fig. 4.19 The basic idea of the continuous wavelet transform. [Top] The signal $f(t) = \sin \omega t$, $\omega \propto t$. [Bottom] In this portion of the signal, the continuous wavelet transformation detects large structures at short times (where the waves have a typical scale $s \approx 1.5$) and small structures at long times (scale $s \approx 0.3$). The frequency and the scale of the oscillations are inversely proportional, which generates the typical curvature of the transform ($s \propto \omega^{-1} \propto t^{-1}$)

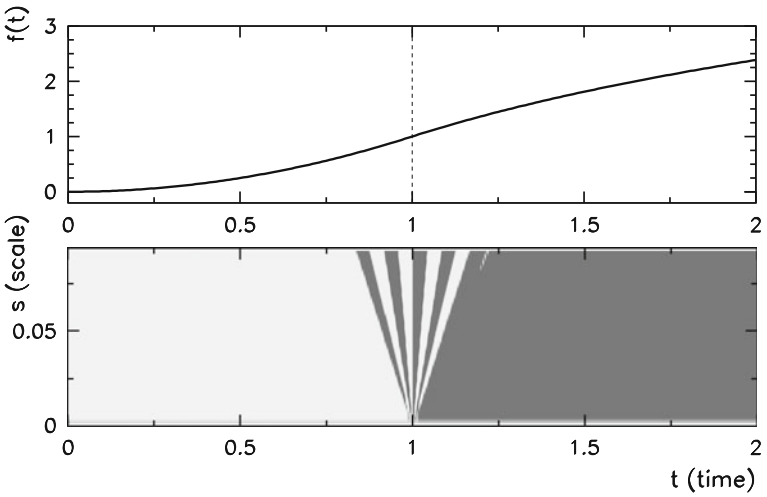


Fig. 4.20 Continuous transform of a real signal with a complex Morlet wavelet (4.71) [TOP] The signal $f(t) = t^2$ $(t < 1)$ or $f(t) = 1 + 2\log t$ $(t \geq 1)$ is continuous and has a continuous first derivative at $t = 1$ while its second derivative is discontinuous. [BOTTOM] The phase of the wavelet transform in the vicinity of $t = 1$ oscillates a couple of times, and reveals the location of the critical point when the scale s is decreased

following, should allow us to change the parameter s (the typical scale of a structure in the signal f) as well as its shift t with respect to the signal f. By denoting

$$\psi_s(t) = \psi^*(-t/s)$$

we can rewrite the definition in the form of a convolution

$$L_\psi[f](s,t) = \frac{1}{\sqrt{c_\psi\, s}} \int_{-\infty}^{\infty} f(\tau)\, \psi_s(t-\tau)\, \mathrm{d}\tau\ . \tag{4.68}$$

The function ψ should satisfy certain conditions. Its "energy" should be bounded, which means $\int_{-\infty}^{\infty} |f(t)|^2\, \mathrm{d}t < \infty$, and the weighted integral of its spectral density (the square of the Fourier transform $\hat{\psi}$) should be finite:

$$c_\psi = 2\pi \int_{-\infty}^{\infty} \frac{1}{|\omega|} \left|\hat{\psi}(\omega)\right|^2 \mathrm{d}\omega < \infty\ .$$

The functions ψ found in the literature usually fulfill this *admissibility condition* by design. Moreover, we require the functions ψ to fulfill

$$\int_{-\infty}^{\infty} \psi(\eta)\, \mathrm{d}\eta = 0\ . \tag{4.69}$$

The functions ψ therefore oscillate around the abscissa and fall off rapidly at large distances from the origin, giving them the appearance of small waves and the nickname *wavelets*. The simplest wavelet is the Haar function

$$\psi_{\text{Haar}}(\eta) = \begin{cases} 1 \; ; \; 0 \le \eta < 1/2 \, , \\ -1 \; ; \; 1/2 \le \eta < 1 \, , \\ 0 \; ; \; \text{otherwise} \, . \end{cases}$$

The *derivative of Gaussian* wavelets DOG(m) are also very simple to use. We obtain them by successive derivatives of the Gauss function,

$$\psi_{\text{DOG}(m)}(\eta) = \frac{(-1)^{m+1}}{\sqrt{\Gamma(m+1/2)}} \frac{\mathrm{d}^m}{\mathrm{d}\eta^m} \left(\mathrm{e}^{-\eta^2/2} \right) \, . \tag{4.70}$$

Another useful wavelet is the complex Morlet wavelet

$$\psi_{\text{Morlet}}(\eta) = \pi^{-1/4} \, \mathrm{e}^{\mathrm{i}\omega_0 \eta} \, \mathrm{e}^{-\eta^2/2} \, , \qquad \omega_0 \in [5, 6] \, . \tag{4.71}$$

(For the Morlet wavelet (4.69) is not exactly fulfilled; the absolute precision to which the equality is valid improves when ω_0 is increased, and amounts to at least $\approx 10^{-5}$ for $\omega_0 > 5$.) When complex wavelets are used, the corresponding transforms should be specified in terms of their magnitudes and phases (see Fig. 4.20). Some typical wavelets are shown in Fig. 4.21.

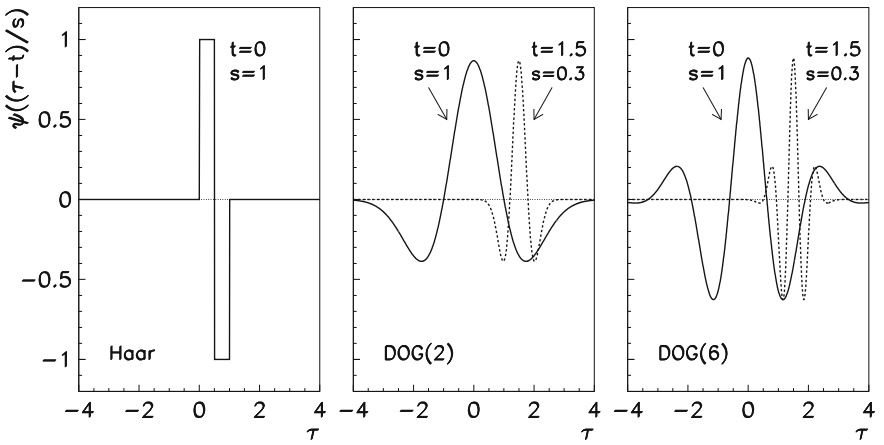


Fig. 4.21 Examples of wavelets used in the continuous wavelet transformation. By horizontal shifts and changes of scale the wavelet probes the features of the investigated signal and the times at which these features appear. [LEFT] Haar wavelet. [CENTER] The DOG(2) wavelet known as the "Mexican hat". [RIGHT] The DOG(6) wavelet

4.6.1 Numerical Computation of the Wavelet Transform

The continuous wavelet transform (4.68) of the discrete values of the signal f_k with the chosen scale parameter s is evaluated by computing the convolution sum [63]

$$L_\psi[f](s,t_n) = \frac{1}{\sqrt{c_\psi s}} \sum_{k=0}^{N-1} f_k\, \psi^*\left(\frac{(k-n)\Delta t}{s}\right), \qquad n = 0, 1, \ldots, N-1\,. \tag{4.72}$$

We take the values of s from an arbitrary set $\{s_m\}_{m=0}^{M-1}$ with some $M < N$. The most simple choice is $s_m = (m+1)\Delta t$. The computation of the sum becomes unacceptably slow for large N and M as the time cost increases as $\mathcal{O}(MN^2)$. Since the convolution of two functions in configuration space is equivalent to the multiplication of their Fourier transforms in the Fourier space, the CWT can be computed by using the fast Fourier transformation (FFT).

Wavelets Given as Continuous Functions The procedure is simple when wavelet functions exist in closed forms and for which the analytic form of their Fourier transforms in known. First we use the FFT to compute the Fourier transform F of the signal f which has been sampled at N points with uniform spacings Δt:

$$F_k = \mathcal{F}_N[f]_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n\, \mathrm{e}^{-\mathrm{i}\,2\pi kn/N}\,. \tag{4.73}$$

If the wavelet is given by the function $\psi(t/s)$ in configuration space, its correlate in Fourier space (in the continuous limit) is the function $\hat{\psi}(s\omega)$. For example, the family of wavelets (4.70) corresponds to the family of transforms

$$\hat{\psi}_{\mathrm{DOG}(m)}(s\omega) = \frac{-\mathrm{i}^m}{\sqrt{\Gamma(m+1/2)}}\,(s\omega)^m\,\mathrm{e}^{-(s\omega)^2/2}\,.$$

The wavelet transform is then evaluated by using the inverse FFT to compute the sum

$$L_\psi[f](s,t_n) = \frac{1}{\sqrt{c_\psi s}} \sum_{k=0}^{N-1} F_k\, \hat{\psi}^*(s\omega_k)\, \mathrm{e}^{\mathrm{i}\,\omega_k n \Delta t}\,,$$

where

$$\omega_k = \begin{cases} 2\pi k/(N\Delta t)\ ;\ k \le N/2\,, \\ -2\pi k/(N\Delta t)\ ;\ k > N/2\,. \end{cases}$$

The numerical cost of this procedure is $\mathcal{O}(MN\log N)$.

Wavelets Given at Discrete Points If the wavelet is not specified as an analytic function or if its Fourier representation is not known, we need to compute both the discrete Fourier transform of the sampled signal (4.73) and the discrete Fourier transform of the wavelet at scale s, i.e.

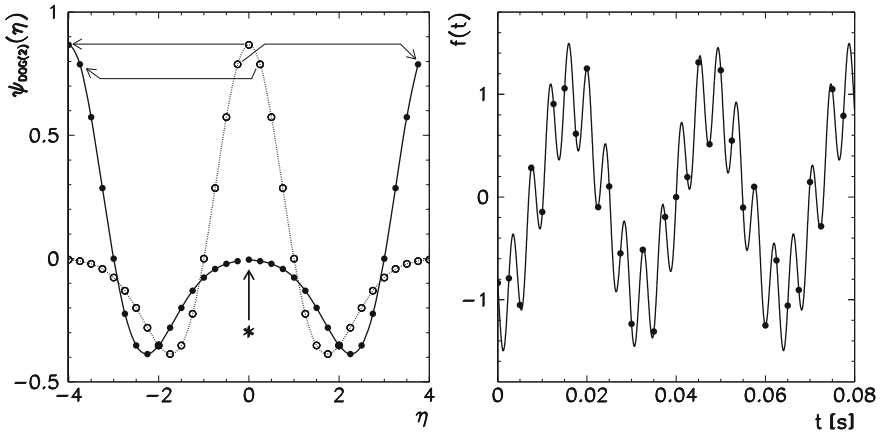


Fig. 4.22 Sampling at $N = 32$ points for the computation of the continuous wavelet transform. [LEFT] Periodic sampling of the wavelet in its natural (dimensionless) scale at the maximum scaling parameter s. The wavelet is sampled at N points on $[\eta_{\min}, \eta_{\max}] = [-4, 4]$. When we wish to reduce the parameter s, we insert zeros at the location marked by the thick arrow and the symbol $*$. [RIGHT] Sampling of the signal at N points of the physical (time) scale or the interval $[0, N\Delta t]$. The relation between the dimensionless variable η and the physical time t is $\eta = t(\eta_{\max} - \eta_{\min})/(N\Delta t)$. For details see [64]

$$Y_k = \mathcal{F}_N[\psi_s]_k = \frac{1}{N}\sum_{n=0}^{N-1} \psi^*(-n\Delta t/s)\, \mathrm{e}^{-\mathrm{i}\,2\pi kn/N} \,.$$

The convolution is at the heart of this procedure and the way the sampling is done requires some attention. For even wavelets it makes sense to adopt the sampling which preserves the symmetry of the wavelet about its origin. The wavelet and the signal are sampled as shown in Fig. 4.22.

We obtain the wavelet transform by multiplying the arrays $F_k \equiv \mathcal{F}_N[f]_k$ and $Y_k \equiv \mathcal{F}_N[\psi_s]_k$ component-wise, diving the result by $\sqrt{c_\psi s}$, and computing the inverse Fourier transform of the product array W_k:

$$L_\psi[f](s) = N\mathcal{F}_N^{-1}[W]\,, \qquad W_k \equiv \frac{1}{\sqrt{c_\psi s}}\, F_k Y_k \,.$$

The inverse procedure is at hand: we reconstruct the original signal from the continuous wavelet transform by deconvolution, i.e. by dividing the Fourier representation of the transform by the Fourier representation of the wavelet. We form the arrays $L_k \equiv \mathcal{F}_N[L_\psi[f](s)]_k$ and $Y_k \equiv \mathcal{F}_N[\psi_s]_k$, divide them, multiply them by $\sqrt{c_\psi s}$, and compute the inverse Fourier transform of the quotient array:

$$f = N^{-1}\mathcal{F}_N^{-1}[F]\,, \qquad F_k \equiv \sqrt{c_\psi s}\,(L_k/Y_k)\,.$$

(The parameter s obviously has to be chosen such that $Y_k \neq 0$ for all k.)

4.7 Discrete Wavelet Transformation ⋆

In Sect. 4.6 we have sampled the signal $f(t)$ and the wavelet $\psi(t)$ only at discrete points, but the described transformation can still be considered continuous, since the scale parameter s and the time axis span all $M \times N$ values. The continuous wavelet transform of a function sampled at N points has $M \times N$ values and is therefore highly redundant. In contrast, the wavelet transformation that represents the signal uniquely at just N points is known as *discrete wavelet transformation* (DWT).

The basic idea of the DWT [65, 66] is to break down a signal into two partial, half-length subsignals called the *trend* and the *detail*, respectively. The former is a coarser version of its predecessor, while the latter encodes the details that were lost in downsampling the original. The fact that this process can be repeated recursively leads to a fast version of the algorithm.

4.7.1 One-Dimensional DWT

A single step of the discrete wavelet transformation of a signal $f(t)$ sampled at times $t_n = n\Delta t$, $n = 0, 1, \ldots, N-1$, and represented by a N-dimensional vector

$$\boldsymbol{f} = (f_0, f_1, \ldots, f_{N-1})^{\mathrm{T}}$$

formally translates to applying digital filters to $\boldsymbol{f}$ [65, 67]. These filters operate as

$$\widetilde{f}_i = \sum_{k=0}^{K-1} c_k f_{2i+k}\,, \qquad i = 0, 1, 2, \ldots\,,$$

where c_k are the filter coefficients and K is the filter length. Note the $2i$: the signals are being downsampled. Two types of filters are used: "low-pass" (denoted by $\mathcal{H}$ and specified by the coefficients h_k) and "high-pass" ($\mathcal{G}$ and g_k).

The simplest variety of the DWT with $K = 2$ is the discrete version of the continuous transformation with Haar wavelets. The first filter with the coefficients $h_0 = h_1 = 1/\sqrt{2}$ (see Table 4.2) is used to compute the running averages of the signal, $f_0^1 = (f_0 + f_1)/\sqrt{2}$, $f_1^1 = (f_2 + f_3)/\sqrt{2}$, ..., resulting in the trend subsignal of length $N/2$,

$$\boldsymbol{f}^1 = \left(f_0^1, f_1^1, \ldots, f_{N/2-1}^1\right)^{\mathrm{T}}\,. \tag{4.74}$$

Clearly $\boldsymbol{f}^1$ still represents the dominant features of the original signal, but at twice poorer resolution: the filter sorts out the highest frequencies, i. e. doubles the scale. The second filter with the coefficients $g_0 = -g_1 = 1/\sqrt{2}$ is used to compute the corresponding differences, $(f_0 - f_1)/\sqrt{2}$, $(f_2 - f_3)/\sqrt{2}$, ..., yielding the detail signal

$$\boldsymbol{d}^1 = \left(d_0^1, d_1^1, \ldots, d_{N/2-1}^1\right)^{\mathrm{T}}$$

describing the *fluctuations* around the trend (4.74). These filter actions are equivalent to computing the components of $\boldsymbol{f}^1$ and $\boldsymbol{d}^1$ by the scalar products

$$f_n^1 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{H}_n^1 , \qquad d_n^1 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{G}_n^1 , \qquad n = 0, 1, \ldots, N/2 - 1 , \tag{4.75}$$

where $\boldsymbol{H}_0^1 = \left(1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots\right)^{\mathrm{T}}$, $\boldsymbol{H}_1^1 = \left(0, 0, 1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots\right)^{\mathrm{T}}, \ldots$ and $\boldsymbol{G}_0^1 = \left(1/\sqrt{2}, -1/\sqrt{2}, 0, \ldots\right)^{\mathrm{T}}$, $\boldsymbol{G}_1^1 = \left(0, 0, 1/\sqrt{2}, -1/\sqrt{2}, 0, \ldots\right)^{\mathrm{T}}, \ldots$ We denote the effect of this single-step (level-1) DWT by

$$\boldsymbol{f} \mapsto \left(\boldsymbol{f}^1 \middle| \boldsymbol{d}^1\right) .$$

The process can now be repeated as many times as the signal can be halved: see Fig. 4.25 (top). In general, a signal containing $N = 2^L$ values can be subject to a maximum of L consecutive transformations, each halving the dimension of the preceding trend and detail vectors. The level-2 DWT results in the $N/4$-dimensional trend and detail signals $\boldsymbol{f}^2$ and $\boldsymbol{d}^2$ derived from $\boldsymbol{f}^1$, as well as the previous $N/2$-dimensional detail vector $\boldsymbol{d}^1$,

$$\boldsymbol{f} \mapsto \left(\boldsymbol{f}^2 \middle| \boldsymbol{d}^2, \boldsymbol{d}^1\right) ,$$

the level-3 DWT produces the $N/8$-dimensional trend and detail signals $\boldsymbol{f}^3$ and $\boldsymbol{d}^3$ derived from $\boldsymbol{f}^2$ as well as the two previous detail signals,

$$\boldsymbol{f} \mapsto \left(\boldsymbol{f}^3 \middle| \boldsymbol{d}^3, \boldsymbol{d}^2, \boldsymbol{d}^1\right) ,$$

and so on. The components of $\boldsymbol{f}^2$ are given by the scalar products

$$f_n^2 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{H}_n^2 , \qquad d_n^2 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{G}_n^2 , \qquad n = 0, 1, \ldots, N/4 - 1 , \tag{4.76}$$

where $\boldsymbol{H}_n^2 = h_0 \boldsymbol{H}_{2n}^1 + h_1 \boldsymbol{H}_{2n+1}^1$ and $\boldsymbol{G}_n^2 = g_0 \boldsymbol{H}_{2n}^1 + g_1 \boldsymbol{H}_{2n+1}^1$, the components of $\boldsymbol{f}^3$ are given by

$$f_n^3 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{H}_n^3 , \qquad d_n^3 = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{G}_n^3 , \qquad n = 0, 1, \ldots, N/8 - 1 , \tag{4.77}$$

where $\boldsymbol{H}_n^3 = h_0 \boldsymbol{H}_{2n}^2 + h_1 \boldsymbol{H}_{2n+1}^2$ and $\boldsymbol{G}_n^3 = g_0 \boldsymbol{H}_{2n}^2 + g_1 \boldsymbol{H}_{2n+1}^2$, and so on: compare (4.76) and (4.77) to (4.75) and note the diminishing range of index n! An example of a three-level DWT of a 64-dimensional signal is shown in Fig. 4.23.

The recursive division of the trend signals to their subordinate trends and details can be tracked on a time versus scale mesh which is a discrete representation of the landscape shown, for instance, in Fig. 4.19 (bottom). Assuming that the original signal was continuous, given by $f(t)$, the detail subsignals $\boldsymbol{d}^1, \boldsymbol{d}^2, \ldots, \boldsymbol{d}^k$ and the final $\boldsymbol{f}^k$ contain the DWT of $\boldsymbol{f}$ at very specific points dictated by this recurrence:

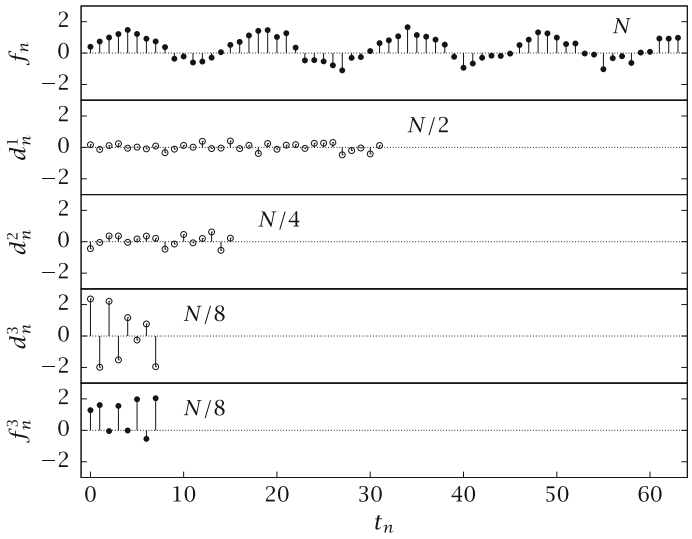
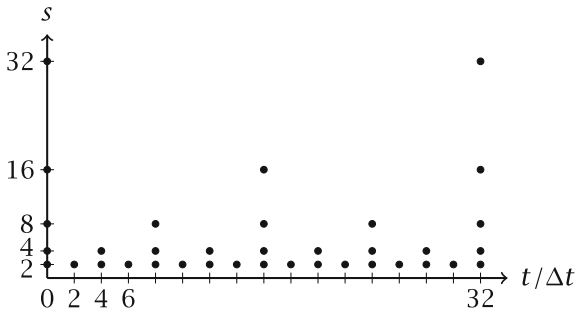


Fig. 4.23 A three-level discrete wavelet transform of signal $\boldsymbol{f}$ of length $N = 64$, resulting in the trend $\boldsymbol{f}^3$ and three detail subsignals $\boldsymbol{d}^1$, $\boldsymbol{d}^2$ and $\boldsymbol{d}^3$

Fig. 4.24 The discrete time versus scale mesh on which the DWT stores its trend and detail subsignals



$$L_\psi[f]\left(s = 2^l, t = 2^l n\right) = \frac{1}{\sqrt{c_\psi}}\, d_n^l\,, \qquad l = 1, 2, \ldots, L\,,$$

which may be compared to the CWT given by formula (4.72). An example of such a mesh is shown in Fig. 4.24. The redundancy of values plaguing the continuous transform is gone: the DWT encodes all information in just N points.

