Computer Arithmetic and Numerical Techniques

Annie CUYT and Brigitte VERDONK

I. Scientific Computing for the CS-student

The course "Computer Arithmetic and Numerical Techniques" at the University of Antwerp, is an introductory scientific computing course for students with a major in computer science or a combined major mathematics/computer science. But it could as well be taught to any exact or applied science student with a reasonable high-school background in mathematics and a fair interest in computers. It is this growing group of science students interested in computer science that we are trying to encourage into scientific computing. When the idea to set up a special course for computer science students was first launched in 1993 — it is by the way often hard to get computer science students interested in scientific computing — many colleagues world-wide reacted very enthusiastically.

Unlike engineering students, CS-students are rarely confronted with real-life scientific computing problems in other courses. Yet we believe that the best approach to a scientific computing course is one where the complete journey is followed from physical problem to computational solution, as described in [1]. The basic steps in this journey are:

1. a motivating problem
2. the identification of the computational problem behind the given real-life problem
3. the selection of an appropriate numerical technique developed for its solution
4. the actual implementation or use of a numerical routine, be it in Fortran, C, Mathematica, Matlab or the like
5. the evaluation or quality control of the numerical output.

The course is structured in such a way that for each topic all of steps (1) through (5) are covered, with special attention to the computer science background of the students. On one hand, special care is taken when choosing motivating examples so that they are not too technical, yet reasonably real-life. On the other hand, all the aspects of step (4) are treated in great detail. Whereas other scientific computing courses exist, this course is special in that it makes very explicit that one should distinguish between the properties of a mathematical algorithm and the properties of the algorithm's implementation in finite precision arithmetic on a computer. Computer science students are interested in computer arithmetic as part of scientific computing, in the same way they are interested in learning about compilers in order to obtain a full understanding of programming languages.

Taking these considerations into account, the course "Computer Arithmetic and Numerical Techniques" consists of two parts, one discussing how computations are performed on a binary machine as compared to mathematical computations in the set of real numbers (15-hour course load), and one on different numerical techniques (30-hour course load). These two parts are now described in detail. At the end of each part, we also discuss how the students get hands-on experience.

II. Computer Arithmetic

A. Theory

The essence of this part of the course is to discuss how the number sets $\mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ can be represented on a computer and to show how computing with the computer representation of these numbers instead of with the numbers themselves influences the computations. Throughout this part of the course, the level of complexity evolves in two directions. On one hand, the complexity evolves as we go from $\mathbb{Z}$ (integer arithmetic) to $\mathbb{Q}$ (exact rational arithmetic) to $\mathbb{R}$ (exactly rounded arithmetic). On the other hand, the complexity increases as we allow more complex operations on each of these number sets.

Section 1 covers computer representations of the mathematical number sets $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$: the set of machine integers $\mathbb{Z}_t$, where t indicates the number of bits provided to represent the integer (including its sign), the set $\mathbb{Q}_M$ of rationals that can be represented in finite machine memory, and the set of floating-point numbers $\mathbb{F}(\beta, t, L, U)$, with base β, precision t, and exponent range $[L, U]$. For short, when no confusion is possible, we shall often write $\mathbb{F}_t$ instead of $\mathbb{F}(\beta, t, L, U)$. Several considerations have to be taken into account when defining these sets of computer numbers: the choice of the base β, the base conversions incurred on input and output of data, the amount of bits provided for the exponent versus the significand in floating-point numbers of predetermined size, the rounding error involved in going from either $\mathbb{Z}, \mathbb{Q}$ or $\mathbb{R}$ to $\mathbb{F}_t$ etc.

In the next sections, operations on these computer number sets are discussed in increasing order of complexity. Section 2 discusses the elementary operations $+, -, \times, /$ and the essential difference between these operations in $\mathbb{Z}_t, \mathbb{Q}_M$ on one hand and $\mathbb{F}_t$ on the other hand, due to the absence, respectively effect of rounding errors. In Section 3 relational operators are discussed, emphasizing that programmers should hardly ever try to test whether two floating-point values are exactly equal to each other. Section 4 discusses in short the problem of implementing the elementary functions. Taylor series expansions bring out the concept of truncation error in a very natural way.

A next level of complexity arises in Section 5 when, instead of a single operation, compound statements are programmed. Compound statements involve such problems as the accumulation of rounding errors and the choice of an evaluation strategy (widest format available, widest needed precision, ...), especially when operands of different precisions are mixed. The effect of the evaluation strategy is clearly illustrated by running the same numeric code on different hardware platforms, for example SUN-Sparc versus Intel-PC.

With the functionality described in the previous sections, all ingredients are there to implement complete numerical algorithms in $\mathbb{F}_t$. The build-up of rounding and data errors in the implementation of algorithms leads to the essential concepts of forward and backward error analysis, numerical stability and ill-conditioning. These are discussed and illustrated in great detail in Section 6.

