Numerical Analysis

Arieh Iserles

Centre for Mathematical Sciences
University of Cambridge
Cambridge CB3 0WA, United Kingdom
Phone: (44) 1223-337891
Fax: (44) 1223-765900
E-mail: A.Iserles@damtp.cam.ac.uk

Synonyms

Computational mathematics, scientific computing, numerical mathematics, real-number computation

Definition

The development of practical algorithms to obtain approximate solutions of mathematical problems and the validation of these solutions through their mathematical analysis.

Overview

Mathematics typically investigates concepts in their qualitative setting: existence, uniqueness and a wide range of analytic, topological, geometric and algebraic features. It is often important, however, to flesh out the numbers, accompany qualitative insight with computation. This is vital in applications of mathematics in science and engineering – it is not enough to prove that the trajectory of a spacecraft obeys dynamical features or geometric invariants, it is also indispensable to know where the spacecraft will be at any given instant. It is, moreover, increasingly important in mathematical research, because computation affords us the means to investigate mathematical phenomena, to gain insight and form conjectures that ultimately lead to theorems.

Numerical analysis has a distinct character from the rest of mathematical analysis in that it is concerned also with accuracy, speed and computational efficiency. The

question, thus, is not just whether a numerical method converges to the exact solution but also what is the rate of convergence and the computational cost.

Numerical analysis should not be confused with algorithmic aspects of discrete mathematics or with symbolic computation. For example, a symbolic package can be used to find the indefinite integral of a given function, provided that it can be obtained from known tables using basic principles, while a numerical package computes an approximate integral by quadrature.

The fundamental origin of the tension between much of mathematical research and computation is that analytic concepts and structures are central to mathematical discourse, while computation is an algebraic process, consisting of a discrete and finite sequence of algebraic operations on finite quantities. This is implicit in the very nature of digital electronic computers.

The process of discretization, a feature of many numerical algorithms, does not take the problem outside the scope of mathematics. A discretized problem can still be addressed in mathematical terms and with rigour – indeed, typically, once discretized, a mathematical problem is likely to become more difficult. Its redeeming virtue (and a basic requirement of any numerical method) is that it can be rendered in an algorithmic manner, suitable for computation.

Many familiar differential equations of mathematical physics originated as limits of discretizations, for example the diffusion equation and Kepler's laws (Wanner 2010). This illustrates the important issue that discretizations and numerical algorithms are not just an attempt to associate numbers with mathematical concepts but a major tool in extending our understanding of these concepts.

Inasmuch as contemporary numerical analysis is inconceivable without the availability of powerful computational platforms, fundamental concepts of mathematical computation with real numbers have exercised mathematicians since the dawn of the discipline. Major concepts and ideas of numerical analysis can be traced to some of the most illustrious names in the history of mathematics, from Archimedes to Johannes Kepler, Sir Isaac Newton, Leonhard Euler and Carl Friedrich Gauss.

The basic tools

Two core methodologies form the broad foundation of numerical analysis: numerical linear algebra and approximation theory.

Numerical linear algebra

No matter which mathematical problem we seek to compute, whether a differential or integral equation or a nonlinear system of algebraic equations, typically the algorithmic task ultimately reduces to linear algebraic computations. Reliability, efficiency and cost of such computations are thus central to any numerical analysis algorithm.

The basic numerical linear algebra problem is the solution of a linear system $A\mathbf{x} = \mathbf{b}$, where A is an $n \times n$ nonsingular matrix and $\mathbf{b}$ is a column vector of length n. This can be done by direct methods, basically variants of *Gaussian elimination*, for

systems of moderate size but a powerful approach for large matrices is to employ iterative methods. Paradoxically, converting a finite problem to an infinite one – in other words, replacing an algebraic by an analytic problem – turns out to be very useful indeed and it allows for efficient solution of very large algebraic systems (Golub & Van Loan 1996). An important tool in the design of iterative algorithms is the concept of a *Krylov subspace* $\mathcal{K}_m(B, \mathbf{v}) = \text{Span}\{\mathbf{v}, B\mathbf{v}, \ldots, B^{m-1}\mathbf{v}\}$. Many successful iterative algorithms, not least the method of *conjugate gradients* and its many variants and generalisations, evolve the solution vector in the space $\mathcal{K}_m(B, \mathbf{v})$ for appropriate choices of a matrix B and a vector $\mathbf{v}$.

Another major numerical linear algebra problem is least squares computation. Thus, A is an $n \times m$ matrix, $\mathbf{b}$ is a column vector of length n and we seek a column vector $\mathbf{x}$ of length m that minimises $\|A\mathbf{x} - \mathbf{b}\|$ in the Euclidean norm (Björck 1996).

Unlike the solution of a linear system or a least squares problem, eigenvalues and singular values of a matrix are not in general available in a finite and explicit form. Their computation belongs in the realm of numerical linear algebra although, strictly speaking, the label 'algebra' refers to the algebraic origin of the problem rather than to the computational methodology, which requires approximation and iterative algorithms (Golub & Van Loan 1996).

