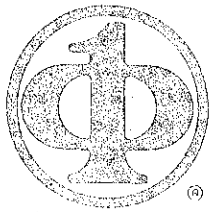
ANOMALIES IN THE IBM ACRITH PACKAGE

W. Kahan and E. LeBlanc

Reprinted from IEEE Proceedings of
7th SYMPOSIUM ON COMPUTER ARITHMECTIC
June 4-6, 1985 Urbana, IL



IEEE COMPUTER SOCIETY
1109 Spring Street, Suite 300
Silver Spring, MD 20910

ANOMALIES IN THE IBM ACRITH PACKAGE

W. Kahan and E. LeBlanc

Mathematics Department
University of California at Berkeley
Version dated Mar. 13, 1985

ABSTRACT

The IBM ACRITH package of numerical software is advertised as reliable and easy to use; but sometimes its results must astonish or confuse a naive user. This report exhibits a few of the surprises. For instance, a finite continued fraction, easy to evaluate in two dozen keystrokes on a handheld calculator, causes ACRITH to overflow either exponent range or 15 **Mega**bytes of virtual memory. Lacking access to source code, we must speculate to explain the anomalies. Some seem attributable to small bugs in the code; some to optimistic claims or oversimplifications in the code's documentation; some to flaws in the doctrine underlying the code. We conclude that different techniques than used by ACRITH might have been about as accurate and yet more economical, robust and perspicuous.

Introduction

This is a report, preliminary and very likely to change, on the "High-Accuracy Arithmetic Subroutine Library" ACRITH offered by IBM for use with their VS Fortran. An imminent new release of ACRITH may not have the same bugs as we found when we ran Version 1 Release 1 Modification Level 0 of ACRITH on our IBM 3081K. And our speculative inferences, necessitated by lack of access to source code, may have to be revised should more information about the algorithms actually used in ACRITH be released after this report's first versions appear.

The laudable intent of ACRITH appears to be the provision of reliable numerical results, accurate to very nearly the last digit displayed, for a selection of numerical tasks encountered by people whose expertise, however refined in other areas, does not include numerical error analysis. Ideally, results should be correct regardless of roundoff, cancellation, ill-condition or numerical instability. ACRITH employs two strategies to approach this ideal. First, to deliver for each desired result either an interval that surely contains it, or else a warning that no such interval could be found, ACRITH uses Interval Arithmetic, a facility described in several books (Alefeld and Hertzberger[1], Moore[2] and Rokne and Ratschek[3]) and not controversial. Second, to deliver narrow intervals that define its results accurately, ACRITH must exploit extra-precise arithmetic, for which the necessity is indisputable.

What *is* controversial is the way ACRITH performs extra-precise arithmetic. It requires a *Super-Accumulator* to calculate scalar products exactly, as prescribed in the book [6] by Kulisch and Miranker, and applies it according to a methodology described in an IBM Symposium [7] edited by them. Also disputable are some of the claims in the promotional literature [9, 10] for ACRITH; it does not always solve problems so cleanly as that literature suggests. Here are examples:

Matrix Multiplication — ACRITH always computes a product of two matrices accurately with every element of the product correctly rounded; that is the foundation of ACRITH's methodology. Now suppose we needed comparable accuracy in the final product of several matrices. It could not be achieved by simply invoking ACRITH's matrix multiply repeatedly; trying that could produce utterly inaccurate results because the roundoff in the first multiplication, albeit correctly rounded, could be amplified by subsequent multiplications. Something else has to be done. In accordance with the methodology advocated by Kulisch and Miranker [8], Rump [11] and Böhm [4] for computing polynomials, we tried to calculate the matrix product by solving an enormous triangular system of linear equations, but that took hundreds of times too long and delivered inaccurate results. Finally we devised a quick and short but devious method to get correct results; but is most unlikely to occur to the naive users targeted as potential customers for ACRITH.

