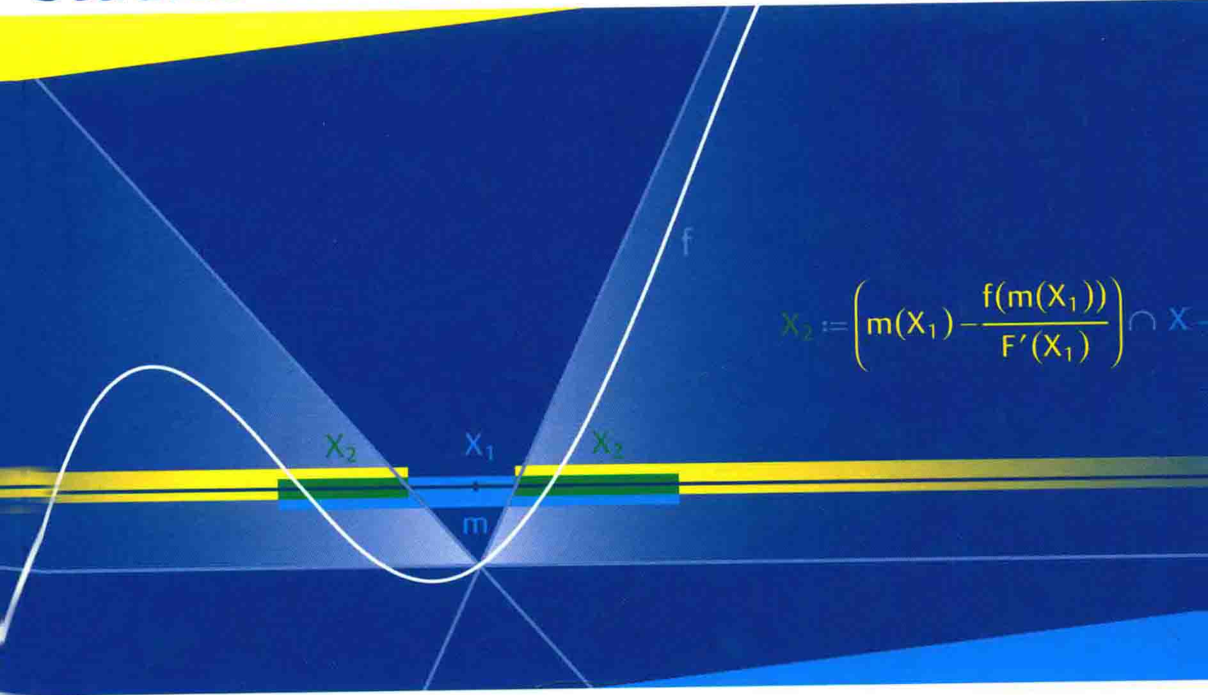
Ulrich Kulisch

# Computer Arithmetic and Validity

## Theory, Implementation, and Applications

Studies in Mathematics 33




de Gruyter

Ulrich Kulisch

# Computer Arithmetic and Validity

## Theory, Implementation, and Applications

Walter de Gruyter
Berlin · New York

*Author*

Ulrich Kulisch
Institute for Applied and Numerical Mathematics
Universität Karlsruhe
Englerstr. 2
76128 Karlsruhe
Germany
E-mail: ulrich.kulisch@math.uka.de

*Series Editors*

Carsten Carstensen
Department of Mathematics
Humboldt University of Berlin
Unter den Linden 6
10099 Berlin
Germany
E-Mail: cc@math.hu-berlin.de

Niels Jacob
Department of Mathematics
Swansea University
Singleton Park
Swansea SA2 8PP, Wales
United Kingdom
E-Mail: n.jacob@swansea.ac.uk

*Mathematics Subject Classification 2000:* 65-04, 65Gxx

*Keywords:* Computer arithmetic, interval arithmetic, floating-point arithmetic, verified computing, interval Newton method

∞ Printed on acid-free paper which falls within the guidelines of the ANSI to ensure permanence and durability.