Two structural features of algebraic problems impact heavily on the difficulty of their computation: normalcy and sparsity. A matrix is *normal* if it has a basis of unitary eigenvectors – in particular, symmetric matrices are normal. Heavily non-normal matrices often present substantive computational challenge, due to their bad conditioning.

The entries of a *sparse* matrices are mostly zero. This can be exploited to a very good effect in their calculation and brings very large algebraic systems and eigenvalue problems within the realm of efficient computation.

Approximation theory

The focus here is on approximating functions using a finite set of simpler functions. Given a function f, a familiar problem is to approximate it as a linear combination of elements from a Banach space $\mathcal{B}$. One instance is interpolation, when we seek a function in $\mathcal{B}$ that coincides with given function values at a finite set of points. Another is when, given a function f, we seek $\tilde{f} \in \mathcal{B}$ that minimises $\|f - \tilde{f}\|$ across $\mathcal{B}$. Familiar Banach spaces used in approximation theory consist of polynomials, trigonometric polynomials, wavelets, splines or translates of a given 'master function'. They are often subspaces of L_p or of a Sobolev space W_p^m for some $p \in [1, \infty]$ and $m > 0$ (Cheney & Light 2000, Powell 1981).

The issues addressed by approximation theory are both the design of efficient and robust algorithms for such problems and an investigation of their features. At a more fundamental level, the theory investigates approximation properties of underlying function spaces (DeVore & Lorentz 1993). Thus, suppose that a function f from the infinite-dimensional Banach space $\mathcal{V}$ is approximated from the n-dimensional subspace $\mathcal{B}_n \subset \mathcal{V}$. What is the least error, as measured in the underlying norm? Does it go to zero – and how fast – as $n \to \infty$?

This basic framework is often generalised. The underlying problem might not

reduce easily to finding the best linear combination from an n-dimensional linear space when, for example, we approximate with spline functions while optimising the location of their knots, or with rational functions, or seek a convex approximation to a convex function. Likewise, instead of a Banach space we may consider a more general metric space once we wish to approximate functions or data residing on manifolds.

Approximation theory uses a wide range of techniques from functional analysis, theory of orthogonal polynomials, analytic function theory, differential geometry and, increasingly, harmonic analysis.

Main subject areas of numerical analysis

The concerns of numerical analysis span a large swathe of mathematical problems in addition to those that can be formulated primarily within the framework of linear algebra or approximation theory.

Nonlinear equations

Finding the solution of a nonlinear algebraic system of equations, $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, or equivalently determining a fixed point $\mathbf{g}(\mathbf{x}) = \mathbf{x}$, is one of the oldest problems of numerical analysis. Univariate methods like bisection and *regula falsi,* as well as the Newton–Raphson iteration

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[\frac{\partial \mathbf{f}(\mathbf{x}_n)}{\partial \mathbf{x}}\right]^{-1} \mathbf{f}(\mathbf{x}_n), \qquad n = 0, 1, \ldots, \qquad \mathbf{x}_0 \text{ given},$$

applicable in the multivariate setting, have been known for centuries.

Provided that the Jacobian matrix can be calculated easily and affordably, the Newton–Raphson method and its derivatives are effective for systems with moderate number of variables. However, most modern methods aim for greater generality (the Jacobian is not computed, indeed the differentiability of $\mathbf{f}$ need not be assumed) and for systems with very large number of variables. Such methods can be roughly classified into two groups: one reformulates the underlying problem as an optimisation of a given objective function (e.g., of $\|\mathbf{f}(\mathbf{x})\|^2$ in Euclidean norm), the other uses homotopy algorithms.

Quadrature and cubature

The standard paradigm of univariate quadrature is to approximate

$$\int_a^b f(x)w(x)dx \approx \sum_{m=1}^{s} b_m f(c_m),$$

where the weights b_m and the nodes c_m depend upon the weight function w, but are independent of the function f (Davis & Rabinowitz 1975). Well-known formalæ

include Gauss–Christoffel quadrature, which selects weights and nodes to maximise the accuracy for polynomial functions f, and Clenshaw–Curtis quadrature, whereby weights and nodes are chosen to allow for rapid calculation with the Fast Fourier Transform.

While this paradigm can be generalised to the multivariate setting by tensor products and bespoke quadrature rules have been introduced for specific multivariate domains, efficiency deteriorates rapidly as the number of variables increases. In that instance there is an advantage in using cubature methods based upon probabilistic concepts, e.g. Monte Carlo and quasi-Monte Carlo techniques, because they are resistant to this "curse of dimensionality".

