CS 279 Annotated Course Bibliography

David Bindel

August 1, 2001

This annotated bibliography is a complement to a set of course notes for CS 279, System Support for Scientific Computation, taught in Spring 2001 by W. Kahan at UC Berkeley. It is meant to be representative, not comprehensive, and includes pointers to survey works and online bibliographies which cover the literature in various areas in more depth. In the hope of making the bilbiography simpler to use, the notes are partitioned by topic. The categorization is far from precise, but where a reference provides pertinent coverage of multiple topics, I have tried to provide appropriate cross-references in the section text.

This bibliography is based largely on the bibliography assembled by Judy Hu for the Spring 1992 version of this course. Other major sources for annotated entries include the bibliography of the *Apple Numerics Manual* and bibliographies from the papers of W. Kahan. Annotations attributable to those sources are clearly labeled.

1 General interest

One of the most popular introductions to floating point arithmetic at this time is David Goldberg's article [1]. Goldberg based his article on the contents of a 1988 class taught at Sun Microsystems by W. Kahan. The notes from a similar class taught in 1973 [4] provide a much more dated coverage of some of the same topics. The article "A Survey of Error Analysis" [3] is recommended by David Hough in a foreword to the 1973 course notes as a summary. The current reader inclined to delve further into topics covered in this course should seek out the article before ordering the 1973 course notes, and would be best advised not to seek either until after reading Goldberg's articles and the articles on Kahan's web page.

The book by Sterbenz [6] is a dated, but still useful, introduction to some of the topics of floating-point analysis discussed in class. The second volume of Knuth's opus magnus [5] covers floating point arithmetic and related matters, though from a rather different perspective than most of the work cited here. Knuth also has a very thorough bibliography.

Finally, a recently translated volume by Ifrah [2] gives a historical account of computing, from the rise of various number systems around the world through the present day.

References

[1] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

> *Judy Hu* A tutorial on IEEE 754/854 Standard and other software and hardware issues of floating point arithmetic that have a direct impact on computer system designers. A version augmented by D. Priest's comments is available on-line at `http://www.validlab.com/`.

[2] Georges Ifrah. *The Universal History of Computing: From the Abacus to the Quantum Computer*. Wiley, New York, 2001. Translated from the French and with Notes by E. F. Harding.

[3] W. Kahan. A survey of error analysis. In *Proc. IFIP Congress, Ljubljana*, Information Processing 71, pages 1214–1239. North Holland, Amsterdam, 1972.

[4] W. Kahan. Implementation of algorithms (lecture notes by W. S. Haugeland and D. Hough). Technical Report 20, Department of Computer Science, University of California, Berkeley, CA, 1973.

[5] Donald E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, second edition, 1981.

> *Judy Hu* (Is the 3rd edition out yet?) Includes two chapters about floating-point arithmetic and multi- precision arithmetic, and many references to earlier literature.

[6] Pat H. Sterbenz. *Floating-Point Computation*. Prentice Hall, Englewood Cliffs, NJ, 1974.

> *Judy Hu*: A good text book on floating-point computation. Includes a detailed discussion of the arithmetics of IBM's 7094 and /360, and CDC's 6600.
> *Apple*: This book is obsolescent. It describes some things you have to know and others you used to have to know; it gives a good idea of why floating-point computation is simpler nowadays.
> *W. Kahan*: From quick courses for IBM's customers in the 1960s; instructive but out of print.

2 Major online resources and bibliographies

A wealth of material pertaining to floating point computation and error analysis is available online. Many articles and writings by W. Kahan, most of which are relevant to topics from this class, are available at his web page [9]. Specific papers will be cited in other sections of the bibliography. Jean-Michel Muller's

page [11] includes pointers to a variety of web pages of people, conferences, journals, and books. David Hough's page [8] contains the highly popular survey article by David Goldberg with the appendix by Doug Priest, as well as pointers to references on a variety of topics and archives of the numerics mailing list he maintains.

Muller's page contains pointers to extensive bibliographies compiled by Muller [12], Kornerup [10], and Beebe [2, 1, 3]. The bibliography from Higham's recent book is also available online [7]. More generally, the Collection of Computer Science Bibliographies [5] provides access to many relevant bibliographies. Citeseer [4] is valuable not only for providing bibliographic information for various papers, but also abstracts, information about where papers were cited, and pointers to online versions.

The Netlib repository [13] hosts many numerical packages (often with accompanying documentation) along with a few online books. The Guide to Available Mathematical Software [6] is a useful index of mathematical software available online.

References

[1] Nelson Beebe. Elementary function bibligraphy. `http://www.math.utah.edu/ftp/pub/tex/bib/elefunt.html`.

[2] Nelson Beebe. Floating-point bibliography. `http://www.math.utah.edu/ftp/pub/tex/bib/fparith.html`.

[3] Nelson Beebe. Interval arithmetic bibligraphy. `http://www.math.utah.edu/ftp/pub/tex/bib/intarith.html`.

[4] ResearchIndex: The NECI scientific digital library. `http://www.citeseer.com/`.

[5] Collection of computer science bibliographies. `http://liinwww.ira.uka.de/bibliography/index.html`.

[6] Guide to available mathematical software. `http://gams.nist.gov/`.

[7] Nicholas Higham. Bibliography for *Accuracy and Stability of Numerical Algorithms*. `http://www.ma.man.ac.uk/~higham/asna-bib.html`.

[8] David Hough. `http://www.validlab.com/`.

[9] W. Kahan. `http://www.cs.berkeley.edu/~wkahan`.

[10] Peter Kornerup. `http://www.ira.uka.de/bibliography/Theory/arith.html`.

[11] Jean-Michel Muller. `http://www.ens-lyon.fr/~jmmuller/`.

[12] Jean-Michel Muller. `http://www.ens-lyon.fr/~jmmuller/biblio`.