The Inverse Transformation The $\boldsymbol{f}^1$ and $\boldsymbol{d}^1$ taken together (or $\boldsymbol{f}^2$, $\boldsymbol{d}^2$ and $\boldsymbol{d}^1$ taken together, and so on) encode all information contained in $\boldsymbol{f}$. This is ensured by requiring that the filter coefficients satisfy $\sum_k h_k^2 = \sum_k g_k^2 = 1$ (energy conservation) as well as $\sum_k h_k = \sqrt{2}$ and $\sum_k g_k = 0$ [59]. In addition, the transformation should be invertible, i. e. the signal $\boldsymbol{f}$ should be recoverable from the transforms $(\boldsymbol{f}^1|\boldsymbol{d}^1)$, $(\boldsymbol{f}^2|\boldsymbol{d}^2, \boldsymbol{d}^1)$, and so on. The inverse DWT is realized by applying the so-called *dual*

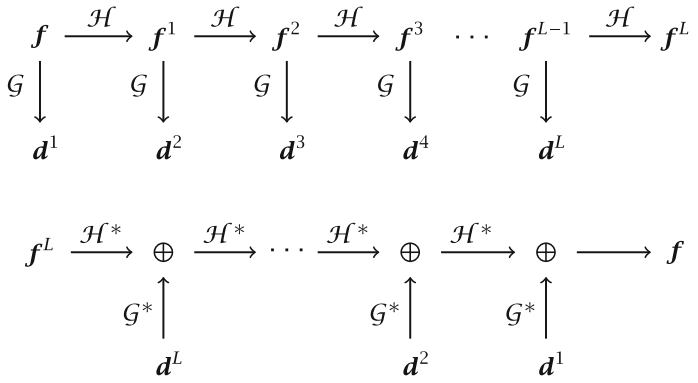


Fig. 4.25 The logical flow of the [TOP] L-level discrete wavelet transformation by using the $\mathcal{H}$ and $\mathcal{G}$ filters and [BOTTOM] its inverse by using their respective duals

filters $\mathcal{H}^*$ and $\mathcal{G}^*$ to subsequent trend and detail signals in reverse order, thereby reconstructing the original signal: see Fig. 4.25 (bottom).

For example, the three-level DWT given by (4.75), (4.76) and (4.77) can be inverted by using the same filter coefficients as the "forward" DWT, by computing

$$\boldsymbol{F}^3 = \sum_{n=0}^{N/8-1} f_n^3 \boldsymbol{H}_n^3 \tag{4.78}$$

as well as

$$\boldsymbol{D}^3 = \sum_{n=0}^{N/8-1} d_n^3 \boldsymbol{G}_n^3\,, \qquad \boldsymbol{D}^2 = \sum_{n=0}^{N/4-1} d_n^2 \boldsymbol{G}_n^2\,, \qquad \boldsymbol{D}^1 = \sum_{n=0}^{N/2-1} d_n^1 \boldsymbol{G}_n^1\,, \tag{4.79}$$

and finally summing the parts to yield $\boldsymbol{f} = \boldsymbol{F}^3 + \boldsymbol{D}^3 + \boldsymbol{D}^2 + \boldsymbol{D}^1$. For a L-level transformation, this obviously generalizes to

$$\boldsymbol{f} = \boldsymbol{F}^L + \boldsymbol{D}^L + \boldsymbol{D}^{L-1} + \cdots + \boldsymbol{D}^2 + \boldsymbol{D}^1\,.$$

Daubechies Wavelets

Many invertible filters can be designed that fulfill the above mentioned requirements for h_k and g_k. Each admissible set of filter coefficients corresponds to a specific family of wavelet functions themselves. The $h_0 = h_1 = 1/\sqrt{2}$ filter corresponds to Haar wavelets, which are the simplest members of a broader class of widely used Daubechies wavelets [68, 69] denoted by db$m$. The m stands for the number of vanishing moments of the wavelet function $\psi_{\mathrm{db}m}(x)$. The requirement that $\int_{-\infty}^{\infty} t^p \psi(t)\,\mathrm{d}t = 0$ for p as high as possible is driven by the desire to maximize the sensitivity of the wavelet transformation: if $\psi(t)$ has enough vanishing moments,

Table 4.2 The $\mathcal{H}$-filter coefficients belonging to the Daubechies (db$m$) wavelets with m ranging from 1 (Haar wavelet) to 5

	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
h_0	$1/\sqrt{2}$	$(1+\sqrt{3})/4\sqrt{2}$	0.332671	0.230378	0.160102
h_1	$1/\sqrt{2}$	$(3+\sqrt{3})/4\sqrt{2}$	0.806892	0.714847	0.603829
h_2		$(3-\sqrt{3})/4\sqrt{2}$	0.459878	0.630881	0.724309
h_3		$(1-\sqrt{3})/4\sqrt{2}$	−0.135011	−0.0279838	0.138428
h_4			−0.0854413	−0.187035	−0.242295
h_5			0.0352263	0.0308414	−0.0322449
h_6				0.032883	0.0775715
h_7				−0.0105974	−0.00624149
h_8					−0.0125808
h_9					0.00333573

the smooth parts of the signal will yield close-to-zero transform strength, while the transform will be concentrated around the instances of rapid signal changes.

A DWT based on db$m$ wavelets has $2m$ filter coefficients for each of the $\mathcal{H}$ and $\mathcal{G}$ filters. Table 4.2 lists the first five $\mathcal{H}$-sets. The corresponding $\mathcal{G}$-coefficients are given by

$$g_k = (-1)^k h_{2m-k-1}\,, \qquad k = 0, 1, \ldots, 2m-1\,.$$

Figure 4.26 shows the db$m$ wavelet functions for $m = 2$, 4, 6, 10, 15 and 20. Note that these functions need not be coded explicitly—and actually can not be as they do not possess closed-form expressions. It is the filters that embody the transformation as if it were done with precisely these functions.

Example The three-level DWT based on the db2 wavelets is calculated by using the formulas (4.75), (4.76) and (4.77), but now the $\boldsymbol{H}^1_n$ vectors are given by

$$\begin{aligned}
\boldsymbol{H}^1_0 &= \big(h_0, h_1, h_2, h_3, 0, 0, \ldots, 0\big)^{\mathrm{T}}\,,\\
\boldsymbol{H}^1_1 &= \big(0, 0, h_0, h_1, h_2, h_3, 0, 0, \ldots, 0\big)^{\mathrm{T}}\,,\\
&\vdots\\
\boldsymbol{H}^1_{N/2-2} &= \big(0, 0, \ldots, 0, h_0, h_1, h_2, h_3\big)^{\mathrm{T}}\,,\\
\boldsymbol{H}^1_{N/2-1} &= \big(h_2, h_3, 0, 0, \ldots, 0, h_0, h_1\big)^{\mathrm{T}}\,,
\end{aligned} \tag{4.80}$$

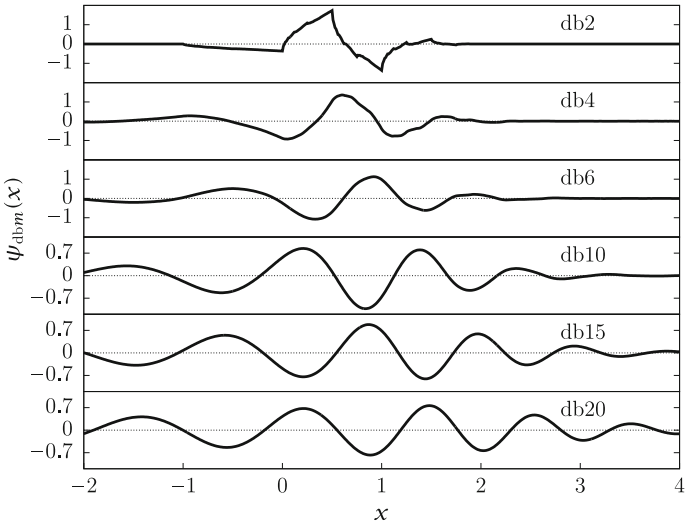


Fig. 4.26 Daubechies wavelets for $m = 2, 4, 6, 10, 15$ and 20

with $\{h_k\}_{k=0}^3$ listed in the $m = 2$ column of Table 4.2. The $\boldsymbol{G}_n^1$ vectors have the same structure with h_k replaced by g_k. Since the filter is specified by four coefficients, one has to decide what to do with the overflowing indices $N/2$ and $N/2+1$. The usual solution is to wrap around the coefficients as shown in (4.80): this corresponds to the assumption that the signal is periodic. By the same token, the $\boldsymbol{H}_n^2$ and $\boldsymbol{H}_n^3$ vectors are constructed as

$$\begin{aligned}
\boldsymbol{H}_n^2 &= h_0\boldsymbol{H}_{2n}^1 + h_1\boldsymbol{H}_{2n+1}^1 + h_2\boldsymbol{H}_{2n+2}^1 + h_3\boldsymbol{H}_{2n+3}^1\,, \qquad n = 0, 1, \ldots, N/4-2\,,\\
\boldsymbol{H}_n^3 &= h_0\boldsymbol{H}_{2n}^2 + h_1\boldsymbol{H}_{2n+1}^2 + h_2\boldsymbol{H}_{2n+2}^2 + h_3\boldsymbol{H}_{2n+3}^2\,, \qquad n = 0, 1, \ldots, N/8-2\,,
\end{aligned}$$

while computing the stranded $\boldsymbol{H}_{N/4-1}^1$ and $\boldsymbol{H}_{N/8-1}^2$ again involves wrap-around:

$$\begin{aligned}
\boldsymbol{H}_{N/4-1}^2 &= h_2\boldsymbol{H}_0^1 + h_3\boldsymbol{H}_1^1 + h_0\boldsymbol{H}_{N/2-2}^1 + h_1\boldsymbol{H}_{N/2-1}^1\,,\\
\boldsymbol{H}_{N/8-1}^3 &= h_2\boldsymbol{H}_0^2 + h_3\boldsymbol{H}_1^2 + h_0\boldsymbol{H}_{N/4-2}^2 + h_1\boldsymbol{H}_{N/4-1}^2\,.
\end{aligned}$$

The calculation of the $\boldsymbol{G}_n^2$ and $\boldsymbol{G}_n^3$ vectors proceeds along the same lines, with h_k replaced by g_k. The inverse transformation is performed by using (4.78) and (4.79), followed by $\boldsymbol{f} = \boldsymbol{F}^3 + \boldsymbol{D}^3 + \boldsymbol{D}^2 + \boldsymbol{D}^1$. The DWT involving db$m$ with larger m require correspondingly more coefficients to be wrapped around: four for $m = 3$, six for $m = 4$, and so on. ◁

4.7.2 Two-Dimensional DWT

The generalization of the discrete wavelet transformation to two-dimensional signals (e. g. images) is straightforward. Assume that we are dealing with a square ($N \times N$) grayscale image f represented by a matrix of values

$$f_{ij}\ , \quad i, j = 0, 1, \ldots, N-1\ .$$

A single-step DWT now also involves processing the image with the $\mathcal{H}$ and $\mathcal{G}$ filters, but this proceeds in four different ways. Scanning with $\mathcal{H}$ along both the columns (c) and rows (r) produces the approximate image $f^1 = \mathcal{H}_\mathrm{r}\mathcal{H}_\mathrm{c}[f]$. Operating with $\mathcal{H}$ along the columns and $\mathcal{G}$ along the rows results in the "vertical detail" image $d^{1\mathrm{v}} = \mathcal{G}_\mathrm{r}\mathcal{H}_\mathrm{c}[f]$, while the opposite procedure gives the "horizontal detail" image $d^{1\mathrm{h}} = \mathcal{H}_\mathrm{r}\mathcal{G}_\mathrm{c}[f]$. Applying $\mathcal{G}$ to both the columns and rows encodes the "diagonal detail" image $d^{1\mathrm{d}} = \mathcal{G}_\mathrm{r}\mathcal{G}_\mathrm{c}[f]$. The first-level DWT of a $N \times N$ image f therefore produces four $N/2 \times N/2$ subimages usually arranged as

$$f \mapsto \begin{pmatrix} f^1 & d^{1\mathrm{h}} \\ d^{1\mathrm{v}} & d^{1\mathrm{d}} \end{pmatrix}\ .$$

The procedure can then be recursively repeated by exact analogy to the one-dimensional case. The approximate ("trend") $N/4 \times N/4$ and $N/8 \times N/8$ images f^2 and f^3, for instance, are generated by calculating

$$f^1 \mapsto \begin{pmatrix} f^2 & d^{2\mathrm{h}} \\ d^{2\mathrm{v}} & d^{2\mathrm{d}} \end{pmatrix}\ , \qquad f^2 \mapsto \begin{pmatrix} f^3 & d^{3\mathrm{h}} \\ d^{3\mathrm{v}} & d^{3\mathrm{d}} \end{pmatrix}\ , \qquad \ldots$$

An example of a three-level two-dimensional DWT is shown in Fig. 4.27.

The inverse transformation proceeds analogously to the flowchart of Fig. 4.25 (bottom), where $\mathcal{H}^*$ and $\mathcal{G}^*$ are replaced by the appropriate products of $\mathcal{H}_\mathrm{r}^*$, $\mathcal{H}_\mathrm{c}^*$, $\mathcal{G}_\mathrm{r}^*$ and $\mathcal{G}_\mathrm{c}^*$, and the $\oplus$ operation stands for the summation of matrices.

Image Compression and Denoising

Wavelet transformations lie at the heart of modern data compression algorithms. The simplest way one-dimensional data can be compacted by means of the *continuous* wavelet transformation is illustrated in Problem 4.8.4. Filtering and compressing two-dimensional data like images, however, calls for fast two-dimensional discrete transformations.

The basic trick of image compression consists of transforming the image matrix by DWT, manipulating the transform, and calculating the inverse transform. The most obvious strategy of this "manipulation" is to truncate the transform, i. e. to strip it of its insignificant coefficients and keep only the prominent ones. This "cut-off" can be realized in many ways, the simplest of which are taking only the coefficients with absolute values above a certain threshold, or keeping a specified number of coefficients. The latter approach is demonstrated in Fig. 4.28. The original image

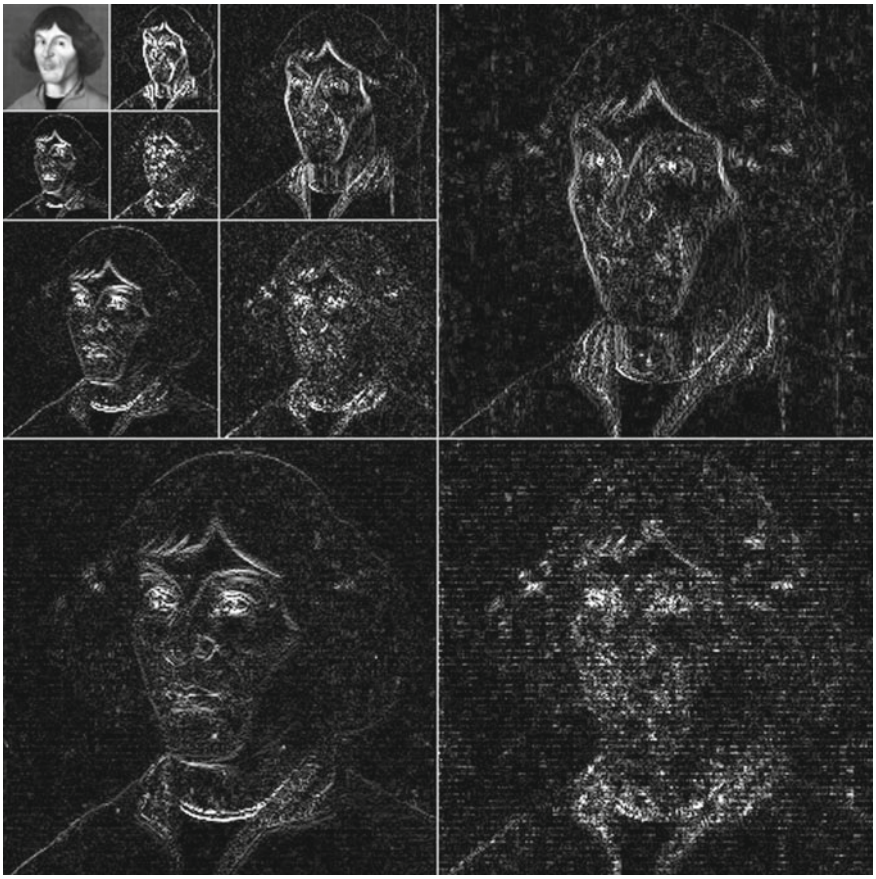


Fig. 4.27 A two-dimensional DWT of a 512 × 512 image. (The original image can be found on the website of the book, the "almost" original is in the bottom right panel of Fig. 3.7.) The 64 × 64 level-3 approximate image f^3 is in the top left corner, encircled (outward and clockwise) by the 64 × 64 detail images d^{3h}, d^{3d}, d^{3v}, followed by their 128 × 128 predecessors d^{2h}, d^{2d}, d^{2v} and their 256 × 256 parents d^{1h}, d^{1d} and d^{1v}

(see caption of Fig. 4.27) has been transformed by a 4-level DWT based on db4 wavelets, the transform truncated, and inverted.

The image format standard JPEG2000 (also known as J2K) [70] exploits a special variety of the DWT relying on *biorthogonal filters*. The biorthogonal DWT utilizes *two* sets of low-pass filters, *two* sets of high-pass filters (and their inverses), thus implying *two* types of wavelets. For lossy J2K compression one uses the 9/7 Cohen-Daubechies-Feauveau (CDF) wavelets, while the lossless compression relies on 5/3 CDF wavelets [71, 72]. Their coefficients and further application details can be found in Sects. 3.7 and 3.8 of [67].

Fig. 4.28 Image compression by DWT. The original image is a 512×512 matrix of pixels with 256 gray levels. Shown are the DWT reconstructions by using the db4 wavelets up to refinement level $L = 4$ and keeping the largest [LEFT] 262 transform coefficients (99.9% compression) and [RIGHT] largest 655 coefficients (99.8% compression). Compare to the top two panels of Fig. 3.7 as well as Problems 3.10.4 and 4.8.4

Software Implementations

For C/C++ environments, the DWT with Daubechies and CDF wavelets is implemented in the GSL library, while the LIBDWT project [73] focuses on DWT and CWT with biorthogonal wavelets optimized for image processing. PYWAVELETS [74] is a free open-source PYTHON-based library offering several varieties of the wavelet transform that also supports custom wavelets. The one-dimensional and two-dimensional DWT are implemented in both MATHEMATICA and the Wavelet Toolbox of MATLAB. For freely accessible MATLAB routines, see [75–77].