Ordinary differential equations

Given the ordinary differential (ODE) system $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, where $\mathbf{f}$ is suitably regular, accompanied by the initial condition $\mathbf{y}(t_0) = \mathbf{y}_0$, it is usual to compute the numerical solution in a time-stepping manner. Thus, having already computed $\mathbf{y}_k \approx \mathbf{y}(t_k)$, where $t_k = t_{k-1} + h_{k-1}$, $k = 1, \ldots, n$, we compute a new approximation $\mathbf{y}_{n+1}$ at $t_{n+1} = t_n + h_n$ (Hairer, Nørsett & Wanner 1993). The two most popular types of time-stepping algorithms are *multistep methods*

$$\sum_{m=0}^{s} \rho_m \mathbf{y}_{n-s+m+1} = h \sum_{m=0}^{s} \sigma_m \mathbf{f}(t_{n-s+m+1}, \mathbf{y}_{n-s+m+1}), \qquad \rho_s = 1$$

(here we assume that $h_k \equiv h$) and *Runge–Kutta methods*

$$\begin{aligned} \mathbf{k}_m &= \mathbf{f}(t_n + c_m h_n, \mathbf{y}_n + h_n \sum_{j=1}^{\nu} a_{m,j} \mathbf{k}_j), \qquad m = 1, \ldots, \nu, \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + h_n \sum_{m=1}^{\nu} b_m \mathbf{k}_m. \end{aligned}$$

Note that practical computation by these methods often requires the solution of an algebraic system of equations at each step.

Convergence is a necessary requirement of a method for ODEs: given a compact interval $[t_0, t^*]$, the numerical solution must tend uniformly to the exact solution when $\max h_n \to 0$. This global concept is closely associated with *consistency*, the numerical solution locally matching the exact solution to the ODE up to $O(h_n^{p+1})$ for some $p \geq 1$. While consistency suffices for the convergence of a Runge–Kutta method, an additional condition (the root condition on $\sum_{m=0}^{s} \rho_m w^m$, also known as zero-stability) is required for multistep methods.

Particular difficulty arises when the ODE models phenomena that, while decaying over time, proceed at vastly different rates. Typically for such *stiff* ODEs, the Jacobian $\partial \mathbf{f}(t, \mathbf{y})/\partial \mathbf{y}$ has eigenvalues with negative real parts, yet different orders of magnitude. Successful solution of stiff ODEs requires the method to possess an appropriate level of stability, e.g. A-stability (Hairer & Wanner 1996).

This general area also includes numerical study of two-point boundary value problems, whereby boundary conditions are given at the endpoints, as well as *Differential-Algebraic Equations* $\mathbf{f}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}$, where the Jacobian $\partial \mathbf{f}/\partial \mathbf{y}'$ is singular.

Partial differential equations

The breadth of the discipline of theoretical Partial Differential Equations (PDEs) and the extent of their applications are mirrored by the wide range of different methodologies employed in their discretization.

Finite difference methods impose a grid upon the underlying domain and approximate derivatives by algebraic relationships of the discretized solution at grid points (Iserles 2008). For example, the diffusion equation $\partial u/\partial t = \partial^2 u/\partial x^2$, where $u(x,t)$ is considered for $x \in [0,1]$ and $t \geq 0$, with initial conditions for $t = 0$ and Dirichlet boundary conditions at $x = 0$ and $x = 1$, can be discretized by

$$\frac{u_m^{n+1} - u_m^n}{\Delta t} = \frac{u_{m-1}^n - 2u_m^n + u_{m+1}^n}{(\Delta x)^2}, \qquad m = 1, \ldots, N,$$

where N is the number of internal grid points, $\Delta x = 1/(N+1)$, $\Delta t > 0$ and $u_m^n \approx u(m\Delta x, n\Delta t)$.

An alternative approach seeks a weak solution u_N to the PDE $\mathcal{L}u = f$ in an N-dimensional subspace $\mathcal{H}_N$ of the underlying Hilbert space $\mathcal{H}$ (typically, a Sobolev space), whether directly, by requiring that $\langle \mathcal{L}u_N - f, v \rangle = 0$ for all $v \in \mathcal{H}_N$ (the *Galerkin method*) or by reformulating the PDE first as a variational problem (the *Ritz method*). This leads to an algebraic system, sometimes through an intermediate stage of solving ODEs. The considerations underlying the choice of a basis for $\mathcal{H}_N$ lead to two major families of methods. Once we wish to have a sparse algebraic system, it is natural to choose basis functions with small overlapping supports, resulting in the *finite element method* (Brenner & Scott 2002). Alternatively, the goal of minimising the number of variables N of the algebraic system requires fast-convergent bases, giving raise to *spectral methods* (Trefethen 2000).

Other important approaches to the discretization of PDEs include *boundary methods*, when differential equations are replaced by integral equations along the boundary of the domain, thereby reducing the number of unknowns, *particle methods,* that restrict the inhomogeneous term of the PDE to a discrete number of 'particles', rendering the computational problem much easier, and *finite volume methods,* converting divergence terms on a grid into volume integrals.