[13] Netlib software repository. `http://www.netlib.org`.

3 IEEE 754 and 854

Early public debates about the IEEE 754 standard were published in a special issue of the SIGNUM Newsletter, number 14 in October of 1979. Shortly after (January of 1980), an article describing the KCS proposal by Jerome Coonen [6] appeared in *Computer*. In March of the following year, another issue of *Computer* carried several articles describing various aspects of the draft standard [11, 5, 7, 10]. A technical report by Kahan [12], also in 1981, detailed some of the guiding rationale behind the evolving standard and provided examples of its use. The final draft of the standard [1] was released by the IEEE in 1985; Jerome Coonen's thesis [8], published the year before, elaborated on the standard. Even before the 754 standard was officially published, a proposed version of the radix-independent 854 standard was published in *IEEE Micro* [3].

Gradual underflow was (and two decades later remains) one of the most controversial features of the 754 standard. A paper by on gradual underflow by Coonen [7] in the 1981 *Computer* article series notes that

> In fact, underflow should not be an important issue. ... Ironically, gradual underflow was expected to go unnoticed by most users, coming into view only when potentially dangerous underflow errors were flagged.

An article by Demmel [9] further analyzed the difference made by gradual underflow.

References

[1] IEEE Standards Committee 754. *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985.* Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

[2] W. J. Cody. Floating-point standards—Theory and practice. In Ramon E. Moore, editor, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, pages 99–107. Academic Press, Boston, 1988.

[3] W. J. Cody, J. T. Coonen, D. M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. N. Ris, and D. Stevenson. A proposed radix- and word-length-independent standard for floating-point arithmetic. *IEEE Micro*, 4(4):86–100, 1984.

> *Judy Hu*: The most readable and precise exposition of the IEEE 854 standard to date; some implementation problems and how to overcome them are also discussed.
> *Apple*: This article makes the proposed IEEE 854 Standard available for public comment and discusses implementation problems. SANE implementations conform to the more general propsed Standard 854, as well as to standard 754.

[4] W. J. Cody and Jerome T. Coonen. ALGORITHM 722: Functions to support the IEEE standard for binary floating-point arithmetic. *ACM Transactions on Mathematical Software*, 19(4):443–451, 1993.

[5] William J. Cody, Jr. Analysis of proposals for the floating-point standard. *IEEE Computer*, 14(3):63–69, March 1981.

Apple: This paper compares the several contending proposals presented to the Working Group.

[6] J. T. Coonen. An implementation guide to a proposed standard for floating-point arithmetic. *Computer*, 13(1):68–79, 1980.

Now of primarily historical interest, this paper describes the features of an early draft of the 754 standard (which changed substantially before the final draft), along with some notes on possible implementation strategies.
Judy Hu: Provides reasonable algorithms for arithmetic operations and exception handling that implement the IEEE standard.
Apple: This paper is a forerunner to work on the draft Standard.

[7] Jerome T. Coonen. Underflow and the denormalized numbers. *Computer*, 14:75–87, 1981.

Apple: These two papers examine one of the major features of the proposed Standard, gradual underflow, and show how problems of bounded exponent range can be handled through the use of denormalized values. (The other paper referred to is [9].)

[8] Jerome Toby Coonen. *Contributions to a Proposed Standard for Binary Floating-Point Arithmetic*. Thesis (ph.d. in mathematics), Department of Mathematics, University of California at Berkeley, Berkeley, CA, USA, December 1984.

Apple: This thesis, developed alongside the standard itself, is a set of clarifications and elaborations of the terse 754 document; it is an aid to implementors and a demonstration that the implementation is feasible.

[9] James W. Demmel. Underflow and the reliability of numerical software. *SIAM Journal on Scientific and Statistical Computing*, 5(4):887–919, 1984.

Judy Hu: Examines the effects of different underflow mechanisms (mainly store zero and gradual underflow) on the reliability of numerical software.
Apple: These two papers examine one of the major features of the proposed Standard, gradual underflow, and show how problems of bounded exponent range can be handled through the use of denormalized values.
W. Kahan: ... Gradual underflow beats flush-to-zero

[10] David Hough. Applications of the proposed IEEE 754 standard for floating-point arithmetic. *Computer*, 14:70–74, 1981.

Apple: This paper is an excellent introduction to the floating-point environment provided by the proposed Standard, showing how it facilitates the implementation of robust numerical computations.

[11] IEEE Computer Society Microprocessor Standards Committee, Floating-Point Working Group. A proposed standard for binary floating-point arithmetic, Draft 8.0 of IEEE Task P754 (with introductory comments by David Stevenson). *Computer*, 14:51–62, 1981.

Apple: This is draft 8.0 of the proposed Standard, which was offered for public comment. The final standard is substantially simpler than this draft; for instance, warning mode and projective mode have been eliminated, and the definition of underflow has changed. However, the intent of the Standard is basically the same, and this paper includes some excellent introductory comments by David Stevenson, Chairman of the Floating-Point Working Group.

[12] W. Kahan. Why do we need a floating-point arithmetic standard? Technical report, University of California, Berkeley, CA, February 1981.

Self-described as "a travelogue about the computer industries arithmetic vagaries," this document gives examples of the challenges facing the would-be designer of portable numerical software, and discusses how the features of the IEEE standard might make the task easier. Though the electronic original was lost, a recently re-typed version will soon be available.

[13] Michael Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, Philadelphia, 2001.

This recently-published textbook describes the fundamental features of the IEEE 754 standard, and how these features can be used in Fortran 2000 and C99.

4 Arithmetic texts

The recommended arithmetic texts for the course were by Scott [5], Koren [3], and Parhami [4]. The appendix by David Goldberg [1] in Patterson and Hennessey's popular book provides a somewhat less in-depth treatment. A recent book by Oberlyn and Flynn looks interesting, but I have not read it, nor have I read or heard any reviews of it.

