literate programming

Moderated by
Christopher J. Van Wyk

LITERATE PROGRAMMING: AN ASSESSMENT

When Donald Knuth wrote the TEX® program, one of his goals was to publish it as a program "of which a professor of computer science might be proud, in spite of the fact that it meets real-world constraints and compromises" [2, p. v]. To this end, he and some of his students wrote systems that were intended to foster documentation as a natural part of programming, and that allowed one to present programs in a fashion tailored for human understanding.

WEB [3, 4] was the culmination of these efforts. WEB input can be "woven" and printed using TEX, so that a person can read the program; thus, the WEB user has access to the full power of the TEX typesetting system in composing documentation. WEB input can also be "tangled" and compiled by a Pascal compiler, so that a computer can execute the program; thus, the WEB user can declare variables, define parts of procedures, and otherwise present pieces of the program in an order natural for exposition, rather than one dictated by Pascal.

After he had used WEB for several years, Knuth realized that it had changed the way he wrote programs, "that at last I'm able to write programs as they should be written," and he dubbed the new style "literate programming" [1, p. 97]. In the May and June, 1986 issues of *Communications of the ACM*, the Programming Pearls column presented two literate programs by Knuth.

Several common aspects are apparent in the various literate programs Knuth has published. The most obvious is *cosmetic*; the documentation is typeset like a book, and the code is typeset in a "reference-Algol" style. This is the easiest thing to notice, but it is hardly essential to a program's being literate. Harold Thimbleby's CWEB system, for example, typesets code in typewriter font [5], and surely one can write literate documentation without access to a multifont typesetting system.

A second aspect of Knuth's literate programs is the *polish* they exhibit; the code and the documentation were written with meticulous care; indeed, the documentation not only explains what the code does, but it explains *why* the approach was chosen, and sometimes why plausible alternatives were rejected. Such polish makes it pleasant to read a literate program, but it can hardly be considered distinctive of literate programs: authors have polished their programs for careful exposition for years.

A less obvious, but ultimately crucial, feature of Knuth's literate programs I would dub their *verisimilitude*: the published programs were produced from *exactly* the same input that was used to prepare the program that the computer executed. Verisimilitude is unique among these three aspects in that it distinguishes a literate program from a program that has merely been highly polished and presented with attention to cosmetic details.

Four programs have appeared since July, 1987, when this column was commissioned to explore different approaches to literate programming. All of the programs' authors worked hard to polish their presentations. And all four share with Knuth's literate programs the cosmetic appearance of code interleaved with typeset comments; each achieved this appearance in a different way, but in all cases the production of the version for presentation followed the writing of the working code, and none of the published programs exhibits complete verisimilitude with working code.

In June, 1989. Thimbleby served as reviewer, and courteously reminded us of the essential role that verisimilitude plays in a program's being literate. He also went on to suggest that the code and the documentation of a literate program must be *produced* (not merely presented to the reader) *simultaneously*, and that a system for literate programming should automatically provide such literary paraphernalia as tables of contents and cross references.

Thimbleby's restatement of fundamental marks of literate programs inspires me to resolve that future columns will publish only programs that were produced using a *system* for literate programming. And that presents a problem. I know of perhaps a half-dozen systems for writing literate programs, each modeled on WEB, perhaps adding or subtracting a few features, or

(*continued on p. 365*)

®TEX is a trademark of the American Mathematical Society.

© 1990 ACM 0001-0782/90/0300-0361 $1.50

ters that minimize the sum of the squared weighted orthogonal distances from a set of observations to a curve or surface determined by the parameters. It can also be used to solve the ordinary nonlinear least squares problem. The weighted orthogonal distance regression procedure has an application to curve and surface fitting and to measurement error models in statistics. The algorithm implemented is an efficient and stable trust region (Levenberg-Marquardt) procedure that exploits the structure of the problem so that the computational cost per iteration is equal to that for the same type of algorithm applied to the ordinary nonlinear least squares problem. The package allows a general weighting scheme, provides for finite difference derivatives, and contains extensive error checking and report generating facilities.

For Correspondence: P. T. Boggs, Applied and Computational Mathematics Division, National Institute of Standards and Technology, Gaithersburg, MD 20899; J. R. Donaldson, Applied and Computational Mathematics Division, National Institute of Standards and Technology, Boulder, CO 80303-3328; R. H. Byrd, Department of Computer Science, University of Colorado, Boulder, CO 80309; R. B. Schnabel, Department of Computer Science, University of Colorado, Boulder, CO 80309 and Applied and Computational Mathematics Division, National Institute of Standards and Technology, Boulder, CO 80303-3328.

ALGORITHM 677
C^1 Surface Interpolation

Laura Bacchelli Montefusco and Giulio Casciola

A method of bivariate interpolation and smooth surface fitting is developed for rapidly varying z values given at points irregularly distributed in the x–y plane. The surface is constructed by means of C^1 triangular interpolants defined on a triangulation of the convex hull of the points set. The needed partial derivative values are estimated by a new method based on a minimization criterion making use of a tension parameter. This method, which is shown to be efficient and accurate, gives the user an interactive tool to control the behavior of the interpolant surface and to dampen unwanted oscillations near steep gradients. The algorithm of this proposed method is described.

For Correspondence: L. B. Montefusco and G. Casciola, Universita Degli Studi di Bologna, Dipartimento di Matematica, Piazza di Porta S. Donato 5, 40127 Bologna, Italy.

Indefinite Integration with Validation

George Corliss and Gary Krenz

We present an overview of two approaches to validated one-dimensional indefinite integration. The first approach is to find an inclusion of the integrand, then integrate this inclusion to obtain an inclusion of the indefinite integral. Inclusions for the integrand may be obtained from Taylor polynomials, Tschebyscheff polynomials, or other approximating forms which have a known error term. The second approach finds an inclusion of the indefinite integral directly as a linear combination of function evaluations plus an interval-valued error term. This requires a self-validating form of a quadrature formula such as Gaussian quadrature. In either approach, composite formulae improve the accuracy of the inclusion.

The result of the validated indefinite integration is an algorithm which may be represented as a character string, a subroutine in a high-level programming language such as Pascal-SC or Fortran, or as a collection of data. An example is given showing the application of validated indefinite integration in constructing a validated inclusion of the error function, erf(x).

For Correspondence: Department of Mathematics, Statistics, and Computer Science, Marquette University, Milwaukee, WI 53233.

ALGORITHM 678
BTPEC: Sampling from the Binomial Distribution

Voratas Kachitvichyanukul and Bruce W. Schmeiser

The FORTRAN implementation of an exact, uniformly fast algorithm for generating the binomial random variables is presented. The algorithm is numerically stable and is faster than other published algorithms. The code uses only standard FORTRAN statements and is portable to most computers; it has been tested on the IBM 370, 3033, 4381, DEC VAX 11/780, SUN 3/50, CDC 6500-6600, ENCORE Multimax, and Apple Macintosh Plus. A driver program is also included.

For Correspondence: U. Kachitvichyanukul, Department of Industrial and Management Engineering, The University of Iowa, Iowa City, IA 52242; B. W. Schmeiser, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907.

Literate Programming (*continued from p. 361*)

working with different programming languages and typesetting systems. Unfortunately, no one has yet volunteered to write a program using another's system for literate programming. A fair conclusion from my mail would be that one must write one's own system before one can write a literate program, and that makes me wonder how widespread literate programming is or will ever become. This column will continue only if I hear from people who use literate-programming systems that they have not designed themselves.

The alternative to this purist approach seems to be to broaden the meaning of "literate programming" to include any effort to program with style, and to make the column an exploration of programming style in general. From my mail I gather that some readers would be sympathetic to this course. Besides dishonoring Knuth's intentions in coining the term "literate programming," however, such a broadening of the column's charter would give me more power as an arbiter of style than I or any other moderator ought to have.

In closing, I would like to thank the many people who have written to or spoken with me about the column. Those whose programs and reviews have appeared have been diligent in meeting deadlines; others whose work has not appeared have been gracious in their acceptance of negative verdicts; and I am always happy to meet readers and learn of their interest in literate programming.

Christopher J. Van Wyk
AT&T Bell Laboratories
600 Mountain Avenue
Room 2C-457
Murray Hill, NJ 07974

REFERENCES

1. Knuth, D.E. Literate programming. *Comput. J.* 27, 2 (May 1984), 97–111.
2. Knuth, D.E. TEX: *The Program*, Volume B of *Computers and Typesetting*. Addison-Wesley, Reading, Mass., 1986.
3. Knuth, D.E. The WEB system of structured documentation. Tech. Rpt. 980, Stanford University Computer Science Department, Stanford, Calif., 1983.
4. Sewell, W. *Weaving a Program*. Van Nostrand Reinhold, N.Y., 1989.
5. Thimbleby, H. Experiences of "literate programming" using CWEB (a variant of Knuth's WEB). *Comput. J.* 29, 3 (June 1986), 201–211.