Computational Techniques

8

David Schultz and Michael R. Strayer

Contents

Abstract

Essential to atomic, molecular, and optical physics is the ability to perform numerical computations accurately and efficiently. Whether the specific approach involves perturbation theory, close coupling expansion, solution of classical equations of motion, or fitting and smoothing of data, basic computational techniques such as integration, differentiation, interpolation, matrix and eigenvalue manipulation, Monte Carlo sampling, and solution of differential equations must be among the standard tool kit.

This chapter outlines a portion of this tool kit with the aim of giving guidance and organization to a wide array of computational techniques. After having digested the present overview, the reader is then referred to detailed treatments given in many of the large number of texts existing on numerical analysis and computational techniques [1–6], mathematical functions [7–9], and mathematical physics [10–18].

In addition to these excellent general references, in the age of the internet, many resources are also available through free publishing projects or research laboratory resources made public. Many of these resources seek to provide techniques and computer codes of high accuracy, portability, robustness, and efficiency, and often take advantage of modern structured programming and computational parallelism, going beyond the highly accessible, broadly applicable, but simple numerical recipes and codes described in the classic texts. A list of such numerical analysis software is given on the Wikipedia, providing very brief descriptions of the packages available [19], and the journal *Computer Physics Communications* (CPC) publishes computational physics research and applications software with many codes applicable to atomic, molecular, and optical physics (see the CPC program library maintained at Queen's University Belfast [20]). Especially in the sections that follow on differential equations and computational linear algebra, mention is made of the role of software packages readily available to aid in implementing practical solutions.

Finally, in this brief introduction to computational techniques, we note the existence of commercial packages for mathematics, including those for computer algebra, performing numerical calculations and visualizing results through proprietary programming languages, and even performing simulations through such tools as finite-element analysis, including Mathematica, Maple, MATLAB, Mathcad, and COMSOL, for example.

D. Schultz (✉)
Department of Astronomy and Planetary Science, Northern Arizona University
Flagstaff, AZ, USA
e-mail: david.schultz@nau.edu

G.W.F. Drake (Ed.), *Springer Handbook of Atomic, Molecular, and Optical Physics*, Springer Handbooks,
https://doi.org/10.1007/978-3-030-73893-8_8

Keywords

numerical methods · numerical analysis · computational methods · numerical approximation · numerical integration · random numbers · linear algebra · differential equations · interpolation · Fourier analysis

8.1 Representation of Functions

The ability to represent functions in terms of polynomials or other basic functions is the key to interpolating or fitting data, and to approximating numerically the operations of integration and differentiation. In addition, using methods such as Fourier analysis, knowledge of the properties of functions beyond even their intermediate values, derivatives, and antiderivatives may be determined (e.g., the *spectral* properties).

8.1.1 Interpolation

Given the value of a function $f(x)$ at a set of points $x_1, x_2, \dots, x_n$, the function is often required at some other values between these abscissas. The process known as *interpolation* seeks to estimate these unknown values by adjusting the parameters of a known function to approximate the local or global behavior of $f(x)$. One of the most useful representations of a function for these purposes utilizes the *algebraic polynomials*, $P_n(x) = a_0 + a_1 x + \dots + a_n x^n$, where the coefficients are real constants, and the exponents are nonnegative integers. The utility stems from the fact that given any continuous function defined on a closed interval, there exists an algebraic polynomial that is as close to that function as desired (Weierstrass theorem).

One simple application of these polynomials is the power series expansion of the function $f(x)$ about some point, x_0, i.e.,

$$f(x) = \sum_{k=0}^{\infty} a_k (x - x_0)^k \ . \tag{8.1}$$

A familiar example is the *Taylor expansion*, in which the coefficients are given by

$$a_k = \frac{f^{(k)}(x_0)}{k!} \ , \tag{8.2}$$

where $f^{(k)}$ indicates the k-th derivative of the function. This form, although quite useful in the derivation of formal techniques, is not very useful for interpolation, since it assumes the function and its derivatives are known and since it is guaranteed to be a good approximation only very near the point x_0 about which the expansion has been made.

Lagrange Interpolation

The polynomial of degree $n-1$ that passes through all n points $[x_1, f(x_1)], [x_2, f(x_2)], \dots, [x_n, f(x_n)]$ is given by

$$P(x) = \sum_{k=1}^{n} f(x_k) \prod_{i=1, i \neq k}^{n} \frac{x - x_i}{x_k - x_i} \tag{8.3}$$

$$= \sum_{k=1}^{n} f(x_k) L_{nk}(x) \ , \tag{8.4}$$

where $L_{nk}(x)$ are the *Lagrange interpolating polynomials*. Perhaps the most familiar example is that of linear interpolation between the points $[x_1, y_1 \equiv f(x_1)]$ and $[x_2, y_2 \equiv f(x_2)]$, namely,

$$P(x) = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \ . \tag{8.5}$$

In practice, it is difficult to estimate the formal error bound for this method, since it depends on knowledge of the $(n+1)$-th derivative. Alternatively, one uses *iterated interpolation*, in which successively higher-order approximations are tried until appropriate agreement is obtained. *Neville's algorithm* defines a recursive procedure to yield an arbitrary order interpolant from polynomials of lower order. This method and subtle refinements of it form the basis for most recommended polynomial interpolation schemes [3–5].

One important word of caution to bear in mind is that the more points used in constructing the interpolant, and therefore the higher the polynomial order, the greater will be the oscillation in the interpolating function. This highly oscillating polynomial most likely will not correspond more closely to the desired function than polynomials of lower order. As a general rule of thumb, fewer than six points should be used.

Cubic Splines

By dividing the interval of interest into a number of subintervals and in each using a polynomial of only modest order, one may avoid the oscillatory nature of high-order (many-point) interpolants. This approach utilizes *piecewise polynomial functions*, the simplest of which is just a linear segment. However, such a straight line approximation has a discontinuous derivative at the data points – a property that one may wish to avoid, especially if the derivative of the function is also desired – and which clearly does not provide a smooth interpolant. The solution, therefore, is to choose the polynomial of lowest order that has enough free parameters (the constants $a_0, a_1, \dots$) to satisfy the constraints that the function and its derivative are continuous across the subintervals, as well as specifying the derivative at the endpoints x_0 and x_n.

Piecewise cubic polynomials satisfy these constraints and have a continuous second derivative as well. *Cubic spline interpolation* does not, however, guarantee that the derivatives of the interpolant agree with those of the function at the data points, much less globally. The cubic polynomial in each interval has four undetermined coefficients,

$$P_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \ , \quad (8.6)$$

for $i = 0, 1, \ldots, n - 1$. Applying the constraints, a system of equations is found that may be solved once the endpoint derivatives are specified. If the second derivatives at the endpoints are set to zero, then the result is termed a *natural spline*, and its shape is like that which a long flexible rod would take if forced to pass through all the data points. A *clamped spline* results if the first derivatives are specified at the endpoints, and is usually a better approximation since it incorporates more information about the function (if one has a reasonable way to determine or approximate these first derivatives).

The set of equations in the unknowns, along with the boundary conditions, constitute a *tridiagonal system* or *matrix*, and is therefore amenable to solution by algorithms designed for speed and efficiency for such systems (Sect. 8.3; [1–5]). Other alternatives of potentially significant utility are schemes based on the use of rational functions and orthogonal polynomials.

Rational Function Interpolation

If the function that one seeks to interpolate has one or more poles for real x, then polynomial approximations are not good, and a better method is to use quotients of polynomials, so-called *rational functions*. This occurs since the inverse powers of the dependent variable will fit the region near the pole better if the order is large enough. In fact, if the function is free of poles on the real axis but its analytic continuation in the complex plane has poles, the polynomial approximation may also be poor. It is this property that slows or prevents the convergence of power series. Numerical algorithms very similar to those used to generate iterated polynomial interpolants exist [1, 3–5] and can be useful for functions that are not amenable to polynomial interpolation. Rational function interpolation is related to the method of *Padé approximation* used to improve convergence of power series. This is a rational function analog of Taylor expansion.

Orthogonal Function Interpolation

Interpolation using functions other than the algebraic polynomials can be defined and are often useful. Particularly worthy of mention are schemes based on *orthogonal polynomials* since they play a central role in numerical quadrature. A set of functions $\phi_1(x), \phi_2(x), \ldots, \phi_n(x)$ defined on the interval $[a, b]$ is said to be orthogonal with respect to a weight function $\mathcal{W}(x)$, if the inner product defined by

$$\langle \phi_i | \phi_j \rangle = \int_a^b \phi_i(x)\phi_j(x)\mathcal{W}(x)\mathrm{d}x \quad (8.7)$$

is zero for $i \neq j$ and positive for $i = j$. In this case, for any polynomial $P(x)$ of degree at most n, there exists unique constants α_k such that

$$P(x) = \sum_{k=0}^{n} \alpha_k \phi_k(x) \ . \quad (8.8)$$

Among the more commonly used orthogonal polynomials are *Legendre, Laguerre*, and *Chebyshev* polynomials.

Chebyshev Interpolation

The significant advantages of employing a representation of a function in terms of Chebyshev polynomials, $T_k(x)$, i.e.,

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x) \ , \quad (8.9)$$

stems from the fact that (i) the expansion rapidly converges, (ii) the polynomials have a simple form, and (iii) the polynomial approximates the solution of the *minimax* problem very closely. This latter property refers to the requirement that the expansion *mini*mizes the *max*imum magnitude of the error of the approximation. In particular, the Chebyshev series expansion can be truncated, so that for a given n it yields the most accurate approximation to the function. Thus, Chebyshev polynomial interpolation is essentially as good as one can hope to do. Since these polynomials are defined on the interval $[-1, 1]$, if the endpoints of the interval in question are a and b, the change of variable

$$y = \frac{x - \frac{1}{2}(b + a)}{\frac{1}{2}(b - a)} \quad (8.10)$$

will affect the proper transformation. *Press* et al. [3–5], for example, give convenient and efficient routines for computing the Chebyshev expansion of a function. See [7, 10] for tabulations, recurrence formulas, orthogonality properties, etc., of these polynomials.