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

ISBN 978-3-11-020318-9



Printed in Germany.
Cover design: Martin Zech, Bremen.
Typeset using the author's LaTeX files: Kay Dimler, Müncheberg.
Printing and binding: Hubert & Co. GmbH & Co. KG, Göttingen.

de Gruyter Studies in Mathematics 33

Editors: Carsten Carstensen · Niels Jacob

# de Gruyter Studies in Mathematics

1 Riemannian Geometry, 2nd rev. ed., *Wilhelm P. A. Klingenberg*
2 Semimartingales, *Michel Métivier*
3 Holomorphic Functions of Several Variables, *Ludger Kaup and Burchard Kaup*
4 Spaces of Measures, *Corneliu Constantinescu*
5 Knots, 2nd rev. and ext. ed., *Gerhard Burde and Heiner Zieschang*
6 Ergodic Theorems, *Ulrich Krengel*
7 Mathematical Theory of Statistics, *Helmut Strasser*
8 Transformation Groups, *Tammo tom Dieck*
9 Gibbs Measures and Phase Transitions, *Hans-Otto Georgii*
10 Analyticity in Infinite Dimensional Spaces, *Michel Hervé*
11 Elementary Geometry in Hyperbolic Space, *Werner Fenchel*
12 Transcendental Numbers, *Andrei B. Shidlovskii*
13 Ordinary Differential Equations, *Herbert Amann*
14 Dirichlet Forms and Analysis on Wiener Space, *Nicolas Bouleau and Francis Hirsch*
15 Nevanlinna Theory and Complex Differential Equations, *Ilpo Laine*
16 Rational Iteration, *Norbert Steinmetz*
17 Korovkin-type Approximation Theory and its Applications, *Francesco Altomare and Michele Campiti*
18 Quantum Invariants of Knots and 3-Manifolds, *Vladimir G. Turaev*
19 Dirichlet Forms and Symmetric Markov Processes, *Masatoshi Fukushima, Yoichi Oshima and Masayoshi Takeda*
20 Harmonic Analysis of Probability Measures on Hypergroups, *Walter R. Bloom and Herbert Heyer*
21 Potential Theory on Infinite-Dimensional Abelian Groups, *Alexander Bendikov*
22 Methods of Noncommutative Analysis, *Vladimir E. Nazaikinskii, Victor E. Shatalov and Boris Yu. Sternin*
23 Probability Theory, *Heinz Bauer*
24 Variational Methods for Potential Operator Equations, *Jan Chabrowski*
25 The Structure of Compact Groups, 2nd rev. and aug. ed., *Karl H. Hofmann and Sidney A. Morris*
26 Measure and Integration Theory, *Heinz Bauer*
27 Stochastic Finance, 2nd rev. and ext. ed., *Hans Föllmer and Alexander Schied*
28 Painlevé Differential Equations in the Complex Plane, *Valerii I. Gromak, Ilpo Laine and Shun Shimomura*
29 Discontinuous Groups of Isometries in the Hyperbolic Plane, *Werner Fenchel and Jakob Nielsen*
30 The Reidemeister Torsion of 3-Manifolds, *Liviu I. Nicolaescu*
31 Elliptic Curves, *Susanne Schmitt and Horst G. Zimmer*
32 Circle-valued Morse Theory, *Andrei V. Pajitnov*

*This book is dedicated to my wife Ursula*

*and to my family,*
*to Brigitte and Joachim,*
*Johanna and Benedikt,*
*to Angelika and Rolf,*
*Florian and Niclas,*

*to*
*all former and present colleagues of my institute,*
*and to my students.*

Lasset uns am Alten,
so es gut ist, halten
und dann auf dem alten Grund
Neues schaffen Stund um Stund.