4.8 Problems

4.8.1 Fourier Spectrum of Signals

The discrete Fourier transformation (DFT, Sect. 4.2.2) is the fundamental tool of signal analysis. First we confirm the formulas for known pairs of periodic signals $f = \{f_j\}_{j=0}^{N-1}$ and their transforms $F = \{F_k\}_{k=0}^{N-1} = \mathcal{F}_N[f]$, for example,

$$f_j = \mathrm{e}^{\mathrm{i}aj/N} \Leftrightarrow F_k = \frac{1}{N}\left\{\left(\mathrm{e}^{\mathrm{i}a} - 1\right)\Big/\left(\mathrm{e}^{\mathrm{i}(a-2k\pi)/N} - 1\right)\right\} ,$$
$$\mathrm{Re}\, f_j\,,\ \mathrm{Im}\, f_j \sim \mathcal{N}(0,1) \Leftrightarrow \mathrm{Re}\, F_k\,,\ \mathrm{Im}\, F_k \sim \mathcal{N}(0,N)\,,$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2.

the SVD help us to eliminate the superfluous combinations of the basis functions. We denote

$$A_{ij} = \frac{\phi_j(x_i)}{\sigma_i}\,, \qquad b_i = \frac{y_i}{\sigma_i}\,,$$

and perform the "thin" SVD of the matrix $A = U\Lambda V^{\mathrm{T}} \in \mathbb{R}^{n\times m}$ (p. 142) where $n \geq m$. The matrix $U = (\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{m-1})$ has columns $\boldsymbol{u}_i$ of dimension n, the matrix $V = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1})$ has columns $\boldsymbol{v}_i$ of dimension m, and the diagonal matrix $\Lambda = \mathrm{diag}(\lambda_0, \lambda_1, \ldots, \lambda_{m-1})$ contains the singular values λ_i. We obtain the vector of parameters $\boldsymbol{a}$ by summing

$$\boldsymbol{a} = \sum_{i=0}^{m-1} \frac{\boldsymbol{u}_i^{\mathrm{T}}\boldsymbol{b}}{\lambda_i}\,\boldsymbol{v}_i\,.$$

The covariance matrix of the parameters $\boldsymbol{a}$ is $\mathrm{cov}(a_j, a_k) = \sum_{i=0}^{m-1} V_{ji}V_{ki}/\lambda_i^2$ and its diagonal elements represent the individual variances

$$\sigma^2(a_j) = \sum_{i=0}^{m-1} \frac{V_{ji}^2}{\lambda_i^2}\,. \tag{5.48}$$

We should pay attention to all singular values λ_i for which the ratio $\lambda_i/\lambda_{\max}$ is smaller than $\approx n\varepsilon_{\mathrm{M}}$. Such values increase the error (5.48) and indicate that the inclusion of new model parameters is meaningless. Moreover, they do not contribute significantly to the minimization of χ^2, so we exclude them. We do this by setting $1/\lambda_i = 0$ (for a detailed explanation, see the comment to the `Fitsvd` algorithm in [25]). We may exclude also those singular values for which the ratio $\lambda_i/\lambda_{\max}$ is *larger* than $\approx n\varepsilon_{\mathrm{M}}$, until χ^2 starts to increase visibly.

5.5.3 Robust Methods for One-Dimensional Regression

Like all estimates of location and dispersion, regression methods are sensitive to outliers (Sect. 5.2.1). The straight line LS in Fig. 5.14 (left) corresponds to the standard regression on the data $\{(x_i, y_i)\}_{i=0}^{15}$ with unknown errors: here we minimize the sum of the squares of the residuals $r_i = y_i - (a_1x_i + a_0)$, so the straight line LS fails to describe the bulk of the data because of the outliers at x_0, x_1, and x_3. If we remove these three outliers, we obtain the straight line denoted by LS_{-013}. Robust regression methods [29] yield a good description of the majority of the data without the need to remove individual outliers by hand.

The literature on robust regression techniques is abundant: for the identification of outliers (*regression diagnostics*) see [30]; for a review of methods see [31]. Numerous methods exist, all having some advantages and deficiencies; many of them are awkward to implement or entail high numerical costs. One of such methods is the regression in which it is not $\sum_i r_i^2$ that is minimized, but rather $\sum_i |r_i|$ — an L_1-type estimator; see e.g. [32, 33]. Here we describe two procedures with proven good properties for everyday use.

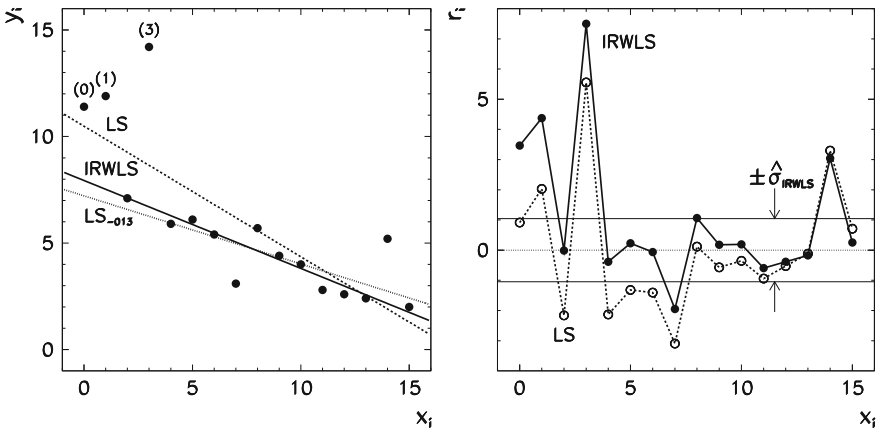


Fig. 5.14 Robust linear regression. (Example adapted from [14].) [LEFT] The regression straight line by the method of least squares (LS), by omitting three outliers (LS_{-013}), and by the iterative reweighting method (IRWLS). [RIGHT] The residuals r_i and the estimate of dispersion by the IRWLS method

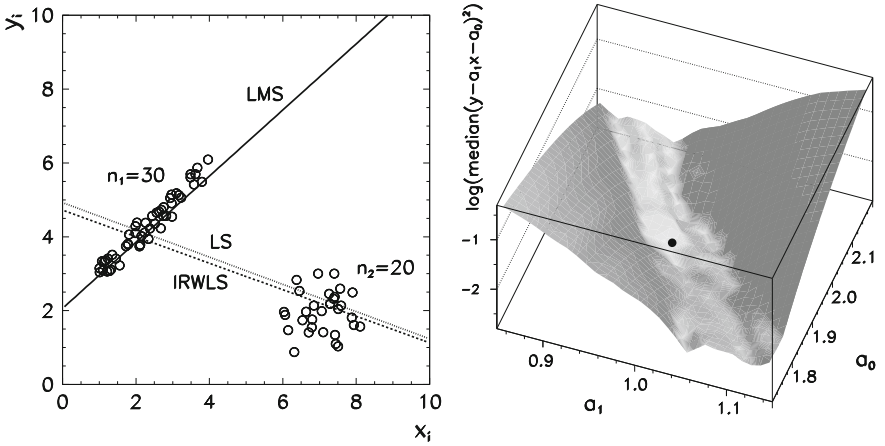


Fig. 5.15 Robust linear regression on the data with many outliers. [LEFT] Only the LMS method correctly describes the majority of the data. [RIGHT] The function being minimized in the LMS method. The symbol • denotes the global minimum

IRWLS The *iterative reweighted least squares method* (IRWLS) closely follows the logic of M-estimates of location (Sect. 5.2.2 and algorithm on p. 257). The iteration to the final values of the parameters a_0 and a_1 of the regression straight line typically converges in ≈ 30 steps. The determination of their uncertainties, the estimate for the dispersion, and other details are discussed in [14], p. 105. An example is shown in Fig. 5.14 (left, IRWLS line). Figure 5.14 (right) shows the residuals r_i in the LS and IRWLS methods.

Input: Values $(x_i, y_i)_{i=0}^{n-1}$, initial approximations for a_0 and a_1 from standard linear regression, relative precision ε
for $i = 0$ **to** $n-1$ **do**
  $r_i^{(0)} = y_i - (a_0 + a_1 x_i)$;
end
$\widehat{\sigma} = 1.4826 \cdot \text{median}(|r_i^{(0)}|,\ r_i^{(0)} \neq 0)$;
$k = 0$;
while $(\ \max_i |r_i^{(k+1)} - r_i^{(k)}| > \varepsilon\widehat{\sigma}\ )$ **do**
  for $i = 0$ **to** $n-1$ **do**
    $w_i = W(r_i^{(k)}/\widehat{\sigma})$; // $W(t)$ given by (5.21)
  end
  Compute new estimates for a_0 and a_1 by solving the linear system

$$a_0 \sum_i w_i + a_1 \sum_i w_i x_i = \sum_i w_i y_i$$
$$a_0 \sum_i w_i x_i + a_1 \sum_i w_i x_i^2 = \sum_i w_i x_i y_i$$

  for $i = 0$ **to** $n-1$ **do**
    $r_i^{(k+1)} = y_i - (a_0 + a_1 x_i)$;
  end
  $k = k + 1$;
end
Output: a_0, a_1 by the IRWLS method

LMS The second method resembles standard linear regression, except that the median of the squares of the residuals $r_i = y_i - (a_1 x_i + a_0)$ is minimized, hence its name, *least median of squares* (LMS). We seek a_0 and a_1 such that

$$\text{median}(y_i - a_1 x_i - a_0)^2 = \min\,. \tag{5.49}$$

The LMS method [34] is very robust and behaves well even in the rare circumstances in which IRWLS fails. An example is shown in Fig. 5.15 (left). We have a sample of $n_1 = 30$ data $y_i = ax_i + b + u_i$, where $a = 1$, $b = 2$, $x_i \sim U(1, 4)$, and $u_i \sim N(0, 0.2)$, and a set of $n_2 = 20$ points assumed to be outliers, $x_i \sim N(7, 0.5)$, $y_i \sim N(2, 0.5)$. We would like to fit a straight line to the data such that the result will be oblivious to the outlier portion of this compound set. Neither the LS nor the IRWLS method yield meaningful results: both simply run the line through both data portions. In contrast, the LMS line fits the non-outlier group well.

The main nuisance of the LMS method is precisely the numerical minimization (5.49). The function we wish to minimize with respect to the parameters a_j has $\mathcal{O}(n^{m+1})$ local minima, where n is the number of data points (x_i, y_i) and m is the degree of the regression polynomial. In the example from the Figure we have $n = n_1 + n_2 = 50$ and $m + 1 = 2$ (straight line), so there are ≈ 2500 local minima, among which the global minimum needs to be located, as shown in Fig. 5.15 (right).

This is best accomplished — there are dangers since the function is continuous, but not continuously differentiable — by forming the function

$$M(a_1) = \min_{a_0} \left\{ \text{median}\,(y_i - a_1 x_i - a_0)^2 \right\} ,$$

where for each a_1 we determine a_0, and then minimize $M(a_1)$ with respect to a_1. This implementation becomes very costly at large n. Fast computations of the regression parameters by LMS methods are non-trivial: see [35, 36].

5.6 Non-linear Regression

In non-linear regression the dependence of the model function on regression parameters is non-linear, for example

$$f(x; \boldsymbol{a}) = a_0 + a_1 \, \mathrm{e}^{a_2 x} + a_3 \sin(x + a_4) \, .$$

If we arrange the observations y_i at x_i into vectors $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})^{\mathrm{T}}$ and $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})^{\mathrm{T}}$, and the components of the regression function into $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{a}) = \big(f(x_0; \boldsymbol{a}), f(x_1; \boldsymbol{a}), \ldots, f(x_{n-1}; \boldsymbol{a})\big)^{\mathrm{T}}$, where $\boldsymbol{a} = (a_0, a_1, \ldots, a_{p-1})^{\mathrm{T}}$, we can define the measure of deviation as

$$\chi^2 = \big(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{a})\big)^{\mathrm{T}} \Sigma_y^{-1} \big(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{a})\big) \, . \tag{5.50}$$

If the measurement errors are uncorrelated, the covariance matrix is diagonal, $\Sigma_y = \text{diag}\,(\sigma_0^2, \sigma_1^2, \ldots, \sigma_{n-1}^2)$, and the above expression can be simplified to

$$\chi^2 = \sum_{i=0}^{n-1} \frac{\big(y_i - f(x_i; \boldsymbol{a})\big)^2}{\sigma_i^2} \, . \tag{5.51}$$

The minimization of χ^2 now implies solving a system of p (in general non-linear) equations $\partial \chi^2 / \partial a_j = 0$ ($j = 0, 1, \ldots, p-1$). Such problems can be solved iteratively: we ride on the sequence

$$\boldsymbol{a}^{(\nu+1)} = \boldsymbol{a}^{(\nu)} + \Delta \boldsymbol{a}^{(\nu)} \, , \qquad \nu = 0, 1, 2, \ldots \, , \tag{5.52}$$

where $\Delta \boldsymbol{a}^{(\nu)}$ is obtained by solving a *linear* problem, to approach the optimal parameter set. In the νth step the update can be calculated by minimizing

$$\chi^2(\Delta \boldsymbol{a}) = \Big[\boldsymbol{y} - \boldsymbol{f}\Big(\boldsymbol{a}^{(\nu)} + \Delta \boldsymbol{a}\Big)\Big]^{\mathrm{T}} \Sigma_y^{-1} \Big[\boldsymbol{y} - \boldsymbol{f}\Big(\boldsymbol{a}^{(\nu)} + \Delta \boldsymbol{a}\Big)\Big] ,$$

where we have omitted the $\boldsymbol{x}$-dependence of $\boldsymbol{f}$. If $\Delta \boldsymbol{a}$ is small, $\boldsymbol{f}$ can be expanded as $\boldsymbol{f}\big(\boldsymbol{a}^{(\nu)} + \Delta \boldsymbol{a}\big) \approx \boldsymbol{f}\big(\boldsymbol{a}^{(\nu)}\big) + J\big(\boldsymbol{a}^{(\nu)}\big)\, \Delta \boldsymbol{a}$, where J is the Jacobi matrix with the elements

6.8 Independent Component Analysis ⋆

Independent component analysis (ICA) is a multivariate analysis technique from the class of latent-variable methods (another representative is the factor analysis discussed in Sect. 5.12). The classical problem that can be solved by ICA is the decomposition of an unknown mixture of signals to its independent components. For example, we measure the signals $x_1(t)$, $x_2(t)$, and $x_3(t)$ of three microphones recording the speech of two persons (sources) $s_1(t)$ and $s_2(t)$. We assume that the signals are linear combinations of the sources,

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{pmatrix} = \begin{pmatrix} A_{11}\ A_{12} \\ A_{21}\ A_{22} \\ A_{31}\ A_{32} \end{pmatrix} \begin{pmatrix} s_1(t) \\ s_2(t) \end{pmatrix} ,$$

where neither the sources s_i nor the matrix elements A_{ij} are known. A generalization of this example is the famous *cocktail-party problem*, in which we wish to reconstruct the signals of r speakers based on n measurements of signals from m microphones arranged in a room. The components of the vector representing the sources are the latent variables mentioned above.

A very similar problem is illustrated in Fig. 6.16 (see also Problem 6.10.4 and Fig. 6.28). The Figure shows eight traces of an electr-cardiogram of a pregnant woman recorded at various places on the thorax and the abdomen. In some signals (x_3, x_2 and, above all, x_1) we can also see the child's heartbeat which has a higher frequency than the mother's heartbeat, but its amplitudes are smaller and partly masked by noise. The task of ICA is to extract the original signals *sources* and to explain the mixing of these sources into the measured signals. The procedure should be done such that the computed sources are mutually as independent as possible.

Here we discuss only *static linear independent component analysis*, for which we assume that each vector of correlated measurements $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^{\mathrm{T}}$ is generated by *linear* mixing of *independent sources* $\boldsymbol{s} = (s_1, s_2, \ldots, s_r)^{\mathrm{T}}$, thus

$$\boldsymbol{x} = A\boldsymbol{s} \,, \tag{6.33}$$

where $A \in \mathbb{R}^{m\times r}$ is a time-independent *mixing matrix*. Usually the number of sources is smaller than the number of measured signals, $r \le m$ (as above, $r = 2$ hearts and $m = 8$ electrodes). We neglect measurement errors that, in principle, could be added to the right side of (6.33). We assume that noise is already contained in the sources $\boldsymbol{s}$: we are performing a *noiseless ICA*.

The independence of the sources is expressed by the statement that their joint probability density factorizes as $p(s_1, s_2, \ldots, s_r) = p_1(s_1)p_2(s_2)\cdots p_r(s_r)$ and therefore also $\langle f(s_i)g(s_j)\rangle = \langle f(s_i)\rangle\langle g(s_j)\rangle$ for arbitrary functions f and g. A pair of statistically independent variables $(s_i, s_{j\neq i})$ is uncorrelated and has the covariance $\mathrm{cov}(s_i, s_j) = 0$; the inverse is not necessarily true. This distinguishes ICA from the decorrelation of

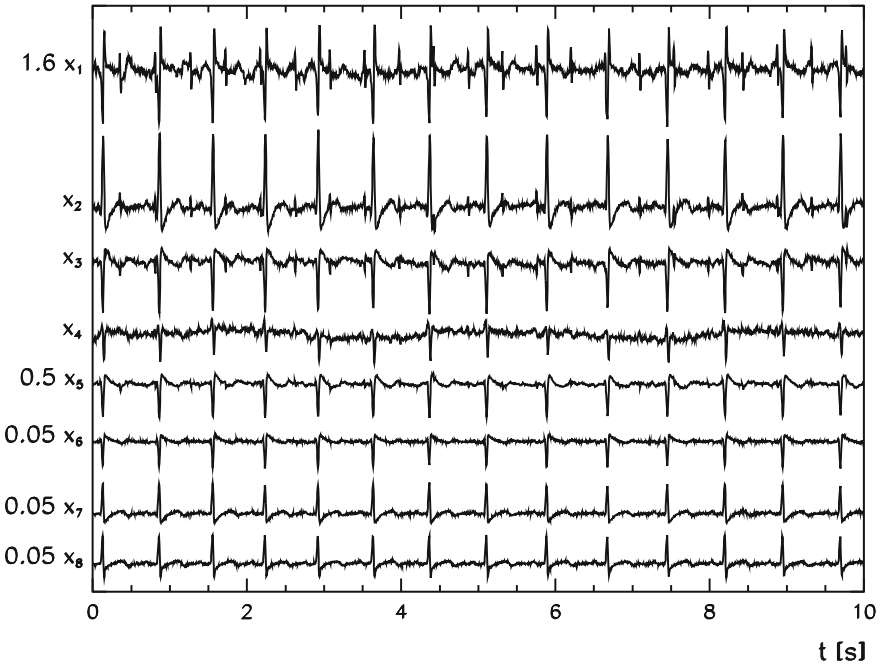


Fig. 6.16 Electro-cardiogram of a pregnant woman. The relatively weak and noisy signal of the child's heartbeat, visible in signals x_1, x_2, and x_3, mixes with the signal of the mother's heartbeat. (Example adapted from [55] based on data from [56].)

variables by PCA (Sect. 5.8), in which principal components of multivariate data are determined by maximizing their variances and minimizing their mutual correlations.

In the following we assume that the average of an individual vector of sources is zero, $\langle \boldsymbol{s} \rangle = \boldsymbol{0}$, and that the corresponding covariance matrix is diagonal, $\Sigma_{ss} = \mathrm{cov}(\boldsymbol{s}, \boldsymbol{s}) = \boldsymbol{s}\boldsymbol{s}^{\mathrm{T}} = \boldsymbol{I}$. Other than that, the components s_i may have any probability distribution except Gaussian. The requirement for a non-Gaussian character of the distributions is essential, otherwise the ICA method allows for the determination of the independent components only up to an orthogonal transformation. At most one component of the signal is allowed to be Gaussian: for a detailed explanation see Sect. 15.3 in [55] and [57, 58].

6.8.1 *Estimate of the Separation Matrix and the FastICA Algorithm*

According to the model (6.33) with the matrix A of full rank, a *separation* or *unmixing matrix* W exists by which the sources $\boldsymbol{s}$ can be exactly reconstructed from the measured signals $\boldsymbol{x}$:

$$\boldsymbol{s} = W\boldsymbol{x}\,, \qquad W = \left(A^{\mathrm{T}}A\right)^{-1}A^{\mathrm{T}} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_r)^{\mathrm{T}} \tag{6.34}$$

(see (3.15)). But in practice, neither the matrix A nor the vector of the sources $\boldsymbol{s}$ in the model (6.33) are known. In order to reconstruct $\boldsymbol{s}$, we therefore attempt to find an estimate of the separation matrix W. We resort to several additional requirements that should be fulfilled by the one-dimensional projections of the measured signals,

$$y_i = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}\,, \qquad i = 1, 2, \ldots, r\,.$$

If the vector $\boldsymbol{w}_i$ is equal to some row of the generalized inverse W of A (see (6.34)), the projection y_i is already one of the independent components, $y_i = s_i$. Since A (or W) is unknown, we try to find the vectors $\boldsymbol{w}_i$ such that the probability distribution of the projection y_i will be *as non-Gaussian as possible*. Seeking such vectors is the key part of the independent component analysis.

The deviation of the distribution of a continuous random variable Y from the normal ("Gaussian") distribution can be measured by the entropy

$$H(Y) = -\int p(y)\log p(y)\,\mathrm{d}y\,,$$

where $p(y)$ is the probability density of Y. Of all random variables with the same variance the normal random variable has the maximum entropy [26]. The value of the entropy can therefore be used as a tool to gauge the similarity of some unknown distribution to the normal distribution. Computationally it is more convenient to work with *negentropy*

$$I(Y) = H(Y_{\mathrm{Gauss}}) - H(Y)\,, \qquad Y_{\mathrm{Gauss}} \sim N(0, 1)\,,$$

which is a non-negative quantity and is equal to zero only in the case that both Y_{Gauss} and Y are distributed normally with zero average and unit variance.