The two-volume set *Computer Arithmetic* [6, 7], edited by Swartzlander, is a collection of classic papers on computer arithmetic, many of them otherwise difficult to track down.

References

[1] D. Goldberg. Computer arithmetic. In *Computer Architecture: A Quantitative Approach by J.L. Hennessy and D.A. Patterson*, chapter Appendix A, pages A–1–A–66. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

> *Judy Hu*: An introduction to algorithms for fixed and floating point arithmetic, with emphasis on the IEEE 754 standard.

[2] K. Hwang. *Computer Arithmetic*. John Wiley and Sons, New York, 1979.

[3] Israel Koren. *Computer Arithmetic Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1993.

> A more detailed introduction to computer arithmetic than the one by Scott [5].

[4] Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

> A recent, very in-depth reference on computer arithmetic. Though this text goes into much more detail than the books by Koren [3] or Scott [5], it says some things about the IEEE standard which are incorrect: for instance, it characterizes gradual underflow as an optional feature.

[5] Norman R. Scott. *Computer Number Systems and Arithmetic*. Prentice Hall, Englewood Cliffs, NJ, 1985.

> A very short introductory text on computer arithmetic. This would have been the recommended textbook for the purposes of this course.

[6] Earl E. Swartzlander, Jr. *Computer Arithmetic*, volume 1 of *IEEE Computer Society Press tutorial*. IEEE, New York, 1990.

> A collection of classic papers in computer arithmetic, with a strong emphasis on computer hardware. Besides the expected papers on addition, multiplication, division, and computation of the elementary functions, there is a special section on logarithms and logarithmic arithmetic, and another section on "floating-point arithmetic" with papers largely treating the statistical analysis of floating point.

[7] Earl E. Swartzlander, Jr. *Computer Arithmetic*, volume 2 of *IEEE Computer Society Press tutorial*. IEEE, New York, 1990.

> A follow-up volume to [6], this collection of influential arithmetic papers includes sections on VLSI implementation of floating point hardware; error detecting and correcting codes and their use in error-tolerant arithmetic units; online arithmetic; and "number representation." The number representation section includes papers on floating point representation in various bases and on alternate representations like floating slash, continued fraction representations, and SLI.

5 Floating point hardware

The bi-annual IEEE Symposia on Computer Arithmetic and the journal *IEEE Transactions on Computers* both contain a wealth of articles on computer arithmetic, and particularly on computer arithmetic hardware. Unfortunately, I consider myself (at present) unqualified to judge their relative merits. The references in this section were lifted primarily from Judy Hu's notes. See also the volumes by Swartzlander referenced in the second section.

References

[1] D. E. Atkins. Higher radix division using estimates of the divisor and partial remainders. *IEEE Transactions on Computers*, C-17(10), October 1968. Reprinted in *Computer Arithmetic*, ed. by Swartzlander.

> *Judy Hu*: Reviews the theory of the SRT division technique, develops analytic expressions for determining the number of bits of divisor and partial remainder which must be inspected for a given radix.

[2] A. D. Booth. A signed binary multiplication technique. *Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951. Reprinted in *Computer Arithmetic*, ed. by Swartzlander.

> *Judy Hu*: Describes a two's complement multiplication algorithm that avoids the need to correct the product when either input is negative.

[3] T. C. Cheng and I. T. Ho. Storage-efficient representation of decimal data. *Communications of the ACM*, 18(1):49–52, 1975.

> *Judy Hu*: Presents a simple binary encoding scheme of decimal digits which achieves high storage efficiency.

[4] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965. Reprinted in *Computer Arithmetic*, ed. by Swartzlander.

Judy Hu: Describles the parallel multiplication process in more detail than Wallace's paper.

[5] R. Duncan. Numeric data processor: the intel 8087. *Dr. Dobb's Journal*, 7(8):47–49, 1982.

[6] P. Y. Lu and K. Dawallu. A VLSI module for IEEE floating-point multiplication/division/square root. In *Proceedings. 1989 IEEE International Conference on Computer Design: VLSI in Computers and Processors (Cat. No.89CH2794-6)*, pages 366–368, Washington, DC, USA, ???? 1989. IEEE Comput. Soc. Press.

Judy Hu: A hardware implementation of division and square root using Newton-Ralphson method and a rounding algorithm are presented.

[7] J. E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, EC-7(3):88–92, September 1958. Reprinted in *Computer Arithmetic*, ed. by Swartzlander.

Judy Hu Describes the SRT division method and presents examples Radix 4 and Radix 10 dividers.

[8] G. S. Taylor. Radix 16 SRT dividers with overlapped quotient selection stages. In Kai Hwang, editor, *IEEE 7th Symposium on Computer Arithmetic*, pages 64–71, 1985.

Judy Hu: Several SRT division methods and their implementation alternatives are compared for performance and costs.

[9] K. D. Tocher. Techniques of multiplication and division for automatic binary computers. *Quarterly Journal of Mechanics and Applied Mathematics*, 11(3):364–384, ???? 1958.

Judy Hu: A survey of some possible schemes for fast multiplication. Applies mathematical analysis to the problem of making fast multipliers and dividers.

[10] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, pages 14–17, February 1964. Reprinted in *Computer Arithmetic*, ed. by Swartzlander.

Judy Hu Introduces the notion of a fully parallel multiplier implemented with carry-save adders.

[11] T. E. Williams and M. A. Horowitz. A zero-overhead self-timed 160-ns 54-b CMOS divider. *IEEE Journal on Solid State Circuits*, 26(11):1651–1661, November 1991.

Describes several performance enhancements used to make this self- timed SRT division implementation one of the fastest to date.

6 Language support

Though every major hardware vendor now supports IEEE arithmetic, language implementors have been much slower to provide the support necessary for programmers to easily exploit the capabilities of their hardware. Many of the issues involved in proper support at the language level were explored in the early days of the standard [7, 11, 6]. Often these concerns often still remain unaddressed two decades later.

When language support is available, it is possible not only to write more robust codes, but also to write codes that run more quickly. One paradigm which remains nearly impossible to write portably is to run a fast common-case computation and only recompute more slowly and carefully if an exceptional condition occurs in the fast computation. This paradigm was described in [4], and also in [10], where it was used in elementary function routines. Hauser [8] investigates language mechanisms for handling IEEE exceptions, and pays attention to how the "careful only when needed" paradigm might be portably supported.

The availability of multiple precisions in the IEEE standard is not only to allow programmers to trade speed for accuracy. In fact, the use of multiple precisions in a single code often increases both robustness and performance in surprising ways. Some examples are given in [12]. Expression evaluation policies and the use of mixed precision are treated in [1, 9, 5]. Related material on techniques for implementing and using arbitrarily extended precisions is covered later in this bibliography.

Specific proposals for how to support floating point in Java [2, 12] and languages such as Ada [3] have recently been proposed. The C99 language also provides much-improved support for IEEE 754.

References

[1] R. P. Corbett. Enhanced arithmetic for fortran. *ACM SIGPLAN Newsletter*, 17(12):41–48, 1982.

Also in ACM Signum Newsletter, 18.1 (1983) *Judy Hu* The problems of interpreting mixed-precision arithmetic expressions are introduced, a strategy – widest need evaluation – is discussed, and its application to Fortran is presented.

[2] Joseph Darcy. Borneo: Adding IEEE 754 floating point support to Java. Master's thesis, Computer Science Division, University of California

at Berkeley, 1998. `http://www.cs.berkeley.edu/~darcy/Borneo/spec.html`.

Describes an upward-compatible language extension to Java which provides structured access to IEEE 754 features to which the original Java specification forbade any access. See also [12].

[3] Samuel A. Figueroa del Cid. *A Rigorous Framework for Fully Supporting the IEEE Standard for Floating-Point Arithmetic in High-Level Programming Languages*. PhD thesis, Department of Computer Science, New York University, January 2000. `http://www.cs.nyu.edu/csweb/Research/Theses/figueroa_sam.pdf`.

Describes generally what languages should do to support floating-point. A set of interfaces is proposed for Ada, and Ada and Java are used for specific examples. Goes into less specific detail about tradeoffs and implementation issues than [2] does.

[4] James W. Demmel and Xiaoye Li. Faster numerical algorithms via exception handling. *IEEE Transactions on Computers*, 43(8):983–992, 1994.

Describes a programming paradigm in which a fast common-case routine which may over/underflow is run, exception flags are tested, and a slower code with careful scaling is run only if necessary. See also [8].

[5] Charles Farnum. Compiler support for floating-point computation. *Software—Practice and Experience*, 18(7):701–709, 1988.

Judy Hu: Addresses a number of issues that compiler writers should know in order to produce good floating-point code.
Apple This paper describes many of the things a compiler writer should know about floating point arithmetic.
W. Kahan: Advice to compiler writers from a classroom project.

[6] Richard J. Fateman. High-level language implications of the proposed IEEE floating-point standard. *ACM Trans. Program. Lang. Syst.*, 4(2):239–257, 1982.

Apple: This paper describes the significance to high-level languages, especially Fortran, of various features of the IEEE proposed standard.

[7] Stuart Feldman. Language support for floating point. In *The Relationship between Numerical Computation and Programming Languages*, pages 263–273. North Holland, Amsterdam, 1982.

Examines support for floating point in languages of the time, and considers features for future languages in the light of the IEEE

754 standard. Some of the suggestions made, such as structured access to modes and flags and widest-need expression evaluation, continue to be echoed (and too often ignored) two decades later.

[8] John. R. Hauser. Handling floating-point exceptions in numeric programs. *ACM Transactions on Programming Languages and Systems*, 18(2):139–174, 1996.

Examines the ways in which a conscientious programmer can avoid overflow and underflow exceptions, with an eye toward what gains the programmer could make if standardized exception handling constructs were available in programming languages. Without such support, techniques like those proposed in [4, 10] cannot be portably programmed.

[9] T. E. Hull, A. Abraham, M. S. Cohen, A. F. X. Curley, C. B. Hall, D. A. Penny, and J. T. M. Sawchuk. Numerical TURING. *ACM SIGNUM Newsletter*, 20(3):26–34, 1985.

Judy Hu Describes the Numerical Turing language with emphasis on its new features: clean decimal arithmetic and complete precision control of variables and operations.

[10] T. E. Hull, T. F. Fargrieve, and P. T. Tang. Implementing complex elementary functions using exception handling. *ACM Transactions on Mathematical Software*, 20(2):215–244, 1994.

See also [4]. Evaluates elementary functions using fast formulas which nearly always works, and uses exception handling to handle cases when scaling is needed or when an exception should be passed on to the user.

[11] W. Kahan and J. T. Coonen. The near orthogonality of syntax, semantics, and diagnostics in numerical programming environments. In *The Relationship between Numerical Computation and Programming Languages*, pages 103–115. North Holland, Amsterdam, 1982.

Judy Hu: Shows that language syntax, arithmetic semantics, and execution-time diagnostics are approximately independent features of the numerical programming environment, and suggests that each should be dealt with by experts in that area.
Apple: This paper describes high-level langauge issues relating to the proposed IEEE standard, including expression evaluation and environment handling.

[12] W. Kahan and Joseph Darcy. How Java's floating-point hurts everyone everywhere. `http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf`.

Describes problems with Java's support for floating-point computation, and gives some examples of how they manifest themselves. Some suggestions are proposed for how to remedy Java's floating point. Also in this document are written some "rules of thumb" for best use of moder floating-point hardware.

7 Elementary functions and approximation

References

[1] J. M. Borwein and P. B. Borwein. The arithmetic-geometric mean and fast computation of elementary functions. *SIAM Review*, 26(3):351–366, 1984.

A very readable and self-contained introduction to the use of the arithmetic-geometric mean sequence to compute elementary functions. Assumes no prior knowledge of the theory of elliptic functions.

[2] E. W. Cheney. *Introduction to Approximation Theory*. Chelsea Publishing Company, New York, 1982.

[3] W. J. Cody. Implementation and testing of function software. In Paul C. Messina and Almerico Murli, editors, *Problems and Methodologies in Mathematical Software Production*, volume 142 of *Lecture Notes in Computer Science*, pages 24–47. Springer-Verlag, Berlin, 1982.

[4] William J. Cody Jr. and William Waite. *Software Manual for the Elementary Functions*. Prentice Hall, Englewood Cliffs, NJ, 1980.

A classic reference for the computer approximation of the elementary functions. Though the software included was once particularly important, at a time when many standard elementary functions libraries were crummy, this book is now mostly useful for its treatment of methods of computer approximation. Associated with this book was the ELEFUNT test suite, still available at `http://www.netlib.org/elefunt/`.

[5] W. Kahan. Branch cuts for complex elementary functions or much ado about nothing's sign bit. In A. Iserles and M. J. D. Powell, editors, *The State of the Art in Numerical Analysis*, pages 165–211. Oxford University Press, 1987.

Judy Hu Discusses the impact of signed and unsigned zeros on complex arithmetic, and supplies formulas to compute complex-valued inverse elementary functions correctly and accurately on their slitted domains.

[6] Peter Markstein. *IA-64 and Elementary Functions: Speed and Precision*. Prentice Hall, Upper Saddle River, NJ, 2000.

Discusses the architecture of the IA-64 and methods to take advantage of IA-64 features for fast computation of elementary functions. Not only transcendental functions, but also hardware-assisted iterative division and square root algorithms are treated. Transcendental function approximation is handled entirely by tables of polynomials; CORDIC style methods are not treated.

[7] Jean-Michel Muller. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, Boston, 1997.

A modern treatment of a variety of algorithms for computing elementary functions. Unlike previous treatments (e.g. [4]), Muller does not assume pure software implementations, and discusses hardware support in various places.

[8] John R. Rice. *The Approximation of Functions*, volume 2. Addison-Wesley, Reading, MA, 1969 (vol. 2).

8 Major vendors

References

[1] P. H. Abbott, D. G. Brush, C. W. Clark III, C. J. Crone, J. R. Ehrman, G. W. Ewart, C. A. Goodrich, M. Hack, J. S. Kapernick, B. J. Minchau, W. C. Shepard, R. M. Smith, Sr., R. Tallman, S. Walkowiak, A. Watanabe, and W. R. White. Architecture and software support in IBM S/390 Parallel Enterprise Servers for IEEE floating-point arithmetic. *IBM Journal of Research and Development*, 43(5/6):723–760, 1999. `http://www.research.ibm.com/journal/rd/435/abbott.html`.

Describes a recent retrofit which allows IBM's latest mainframe to use IEEE arithmetic rather than the old hexadecimal system. The new capabilities are grafted onto the old data path, making the IEEE instructions slightly slower than their hex arithmetic equivalents.

[2] Apple Computer. *Inside Macintosh: PowerPC Numerics*. Available online at
http://developer.apple.com/techpubs/mac/PPCNumerics/PPCNumerics-2.html.

(David Bindel) Documents the evolution of the SANE environment. The first appendix, "SANE vs PowerPC Numerics", tells what has changed. Sadly, the extended precision available in SANE is replaced in PowerPC Numerics by doubled-double.

[3] Apple Computer. *Apple Numerics Manual*. Addison Wesley, Reading, MA, 1988.

(David Bindel) Documents the Standard Apple Numerics Environment (SANE), a high-quality floating point software and hardware environment once supported by Apple.

[4] R.K. Montoye, E. Hokenek, and S.L. Runyon. Design of the ibm risc system/6000 floating-point execution unit. *IBM Journal of Research and Development*, 34(1):59–70, 1990.

[5] J. Palmer and S. Morse. *The 8087 Primer*. Wiley, New York, 1984.

Describes the 8087 coprocessor architecture and how to program it. Also provides a good description of the rationale behind the arithmetic design. The 8087 design predates the final IEEE standard, and so has certain features (like projective mode) not found in the final draft.

[6] Sun Microsystems. *Numerical Computation Guide*, 1996. `ftp://192.18.99.138/802-5692/802-5692.pdf`
Also linked from `http://www.validlab.com/`.

Includes a description of the IEEE standard features and the way Sun's numerical computing environment accomodates them. David Goldberg's popular article is included as an appendix.

9 Floating-point models

References

[1] W. S. Brown. A simple but realistic model of floating-point computation. *ACM Transactions on Mathematical Software*, 7(4), 1981.

This model of floating point computation was much simpler than that of van Wijngaarden [4], and enjoyed success as one of the main inspirations for the treatment of floating point in Ada and in the Language Independent Arithmetic standard.

[2] ISO. Information technology– language independent arithmetic – part 1: Integer and floating point arithmetic, 1994. ISO/IEC 10967-1:1994(E). `ftp://crl.dec.com/pub/misc/lia-1-is.ps.Z`.

The successor to the LCAS, LIA-1 proposes a parameterized floating-point environment model similar in spirit to Brown's model. See also [3].

[3] W. Kahan. Analysis and refutation of the LCAS. *ACM SIGNUM Newsletter*, 26(3):2–15, 1991.