*House inscription in the Black Forest.*

# Preface

This book deals with computer arithmetic in a more general sense than usual, and shows how the arithmetic and mathematical capability of the digital computer can be enhanced in a quite natural way. The work is motivated by the desire and the need to improve the accuracy of numerical computing and to control the quality of the computed result.

As a first step towards achieving this goal, the accuracy requirements for the elementary floating-point operations as defined by the IEEE arithmetic standard [644], for instance, are extended to the customary product spaces of computation: the complex numbers, the real and complex intervals, the real and complex vectors and matrices, and the real and complex interval vectors and interval matrices. *All computer approximations of arithmetic operations in these spaces should ideally deliver a result that differs from the correct result by at most one rounding.* For all these product spaces this accuracy requirement leads to operations which are distinctly different from those traditionally available on computers. This expanded set of arithmetic operations is taken as a definition of what is called *basic computer arithmetic*.

Central to this treatise is the concept of semimorphism. It provides a mapping principle between the mathematical product spaces and their digitally representable subsets. The properties of a semimorphism are designed to preserve as many of the ordinary mathematical laws as possible. All computer operations of basic computer arithmetic are defined by semimorphism.

The book has three antecedents:

(I) Kulisch, U. W., *Grundlagen des numerischen Rechnens – Mathematische Begründung der Rechnerarithmetik*, Bibliographisches Institut, Mannheim, Wien, Zürich, 1976, 467 pp., ISBN 3-411-015617-9.

(II) Kulisch, U. W. and Miranker W. L., *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981, 249 pp., ISBN 0-12-428650-X.

(III) Kulisch, U. W., *Advanced Arithmetic for the Digital Computer – Design of Arithmetic Units*, Springer-Verlag, Wien, New York, 2002, 139 pp., ISBN 3-211-83870-8.

The need to define all computer approximations of arithmetic operations by semimorphism goes back to the first of these books. By the time the second book had been written, early microprocessors were on the market. They were made with a few thousand transistors, and ran at 1 or 2 MHz. Arithmetic was provided by an 8-bit adder. Floating-point arithmetic could only be implemented in software. In 1985 the IEEE binary floating-point arithmetic standard was internationally adopted. Floating-point arithmetic became hardware supported on microprocessors, first by coprocessors and later directly within the CPU. Of course, all operations of basic computer arithmetic can be simulated using elementary floating-point arithmetic. This, however, is rather complicated and results in unnecessarily slow performance. A consequence of this is that for large problems the high quality operations of basic computer arithmetic are hardly ever applied. Higher precision arithmetic suffers from the same problem if it is simulated by software.

Dramatic advances in speed and memory size of computers have been made since 1985. Today a computer chip holds more than one billion transistors and runs at 3 GHz or more. Results of floating-point operations can be delivered in every cycle. Arithmetic speed has gone from megaflops to gigaflops, to teraflops, and to petaflops. This is not just a gain in speed. A qualitative difference goes with it. If the numbers a petaflops computer produces in one hour were to be printed (500 on one page, 1000 on one sheet, 1000 sheets 10 cm high) they would form a pile that reaches from the earth to the sun and back. With increasing speed, problems that are dealt with become larger and larger. Extending the word size cannot keep up with the tremendous increase in computer speed. Computing that is continually and greatly speeded up calls conventional computing into question. Even with quadruple and extended precision arithmetic the computer remains an experimental tool. The capability of a computer should not just be judged by the number of operations it can perform in a certain amount of time without asking whether the computed result is correct. It should also be asked how fast a computer can compute correctly to 3, 5, 10 or 15 decimal places. If the question were asked that way, it would very soon lead to better computers. Mathematical methods that give an answer to this question are available. Computers, however, are not built in a way that allows these methods to be used effectively.