For an exact calculation of $I(Y)$ we need the probability density $p(y)$, which we almost never know in practice. We therefore use approximations of the negentropy, most often [57]

$$I(Y) \approx \left[\langle G(Y)\rangle - \langle G(Y_{\mathrm{Gauss}})\rangle\right]^2\,, \tag{6.35}$$

where $G(y)$ is a non-quadratic function of y (two popular choices are given in Table 6.2). We compute the estimate y_i for a single independent component s_i by finding a vector $\boldsymbol{w}_i$ such that the projection $y_i = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}$ will have maximum negentropy $I(y_i)$ (then its probability density will least resemble the Gaussian). We achieve the decomposition to independent components when we ultimately find all vectors $\boldsymbol{w}_i$ $(1 \le i \le r)$ corresponding to local maxima of $I(y_i)$. Since the independent components s_i (or the estimates y_i for them) are uncorrelated, they can be computed individually. Finally, the vectors $\boldsymbol{w}_i$ are arranged in the matrix W and we use (6.34) to compute the independent components $\boldsymbol{s}$.

Table 6.2 Typical choices for the function G and its derivatives appearing in the approximation of negentropy (6.35) and in the FastICA algorithm to determine the independent components. For general use we recommend the functions (1) with the parameter $\alpha = 1$. For signals with pronounced outliers or super-Gaussian probability distributions, we recommend the functions (2)

	$G(y)$	$G'(y)$	$G''(y)$	Remark
(1)	$\frac{1}{\alpha}\log\cosh\alpha y$	$\tanh\alpha y$	$\alpha(1-\tanh^2\alpha y)$	$1 \le \alpha \le 2$
(2)	$-\mathrm{e}^{-y^2/2}$	$y\,\mathrm{e}^{-y^2/2}$	$(1-y^2)\,\mathrm{e}^{-y^2/2}$	

The FastICA Algorithm

The independent components of the signals $\boldsymbol{x}_i = (x_1, x_2, \ldots, x_m)_i^{\mathrm{T}}$, which are measured at n consecutive times ($0 \le i \le n-1$), can be determined by the FastICA algorithm described in the following. The measured signals are first arranged in the matrix

$$X = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})^{\mathrm{T}} \in \mathbb{R}^{n\times m}\,,$$

in which each of the n rows represents one m-variate data entry (for example, voltages on m microphones at time $t = n\Delta t$). We compute the mean $\overline{\boldsymbol{x}}$ by (5.59) and the covariance matrix Σ_{xx} by (5.65). We diagonalize Σ_{xx} as $\Sigma_{xx} = U\Lambda U^{\mathrm{T}}$ and thus obtain the orthogonal matrix U and the diagonal matrix Λ. The data is then transformed as

$$\boldsymbol{x}_i \longleftarrow \Lambda^{-1/2}U^{\mathrm{T}}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})\,, \qquad i = 0, 1, \ldots, n-1\,. \tag{6.36}$$

The transformation (6.36) is known as *data whitening* and is equivalent to a decomposition of the standardized data to their principal components (see Sects. 5.8.2 and 5.8.3). Whitening removes any trace of scale or correlation from the data. We then follow the algorithm [59]:

1. Choose the number of independent components r you wish to determine.
2. Randomly initialize the vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_r$ ($\boldsymbol{w}_k \in \mathbb{R}^m$) normalized to $\|\boldsymbol{w}_k\|_2 = 1$, and arrange them in the matrix $W = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_r)^{\mathrm{T}} \in \mathbb{R}^{r\times m}$.
3. Perform a symmetric orthogonalization of W,

$$W \longleftarrow \left(WW^{\mathrm{T}}\right)^{-1/2}W\,.$$

   This step ensures that all previously found $\boldsymbol{w}_k$ are mutually orthogonal. The square root of the matrix is computed as described in Appendix A.8.
4. For each $k = 1, 2, \ldots, r$ compute new vectors

$$\boldsymbol{w}_k \longleftarrow \frac{1}{n}\left\{\sum_{i=0}^{n-1}\boldsymbol{x}_i\,G'\left(\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}_i\right) - \boldsymbol{w}_k\sum_{i=0}^{n-1}G''\left(\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}_i\right)\right\}\,, \tag{6.37}$$

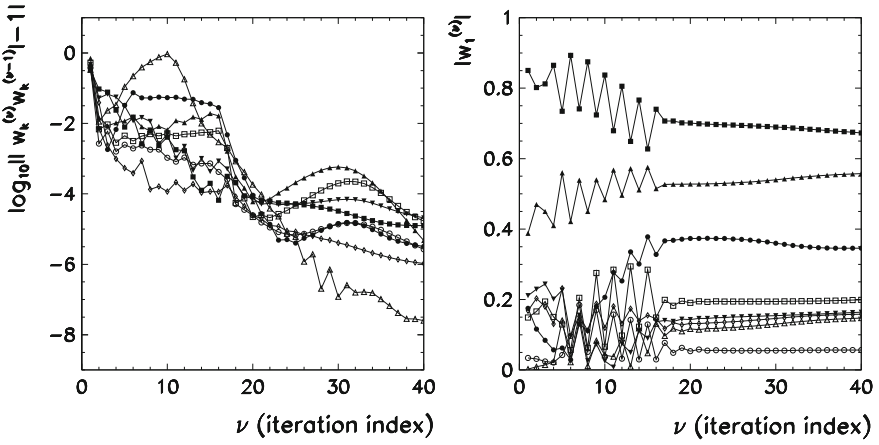


Fig. 6.17 Typical convergence in the FastICA algorithm applied to the relatively complex signals from Fig. 6.16. [LEFT] The difference of the scalar product $|\boldsymbol{w}_k^{(\nu)\mathrm{T}} \boldsymbol{w}_k^{(\nu-1)}|$ from 1 in subsequent iterations (ν) in the case eight independent components ($k = 1, 2, \ldots, 8$). [RIGHT] The convergence of eight components of the vector $\boldsymbol{w}_1$ in consecutive iterations (ν)

where the function pair $G'(y)$ and $G''(y)$ can be chosen from Table 6.2. This step is the crucial part of the algorithm ensuring that the iteration leads to the fixed point corresponding to the maximum of negentropy (6.35). Normalize the obtained vectors to unit length, $\boldsymbol{w}_k \longleftarrow \boldsymbol{w}_k / \|\boldsymbol{w}_k\|_2$.

5. Repeat items 3 and 4 until convergence is achieved in all components of the vectors $\boldsymbol{w}_k$. A good measure of convergence is the configuration in which the direction of the vector $\boldsymbol{w}_k$ in consecutive iterations (ν) and ($\nu - 1$) no longer changes, i.e. when the absolute value of the scalar product $|\boldsymbol{w}_k^{(\nu)\mathrm{T}} \boldsymbol{w}_k^{(\nu-1)}|$ is close to unity. Typically a few times ten iterations are needed (see Fig. 6.17).
6. The independent components are the rows of the matrix $S = WX^{\mathrm{T}} \in \mathbb{R}^{r \times n}$.

The FastICA algorithm allows us to compute the matrix of independent components (sources) S in which the sources appear to be arranged arbitrarily. Namely, any permutation P of the source components $\boldsymbol{s}$ in (6.33) implies just a different mixing matrix, $\boldsymbol{x} = (AP^{-1})P\boldsymbol{s}$. From the viewpoint of the ICA method, the vectors $\boldsymbol{s}$ and $P\boldsymbol{s}$ are indistinguishable.

Moreover, the signals S determined by ICA are given up to a multiplicative constant: we may choose any constant c_j to divide the source s_j and multiply the jth row of the mixing matrix A without modifying the product $A\boldsymbol{s}$. Even vectors of opposite directions ($\boldsymbol{w}_k$ and $-\boldsymbol{w}_k$ as the rows of W) that frequently occur during the iterations of the FastICA algorithm, are therefore equivalent.

Stabilization of the FastICA Algorithm

The iteration step (6.37) follows from the Newton's method to search for the maximum of negentropy, so occasionally we may encounter convergence problems. In

6.10.4 Independent Component Analysis

Independent component analysis (ICA) is a method that can be applied to a set of measured signals $\boldsymbol{x}_j(t)$ in order to determine independent components (sources) $\boldsymbol{s}_j(t)$ that caused these signals, as well as the nature of the mixing of sources. We assume that the measured signals and their sources are linearly related,

$$\boldsymbol{x}(t) = A\boldsymbol{s}(t) \ ,$$

where A is the mixing matrix (see Sect. 6.8). At each instant t the sources $\boldsymbol{s}$ (for example, speakers in a room) and the measured signals $\boldsymbol{x}$ (for example, voltages on the microphones) have several components.

⊙ In the first part of the Problem we assume that the mixing matrix is known. Suppose that we have four sources:

$$\begin{aligned} s_{1i} &= \sin(10\,\pi\, i/n) \ , \\ s_{2i} &= \exp[-10(i-n/2)^2/n^2] + 0.2[\mathcal{R}(0,1) - 0.5] \ , \\ s_{3i} &= \sin^3(40\,\pi\, i/n)\exp(-1.5i/n) \ , \\ s_{4i} &= \sin(100\, i^2/n^2) \ , \end{aligned} \tag{6.41}$$

shown in Fig. 6.27 (left). The index i measures the time, $t = i\Delta t$ ($i = 0, 1, \ldots, n-1$) and $n = 1000$. We use $\mathcal{R}(0, 1)$ to denote a uniformly distributed random number from the interval $[0, 1]$. The matrix A mixes the sources into the measured signals,

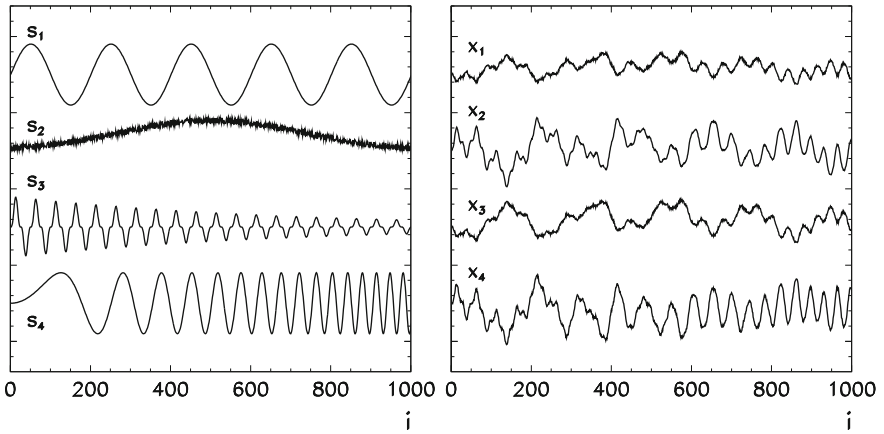


Fig. 6.27 [LEFT] The original signals (sources) from (6.41). [RIGHT] The measured signals which are known mixtures of the sources (6.42)

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{pmatrix} = \underbrace{\frac{1}{5}\begin{pmatrix} -1 & 2 & -1 & 1 \\ 2 & 1 & 3 & -2 \\ -2 & 2 & -1 & 1 \\ 1 & -2 & 3 & -3 \end{pmatrix}}_{A} \begin{pmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \\ s_4(t) \end{pmatrix}, \tag{6.42}$$

which are shown in Fig. 6.27 (right). Use the FastICA algorithm described in Sect. 6.8.1 to determine the independent components (sources) $\boldsymbol{s}$ from the signals $\boldsymbol{x}$. Compare the computed sources to the original ones (6.41).

In the case described here the number of sources is equal to the number of measured signals, so the model $\boldsymbol{x} = A\boldsymbol{s}$ is exactly invertible, $\boldsymbol{s} = W\boldsymbol{x} = A^{-1}\boldsymbol{x}$. The computed and exact sources can therefore be accurately compared. Discuss the case with fewer sources than signals (invent your own mixing matrix).

⊕ Perform the independent component analysis of eight electro-cardiogram (ECG) traces of a pregnant woman shown in Fig. 6.16 [56]. There are 2500 voltage readouts with a sampling frequency of 500 Hz. The final result of the analysis are the eight independent components which should closely resemble those of Fig. 6.28. Observe the convergence of the algorithm by monitoring the direction of the vectors

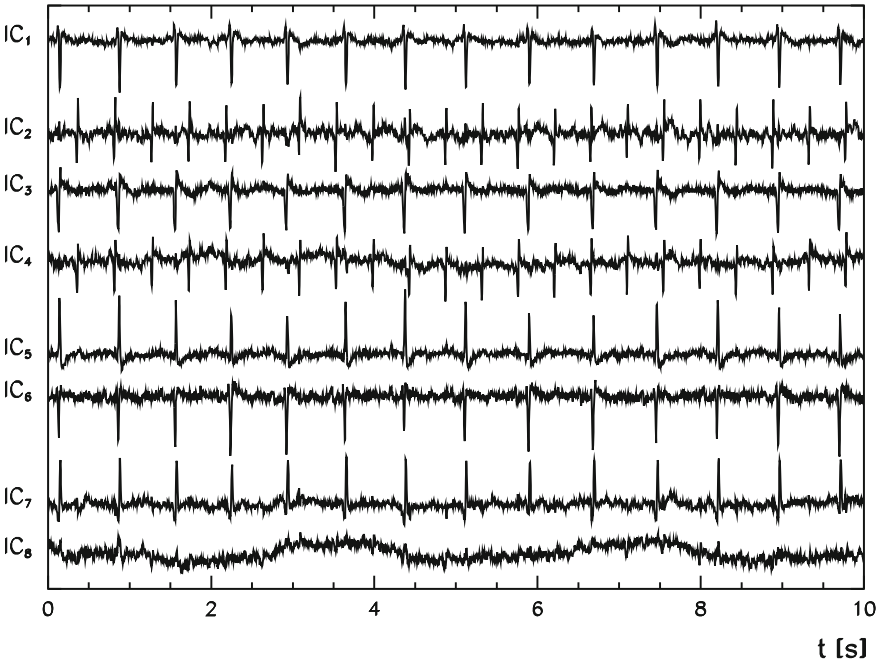


Fig. 6.28 Independent components (sources) of the signals shown in Fig. 6.16. The child's fast heartbeat is clearly identifiable in the independent components IC_2 and IC_4. Other components display the mother's heartbeat, except IC_8 which has probably been generated by breathing during the measurement

taken care of. We can explicitly solve (7.33) for x and thus obtain a system of linear equations for the quantities $\boldsymbol{k}_i$:

$$\boldsymbol{k}_i = h\boldsymbol{f}\left(x_n + \alpha_i h\,,\, \boldsymbol{y}_n + \sum_{j=1}^{i-1} \alpha_{ij}\boldsymbol{k}_j\right) + h\, J_y^{(n)} \sum_{j=1}^{i} \gamma_{ij}\boldsymbol{k}_j + h^2\gamma_i \boldsymbol{J}_x^{(n)}\,.$$

The concluding part of each time step — (7.34) — remains the same. Above we have used the abbreviations

$$\alpha_i = \sum_{j=1}^{i-1} \alpha_{ij}\,, \qquad \gamma_i = \sum_{j=1}^{i} \gamma_{ij}\,, \qquad \boldsymbol{J}_x^{(n)} = \frac{\partial \boldsymbol{f}}{\partial x}(x_n, \boldsymbol{y}_n)\,, \qquad J_y^{(n)} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(x_n, \boldsymbol{y}_n)\,.$$

Implicit differential equations Implicit equations of the form $M\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$ with constant non-singular matrices M can be rewritten in the equivalent form $\boldsymbol{y}' = M^{-1}\boldsymbol{f}(x, \boldsymbol{y})$, resulting in

$$M\boldsymbol{k}_i = h\boldsymbol{f}\left(x_n + \alpha_i h,\ \boldsymbol{y}_n + \sum_{j=1}^{i-1} \alpha_{ij}\boldsymbol{k}_j\right) + h J_y^{(n)} \sum_{j=1}^{i} \gamma_{ij}\boldsymbol{k}_j + h^2\gamma_i \boldsymbol{J}_x^{(n)}\,.$$

By simple transformations among the variables and by exploiting the banded structure of the matrices M and J_y, the numerical efficiency of Rosenbrock methods can be strongly enhanced [20]. In similar ways, problems with non-constant matrices $M(x)$ can be harnessed [23].

7.10 Stiff Problems

To obtain a feeling about the concept of stiffness, consider the problem

$$y'(x) = -100\big(y(x) - \cos x\big) - \sin x\,, \qquad y(0) = 1\,, \tag{7.35}$$

with the analytic solution $y(x) = \cos x$ (example adapted from [24]). Figure 7.10 shows its numerical solution by using the implicit (IE, left) and explicit (EE, right) Euler's method. The correct solution is the smooth curve in the center of the left panel. All other solutions with different initial conditions converge to this solution, but only after rapidly ramping up or down, following almost vertical trajectories. Such short-lived transient behavior that is dictated by the differential equation itself but is absent in the actual solution is one of the landmarks of a stiff problem.

Figure 7.10 also reveals that the implicit Euler's method nicely follows the solution while the explicit variant diverges unless the step size is sufficiently reduced. Providing stability in addition to accuracy is therefore another challenge in stiff

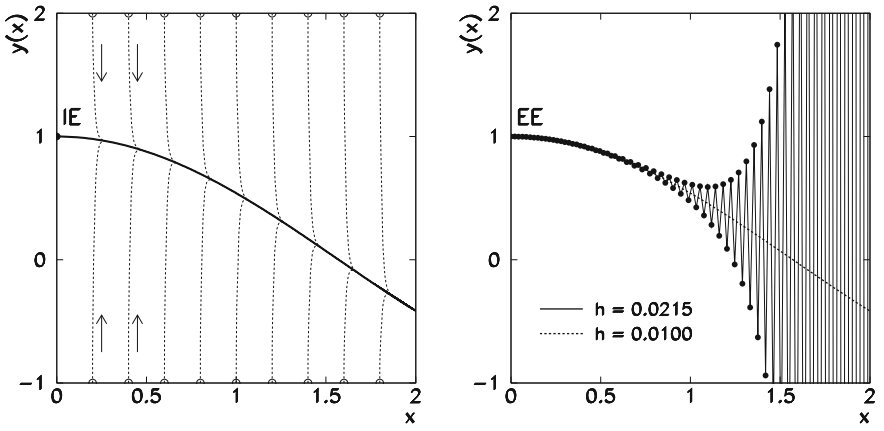


Fig. 7.10 Solving the stiff differential equation $y' = -100(y - \cos x) - \sin x$ with the initial condition $y(0) = 1$. [LEFT] Using the implicit Euler's method with step size $h = 0.0215$. Also shown are families of solutions with various initial conditions along the $y = 2$ and $y = -1$ axes. [RIGHT] Solutions by the explicit Euler's method with $h = 0.0215$ (unstable) and $h = 0.0100$ (stable)

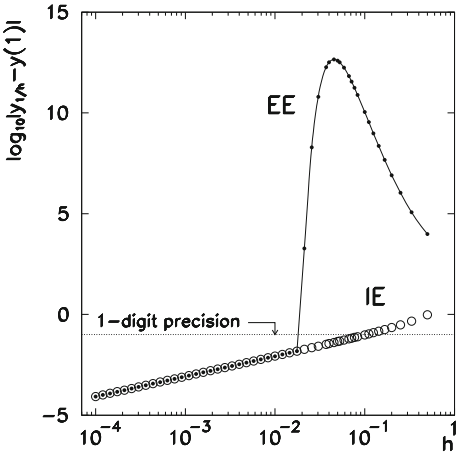


Fig. 7.11 The error of the numerical solution of (7.35) at $x = 1$ as a function of the step size h. The explicit Euler scheme (EE) is unstable unless a small enough h is used, while the implicit method (IE) is stable for any h

problems. Figure 7.11 shows that for, say, single-digit precision of $y(1)$, the explicit method requires about five times shorter steps than the implicit one, and the disparity becomes much more pronounced if the leading coefficient -100 in (7.35) is increased. Thus, as a rule, stiff problems should be integrated by using implicit schemes: although in general the solution of implicit equations at each step may represent a complication, implicitness provides the needed stabilization.

This leads to the suspicion that initial-value problems with ordinary differential equations might be stiff when two or more very different scales are involved in the solution, for example, two characteristic times. Think of a robotic arm that moves with angular velocities of $\approx$1 Hz, which we hit by a small hammer, exciting oscillations

which has four maxima at $(x, y) = (1, \pm 1)$ and $(-1, \pm 1)$ with values $E_\mathrm{m} = 1/\mathrm{e}^2$ [61]. Let the particle with mass $M = 1$ and energy E impinge towards the central part of the potential from large distances with impact parameter b (parallel to the x-axis and at constant distance b from it).