The analysis of all these methods shares a number of general organising principles, although mathematical methodology and toolbox vary. Convergence is a necessary requirement: as the discretization becomes finer (e.g. grid spacing tends to zero or the dimension of $\mathcal{H}_N$ tends to infinity), we wish the numerical solution to approach the exact solution uniformly in compact domains. According to the *Lax Equivalence Theorem,* for time-evolving problems this is equivalent to consistency and stability, the latter referring to uniform well-posedness of the numerical scheme in compact time intervals once the discretization becomes increasingly finer (Iserles 2008, Richtmyer & Morton 1967).

Discretization methods reduce the task to the solution of (possibly nonlinear) algebraic systems and a major objective in their implementation is to solve such systems with great efficiency. This is assisted by the algorithms of numerical linear algebra, fine-tuned to problems originating in the approximation of PDEs (an important approach is multigrid, exploiting a hierarchy of nested grids) and by the availability of fast transforms, e.g. the FFT. An important technique often rendering the solution of such algebraic systems easier is *domain decomposition*.

Integral equations

Fredholm equations can be approximated by a variety of techniques (Atkinson 1997). Finite differences are a popular option: thus, for example,

$$\int_0^1 K(x,y)f(x)dx = g(y), \qquad y \in [0,1].$$

where $K \in C([0,1]^2)$, $g \in C[0,1]$ and f is the unknown, can be at its simplest approximated by the linear algebraic system

$$\frac{1}{N+1}\sum_{k=0}^{N} K(\tfrac{k}{N},\tfrac{m}{N})f_k = g(\tfrac{m}{N}), \qquad m = 0,1,\ldots,N,$$

where $f_k \approx f(k/N)$. An alternative is presented by *Galerkin methods,* whereby the solution is projected on a finite-dimensional subspace: like for PDEs, we have the alternative of subspaces leading to sparse linear systems, e.g. by using spline functions, or subspaces of rapidly convergent basis functions, similarly to spectral methods.

Another problem associated with Fredholm equations is the calculation of the spectrum of the underlying integral operator. The problem can be discretized by similar algorithms, except that the outcome is an algebraic eigenvalue problem, rather than a linear algebraic system.

Similar techniques can be applied to Volterra equations but they lead to initial-value problems, rather than algebraic equations, hence convergence and stability considerations similar to those pertaining to initial-value ODEs become important (Brunner 2004).

Computational dynamics

The main interest in dynamics is in the evolution of *flows*: continuous dynamical systems whose behaviour depends on some parameters. In numerical analysis flows are replaced by *maps*, discrete dynamical systems. In principle, this requires the same discretization techniques as in the case of ODEs, PDEs and integral equations, except that the range of interesting questions is different, centred upon the interplay between the value of the parameters and (typically, long-term) behaviour of the underlying system. In other words, as parameters vary and dynamical features of the system undergo change, we wish to recover them faithfully under discretization (Stuart & Humphries 1998).

Another central challenge in dynamical systems is the computation of a *bifurcation diagram*, namely the identification and classification of parameter values that correspond to qualitative changes of the underlying system (Seydel 1994).

Research into nonlinear dynamical systems depends in large measure on the availability of excellent numerical software, not least the AUTO package (Computational Mathematics and Visualization Laboratory (CMVL) 1996).

Optimization

The problem of unconstrained optimisation is to determine the (local or global) minimum of a continuous, multivariate objective function $f : \mathbf{R}^m \to \mathbf{R}$. In constrained optimisation we seek to minimise f subject to $\mathbf{x}$ residing in a closed set $\Omega \subset \mathbf{R}^m$.

Most modern algorithms for unconstrained optimisation are iterative (Fletcher 2001), in particular Newton-type methods, conjugate gradient methods and the Levenberg–Marquardt method. The underlying idea is to decrease the value of the objective function in each iteration until an optimum is reached. Special attention is afforded to problems with a very large number of variables and to structured objective functions and most algorithms require of f very low regularity conditions.

The most ubiquitous constrained optimisation problem is *linear programming*, whereby one minimises $\mathbf{f}(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$. The optimum resides at a vertex of the set of constraints, a convex polytope. The *simplex algorithm* allows to jump from a vertex to one of its neighbours while reducing the value of $\mathbf{f}$, hence it terminates at the minimum in a finite number of steps. Since its introduction by George Dantzig, the simplex algorithm has been instrumental in a wide range of practical computations. Lately, however, increasing prominence is afforded in constrained optimisation to *interior-point methods,* which approach the minimum while moving across a path inside a convex set of constraints.