Computer arithmetic must move strongly towards more reliability in computation. Instead of the computer being merely a fast calculating tool it must be developed into a scientific instrument of mathematics. Two simple steps in this direction would have great effect. They are both simple and practical:

I. fast hardware support for (extended[1]) interval arithmetic and

II. a fast and exact multiply and accumulate operation or, what is equivalent to it, an exact scalar product.

These two steps together with *basic computer arithmetic* comprise what is here called *advanced computer arithmetic*. Fast hardware circutries for I. and II. are developed in Chapters 7 and 8, respectively. This additional computational capability is gained at very modest hardware cost. Besides being more accurate the new computer operations greatly speed up computation. I. and II., of course, can be used to execute and speed up the operations of basic computer arithmetic. This would boost both the speed of a computation and the accuracy of its result.

[1] including division by an interval that includes zero

Advanced computer arithmetic opens the door to very many additional applications. All these applications are extremely fast. I. and II. in particular are basic ingredients of what is called *validated numerics* or *verified computing*.

This book has three parts. Part 1, of four chapters, deals with the theory of computer arithmetic, while Part 2, also of four chapters, treats the implementation of arithmetic on computers. Part 3, of one chapter, illustrates by a few sample applications how advanced computer arithmetic can be used to compute highly accurate and mathematically verified results.

**Part 1:** The implementation of semimorphic operations on computers requires the establishment of various isomorphisms between different definitions of arithmetic operations on the computer. These isomorphisms are to be established in the mathematical spaces in which the actual computer operations operate. This requires a careful study of the structure of these spaces. Their properties are defined as invariants with respect to semimorphisms. These concepts are developed in Part 1 of the book. Part 1 is organized along the lines of its second antecedent. However it differs in many details from the earlier one, details that spring from advances in computer technology, and many derivations and proofs have been reorganized and simplified.

**Part 2:** In Part 2 of the book, basic ideas for the implementation of advanced computer arithmetic are discussed under the assumption that the data are floating-point numbers. Algorithms and circuits are developed which realize the semimorphic operations in the various spaces mentioned above. The result is an arithmetic with many desirable properties, such as high speed, optimal accuracy, theoretical describability, closedness of the theory, and ease of use.

Chapters 5 and 6 consider the implementation of *elementary floating-point arithmetic* on the computer for a large class of roundings. A particular section of Chapter 5 comments on the IEEE floating-point arithmetic standard. The final section of Chapter 6 contains a brief discussion of all arithmetic operations defined in the product sets mentioned above as well as between these sets. The objective here is to summarize the definition of these operations and to point out that they all can be performed as soon as an *exact scalar product* is available in addition to the operations that have been discussed in Chapters 5 and 6.

Floating-point operations with directed roundings are basic ingredients of interval arithmetic. But with their isolated use in software interval arithmetic is too slow to be widely accepted in the scientific computing community. Chapter 7 shows, in particular, that with very simple circuitry interval arithmetic can be made practically as fast as elementary floating-point arithmetic. To enable high speed, the case selections for interval multiplication (9 cases) and division (14 cases including division by an interval that includes zero) are done in hardware where they can be chosen without any time penalty. The lower bound of the result is computed with rounding downwards and the upper bound with rounding upwards by parallel units simultaneously. The rounding mode needs to be an integral part of the arithmetic operation. Also the basic comparisons for intervals together with the corresponding lattice operations and the result selection in more complicated cases of multiplication and division are done in hardware. There they are executed by parallel units simultaneously. The circuits described in this chapter show that with modest additional hardware costs interval arithmetic can be made almost as fast as simple floating-point arithmetic. Such high speed cannot be obtained just by running many elementary floating-point arithmetic processors in parallel.

A basic requirement of basic computer arithmetic is that all computer approximations of arithmetic in the usual product spaces should deliver a result that differs from the correct result by at most one rounding. This requires scalar products of floating-point vectors to be computed with but a single rounding. The question of how a scalar product with a single rounding can be computed just using elementary floating-point arithmetic has been carefully studied in the literature. A good summary and what is probably the fastest solution is given in [456] and [531]. However, we do not follow this line here. No software simulation can compete with a simple and direct hardware solution.