8.1.2 Fitting

Fitting of data stands in distinction from interpolation in that the data may have some uncertainty. Therefore, simply determining a polynomial that passes through the points may not yield the best approximation of the underlying function. In fitting, one is concerned with minimizing the deviations of some model function from the data points in an optimal or best-fit manner. For example, given a set of data points, even a low-order interpolating polynomial might have significant oscillation. In fact, if one accounts for the statistical uncertainties in the data, the best fit may be obtained simply by considering the points to lie on a line.

In addition, most of the traditional methods of assigning this quality of best fit to a particular set of parameters of the model function rely on the assumption that the random deviations are described by a Gaussian (normal) distribution. Results of physical measurements, for example the counting of events, is often closer to a Poisson distribution, which tends (not necessarily uniformly) to a Gaussian in the limit of a large number of events, or may even contain *outliers* that lie far outside a Gaussian distribution. In these cases, fitting methods might significantly distort the parameters of the model function in trying to force these different distributions to the Gaussian form. Thus, the *least-squares* and *chi-square* fitting procedures discussed below should be used with this caveat in mind. Other techniques, often termed *robust fitting* [3–5, 21], should be used when the distribution is not Gaussian or replete with outliers.

Least Squares

In this common approach to fitting, we wish to determine the m parameters a_l of some function $f(x;a_1,a_2,\ldots,a_m)$, in this example depending on one variable, x. In particular, we seek to minimize the sum of the squares of the deviations

$$\sum_{k=1}^{n}[y(x_k) - f(x_k;a_1,a_2,\ldots,a_m)]^2 \tag{8.11}$$

by adjusting the parameters, where the $y(x_k)$ are the n data points. In the simplest case, the model function is just a straight line, $f(x;a_1,a_2) = a_1x + a_2$. Elementary multivariate calculus implies that a minimum occurs if

$$a_1\sum_{k=1}^{n}x_i^2 + a_2\sum_{k=1}^{n}x_i = \sum_{k=1}^{n}x_iy_i\ , \tag{8.12}$$

$$a_1\sum_{k=1}^{n}x_i + a_2n = \sum_{k=1}^{n}y_i\ , \tag{8.13}$$

which are called the *normal equations*. Solution of these equations is straightforward, and an error estimate of the fit can be found [3–5]. In particular, variances may be computed for each parameter, as well as measures of the correlation between uncertainties and an overall estimate of the *goodness of fit* of the data.

Chi-Square Fitting

If the data points each have associated with them a different standard deviation, σ_k, the least-squares principle is modified by minimizing the *chi-square*, defined as

$$\chi^2 \equiv \sum_{k=1}^{n}\left[\frac{y_k - f(x_k;a_1,a_2,\ldots,a_m)}{\sigma_k}\right]^2 . \tag{8.14}$$

Assuming that the uncertainties in the data points are normally distributed, the chi-square value gives a measure of the goodness of fit. If there are n data points and m adjustable parameters, then the probability that χ^2 should exceed a particular value purely by chance is

$$Q = Q\left(\frac{n-m}{2},\frac{\chi^2}{2}\right)\ , \tag{8.15}$$

where $Q(a,x) = \Gamma(a,x)/\Gamma(a)$ is the incomplete gamma function. For small values of Q, the deviations of the fit from the data are unlikely to be by chance, and values close to one are indications of better fits. In terms of the chi-square, reasonable fits often have $\chi^2 \approx n - m$.

Other important applications of the chi-square method include simulation and estimating standard deviations. For example, if one has some idea of the actual (i.e., non-Gaussian) distribution of uncertainties of the data points, Monte Carlo simulation can be used to generate a set of test data points subject to this presumed distribution, and then the fitting procedure may be performed on the simulated data set. This allows one to test the accuracy or applicability of the model function chosen. In other situations, if the uncertainties of the data points are unknown, one can assume that they are all equal to some value, say σ, fit using the chi-square procedure, and solve for the value of σ. Thus, some measure of the uncertainty from this statistical point of view can be provided.

General Least Squares

The least-squares procedure can be generalized, usually by allowing any linear combination of basis functions to determine the model function

$$f(x;a_1,a_2,\ldots,a_m) = \sum_{l=1}^{m}a_l\psi_l(x)\ . \tag{8.16}$$

The basis functions need not be polynomials. Similarly, the formula for chi-square can be generalized and normal equations determined through minimization. The equations may be written in compact form by defining a matrix $\boldsymbol{A}$ with elements

$$A_{i,j} = \frac{\psi_j(x_i)}{\sigma_i}\ , \tag{8.17}$$

and a column vector $\boldsymbol{B}$ with elements

$$B_i = \frac{y_i}{\sigma_i}\ . \tag{8.18}$$

Then the normal equations are [3–5]

$$\sum_{j=1}^{m}\alpha_{kj}a_j = \beta_k\ , \tag{8.19}$$

where

$$[\alpha] = \boldsymbol{A}^T\boldsymbol{A}\ ,\quad [\beta] = \boldsymbol{A}^T\boldsymbol{B}\ , \tag{8.20}$$

and a_j are the adjustable parameters. These equations may be solved using standard methods of computational linear algebra such as Gauss–Jordan elimination. Difficulties involving sensitivity to round-off errors can be avoided by using carefully developed codes to perform this solution [3–5]. We note that elements of the inverse of the matrix $\boldsymbol{\alpha}$ are related to the variances associated with the free parameters and to the covariances relating them.

Statistical Analysis of Data

Data generated by an experiment, or from a Monte Carlo simulation, have uncertainties due to the statistical, or random, character of the processes by which they are acquired. Therefore, one must be able to describe certain features of the data statistically, such as their mean, variance and skewness, and the degree to which correlations exist, either between one portion of the data and another, or between the data and some other standard or model distribution. A very readable introduction to this type of analysis was given by *Young* [22], while more comprehensive treatments are also available [23].

8.1.3 Fourier Analysis

The *Fourier transform* takes, for example, a function of time into a function of frequency, or vice versa, namely

$$\tilde{\varphi}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \varphi(t)\,\mathrm{e}^{\mathrm{i}\omega t}\,\mathrm{d}t \,, \tag{8.21}$$

$$\varphi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{\varphi}(\omega)\,\mathrm{e}^{-\mathrm{i}\omega t}\,\mathrm{d}\omega \,. \tag{8.22}$$

In this case, the time history of the function $\varphi(t)$ may be termed the *signal* and $\tilde{\varphi}(\omega)$ the *frequency spectrum*. Also, if the frequency is related to the energy by $E = \hbar\omega$, one obtains an *energy spectrum* from a signal, and thus the name *spectral methods* for techniques based on the Fourier analysis of signals.

The Fourier transform also defines the relationship between the spatial and momentum representations of wave functions, i.e.,

$$\psi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{\psi}(p)\,\mathrm{e}^{\mathrm{i}px}\,\mathrm{d}p \,, \tag{8.23}$$

$$\tilde{\psi}(p) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \psi(x)\,\mathrm{e}^{-\mathrm{i}px}\,\mathrm{d}x \,. \tag{8.24}$$

Along with the closely related *sine*, *cosine*, and *Laplace transforms*, the Fourier transform is an extraordinarily powerful tool in the representation of functions, spectral analysis, convolution of functions, filtering, and analysis of correlation. Good introductions to these techniques with particular attention to applications in physics can be found in [10, 16, 24]. To implement the Fourier transform numerically, the integral transform pair can be converted to sums

$$\tilde{\varphi}(\omega_j) = \frac{1}{\sqrt{2N\,2\pi}} \sum_{k=0}^{2N-1} \varphi(t_k)\,\mathrm{e}^{\mathrm{i}\omega_j t_k} \,, \tag{8.25}$$

$$\varphi(t_k) = \frac{1}{\sqrt{2N\,2\pi}} \sum_{j=0}^{2N-1} \tilde{\varphi}(\omega_j)\,\mathrm{e}^{-\mathrm{i}\omega_j t_k} \,, \tag{8.26}$$

where the functions are *sampled* at $2N$ points. These equations define the *discrete Fourier transform* (DFT). Two cautions in using the DFT are as follows.

First, if a continuous function of time is sampled at, for simplicity, uniformly spaced intervals, (i.e., $t_{i+1} = t_i + \Delta$), then there is a critical frequency $\omega_\mathrm{c} = \pi/\Delta$, known as the *Nyquist frequency*, which limits the fidelity of the DFT of this function in that it is *aliased*. That is, components outside the frequency range $-\omega_\mathrm{c}$ to ω_c are falsely transformed into this range due to the finite sampling. This effect can be remediated by filtering or windowing techniques. If, however, the function is *bandwidth limited* to frequencies smaller than ω_c, then the DFT does not suffer from this effect, and the signal is completely determined by its samples.

Second, implementing the DFT directly from the equations above would require approximately N^2 multiplications to perform the Fourier transform of a function sampled at N points. A variety of *fast Fourier transform* (FFT) algorithms have been developed (e.g., the Danielson–Lanczos and Cooley–Tukey methods) that require only on the order of $(N/2)\log_2 N$ multiplications. Thus, for even moderately large sets of points, the FFT methods are, indeed, much faster than the direct implementation of the DFT. Issues involved in sampling, aliasing, and selection of algorithms for the FFT are discussed in great detail, for example, in [3–5, 15, 25]. In addition to basic computer codes with which to implement the FFT and related tasks, given, for example, in *Numerical Recipes* [3–5], codes for real and complex valued FFTs that have been implemented and benchmarked on a variety of platforms including parallel computer systems are available, for example, the Fastest Fourier Transform in the West (FFTW) [26].