Plot some typical particle trajectories in the (x, y) plane as a function of the parameter b (Fig. 7.25 (top right)). Compute the scattering angles ϕ for a large set of values of b, $-3 \le b \le 3$, at $E/E_\mathrm{m} = 1.626$ (regular scattering) and at $E/E_\mathrm{m} = 0.260$ (chaotic scattering). Look closer at the obtained results in the ever narrower regions of b, e.g. for $-0.6 < b < -0.1$, $-0.400 < b < -0.270$, and $-0.3920 < b < -0.3770$ (Fig. 7.25 (bottom left and right)). Compute the time delay (time spent by the particle in the central part of the potential) as a function of b. By using a constant step-size integrator the delay is simply proportional to the number of steps taken.

7.14.11 Hydrogen Burning in the pp I Chain

Fusion of hydrogen nuclei occurs in stars with masses less than $\approx 1.3\, M_\odot$ in three branches of the pp chain. The first stage (pp I branch) involves the processes [62]

$$
\begin{aligned}
{}^1\mathrm{H} + {}^1\mathrm{H} &\longrightarrow {}^2\mathrm{H} + \mathrm{e}^+ + \nu_\mathrm{e} && (\lambda_\mathrm{pp})\,, \\
{}^1\mathrm{H} + {}^1\mathrm{H} + \mathrm{e}^- &\longrightarrow {}^2\mathrm{H} + \nu_\mathrm{e} && (\lambda'_\mathrm{pp})\,, \\
{}^2\mathrm{H} + {}^1\mathrm{H} &\longrightarrow {}^3\mathrm{He} + \gamma && (\lambda_\mathrm{pd})\,, \\
{}^3\mathrm{He} + {}^3\mathrm{He} &\longrightarrow {}^4\mathrm{He} + {}^1\mathrm{H} + {}^1\mathrm{H} && (\lambda_{33})\,,
\end{aligned}
$$

where λ_i are the reaction rates. The pp I branch contributes 85% to the solar luminosity, but the process rates in it are very different due to a large span of reaction cross-sections and Coulomb barriers [63, 64]. We express λ_i in units of reactions per unit time per $(\mathrm{mol/cm}^3)^{N-1}$, where N is the number of interacting particles excluding photons. At temperatures in the solar interior ($\approx 15 \cdot 10^6$ K) they are $\lambda_\mathrm{pp} = 8.2 \cdot 10^{-20}$, $\lambda'_\mathrm{pp} = 2.9 \cdot 10^{-24}$, $\lambda_\mathrm{pd} = 1.3 \cdot 10^{-2}$, $\lambda_{33} = 2.3 \cdot 10^{-10}$.

For the listed processes $\lambda'_\mathrm{pp} \ll \lambda_\mathrm{pp} \ll \lambda_{33} \ll \lambda_\mathrm{pd}$. The first two processes are by far the slowest ones, as they are governed by the weak interaction. Here we approximate $\lambda'_\mathrm{pp} = 0$. The equations for the pp I branch then become

$$
\begin{aligned}
\frac{\mathrm{d}n_\mathrm{p}}{\mathrm{d}t} &= -\lambda_\mathrm{pp} n_\mathrm{p}^2 - \lambda_\mathrm{pd} n_\mathrm{p} n_\mathrm{d} + \lambda_{33} n_3^2\,, \\
\frac{\mathrm{d}n_\mathrm{d}}{\mathrm{d}t} &= \lambda_\mathrm{pp} \frac{n_\mathrm{p}^2}{2} - \lambda_\mathrm{pd} n_\mathrm{p} n_\mathrm{d}\,, \\
\frac{\mathrm{d}n_3}{\mathrm{d}t} &= \lambda_\mathrm{pd} n_\mathrm{p} n_\mathrm{d} - \lambda_{33} n_3^2\,, \\
\frac{\mathrm{d}n_4}{\mathrm{d}t} &= \lambda_{33} \frac{n_3^2}{2}\,,
\end{aligned}
\qquad (7.69)
$$

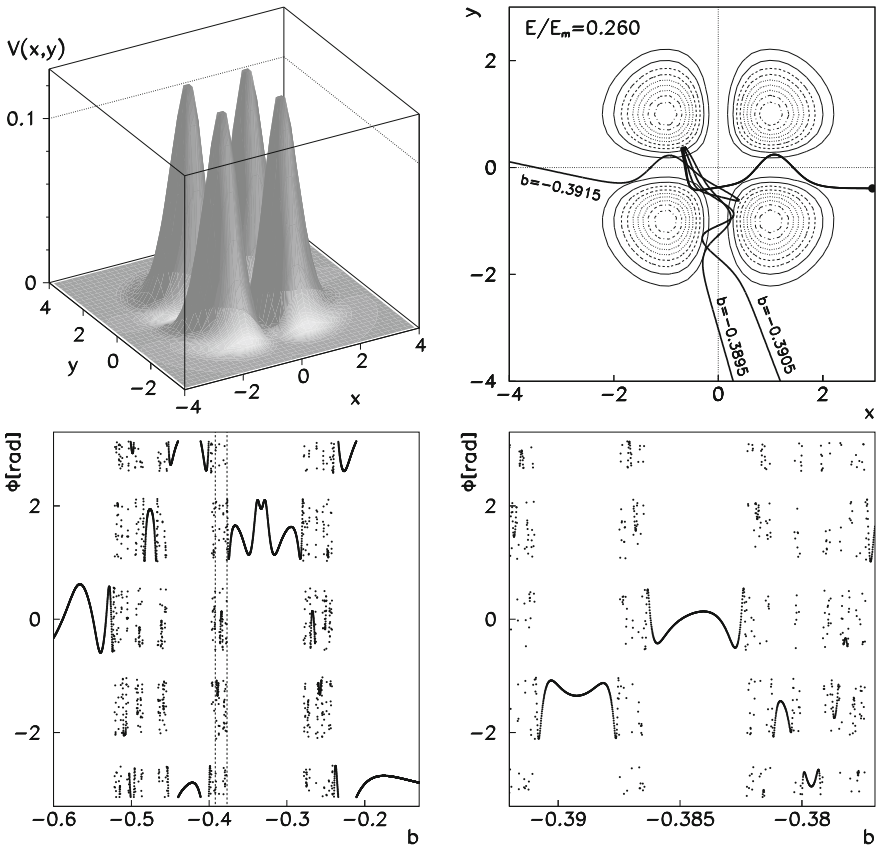


Fig. 7.25 Chaotic scattering on the potential $V(x, y) = x^2y^2 \exp\left[-(x^2 + y^2)\right]$. [TOP LEFT] The potential $V(x, y)$. [TOP RIGHT] Dependence of the solution on initial conditions $x(0) = 3$, $y(0) = b$, $\dot{x}(0) = 0$, and $\dot{y}(0) = -\sqrt{2.0(E - V(x(0), y(0)))/M}$ with parameters $M = 1$, $E = 0.260E_{\mathrm{m}}$. Shown are the solutions at three different impact parameters $b = -0.3915$, 0.3905, and 0.3895. [BOTTOM LEFT] Dependence of the scattering angle ϕ on b with $-0.60 \leq b \leq -0.13$. [BOTTOM RIGHT] The zoom-in in the region $-0.392 \leq b \leq -0.377$ (denoted by dashed vertical lines in the Figure at left). We are witnessing classical chaos: we observe a large sensitivity of the system to the initial conditions, and self-similarity (equal or similar structures appear at different scales)

where we have denoted the isotope concentrations by $n_{\mathrm{p}} = n(^1\mathrm{H})$, $n_{\mathrm{d}} = n(^2\mathrm{H})$, $n_3 = n(^3\mathrm{He})$, and $n_4 = n(^4\mathrm{He})$.

⊙ Solve the system (7.69) on the time interval $10^{10}\mathrm{s} \leq t \leq 10^{22}\mathrm{s}$. The initial condition for n_{p} can be computed from the estimate $(\lambda_{\mathrm{pp}} n_{\mathrm{p}})^{-1} \approx 10^{10}$ years which is valid at the conditions in the solar interior, and we set $n_{\mathrm{d}} = n_3 = n_4 = 0$. At first you may restrict the computation to large times, when $^2\mathrm{H}$ and $^3\mathrm{He}$ are in equilibrium ($\mathrm{d}n_{\mathrm{d}}/\mathrm{d}t = \mathrm{d}n_3/\mathrm{d}t = 0$). In that case the stiff system of four equations reduces to a non-stiff system of two equations. To consider all times except the shortest, assume that $^2\mathrm{H}$ is in equilibrium: then the equation for $^3\mathrm{He}$ can be solved analytically by

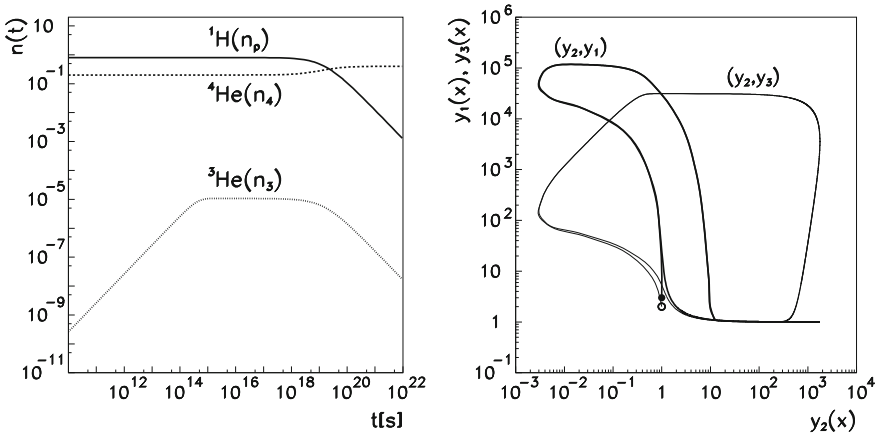


Fig. 7.26 Examples of stiff differential equations. [LEFT] Hydrogen burning in the pp I chain (7.69). The solution for n_d lies below the shown area. [RIGHT] Limit cycles of the Oregonator (7.70). The symbols ○ and ● denote the initial conditions

assuming that the concentrations of ^{1}H and ^{4}He do not change substantially, and by using the solution for n_3 in the equations for n_p and n_4. For finding the solution at arbitrary times, use an integrator tailored to stiff systems (see Fig. 7.26 (left)).

⊕ Augment the basic system of equations for the pp I branch with contributions of heavier isotopes, and compare the results. Use the reactions given in [62–65], which list the reaction rates also for temperatures that are different from those in the solar interior.

7.14.12 Oregonator

Oregonator is a domesticated name for the chemical reaction of $HBrO_2$, Br^-, and Ce(IV). The dynamics is described by a stiff set of equations [66]

$$\begin{aligned} y_1' &= \kappa_1 \left(y_2 + y_1 \left(1 - \alpha y_1 - y_2 \right) \right) , \\ y_2' &= \kappa_2 \left(y_3 - y_2 \left(1 + y_1 \right) \right) , \\ y_3' &= \kappa_3 \left(y_1 - y_3 \right) , \end{aligned} \tag{7.70}$$

where $\kappa_1 = 1/\kappa_2 = 77.27$, $\kappa_3 = 0.161$, and $\alpha = 8.375 \cdot 10^{-6}$. As expected from a stiff system, the solutions change over many orders of magnitude (Fig. 7.26 (left and right)).

⊙ Solve the system (7.70) with initial conditions $y_1(0) = 3$, $y_2(0) = 1$, $y_3(0) = 2$. Use an explicit integrator of your own choice and the implicit fifth-order integrator Radau 5 described on p. 417. If possible, resort to adaptive step size control in both cases. Plot the solutions $y_1(x)$, $y_2(x)$, and $y_3(x)$ for $0 \le x \le 360$. The solutions

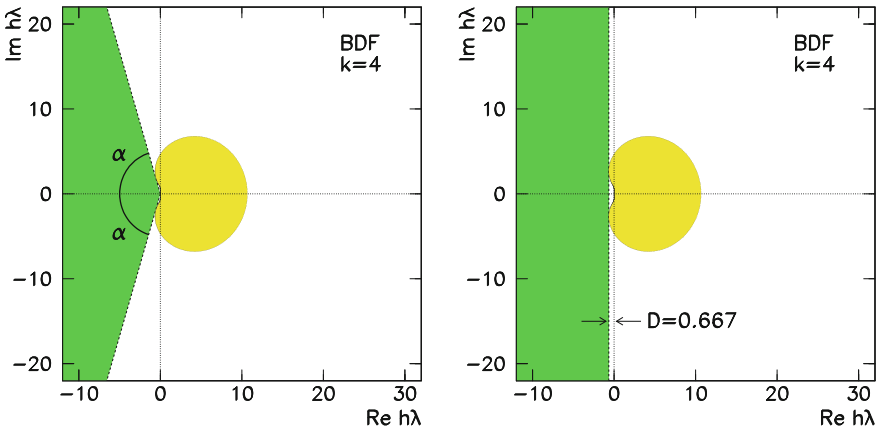


Fig. 7.14 Stability of the fourth-order BDF method (7.27). [LEFT] $A(\alpha)$-stability with angle $\alpha = 73.35°$. [RIGHT] Stiff stability with $D = 0.667$. See also Fig. 7.7

method is therefore $A(\pi/2)$-stable. We also define "stiff stability" for which we require the stability region $\mathrm{Re}\, z < -D$ for some $D > 0$ and a "sufficient precision" of the method within the rectangle $-D \le \mathrm{Re}\, z \le a$, $-\theta \le \mathrm{Im}\, z \le \theta$ for some $a > 0$ and $\theta \sim \pi/5$ (Fig. 7.8 (right)).

BDF methods of orders $3 \le p \le 6$ (Fig. 7.7) correspond to $\alpha = 86.03°$, $D = 0.083$ ($k = 3$), $\alpha = 73.35°$, $D = 0.667$ ($k = 4$), $\alpha = 51.84°$, $D = 2.327$ ($k = 5$), $\alpha = 17.84°$, $D = 6.075$ ($k = 6$). Figure 7.14 shows the region of $A(\alpha)$-stability and stiff stability for the fourth-order method (7.27). $A(\alpha)$-stable multi-step methods of higher orders with $\alpha \lesssim \pi/2$ do exist, but they possess large leading error constants and they are only of limited use. In order to cross the Dahlquist barrier more general multi-step methods can be devised: see [20].

7.12 Geometric Integration ⋆

In the following we use the variable pair $(t, \boldsymbol{y})$ instead of $(x, \boldsymbol{y})$, in the spirit of dynamical analysis pervading this section. We are interested in the solutions of equations $\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y})$ where $\dot{}$ denotes the time derivative, especially in the context of autonomous Hamiltonian systems [27]. Such systems are described by the Hamiltonians $H(\boldsymbol{p}, \boldsymbol{q})$, where

$$\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y}) = J^{-1} \nabla H(\boldsymbol{y}) \,, \qquad \boldsymbol{y} = \begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{pmatrix} \,, \qquad J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \,, \tag{7.42}$$

$\boldsymbol{p} \in \mathbb{R}^d, \boldsymbol{q} \in \mathbb{R}^d$ and $\nabla = (\nabla_p, \nabla_q)^{\mathrm{T}} = (\partial_{p_1}, \partial_{p_2}, \cdots, \partial_{p_d}, \partial_{q_1}, \partial_{q_2}, \cdots, \partial_{q_d})^{\mathrm{T}}$. The dynamical equations for the canonical variables $\boldsymbol{p}$ and $\boldsymbol{q}$ are

$$\dot{\boldsymbol{p}} = -\nabla_q H(\boldsymbol{p}, \boldsymbol{q}) \,, \qquad \dot{\boldsymbol{q}} = \nabla_p H(\boldsymbol{p}, \boldsymbol{q}) \tag{7.43}$$

(see also Appendix H). How successful are the methods of previous sections in such problems? We provide the answer from several viewpoints. All single-step methods discussed so far can be understood as mappings of the solution from "time" nh to "time" $(n+1)h$,

$$\boldsymbol{y}_{n+1} = \phi_h(\boldsymbol{y}_n) \,. \tag{7.44}$$

Does the numerical solution preserve the invariants of the continuous problem, for example, the Hamiltonian $H(\boldsymbol{p}, \boldsymbol{q})$? Does numerical integration of the Hamiltonian system (the solution of the initial-value problem) preserve the symplectic structure of the phase space? We may also ask whether the integrator is symmetric and reversible. A numerical method that satisfies at least one of these requirements is known as a *geometric integrator*.

7.12.1 Preservation of Invariants

A non-constant function $I(\boldsymbol{y})$ is called the *first integral* (or *invariant*, or *constant of motion*) of (7.42) if

$$\nabla I(\boldsymbol{y}) \cdot \boldsymbol{f}(\boldsymbol{y}) = 0 \qquad \forall \boldsymbol{y} \,. \tag{7.45}$$

This means that at any point of the phase space (along any solution $\boldsymbol{y}$) the gradient $\nabla I(\boldsymbol{y})$ is orthogonal to the vector field $\boldsymbol{f}(\boldsymbol{y})$. A nice scalar example is the mathematical pendulum with the Hamiltonian $H(p,q) = \frac{1}{2}p^2 - \cos q$. The equations of motion (Newton's law) are $\dot{p} = -\sin q$ and $\dot{q} = p$, which can be written in the form (7.42) with $\boldsymbol{f}(\boldsymbol{y}) = (-\sin q, p)^{\mathrm{T}}$. Obviously $\nabla H(\boldsymbol{y}) \cdot \boldsymbol{f}(\boldsymbol{y}) = (p, \sin q)(-\sin q, p)^{\mathrm{T}} = 0$. A further example are the three invariants of the classical Kepler problem, discussed in Problem 7.14.13.

Let us test some integrators on an even simpler case of the one-dimensional harmonic oscillator (linear pendulum) with the spring constant k^2, described by the Hamiltonian

$$H(p,q) = \frac{1}{2}\left(p^2 + k^2 q^2\right) \,, \tag{7.46}$$

with the analytic solution

$$\begin{pmatrix} \widetilde{p}(h) \\ \widetilde{q}(h) \end{pmatrix} = \begin{pmatrix} \cos h & -k^2 \sin h \\ \sin h & \cos h \end{pmatrix} \begin{pmatrix} p(0) \\ q(0) \end{pmatrix} \tag{7.47}$$

at time $t = h$. The basic explicit Euler's method (7.5) is first order, so it also approximates the solution (7.47) only to first order. This implies

$$\begin{pmatrix} \widetilde{p}(h) \\ \widetilde{q}(h) \end{pmatrix}_{\text{Euler}} = \begin{pmatrix} 1 & -k^2 h \\ h & 1 \end{pmatrix} \begin{pmatrix} p(0) \\ q(0) \end{pmatrix}, \tag{7.48}$$

which can be seen if one explicit Euler's step is done for (7.43). Instead of the correct value of the energy (7.46) we get

$$\tfrac{1}{2}(\widetilde{p}^2 + k^2\widetilde{q}^2) = \tfrac{1}{2}(1 + k^2h^2)(p^2 + k^2q^2)\,,$$

which is unbounded when the number of steps goes to infinity, regardless of the step size h. The RK4 method does not fare much better, as it results in damping

$$\tfrac{1}{2}(\widetilde{p}^2 + k^2\widetilde{q}^2) = \tfrac{1}{2}\left(1 - k^6h^6/72\right)(p^2 + k^2q^2)\,.$$

We also obtain damping with the implicit Euler's method. Figure 7.15 (left) shows the solution (p, q) by the explicit Euler's (7.5), implicit Euler's (7.30), and implicit midpoint method (7.31) in the form for an autonomous Hamiltonian system:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + hJ^{-1}\nabla H\left(\tfrac{1}{2}(\boldsymbol{y}_n + \boldsymbol{y}_{n+1})\right)\,. \tag{7.49}$$

In general, explicit and implicit RK methods preserve only linear invariants. As an example [20], consider the conservation of mass in the process (7.38): from the system of equations we see $\dot{y}_1 + \dot{y}_2 + \dot{y}_3 = 0$, so $I(\boldsymbol{y}) = y_1 + y_2 + y_3$ is a linear invariant of the system. If the coefficients of an m-stage RK method (see (7.8) and Sect. 7.9) satisfy