The most natural way to accumulate numbers is fixed-point accumulation. It is simple, error free and fast. In Chapter 8 circuitry for *exact* computation of the scalar product of two floating-point vectors is developed for different kinds of computers. To make the new capability conveniently available to the user a new data format called *complete* is used together with a few simple arithmetic operations associated with each floating-point format. *Complete arithmetic* computes all scalar products of floating-point vectors exactly. The result of complete arithmetic is always exact; it is complete, not truncated. Not a single bit is lost. A variable of type complete is a fixed-point word wide enough to allow exact accumulation (continued summation) of floating-point numbers and of simple products of such numbers.

If register space for the complete format is available complete arithmetic is very very fast. The arithmetic needed to perform complete arithmetic is not much different from what is available in a conventional CPU. In the case of the IEEE double precision format a *complete register* consists of about $1/2$ K bytes. Straightforward pipelining leads to very fast and simple circuits. The process is at least as fast as any conventional way of accumulating the products including the so-called partial sum technique on existing vector processors which alters the sequence of the summands and causes errors beyond the usual floating-point errors.

Complete arithmetic opens a large field of new applications. An exact scalar product rounded into a floating-point number or a floating-point interval serves as building block for semimorphic operations in the product spaces mentioned above. Fast multiple precision floating-point and multiple precision interval arithmetic are other important applications. All these applications are very very fast. Complete arithmetic is an instrumental addition to floating-point arithmetic. In many instances it allows recovery of information that has been lost during a preceding pure floating-point computation.

Because of the many applications of the hardware support for interval arithmetic developed in Chapter 7, and of the exact scalar product developed in Chapter 8, these two modules of advanced computer arithmetic emerge as its central components.

*Fast hardware support for all operations of* advanced computer arithmetic *is a fundamental and overdue extension of elementary floating-point arithmetic. Arithmetic operations which can be performed correctly with very high speed and at low cost should never just be done approximately or simulated by slow software. The minor additional hardware cost allows their realization on every CPU. The arithmetic operations of* advanced computer arithmetic *transform the computer from a fast calculating tool into a mathematical instrument.*

**Part 3:** Mathematical analysis has provided algorithms that deliver highly accurate and completely verified results. Part 3 of the book goes over some examples. Such algorithms are not widely used in the scientific computing community because they are very slow when the underlying arithmetic has to be carried out on conventional processors.

The first section describes some basic properties of interval mathematics and shows how these can be used to compute the range of a function's values. Used with automatic differentiation, these techniques lead to powerful and rigorous methods for global optimization. The following section then deals with differentiation arithmetic or automatic differentiation. Values or enclosures of derivatives are computed directly from numbers or intervals, avoiding the use of a formal expression for the derivative of the function. Evaluation of a function for an interval $X$ delivers a superset of the function's values over $X$. This overestimation tends to zero with the width of the interval $X$. Thus for small intervals interval evaluation of a function practically delivers the range of the functions's values. Many numerical methods proceed in small steps. So this property together with differentiation arithmetic to compute enclosures of derivatives is the key technique for validated numerical computation of integrals and for solution of differential equations, and for many other applications.

Newton's method is considered in two sections of Chapter 9. It attains its ultimate elegance and power in the *extended interval Newton method*, which is globally convergent and computes all zeros of a function in a given domain. The key to achieving these fascinating properties is division by an interval that includes zero.

The basic ideas needed for verified solution of systems of linear equations are developed in Section 9.5. Highly accurate bounds for a solution can be computed in a way that proves the existence and uniqueness of the solution within these bounds. Mathematical fixed-point theorems, interval arithmetic combined with defect correction or iterative refinement techniques using complete arithmetic are basic tools for achieving these results.

In Section 9.6 a method is developed that allows highly accurate and guaranteed evaluation of polynomials and of other arithmetic expressions.