Also published in SIGPLAN Bulletin 27.1(1992):61-74.
Judy Hu Points out serious flaws in LCAS, analyses why it will discourage the promulgation of portable software.

[4] A. van Wijngaarden. Numerical analysis as an independent science. *BIT*, 6:66–81, 1966.

Judy Hu: Tries to define computer arithmetic upon real numbers in 32 axioms.

10 Error analysis

Nicholas Higham's book *Accuracy and Stability of Numerical Algorithms* [5] covers a great deal of error analytic lore, including tricks for highly accurate summations, issues relating to IEEE standard arithmetic, and the analysis of problems from numerical linear algebra. Besides being remarkably clear and thorough, particularly for an area with such a reputation for inscrutability, Higham's book has a bibliography of over a thousand entries, and his chapter end notes contain exposition on those references.

Higham's book is billed by SIAM Publishing as an excellent companion to the LAPACK guide []. LAPACK, a linear algebra package that superseded the well-respected LINPACK [3] and EISPACK [4] libraries, is very carefully engineered to work on a variety of platforms, not all of them IEEE compliant. The LAPACK guide, along with the working notes, describes some of the error analysis, software engineering, and general cleverness that went into making a package which is simultaneously reliable and portable.

References

[1] James W. Demmel. The geometry of ill-conditioning. *Journal of Complexity*, 3:201–229, 1987.

[2] James W. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Numerische Mathematik*, 51:251–289, 1987.

See also "Geometry of ill-conditioning"

[3] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM Press, Philadelphia, PA, 1979.

LINPACK has since been superseded by LAPACK, q.v.

[4] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler. *Matrix Eigensystem Routines—EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[5] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[6] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

Shows how to apply error analysis of the individual arithmetic operations to the error analysis of large-scale problems.

11 Compensated sums and simulated multiprecision

A variety of techniques for accurate summation are explored by Higham in a paper from 1993 [9], and more recently in chapter four of his book (cited elsewhere in this bibliography). Douglas Priest's thesis [21] and an earlier paper [20] demonstrate algorithms that allow simulated precision bounded only by under/overflow. Papers by Shewchuk [24, 22, 23] treat the use of simulated high precision in geometric computation; see also the notes of Kahan [10, 11]. Popular packages for extended precision computation include Brent's package [6], and more recently Bailey's package [3, 4, 1]. Another package is MPFR [18]. Use of extended precision in the new BLAS standard is described in [13].

References

[1] David Bailey. Multiprecision software directory. `http://www.nersc.gov/~dhbailey/mpdist/mpdist.html`.

Extended precision and multiprecision software and documentation.

[2] David H. Bailey. The computation of π to 29,360,000 decimal digits using Borweins' quartically convergent algorithm. *Mathematics of Computation*, 50(181):283–296, 1988.

Judy Hu: See pp. 287-289 for more discussion on multi-precision multiplication algorithms including Complex Fourier Transform and Prime Modulus Transform methods.

[3] David H. Bailey. Algorithm 719: Multiprecision translation and execution of FORTRAN programs. *ACM Transactions on Mathematical Software*, 19(3):288–319, 1993.
Online at `http://www.nersc.gov/~dhbailey/dhbpapers/dhbpapers.html`.

Judy Hu Describes the MPFUN multiprecision package of Fortran routines that perform a variety of arithmetic operations and transcendental functions on floating point numbers of arbitrarily

high precision. Includes a discussion of some advanced multiprecision multiplication algorithms employed.
W. Kahan: High Performance Multiprecision Package" Research Center ... Uses Fortran's REAL data-type for the multi-word integer argument upon which his otherwise conventional floating point is based. This package is still being refined and enhanced. Also cf. Brent (1978) (1991) in Progress dated Sept 16, 1991 special commments, up ordinary REAL and DOUBLE PRECISION at no cost but the time conversions, unlike Brent (1978).

[4] David H. Bailey. A fortran-90 based multiprecision system. *ACM Transactions on Mathematical Software*, 21(4):379–387, 1995.
Online at `http://www.nersc.gov/~dhbailey/dhbpapers/dhbpapers.html`.

More recent multiprecision software in the vein of [3].

[5] G. Bohlender. Floating-point computation of functions with maximum accuracy. *IEEE Transactions on Computers*, C-26(7):621–632, 1977.

W Kahan: "... One improvement was an elegant formalism by which to prove the convergence of distillation iterations. The other was a family of stopping criteria suitable for use when less than full accuracy is required in the final sum."

[6] Richard P. Brent. A Fortran multiple-precision arithmetic package. *ACM Transactions on Mathematical Software*, 4(1):57–70, 1978.

Judy Hu: Describes the MP package of Fortran subroutines for performing multi- precision floating point arithmetic, and some of the algorithms are presented.
W. Kahan: Describes the MP Package, which uses Fortran's INTEGER data type to implement the multi-word integer arithmetic upon which his otherwise conventional floating-poing is based. Also available through netlib. C.f. Bailey (1990-91)

[7] T. J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971.

Judy Hu: A technique is described for expressing multilength floating-point arithmetic in terms of single length floating-point arithmetic. See also D. Priest's article.

[8] Richard J. Fateman. The MACSYMA big-floating-point arithmetic system. In R.D. Jenks, editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*. ACM, New York, 1976.

Judy Hu: Describes the bigfloat data type and arithmetic system in MACSYMA.

[9] Nicholas J. Higham. The accuracy of floating point summation. *SIAM Journal of Scientific Computing*, 14(4):783–799, July 1993. `http://citeseer.nj.nec.com/higham93accuracy.html`.

Describes and analyzes several summation techniques, including compensated summation. Includes a thorough bibliography.

[10] W. Kahan. Miscalculating area and angles of a needle-like triangle. `http://www.cs.berkeley.edu/~wkahan/Triangle.pdf`.

[11] W. Kahan. What has the volume of a tetrahedron to do with computer programming languages? `http://www.cs.berkeley.edu/~wkahan/VtetLang.pdf`.

[12] W. Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965.

[13] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskander, A. Kapur, M. C. Martin, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision blas. Technical Report LBNL-45991, Lawrence Berkeley National Lab, 2000. Submitted to ACM TOMS.

[14] S. Linnainmaa. Analysis of some known methods of improving the accuracy of floating-point sums. *BIT*, 14:167–202, 1974.

W. Kahan: "A thorough analysis, validation, and comparison of tricks like these..."

[15] Seppo Linnainmaa. Software for doubled-precision floating-point computations. *ACM Transactions on Mathematical Software*, 7(3):272–283, 1981.

Generalized Dekker's algorithms, proofs, etc.

[16] Ole Møller. Note on quasi double-precision. *BIT*, 5:251–255, 1965.

[17] Ole Møller. Quasi double-precision in floating point addition. *BIT*, 5:37–50, 1965. See also [16] for remarks on this article.

[18] Mpfr library. `http://www.loria.fr/projets/mpfr/`.

Another multi-precision floating point library, the apparent successor of the corresponding module in GMP (the GNU Multi-precision Package).

[19] M. Pichat. Correction d'une somme en arithmétique à virgule flottante. *Numerische Mathematik*, 19:400–406, 1972.

W. Kahan: "It is astonishingly simple, without the presort and backward pass in my algorithm; yet his will distill a large number N of summands in not much more time on average than twice what mine takes. In the worst case his scheme can take time proportional to N^2, whereas the worst case for mine, though far better than that, has yet to be determined."

[20] Douglas M. Priest. Algorithms for arbitrary precision floating point arithmetic. In Peter Kornerup and David W. Matula, editors, *Proc. 10th IEEE Symposium on Computer Arithmetic*, pages 132–143. IEEE Computer Society Press, Los Alamitos, CA, USA, 1991.

Judy Hu: Presents techniques to perform computation of high accuracy using floating-point operations of limited precision. Points out that it is cost, not precision, that limits accuracy. *W. Kahan*: ... Portable, not fast

[21] Douglas M. Priest. *On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations*. PhD thesis, Mathematics Department, University of California, Berkeley, CA, November 1992. `ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z`.

See also [20]. Besides presenting algorithms for arbitrary precision and examining the cost of using (or sometimes of not using) simulated extended precision, Priest gives a critique of various models of floating-point, which in some cases are too pessimistic to accomodate the style of simulated higher precision he studies.

[22] Jonathan Richard Shewchuk. Robust Adaptive Floating-Point Geometric Predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery, May 1996. `http://www.cs.cmu.edu/~jrs/jrspapers.html`.

[23] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997. `http://www.cs.cmu.edu/~jrs/jrspapers.html`.

[24] Jonathon Shewchuk. Fast robust predicates for computational geometry. `http://www.cs.cmu.edu/~quake/robust.html`.

Contains pointers to [22, 23]. Describes fast algorithms based on Priest's techniques, and their use to test the sign of certain determinants describing, for instance, whether a point is inside or outside of a circle. The software is also available on this page.

12 Interval arithmetic and Kulisch-Miranker theory

References

[1] Göltz Alefeld and Jürgen Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983.

Judy Hu: A systematic introduction to the concepts and applications of interval analysis.
Apple: This book presents a mathematically rigorous description of interval arithmetic.

[2] J. H. Bleher, A. E. Roeder, and S. M. Rump. ACRITH: High-accuracy arithmetic. An advanced tool for numerical computation. In Kai Hwang, editor, *Proceedings of the 7th Symposium on Computer Arithmetic*, pages 318–321. IEEE Computer Society Press, Silver Spring, MD, USA, 1985.

Judy Hu The IBM ACRITH package of numerical software is introduced, its application environment and underlying 20-instruction facility are described. See the critique by Kahan and LeBlanc.

[3] J. H. Bleher, S. M. Rump, U. Kulisch, M. Metzger, Ch. Ullrich, and W. Walter. A study of a FORTRAN extension for engineering/scientific computation with access to ACRITH. *Computing*, 39:93–110, 1987.

Judy Hu Describes the language FORTRAN-SC, and features intended to facilitate engineering and scientific computation as well as the use of the ACRITH subroutine library.

[4] Paul Jansen and Peter Weidner. High-accuracy arithmetic software—some tests of the ACRITH problem-solving routines. *ACM Transactions on Mathematical Software*, 12(1):62–70, 1986.

Judy Hu Test results for most of the routines in the ACRITH package are presented, comments on the quality and reliability of these routines are given.

[5] W. Kahan. Interval arithmetic options in the proposed IEEE floating point arithmetic standard. In Karl L. E. Nickel, editor, *Interval Mathematics 1980*, pages 99–128. Academic Press, New York, 1980.

Judy Hu Exposes common misconceptions about computation, interval arithmetic, and the proposed IEEE standard. Explains some controversial features of that standard. *Apple* This paper attempts to allay certain widespread misconceptions about computer arithmetic.

[6] W. Kahan and E. LeBlanc. Anomalies in the IBM ACRITH package. In Kai Hwang, editor, *Proceedings of the 7th Symposium on Computer Arithmetic*, pages 322–331. IEEE Computer Society Press, Silver Spring, MD, USA, 1985.

Questions the reliability of ACRITH and its underlying methodology to manage extra-precise arithmetic.

[7] R. B. Kearfott. Interval computations: Introduction, uses, and resources. *Euromath bulletin*, 2(1). `http://interval.louisiana.edu/preprints/survey.pdf`, `ftp://interval.louisiana.edu/pub/interval_math/papers/`.

As the name suggests, this article provides an overview of interval analysis and areas where it has been successfully applied, along with a substantial list of additional sources of information on the topic.

[8] Ulrich W. Kulisch and Willard L. Miranker. *Computer Arithmetic in Theory and in Practice*. Academic Press, New York, 1981.

Judy Hu As the title suggests, the book deals with both the theory and implementation of computer arithmetic taking for granted the alleged advantages of a "super-accumulator".

[9] Ulrich W. Kulisch and Willard L. Miranker, editors. *A New Approach to Scientific Computation*. Academic Press, New York, 1983.

Includes a collection of papers about the concepts of a new theory of computer arithmetic including a "super-accumulator" (which forms the theoretical basis of ACRITH), and its software and hardware implementation.

[10] Ramon E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

Judy Hu A survey/text of the principal methods and applications of interval analysis.
Apple This is a good introductory book, helpful to someone implementing an interval arithmetic.

13 Alternate arithmetic systems

References

[1] M. G. Arnold, T. A. Bailey, J. R. Cowles, and J. J. Cupal. Redundant logarithmic number systems. In M. D. Ercegovac and E. Swartzlander, editors, *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, pages 144–151, 1989. Also available online via IEEE.

(W Kahan) Advocates fast but inaccurate subtraction

[2] G. Bohlender, W. Walter, P. Kornerup, and D. Matula. Semantics for exact floating point operations. In P. Kornerup and D. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 22–26, 1991. Available on line through IEEE.

(W Kahan) Once again the easier almost-right is not quite right.

[3] C. W. Clenshaw and F. W. J. Olver. Beyond floating point. *Journal of the ACM*, 31(2):319–328, 1984.

A new number system – level-index arithmetic – is proposed for computer arithmetic based on iterated exponential functions to eradicate over/underflow. See also Demmel's critique.

[4] George F. Corliss and Louis B. Rall. Automatic generation of Taylor series in Pascal-SC: Basic operations and applications to differential equations. In *Trans. of the First Army Conference on Applied Mathematics and Computing (Washington, DC, 1983)*, pages 177–209. ARO Rep. 84-1, U. S. Army Res. Office, Research Triangle Park, NC, USA, 1984.

[5] James W. Demmel. On error analysis in arithmetic with varying relative precision. In Mary Jane Irwin and Renato Stefanelli, editors, *Proceedings of the Eighth Symposium on Computer Arithmetic, Como, Italy*, pages 148–152. IEEE Computer Society, Washington, D.C., 1987.

Judy Hu Illustrates that nonconventional floating-point representations proposed by Clenshau/Olver and Iri/Matsui require extra effort in error analysis; they are not shortcuts to writing reliable numerical code. (W. Kahan) ... Don't try.

[6] James W. Demmel. On the odor of IEEE arithmetic. *NA Digest*, Volume 91, Issue 39, 1991. (Response to a message "IEEE Arithmetic Stinks" in Volume 91, Issue 33).

[7] Hozumi Hamada. A new real number representation and its operation. In Mary Jane Irwin and Renato Stefanelli, editors, *Proceedings of the Eighth Symposium on Computer Arithmetic, Como, Italy*, pages 153–157. IEEE Computer Society, Washington, D.C., 1987.

Judy Hu An internal representation called URR is proposed for real numbers. A variable length exponent part is used to "eliminate" overflow and underflow. See Demmel's critique.

[8] P. Kornerup and D. W. Matula. A bit-serial arithmetic unit for rational arithmetic. In M. J. Irwin and R. Stefanelli, editors, *Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, pages 204–211, 1987.

[9] P. Kornerup and D. W. Matula. Exploiting redundancy in bit-pipelined rational arithmetic. In M. D. Ercegovac and E. Swartzlander, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 119–126, 1991.

Judy Hu Algorithms that perform rational operations on finite continued fractions; see also Vuillemin (1990).

[10] F-S. Lai and C-F.E. Wu. A hybrid number system processor with geometric and complex number capabilities. *IEEE Transactions on Computers*, 40(8):952–962, 1991. Available online via IEEE.

(W. Kahan) "... Pipelined VLSI implementation with fast $\lg x$ and 2^x conversion, said to be faster for FFTs and curvelinear graphics because it converts to logs before multiplication, division, sqrt, or exponentiation, and then converts back to conventional floating-point. Its arithmetic is not really logarithmic because its user puts conventional floating point numbers in and gets them out.

[11] Shouichi Matsui and Masao Iri. An overflow/underflow-free floating-point representation of numbers. *J. Information Processing*, 4(3):123–133, 1981.

Judy Hu See Demmel's critique.

[12] David W. Matula and Peter Kornerup. Finite precision rational arithmetic: Slash number systems. *IEEE Transactions on Computing*, C-34(1):3–18, 1985.

Judy Hu Specifies the fixed-slash and floating-slash number systems and the exact rational and approximate real arithmetic they support.

[13] L. B. Rall. Differentiation in pascal-sc: Type gradient. *ACM Transactions on Mathematical Software*, 10:161–184, 1984.

Judy Hu Shows how automatic differentiation can be carried out in a modern computer language which permits user-defined operators and data type.

[14] P. R. Turner. Implementation and analysis of extended SLI operations. In *Proc 10th IEEE Symp Computer Arithmetic*, pages 118–126, 1991.

(W. Kahan) ... Misplaced ingenuity

[15] Jean Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, 1990.

Judy Hu Vuillemin, Jean. "Exact Real Computer Arithmetic with Continued Fractions." in Proceedings of the 1988 ACM Conference on LISP and Functional Programming, 14-27. New York: ACM Press, 1988. Introduces a representation of the computable real numbers by continued fractions and the general algorithms for performing arithmetic operations on them. (W Kahan) ... Algorithms that perform transcendental as well as rational operations upon non-terminating continued fractions.