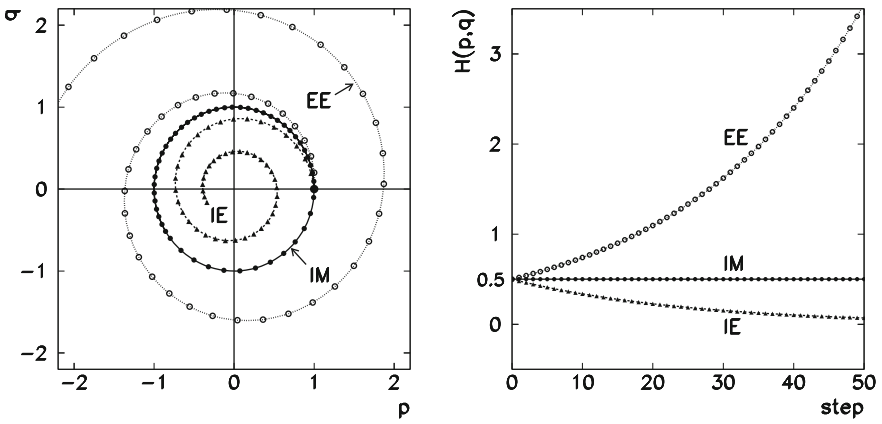


Fig. 7.15 [LEFT] The solution of the harmonic oscillator problem $\dot{p} = -k^2q$, $\dot{q} = p$ with initial condition $(p, q) = (1, 0)$ (large symbol •) and $k = 1$ by explicit Euler's (EE), implicit Euler's (IE), and implicit midpoint method (IM) in 50 steps of $h = 0.2$. [RIGHT] The values of $H(p, q) = \frac{1}{2}(p^2 + k^2q^2)$ at current p and q for these three methods

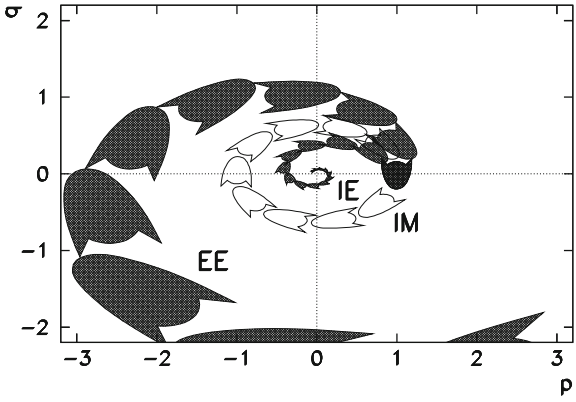


Fig. 7.16 Evolution of the Arnold's cat in the (p, q) plane for the harmonic oscillator problem with $k = 1.6$. The explicit Euler's method (EE) enlarges the area while the implicit (IE) reduces it. The implicit midpoint method (IM) preserves the area but not its shape

the trajectory does not change; we just reverse the motion. An invertible linear transformation ρ helps us to define a more general notion of ρ-reversibility. A differential equation $\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y})$ and the vector field $\boldsymbol{f}(\boldsymbol{y})$ are ρ-reversible if

$$\rho \boldsymbol{f}(\boldsymbol{y}) = -\boldsymbol{f}(\rho \boldsymbol{y}) \qquad \forall \boldsymbol{y} \,. \tag{7.57}$$

An analogous concept can be defined for single-step numerical methods (7.44). The method ϕ_h is symmetric (with respect to time reversal) if $\phi_h \circ \phi_{-h} = 1$. If the method ϕ_h, applied to a ρ-reversible differential equation, satisfies

$$\rho \circ \phi_h = \phi_{-h} \circ \rho \,, \tag{7.58}$$

then ϕ_h is a ρ-reversible mapping precisely when ϕ_h is symmetric. (Symmetry and ρ-reversibility of a discrete map are equivalent properties.) All explicit and implicit methods of the RK type satisfy (7.58) if (7.57) is valid. Partitioned RK methods satisfy the condition (7.58) if ρ can be written in the form $\rho(\boldsymbol{u}, \boldsymbol{v}) = (\rho_1(\boldsymbol{u}), \rho_2(\boldsymbol{v}))$, where ρ_1 and ρ_2 are invertible mappings. The symplectic Euler's methods (7.54) are not symmetric (or ρ-reversible). The Störmer–Verlet methods (7.55) and (7.56) are symmetric and therefore also ρ-reversible [29].

7.12.4 Modified Hamiltonians and Equations of Motion

Unfortunately it is difficult to find an integration method for general (also non-integrable) Hamiltonian systems that would preserve the symmetry properties of the system and its invariants, and at the same time fulfill the symplectic condition [30, 31]. We therefore often opt for methods satisfying at least one of these requirements: for the integration of astronomical orbits we might prefer to ensure the periodicity and preservation of energy, while in the study of charged particle motion in storage

Table 9.1 Select explicit (E) and implicit (I) difference schemes for the solution of the one-dimensional diffusion equation $v_t - Dv_{xx} = Q$, where $r = D\Delta t/\Delta x^2$

Scheme	Type	Order of error	Stability
FTCS (9.13)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$	$0 < r \leq 1/2$
Leapfrog (9.35)	E	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	unstable
Dufort-Frankel (9.36)	E	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	stable
FTCS5 (9.37)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^4)$	$0 < r \leq 3/8$
BTCS (9.20)	I	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$	$\forall r > 0$
Crank–Nicolson (9.21)	I	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	$\forall r > 0$

$$u_j^{n+1} = \frac{2r}{1+2r}\left(u_{j+1}^n + u_{j-1}^n\right) + \frac{1-2r}{1+2r}\,u_j^{n-1} + \frac{1}{1+2r}\,\Delta t q_j^n\,, \tag{9.36}$$

which is unconditionally stable, but only conditionally consistent (and thus only conditionally convergent). Convergence is ensured if r is constant [1].

On the other hand, we can increase the order of the FTCS scheme in the space variable by replacing the three-point difference by a five-point formula,

$$u_j^{n+1} = u_j^n + r\left(-\frac{1}{12}\,u_{j+2}^n + \frac{4}{3}\,u_{j+1}^n - \frac{5}{2}\,u_j^n + \frac{4}{3}\,u_{j-1}^n - \frac{1}{12}\,u_{j-2}^n\right) + \Delta t q_j^n\,. \tag{9.37}$$

The resulting scheme is of order $\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^4)$ and remains stable, but additional boundary conditions must be specified at the points x_{-1} and x_{N+1} that lie outside of the definition domain (Fig. 9.2 (right)). In this case we prescribe *numerical boundary conditions*. We solve the diffusion equation with homogeneous Dirichlet condition to this order if we use $u_{-1} = u_{N+1} = 0$ (solution vanishes at the ghost points) or $u_{-1}^n - 2u_0^n + u_1^n = u_{N+1}^n - 2u_N^n + u_{N-1}^n = 0$ (the second derivatives at the endpoints of the rod are zero).

We have described just a few representative explicit schemes for the solution of the one-dimensional diffusion equation. By Taylor expansions and the method of undetermined coefficients it is possible to construct many other schemes [1]. As in ordinary differential equations (Chap. 7), an alternative is offered by the implicit schemes (Table 9.1).

It is difficult to formulate a general advice when to use a high-order scheme. Now and then high-order schemes are a good choice, but if the numerical cost allows it, a finer discretization might be preferable to increasing the order. Table 9.1 summarizes the basic properties of the methods for the solution of (9.2).

Table 9.2 Consistency and stability properties of the explicit (E) and implicit (I) difference schemes for solving the one-dimensional hyperbolic problem $v_t + cv_x = 0$ with the parameter $R = c\Delta t/\Delta x$

Scheme	Type	Order of error	Stability
FTFS (9.39)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x)$	$-1 \le R \le 0$
FTBS (9.40)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x)$	$0 \le R \le 1$
FTCS (9.41)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$	Unstable
Lax–Wendroff (9.42)	E	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	$\lvert R\rvert \le 1$
Lax–Friedrichs (9.43)	E	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2/\Delta t)$	$\lvert R\rvert \le 1$
BTFS (9.44)	I	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x)$	$R \le 0$
BTBS (9.45)	I	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x)$	$R \ge 0$
BTCS (9.46)	I	$\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$	$\forall R$
Lax–Wendroff (9.47)	I	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	$\forall R$
Crank–Nicolson (9.48)	I	$\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$	$\forall R$

The approximations $v_x \approx (\Delta_0^{(x)} u_j^n)/\Delta x$ and $v_{xx} \approx (\Delta_2^{(x)} u_j^n)/\Delta x^2$ give us the *linear Lax–Wendroff scheme*

$$u_j^{n+1} = u_j^n - \frac{R}{2}\,\Delta_0^{(x)} u_j^n + \frac{R^2}{2}\,\Delta_2^{(x)} u_j^n\,, \tag{9.42}$$

with the error of order $\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$. (Beware that in this book and in literature there are other schemes that carry the same name.) From the symbol $|\rho(\xi)|^2 = 1 - 4R^2 \sin^2(\xi/2) + 4R^4 \sin^4(\xi/2)$ we infer the stability criterion $|R| \le 1$. We have sinned a bit. If we read the scheme backwards, the equation

$$v_t + cv_x = \frac{c^2 \Delta t}{2}\, v_{xx}\,.$$

emerges: a diffusive (dissipative) term has appeared at the right-hand side of the equation that damps the solution for all $t > 0$. The consequences will be discussed in Sect. 9.10. Let us also mention the Lax–Friedrichs scheme

$$u_j^{n+1} = \frac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) - \frac{R}{2}\Delta_0^{(x)} u_j^n\,, \tag{9.43}$$

which is conditionally stable ($|R| \le 1$) and consistent to $\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2/\Delta t)$. Both schemes will become more familiar in Problem (9.13.2) and will be useful for the solution of PDE that can be expressed in conservative form (Sect. 9.12).

in the second, we include the origin (thus $a = 0$), so that the first initial condition (10.56) does not apply. In both problems, the spatial part of the difference operator is the same as for the diffusion equation (10.53), so precisely that discretization can be adopted. We span the mesh (r_j, θ_k) on the annulus such that $r_0 = a$, $r_{N_r} = 1$, $\theta_0 = 0, \theta_{N_\theta} = 2\pi$, $\Delta r = (1-a)/N_r$, and $\Delta\theta = 2\pi/N_\theta$; if the origin is included, we have $r_0 = 0$ and $\Delta r = 1/N_r$. For either of the problems, this boils down to

$$-\frac{1}{\Delta r^2}\frac{1}{r_j}\left[r_{j+1/2}\left(u_{j+1k}-u_{jk}\right)-r_{j-1/2}\left(u_{jk}-u_{j-1k}\right)\right]-\frac{1}{\Delta\theta^2}\frac{1}{r_j^2}\underline{\Delta_2^{(\theta)}u_{jk}}=q_{jk}\,,$$

where $q_{jk} = Q(r_j, \theta_k)$ for $j = 1, 2, \ldots, N_r - 1$ and $k = 1, 2, \ldots, N_\theta - 1$. When $k = 0$, the scheme accesses the point u_{j-1} in the underlined term; periodicity in θ comes to rescue by setting $u_{j-1} = u_{jN_\theta-1}$. The remaining components of the solution are given by the boundary conditions. In the annulus problem we have

$$\begin{aligned} u_{jN_\theta} &= u_{j0}\,, & j &= 0, 1, \ldots, N_r\,,\\ u_{0k} &= f_1(\theta_k)\,, & k &= 0, 1, \ldots, N_\theta\,,\\ u_{N_rk} &= f_2(\theta_k)\,, & k &= 0, 1, \ldots, N_\theta\,. \end{aligned}$$

In the problem that includes the origin we must be careful about—the origin: the values u_{0k} and q_{0k} actually can not depend on k, thus we abbreviate $u_{0k} = u_0$ and $q_{0k} = q_0$. The boundary conditions then become

$$\begin{aligned} u_{jN_\theta} &= u_{j0}\,, & j &= 0, 1, \ldots, N_r\,,\\ u_{N_rk} &= f_2(\theta_k)\,, & k &= 0, 1, \ldots, N_\theta\,, \end{aligned}$$

and

$$\frac{4}{\Delta r^2}u_0-\frac{2\Delta\theta}{\pi\Delta r^2}\sum_{k=0}^{N_\theta-1}u_{1k}=q_0\,.$$

10.5 Boundary Element Method ⋆

The *boundary element method* (BEM) allows us to solve PDE in non-trivial geometries in which the discretization of the interior of the definition domain is difficult, while it is relatively easy to express (at least in some approximation) the boundary conditions on the boundaries of this domain. This is what makes the BEM method so appealing: just by using the information from the boundaries we solve the problem on the whole domain.

Here the basic outline of BEM is presented, following closely [9]. As an example we discuss the two-dimensional Laplace equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \,, \tag{10.57}$$

which we try to solve in the xy-plane in the domain R bounded by the piecewise smooth closed curve C. Along the individual segments C_i of C, either Dirichlet or Neumann boundary conditions are specified:

$$\begin{aligned} \phi(x, y) &= f_i(x, y) \,, \qquad (x, y) \in C_i \,, \\ \frac{\partial \phi(x, y)}{\partial n} &= f_j(x, y) \,, \qquad (x, y) \in C_j \,, \end{aligned}$$

as shown in Fig. 10.8 (left). The normal derivative $\partial\phi/\partial n = n_x(\partial\phi/\partial x) + n_y(\partial\phi/\partial y)$ is defined by the components of the unit normal vector $\boldsymbol{n} = (n_x, n_y)^{\mathrm{T}}$, which points away from the domain R. A physical example of such a problem is the stationary state of heat conduction in isotropic matter. The solution $\phi(x, y)$ describes the distribution of temperature in the domain R with the boundary C, some pieces of which are held at constant temperature, while the others experience a constant heat flux $-\lambda(\partial\phi/\partial n)$.

The particular solution of (10.57) is $\phi(x, y) = A \log\sqrt{x^2 + y^2} + B$ for $(x, y) \neq (0, 0)$. We choose $A = 1/(2\pi)$, $B = 0$, and move the origin from $(0, 0)$ to (ξ, η). This results in the *fundamental solution* of the Laplace equation

$$\Phi(x, y; \xi, \eta) = \frac{1}{4\pi} \log\left[(x - \xi)^2 + (y - \eta)^2\right] \,,$$

which is defined everywhere except at (ξ, η). Gauss theorem for vector functions, $\int_C \boldsymbol{V} \cdot \boldsymbol{n}\, \mathrm{d}s(x, y) = \int_R \nabla \cdot \boldsymbol{V}\, \mathrm{d}x\, \mathrm{d}y$, can be applied to show that for any two solutions ϕ_1 and ϕ_2 of (10.57), we have

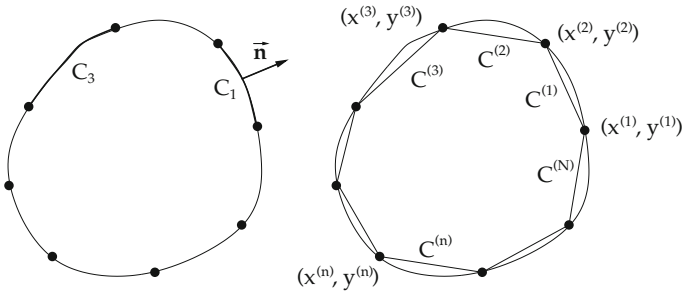


Fig. 10.8 [LEFT] The domain R with the smooth boundary C on which we solve the two-dimensional Poisson equation. As an example, we have Dirichlet boundary conditions on segment C_3 and Neumann boundary conditions on C_1. [RIGHT] The approximation of the boundary C by the inscribed polygon with sides $C^{(n)}$

$$\int_C [\phi_2(\partial\phi_1/\partial n) - \phi_1(\partial\phi_2/\partial n)]\, \mathrm{d}s(x, y) = 0\,.$$

We choose $\phi_1 = \Phi(x, y; \xi, \eta)$ and $\phi_2 = \phi(x, y)$, where $\phi(x, y)$ is the desired solution of (10.57). A few basic tricks of complex analysis are then needed to connect the desired solution and the fundamental solution by the *boundary* integral equation

$$\lambda(\xi, \eta)\phi(\xi, \eta) = \int_C \left[\phi(x, y)\frac{\partial}{\partial n}\Phi(x, y; \xi, \eta) - \Phi(x, y; \xi, \eta)\frac{\partial}{\partial n}\phi(x, y)\right] \mathrm{d}s\,, \tag{10.58}$$

where $\lambda(\xi, \eta) = 1/2$. In the boundary element method, the solution of the basic problem in the interior of the domain R is obtained by solving this integral equation *on the boundary* C. The form of the integral equation remains the same in all parts of the domain of (10.57), only the parameter $\lambda(\xi, \eta)$ changes:

$$\lambda(\xi, \eta) = \begin{cases} 0 \;;\; (\xi, \eta) \notin R \cup C\,, \\ \frac{1}{2} \;;\; (\xi, \eta) \text{ on smooth part of } C\,, \\ 1 \;;\; (\xi, \eta) \in R\,. \end{cases}$$

We approximate the boundary C by a polygon with N sides, as shown in Fig. 10.8 (right), such that

$$C \approx C^{(1)} \cup C^{(2)} \cup \cdots \cup C^{(N)}\,.$$

Each side $C^{(n)}$ is a segment between the points $(x^{(n)}, y^{(n)})$ and $(x^{(n+1)}, y^{(n+1)})$. We assume that the values of the functions and their derivatives are constant along individual sides $C^{(n)}$, i.e.

$$\phi(x, y) \approx v^{(n)}\,, \qquad \frac{\partial\phi(x, y)}{\partial n} \approx d^{(n)}\,, \qquad (x, y) \in C^{(n)}\,, \qquad n = 1, 2, \ldots, N\,,$$

where $v^{(n)}$ is the value of ϕ and $d^{(n)}$ is the value of $\partial\phi/\partial n$ in the middle of $C^{(n)}$. Now the integral equation (10.58) can be approximately written as

$$\lambda(\xi, \eta)\phi(\xi, \eta) \approx \sum_{n=1}^{N} \left[v^{(n)}\mathcal{D}^{(n)}(\xi, \eta) - d^{(n)}\mathcal{V}^{(n)}(\xi, \eta)\right]\,, \tag{10.59}$$

where

$$\mathcal{V}^{(n)}(\xi, \eta) = \int_{C^{(n)}} \Phi(x, y; \xi, \eta)\, \mathrm{d}s(x, y)\,, \tag{10.60}$$

$$\mathcal{D}^{(n)}(\xi, \eta) = \int_{C^{(n)}} \frac{\partial\Phi}{\partial n}(x, y; \xi, \eta)\, \mathrm{d}s(x, y)\,. \tag{10.61}$$

For some index n, the boundary condition defines either the value $v^{(n)}$ or the derivative $d^{(n)}$ (but not both), so there are N unknowns at the right-hand side of (10.59). We choose (ξ, η) to lie in the middle of the sides $C^{(n)}$ and obtain

$$\frac{1}{2}\, v^{(m)} = \sum_{n=1}^{N} \left[v^{(n)} \mathcal{D}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) - d^{(n)} \mathcal{V}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) \right] , \quad m = 1, 2, \ldots, N ,$$

where $\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right)$ is the midpoint of $C^{(m)}$. In (10.59) we have used $\lambda = 1/2$, since all midpoints lie on the smooth parts of the approximate boundary C. This is a system of N linear equation, which can be written as

$$\sum\nolimits_{n=1}^{N} a_{mn} z_n = \sum\nolimits_{n=1}^{N} b_{mn} , \qquad m = 1, 2, \ldots, N , \tag{10.62}$$

where

$$\begin{aligned} a_{mn} &= -\mathcal{V}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) , \\ b_{mn} &= v^{(n)} \left[-\mathcal{D}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) + \tfrac{1}{2}\, \delta_{m,n} \right] , \\ z_n &= d^{(n)} , \end{aligned}$$

if the boundary condition on $C^{(n)}$ prescribes the value ϕ, or

$$\begin{aligned} a_{mn} &= \mathcal{D}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) - \tfrac{1}{2}\, \delta_{m,n} , \\ b_{mn} &= d^{(n)} \mathcal{V}^{(n)}\left(\overline{x}^{(m)}, \overline{y}^{(m)}\right) , \\ z_n &= v^{(n)} , \end{aligned}$$

if the boundary condition on $C^{(n)}$ prescribes the derivative $\partial\phi/\partial n$. When this system is solved, each component z_n contains precisely the missing information from $C^{(n)}$ that was not expressed by the boundary condition: if the derivative was specified, (10.62) provides the value of the function, and vice-versa. We end up with N values of ϕ and N values of $\partial\phi/\partial n$ on N segments of C. Finally, the solution in the interior of R is obtained by (10.59), in which we set $\lambda = 1$,

$$\phi(\xi, \eta) \approx \sum_{n=1}^{N} \left[v^{(n)} \mathcal{D}^{(n)}(\xi, \eta) - d^{(n)} \mathcal{V}^{(n)}(\xi, \eta) \right] , \qquad (\xi, \eta) \in R .$$

The elegance of the method has thus been clearly revealed: we obtain the solution on the whole domain R by manipulating information from its boundary C. In constructing $\mathcal{V}(\xi, \eta)$ and $\mathcal{D}(\xi, \eta)$, the line integrals (10.60) and (10.61) need to be computed along individual segments $C^{(n)}$. The segments are parameterized as

$$x(t) = x^{(n)} - t l^{(n)} n_y^{(n)} , \qquad y(t) = y^{(n)} + t l^{(n)} n_x^{(n)} , \qquad 0 \le t \le 1 . \tag{10.63}$$