Stochastic computations

Many mathematical phenomena in need of discretization possess stochastic character. Perhaps the most important are *stochastic differential equations* (SDEs), e.g.

$$d\mathbf{y} = \mathbf{f}(t, \mathbf{y})dt + \mathbf{g}(t, \mathbf{y})dW(t), \quad t \geq t_0, \qquad \mathbf{y}(t_0) = \mathbf{y}_0,$$

where W is a Wiener process. There are two general approaches to the discretization of SDEs. The first seeks to compute deterministic functionals like the expectation and variance of the solution and its probability density function: the latter is described by the (deterministic) *Fokker–Planck equation.* The second computes solution trajectories at grid points, the random process being modelled by a random number generator (itself a computational problem). This results in numerical schemes similar to the more familiar ODE and PDE time stepping methods, e.g. in place of the Euler method $\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \mathbf{f}(t_n, \mathbf{y}_n)$ for the ODE $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, we may use the *Euler–Maruyama method*

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{g}(t_n, \mathbf{y}_n)[W(t_{n+1}) - W(t_n)]$$

for the above SDE. However, this similarity is deceptive, since the design of effective SDE solvers requires different qualitative attributes.

Other numerical calculations with significant stochastic components are the computation of Markov chains and generation of random numbers. Moreover, the computation of deterministic problems can be often done more efficiently by introducing stochasticity, an important case in point being Monte Carlo methods for the computation of multivariate integrals.

Organising principles of numerical analysis

Computer arithmetic

Numerical calculations are usually performed using floating point numbers, mostly adhering to the IEEE 754-2008 standard of floating-point arithmetic (IEEE Standards Association 2008). Effective estimation of true error in computations must reckon with two sources: imprecisions due to discretization (*truncation errors*) and consequences of working, in place of reals, with floating-point numbers (*roundoff errors*).

The interplay between truncation and roundoff errors varies across numerical analysis, although it is fair to state that truncation errors are far more important in majority of situations. Roundoff errors might be perilous in 'static' algorithms, like Gaussian elimination, while truncation errors tend to dominate in self-correcting iterative algorithms and in the computation of differential equations. Having said so, it is important to bear in mind that the analysis of discretization errors and of convergence of iterative processes is insufficient for practical implementation of numerical algorithms, unless accompanied by valid reasons for their robustness with regard to roundoff errors (Wilkinson 1988).

Stability and conditioning

The phrase 'stability' covers a wide range of desirable, often necessary features of numerical algorithms. Informally, stability can assume one of two meanings: either a dynamical system changes in a bounded manner in compact time intervals in response to small changes in its initial value or other parameters (structural stability), or it exhibits bounded and 'nice' asymptotic behaviour (dynamical stability). It is always sound policy to verify the definition of stability in any specific setting because careless use of this concept might be misleading. Thus, in the context of numerical ODEs, zero-stability is structural, while A-stability is dynamical, while stability in the context of numerical PDEs is structural.

In general, stability is related to the robustness of numerical computation. If the algorithm is structurally stable, small departures from the exact solution, originating in either truncation or roundoff errors, are unlikely to cause breakdown. Likewise, once the algorithm is dynamically stable, such errors are unlikely to accumulate across large number of time steps or iterations.

All this assumes, however, that the problem being computed does not itself exhibit undue sensitivity to small perturbations. Otherwise, even the most stable computation might be unsafe. To rephrase, robust computation requires the confluence of a stable problem and a stable algorithm. Structural stability of a problem can be often quantified in terms of its condition number. For example, structural stability of the linear equation $A\mathbf{x} = \mathbf{b}$ is measured by the *condition number* $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ – in the Euclidean norm this is the ratio of the largest to the smallest singular value of A. The larger $\kappa(A)$, the more sensitive is the solution of the linear system to perturbations.

An important tool in identifying the sensitivity of computations to instability is *backward error analysis* (Wilkinson 1988): instead of asking "what is the error committed in a numerical calculation?", we ask "what is the problem solved exactly by our calculation and how far away is it from the original problem?". Backward error analysis is of major importance in linear algebra calculations but recently it became increasingly relevant to the discretization of time-dependent ODEs (Hairer, Lubich & Wanner 2006).

Divide and conquer

A popular strategy in the computation of large problems is to split them into a number of smaller problems, subsequently assembling, perhaps in an iterative manner, the solution of the original problem. This is often advantageous when the cost of an algorithm is superlinear in the number of variables, since then solving several smaller problems may cost less than solving one large problem. An important advantage of this approach is that, once smaller problems are independent of each other, they can be solved efficiently in a parallel manner.

An example is the divide-and-conquer strategy, popular in many computer-science algorithms, but also in numerical calculations. For example, computing the eigenvalues of a large matrix, it is often possible to devise an iterative, fast-convergent procedure, computing in each iteration the eigenvalues of smaller matrices. Likewise, once a PDE is solved in a large spatial domain, possibly with complicated geometry, it is possible, in a procedure known as *domain decomposition,* to solve the problem iteratively in subdomains (Chan & Mathew 1994).