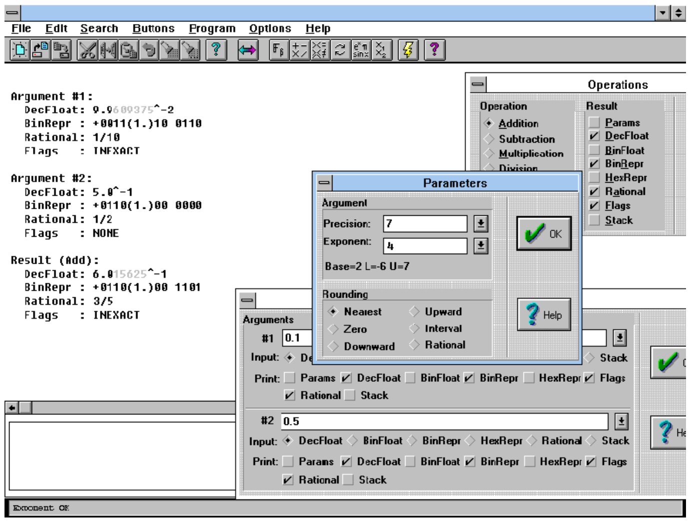
To top off the build-up in the previous sections, Section 7 discusses the IEEE standard [2], [3] for floating-point arithmetic. This standard embodies all of the details encountered when effectively implementing floating-point arithmetic on a binary machine. Several important but very detailistic concepts such as denormals, special representations, exception flags and so on come about.

For users not satisfied with IEEE floating-point arithmetic, the next few sections present and discuss alternatives. A first alternative is multi-precision floating-point arithmetic, discussed in Section 8. Several software implementations of multi-precision floating-point arithmetic exist, some of which are based on the vector (or doubled-precision) approach, while others are based on increasing the precision t and the base β. Another alternative for floating-point arithmetic is interval arithmetic. This is discussed in some detail in Section 9. The last section covers rational arithmetic or computing in $\mathbb{Q}_M$, as an alternative to computing in $\mathbb{F}_t$. For the implementation of rational arithmetic other building blocks are required than for the implementation of floating-point arithmetic: computing the Greatest Common Divisor, rational rounding (with lowest complexity) from $\mathbb{R}$ to $\mathbb{Q}_M$, etc. This section, which concludes the computer arithmetic part of the course, is at the same time a good starting-point for extra material on polynomial and symbolic computation.

B. Hands-on experience

The material covered in the "Computer Arithmetic" part of the course is essentially simple and clear-cut. Yet students have difficulties grasping all the intricacies of the material and understanding the interaction of the different types of errors that can occur. It is clear that concepts such as data and rounding error, overflow and underflow, catastrophic and benign cancellation, numerical instability, ill-conditioning, and so on should best not only be studied from a theoretical point of view. To make them as tangible as possible, we have developed what we call an "arithmetic explorer".

In this software environment it is possible to specify and simulate that computations are performed, not in the hardware IEEE singles or doubles, but in a user-defined set of floating-point numbers. By choosing a small precision t and a limited exponent range, students can then easily follow the computations at the bit-level. Moreover, in a low-precision floating-point set, one can easily zoom in on the unmistakable effects of data and rounding error, cancellation, ill-conditioning and numerical instability and develop a better feeling for different computer arithmetic issues. In this respect it is important to mention that the floating-point arithmetic, as implemented in the didactical tool, fully complies with the philosophy of the IEEE standard. Except for the fact that the precision and exponent range can be specified freely, all aspects of the IEEE standard are implemented and can be visualized, including exact rounding, denormals, signed zero, infinities, not-a-numbers and exception flags to name just a few. Hence it is possible for students to really "discover" the full details of floating-point arithmetic, something which is not so obvious when other tools are used. While a similar analysis can be done by using Mathematica or for instance by direct programming and using a traditional compiler, the result is often a time-consuming and confusing task, obscuring more points than one is trying to make.



```
Argument #1:
  DecFloat: 9.9609375^-2
  BinRepr : +0011(1.)10 0110
  Rational: 1/10
  Flags   : INEXACT

Argument #2:
  DecFloat: 5.0^-1
  BinRepr : +0110(1.)00 0000
  Rational: 1/2
  Flags   : NONE

Result (Add):
  DecFloat: 6.015625^-1
  BinRepr : +0110(1.)00 1101
  Rational: 3/5
  Flags   : INEXACT
```

Operation: [✓ Addition] Subtraction Multiplication Division
Result: Params [✓ DecFloat] BinFloat [✓ BinRepr] HexRepr [✓ Rational] [✓ Flags] Stack

