AIAA JOURNAL
Vol. 34, No. 6, June 1996

Book Reviews

REAL Computing Made Real

Forman S. Acton, Princeton University Press, Princeton, NJ, 1996, 259 pp., $29.95

In the first half of the century, mathematicians were still considered philosophers. Applied mathematicians were philosophers deriving their inspiration and interest from physical or technical problems, searching for a solution through formal analyses and computing by hand with mechanical adding machines. Engineers were happy to get three significant digits from slide rules, since no mechanical tolerance could be made any tighter. An airfoil would be mapped on a circle using 12 constants, one every 30 deg. Singularities were circumvented after having been located and understood. In the second half of the century, computers allowed studies of high-speed fluid mechanics, complex chemical evolutions, and the like, requiring lengthy calculations and minute details, to be made. Meanwhile, applied mathematicians (at least, in the aerospace community) branched out into two different species; let us call them the Nobles and the Yeomen. Typical Nobles scrutinize discretization techniques, seldom trying to relate their mathematical analysis to the physical origin of the problem, thus producing theorems hardly acceptable in practice; they are generally not interested in numerical results. Typical Yeomen generate programs to answer specific questions in physics and engineering; they are handicapped by the lack of a formal mathematical training and have only a vague knowledge, if any, of classical numerical techniques and transcendental functions. Consequently, they acquire an almost religious faith in the Scriptures produced by the Nobles and in the thaumaturgical powers of prefabricated software, no questions asked. Only a few people who can make intelligent use of certain abstract considerations and have a robust background in classical mathematics and the physics of their problem can positively contribute to a progress in science. Even they, however, may see their work falling into some booby trap in its final stage, that is, when numbers are produced, because most of the time they must rely on Yeomen, and supervision is hard.

Let us assume, thus, that a complicated code has been written, where all discretizations are mathematically correct and all physical principles are respected (conservation, boundary conditions, domains of dependence, discontinuities, etc.). Let us also assume that all equations and their codes were checked; that all constants and signs are correct; that there are no misprints; and that the logic, including the operations of counters in loops, is sound. Is the code reliable for any set of parameters? Generally, the answer is No, and certain limitations on the range of such parameters have to be imposed. Even then, are we sure that the numbers we obtain are always reliable? More precisely, if we need an accuracy within a given number of digits, are we sure that we have them? That means, how many digits did we lose over the entire computation, where and why?

Acton's book is a primer for those interested in answering such questions. Most often, a single inadequate step in a program may destroy accuracy in an entire range of results, for example, when two big numbers are subtracted to generate a small number that in turn is divided by another small number. Religious reliance on software produces such errors. I love Acton's book because it goes straight to the origin of the booby traps and, from the beginning, gives the reader the best possible advice: Do not attempt to code without having understood the functions that the code should generate, and their interactions; do not produce numbers before you have a sketch of what the resulting curves should look like. I love such emphasis on geometrical insight; I have myself spent a lifetime making sketches before graphic software was invented, and even after, and I have forced my students to do the same (they did not hide their annoyance, but in the end they showed their gratitude). Another important piece of advice from the book is not to ask software to produce a transcendental function without knowing the function itself. The book, of course, is not a tract on transcendental functions, and it does not dwell on them specifically; but the advice should be taken very seriously by the reader, and even more importantly by the teachers [I would like to take a poll on the physical implications of x being smaller or larger than p in the Bessel function $J_p(x)$]. The book is not a cookbook; quite the opposite, it aims to enhance sensitivity through exercises and analogies. In aerospace calculations, where wide regions often must be evaluated together with thin layers (and the thin layers, where significant digits are commonly lost, are the most important), brute-force approaches and reliance on prefabricated software are catastrophic. The problems in the book are deceptively simple; to be mastered they require time and labor, that is, a dedicated effort by the reader. The author, as a good teacher, guides the reader by the hand, but he does not do the homework. I would like to see the book on the desks of seniors, graduate students, and young engineers and physicists. Safely sheltered by my retirement, I would also strongly recommend the book to all people who, as teachers or senior scientists in the industry, have influence on the Yeomen. Would that suffice to arrest the current trend toward laziness? How many readers would perform the requested work as if it were needed for a credit or a promotion? Would their teachers be willing to supervise their efforts? That we do not know. I may only say that, in my opinion, the author has done his best to lead; he should be praised and followed, and the spirit of his book should live even in a future of increasingly better software.

Gino Moretti
Polytechnic University