While domain decomposition acts in spatial variables, *operator splitting* acts in time. Thus, for example, given the initial-value problem $\partial u/\partial t = \mathcal{L}_1[u] + \mathcal{L}_2[u]$, where $\mathcal{L}_k$ may be functions or differential operators, we can assemble approximate solution by solving the (often much easier) problems $\partial v_k/\partial t = \mathcal{L}_k[v_k]$, $k = 1, 2$ (McLachlan & Quispel 2002).

The computation of many problems can be nested using the *fast multipole algorithm* (Greengard & Rokhlin 1987). It is particularly effective for n-body problems for very large value of n, e.g. in electromagnetics, since it often reduces the cost of matrix/vector multiplication from $O(n^2)$ to $O(n)$.

Perhaps the one divide-and-conquer technique with greatest impact is the *Fast Fourier Transform* (FFT) for the computation of discrete Fourier transform of n variables in $O(n \log n)$ operations. This is one of the most important algorithms ever,

with long list of critical applications in engineering, computer science and numerical analysis itself (Henrici 1979).

Homotopy

Suppose that we wish to solve a 'difficult' problem P_1, say, while we can easily solve another problem P_0, of similar character. The *homotopy* (or continuation) methodology considers a path of problems P_t, $t \in [0, 1]$, all of the same kind as P_0 and P_1, which continuously deform P_0 into P_1. The idea then is to commence from $t = 0$ and advance in small steps along the path, the assumption being that, once we have determined $P_{m\Delta t}$, it is fairly easy to compute $P_{(m+1)\Delta t}$ (Allgower & Georg 2003, Computational Mathematics and Visualization Laboratory (CMVL) 1996).

For example, P_1 might be determining the eigenvalues of an $n \times n$ real symmetric matrix A, while the eigenvalues of an $n \times n$ symmetric matrix B corresponding to P_0 are known. In that case we may let P_t be the eigenvalue problem for $(1 - t)B + tA$. This approach lends itself to parallelism since we can advance in parallel along the n homotopy paths linking individual eigenvalues of B and A (Li 1997).

Multiscale

Numerous science and engineering models exhibit a range of processes operating at widely differing scales. Pertinent qualitative features of the model are often described in a fairly comprehensive manner by its slower-varying components. Unfortunately, many standard numerical methods require sufficiently fine resolution to cater for the fastest component – even when the amplitude of this component is, to all intents and purposes, negligible. This behaviour is pervasive in the solution of time-dependent differential equations and it requires great deal of care in the design and analysis of computational algorithms.

Structure preservation

Although mathematical analysis usually falls short of solving exactly complicated mathematical constructs, it often produces important information about their qualitative features, in particular about their *dynamics* (long-term behaviour) and *geometry* (integrals, symmetries and invariants). This information, which may reflect crucial physical or mathematical features of the underlying problem, is often lost under discretization. In the context of initial-value problems this motivates an approach, sometimes termed *geometric numerical integration*, whereby numerical algorithms are designed to preserve underlying structure (Hairer et al. 2006). For example, it might be known that the exact solution evolves on a smooth manifold $\mathcal{M}$, in which case the aim is to design a time-stepping algorithm that also evolves on $\mathcal{M}$. Likewise, the underlying system might be Hamiltonian, whereby one wishes to retain symplecticity under discretization, or divergence free, when the effort is in designing volume-preserving methods.

Ranging beyond geometry, increasing attention is being paid to the preservation of topological structure of differential equations under discretization, in particular by finite element methods. This results in more stable and accurate methods (Arnold, Falk & Winther 2010).

Retention of structure under discretization is often desirable, or even essential, in the modelling of the underlying physical phenomenon. Moreover, it often leads to more effective numerical methods – for example, symplectic methods for Hamiltonian systems are known to accumulate error slower (Hairer et al. 2006).

Complexity and cost

Complexity is a feature of the underlying problem, quantifying the effort required to solve it to given accuracy. Cost is a feature of a numerical algorithm, telling how close it approaches the complexity of the problem. Confusingly, "complexity analysis" usually refers to the analysis of both complexity and cost.

While the concept of complexity is fairly straightforward in the discrete setting of combinatorial problems and theoretical computer science, it is much more complicated, often ambiguous, in numerical analysis. Roughly, one can distinguish four distinct approaches to complexity analysis: (a) Mirroring the discrete concept of complexity. Thus, a yardstick measuring the number of operations (e.g. a "flop" – a shortcut for "floating point operation") is adopted and the performance of algorithms quantified accordingly. (b) Information-based complexity. The goal here is to understand how information, which is usually incomplete and often contaminated by error and noise, can be used to deduce complexity and cost (Traub, Woźniakowski & Wasilkowski 1988). This leads to a formal framework, employing tools of functional analysis. (c) The Blum–Shub–Smale model. Discrete complexity being based upon the *Turing machine,* its real-number alternative is the BSS machine. It is a formal, algorithmic construct and, at least in principle, a numerical algorithm is reducible to a set of instructions on the BSS machine (Blum, Cucker, Shub & Smale 1998). This model has had a number of genuine successes, not least in the computation of nonlinear algebraic systems. (d) Smoothed complexity analysis. Many numerical methods, e.g. the simplex algorithm and Gaussian elimination with partial pivoting, have high worst-case cost but perform very well in practice. The main idea of smoothed analysis is to provide an intermediate framework between worst-case and average-case scenarios and it has already led to enhanced understanding of many popular algorithms (Spielman & Teng 2001).