Argument
Precision: 7
Exponent: 4
Base=2 L=-6 U=7

Rounding: [✓ Nearest] Upward Zero Interval Downward Rational

#1 0.1
Input: [✓ De] Stack
Print: Params [✓ DecFloat] BinFloat [✓ BinRepr] HexRepr [✓ Flags] [✓ Rational] Stack

#2 0.5
Input: [✓ DecFloat] BinFloat BinRepr HexRepr Rational Stack
Print: Params [✓ DecFloat] BinFloat [✓ BinRepr] HexRepr [✓ Flags] [✓ Rational] Stack

Exponent OK

Computing 0.1+0.5 in $\mathbb{F}(2, 7, -6, 7)$ and in $\mathbb{Q}_M$.

For a specified set of floating-point numbers, the didactical tool provides the whole functionality described in Sections 2–6, from basic operations to complete algorithms. Moreover, interval arithmetic, where the intervals have endpoints in the user-defined set of floating-point numbers, and rational arithmetic are supported. In the future the program will include rational interval arithmetic and rational rounding for irrational numbers.

The implementation of the didactical environment was realized using C++ classes, and was developed in cooperation with the students themselves, who were fascinated by the computer arithmetic issues discussed in the course. Full use of operator overloading also enables a user, in addition to using the environment for simple expressions, to run downloaded code from within the didactical software tool, using a user-specified precision and exponent range, rather than the IEEE single or double hardware floats. It should be mentioned that, since this is a true didactical program, no attention has been paid to the efficiency of the implementation. The tool was developed in Borland C++ with a Windows interface. It is in its last debugging phase and will be available in the coming winter term.

III. Numerical Techniques

We now describe the 4 main modules of the "Numerical Techniques" part of the course. As pointed out, each module follows the same general pattern from real-life problem to evaluation of the computational solution.

A. Linear algebra

This subject is of uttermost importance because so many real-life problems involve the solution of a system of linear equations. Hence the real-life problems that we discuss come from quite different areas such as

- computer graphics: finding the intersection points of lines and planes
- robotics: the movement of a robot can be described by matrix manipulations (multiplication and inversion) applied to the robot's state vector
- approximation theory: computing an approximation for the real number e by solving a tridiagonal system of linear equations, obtained from computing the convergents of a continued fraction expansion for e.

All the numerical methods which are discussed in this module are exact methods: Gaussian elimination (without and with partial pivoting) and QR-factorization. Having a proper understanding of computer arithmetic, students easily see that the implementation of an exact method only yields an exact solution if exact arithmetic is carried out with exact data. Since this is clearly not the case for any implementation in floating-point arithmetic, notions like "rounding error", "ill-conditioning" and "numerical stability" pop up naturally and are discussed thoroughly. Students get hands-on experience solving different problems, e.g. inverting the Hilbert matrix.

B. Root-finding

Motivating examples for this subject are:

- the implementation on a chip of a routine to compute the square root
- the difficult problem of polynomial root solving [4], [5]

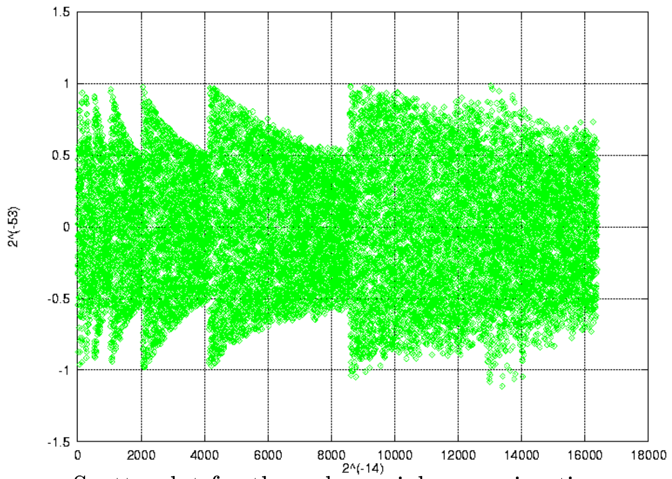
In contrast to the "Linear Algebra" module, the methods that are discussed here are iterative, and hence non-exact: bisection, Newton's method and regula falsi. These are fairly general purpose methods, in the sense that not too stringent requirements are imposed on the function of which the roots must be computed. For the implementation of the different methods, the students are asked to estimate the root-solving problem graphically and to determine starting-points for the iterative method from the graph. In this way, they confront the problem of reliable graphical output. Several non-linear equations resulting from real-life problems are given as projects.