Here $l^{(n)}$ is the length of $C^{(n)}$, while the unit vector $\boldsymbol{n}^{(n)} = (n_x^{(n)}, n_y^{(n)})^{\mathrm{T}}$ perpendicular to this segment and pointing away from R has the components

$$n_x^{(n)} = \frac{1}{l^{(n)}} \left(y^{(n+1)} - y^{(n)}\right) , \qquad n_y^{(n)} = \frac{1}{l^{(n)}} \left(x^{(n)} - x^{(n+1)}\right) .$$

We define

$$\begin{aligned} A^{(n)} &= \left(l^{(n)}\right)^2 , \\ B^{(n)}(\xi,\eta) &= 2l^{(n)} \left(-n_y^{(n)} \left(x^{(n)} - \xi\right) + n_x^{(n)} \left(y^{(n)} - \eta\right)\right) , \\ C^{(n)}(\xi,\eta) &= \left(x^{(n)} - \xi\right)^2 + \left(y^{(n)} - \eta\right)^2 . \end{aligned}$$

In the parameterization (10.63), the quantity $4A^{(n)}C^{(n)} - (B^{(n)})^2$ is non-negative,

$$F^{(n)} \equiv 4A^{(n)}C^{(n)}(\xi,\eta) - \left[B^{(n)}(\xi,\eta)\right]^2 \geq 0 , \qquad \forall (\xi,\eta) .$$

Elementary integration [9] brings us to the final expressions for the line integrals $\mathcal{V}^{(n)}$ and $\mathcal{D}^{(n)}$, which are distinguished according to the value of $F^{(n)}$. If $F^{(n)} > 0$, we evaluate

$$\begin{aligned} \mathcal{V}^{(n)} &= \frac{l^{(n)}}{4\pi} \left\{ 2\left(\log l^{(n)} - 1\right) - \frac{B^{(n)}}{2A^{(n)}} \log \left| \frac{C^{(n)}}{A^{(n)}} \right| + \left[1 + \frac{B^{(n)}}{2A^{(n)}} \right] \log \left| 1 + \frac{B^{(n)}}{A^{(n)}} + \frac{C^{(n)}}{A^{(n)}} \right| \right. \\ &\quad \left. + \frac{\sqrt{F^{(n)}}}{A^{(n)}} \left[\arctan \frac{2A^{(n)} + B^{(n)}}{\sqrt{F^{(n)}}} - \arctan \frac{B^{(n)}}{\sqrt{F^{(n)}}} \right] \right\} , \\ \mathcal{D}^{(n)} &= \frac{l^{(n)} \left[n_x^{(n)} \left(x^{(n)} - \xi\right) + n_y^{(n)} \left(y^{(n)} - \eta\right) \right]}{\pi \sqrt{F^{(n)}}} \left[\arctan \frac{2A^{(n)} + B^{(n)}}{\sqrt{F^{(n)}}} - \arctan \frac{B^{(n)}}{\sqrt{F^{(n)}}} \right] , \end{aligned}$$

On the other hand, if $F^{(n)} = 0$, we need to compute

$$\begin{aligned} \mathcal{V}^{(n)} &= \frac{l^{(n)}}{2\pi} \left\{ \log l^{(n)} + \left[1 + \frac{B^{(n)}}{2A^{(n)}} \right] \log \left| 1 + \frac{B^{(n)}}{2A^{(n)}} \right| - \frac{B^{(n)}}{2A^{(n)}} \log \left| \frac{B^{(n)}}{2A^{(n)}} \right| - 1 \right\} , \\ \mathcal{D}^{(n)} &= 0 . \end{aligned}$$

The boundary element method is also applicable to non-stationary problems: time integration is performed separately. A superb introduction is given in [9].

10.6 Finite Element Method ⋆

Finite element methods (FEM) are five decades old, yet they remain the basic tool of engineers and natural scientists, especially for problems in elastomechanics, hydro- and aero-dynamics in complex geometries. In difference methods we use finite dif-

Chapter 12
Inverse and Ill-Posed Problems ⋆

When we evaluate the expression $\boldsymbol{f} = A\boldsymbol{u}$, where $\boldsymbol{u}$ and $\boldsymbol{f}$ are vectors and A is a matrix, we solve a *direct* or *forward* problem. Given A we can precisely calculate $\boldsymbol{f}$ for any $\boldsymbol{u}$. The only danger we may anticipate in a computer is the one of over- or underflow, or perhaps loss of precision. Solving a *backward* or *inverse* problem is a step in the opposite direction: we either wish to reconstruct $\boldsymbol{u}$ from $\boldsymbol{f}$, knowing A — this is the so-called *inverse reconstruction problem* — or learn something about A from given $\boldsymbol{u}$ and $\boldsymbol{f}$, which is known as the *inverse identification problem* because we are identifying the parameters of the underlying "model" — the presumed linear mapping embodied by the matrix A. In either case trouble is brewing: A may not be invertible, and even if it were, it may be ill-conditioned. The difficulties could be compounded by the data $\boldsymbol{f}$ being noisy, or A depending on $\boldsymbol{u}$, introducing nonlinearity.

One could say that everything we ever do in physics research — identifying inputs to the system based on its outputs or assessing models — amounts to solving inverse problems in the above sense [1–4]. Most often such problems are *ill-posed*. A problem is considered to be *well-posed* if its solution exists, if this solution is unique, and if the solution depends smoothly on the data. If any of these three qualifiers is absent, the problem is said to be ill-posed.

Example Consider the linear equation $A\boldsymbol{u} = \boldsymbol{f}$ with

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 10^{-3} \end{pmatrix}, \qquad \boldsymbol{f} = \begin{pmatrix} 1 \\ 10^{-3} \end{pmatrix},$$

whose unique solution is $\boldsymbol{u} = A^{-1}\boldsymbol{f} = (1, 1)^{\mathrm{T}}$. Let us modify the second element of $\boldsymbol{f}$ by $\delta = 10^{-3}$, such that $\boldsymbol{f}^{\delta} = \boldsymbol{f} + (0, \delta)^{\mathrm{T}} = (1, 2{\cdot}10^{-3})^{\mathrm{T}}$. The solution $\boldsymbol{u}^{\delta}$ of the perturbed problem, $A\boldsymbol{u}^{\delta} = \boldsymbol{f}^{\delta}$, is $\boldsymbol{u}^{\delta} = (1, 2)^{\mathrm{T}}$, a large change in the result! Indeed, the ratio of the norm of the solution difference, $\boldsymbol{u} - \boldsymbol{u}^{\delta}$, to the norm of the perturbation, $\boldsymbol{f} - \boldsymbol{f}^{\delta}$, is

S. Širca and M. Horvat, *Computational Methods in Physics*,
Graduate Texts in Physics, https://doi.org/10.1007/978-3-319-78619-3_12

$$r = \frac{\|\boldsymbol{u} - \boldsymbol{u}^\delta\|_2}{\|\boldsymbol{f} - \boldsymbol{f}^\delta\|_2} = 1000\,.$$

Now suppose we change the matrix A by introducing a parameter α as

$$A_\alpha = \begin{pmatrix} 1 & 0 \\ 0 & (1-\alpha)^{-3}10^{-3} \end{pmatrix}\,,$$

and choose $\alpha = 0.2$. By keeping the same perturbation as before, the solution of $A_\alpha \boldsymbol{u}^\delta = \boldsymbol{f}^\delta$ becomes $\boldsymbol{u}^\delta = (1, 1.024)^{\mathrm{T}}$, with the error ratio of just $r = 24$. By guesswork we quickly find that taking $\alpha = 0.2063$ yields $u^\delta \approx (1, 0.999998)^{\mathrm{T}}$ and $r \approx 0.002$! In this obviously ill-posed problem, a particular choice of the functional form of the correction and a magical value of its parameter α dramatically reduce the reconstruction error. ◁

Example Figure 12.1 (left) shows a discrete square-wave signal $\boldsymbol{u}$ sampled at $N = 100$ points spaced $h = 1/N$ apart, its smoothly smeared version $A\boldsymbol{u}$ by using the matrix A with the elements

$$A_{ij} = \frac{h}{\sqrt{2\pi}\gamma} \exp\left[-\frac{\big((i-j)h\big)^2}{2\gamma^2}\right]\,, \quad i, j = 1, 2, \ldots, 100\,, \tag{12.1}$$

where $\gamma = 0.03$, as well as the signal $\boldsymbol{f}^\delta = A\boldsymbol{u} + \boldsymbol{\delta}$ contaminated by seemingly harmless normally distributed noise, $\delta_i \sim 0.2N(0, \sigma^2)$ with $\sigma = 0.2$.

The right panel shows a naive reconstruction of $\boldsymbol{u}$ by calculating $A^{-1}\boldsymbol{f}^\delta$. Since A is ill-conditioned, the attempt fails miserably. The problem obviously needs to be modified in some way in order to deliver a stable solution. ◁

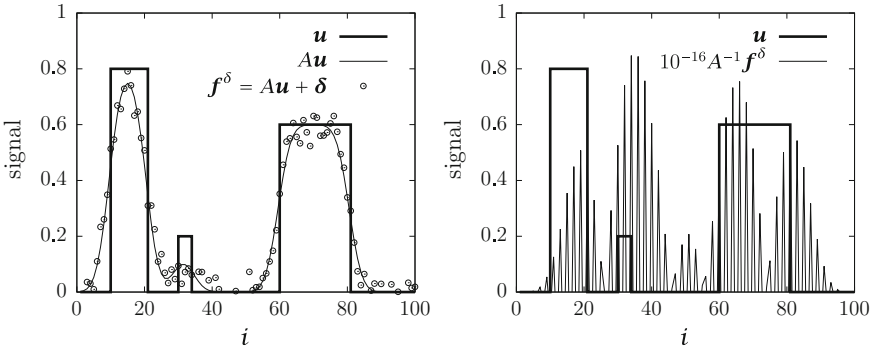


Fig. 12.1 [LEFT] The signal $\boldsymbol{u}$, its smeared version $A\boldsymbol{u}$ and the final signal with added Gaussian noise. [RIGHT] The naive reconstruction of $\boldsymbol{u}$ by solving the equation $A\boldsymbol{u} = \boldsymbol{f}^\delta$ for $\boldsymbol{u}$. The recovered solution is utterly useless: in the figure it has been multiplied by 10^{-16} (!) to fit into the scale of the plot

12.3.1 Truncated Singular Value Decomposition

The simplest way to suppress small singular values is to truncate the expansion (12.12), i.e. discard all $1/\sigma$ that fall below the threshold specified by α,

$$g_\alpha(\sigma) = \begin{cases} 1/\sigma \ ; \ \sigma \geq \alpha \ , \\ 0 \quad\ ; \ \sigma < \alpha \ . \end{cases}$$

Thus the truncated SVD (TSVD) solution is given by

$$u_\alpha = \Gamma_\alpha f = \sum_{\sigma_i \geq \alpha} \frac{1}{\sigma_i} \langle f, v_i \rangle_{\mathcal{Q}} \, u_i \ . \tag{12.14}$$

If the data error can be bounded as $\|f - f^\delta\|_{\mathcal{Q}} \leq \delta$, the TSVD results in the error estimate $\|Au_\alpha - Au_\alpha^\delta\|_{\mathcal{Q}} \leq \delta$ and, more importantly,

$$\left\|u_\alpha - u_\alpha^\delta\right\|_{\mathcal{P}} \leq \delta/\alpha \ .$$

This inequality tells us that higher noise implies that more and more singular values need to be suppressed if the solution error is to be kept below a desired value. Moreover, the choice of α is not arbitrary as it may not lead to a convergent regularization: in particular, α may be assumed to depend on δ alone, on f^δ alone, or on both δ and f^δ. The methods in which the optimal α can be determined will be discussed in Sect. 12.3.7.

12.3.2 Tikhonov Regularization

Classic Tikhonov regularization uses the function

$$g_\alpha(\sigma) = \frac{\sigma}{\sigma^2 + \alpha} \ , \tag{12.15}$$

which generates the regularized inverse

$$u_\alpha = \Gamma_\alpha f = \sum_{i=1}^{\infty} \frac{\sigma_i}{\sigma_i^2 + \alpha} \langle f, v_i \rangle_{\mathcal{Q}} \, u_i \ . \tag{12.16}$$

Assuming that α depends only on δ and that $\lim_{\delta \to 0} \delta/(2\sqrt{\alpha(\delta)}) = 0$, the Tikhonov regularization brings about the error estimate

$$\left\|u_\alpha - u_\alpha^\delta\right\|_{\mathcal{P}} \leq \delta \Big/ \left(2\sqrt{\alpha(\delta)}\right) . \tag{12.17}$$

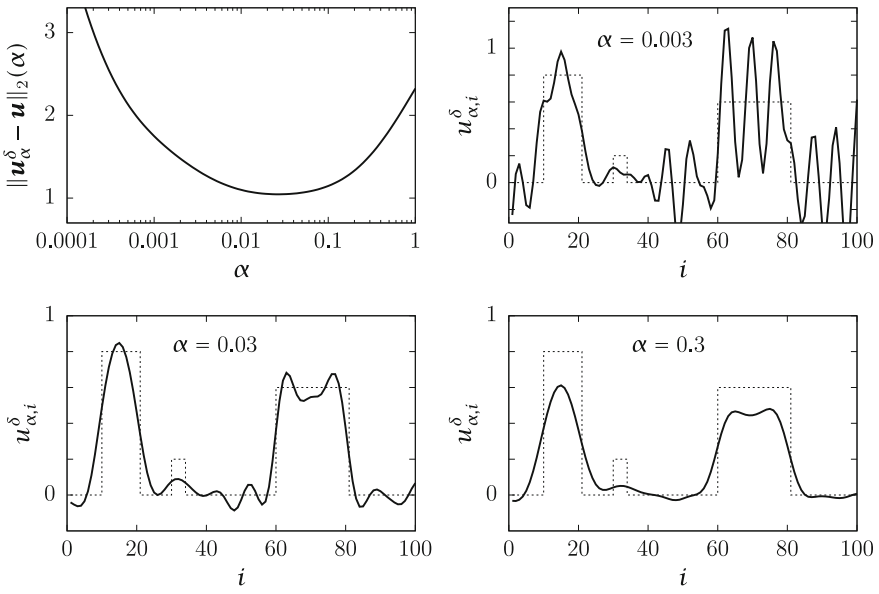


Fig. 12.4 [TOP LEFT] The norm of the reconstructed solution error and [TOP RIGHT, BOTTOM] the regularized solutions (12.18) for three values of α

Example Let us revisit the Example of Fig. 12.1 and try to regularize the ill-conditioned inverse by using the Tikhonov technique. We are dealing with an algebraic problem $A\boldsymbol{u} = \boldsymbol{f}^\delta$ with the matrix $A \in \mathbb{R}^{n\times n} = \mathcal{P} = \mathcal{Q}$ given by (12.1), $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{f}^\delta \in \mathbb{R}^n$, $n = 100$. The singular value decomposition of A,

$$A = Q\Sigma P^{\mathrm{T}} ,$$

yields the singular values σ_i (diagonal of Σ) and the singular vectors $\boldsymbol{u}_i \in \mathbb{R}^n$ (columns of P) and $\boldsymbol{v}_i \in \mathbb{R}^n$ (columns of Q), so the regularized inverse (12.16) applied to the noisy data $\boldsymbol{f}^\delta$ becomes

$$\boldsymbol{u}_\alpha^\delta = \Gamma_\alpha \boldsymbol{f}^\delta = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^2 + \alpha} \left(\boldsymbol{f}^\delta \cdot \boldsymbol{v}_i\right)\boldsymbol{u}_i \, . \tag{12.18}$$

Figure 12.4 shows the regularized solutions $\boldsymbol{u}_\alpha^\delta$ (top right and bottom panels) for three choices of α. For illustration we can assume that we know the exact solution $\boldsymbol{u}$ and take the norm of the error $\|\boldsymbol{u}_\alpha^\delta - \boldsymbol{u}\|_2$ as a measure of the quality of the regularized inverse. It is shown in Fig. 12.4 (top left). The optimal $\alpha \approx 0.03$ corresponds to the minimum of the error. We shall have another look at how to stabilize the deconvolution in Sect. 12.3.6. ◁

in discrete form; as three-fold integrals or sums are involved, the numerical cost of a naive implementation is $\mathcal{O}(N^3)$, where N is the typical size of the discretization in any variable (usually they are all comparable). See Problem 12.7.4. For information on fast versions of the Radon transformation and its inverse (typically $\mathcal{O}(N^2 \log N)$) consult [45] and references therein.

12.5 Inverse Sturm–Liouville Problems

Let us motivate this section by the paper [46] in which Mark Kac asked whether one can "*hear the shape of a drum*". In other words, can we uniquely determine the *shape* of the drum membrane if we measure its eigenfrequencies? To play a drum means to solve the wave equation $u_{tt} = \nabla^2 u$ on the domain (drum) Ω by the ansatz $u(x, y, t) = \phi(x, y)T(t)$, whence $T_{tt}/T = \nabla^2\phi/\phi = \text{const} = \lambda$. The displacement of the membrane from equilibrium is $u(x, t) = \phi(x, y) \sin \sqrt{\lambda}t$, where $\phi(x, y)$ is the solution of the two-dimensional eigenvalue problem

$$\begin{aligned} -\nabla^2\phi &= \lambda\phi \quad \text{on membrane } \Omega\,, \\ \phi &= 0 \quad \text{on membrane boundary } \partial\Omega\,, \end{aligned}$$

and where the eigenvalues λ_i are the squares of the drum's eigenfrequencies. The inverse problem consists of asking whether the geometry of the domain Ω is uniquely determined by the discrete set of eigenvalues (the spectrum). The non-trivial answer is: it is not. One can show that the observed eigenvalues do determine some properties of Ω, like its surface area, circumference, and connectivity [46], but geometrically different domains can be found with precisely the same spectrum, including possible degeneracies [47, 48]. We call such domains (and inverse problems) *isospectral*. Figure 12.12 shows the example of two membranes with identical spectra.

Similarly, one could introduce a coefficient function $c(x, y)$ in the wave equation to represent the position-dependent speed of sound, and try to infer the functional form of c from the measured spectrum. Problems of this kind are the topic of this section. To keep the presentation within the scope of this book, the

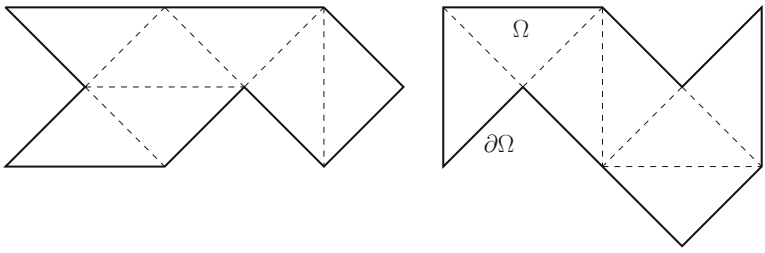


Fig. 12.12 Two geometrically different drum membranes (isospectral domains) with identical discrete sets of eigenfrequencies

discussion will be restricted to one-dimensional isospectral problems involving regular Sturm–Liouville operators on a finite interval.

12.5.1 Information Needed to Recover $q(x)$

The classical inverse Sturm–Liouville problem consists of reconstructing the potential function $q(x)$ from the solution of the direct problem

$$\left.\begin{aligned} -y''(x) + q(x)y(x) &= \lambda y(x)\,, \quad 0 < x < 1\,, \\ y'(0) - hy(0) &= 0\,, \\ y'(1) + Hy(1) &= 0\,, \end{aligned}\right\} \tag{12.53}$$

by knowing something about its eigenvalues λ_n, the properties of its eigenfunctions $y_n = y(x, \lambda_n)$, or the symmetries of q. Note that the above boundary conditions translate to $\alpha_1 = h$, $\alpha_2 = 1$ (or $h = \alpha_1/\alpha_2$) and $\beta_1 = H$, $\beta_2 = 1$ (or $H = \beta_1/\beta_2$) with $B = 1$ in the notation of (8.86) and (8.87).

Even if one knows the complete spectrum $\{\lambda_n\}_{n=0}^{\infty}$ of (12.53), this is insufficient to recover q; some additional information must be provided. For instance, if one knows in advance that q is symmetric about the midpoint of the interval, $q(x) = q(1-x)$, and that $h = H$, q can be recovered uniquely from the spectrum [49]. Similarly, if the values of q are given over at least a half of the interval, knowing the spectrum allows for a unique determination of the remaining portion of q [50]. The third option for the recovery of q is that we know the spectrum as well as the eigenfunction norms, usually defined by

$$\rho_n = \|y_n\|^2 \big/ y_n^2(0) \quad (\text{finite } h) \qquad \text{or} \qquad \rho_n = \|y_n\|^2 \big/ y_n'^2(0) \quad (h = \infty)\,,$$

where $\|\cdot\| = \|\cdot\|_{L^2(0,1)}$. The sets $\{\lambda_n\}_{n=0}^{\infty}$ and $\{\rho_n\}_{n=0}^{\infty}$ constitute the so-called *spectral data* of the Sturm–Liouville problem. Yet another possibility to uniquely reconstruct q is to specify *two* spectra of related Sturm–Liouville problems differing only in their boundary conditions, for instance, the spectrum $\{\lambda_n\}_{n=0}^{\infty}$ from (12.53) and the spectrum $\{\mu_n\}_{n=0}^{\infty}$ from the same problem but with the constant H replaced by $H' \neq H$ [51].