Section 9.7 finally shows how fast multiple precision arithmetic and multiple precision interval arithmetic can be provided using complete arithmetic and other tools developed in the book.

Of course, the computer may often have to work harder to produce verified results, but the mathematical certainty makes it worthwhile. After all, the step from assembler to higher programming languages or the use of convenient operating systems also consumes a lot of computing power and nobody complains about it since it greatly enlarges the safety and reliability of the computation.

Computing is being continually and greatly speeded up. An avalanche of numbers is produced by a teraflops or petaflops computer (1 teraflops corresponds to $10^{12}$ floating-point operations per second). Fast computers are often used for safety critical applications. Severe, expensive, and tragic accidents can occur if the eigenfrequencies of a large electricity generator, for instance, are erroneously computed, or if a nuclear explosion is incorrectly simulated. Floating-point operations are inherently inexact. It is this inexactness at very high speed that calls conventional computing, just using naïve floating-point arithmetic, into question.

This book can, of course, be used as a textbook for lectures on the subject of computer arithmetic. If one is interested only in the more practical aspects of implementing arithmetic on computers, Part 2, with acceptance *a priori* of some results of Part 1, is also suitable as a basis for lectures. Part 3 can be used as an introduction to verified computing.

The second previous book was jointly written with Willard L. Miranker. On this occasion Miranker was very busy with other studies and could not take part, so this new book has been compiled solely by the other author and he takes full responsibility for its text. However, there are contributions and formulations here which go back to Miranker without being explicitly marked as such. I deeply thank Willard for his collaboration on the earlier book as well as on other topics, and for a long friendship. Contact with him was always very inspiring for me and for my Institute.

I would like to thank all former collaborators at my Institute. Many of them have contributed to the contents of this book, have realized advanced computer arithmetic in software on different platforms and in hardware in different technologies, have embedded advanced computer arithmetic into programming languages and implemented corresponding compilers, developed problem solving routines for standard problems of numerical analysis, or applied the new arithmetic to critical problems in the sciences. Among these colleagues are: Christian Ullrich, Edgar Kaucher, Rudi Klatte, Gerd Bohlender, Dalcidio M. Claudio, Kurt Grüner, Jürgen Wolff von Gudenberg, Reinhard Kirchner, Michael Neaga, Siegfried M. Rump, Harald Böhm, Thomas Teufel, Klaus Braune, Walter Krämer, Frithjof Blomquist, Michael Metzger, Günter Schumacher, Rainer Kelch, Wolfram Klein, Wolfgang V. Walter, Hans-Christoph Fischer, Rudolf Lohner, Andreas Knöfel, Lutz Schmidt, Christian Lawo, Alexander Davidenkoff, Dietmar Ratz, Rolf Hammer, Dimitri Shiriaev, Manfred Schlett, Matthias Hocks, Peter Schramm, Ulrike Storck, Christian Baumhof, Andreas Wiethoff, Peter Januschke, Chin Yun Chen, Axel Facius, Stefan Dietrich, and Norbert Bierlox. For their contributions I refer to the bibliograhy.

I owe particular thanks to Axel Facius, to Gerd Bohlender, and to Klaus Braune. Axel Facius keyed in and laid out the entire manuscript in LaTeX and he did most of the drawings. Drawings were also done by Gerd Bohlender. Klaus Braune helped to prepare the final version of the text. I thank Bo Einarsson for proofreading the book. I also thank my colleagues at the institute Götz Alefeld and Willy Dörfler for their support of the book project.

I gratefully acknowledge the help of Neville Holmes who went carefully through great parts of the manuscript, sending back corrections and suggestions that led to many improvements. His help was indeed vital for the completion of the book.

The Karlsruher Universitätsgesellschaft supported typing the first draft of the manuscript.

Karlsruhe, November 2007 Ulrich W. Kulisch

# Contents