High performance computing

Much of large-scale contemporary computing combines numerical analysis algorithms with sophisticated computing architectures, typically displaying massive parallelism. These two activities are closely related, because what is good on a single processor might be suboptimal in a parallel setting. In addition, while single-processor computation is usually quantified by means of floating-point operations, in parallel architecture one must consider also communication costs – indeed, data passing among processors

might be often more expensive than computation itself. This changes the definition of what is a good algorithm and has fostered new computational approaches.

Applications of numerical analysis

The spread of numerical analysis mirrors the reach of mathematics across sciences, engineering and medicine. It is important to realise that scientific computing at its best is not just a matter for algorithmic dexterity and careful mathematical analysis. Addressing difficult problems in application areas requires a dialogue between different groups of experts and an incorporation of a wide range of ideas relevant to the problem being modelled. Typically there are two sources of inevitable error in numerical simulation: not just the error incurred by the algorithm, but the error already implicit in the many simplifications and imperfections in the model being solved. Although this is true in general, some application areas, because of their wide scope and the challenge implicit in their computational problems, have led to genuinely new disciplines, representing a synthesis between modelling and computation.

Computational engineering

Numerical simulation of the *Navier–Stokes equations* and their numerous simplifications (in particular, once viscosity terms are excised, the *Euler equations*) is central to computational fluid dynamics (CFD). Such equations are typically in three space dimensions, given in complicated geometries and their solutions might vary rapidly and exhibit a range of turbulent and transient phenomena. No wonder, thus, that successful CFD rests upon insight originating in fluid dynamics. Moreover, CFD has led to interest in a range of algorithms which come into their own in this setting, e.g. finite volume methods, vorticity methods, smoothed particle hydrodynamics and lattice Bolzmann methods.

The importance of CFD to contemporary science and engineering based upon fluid mechanics concepts, e.g. aerodynamics, weather forecasting and reservoir modelling, can be hardly overstated. Computer models increasingly replace experiment: modelling aircraft flight in a computer, instead of a wind tunnel, is not just considerably more affordable and faster but allows testing at a much broader range of parameters.

Engineering computations range beyond CFD. Solid mechanics is a rich source of challenging computational problems, in particular in the study of microstructures and cracks. Electrical and electronic engineering presents a raft of computationally demanding problems in circuit simulation and data transmission, many control engineering problems are reducible to computation and optimisation of trajectories, bioengineering increasingly rests upon the computational modelling of biofluids, tissues and entire organisms... Numerical computation is not simply one of the tools available to a modern engineer, it is at the very heart of what contemporary engineering is all about.

Computational physics

Computation plays a fundamental role in contemporary physics. Many computational problems in physics are not very different from core concerns of numerical analysis, in particular the discretization of differential equations. However, contemporary physics research leads to a number of new and important computational challenges. One example is the computation of spectra of Schrödinger operators and their distribution. Another is the interaction of large number of particles, e.g. in plasma physics or in molecular dynamics. Particle models incorporate a wide range of physical laws, from classical to quantum mechanics, and often have substantive stochastic component. Other major computational challenge in physics is calculations in lattice models, e.g. in gauge theory, quantum field theory and quantum chromodynamics.

Mathematics of information

Modern technological society produces increasing reams of information which needs collection, processing, transmission, classification and analysis. This is increasingly leading to new computational challenges, e.g. in image processing, signal processing, data mining, medical imaging, machine learning, computer vision, data compression and cryptography. Such problems often share a number of common structural features: they are concerned with a very large number of variables, incorporate noise and stochastic components, bring together discrete and continuous data and model data which is often intermediated by electromagnetic waves. This creates a common agenda to numerical computation with harmonic analysis, combinatorics, theoretical computer sciences and stochastic analysis.

An increasingly important organising principle in understanding very large data sets is *sparsity*. Although the size of data might be very large indeed, the information it contains is largely redundant and repetitive. Once the mathematical mechanism underlying this redundancy is understood, it is possible to collect significantly smaller amounts of data without impairing the information content, fill-in missing data and understand better their structure in terms of a small number of variables. This has led recently to new approaches to computation, e.g. compressed sensing, sparsity recovery and greedy algorithms.

Acknowledgements

This will be an item in the forthcoming *Encyclopaedia of Applied and Computational Mathematics*, to be published in 2013 by Springer Verlag under the editorship of Björn Engquist.