C. Approximation theory

Motivating problems in this context are:

- CAD-CAM applications
- implementation of elementary functions on a chip

Of all the modules, this is the most comprehensive one since it covers several mathematical techniques, including interpolation, Chebyshev approximation, splines, least squares and rational approximation. For all except the last of these techniques, polynomials are used as approximating functions. After a brief theoretical discussion, we emphasize how the nature of the problem influences the choice of the approximation method. There is a cross-reference to the arithmetic part of the course, since the quality of the numeric data will determine whether the problem needs to be tackled in the least-squares or in the interpolation sense. The implementation of the function e^x in the mathematical function library `fdlibm`, made publicly available by SUN [6], is thoroughly discussed as an example, including the choice of approximation and the computer arithmetic details of the implementation.



Scatterplot for the polynomial approximation of degree 13 for $\sin(x)$ used in `fdlibm`

D. Random number generators

The course concludes with a module covering random number generators and Monte Carlo methods. Simulation techniques are essential when studying many of today's complex problems such as

- traffic engineering
- the computation of irregular tank volumes

In this module, we do not discuss the simulation of physical phenomena themselves, but rather the random number generators which underlie the simulation and which are essential for the reliability of the simulation. It is well-known that not all random number generators are equally good and the students are taught criteria to evaluate the quality of a random number generator. In the exercises, software for random number generation is retrieved via the Internet and tested according to several criteria.

E. Hands-on experience

To get acquainted with the topics covered in the second part, several small-scale problems are formulated and students are asked to perform the physical journey from problem to solution in some individual projects. After choosing a suitable algorithm, correct implementations can usually be found in well-known software environments such as Matlab, Maple or Mathematica. Students can also download software from well-known software sites such as Netlib (**http://www.netlib.org/**). In this respect, the Guide to Available Mathematical Software (**http://gams.nist.gov**) is also very helpful.

Acknowledgements

The authors greatly acknowledge the valuable support from their academic authorities for the extension of CS&E within the computer science curriculum at the University of Antwerp. Thanks are also due to Gene H. Golub for his hospitality and encouragement; the course material did benefit a great deal from a fruitful summer term at the Department of Computer Science in Stanford.

Colophon

The course "Computer Arithmetic and Numerical techniques" has recently won an Undergraduate Computational Science Education Award for its innovative contribution to the field.
More information on the award program can be found in `http://www.krellinst.org/UCES/awards/ugcsa97/`.
Direct information on the course can also be obtained from `http://win-www.uia.ac.be/u/cuyt/cant.html`.

About the authors

Annie Cuyt is Research Director at the FWO-Vlaanderen, the Fund for Scientific Research–Flanders, and teaches several computing courses at the University of Antwerp (UIA), among which the course "Computer Arithmetic and Numerical Techniques" described above. She received her Doctor Scientia degree in 1982 from the same university. She is the author of more than 90 publications in international journals and conference proceedings and the author or editor of several books. Her current interests are in "Computer Arithmetic" and in "Numerical Approximation Theory". In view of these interests she is an editorial board member of the new journal "Reliable Computing". Since 1997 she also serves as a member on the scientific committee of the Flemish Science Foundation.

Brigitte Verdonk is a postdoctoral fellow at the FWO-Vlaanderen and is affiliated with the Department of Mathematics and Computer Science of the University of Antwerp (UIA). She obtained a Master's degree in Computer Science from Stanford University in 1984 and a Doctor Scientia degree from the University of Antwerp in 1988. After a few years in industry, she returned to academic life and is now the author of approximately 30 publications in international journals and proceedings. She is currently coordinating the development of a programming language environment enabling the evaluation, with maximal accuracy, of hybrid numeric expressions (hardware and multiprecision floats, intervals, rationals, ...).

Authors' address: Department of Mathematics and Computer Science, University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerp (Belgium), Email: {cuyt,verdonk}@uia.ua.ac.be

References

[1] Thomas L. Marchioro, David M. Martin, and W. Donald Payne, "UCES: An undergraduate CSE initiative," *IEEE Computational Science & Engineering*, vol. 2, no. 3, pp. 69–73, 1995.

[2] ANSI/IEEE Std 754-1985, "IEEE standard for binary floating-point arithmetic," *ACM SIGPLAN*, vol. 22, no. 2, pp. 9–25, 1987.

[3] ANSI/IEEE Std 854-1987, "IEEE standard for radix-independent floating-point arithmetic," New York, 1987.

[4] "PoSSo (Polynomial System Solving)," `http://janet.dm.unipi.it/`, 1996, ESPRIT Basic Research contract with the Commision of the European Community (Contract BRA 6846).

[5] "FRISCO (A FRamework for Integrated Symbolic/numeric COmputation)," `http://www.nag.co.uk/projects/FRISCO.html`, 1996, ESPRIT Reactive LTR Scheme with the Commision of the European Community under (Project No. 21.024).

[6] SunSoft, "Fdlibm, a freely distributable C math library," Version 5, `netlib@research.att.com`.