8.1.4 Approximating Integrals

Polynomial Quadrature

Definite integrals may be approximated through a procedure known as numerical quadrature by replacing the integral by an appropriate sum, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{k=0}^{n} a_k f(x_k) . \qquad (8.27)$$

Most formulas for such approximations are based on the interpolating polynomials described in Sect. 8.1.1, especially the Lagrange polynomials, in which case the coefficients a_k are given by

$$a_k = \int_a^b L_{nk}(x_k)\mathrm{d}x . \qquad (8.28)$$

If first or second degree Lagrange polynomials are used with a uniform spacing between the data points, one obtains the *trapezoidal* and *Simpson's rules*, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \frac{\delta}{2}[f(a)+f(b)] + \mathcal{O}\left[\delta^3 f^{(2)}(\zeta)\right] , \qquad (8.29)$$

$$\int_a^b f(x)\mathrm{d}x \approx \frac{\delta}{3}\left[f(a)+4f\left(\frac{\delta}{2}\right)+f(b)\right] + \mathcal{O}\left[\delta^5 f^{(4)}(\zeta)\right] , \qquad (8.30)$$

respectively, with $\delta = b - a$, and for some ζ in $[a, b]$.

Other commonly used formulas based on low-order polynomials, generally referred to as *Newton–Cotes* formulas, are described and discussed in detail in numerical analysis texts [1, 2]. Since potentially unwanted rapid oscillations in interpolants may arise, it is generally the case that increasing the order of the quadrature scheme too greatly does not generally improve the accuracy of the approximation. Dividing the interval $[a, b]$ into a number of subintervals and summing the result of application of a low-order formula in each subinterval is usually a much better approach. This procedure, referred to as *composite quadrature*, may be combined with choosing the data points at a nonuniform spacing, decreasing the spacing where the function varies rapidly, and increasing the spacing for economy where the function is smooth to construct an *adaptive quadrature*.

Gaussian Quadrature

If the function whose definite integral is to be approximated can be evaluated explicitly, then the data points (abscissas) can be chosen in a manner in which significantly greater accuracy may be obtained than using Newton–Cotes formulas of equal order. *Gaussian quadrature* is a procedure in which the error in the approximation is minimized owing to this freedom to choose both data points (abscissas) and coefficients. By utilizing orthogonal polynomials and choosing the abscissas at the roots of the polynomials in the interval under consideration, it can be shown that the coefficients may be optimally chosen by solving a simple set of linear equations.

Thus, a Gaussian quadrature scheme approximates the definite integral of a function multiplied by the weight function appropriate to the orthogonal polynomial being used as

$$\int_a^b \mathcal{W}(x) f(x)\mathrm{d}x \approx \sum_{k=1}^{n} a_k f(x_k) , \qquad (8.31)$$

where the function is to be evaluated at the abscissas given by the roots of the orthogonal polynomial, x_k. In this case, the coefficients a_k are often referred to as *weights* but should not be confused with the weight function $\mathcal{W}(x)$ (Sect. 8.1.1). Since the Legendre polynomials are orthogonal over the interval $[-1, 1]$ with respect to the weight function $\mathcal{W}(x) \equiv 1$, this equation has a particularly simple form, leading immediately to the *Gauss–Legendre quadrature*. If $f(x)$ contains the weight function of another of the orthogonal polynomials as a factor, the corresponding *Gauss–Laguerre* or *Gauss–Chebyshev* quadrature should be used.

The roots and coefficients have been tabulated [7] for many common choices of the orthogonal polynomials (e.g., Legendre, Laguerre, Chebyshev) and for various orders. Simple computer subroutines are also available that conveniently compute them [3–5]. Since the various orthogonal polynomials are defined over different intervals, use of the change of variables such as that given in Eq. (8.10) may be required. So, for Gauss–Legendre quadrature we make use of the transformation

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{(b-a)}{2}\int_{-1}^{1} f\left(\frac{(b-a)y+b+a}{2}\right)\mathrm{d}y . \qquad (8.32)$$

Other Methods

Especially for multidimensional integrals that cannot be reduced analytically to separable or iterated integrals of lower dimension, *Monte Carlo* integration may provide the only means of finding a good approximation. This method is described in Sect. 8.4.3. Also, a convenient quadrature scheme can be devised based on the cubic spline interpolation described in Sect. 8.1.1, since in each subinterval, the definite integral of a cubic polynomial of known coefficients is evident.

8.1.5 Approximating Derivatives

Numerical Differentiation

The calculation of derivatives from a numerical representation of a function is generally less stable than the calculation of integrals, because differentiation tends to enhance fluctuations and worsen the convergence properties of power series.

For example, if $f(x)$ is twice continuously differentiable on $[a, b]$, then differentiation of the linear Lagrange interpolation formula Eq. (8.5) yields

$$f^{(1)}(x_0) = \frac{f(x_0+\delta) - f(x_0)}{\delta} + \mathcal{O}\big[\delta f^{(2)}(\zeta)\big] \,, \qquad (8.33)$$

for some x_0 and ζ in $[a, b]$, where $\delta = b - a$. In the limit $\delta \to 0$, Eq. (8.33) coincides with the definition of the derivative. However, in practical calculations with finite precision arithmetic, δ cannot be taken too small because of numerical cancellation in the calculation of $f(a+\delta) - f(a)$.

In practice, increasing the order of the polynomial used decreases the truncation error, but at the expense of increasing round-off error, the upshot being that three and five-point approximations are usually the most useful. Various three and five-point formulas are given in standard texts [2, 7, 9]. Two common five-point formulas (centered and forward/backward) are

$$\begin{aligned} f^{(1)}(x_0) = \frac{1}{12\delta}\big[& f(x_0-2\delta) - 8f(x_0-\delta) \\ &+ 8f(x_0+\delta) - f(x_0+2\delta)\big] \\ &+ \mathcal{O}\big[\delta^4 f^{(5)}(\zeta)\big] \qquad (8.34) \\ f^{(1)}(x_0) = \frac{1}{12\delta}\big[& -25f(x_0) + 48f(x_0+\delta) \\ &- 36f(x_0+2\delta) + 16f(x_0+3\delta) \\ &- 3f(x_0+4\delta)\big] + \mathcal{O}\big[\delta^4 f^{(5)}(\zeta)\big] \,. \qquad (8.35) \end{aligned}$$

The second formula is useful for evaluating the derivative at the left or right endpoint of the interval, depending on whether δ is positive or negative, respectively.

Derivatives of Interpolated Functions

An interpolating function can be directly differentiated to obtain the derivative at any desired point. For example, if $f(x) \approx a_0 + a_1 x + a_2 x^2$, then $f^{(1)}(x) = a_1 + 2a_2 x$. However, this approach may fail to give the best approximation to $f^{(1)}(x)$ if the original interpolation was optimized to give the best possible representation of $f(x)$.

8.2 Differential and Integral Equations

The subject of differential and integral equations is immense in both richness and scope. The discussion here focuses on techniques and algorithms, rather than the formal aspect of the theory. Further information can be found elsewhere under the broad categories of finite-element and finite-difference methods. The *Numerov method*, which is particularly useful in integrating the Schrödinger equation, is described in great detail in [18].

8.2.1 Ordinary Differential Equations

An *ordinary differential equation* is an equation involving an unknown function and one or more of its derivatives that depend on only one independent variable [27]. The *order* of a differential equation is the order of the highest derivative appearing in the equation. A solution of a general differential equation of order n,

$$f\big(t, y, \dot{y}, \ldots, y^{(n)}\big) = 0 \,, \qquad (8.36)$$

is a real-valued function $y(t)$ having the following properties: (1) $y(t)$ and its first n derivatives exist, so $y(t)$ and its first $n-1$ derivatives must be continuous, and (2) $y(t)$ satisfies the differential equation for all t. A unique solution requires the specification of n conditions on $y(t)$ and its derivatives. The conditions may be specified as n initial conditions at a single t to give an *initial-value problem*, or at the end points of an interval to give a *boundary value problem*.

First consider solutions to the simple equation

$$\dot{y} = f(t, y) \,, \qquad y(a) = A \,. \qquad (8.37)$$

The methods discussed below can be extended to systems of first-order differential equations and to higher-order differential equations. The methods are referred to as *discrete-variable methods* and generate a sequence of approximate values for $y(t)$, $y_1, y_2, y_3, \ldots$ at points $t_1, t_2, t_3, \ldots$. For simplicity, the discussion here assumes a constant spacing h between t points. We shall first describe a class of methods known as *one-step* methods [28]. They have no memory of the solutions at past times; given y_i, there is a recipe for y_{i+1} that depends only on information at t_i. Errors enter into numerical solutions from two sources. The first is the *discretization error* and depends on the method being used. The second is the *computational error* that includes such things as round-off error.

For a solution on the interval $[a, b]$, let the t points be equally spaced; so for some positive integer n and $h = (b-a)/n$, $t_i = a + ih$, $i = 0, 1, \ldots, n$. If $a < b$, h is positive, and the integration is forward; if $a > b$, h is negative, and the integration is backward. The latter case could occur in solving for the initial point of a solution curve given the terminal point. A general one-step method can then be written in the form

$$y_{i+1} = y_i + h\Delta(t_i, y_i) \,, \qquad y_0 = y(t_0) \,, \qquad (8.38)$$

where Δ is a function that characterizes the method. Different Δ functions are displayed next, giving rise to the Taylor-series methods and the Runge–Kutta methods.

Taylor-Series Algorithm

To obtain an approximate solution of order p on $[a, b]$, generate the sequence

$$\begin{aligned} y_{i+1} &= y_i + h\left[f(t_i, y_i) + \cdots + f^{(p-1)}(t_i, y_i)\frac{h^{p-1}}{p!}\right] ,\\ t_{i+1} &= t_i + h, \quad i = 0, 1, \ldots, n-1 , \end{aligned} \tag{8.39}$$

where $t_0 = a$, and $y_0 = A$. The Taylor method of order $p = 1$ is known as *Euler's method*:

$$\begin{aligned} y_{i+1} &= y_i + hf(t_i, y_i) ,\\ t_{i+1} &= t_i + h . \end{aligned} \tag{8.40}$$

Taylor-series methods can be quite effective if the total derivatives of f are not too difficult to evaluate. Software packages are available that perform exact differentiation (ADIFOR, Maple, Mathematica, etc.), facilitating the use of this approach.