12.5.2 The Gelfand–Levitan Method

Gelfand and Levitan [52] have shown that the Sturm–Liouville problem for which the spectral data $\{\lambda_n\}_{n=0}^{\infty}$ and $\{\rho_n\}_{n=0}^{\infty}$ are known can be inverted analytically, recovering q as well as the boundary-value parameters h and H. Here we sketch the main steps of the method [53] for the direct problem in the form (12.53). Other types of boundary conditions are treated similarly; see, for instance, [54] or Chap. 4 of

at each step of the iteration. Recall that we are trying to recover a symmetric q, hence we must set $H = h$ also in the target boundary condition at $x = 1$. The iterative algorithm, then, is

Input: subset of spectrum $\{\lambda_n\}_{n=1}^{N}$
set $\mathcal{B}[y] = y'(1) + Hy(1)$ // $H = h$ for symmetric q
set $q^{(0)}(x)$ // can be zero
for $i = 1$ **step** 1 **to** $i_{\max}$ **do**
  for $n = 1$ **step** 1 **to** N **do**
    solve initial-value problem (12.61) for y_n
    $b_n = \mathcal{B}[y_n]$
    for $k = 1$ **step** 1 **to** B **do**
      solve initial-value problem (12.62) for $\widehat{y}_{nk}$
      $A_{nk} = \mathcal{B}[\widehat{y}_{nk}]$
    end
  end
  solve $A_{nk}c_k = -b_n$
  $\Delta q(x) = \sum_{k=1}^{B} c_k \phi_k(x)$
  $q^{(i)}(x) = q^{(i-1)}(x) + \Delta q(x)$
end
Output: reconstructed approximate $q(x)$

Note that no additional information on q is needed except the fact that it is symmetric about $x = 1/2$. In particular, we do not have to provide the average potential $\int_0^1 q(t)\,\mathrm{d}t$ that some methods require on input.

Example (Adapted from [62].) Consider a Sturm–Liouville problem with homogeneous Dirichlet boundary conditions at both $x = 0$ ($h = \infty$, hence $y_n(0) = 0$ and $y'(0) = 1$) and $x = 1$ ($H = \infty$, thus $\mathcal{B}[y] = y[1]$) with the symmetric potential

$$q(x) = 1 - \exp\bigl(-20(x - 0.5)^2\bigr)$$

shown by the thin full curve in Fig. 12.13 (left). The first three ($N = 3$) eigenvalues, computed by the SLEIGN2 code [64], are

$$\{\lambda_n\}_{n=1}^{3} = \bigl\{10.2302265,\ 40.1367683,\ 89.4264374\bigr\}\,.$$

The accuracy of this calculation is crucial! Let us take three ($B = 3$) basis functions for the expansion of the updates $\Delta q^{(i)}$, namely

$$\{\phi_k(x)\}_{k=1}^{B} = \bigl\{1,\ \cos(2k\pi x),\ \cos(4k\pi x)\bigr\}\,.$$

One is free to use other types of basis functions, for instance, linear or cubic splines, but in general they should be matched to the anticipated symmetry properties of q.

The reconstructed potential after a single iteration ($i_{\max} = 1$) and after two iterations ($i_{\max} = 2$) is shown by the dashed and thick full curve in Fig. 12.13 (left),

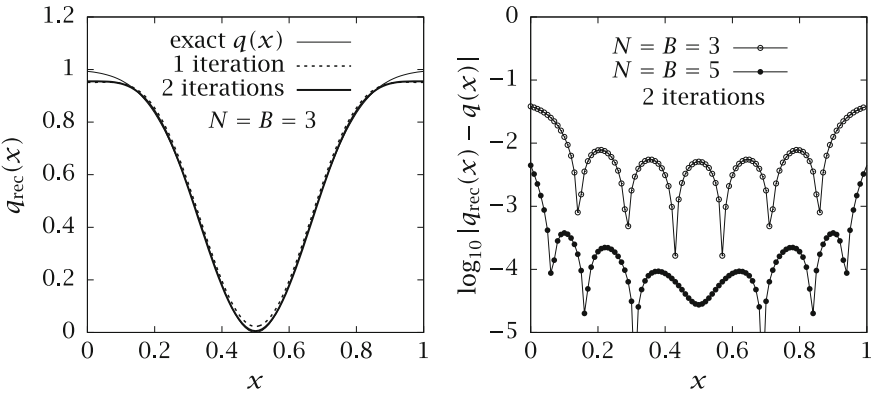


Fig. 12.13 Recovering a symmetric potential $q(x)$ of a Sturm–Liouville problem on $[0, 1]$ with homogeneous Dirichlet boundary conditions from its partial spectrum. [LEFT] Reconstruction from the lowest three eigenvalues by using an expansion in terms of three basis functions. [RIGHT] Error of the reconstruction after two iterations with $N = B = 3$ and $N = B = 5$

respectively. Figure 12.13 (right) shows the corresponding absolute errors. The algorithm converges very quickly, and increasing $i_{\max}$ does not eliminate the residual disagreement observed near the edges of the domain.

Choosing $N = B = 3$ and $N = B = 5$ in the above example makes the matrix A square, letting us exploit as much information as possible and keep A well conditioned. In practice, the size of the spectrum is usually beyond our control: we are forced to live with whatever low N we have available. If, however, we happen to have a larger set of N presumably noisy eigenvalues and opt for a relatively short expansion basis, $B < N$, the problem can still be solved in the least-squares sense: just prior to solving the system $A_{nk}c_k = -b_n$, perform the singular value decomposition $A = U\Sigma V^{\mathrm{T}}$, filter or zero out small singular values σ_i contained in Σ, and recombine to obtain the whittled-down A. This is instrumental not only in solving a non-square system, but primarily in weeding out the components causing inversion instabilities. ◁

Other types of inverse Sturm–Liouville problems For the solution of inverse Sturm–Liouville problems in the so-called impedance form

$$-\big(a(x)y'(x)\big)' = \lambda a(x)y(x)\,,$$

where the task is to recover $a(x)$ from spectral data, see [65, 66]. Inverse Sturm–Liouville problems in which the eigenvalues are contained in the boundary conditions are discussed in [67].

$$q_{1,0}\int_0^t q_2(\tau)\,\mathrm{d}\tau + \sqrt{2}\sum_{n=1}^{\infty} q_{1,n}\cos(n\pi x_0)\int_0^t q_2(\tau)\,\mathrm{e}^{-n^2\pi^2(t-\tau)}\,\mathrm{d}\tau = f(t)\,.$$

Assuming continuity of q_1 and q_2 we may change the order of summation and integration, which leads to a Volterra equation of the first kind for q_2,

$$\int_0^t K(t,\tau)q_2(\tau)\,\mathrm{d}\tau = f(t)\,, \qquad t\in[0,T]\,,$$

with the kernel

$$K(t,\tau) = q_{1,0} + \sqrt{2}\sum_{n=1}^{\infty} q_{1,n}\,\mathrm{e}^{-n^2\pi^2(t-\tau)}\cos(n\pi x_0)\,.$$

If $q_1(x_0) = 0$ the solution of the inverse problem (12.72) is not necessarily unique. This can be seen by choosing $x_0 = 1/2$ and letting q_1 be odd with respect to x_0, that is, $q_1(x) = -q_1(x_0 - x)$. The solution $q_2(t)$ *is* unique if q_1 satisfies $q_1 \in C^4[0,1]$, $q_1(x_0) \neq 0$ and $q_1'(+0) = q_1'(1-0) = 0$, and if f satisfies $f \in C^4[0,T]$ and $f(+0) = 0$ [31].

12.6.3 *Inverse Source Problems for the Wave Equation*

Consider the wave equation describing the transverse displacements along a unit-length string attached at both ends, with given initial conditions for the displacement and velocity,

$$\begin{aligned} u_{tt} &= c^2(x)u_{xx} + q_1(x)q_2(t)\,, & x\in(0,1)\,,\ t\in(0,T)\,,\\ u(x,0) = f(x)\,,\ \ u_t(x,0) &= g(x)\,, & x\in(0,1)\,,\\ u(0,t) = u(1,t) &= 0\,, & t\in(0,T)\,, \end{aligned}$$

Assuming that $c(x)$ and the temporal dependence of the source term, $q_2(t)$, are known, one might think that providing additional information on the displacement at some later time $t = T$, e.g.

$$u_T(x) = u(x,T)\,, \qquad x\in(0,1)\,, \tag{12.74}$$

would allow us to recover the spatial load $q_1(x)$, just as we have done in Sect. 12.6.2 for the heat equation. However, it can be shown that the final-state over-determination in the form (12.74) allows for a unique reconstruction of $q_1(x)$ only if T is not a rational number, which is impossible to ensure in practice. The analogous kind of over-determination in which the velocity distribution $v_T(x) = u_t(x,T)$ is known, has

the same insurmountable condition: see p. 95 of [12] for details. In the following, we shall restrict the discussion to the cases where final over-determination is not needed.

Determining the purely temporal part of the source Consider the initial-value problem

$$\left.\begin{aligned} u_{tt} = u_{xx} + \rho(x,t)q(t)\,, &\qquad x,t>0\,,\\ u(x,0) = u_t(x,0) = 0\,, &\qquad x>0\,,\\ u_x(0,t) = 0\,, &\qquad t>0\,, \end{aligned}\right\} \tag{12.75}$$

with the missing boundary condition provided by the trace of the function $u(x,t)$ measured at $x=0$,

$$u(0,t) = f(t)\,, \qquad t \geq 0\,. \tag{12.76}$$

This is the data of the inverse problem that must fulfill the consistency condition $f(0) = f'(0) = 0$. Assuming that $\rho(x,t)$ is known and that $\rho(0,t) \neq 0$ for all $t \in [0,T]$, it is then easy to show by extending the problem to negative x that $q(t)$ satisfies a Volterra equation of the second kind,

$$q(t) + \int_0^t K(t-\tau,\tau)q(\tau)\,\mathrm{d}\tau = F(t)\,, \qquad t \in [0,T]\,, \tag{12.77}$$

where

$$K(x,\cdot) = \frac{\rho_x(x,\cdot)}{\rho(0,t)}\,, \qquad F(t) = \frac{f''(t)}{\rho(0,t)}\,.$$

This integral equation is best solved by iteration

$$q^{(n)}(t) = F(t) - \int_0^t K\,(t-\tau,\tau)\,q^{(n-1)}(\tau)\,\mathrm{d}\tau\,, \qquad n = 1,2,\ldots\,,$$

with the initial guess $q^{(0)}(t) = F(t)$, or by any other method discussed in Sect. 12.4. Note that even though this integral equation has a unique solution and converges rapidly, there is no way to overcome the burden of (numerically) evaluating the second derivative of the data function f. Do not despair: refer to Appendix F.

Example Let us demonstrate the reconstruction of the purely temporal part of the source in the wave equation (12.75) by choosing

$$\rho(x,t) = 1 + x^2 t\,, \qquad q(t) = \sin t\,.$$

Our task is first to solve the direct (initial-value) problem with the known $q(t)$, sample the resulting $u(x,t)$ at $x=0$ to obtain the function $u(0,t) = f(t)$, construct $F(t)$ by taking the numerical second derivative of $f(t)$ and, finally, handle the inverse problem by solving (12.77) for q, ending up with the reconstructed $q_{\mathrm{rec}}(t)$.

The solution $u(x,t)$ of the direct problem calculated on the domain $(x,t) \in [0,T]\times[0,T]$ with $T=20$ is shown in Fig. 12.16 (top left). This calculation must

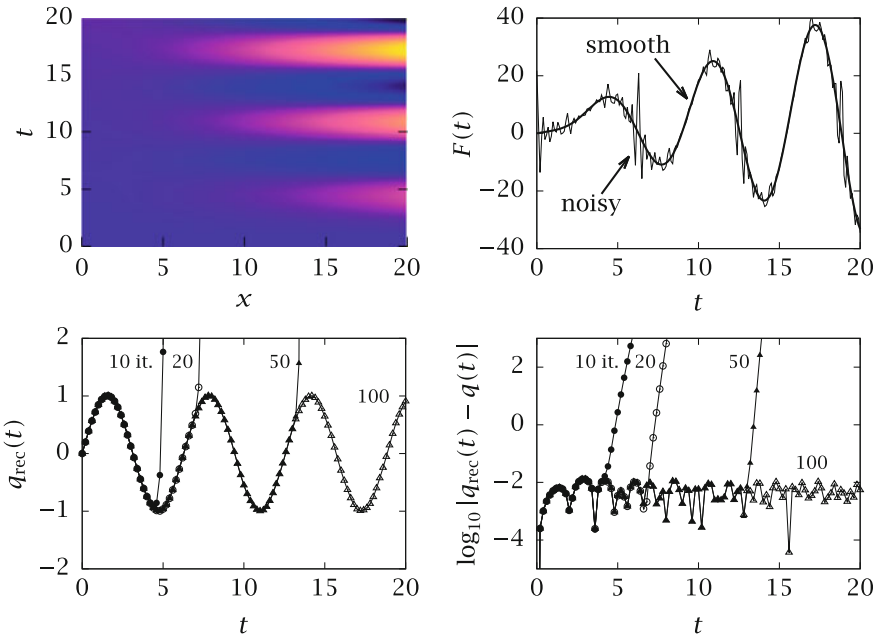


Fig. 12.16 Recovering the purely temporal part of the source term in the wave equation. [TOP LEFT] The solution $u(x,t)$ on the domain $(x,t) \in [0,20] \times [0,20]$. [TOP RIGHT] The normalized second derivative of $f(t)$. [BOTTOM LEFT] The reconstructed temporal part of the source. [BOTTOM RIGHT] The error of the reconstruction

be as precise as possible in order to achieve sufficient smoothness of $F(t)$. The solution in the figure has been calculated by the `NDSolve` routine of MATHEMATICA to a precision of six digits and interpolated, so that the second derivative of f was performed on the interpolant. The resulting $F(t)$ is shown by the smooth curve in Fig. 12.16 (top right).

The integral equation (12.77) for q is then solved by iteration with the initial guess $q^{(0)} = F(t)$. The reconstructed potential after 10, 20, 50 and 100 iterations is shown in Fig. 12.16 (bottom left). The corresponding errors with respect to the exact source are shown in the bottom right panel.

Homework: perturb $F(t)$ in some manner — obtaining something like the thin noisy trace in Fig. 12.16 (top right) — and repeat the exercise. Study the sensitivity of the reconstruction to the level of noise introduced at other stages of the procedure. ◁

Determining the purely spatial part of the source The treatment of the inverse problem in which the purely *spatial* component of the source needs to be recovered, i.e. the inverse of the initial-value problem

$$\begin{aligned} u_{tt} &= u_{xx} + \rho(x,t)q(x)\,, \qquad x,t>0\,,\\ u(x,0) &= u_t(x,0) = 0\,, \qquad x>0\,,\\ u_x(0,t) &= 0\,, \qquad t>0\,, \end{aligned}$$

If (U, V) is a random variable uniformly distributed over C_p, we have $X = V/U$ [14]. This transformation is embodied in the algorithm

Input: Distribution p, constants $a = \sup_x \sqrt{p(x)}$, $b = \inf_x x\sqrt{p(x)}$, and $c = \sup_x x\sqrt{p(x)}$.
repeat
  Independently draw u and v from $U(0, 1)$.
  $u_1 = au$;
  $u_2 = b + (c - b)v$;
  $x = v/u$;
until $(u^2 \leq p(x))$
Output: x is the value of one realization of the random variable distributed according to p.

This algorithm is a variation of the rejection method, and can be optimized similarly. An example of optimized generation of random numbers from $N(0, 1)$ is described in [15]: the region C_p where the points are accepted is shown in Fig. C.3 (right). This method requires $\approx$2.74 draws from $U(0, 1)$ to generate one number from $N(0, 1)$ [4], which is worse than in the Box-Muller method.

C.3 Random Number Generators and Tests of Their Reliability

Generators of random numbers $x_i \in \mathbb{Z}_m = [0, m - 1]$ where $i \in \mathbb{N}_0 = \{0, 1, \ldots\}$ [3, 4] are defined by the *transition function* F and the relation

$$x_i = F(x_{i-1}, \ldots, x_{i-k}) \mod m .$$

The function F is thus restricted to $\mathbb{Z}_m$ by the congruence relation [2]. The initial state $\{x_0, x_1, \ldots, x_{k-1}\}$ of the generator is a unique function of a number known as the *seed* which completely determines the generated sequence: a generator initialized with the same seed always yields the same sequence of numbers. The properties of F define two classes of generators. If F is linear in its parameters, the generator is *linear*; in other cases, it is *non-linear*.

Random number generators are implemented in all major numerical packages (MATLAB, MATHEMATICA, MAPLE, the R project) and in libraries (NUMERICAL RECIPES, GSL, BOOST). Interesting thoughts on the implementations of generators are preserved in [16] and in appendices A–C of [17]. A pedagogically systematic overview of random generator classes is offered by [2, 6].

C.3.1 Linear Generators

Typical linear generators are the *linear congruential generators* (LCG):

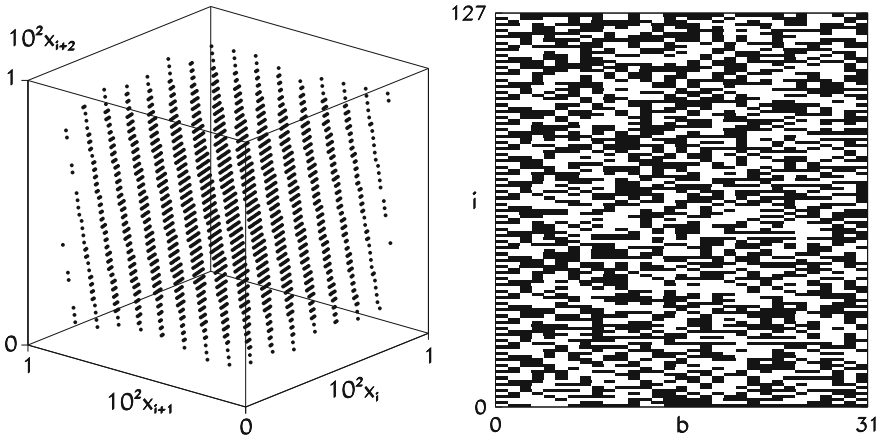


Fig. C.4 [LEFT] Zoom-in of the phase space $[0, 1]^3$ of the points $2^{-31}(x_i, x_{i+1}, x_{i+2})$ picked from the sequence $\{x_i\}$ generated by the standard 32-bit `glibc` generator with $x_0 = 12345$. [RIGHT] The bits b of numbers x_i (black = 1, white = 0)

$$x_{n+1} = ax_n + c \mod m ,$$

where $x_n \in \mathbb{Z}_m$. The result $\{x_0, x_1, \ldots\}$ is called the Lehmer sequence. The *multiplier* a and the *carry* $c \in \mathbb{Z}_m$ are adjustable constants, while the initial value x_0 is the seed. A LCG generator for $c \neq 0$ attains full periods of length m precisely when c and m have no common factors except 1, when $(a - 1)$ is divisible by all prime factors m, and when $(a - 1)$ is a multiple of 4, if m is a multiple of 4 [3]. In the case $c = 0$ we attain the longest period of length $m - 1$ if m is prime. On the average, the period of the LCG-type generator is increased by using a non-zero c.

If $c = 0$, the points $(1/m)\{x_i, \ldots, x_{i+k-1}\}$ for given k and x_0 do not fill the whole k-dimensional hypercube but lie on at most $(mk!)^{1/k}$ hyperplanes (similarly for $c \neq 0$ [18]). A good generator should generate numbers over many hyperplanes [19]. It also turns out that the least significant bits are less random than the more significant ones [4, 20]: Fig. C.4 shows the numbers generated by the default generator in the 32-bit `glibc` library. It belongs to the LCG family with the parameters $m = 2^{32}$, $a = 1103515245$, and $c = 123454$. The figure clearly shows that the points are distributed in planes and that the less significant bits are not random.

The LCG generators are therefore not suitable for certain applications. In those cases where the deficiencies discussed above are irrelevant, we nevertheless use them extensively, since they are supported by all programming languages and because they are simple and fast.

Further members of the LCG family are the generators Add-with-Carry (AWC), Subtract-with-Borrow (SWB), and Multiply-with-Carry (MWC) [21]: