Editor: Thomas L. Marchioro II, tlm@u.washington.edu

CSE AT BELGIAN UNIVERSITIES

Annie Cuyt and Brigitte Verdonk

Computational science and engineering education in Belgium is evolving favorably. Even though no Belgian university has a complete interdisciplinary CSE educational program, universities are including more CSE in their engineering, mathematics, and computer science curricula.

Earning a degree

To better understand Belgian CSE education, you should know something about our university system. To obtain a degree, students must complete two cycles of courses in a discipline. The first, *candidature*, provides a broad background in the discipline and includes basic courses in closely related subjects. Depending on the discipline, it is a two- or three-year program. The second, usually called *license*, is a two-, three-, or four-year program. It is more specialized than candidature, with different options to choose from. In the last year of the second cycle, most disciplines require students to write a thesis showing that they can independently understand new material.

After successfully completing two cycles, students can enter the job market, take a third cycle in a related discipline, or enter the doctoral program. The Belgian university system has a good reputation worldwide.

Crossing the borders

CSE is a methodology rather than a discipline.[1] More important, it is a methodology that requires crossing the borders between disciplines. Three A's characterize the different aspects of CSE: applications, algorithms, and (computer) architectures.[2,3] To meet current demands for CSE education, cross-fertilization occurs among the existing engineering, computer science, and mathematics curricula, in line with the cross-fertilization of applications, architectures, and algorithms. (The boxed text, "Belgian Universities with CSE-Related Programs" on page 80 gives more details.) A recent evaluation of the mathematics curricula of the Dutch-speaking universities confirms the need for this evolution: more emphasis should be given to actual computing in the mathematics and applied mathematics curricula.

All exact-science curricula have in the first cycle at least one extensive mathematics or applied-mathematics course and one introductory computer science course. Traditionally, however, little interaction occurs between the math/computer science and the respective science departments. Opportunities to develop these courses in an interdisciplinary fashion according to the CSE problem-solving methodology are not exploited.[1] The third cycle is, in the current context, the only framework for interdisciplinary programs such as computer science and industrial mathematics, biomedical and clinical engineering, biostatistics, and so on.

That Belgium has no true CSE program is due partly to the fact that it is a small country. Even if the need for such a program is acknowledged, the expense would be too great for several universities to offer a CSE curriculum. Yet promoting one university over all others for the organization of a CSE program would be politically delicate, without nationwide reform of the higher-education policy. A proposal to introduce university *gravitation points* is the subject of an ongoing political debate and might shed new light on the CSE situation. Rather than nearly all universities offering nearly all disciplines (as is the case now), each university would attract all students in specific disciplines.

Scientific computing evolved because of the rapid changes in hardware and the advent of symbolic computing.

CSE research

Several scientific research communities group researchers from engineering, applied mathematics, and computer science. Yet until recently, funding agencies tended to underestimate the opportunities provided by a computational approach to science. CSE projects—for instance, in computational chemistry—seemed to suffer an image problem, being regarded as neither chemistry nor computer science research projects. Moreover, with the advent of computer science as a full-fledged discipline, interest in and funding for numerical analysis and scientific computing dwindled. In recent years, there has been a reversal of that trend. Scientific computing, having greatly evolved because of the rapid changes in computer hardware, together with the advent of symbolic computation, has taken up a forefront position again.

Belgian Universities with CSE-Related Programs

These universities offer a degree in one or more of the disciplines of engineering, mathematics, and computer science:

Dutch-speaking

♦ Katholieke Universiteit Leuven (KUL), http://www.kuleuven.ac.be/kuleuven

♦ Universiteit Gent (UG), http://www.rug.ac.be

♦ Universiteit Antwerpen (UA), http://www.ua.ac.be

♦ Vrije Universiteit Brussel (VUB), http://www.vub.ac.be

♦ Limburgs Universitair Centrum (LUC), http://www.luc.ac.be

French-speaking

♦ Université Catholique de Louvain (UCL), http://www.ucl.ac.be

♦ Université Libre de Bruxelles (ULB), http://www.ulb.ac.be

♦ Université de Liège (ULg), http://www.ulg.ac.be

♦ Facultés Universitaires Notre-Dame de la Paix in Namur (FUNDP), http://www.fundp.ac.be

♦ Université de Mons–Hainaut (UMH), http://www.umh.ac.be

♦ Faculté Polytechnique de Mons (FPMs), http://www.fpms.ac.be

Engineering

Regarding the Dutch-speaking universities, the programs at the UG and at the KUL offer a specialization in computer science. This specialization is further split into options. In particular, Leuven's applied-mathematics option centers on algorithmic and numerical problems and their implementation on different computer architectures. As for the French-speaking schools, the UCL offers applied-mathematics and computer science specializations, and the ULB offers a computer science specialization. These specializations, in turn, have several options covering different CSE aspects.

Computer science

The UG offers a numerical computer science option, and the KUL offers a numerical-analysis option. On the French-speaking side, the ULg offers a mathematics option.

Mathematics

All Dutch-speaking universities offer an applied-mathematics specialization, but only the VUB and the UA have a computer science specialization. Evaluating the contribution to CSE of the applied-mathematics specialization is very difficult, because its content varies greatly between universities. Regarding the French-speaking universities, they are very diverse with respect to the different specializations offered. The UCL, the UMH, and the FUNDP offer a computer science specialization. The latter university offers this specialization in the first cycle of the mathematics curriculum only, while in the second cycle applied-mathematics specializations such as numerical analysis, optimization, and mechanics and differential equations are offered.

The growth of CS studies

The most striking feature of the evolution of the student population at Dutch-speaking universities is the rapidly increasing number of computer science students. This phenomenon is not only typical of Flanders or of Belgium as a whole, but is also evident in many European countries. At some universities in neighboring countries, the decrease in the proportion of mathematics students to computer science students has already led to the merging of mathematics and computer science departments.

The University of Antwerp, which does not have an engineering school, offers computer science and mathematics. Although only one-eighth of the Flemish student population studies at our university, our computer science program accounts for almost a fourth of all Flemish computer science degrees. Thus, we have the opportunity to address a large audience potentially interested in CSE. This was reason enough to develop a scientific-computing course with special attention to the computer science background of the students. As the boxed text "Scientific Computing for the CS Student" on the nest page illustrates, getting computer science majors interested in CSE takes more than classical numerical analysis courses! ♦

Acknowledgments

We thank our academic authorities for their support in extending CSE in the computer science curriculum at the University of Antwerp. We also thank Gene H. Golub for his hospitality and encouragement; the course material described in the boxed text benefited greatly from a fruitful summer term at Stanford's Department of Computer Science.

References

1. T.L. Marchioro II, D.M. Martin, and W.D. Payne, "UCES: An Undergraduate CSE Initiative," *IEEE Computational Science & Eng.*, Vol. 2, No. 3, Fall 1995, pp. 69–73.
2. J.R. Rice, "Academic Programs in Computational Science and Engineering," *IEEE CS&E*, Vol. 1, No. 1, Spring 1994, pp. 13–21.
3. Robert M. Panoff, "The Four A's of CSE Education: Application, Algorithm, Architecture, and Active Learning," *IEEE CS&E*, Vol. 2, No. 4, Winter 1995, pp. 6–9.

Annie Cuyt is research director and Brigitte Verdonk is a senior researcher at the FWO-Vlaanderen (Fund for Scientific Research-Flanders); e-mail, {verdonk, cuyt}@ uia.ua.ac.be.

Scientific Computing for the CS Student

The University of Antwerp's Computer Arithmetic and Numerical Techniques course is an introduction to scientific computing for students with a major in computer science or a combined major of mathematics and computer science. It is also appropriate for any exact-science or applied-science student with a reasonable high school background in math and an interest in computers. It is this growing group of science students interested in computer science that we are trying to encourage into scientific computing.

Unlike engineering students, CS students rarely confront real-life scientific-computing problems in other courses. Yet we believe that the best approach to a scientific-computing course is to follow the complete journey from physical problem to computational solution:[1]

1. Introduction of a motivating problem
2. Identification of the computational problem behind the given real-life problem
3. Selection of an appropriate numerical technique to solve the problem
4. Implementation of a numerical routine
5. Evaluation or quality control of the numerical output

Each topic in the course covers all these steps, with special attention to the computer science background of the students. We use reasonably realistic examples that are not too technical. Moreover, we treat all aspects of Step 4 in detail. This approach differs from other scientific-computing courses in that we emphasize the distinction between the properties of a mathematical algorithm and the properties of the algorithm's implementation in finite precision arithmetic. Computer science students are interested in computer arithmetic as part of scientific computing, in the same way they are interested in learning about compilers to obtain a full understanding of programming languages.

The course consists of two parts: computer arithmetic (a 15-hour course load) and numerical techniques (a 30-hour course load).

Computer arithmetic

This part of the course discusses how to represent the number sets $\mathbb{Z}$ (integer arithmetic), $\mathbb{Q}$ (exact rational arithmetic), and $\mathbb{R}$ (exactly rounded arithmetic) on a computer and how using these representations instead of the numbers themselves influences computation. The level of complexity evolves as we go from $\mathbb{Z}$ to $\mathbb{Q}$ to $\mathbb{R}$, and increases as we perform more complex operations on each set.

Section 1 covers computer representations of $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ and I/O from and to these sets:

- the set of machine integers $\mathbb{Z}_t$, where t indicates the number of bits that 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 $[L, U]$, often denoted by $\mathbb{F}_t$.

In the next three sections of the course, we discuss operations on the computer number set $\mathbb{F}_t$; that is, the basic operations +, –, *, and /, the relational operators, and the implementation of some elementary functions.

Section 5 deals with compound statements, which involve such problems as the accumulation of rounding errors, benign and catastrophic cancellation, and the choice of an evaluation strategy (the widest format available, the widest-needed precision, and so on), especially when operands of different precisions are mixed. We illustrate the effect of the evaluation strategy by running the same numeric code on different hardware platforms, for example Sun Sparcs versus Intel PCs.

At this point, the students have all the ingredients to implement complete numerical algorithms in $\mathbb{F}_t$. The buildup of rounding and data errors in the implementation of algorithms leads to the essential concepts of forward and backward error analysis, numerical instability, and ill-conditioning. Section 6 covers these in detail.

Section 7 discusses the IEEE standard.[2,3] This standard embodies all the details encountered when effectively implementing floating-point arithmetic on a binary machine: denormals, special representations, exception flags, and so on.

The next sections of the class present alternatives to IEEE floating-point arithmetic. Section 8 discusses multiprecision floating-point arithmetic, while Section 9 discusses interval arithmetic. Section 10 concludes with rational arithmetic and is a good starting point for extra material on polynomial and symbolic computation.

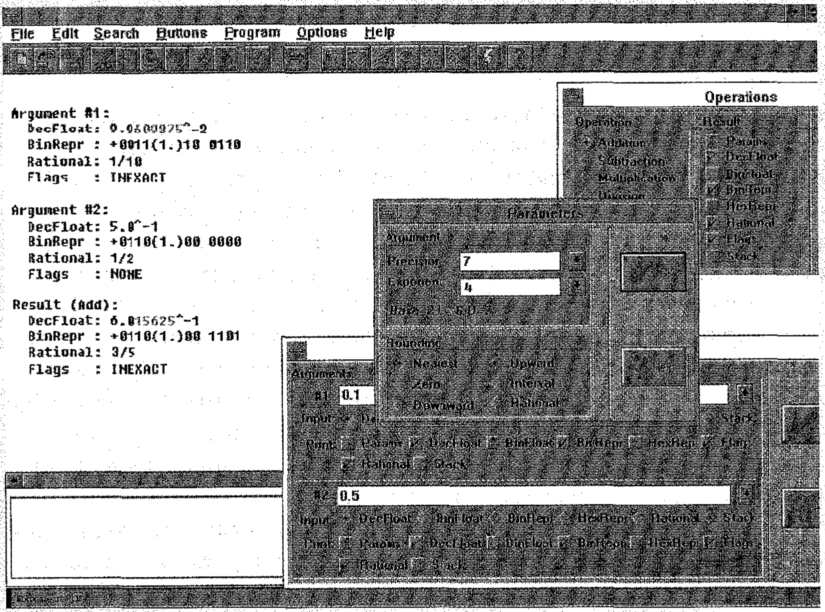
Hands-on experience. Although the above material is simple and clear-cut, students have difficulty grasping its intricacies and understanding the interaction of the different errors that can occur. To help them overcome this difficulty, we have developed the software tool Arithmetic Explorer, shown in Figure A.

In the Arithmetic Explorer, students can define their own set of floating-point numbers. By choosing a small precision t and a limited exponent range, they can easily follow the computations at the bit level. Moreover, in a low-precision floating-point set, they can easily zoom in on the unmistakable effects of data and rounding error, cancellation, ill-conditioning, and numerical instability, and can develop a better feeling for computer arithmetic issues.

We have taken care to implement the tool's floating-point arithmetic in full compliance with the philosophy of

Continued on page 82

Continued from page 81



```
Argument #1:
  DecFloat: 0.0800975^-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
```

Figure A. 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 Explorer provides the whole functionality described in Sections 2 through 6, from basic operations to complete algorithms. Moreover, it supports interval arithmetic and rational arithmetic. In the future, the program will include rational interval arithmetic and rational rounding for irrational numbers.

We developed Arithmetic Explorer in Borland C++. Several students who were fascinated by the computer arithmetic issues we discussed cooperated in the tool's development. Because it is a teaching program, we paid no attention to the implementation's efficiency. It is in its last debugging phase and will be available in the coming winter term.

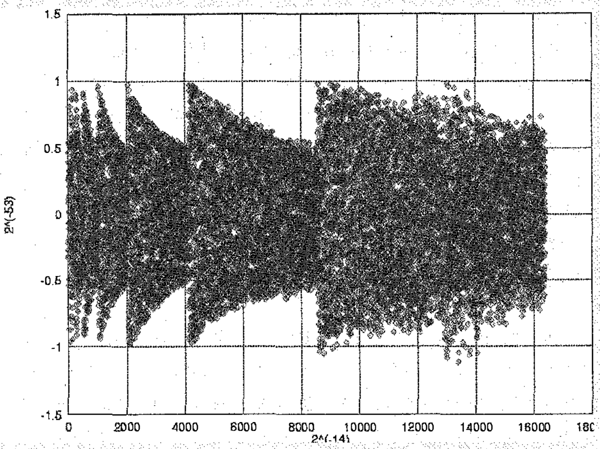


Figure B. Scatterplot for the polynomial approximation of degree 13 for sin (x) used in `fdlibm`.

the IEEE standard. Except that users can freely specify the precision and exponent range, all aspects of the IEEE standard are supported and can be visualized, including exact rounding, denormals, signed zero, infinities, not-a-numbers, and exception flags. In this way, students can discover the details of floating-point arithmetic. While a similar analysis can be done with other tools, such as Mathematica, or by direct programming using a traditional compiler, the result is often a time-consuming and confusing task, obscuring more points than one is trying to make.

Numerical techniques

In the four modules—linear algebra, root finding, approximation theory, and random-number generators—we follow the same journey from real-life problem to computational solution.

Linear algebra is extremely important because many real-life problems involve solving a system of linear equations. This module covers problems from such areas as computer graphics and robotics.

We discuss exact numerical methods such as Gaussian elimination (without and with partial pivoting) and QR-factorization. Having a proper understanding of computer arithmetic, students easily see that implementing an exact method yields an exact solution only if exact arithmetic is carried out with exact data. Because this is clearly not the case in floating-point arithmetic, concepts such as rounding error, ill-conditioning, and numerical stability pop up naturally and are discussed thoroughly.

The module on root finding treats real-life problems such as the implementation on a chip of a routine to compute the square root and the difficult problem of polynomial root solving.[4,5]

Unlike the linear algebra module, we introduce iterative and hence nonexact methods: bisection, Newton's method, and regula falsi. To determine starting points for an iterative method, the students estimate the root-solving problem graphically, thereby confronting the problem of reliable graphical output.

The approximation theory module is the most comprehensive; it starts off with motivating problems from CAD-CAM and the implementation of elementary functions on a chip. Figure B shows the scatterplot for the approximation used in the mathematical function library—`fdlibm` for sin(x).[6] Here we cover several mathematical techniques, including interpolation, Chebyshev approximation, splines, least squares, and rational approximation. All these techniques except the last use polynomials as approximating functions. After a brief theoretical discussion, we emphasize how the problem's nature influences the choice of the approximation method.

The last module covers random-number generators. They are at the basis of simulation techniques, which are essential in the study of complex problems, such as traffic engineering and the computation of irregular tank volumes. Students learn that not all random-number generators are equally good and study criteria for evaluating their quality.

Projects. The students work on several small-scale problems and on individual projects. After choosing a suitable algorithm, students can usually find correct implementations in well-known software environments such as Matlab, Maple, or Mathematica. They can also download software from well-known software sites such as Netlib (http://www.netlib.org/) or GAMS (the Guide to Available Mathematical Software, http://gams.nist.gov).

Further enlightenment

Our course recently won an Undergraduate Computational Science Education Award for its innovative approach. For information on the award program, access http://www.krellinst.org/UCES/awards/ugcsa97/. For more information on the course, access http://win-www.uia.ac.be/u/cuyt/cant.html.

References

1. T.L. Marchioro II, D.M. Martin, and W.D. Payne, "UCES: An Undergraduate CSE Initiative," *IEEE Computational Science & Engineering*, Vol. 2, No. 3, Fall 1995, pp. 69–73.
2. "IEEE Standard for Binary Floating-Point Arithmetic," ANSI/IEEE Std 754-1985. Reprinted in *ACM SIGPLAN*, Vol. 22, No. 2, 1987, pp. 9–25.
3. "IEEE Standard for Radix-Independent Floating-Point Arithmetic," ANSI/IEEE Std 854-1987, New York, 1987.
4. "PoSSo Home Page" (Polynomial System Solving), ESPRIT Basic Research Contract with the Commission of the European Community, BRA 6846, 1996, http://janet.dm.unipi.it/.
5. "Frisco (A Framework for Integrated Symbolic/Numeric Computation)," Commission of the European Community under ESPRIT Reactive LTR Scheme, Project No. 21.024, 1996, http://www.nag.co.uk/projects/FRISCO.html.
6. SunSoft, "fdlibm, A Freely Distributable C Math Library," version 5, http://www.netlib.org/fdlibm.

Computational Science Education Opportunity

Since 1987 the EPCC Summer Scholarship Programme has trained over 150 senior undergraduates to use state-of-the-art high-performance computers. The 10-week program gives students a mixture of training and project work, as well as the opportunity to meet students from different cultures and backgrounds.

The program will begin in Edinburgh, UK, on 6 July 1998. It is open to senior undergraduates of any nationality from any discipline with a strong computing emphasis.

The closing date for 1998 applications is 13 February. Forms are available at http://www.epcc.ed.ac.uk/ssp/.