The author wishes to thank Douglas Arnold (Minneapolis), Nick Higham (Manchester), Robert McLachlan (Palmerston North), Jesús María Sanz-Serna (Valladolid), Gil Strang (MIT), Endre Süli (Oxford), Nick Trefethen (Oxford), Gerhard Wanner (Geneva) and Antonella Zanna (Bergen), who have read an early draft of this essay and commented so knowledgeably upon it. Whatever is wrong, misguided or incom-

plete about this essay is entirely the responsibility of the author, otherwise I owe an immense debt of gratitude to all the above for their insight and wisdom.

References

Allgower, G. L. & Georg, K. (2003), *Introduction to Numerical Continuation Methods*, SIAM, Philadelphia.

Arnold, D. N., Falk, R. S. & Winther, R. (2010), 'Finite element exterior calculus: from Hodge theory to numerical stability', *Bull. Amer. Math. Soc.* **47**, 281–3543.

Atkinson, K. E. (1997), *The Numerical Solution of Integral Equations of the Second Kind*, Cambridge University Press, Cambridge.

Björck, Å. (1996), *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia.

Blum, L., Cucker, F., Shub, M. & Smale, S. (1998), *Complexity and Real Computation*, Springer, New York.

Brenner, S. C. & Scott, L. R. (2002), *The Mathematical Theory of Finite Element Methods*, Springer, New York.

Brunner, H. (2004), *Collocation Methods for Volterra Integral and Related Functional Differential Equations*, Cambridge University Press, Cambridge.

Chan, T. F. & Mathew, T. P. (1994), 'Domain decomposition algorithms', *Acta Numerica* **3**, 61–143.

Cheney, E. W. & Light, W. A. (2000), *A Course in Approximation Theory*, American Mathematical Soc., Providence, RI.

Computational Mathematics and Visualization Laboratory (CMVL) (1996), 'AUTO software for continuation and bifurcation problems in Ordinary Differential Equations', http://cmvl.cs.concordia.ca/auto/.

Davis, P. J. & Rabinowitz, P. (1975), *Methods of Numerical Integration*, Academic Press, New York.

DeVore, R. A. & Lorentz, G. G. (1993), *Constructive Approximations*, Springer, Heidelberg.

Fletcher, R. (2001), *Practical Methods of Optimizations*, 2nd edn, Wiley-Interscience, London.

Golub, G. H. & Van Loan, C. F. (1996), *Matrix Computations*, 3rd edn, Johns Hopkins Press, Baltimore.

Greengard, L. & Rokhlin, V. (1987), 'A fast algorithm for particle simulations', *J. Comput. Phys.* **73**, 325–348.

Hairer, E. & Wanner, G. (1996), *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, 2nd edn, Springer, Berlin.

Hairer, E., Lubich, C. & Wanner, G. (2006), *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, 2nd edn, Springer, Berlin.

Hairer, E., Nørsett, S. P. & Wanner, G. (1993), *Solving Ordinary Differential Equations I. Nonstiff Problems*, 2nd edn, Springer, Berlin.

Henrici, P. (1979), 'Fast Fourier methods in computational complex analysis', *SIAM Rev.* **21**, 481–527.

IEEE Standards Association (2008), 'IEEE Standard for Floating-Point Arithmetic', http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4610933.

Iserles, A. (2008), *A First Course in the Numerical Analysis of Differential Equations*, 2nd edn, Cambridge University Press, Cambridge.

Li, T. Y. (1997), 'Numerical solution of multivariate polynomial systems by homotopy continuation methods', *Acta Numerica* **6**, 399–436.

McLachlan, R. I. & Quispel, G. R. W. (2002), 'Splitting methods', *Acta Numerica* **11**, 341–434.

Powell, M. J. D. (1981), *Approximation Theory and Methods*, Cambridge University Press, Cambridge.

Richtmyer, R. D. & Morton, K. W. (1967), *Difference Methods for Initial-Value Problems*, 2nd edn, Wiley-Interscience, New York.

Seydel, R. (1994), *Practical Bifurcation and Stability Analysis: From Equilibrium to Chaos*, Springer, Berlin.

Spielman, D. & Teng, S.-H. (2001), Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time, *in* 'Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing', pp. 296–3058.

Stuart, A. M. & Humphries, A. R. (1998), *Dynamical Systems and Numerical Analysis*, Cambridge University Press, Cambridge.

Traub, J. F., Woźniakowski, H. & Wasilkowski, G. W. (1988), *Information-Based Complexity*, Academic Press, New York.

Trefethen, L. N. (2000), *Spectral Methods in MATLAB*, SIAM, Philadelphia.

Wanner, G. (2010), 'Kepler, Newton and numerical analysis', *Acta Numerica* **19**, 561–598.

Wilkinson, J. H. (1988), *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford.