Contents

Keeping up to date

Addenda and corrigenda, comments, solutions to selected exercises, and ordering information can be found on the book’s web page:

`http://www-math.upb.de/mca/`

A Beggar's Book Out-worths a Noble's Blood.[1]
William Shakespeare (1613)

Some books are to be tasted, others to be swallowed,
and some few to be chewed and digested.
Francis Bacon (1597)

Les plus grands analystes eux-mêmes ont bien rarement dédaigné de se tenir à la portée de la classe *moyenne* des lecteurs; elle est en effet la plus nombreuse, et celle qui a le plus à profiter dans leurs écrits.[2]
Anonymous referee (1825)

It is true, we have already a great many Books of *Algebra*,
and one might even furnish a moderate Library
purely with Authors on that Subject.
Isaac Newton (1728)

فخر رت هذا الكتاب وجمعت فيه جميع ما يحتاج اليه الحاسب
محترزا عن اشباع ممل و اختصار مخل[3]
Ghiyāth al-Dīn Jamshīd bin Masʿūd bin Maḥmūd al-Kāshī (1427)

[1] The sources for the quotations are given on pages 715–719.

[2] The greatest analysts [mathematicians] themselves have rarely shied away from keeping within the reach of the *average* class of readers; this is in fact the most numerous one, and the one that stands to profit most from their writing.

[3] I wrote this book and compiled in it everything that is necessary for the computer, avoiding both boring verbosity and misleading brevity.

Introduction

In science and engineering, a successful attack on a problem will usually lead to some equations that have to be solved. There are many types of such equations: differential equations, linear or polynomial equations or inequalities, recurrences, equations in groups, tensor equations, etc. In principle, there are two ways of solving such equations: approximately or exactly. *Numerical analysis* is a well-developed field that provides highly successful mathematical methods and computer software to compute *approximate* solutions.

Computer algebra is a more recent area of computer science, where mathematical tools and computer software are developed for the *exact* solution of equations.

Why use approximate solutions at all if we can have exact solutions? The answer is that in many cases an exact solution is not possible. This may have various reasons: for certain (simple) ordinary differential equations, one can prove that no closed form solution (of a specified type) is possible. More important are questions of efficiency: any system of linear equations, say with rational coefficients, can be solved exactly, but for the huge linear systems that arise in meteorology, nuclear physics, geology or other areas of science, only approximate solutions can be computed efficiently. The exact methods, run on a supercomputer, would not yield answers within a few days or weeks (which is not really acceptable for weather prediction).

However, within its range of exact solvability, computer algebra usually provides more interesting answers than traditional numerical methods. Given a differential equation or a system of linear equations with a parameter t, the scientist gets much more information out of a closed form solution in terms of t than from several solutions for specific values of t.

Many of today's students may not know that the *slide rule* was an indispensable tool of engineers and scientists until the 1960s. *Electronic pocket calculators* made them obsolete within a short time. In the coming years, *computer algebra systems* will similarly replace calculators for many purposes. Although still bulky and expensive (hand-held computer algebra calculators are yet a novelty), these systems can easily perform exact (or arbitrary precision) arithmetic with numbers, matri-

ces, polynomials, etc. They will become an indispensable tool for the scientist and engineer, from students to the work place. These systems are now becoming integrated with other software, like numerical packages, CAD/CAM, and graphics.

The goal of this text is to give an introduction to the basic methods and techniques of computer algebra. Our focus is threefold:

- complete presentation of the mathematical underpinnings,
- asymptotic analysis of our algorithms, sometimes "Oh-free",
- development of asymptotically fast methods.

It is customary to give bounds on running times of algorithms (if any are given at all) in a "big-Oh" form (explained in Section 25.7), say as $O(n \log n)$ for the FFT. We often prove "Oh-free" bounds in the sense that we identify the numerical coefficient of the leading term, as $\frac{3}{2} n \log_2 n$ in the example; we may then add $O(\text{smaller terms})$. But we have not played out the game of minimizing these coefficients; the reader is encouraged to find smaller constants herself.

Many of these fast methods have been known for a quarter of a century, but their impact on computer algebra systems has been slight, partly due to an "unfortunate myth" (Bailey, Lee & Simon 1990) about their practical (ir)relevance. But their usefulness has been forcefully demonstrated in the last few years; we can now solve problems—for example, the factorization of polynomials—of a size that was unassailable a few years ago. We expect this success to expand into other areas of computer algebra, and indeed hope that this text may contribute to this development. The full treatment of these fast methods motivates the "modern" in its title. (Our title is a bit risqué, since even a "modern" text in a rapidly evolving discipline such as ours will obsolesce quickly.)

The basic objects of computer algebra are numbers and polynomials. Throughout the text, we stress the structural and algorithmic similarities between these two domains, and also where the similarities break down. We concentrate on polynomials, in particular univariate polynomials over a field, and pay special attention to finite fields.

We will consider arithmetic algorithms in some basic domains. The tasks that we will analyze include conversion between representations, addition, subtraction, multiplication, division, division with remainder, greatest common divisors, and factorization. The domains of fundamental importance for computer algebra are the natural numbers, the rational numbers, finite fields, and polynomial rings.

Our three goals, as stated above, are too ambitious to keep up throughout. In some chapters, we have to content ourselves with sketches of methods and outlooks on further results. Due to space limitations, we sometimes have recourse to the lamentable device of "leaving the proof to the reader". Don't worry, be happy: solutions to the corresponding exercises are available on the book's web site.

After writing most of the material, we found that we could structure the book into five parts, each named after a mathematician that made a pioneering contribution on which some (but, of course, not all) of the modern methods in the respective part rely. In each part, we also present selected applications of some of the algorithmic methods.

The first part EUCLID examines Euclid's algorithm for calculating the gcd, and presents the subresultant theory for polynomials. Applications are numerous: modular algorithms, continued fractions, Diophantine approximation, the Chinese Remainder Algorithm, secret sharing, and the decoding of BCH codes.

The second part NEWTON presents the basics of fast arithmetic: FFT-based multiplication, division with remainder and polynomial equation solving via Newton iteration, and fast methods for the Euclidean Algorithm and the solution of systems of linear equations. The FFT originated in signal processing, and we discuss one of its applications, image compression.

The third part GAUSS deals exclusively with polynomial problems. We start with univariate factorization over finite fields, and include the modern methods that make attacks on enormously large problems feasible. Then we discuss polynomials with rational coefficients. The two basic algorithmic ingredients are Hensel lifting and short vectors in lattices. The latter has found many applications, from breaking certain cryptosystems to Diophantine approximation.

The fourth part FERMAT is devoted to two integer problems that lie at the foundation of algorithmic number theory: primality testing and factorization. The most famous modern application of these classical topics is in public key cryptography.

The fifth part HILBERT treats three different topics which are somewhat more advanced than the rest of the text, and where we can only exhibit the foundations of a rich theory. The first area is Gröbner bases, a successful approach to deal with multivariate polynomials, in particular questions about common roots of several polynomials. The next topic is symbolic integration of rational and hyperexponential functions. The final subject is symbolic summation; we discuss polynomial and hypergeometric summation.

The text concludes with an appendix that presents some foundational material in the language we use throughout the book: The basics of groups, rings, and fields, linear algebra, probability theory, asymptotic O-notation, and complexity theory.

Each of the first three parts contains an implementation report on some of the algorithms presented in the text. As case studies, we use two special purpose packages for integer and polynomial arithmetic: NTL by Victor Shoup and BIPOLAR by the authors.

Most chapters end with some bibliographical and historical notes or supplementary remarks, and a variety of exercises. The latter are marked according to their difficulty: exercises with a * are somewhat more advanced, and the few marked with ** are more difficult or may require material not covered in the text.

Laborious (but not necessarily difficult) exercises are marked by a long arrow $\longrightarrow$. The book's web page `http://www-math.upb.de/mca/` provides some solutions.

This book presents foundations for the mathematical engine underlying any computer algebra system, and we give substantial coverage—often, but not always, up to the state of the art—for the material of the first three parts, dealing with Euclid's algorithm, fast arithmetic, and the factorization of polynomials. But we hasten to point out some unavoidable shortcomings. For one, we cannot cover completely even those areas that we discuss, and our treatment leaves out major interesting developments in the areas of computational linear algebra, sparse multivariate polynomials, combinatorics and computational number theory, quantifier elimination and solving polynomial equations, and differential and difference equations. Secondly, some important questions are left untouched at all; we only mention computational group theory, parallel computation, computing with transcendental functions, isolating real and complex roots of polynomials, and the combination of symbolic and numeric methods. Finally, a successful computer algebra system involves much more than just the mathematical engine: efficient data structures, a fast kernel and a large compiled or interpreted library, user interface, graphics capability, interoperability of software packages, clever marketing, etc. These issues are highly technology-dependent, and there is no single good solution for them.

The present book can be used as the textbook for a one-semester or a two-semester course in computer algebra. The basic arithmetic algorithms are discussed in Chapters 2 and 3, and Sections 4.1–4.4, 5.1–5.5, 8.1–8.2, 9.1–9.4, 14.1–14.6, and 15.1–15.2. In addition, a one-semester undergraduate course might be slanted towards computational number theory (9.5, 18.1–18.4, and parts of Chapter 20), geometry (21.1–21.6), or integration (4.5, 5.11, 6.2–6.4, and Chapter 22), supplemented by fun applications from 4.6–4.8, 5.6–5.9, 6.8, 9.6, Chapter 13, and Chapters 1 and 24. A two-semester course could teach the "basics" and 6.1–6.7, 10.1–10.2, 15.4–15.6, 16.1–16.5, 18.1–18.3, 19.1–19.2, 19.4, 19.5 or 19.6–19.7, and one or two of Chapters 21–23, maybe with some applications from Chapters 17, 20, and 24. A graduate course can be more eclectic. We once taught a course on "factorization", using parts of Chapters 14–16 and 19. Another possibility is a graduate course on "fast algorithms" based on Part II. For any of these suggestions, there is enough material so that an instructor will still have plenty of choice of which areas to skip. The logical dependencies between the chapters are given in Figure 1.

The prerequisite for such a course is linear algebra and a certain level of mathematical maturity; particularly useful is a basic familiarity with algebra and analysis of algorithms. However, to allow for the large variations in students' background, we have included an appendix that presents the necessary tools. For that material, the borderline between the boring and the overly demanding varies too much to get it right for everyone. If those notions and tools are unfamiliar, an instructor

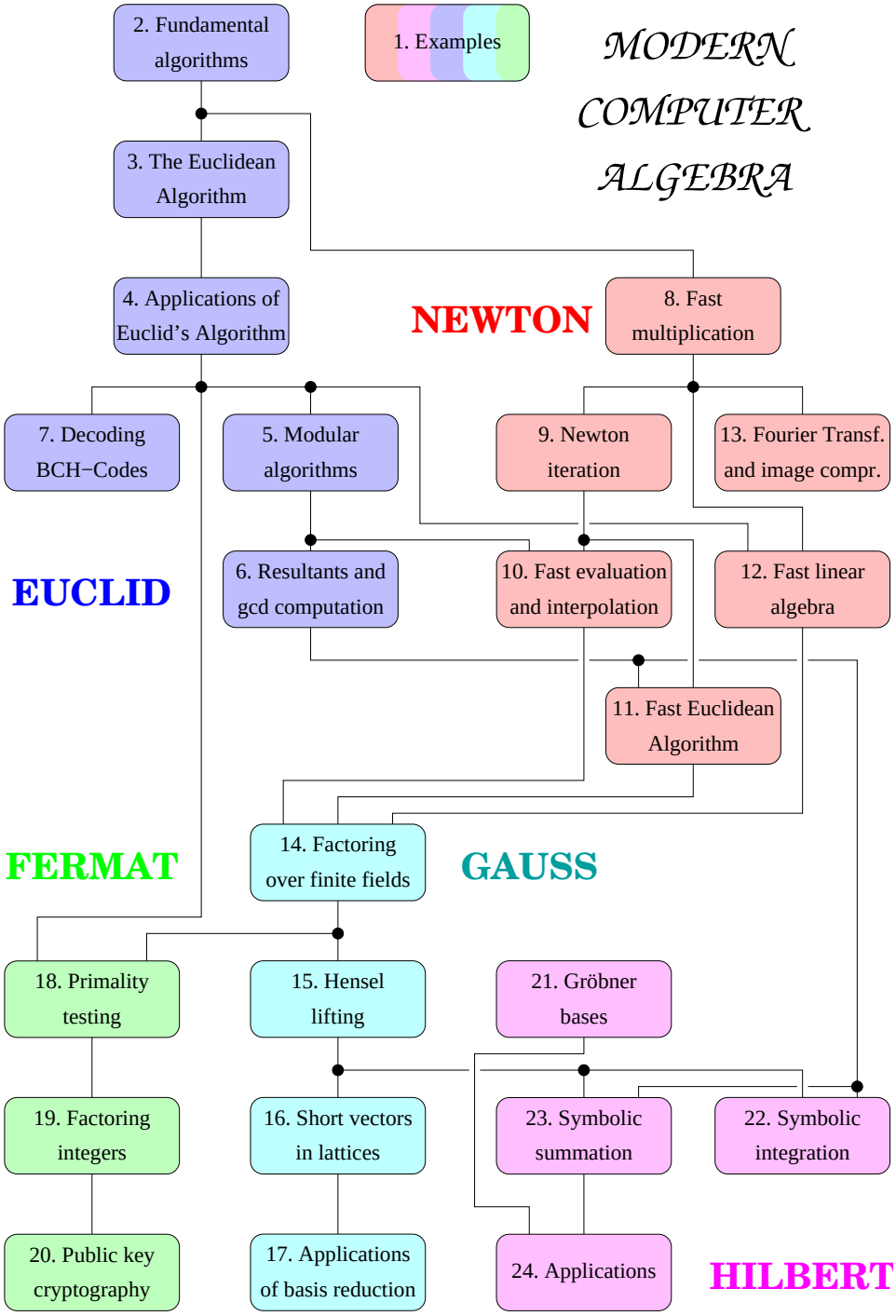

FIGURE 1: Leitfaden.

may have to expand beyond the condensed description in the appendix. Otherwise, most of the presentation is self-contained, and the exceptions are clearly indicated. By their nature, some of the applications assume a background in the relevant area.

The beginning of each part presents a biographical sketch of the scientist after which it is named, and throughout the text we indicate some of the origins of our material. For lack of space and competence, this is not done in a systematic way, let alone with the goal of completeness, but we do point to some early sources, often centuries old, and quote some of the original work. Interest in such historical issues is, of course, a matter of taste. It is satisfying to see how many algorithms are based on venerable methods; our essentially "modern" aspect is the concern with asymptotic complexity and running times, faster and faster algorithms, and their computer implementation.

Acknowledgements. This material has grown from undergraduate and graduate courses that the first author has taught over more than a decade in Toronto, Zürich, Santiago de Chile, Canberra, and Paderborn. He wants to thank above all his two teachers: Volker Strassen, who taught him mathematics, and Allan Borodin, who taught him computer science. To his friend Erich Kaltofen he is grateful for many enlightening discussions about computer algebra.

The second author wants to thank his two supervisors, Helmut Meyn and Volker Strehl, for many stimulating lectures in computer algebra.

The support and enthusiasm of two groups of people have made the courses a pleasure to teach. On the one hand, the colleagues, several of whom actually shared in the teaching: Leopoldo Bertossi, Allan Borodin, Steve Cook, Faith Fich, Shuhong Gao, John Lipson, Mike Luby, Charlie Rackoff, and Victor Shoup. On the other hand, lively groups of students took the courses, solved the exercises and tutored others about them, and some of them were the scribes for the course notes that formed the nucleus of this text. We thank particularly Paul Beame, Isabel de Correa, Wayne Eberly, Mark Giesbrecht, Rod Glover, Silke Hartlieb, Jim Hoover, Keju Ma, Jim McInnes, Pierre McKenzie, Sun Meng, Rob Morenz, Michael Nöcker, Daniel Panario, Michel Pilote, and François Pitt.

Thanks for help on various matters go to Eric Bach, Peter Blau, Wieb Bosma, Louis Bucciarelli, Désirée von zur Gathen, Keith Geddes, Dima Grigoryev, Johan Håstad, Dieter Herzog, Marek Karpinski, Wilfrid Keller, Les Klinger, Werner Krandick, Ton Levelt, János Makowsky, Ernst Mayr, François Morain, Gerry Myerson, Michael Nüsken, David Pengelley, Bill Pickering, Tomás Recio, Jeff Shallit, Igor Shparlinski, Irina Shparlinski, and Paul Zimmermann.

We thank Sandra Feisel, Carsten Keller, Thomas Lücking, Dirk Müller, and Olaf Müller for programming and the substantial task of producing the index, and Marianne Wehry for tireless help with the typing.

We are indebted to Sandra Feisel, Adalbert Kerber, Preda Mihăilescu, Michael

Nöcker, Daniel Panario, Peter Paule, Daniel Reischert, Victor Shoup, and Volker Strehl for carefully proofreading parts of the draft.

Paderborn, January 1999

The 2002 edition. The great French mathematician Pierre Fermat never published a thing in his lifetime. One of the reasons was that in his days, books and other publications often suffered vitriolic attacks for perceived errors, major or minor, frequently combined with personal slander.

Our readers are friendlier. They pointed out about 160 errors and possible improvements in the 1999 edition to us, but usually sugared their messages with sweet compliments. Thanks, friends, for helping us feel good and produce a better book now! We gratefully acknowledge the assistance of Sergeĭ Abramov, Michael Barnett, Andreas Beschorner, Peter Bürgisser, Michael Clausen, Rob Corless, Abhijit Das, Ruchira Datta, Wolfram Decker, Emrullah Durucan, Friedrich Eisenbrand, Ioannis Emiris, Torsten Fahle, Benno Fuchssteiner, Rod Glover, David Goldberg, Mitch Harris, Dieter Herzog, Andreas Hirn, Mark van Hoeij, Dirk Jung, Kyriakos Kalorkoti, Erich Kaltofen, Karl-Heinz Kiyek, Andrew Klapper, Don Knuth, Ilias Kotsireas, Werner Krandick, Daniel Lauer, Daniel Bruce Lloyd, Martin Lotz, Thomas Lücking, Heinz Lüneburg, Mantsika Matooane, Helmut Meyn, Eva Mierendorff, Daniel Müller, Olaf Müller, Seyed Hesameddin Najafi, Michael Nöcker, Michael Nüsken, Andreas Oesterhelt, Daniel Panario, Thilo Pruschke, Arnold Schönhage, Jeff Shallit, Hans Stetter, David Theiwes, Thomas Viehmann, Volker Weispfenning, Eugene Zima, and Paul Zimmermann.

Our thanks also go to Christopher Creutzig, Katja Daubert, Torsten Metzner, Eva Müller, Peter Serocka, and Marianne Wehry.

Besides correcting the known errors and (unintentionally) introducing new ones, we smoothed and updated various items, and made major changes in Chapters 3, 15, and 22. Separate errata pages for both editions will be kept on the book's website `http://www-math.upb.de/mca/`.

Dear readers, the hunt for errors is not over. Please keep on sending them to us at {`gathen,jngerhar`}`@upb.de`. And while hunting, enjoy the reading!

Paderborn, February 2002

Note. We produced the postscript files for this book with the invaluable help of the following software packages: Leslie Lamport's LaTeX, based on Don Knuth's TeX, Klaus Lagally's ArabTeX, Oren Patashnik's BibTeX, Pehong Chen's MakeIndex, Maple, MuPAD, Victor Shoup's NTL, Thomas Williams' and Colin Kelley's gnuplot, the Persistence of Vision Ray Tracer POV-Ray, and xfig.