Runge–Kutta Methods

Runge–Kutta methods are designed to approximate Taylor-series methods [29] but have the advantage of not requiring explicit evaluations of the derivatives of $f(t, y)$. The basic idea is to use a linear combination of values of $f(t, y)$ to approximate $y(t)$. This linear combination is matched up as closely as possible with a Taylor series for $y(t)$ to obtain methods of the highest possible order p. Euler's method is an example using one function evaluation.

To obtain an approximate solution of order $p = 2$, let $h = (b - a)/n$ and generate the sequences

$$\begin{aligned} y_{i+1} &= y_i + h\Bigg[(1-\gamma)f(t_i, y_i)\\ &\qquad + \gamma f\left[t_i + \frac{h}{2\gamma}, y_i + \frac{h}{2\gamma}f(t_i, y_i)\right]\Bigg] ,\\ t_{i+1} &= t_i + h , \qquad i = 0, 1, \ldots, n-1 , \end{aligned} \tag{8.41}$$

where $\gamma \neq 0$, $t_0 = a$, $y_0 = A$.

Euler's method is the special case, $\gamma = 0$, and has order 1; the improved Euler method has $\gamma = 1/2$, and the Euler–Cauchy method has $\gamma = 1$.

The Adams–Bashforth and Adams–Moulton Formulas

These formulas furnish important and widely used examples of multistep methods [30]. On reaching a mesh point t_i with approximate solution $y_i \cong y(t_i)$, (usually) approximate solutions $y_{i+1-j} \cong y(t_{i+1-j})$ for $j = 2, 3, \ldots, p$ are available. From the differential equation itself, approximations to the derivatives $\dot{y}(t_{i+1-j})$ can be obtained.

An attractive feature of the approach is the form of the underlying polynomial approximation, $P(t)$, to $\dot{y}(t)$ because it can be used to approximate $y(t)$ between mesh points

$$y(t) \cong y_i + \int_{t_i}^{t} P(t)\,\mathrm{d}t . \tag{8.42}$$

The lowest-order Adams–Bashforth formula arises from interpolating the single value $f_i = f(t_i, y_i)$ by $P(t)$. The interpolating polynomial is constant, so its integration from t_i to t_{i+1} results in $hf(t_i, y_i)$, and the first-order Adams–Bashforth formula:

$$y_{i+1} = y_i + hf(t_i, y_i) . \tag{8.43}$$

This is just the forward Euler formula. For constant step size h, the second-order Adams–Bashforth formula is

$$y_{i+1} = y_i + h\left[\left(\frac{3}{2}\right)f(t_i, y_i) - \left(\frac{1}{2}\right)f(t_{i-1}, y_{i-1})\right] . \tag{8.44}$$

The lowest-order Adams–Moulton formula involves interpolating the single value $f_{i+1} = f(x_{i+1}, y_{i+1})$ and leads to the backward Euler formula

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}) , \tag{8.45}$$

which defines y_{i+1} implicitly. From its definition it is clear that it has the same accuracy as the forward Euler method; its advantage is vastly superior stability. The second-order Adams–Moulton method also does not use previously computed solution values; it is called the trapezoidal rule, because it generalizes the trapezoidal rule for integrals to differential equations:

$$y_{i+1} = y_i + \frac{h}{2}[f(t_{i+1}, y_{i+1}) + f(t_i, y_i)] . \tag{8.46}$$

The Adams–Moulton formula of order p is more accurate than the Adams–Bashforth formula of the same order. Hence, it can use a larger step size; the Adams–Moulton formula is also more stable. A code based on such methods is more complex than a Runge–Kutta code, because it must cope with the difficulties of starting the integration and changing the step size. Modern Adams codes attempt to select the most efficient formula at each step, as well as to choose an optimal step size h to achieve a specified accuracy.

8.2.2 Differencing Algorithms for Partial Differential Equations

Differencing schemes, based on flux conservation methods [31], are the modern approach to solving partial differential equations describing the evolution of physical systems.

One begins by writing the balance equations for a single cell and subsequently applying quadratures and interpolation formulas. Such approaches have been successful for the full spectrum of hyperbolic, elliptic, and parabolic equations. For simplicity, we begin by discussing systems involving only one space variable.

As a prototype, consider the parabolic equation

$$c\frac{\partial}{\partial t}u(x,t) = \sigma\frac{\partial^2}{\partial x^2}u(x,t) , \quad (8.47)$$

where c and σ are constants, and $u(x,t)$ is the solution. We begin by establishing a grid of points on the xt-plane with step size h in the x direction and step size k in the t-direction. Let spatial grid points be denoted by $x_n = x_0 + nh$ and time grid points by $t_j = t_0 + jk$, where n and j are integers, and (x_0, t_0) is the origin of the space–time grid. The points ξ_{n-1} and ξ_n are introduced to establish a *control interval*. We begin with a conservation statement

$$\begin{aligned}&\int_{\xi_{n-1}}^{\xi_n} \mathrm{d}x \left[r(x,t_{j+1}) - r(x,t_j)\right]\\ &= \int_{t_j}^{t_{j+1}} \mathrm{d}t \left[q(\xi_{n-1},t) - q(\xi_n,t)\right] . \quad (8.48)\end{aligned}$$

This equation states that the change in the field density on the interval (ξ_{n-1}, ξ_n) from time $t = t_j$ to time $t = t_{j+1}$ is given by the flux into this interval at ξ_{n-1} minus the flux out of the interval at ξ_n from time t_j to time t_{j+1}. This expresses the conservation of material in the case that no sources or sinks are present. We relate the field variable u to the physical variables (the density r and the flux q). We consider the case in which the density is assumed to have the form

$$r(x,t) = cu(x,t) + b , \quad (8.49)$$

with c and b constants; thus

$$\begin{aligned}&c\int_{\xi_{n-1}}^{\xi_n} \mathrm{d}x \left[u(x,t_{j+1}) - u(x,t_j)\right]\\ &\approx c[u(x_n,t_{j+1}) - u(x_n,t_j)]h . \quad (8.50)\end{aligned}$$

When developing conservation-law equations, there are two commonly used strategies for approximating the right-hand-side of Eq. (8.48): (i) left-end-point quadrature

$$\begin{aligned}&\int_{t_j}^{t_{j+1}} \mathrm{d}t \left[q(\xi_{n-1},t) - q(\xi_n,t)\right]\\ &\approx \left[q(\xi_{n-1},t_j) - q(\xi_n,t_j)\right]k , \quad (8.51)\end{aligned}$$

and (ii) right-end-point quadrature

$$\begin{aligned}&\int_{t_j}^{t_{j+1}} \mathrm{d}t \left[q(\xi_{n-1},t) - q(\xi_n,t)\right]\\ &\approx \left[q(\xi_{n-1},t_{j+1}) - q(\xi_n,t_{j+1})\right]k . \quad (8.52)\end{aligned}$$

Combining Eq. (8.48) with the respective approximations yields: from (i) an *explicit* method

$$\begin{aligned}&c\left[u(x_n,t_{j+1}) - u(x_n,t_j)\right]h\\ &\approx \left[q(\xi_{n-1},t_j) - q(\xi_n,t_j)\right]k , \quad (8.53)\end{aligned}$$

and from (ii) an *implicit* method

$$\begin{aligned}&c\left[u(x_n,t_{j+1}) - u(x_n,t_j)\right]h\\ &\approx \left[q(\xi_{n-1},t_{j+1}) - q(\xi_n,t_{j+1})\right]k . \quad (8.54)\end{aligned}$$

Using centered-finite-difference formulas to approximate the fluxes at the control points ξ_{n-1} and ξ_n yields

$$q(\xi_{n-1},t_j) = -\sigma\frac{u(x_n,t_j) - u(x_{n-1},t_j)}{h} , \quad (8.55)$$

and

$$q(\xi_n,t_j) = -\sigma\frac{u(x_{n+1},t_j) - u(x_n,t_j)}{h} , \quad (8.56)$$

where σ is a constant. We also obtain similar formulas for the fluxes at time t_{j+1}.

We have used a lower case u to denote the continuous field variable, $u = u(x,t)$. Note that all of the quadrature and difference formulas involving u are stated as approximate equalities. In each of these approximate equality statements, the amount by which the right-hand side differs from the left-hand side is called the truncation error. If u is a well-behaved function (has enough smooth derivatives), then it can be shown that these truncation errors approach zero as the grid spacings, h and k, approach zero.

If U_n^j denotes the exact solution on the grid, from (i) we have the result

$$c\left(U_n^{j+1} - U_n^j\right)h^2 = \sigma k\left(U_{n-1}^j + U_{n+1}^j - 2U_n^j\right) . \quad (8.57)$$

This is an *explicit* method, since it provides the solution to the difference equation at time t_{j+1}, knowing the values at time t_j.

If we use the numerical approximations (ii), we obtain the result

$$c\left(U_n^{j+1} - U_n^j\right)h^2 = \sigma k\left(U_{n-1}^{j+1} + U_{n+1}^{j+1} - 2U_n^{j+1}\right) . \quad (8.58)$$

Note that this equation defines the solution at time t_{j+1} *implicitly*, since a system of algebraic equations is required to be satisfied.