Matrix Inversion — The accurate inversion of a 21×21 Hilbert matrix, with its condition number over 10^{29}, is a prodigious accomplishment for a program that allegedly carries only about 16 significant decimals for all intermediate variables. In demonstrations at symposia like the one at the Mathematics Research Center in Madison Wisconsin in Sept. 1984, in ACRITH's General Information Manual [9, pg.31], and in scientific papers on its methodology (Rump, [7, pg.53, pg.68]), that inversion is portrayed as typical of the way ACRITH triumphs over problems that would confound conventional

methods. But when we shuffled the columns of that Hilbert matrix ACRITH could not invert it even though shuffling columns does not affect the condition of a matrix. ACRITH refused to invert a 7×7 matrix that was inverted accurately in each of single-, double- and quadruple-precision arithmetic by standard LINPACK [**14**] library programs. ACRITH either refused to invert or obtained pessimistic estimates for inverses of certain 3×3 matrices that are all obviously well-conditioned by its own definition [**10**, pg.10]. None of the matrices mentioned so far is typical of matrices that walk in off the street every day to be inverted, but they are all good examples of the pathologies that justify ACRITH's existence. Neither can the results cited above be declared typical of the way LINPACK or ACRITH might handle similar examples. We know LINPACK's success with the 7×7 example was a fluke because we know from LINPACK's published source code how it typically copes with various pathologies. ACRITH's typical behavior is unpredictable because its source code is unavailable for critical scrutiny.

Real Zeros of Polynomials — The ACRITH procedure DPZERO purports to locate a real zero of a polynomial given its coefficients and a guess at the zero. What is claimed for the procedure [**10**, pg.54] is that it will bracket the zero inside a relatively *narrow* interval unless the procedure reports that such an interval cannot be found. We found a polynomial of degree 13 whose real zero 1/3 matched our guess to several significant decimals, but DPZERO produced the *wide* interval $[1/9\ ,\ 1/\sqrt{3}]$ as a result without comment. We got better results than that on a handheld calculator.

Arithmetic Expressions — ACRITH contains two utility programs, FTRANS and FEVAL, that purport to evaluate arbitrary rational expressions to "high accuracy" [**9**, pg.47] (about 15 significant decimals) despite whatever cancellation may occur in intermediate calculations, provided no over/underflows occur. However, relatively simple polynomials and rational functions, expressible with fewer than 50 keystrokes, can cause ACRITH to overflow 15 Megabytes of memory; and simpler rational functions can cause ACRITH to fail because of floating-point overflow beyond about 10^{76} even though the desired rational function and all its subexpressions lie between 0.001 and 1000. The only pathology in these examples is that ACRITH doesn't like them.

Speed — To achieve more reliable computation than they thought was afforded by conventional modalities, the designers of ACRITH were willing to sacrifice some speed. The extent of that sacrifice is hard to gauge, and hard to reconcile with their claims [**9**, pg.35] that ACRITH is usually not much worse than six times as slow as conventional calculation. The rational expressions mentioned above took thousands of times longer for ACRITH to compute than would have been required for quadruple-precision. Worse than that is the unreproducibility of execution times in the face of what might appear otherwise to be inconsequential changes in data. We found a well-conditioned matrix that ACRITH inverts in times that change by a factor larger than ten when the matrix is reflected in its skew diagonal, although the effect of this reflection upon the inverse is merely to swap its first and last diagonal entries.

Evidently ACRITH is neither efficient nor reliable enough to be embedded in an engineering application and left unattended. The same might be said of much numerical software most of which would not repay further study. ACRITH is distinguished by extraordinary claims made for its reliability [**9**, pp.1,3,11,35; and **10**, pp.iii,11] and for the rationale [**9**, pg.3; and **11**] behind its methods. We seek to understand how that rationale helps or hinders progress towards ACRITH's goals.

Multi-Precision vs. ACRITH

To keep this report self-contained, we shall not resort to the terminology introduced by Kulisch and Miranker [**6** and **7**] but shall instead describe how ACRITH's approach to extra-precise calculation differs from conventional wisdom.

Adequate support for certifiably reliable computation cannot possibly be derived from only the data types ARRAY, INTEGER/REAL/COMPLEX, and SINGLE/DOUBLE PRECISION proffered by the most popular computer languages. They lack the essential ingredient INTERVAL. And if, instead of merely announcing that a problem is too nearly pathological, we wish to solve it, then the extra ingredient we need is EXTENDED PRECISION, extended to an extent coarsely controllable by a programmer. Note that no amount of high precision is enough by itself to guarantee correct results; error-analysis is necessary too [**16**], and *automatic* error-analysis is practical only with Interval Arithmetic. Increasing precision, like increasing memory, serves solely to diminish the set of problems that cannot be solved, and hence must submit to a law of diminishing returns. The probability that higher precision will be needed to solve problems declines exponentially with the wordlength, whereas the time consumed grows superlinearly, especially for division. To compute efficiently, programmers must employ high-precision arithmetic parsimoniously.

If a programmer knew in advance where high precision will be needed and how much will suffice, he could use compile-time declarations to allocate statically as much memory as that needed precision requires. But then, if he guessed wrong, he might have to recompile with revised declarations and recompute. Another approach is to determine adaptively at run-time how much precision is needed, which entails dynamic memory management and its associated overheads, which matter only when the precision in use is not very high, which is most of the time. Evidently, managing high precisions efficiently can be complicated.

Kulisch and Miranker [6 and 7] would cut through those complications. Numerically naive computer users are to be provided with a library of pre-programmed procedures for all their computational needs. The implementors of that library are to be protected from needless complexity by a methodology that forbids explicit mention of any higher precision than is being used to store input data and to display final results. Therefore extra-precise calculation must occur surreptitiously, hidden within the subterfuges enumerated in the following section. Only the first of them is untainted by controversy.

ACRITH's Methodology

1. Iterative Refinement is used to compute, in each iteration, a correction term that refines the accuracy of what was computed in the previous iteration.
2. Every variable is approximated implicitly to extra precision by an unevaluated sum of correction terms, each term much tinier than its predecessor, and each represented exactly to the same precision as the data. Only as much precision as is needed appears to be provided, perhaps limited by a prior static allocation of memory for the terms; but the scheme is obliged to re-evaluate the same scalar products repeatedly during iterative refinements, so it does waste time.
3. Iterative refinement requires residuals that reveal how much an approximate solution dissatisfies the equations to be solved. By a mechanical translation process, each variable can be replaced symbolically by the sum of corrections that approximates it, and then residuals can be expressed in terms of scalar products of correction terms. A Super-Accumulator is introduced to evaluate such scalar products exactly, despite massive cancellation, before rounding them to the same precision as the data; therefore residuals can almost always be calculated as accurately as needed to obtain one more correction term. But the translated expressions are cumbersome.
4. The super-accumulator is the only site in ACRITH where extra-precise arithmetic is performed explicitly. Moreover, in its pristine form the methodology prohibits any reference to the super-accumulator other than to round off the exact value of a scalar product. Consequently multivariate polynomials cannot be evaluated with the aid of products of extra-precise intermediate results, nor by rapid recurrences with arrays of extra-precise accumulations, but must be calculated slowly via nested iterative refinements using at any instant only one extra-precise entity, the super-accumulator. Finally, since the super-accumulator sums products but cannot cope with quotients, almost every rational function must be first transformed into a ratio of polynomials before ACRITH can evaluate it accurately regardless of how inauspicious that transformation may be.

The foregoing overview of our view of ACRITH would probably not persuade a champion of ACRITH's methodology to change his mind, but it may explain why we searched where we did for anomalies that, once found, undermine confidence in ACRITH's reliability.

How reliable is ACRITH?

Software is considered reliable to the extent that it conforms to reasonable expectations. Ideally, reliable numerical software never misleads, almost always works efficiently, and fails to cope efficiently only with problems that lie near or beyond the boundary of what is economically feasible with the resources available. But this ideal is too much to expect in general. At best, this ideal can be approached to a degree that depends upon the prowess of programmers, the clarity of explanatory documentation, the modesty of promotional claims, and the skills and perceptions of the software's users. They form a picture of the software's effective domain, bounded perhaps by a no man's land in which performance is unpredictable. Typical examples, good and bad, can serve as landmarks to delineate that domain experimentally when it cannot be revealed by analysis.

Lacking source code to analyse, we cannot describe ACRITH's effective domain with confidence. If we were told only that

> "The key feature of any of the algorithms of the ACRITH Subroutine Library is that all results are absolutely reliable; that means that no false result is possible." [**10**, preface],

or if the error codes listed in its documentation were our only guide, we might infer that ACRITH delivers simply either an error code or else an interval containing each desired numerical result. But Interval Arithmetic without a Super-Accumulator already does that, especially if nobody cares whether intervals are as accurate (narrow) as the data warrants, whether results appear quickly, whether correct usage requires special skills, nor whether error messages appear often when the data has no intrinsic pathology. ACRITH's architects aimed higher.

> "The routines provide solutions with high accuracy which could not be achieved previously with conventional means. High accuracy means that maximum tolerances for the exact solution of a specified problem are delivered which differ only in the last figure of the mantissa." [**10**, preface].

That should allay any qualms about accuracy. As for speed, ...

> "The run time performance of the ACRITH routines is generally of the same order of magnitude as of conventional ones. ... For instance, a run time increase by a factor of 6 to 8 relative to a widely used conventional routine was measured for solving a system of linear equations of order 100 in double precision." [**9**, pg.35].

The factor of 6 is roughly what S. Rump [5, pg.41] predicts from theoretical considerations. Who can use ACRITH?

> "...every FORTRAN programmer who is writing programs to solve linear and linearized problems of numerical algebra. ... A reasonably experienced FORTRAN programmer with some background in numerical algebra ..." [9, pg.4-5].

No mention of error analysis as a required skill. On the contrary,

> "If the situation is prohibitively ill-conditioned the user will be told by a return code and an error message of his routines. Thus he will always be safe in his computations." [9, pg.11]
> ... "If overflow occurs in an algorithm, usually no result can be computed. This condition is caught with the error indicator IER. Underflow generally does not exclude the computation of a correct result of an algorithm, but may influence the accuracy of the result - that means the width of the resulting intervals. This condition is not considered an error." [9, pg.33].

Apparently underflow may degrade ACRITH's accuracy to the extent that, for instance, a calculated value x that should be zero may instead be located in an interval $-\epsilon < x < \epsilon$, where the underflow threshold $\epsilon = 16^{-65} \simeq 5.4 * 10^{-79}$. Otherwise there is no indication that ACRITH might produce an error message that the data does not deserve.

These claims for ACRITH and its methodology are formidable; and they are backed by widely published [4, 6, 7 and citations therein] examples over all of which ACRITH triumphs, whereas conventional methods fail allegedly for lack of a super-accumulator. Are these triumphs typical of the results we could all enjoy if we adopted its new arithmetic methodology?

Triple Matrix Product, P = ABC

ACRITH's procedure DMAMB computes the product of two matrices to full accuracy, but no procedure built into ACRITH computes the product of three matrices accurately. Can such a procedure be devised easily?

Three matrices A, B and C were constructed with ostensibly random integer entries subject to the requirement that $ABC = 0$ but $AB \neq 0$ and $BC \neq 0$. These matrices were scaled to possess large entries, but not so large as to cause overflow while ABC was being calculated. Here they are:

$$A = \alpha \begin{pmatrix} 12 & 13 & -1 & 11 \\ -23 & -34 & 11 & -12 \\ -1 & -5 & 4 & 3 \\ 24 & 35 & -11 & 13 \end{pmatrix} \quad B = \beta \begin{pmatrix} -11 & 5 & -6 & -10 \\ 9 & -2 & -1 & 10 \\ 7 & -3 & 0 & 6 \\ 3 & -5 & 10 & -1 \end{pmatrix}$$

$$C = \gamma \begin{pmatrix} -1 & -5 & 4 & -3 \\ -11 & -2 & 14 & 6 \\ -10 & 3 & 10 & 9 \\ -9 & 8 & 6 & 12 \end{pmatrix} \quad \text{where } \begin{aligned} \alpha &= 2058788401083655 * 16^{10} \\ \beta &= 6550690367084357 * 16^{10} \\ \gamma &= 5146971002709137 * 16^{10} \end{aligned}$$

Consider the calculation of $P = ABC$. The straightforward approach calculates $T := AB$, then $P := TC$, where the products are performed using the ACRITH matrix multiply routines DMAMB and DIMAM, and stored to double precision. This method's results, presented in the first line of table "$P = ABC$" below, are very inaccurate because actually $T = AB - R$, where R consists of roundoff, so the value computed for P is $ABC - RC = -RC$, which is huge.

Since $P = ABC$ is a polynomial in the elements of A, B and C, ACRITH's methodology can be invoked to evaluate P by solving a lower triangular system of equations, using iterative refinement to achieve full accuracy. Here is such a system:

$$\begin{pmatrix} 1 & 0 & 0 \\ B & -1 & 0 \\ 0 & A & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ P \end{pmatrix} = \begin{pmatrix} C \\ 0 \\ 0 \end{pmatrix}$$

where the entries are all square matrices. The accuracy and the speed with which ACRITH calculated the matrix product P using this method are tabulated in the second line of table "$P = ABC$" below, in which $\epsilon = 16^{-65}$ is the smallest positive number representable in double precision. Evidently both the time taken and the accuracy achieved are disappointingly worse than a naive user might have expected from ACRITH's documentation. What has gone wrong?

We suspected that ACRITH did not notice that the system was triangular but, instead, performed Gaussian elimination with pivotal exchanges, thereby converting the equations into something grossly ill-conditioned. Consequently, iterative refinement was slow to converge and stopped prematurely.

To avoid what we thought had gone wrong, we reformulated the problem as an upper triangular system

$$\begin{pmatrix} -1 & A & 0 \\ 0 & -1 & B \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} P \\ Y \\ X \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ C \end{pmatrix}.$$

The results of this calculation are presented in the third line of table "$P = ABC$" below. The time was much better, but the accuracy much worse.

Here is a better way to use ACRITH's matrix multiply procedures to compute $P = ABC$. Define:

$T := AB(\textit{rounded})$; $\quad R := (\,A \;\; -1\,)\begin{pmatrix} B \\ T \end{pmatrix}$; and

if $(\,A \;\; -1 \;\; -1\,)\begin{pmatrix} B \\ T \\ R \end{pmatrix} = 0$ then $P := (\,T \;\; R\,)\begin{pmatrix} C \\ C \end{pmatrix}$

else $\cdots$ (subsequent refinements were not needed).

These formulas conform to the Kulisch-Miranker discipline, but at the cost of computing AB three times. We rearranged the formulas slightly to compute AB only once by referring directly to the super-accumulator; this

renegade algorithm is the one whose results appear in the last line of table "$P = ABC$" below.

$P = ABC$: Triple Matrix Product Calculations		
Method	Typical Element of P	Execution Time
$(AB)C$	$[-10^{69}, 10^{69}]$	.006
Lower Triangular	$[-10^{31}\epsilon, 10^{31}\epsilon]$	3.07
Upper Triangular	$[-10^{41}\epsilon, 10^{41}\epsilon]$	.34
Renegade	0.0	.007

Times are in seconds on an IBM 3081K.

The elements in table "$P = ABC$" above are typical of the elements found in P after the various methods were tried; the intervals returned by the first three methods were all within a factor of about 10 of the typical element shown there, and for the fourth method P was exactly zero.

The calculation of P illustrates a problem that arises whenever two of ACRITH's procedures are composed. Even if the errors in the first procedure's output are confined to the last digit delivered, they can be so magnified by the second procedure as to swamp the desired result. That is what happened when the error R in the first product AB was expanded into RC by the second product. To avoid such error magnification, the first procedure's result could be passed as a multi-precision entity, a series of corrections, to the second procedure; but ACRITH's procedures are not designed to accept multi-precision inputs. Instead, a new procedure has to be devised from scratch to accomplish the desired result. We think this problem exposes a fundamental flaw in ACRITH's methodology; anyway, it might well defeat a naive user.

Inverting The Hilbert Matrix

When n is large the n-dimensional Hilbert matrix H is well known to be extremely ill conditioned. The elements of H are $(H_{ij}) = (k/(i+j-1))$ for $1 \le i,j \le n$. Strictly speaking k should be 1 but, to avoid distracting rounding errors when the elements of H are constructed in the computer, it is customary to choose $k = lcm(1,2,3,...,2n-1)$. Here lcm is the least common multiple function. This definition makes the entries of H integers representable exactly provided n is not too large. The elements of kH^{-1} are all integers too, and their calculation is a standard test for floating-point matrix inversion routines.

Among the "Typical Scenarios" in ACRITH's promotional literature [**9**, pg.31], and among the examples that are cited [**11**, pg.68] to vindicate the Kulisch-Miranker methodology, pride of place belongs to ACRITH's inversion of a 21×21 Hilbert matrix H on an IBM machine carrying only about 16 significant decimals for all intermediate calculations except the scalar products in the super-accumulator. Since H has a condition number over 10^{20}, this is a phenomenal accomplishment if it is truly typical. Could it be an accident?

Exchanging columns of a matrix merely exchanges corresponding rows of its inverse with no change to its condition number. After we swapped columns 1 and 19, 2 and 18, 3 and 17, ... of H, ACRITH refused to invert it but signaled instead "The matrix is extremely ill-conditioned or singular; no inclusion could be computed." So ACRITH's performance upon H is not typical.

We cannot say what kind of performance is typical for ACRITH because we do not know what method it used to invert H. We do know that the algorithms Rump described in [**11**, pg.62, pg.65], and which might have been presumed to have produced the results for H^{-1} he presented three pages later, are not the algorithm ACRITH uses. We tried them; they cannot improve an approximation to H^{-1} by iterative refinement unless that approximation is H^{-1} itself exactly. Besides, during a conversation in Madison Wisconsin in Sept. 1984, Rump admitted that the results in his paper were not obtained by the methods described therein but by some others he could not describe because they were "Proprietary." We have tried to guess what ACRITH does and, with Dr. K.-C. Ng's help, have devised algorithms that iteratively refine triangular factors of H before using them to compute an approximation to H^{-1} that can subsequently be iteratively refined. Such an algorithm seemed capable of failing or succeeding capriciously depending upon the ordering of the columns of H. Much better algorithms are easy to find, especially by one willing either to exploit the IBM 370's quadruple-precision hardware or to access the super-accumulator in a renegade way contrary to the Kulisch-Miranker doctrine, but they are not the subject of this report.

Another Atypical Example

The numerous examples that show how ACRITH triumphs where mundane methods fail are clouded by the suspicion that some of them may be atypical. Therefore a potential user must discover by trial and error what ACRITH can do. Such trials can mislead too. Consider the following matrix D devised along lines suggested by Prof. James W. Demmel of New York University's Courant Institute:

$$D = \begin{pmatrix} 1 & \rho & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & \rho \\ 0 & 0 & 0 & 1 & 0 & 0 & \rho^2/2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & -\rho^2/2 \\ \rho & -\rho^4/4 & \rho & \rho^2/2 & 1 & \rho^2/2 & 1 \end{pmatrix}$$

We set $\rho = 2^{-14}$ and submitted D to be inverted on the IBM 3081K in single-, double- and quadruple-precision respectively by the pairs of standard LINPACK [**14**] library routines *SGECO* and *SGESL*, *DGECO* and *DGESL*, and *QGECO* and *QGESL*. (The last two are obvious adaptations of the two before.) The computed

inverses were all correct to the last digit of their respective precisions, as was verified by comparison with an inverse computed symbolically. On the basis of this experience, D seems well-conditioned.

ACRITH's procedures *INV* and *DINV* [**10**, pg.65, pg.67] perform "Inversion of a single [double] precision point matrix with high accuracy." (A "point matrix" is one whose elements are all ordinary numbers rather than intervals.) Both procedures signaled that D is too ill-conditioned to permit an inverse to be calculated. How can this signal be reconciled with the correct inverses returned by LINPACK? LINPACK and ACRITH are both right!