8.2.3 Variational Methods

A common problem in atomic, molecular, and optical physics is to find the extrema or the stationary values of functionals. For example, one might seek the eigenvalues and eigenvectors of a Hamiltonian system, such as via the minimization of the expectation value of the energy of a trial wave function to determine the ground state of an atom or molecule, via so-called *variational methods*. We shall outline in detail the Rayleigh–Ritz method [32]. This method is limited to boundary value problems that can be formulated in terms of the minimization of a functional $J[u]$. For definiteness we consider the case of a differential operator defined by

$$Lu(\boldsymbol{x}) = f(\boldsymbol{x}) \,, \tag{8.59}$$

with $\boldsymbol{x} = x_i, i = 1, 2, 3$ in R, for example, and with $u = 0$ on the boundary of R. The function $f(x)$ is the source. It is assumed that L is always nonsingular, and in addition, for the Ritz method L is Hermitian. The real-valued functions u are in the Hilbert space Ω of the operator L. We construct the functional $J[u]$ defined as

$$J[u] = \int\limits_{\Omega} \mathrm{d}x\, [u(x)Lu(x) - 2u(x)f(x)] \,. \tag{8.60}$$

The variational ansatz considers a subspace of Ω, Ω_n, spanned by a class of functions $\phi_n(x)$, and we construct the function $u^n \approx u$ as

$$u^n(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,. \tag{8.61}$$

We solve for the coefficients c_i by minimizing $J[u^n]$:

$$\partial_{c_i} J[u^n] = 0 \,, \quad i = 1, \ldots, n \,. \tag{8.62}$$

These equations are cast into a set of well-behaved algebraic equations

$$\sum_{j=1}^{n} A_{i,j} c_j = g_i \,, \quad i = 1, \ldots, n \,, \tag{8.63}$$

with $A_{i,j} = \int_\Omega \mathrm{d}x \phi_i(x) L \phi_j(x)$, and $g_i = \int_\Omega \mathrm{d}x \phi_i(x) f(x)$.

Under very general conditions, the functions u^n converge uniformly to u. The main drawback of the Ritz method lies in the assumption of hermiticity of the operator L. For the *Galerkin method* we relax this assumption with no other changes. Thus, we obtain an identical set of equations as above, with the exception that the function g is no longer symmetric. The convergence of the sequence of solutions u^n to u is no longer guaranteed, unless the operator can be separated into a symmetric part L_0, $L = L_0 + K$, so that $L_0^{-1} K$ is bounded.

8.2.4 Finite Elements

As discussed in Sect. 8.2.2, in the finite-difference method for classical partial differential equations, the solution domain is approximated by a grid of uniformly spaced nodes. At each node, the governing differential equation is approximated by an algebraic expression that references adjacent grid points. A system of equations is obtained by evaluating the previous algebraic approximations for each node in the domain. Finally, the system is solved for each value of the dependent variable at each node. The *finite-element method* evolved from computational approaches to implementation of the variational method and of potentially greater accuracy and flexibility.

In the finite-element method [33], the solution domain can be discretized into a number of uniform or nonuniform finite elements that are connected via nodes. The change of the dependent variable with regard to location is approximated within each element by an interpolation function. The interpolation function is defined relative to the values of the variable at the nodes associated with each element. The original boundary value problem is then replaced with an equivalent integral formulation. The interpolation functions are substituted into the integral equation, integrated, and combined with the results from all other elements in the solution domain.

The results of this procedure can be reformulated into a matrix equation of the form

$$\sum_{j=1}^{n} A_{i,j} c_j = g_i \,, \quad i = 1, \ldots, n \,, \tag{8.64}$$

with $A_{i,j} = \int_\Omega \mathrm{d}x \phi_i(x) L \phi_j(x)$, and $g_i = \int_\Omega \mathrm{d}x \phi_i(x) f(x)$ exactly as obtained in Sect. 8.2.3. The only difference arises in the definitions of the support functions $\phi_i(x)$. In general, if these functions are piecewise polynomials on some finite domain, they are called finite elements or splines. Finite elements make it possible to deal in a systematic fashion with regions having curved boundaries of an arbitrary shape. Also, one can systematically estimate the accuracy of the solution in terms of the parameters that label the finite-element family, and the solutions are no more difficult to generate than more complex variational methods.

In one space dimension, the simplest finite-element family begins with the set of step functions defined by

$$\phi_i(x) = \begin{cases} 1 & x_{i-1} \le x \le x_i \\ 0 & \text{otherwise} \,. \end{cases} \tag{8.65}$$

The use of these simple *hat* functions as a basis does not provide any advantage over the usual finite-difference schemes. However, for certain problems in two or more dimensions, finite-element methods have distinct advantages over other methods. Generally, the use of finite elements requires complex, sophisticated computer programs for implementation. The use of higher-order polynomials, commonly called splines, as a basis has been extensively used in atomic and molecular physics. An extensive literature is available [34, 35].

We illustrate the use of the finite-element method by applying it to the Schrödinger equation. In this case, the linear operator L is $H-E$, where, as usual, E is the energy, and the Hamiltonian H is the sum of the kinetic and potential energies, that is, $L = H - E = \mathcal{T} + V - E$ and $Lu(x) = 0$. We define the finite elements through support points, or knots, given by the sequence $\{x_1, x_2, x_3, \ldots\}$, which are not necessarily spaced uniformly. Since the *hat* functions have vanishing derivatives, we employ the next more complex basis, that is, *tent* functions, which are piecewise linear functions given by

$$\phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & x_{i-1} \le x \le x_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \,, \end{cases} \tag{8.66}$$

and for which the derivative is given by

$$\frac{\mathrm{d}}{\mathrm{d}x}\phi_i(x) = \begin{cases} \dfrac{1}{x_i - x_{i-1}} & x_{i-1} \le x \le x_i \\ \dfrac{-1}{x_{i+1} - x_i} & x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \,. \end{cases} \tag{8.67}$$

The functions have a maximum value of 1 at the midpoint of the interval $[x_{i-1}, x_{i+1}]$, with partially overlapping adjacent elements. In fact, the overlaps may be represented by a matrix $\boldsymbol{O}$ with elements

$$O_{ij} = \int_{-\infty}^{\infty} \mathrm{d}x\, \phi_i(x)\phi_j(x) \,. \tag{8.68}$$

Thus, if $i = j$,

$$\begin{aligned} O_{ii} &= \int_{x_{i-1}}^{x_i} \mathrm{d}x\, \frac{(x - x_{i-1})^2}{(x_i - x_{i-1})^2} + \int_{x_i}^{x_{i+1}} \mathrm{d}x\, \frac{(x - x_i)^2}{(x_{i+1} - x_i)^2} \\ &= \frac{1}{3}(x_{i+1} - x_{i-1}) \,; \end{aligned} \tag{8.69}$$

if $i = j - 1$,

$$\begin{aligned} O_{ij} &= \int_{x_i}^{x_{i+1}} \mathrm{d}x\, \frac{(x - x_i)(x_{i+1} - x)}{(x_{i+1} - x_i)^2} \\ &= \frac{1}{6}(x_{i+1} - x_i) \,; \end{aligned} \tag{8.70}$$

if $i = j + 1$,

$$\begin{aligned} O_{ij} &= \int_{x_{i-1}}^{x_i} \mathrm{d}x\, \frac{(x - x_{i-1})(x_i - x)}{(x_i - x_{i-1})^2} \\ &= \frac{1}{6}(x_i - x_{i-1}) \,; \end{aligned} \tag{8.71}$$

and $O_{ij} = 0$ otherwise.

The potential energy is represented by the matrix

$$V_{ij} = \int_{-\infty}^{\infty} \mathrm{d}x\, \phi_i(x)V(x)\phi_j(x) \,, \tag{8.72}$$

which may be well approximated by

$$\begin{aligned} V_{ij} &\approx V(x_i) \int_{-\infty}^{\infty} \mathrm{d}x\, \phi_i(x)\phi_j(x) \\ &= V(x_i)O_{ij} \,, \end{aligned} \tag{8.73}$$

if $x_j - x_i$ is small. The kinetic energy, $\mathcal{T} = -\frac{1}{2}\mathrm{d}^2/\mathrm{d}x^2$, is similarly given by

$$\mathcal{T}_{ij} = -\frac{1}{2} \int_{-\infty}^{\infty} \mathrm{d}x \phi_i(x) \frac{\mathrm{d}^2}{\mathrm{d}x^2} \phi_j(x) \,, \tag{8.74}$$

which we compute by integrating by parts, since the tent functions have a singular second derivative

$$\mathcal{T}_{ij} = \frac{1}{2} \int_{-\infty}^{\infty} \mathrm{d}x \left(\frac{\mathrm{d}}{\mathrm{d}x}\phi_i(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x}\phi_j(x) \right) \,, \tag{8.75}$$

which in turn is evaluated to yield

$$\mathcal{T}_{ij} = \begin{cases} \dfrac{x_{i+1} - x_{i-1}}{2(x_i - x_{i-1})(x_{i+1} - x_i)} & i = j \\ \dfrac{1}{2(x_i - x_{i+1})} & i = j - 1 \\ \dfrac{1}{2(x_{i-1} - x_i)} & i = j + 1 \\ 0 & \text{otherwise} \,. \end{cases} \tag{8.76}$$

Finally, since the Hamiltonian matrix is $H_{ij} = \mathcal{T}_{ij} + V_{ij}$, the solution vector $u_i(x)$ may be found by solving the eigenvalue equation

$$[H_{ij} - EO_{ij}]u_i(x) = 0 \,. \tag{8.77}$$

Going beyond this simple example, discrete variable representation (DVR) methods, also known as pseudospectral methods, and direct solution of the Schrödinger equation on numeral grids via finite-difference, finite-element, and high-order interpolant methods, have been adopted broadly in atomic, molecular, and optical physics as the power of computational resources has grown in recent decades. An introduction to DVR methods applied to solving the time-dependent Schrödinger equation for quantum dynamics of molecules, as an example, has been given by *Light* [36, 37].

8.2.5 Integral Equations

Central to much of practical and formal scattering theory is the integral equation and techniques of its solution. For example, in atomic collision theory, the Schrödinger differential equation

$$[E - H_0(\boldsymbol{r})]\psi(\boldsymbol{r}) = V(\boldsymbol{r})\psi(\boldsymbol{r}) , \tag{8.78}$$

where the Hamiltonian $H_0 \equiv -(\hbar^2/2m)\nabla^2 + V_0$ may be solved by exploiting the solution for a delta function source, i.e.,

$$(E - H_0)G(\boldsymbol{r},\boldsymbol{r}') = \delta(\boldsymbol{r}-\boldsymbol{r}') . \tag{8.79}$$

In terms of this *Green's function* $G(\boldsymbol{r},\boldsymbol{r}')$, and any solution $\chi(\boldsymbol{r})$ of the homogeneous equation (i.e., with $V(\boldsymbol{r}) = 0$), the general solution is

$$\psi(\boldsymbol{r}) = \chi(\boldsymbol{r}) + \int \mathrm{d}\boldsymbol{r}'\, G(\boldsymbol{r},\boldsymbol{r}')V(\boldsymbol{r}')\psi(\boldsymbol{r}') , \tag{8.80}$$

for which, given a choice of the functions $G(\boldsymbol{r},\boldsymbol{r}')$ and $\chi(\boldsymbol{r})$, particular boundary conditions are determined. This integral equation is the Lippmann–Schwinger equation of potential scattering. Further topics on scattering theory are covered in other chapters (especially Chaps. 49 to 62) and in standard texts such as those by *Joachain* [38], *Rodberg*, and *Thaler* [39], and *Goldberger* and *Watson* [40]. Owing especially to the wide variety of specialized techniques for solving integral equations, we briefly survey only a few of the most-frequently applied methods.

Integral Transforms

Certain classes of integral equations may be solved using integral transforms such as the Fourier or Laplace transforms. These integral transforms typically have the form

$$f(x) = \int \mathrm{d}x'\, K(x,x')g(x') , \tag{8.81}$$

where $f(x)$ is the integral transform of $g(x')$ by the kernel $K(x,x')$. Such a pair of functions is the solution of the Schrödinger equation (spatial wave function) and its Fourier transform (momentum representation wave function). *Arfken* [10], *Morse*, and *Feshbach* [12], and *Courant* and *Hilbert* [13] give other examples, as well as being excellent references for the application of integral equations and Green's functions in mathematical physics. In their analytic form, these transform methods provide a powerful method of solving integral equations for special cases. In addition, they may be implemented by performing the transform numerically.

Power Series Solution

For an equation of the form (in one dimension for simplicity)

$$\psi(r) = \chi(r) + \lambda \int \mathrm{d}r'\, K(r,r')\psi(r') , \tag{8.82}$$

a solution may be found by iteration. That is, as a first approximation, set $\psi_0(r) = \chi(r)$ so that

$$\psi_1(r) = \chi(r) + \lambda \int \mathrm{d}r'\, K(r,r')\chi(r') . \tag{8.83}$$

This may be repeated to form a power series solution, i.e.,

$$\psi_n(r) = \sum_{k=0}^{n} \lambda^k I_k(r) , \tag{8.84}$$

where

$$I_0(r) = \chi(r) , \tag{8.85}$$

$$I_1(r) = \int \mathrm{d}r'\, K(r,r')\chi(r') , \tag{8.86}$$

$$I_2(r) = \int \mathrm{d}r'' \int \mathrm{d}r'\, K(r,r')K(r',r'')\chi(r'') , \tag{8.87}$$

$$I_n(r) = \int \mathrm{d}r' \cdots \int \mathrm{d}r^{(n)}\, K(r,r')K(r,r'') \cdots K(r^{(n-1)},r^{(n)}) . \tag{8.88}$$

If the series converges, then the solution $\psi(r)$ is approached by the expansion. When the Schrödinger equation is cast as an integral equation for scattering in a potential, this iteration scheme leads to the *Born series*, the first term of which is the incident, unperturbed wave, and the second term is usually referred to simply as the *Born approximation*.

Separable Kernels

If the kernel is separable, i.e.,

$$K(r,r') = \sum_{k=1}^{n} f_k(r)g_k(r') , \tag{8.89}$$

where n is finite, then substitution into the prototype integral equation (8.82) yields

$$\psi(r) = \chi(r) + \lambda \sum_{k=1}^{n} f_k(r) \int \mathrm{d}r'\, g(r')\psi(r') . \tag{8.90}$$

Multiplying by $f_k(r)$, integrating over r, and rearranging yields the set of algebraic equations

$$c_j = b_j + \lambda \sum_{k=1}^{n} a_{jk} c_k \,, \tag{8.91}$$

where

$$c_k = \int \mathrm{d}r' g_k(r')\psi(r') \,, \tag{8.92}$$

$$b_k = \int \mathrm{d}r f_k(r)\chi(r) \,, \tag{8.93}$$

$$a_{jk} = \int \mathrm{d}r g_j(r) f_k(r) \,, \tag{8.94}$$

or, if $\boldsymbol{c}$ and $\boldsymbol{b}$ denote vectors, and $\boldsymbol{A}$ denotes the matrix of constants a_{jk},

$$\boldsymbol{c} = (\mathbf{1} - \lambda \boldsymbol{A})^{-1}\boldsymbol{b} \,. \tag{8.95}$$

The eigenvalues are the roots of the determinantal equation. Substituting these into $(\mathbf{1} - \lambda \boldsymbol{A})\boldsymbol{c} = 0$ yields the constants c_k that determine the solution of the original equation. This derivation may be found in the text by *Arfken* [10], along with an explicit example. Even if the kernel is not exactly separable, if it is approximately so, then this procedure can yield a result that can be substituted into the original equation as a first step in an iterative solution.

Numerical Integration

Perhaps the most straightforward method of solving an integral equation is to apply a numerical integration formula such as Gaussian quadrature. An equation of the form

$$\psi(r) = \int \mathrm{d}r' \, K(r, r')\chi(r') \tag{8.96}$$

can be approximated as

$$\psi(r_j) = \sum_{k=1}^{n} w_k K(r_j, r_k')\chi(r_k) \,, \tag{8.97}$$

where w_k are quadrature weights, if the kernel is well behaved. However, such an approach is not without pitfalls. In light of the previous section, this approach is equivalent to replacing the integral equation by a set of algebraic equations. In this example, we have

$$\psi_j = \sum_{k=1}^{n} M_{jk}\chi_k \,, \tag{8.98}$$

so that the solution of the equation is found by inverting the matrix $\boldsymbol{M}$. Since there is no guarantee that this matrix is not ill conditioned, the numerical procedure may not produce meaningful results. In particular, only certain classes of integral equations and kernels will lead to stable solutions.

8.3 Computational Linear Algebra

Previous sections of this chapter dealt with interpolation, differential equations, and related topics. Generally, discretization methodologies lead to classes of algebraic equations. In recent decades, enormous progress has been made in developing algorithms for solving linear-algebraic equations [41]. Many of the most widely adopted computational linear algebra routines are available through *Netlib* [42], a portal developed to facilitate the distribution of such software for use in scientific computation. These routines include packages such as the Basic Linear Algebra Subprograms (BLAS) [42, 43], which performs vector addition, dot products, matrix multiplications, etc., and the Linear Algebra Package (LAPACK, and ScaLAPACK its distributed memory implementation) [42, 44], which is used for solving linear systems of equations, eigenvalue problems, factorizations, and decompositions, etc.

Here, we discuss methods for solving systems of equations such as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \,, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \,, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \,. \end{aligned} \tag{8.99}$$

In these equations, a_{ij} and b_i form the set of known quantities, and x_i must be determined. The solution to these equations can be found if they are linearly independent. Numerically, problems can arise due to truncation and round-off errors that lead to an approximate linear dependence [45]. In this case, the set of equations are approximately singular, and special methods must be invoked. Much of the complexity of modern algorithms comes from minimizing the effects of such errors. For relatively small sets of nonsingular equations, direct methods in which the solution is obtained after a definite number of operations can work well. However, for very large systems iterative techniques are preferable [46].

A great many algorithms are available for solving Eq. (8.99), depending on the structure of the coefficients. For example, if the matrix of coefficients $\boldsymbol{A}$ is dense, using Gaussian elimination takes $2n^3/3$ operations; if $\boldsymbol{A}$ is also symmetric and positive definite, using the Cholesky algorithm takes a factor of 2 fewer operations. If $\boldsymbol{A}$ is triangular, that is, either zero above the diagonal or zero below the diagonal, we can solve the above system by simple substitution in only n^2 operations. For example, if $\boldsymbol{A}$ arises from solving certain elliptic partial differential equations, such as Poisson's equation, then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ can be solved using multigrid methods in only n operations.

We shall outline below how to solve Eq. (8.99) using elementary Gaussian elimination. More advanced methods, such as the *conjugate gradient, generalized minimum residuals*, and the *Lanczos method* are treated elsewhere [47].

To solve $\boldsymbol{Ax} = \boldsymbol{b}$, we first use Gaussian elimination to factor the matrix $\boldsymbol{A}$ as $\boldsymbol{PA} = \boldsymbol{LU}$, where $\boldsymbol{L}$ is lower triangular, $\boldsymbol{U}$ is upper triangular, and $\boldsymbol{P}$ is a matrix that permutes the rows of $\boldsymbol{A}$. Then we solve the triangular system $\boldsymbol{Ly} = \boldsymbol{Pb}$ and $\boldsymbol{Ux} = \boldsymbol{y}$. These last two operations are easily performed using standard linear algebra libraries. The factorization $\boldsymbol{PA} = \boldsymbol{LU}$ takes most of the time. Reordering the rows of $\boldsymbol{A}$ with $\boldsymbol{P}$ is called pivoting and is necessary for numerical stability. In the standard partial pivoting scheme, $\boldsymbol{L}$ has 1s on its diagonal and other entries bounded in absolute value by 1. The simplest version of Gaussian elimination involves adding multiples of one row of $\boldsymbol{A}$ to others to zero out subdiagonal entries, and overwriting $\boldsymbol{A}$ with $\boldsymbol{L}$ and $\boldsymbol{U}$.

We first describe the decomposition of $\boldsymbol{PA}$ into a product of upper and lower triangular matrices,

$$\boldsymbol{A}' = \boldsymbol{LU} , \tag{8.100}$$

where the matrix $\boldsymbol{A}'$ is defined by $\boldsymbol{A}' = \boldsymbol{PA}$. Writing out the indices, we obtain

$$A'_{ij} = \sum_{k=1}^{\min(i,j)} L_{ik} U_{kj} . \tag{8.101}$$

We shall make the choice

$$L_{ii} = 1 . \tag{8.102}$$

These equations have the remarkable property that the elements A'_{ij} of each row can be scanned in turn, writing L_{ij} and U_{ij} into the locations A'_{ij} as we go. At each position (i, j), only the current A'_{ij} and already-calculated values of $L_{i'j'}$ and $U_{i'j'}$ are required. To see how this works, consider the first few rows. If $i = 1$,

$$A'_{1j} = U_{1j} , \tag{8.103}$$

defining the first row of $\boldsymbol{L}$ and $\boldsymbol{U}$. The U_{1j} are written over the A'_{1j}, which are no longer needed. If $i = 2$,

$$\begin{aligned} A'_{21} &= L_{21} U_{11} , && j = 1 \\ A'_{2j} &= L_{21} U_{1j} + U_{2j} , && j \geq 2 . \end{aligned} \tag{8.104}$$

The first line gives L_{21} and the second U_{2j}, in terms of existing elements of $\boldsymbol{L}$ and $\boldsymbol{U}$. The U_{2j} and L_{21} are written over the A'_{2j}. (Remember that $L_{ii} = 1$ by definition.) If $i = 3$,

$$\begin{aligned} A'_{31} &= L_{31} U_{11} , && j = 1 \\ A'_{32} &= L_{31} U_{12} + L_{32} U_{22} , && j = 2 \\ A'_{3j} &= L_{31} U_{1j} + L_{32} U_{2j} + U_{3j} , && j \geq 3 , \end{aligned} \tag{8.105}$$

yielding in turn L_{31},L_{32}, and U_{3j}, which are written over A'_{3j}.

The algorithm should now be clear. At the i-th row,

$$\begin{aligned} L_{ij} &= U_{jj}^{-1} \left(A'_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \right) , && j \leq i - 1 \\ U_{ij} &= A'_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj} , && j \geq i . \end{aligned} \tag{8.106}$$

We observe from the first line of these equations that the algorithm may run into numerical inaccuracies if any U_{jj} becomes very small. Now $U_{11} = A'_{11}$, while in general $U_{ii} = A'_{ii} - \cdots$. Thus the absolute values of the U_{ii} are maximized if the rows are rearranged so that the absolutely largest elements of $\boldsymbol{A}'$ in each column lie on the diagonal. Note that the solutions are unchanged by permuting the rows (same equations, different order).

The *LU decomposition* can now be used to solve the system. This relies on the fact that the inversion of a triangular matrix is a simple process of back substitution. We replace Eq. (8.99) by two systems of equations. Written out in full, the equations for a typical column of $\boldsymbol{y}$ look like

$$\begin{aligned} L_{11} y_1 &= b'_1 , \\ L_{21} y_1 + L_{22} y_2 &= b'_2 , \\ L_{31} y_1 + L_{32} y_2 + L_{33} y_3 &= b'_3 , \\ &\vdots , \end{aligned} \tag{8.107}$$

where the vector $\boldsymbol{b}'$ is $\boldsymbol{p}' = \boldsymbol{Pb}$. Thus, from successive rows, we obtain $y_1, y_2, y_3, \ldots$ in turn

$$\begin{aligned} U_{11} x_1 &= y_1 , \\ U_{12} x_1 + U_{22} x_2 &= y_2 , \\ U_{13} x_1 + U_{23} x_2 + U_{33} x_3 &= y_3 , \\ &\vdots , \end{aligned} \tag{8.108}$$

and from successive rows of the latter, we obtain $x_1, x_2, x_3, \ldots$ in turn.

Software libraries (*Netlib* [42], described above) also exists for evaluating all the error bounds for dense and band matrices. Gaussian elimination with pivoting is almost always numerically stable, so the error bound one expects from solving these equations is of the order of $n\epsilon$, where ϵ is related to the condition number of the matrix $\boldsymbol{A}$. A good discussion of errors and conditioning is given in [3–5].

8.4 Monte Carlo Methods

Owing to the continuing rapid development of computational facilities and the ever-increasing desire to perform *ab initio* calculations, the use of Monte Carlo methods is becoming widespread as a means to evaluate previously intractable multidimensional integrals and to enable complex modeling and simulation.

For example, a wide range of applications broadly classified as *quantum Monte Carlo* have been used to compute, for example, the ground-state eigenfunctions of simple molecules. Also, guided random walks have found application in the computation of Green's functions, and variables chosen randomly, subject to particular constraints, have been used to mimic the electronic distribution of atoms. The latter application, used in the *classical trajectory Monte Carlo technique* (CTMC) described in Chap. 62, allows the statistical quasiquantal representation of ion–atom collisions. CTMC is akin to another class of Monte Carlo simulations, classical molecular dynamics, which is used to describe systems ranging from molecules to solids as being composed of atoms whose movement is governed by classical mechanics subject to quantum mechanically derived potentials.

Here, we summarize the basic tools needed in these methods and how they may be used to produce specific distributions and make tractable the evaluation of multidimensional integrals with complicated boundaries. Detailed descriptions of these methods can be found in [3–5, 18, 48].

8.4.1 Random Numbers

An essential ingredient of any Monte Carlo procedure is the availability of a computer-generated sequence of random numbers that is not periodic and is free of other significant statistical correlations. Often, such numbers are termed *pseudorandom* or *quasirandom*, in distinction to truly random physical processes. While the quality of random number generators supplied with computers has greatly improved over time, it is important to be aware of the potential dangers that can be present. For example, many systems are supplied with a random number generator based on the *linear congruential method*. Typically, a sequence of integers $n_1, n_2, n_3, \ldots$ is first produced between 0 and $N-1$ by using the recurrence relation

$$n_{i+1} = (an_i + b) \bmod N\ , \qquad 0 \le i < N-1\ , \quad (8.109)$$

where a, b, N, and the seed value n_0 are positive integers. Real numbers between 0 and (strictly) 1 are then obtained by dividing by N. The period of this sequence is at most N and depends on the judicious choice of the constants, with N being limited by the word size of the computer. A user who is unsure whether the character of the random numbers generated on a particular computer platform is proper can perform additional randomizing shuffles or use a portable random number generator; both procedures are described in detail in the texts by *Knuth* [6] and *Press* et al. [3–5], for example. In addition, tests of random number sequences have been developed, for example, the widely used tests published by *L'Ecuyer* and *Simard* [49].

The need for such tests has grown in the present era of parallel computing, relevant not only to use of parallel random number generators on supercomputers and clusters but also the use of personal computers and workstations employing multicore/multithread, computation. In fact, the scale of Monte Carlo calculations possible on contemporary platforms has exposed deficiencies of many commonly used methods of generating random numbers, leading to development of more robust techniques. These new methods seek to ensure the quality of random numbers, first of all for large sequences within a single stream, as well as for multiple streams (other threads on a single processing unit or across nodes within a cluster). A widely used set of parallel random number generators that pass robust tests was developed and described by *Srinivasan*, *Mascagni*, and *Ceperley* [50], for example.

8.4.2 Distributions of Random Numbers

Most distributions of random numbers begin with sequences generated uniformly between a lower and an upper limit, and are therefore called *uniform deviates*. However, it is often useful to draw the random numbers from other distributions, such as the Gaussian, Poisson, exponential, gamma, or binomial distributions. These are particularly useful in modeling data or supplying input for an event generator or simulator. In addition, as described below, choosing the random numbers according to some weighting function can significantly improve the efficiency of integration schemes based on Monte Carlo sampling.

Perhaps the most direct way to produce the required distribution is the *transformation method*. If we have a sequence of uniform deviates x on $(0, 1)$ and wish to find a new sequence y that is distributed with probability given by some function $f(y)$, it can be shown that the required transformation is given by

$$y(x) = \left[\int_0^y f(y)\mathrm{d}y\right]^{-1} . \quad (8.110)$$

Evidently, the indefinite integral must be both known and invertible, either analytically or numerically. Since this is seldom the case for distributions of interest, other less direct methods are most often applied. However, even these other methods often rely on the transformation method as one stage of the procedure. The transformation method may also be generalized to more than one dimension [3–5].

A more widely applicable approach is the *rejection method*, also known as *von Neumann rejection*. In this case, if one wishes to find a sequence y distributed according to $f(y)$, one first chooses another function $\tilde{f}(y)$, called the comparison function, which is everywhere greater than $f(y)$ on the desired interval. In addition, a way must exist to generate y according to the comparison function, such as use of the transformation method. Thus, the comparison function must be simpler or better known than the distribution to be found. One simple choice is a constant function that is larger than the maximum value of $f(y)$, but choices that are *closer* to $f(y)$ will be much more efficient.

To proceed, y is generated uniformly according to $\tilde{f}(y)$, and another deviate x is chosen uniformly on $(0, 1)$. One then rejects or accepts y, depending on whether x is greater than or less than the ratio $f(y)/\tilde{f}(y)$, respectively. The fraction of trial numbers accepted depends on the ratio of the area under the desired function to that under the comparison function. Clearly, the efficiency of this scheme depends on how few of the numbers initially generated must be rejected, and, therefore, on how closely the comparison function approximates the desired distribution. The Lorentzian distribution, for which the inverse definite integral is known (the tangent function), is a good comparison function for a variety of *bell-shaped* distributions such as the Gaussian (normal), Poisson, and gamma distributions.

Especially for distributions that are functions of more than one variable and possess complicated boundaries, the rejection method is impractical, and the transformation method simply inapplicable. In the 1950s, a method to generate distributions for such situations was developed and applied in the study of statistical mechanics, where multidimensional integrals (e.g., the partition function) must often be solved numerically. It is known as the *Metropolis algorithm*. This procedure, or its variants, has more recently been adopted to aid in the computation of eigenfunctions of complicated Hamiltonians and scattering operators. In essence, the Metropolis method generates a random walk through the space of the dependent variables, and in the limit of a large number of steps in the walk, the points visited approximate the desired distribution.

In its simplest form, the Metropolis method generates this distribution of points by stepping through this space, most frequently taking a step *downhill* but sometimes taking a step *uphill*. That is, given a set of coordinates $\boldsymbol{q}$ and a desired distribution function $f(\boldsymbol{q})$, a trial step is taken from the i-th configuration $\boldsymbol{q}_i$ to the next, depending on whether the ratio $f(\boldsymbol{q}_i+1)/f(\boldsymbol{q}_i)$ is greater or less than 1. If the ratio is greater than 1, the step is accepted, but if it is less than 1, the step is accepted with a probability given by the ratio.

8.4.3 Monte Carlo Integration

The basic idea of Monte Carlo integration is that if a large number of points is generated uniformly randomly in some n-dimensional space, the number falling inside a given region is proportional to the volume, or definite integral, of the function defining that region. Although this idea is as true in one dimension as it is in n, unless there is a large number (*large* could be as little as three) of dimensions or the boundaries are quite complicated, the numerical quadrature schemes described previously are more accurate and efficient. However, since the Monte Carlo approach is based on just sampling the function at representative points rather than evaluating the function at a large number of finely spaced quadrature points, its advantage for very large problems is apparent.

For simplicity, consider the Monte Carlo method for integrating a function of only one variable; the generalization to n dimensions being straightforward. If we generate N random points uniformly on (a, b), then in the limit of large N, the integral is

$$\int_a^b f(x)\mathrm{d}x \approx \frac{1}{N}\langle f(x)\rangle \pm \sqrt{\frac{\langle f^2(x)\rangle - \langle f(x)\rangle^2}{N}}\,, \quad (8.111)$$

where

$$\langle f(x)\rangle \equiv \frac{1}{N}\sum_{i=1}^{N} f(x_i) \quad (8.112)$$

is the arithmetic mean. The probable error given is appropriately a statistical one rather than a rigorous error bound and is the one standard error limit. From this, one can see that the error decreases only as $N^{1/2}$, more slowly than the rate of decrease for the quadrature schemes based on interpolation. Also, the accuracy is greater for relatively smooth functions, since the Monte Carlo generation of points is unlikely to sample narrowly peaked features of the integrand well. To estimate the integral of a multidimensional function with complicated boundaries, find an enclosing volume and generate points uniformly randomly within it. Keeping the enclosing volume as close as possible to the volume of interest minimizes the number of points that fall outside, and therefore increases the efficiency of the procedure.

The Monte Carlo integral is related to the techniques for generating random numbers according to prescribed distributions described in Sect. 8.4.2. If we consider a normalized distribution $w(x)$, known as the *weight function*, then with the change of variables defined by

$$y(x) = \int_a^x w(x')\mathrm{d}x'\,, \quad (8.113)$$

the Monte Carlo estimate of the integral becomes

$$\int_a^b f(x)\mathrm{d}x \approx \frac{1}{N}\left\langle \frac{f[x(y)]}{w[x(y)]}\right\rangle, \quad (8.114)$$

assuming that the transformation is invertible. Choosing $w(x)$ to behave approximately as $f(x)$ allows a more efficient generation of points within the boundaries of the integrand. This occurs since the uniform distribution of points y results in values of x distributed according to w and, therefore, close to f. This procedure, generally termed the *reduction of variance* of the Monte Carlo integration, improves the efficiency of the procedure to the extent that the transformed function f/w can be made smooth, and that the sampled region is as small as possible but still contains the volume to be estimated.

References

1. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer, New York (2002)
2. Burden, R.L., Faires, J.D.: Numerical Analysis. Thomson Brooks/Cole, Belmont (2005)
3. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes, the Art of Scientific Computing. Cambridge Univ. Press, Cambridge (2007)
4. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in Fortran 77: The Art of Scientific Computing. Cambridge Univ. Press, Cambridge (1992)
5. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in Fortran 90: The Art of Parallel Scientific Computing. Cambridge Univ. Press, Cambridge (1996)
6. Knuth, D.E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, Boston (1998)
7. Abramowitz, M., Stegun, I.A. (eds.): Handbook of Mathematical Functions. Applied Mathematics Series, vol. 55. National Bureau of Standards/Dover, Washington/New York (1972). NIST Digital Library of Mathematical Functions: dlmf.nist.gov
8. Gradshteyn, I.S., Ryzhik, I.M.: Tables of Integrals, Series, and Products. Elsevier, Amsterdam (2007)
9. Zwillinger, D.: CRC Standard Mathematical Tables and Formulae. CRC, Boca Raton (2012)
10. Arfken, G.B., Weber, H.J., Harris, F.E.: Mathematical Methods for Physicists. Elsevier, Amsterdam (2013)
11. Whittaker, E.T., Watson, G.N.: A Course of Modern Analysis. Cambridge Univ. Press, Cambridge (2006)
12. Morse, P.M., Feshbach, H.: Methods of Theoretical Physics. McGraw-Hill, Boston (1999)
13. Courant, R., Hilbert, D.: Methods of Mathematical Physics. Interscience, New York (2009)
14. Margenau, H., Murphy, G.M.: The Mathematics of Physics and Chemistry. Van Nostrand, New York (1976)
15. Hamming, R.W.: Numerical Methods for Scientists and Engineers. McGraw-Hill, New York (1973)
16. Jeffreys, H., Jeffreys, B.S.: Methods of Mathematical Physics. Cambridge Univ. Press, Cambridge (1999)
17. Bender, C.M., Orszag, S.: Advanced Mathematical Methods for Scientists and Engineers. Springer, New York (1999)
18. Koonin, S.E., Meredith, D.C.: Computational Physics. Addison-Wesley, Reading (1995)
19. Wikipedia: List of numerical-analysis software (2021). https://en.wikipedia.org/wiki/List_of_numerical_analysis_%software
20. Computer Physics Communications Program Library: www.cpc.cs.qub.ac.uk/
21. Huber, P.J., Ronchetti, E.M.: Robust Statistics. Wiley, Hoboken (2009)
22. Young, H.D.: Statistical Treatment of Experimental Data: An Introduction to Statistical Methods. Waveland Press, Prospect Heights (1996)
23. Bevington, P.R., Robinson, D.K.: Data Reduction and Error Analysis for the Physical Sciences. McGraw-Hill, New York (2003)
24. Champeney, D.C.: Fourier Transforms and Their Physical Applications. Academic Press, London (1988)
25. Elliott, D.F., Rao, K.R.: Fast Transforms: Algorithms, Analyses, Applications. Academic Press, London (1982)
26. FFTW: https://en.wikipedia.org/wiki/FFTW; www.fftw.org
27. Lambert, J.D.: Numerical Methods for Ordinary Differential Equations: The Initial Value Problem. Chichester, New York (2000)
28. Shampine, L.F.: Numerical Solution of Ordinary Differential Equations. Chapman Hall, New York (1994)
29. Butcher, J.: The Numerical Analysis of Ordinary Differential Equations: Runge–Kutta and General Linear Methods. Wiley, New York (1987)
30. Hall, G., Watt, J.M.: Modern Numerical Methods for Ordinary Differential Equations. Clarendon, Oxford (1976)
31. Gladwell, I., Wait, R. (eds.): A Survey of Numerical Methods for Partial Differential Equations. Clarendon, Oxford (1979)
32. Rektorys, K.: Variational Methods in Mathematics, Science, and Engineering. Springer Netherlands, Amsterdam (2012)
33. Cook, R.D., Malkus, D.S., Plesha, M.E., Witt, R.J.: Concepts and Applications of Finite Element Analysis. Wiley, New York (2002)
34. Nürnberger, G.: Approximation by Spline Functions. Springer, Berlin, Heidelberg (1989)
35. DeBoor, C.: Practical Guide to Splines. Springer, Berlin (2013)
36. Light, J.C., Carrington, T.: Discrete variable representation and their utilization. Adv. Chem. Phys. **114**, 263 (2000)
37. Light, J.C., Hamilton, I.P., Lill, J.V.: Generalized discrete variable approximation in quantum mechanics. J. Chem. Phys. **82**, 1400 (1985)
38. Joachain, C.J.: Quantum Collision Theory. North Holland, Amsterdam (1987)
39. Rodberg, L.S., Thaler, R.M.: Introduction to the Quantum Theory of Scattering. Academic Press, New York (1970)
40. Goldberger, M.L., Watson, K.M.: Collision Theory. Dover, Mineola (2004)
41. Ciarlet, P.G.: Introduction to Numerical Linear Algebra and Optimisation. Cambridge Univ. Press, Cambridge (2001)
42. Netlib Repository at UTK and ORNL: www.netlib.org
43. en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprogams
44. Wikipedia: LAPACK (2021). https://en.wikipedia.org/wiki/LAPACK
45. Golub, G., Van Loan, C.: Matrix Computations. Johns Hopkins Univ. Press, Baltimore (2013)
46. Hackbusch, W.: Iterative Solution of Large Sparse Systems of Equations. Springer, Cham (2016)
47. George, A., Liu, J.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs (1981)
48. Kalos, M.H., Whitlock, P.A.: The Basics of Monte Carlo Methods. Wiley, Weinheim (2008)
49. L'Ecuyer, P., Simard, R.: TestU01: A C library for empirical testing of random number gen-erators. ACM Trans. Math. Softw. **33**(4), 22 (2007)
50. Srinivasan, A., Mascagni, M., Ceperley, D.: Testing parallel random number generators. Parallel Comput. **29**, 69 (2003)

David Schultz David Schultz is Vice President for Research at Northern Arizona University and has held research and administrative positions at the University of North Texas, Oak Ridge National Laboratory, and the University of Tennessee. His interests are in computational atomic physics, using both molecular dynamics simulation and quantum mechanical, discrete variable representations, and in applications in plasma science and astrophysics.