The condition number of D is roughly $48/\rho^4 \simeq 3.5*10^{18}$, just beyond what can typically be inverted using IBM's double-precision format (14 significant hexadecimal digits) to hold triangular factors and other intermediate results. However, D is so contrived that in single- and double-precision LINPACK's rounding errors will cancel out, leading to an atypically correct result; and in quadruple-precision LINPACK commits no rounding error at all. On the other hand, ACRITH commits a rounding error forced upon it by the Kulisch-Miranker methodology, and D is so designed that the rounding error blossoms despite the super-accumulator. D is just that rounding error away from singular, so ACRITH's warning is deserved; almost all matrices so ill-conditioned as D lie beyond what we believe is ACRITH's effective domain.

Suppose a numerically naive programmer wished to invert matrices accurately; how might the foregoing results influence his choice between ACRITH and, say, LINPACK? Given results only for H (the Hilbert matrix of the previous section), or only for D, he might choose badly. Given both results, and results for H with columns swapped too, he would realize with diminished naivety that neither H nor D exhibits ACRITH's potency fairly. Too many of the examples published to promote ACRITH and its methodology seem unfair that way; they intimate that ACRITH never fails to deliver fully accurate results unless some extreme pathology in the data justifies a warning message instead. They afford no hint that a no man's land of innocuous problems may exist where ACRITH gives inaccurate results, or none, or takes far too long.

Two Well-Conditioned Matrices

The extent to which a matrix is well- or ill-conditioned can be measured in several mathematically equivalent ways. One is the condition number $\gamma(M) = \|M\|\cdot\|M^{-1}\|$, where $\|\cdots\|$ is some matrix norm that measures the overall magnitude of a matrix analogously to the length of a vector. A second is the relative nearness of M to singularity; the singular matrix S nearest M lies at a distance $\|S - M\| = \|M\|/\gamma(M)$ roughly. A third is the extent to which relatively small perturbations ΔM can become magnified by inversion; when $\|\Delta M\|$ is tiny enough, $\|(M + \Delta M)^{-1} - M^{-1}\|/\|M^{-1}\|$ can be about as big as $\gamma(M)\cdot\|\Delta M\|/\|M\|$ but not much bigger. This is quantitatively what the next quotation means:

> "If the solution value changes only moderately at a specified change of a certain data component the problem is *well-conditioned* with respect to this component. If a large change in the solution value is induced the problem is *ill-conditioned* with respect to this data component." [**10**, pg.10]

Regardless of how he understands this quotation, a programmer must find the next ones reassuring:

> "One of the unique advantages of the ACRITH Subroutine Library is the generation of highly accurate and verified solutions even in cases of extreme ill-conditioning. ... The observed changes in the computed solution values are exclusively due to the specified changes in the data. ... This will particularly cover the many cases where the ill-conditioning has inadvertently been introduced through an inconsiderate formulation of the mathematical model (for example, by poor scaling) but is not an intrinsic property of the situation." [**10**, pg.11].

Let λ be a tiny number: we used $\lambda = 16^{-8} \simeq 2.3*10^{-10}$. When λ is small the matrix M below is very well-conditioned because $\gamma(M) = \|M\|\cdot\|M^{-1}\| < 5$ for the commonplace norms:

$$M = \begin{pmatrix} 2\lambda^2 & -1 & 1 \\ -1 & -1 & -1 \\ 1 & -1 & 0 \end{pmatrix}, \quad M^{-1} = \frac{1}{2\lambda^2 - 3}\begin{pmatrix} 1 & 1 & -2 \\ 1 & 1 & 1 - 2\lambda^2 \\ -2 & 1 - 2\lambda^2 & 1 + 2\lambda^2 \end{pmatrix}$$

Changing M very slightly to $M + \Delta M$ changes M^{-1} very slightly to $(M + \Delta M)^{-1} = M^{-1} + M^{-1}\Delta M M^{-1} + \cdots$. For instance, varying the elements of M each in its twelfth significant decimal will vary the elements of M^{-1} each in its eleventh or beyond, but not in its tenth. This well-condition was confirmed by ACRITH when we submitted the interval matrix $[(1 - 16^{-10})M, (1 + 16^{-9})M]$ to program DIINV, "Inversion of a double precision interval matrix with high accuracy" [**10**, pg.68], which produced an interval inverse whose interval elements each had endpoints agreeing to eleven significant decimals.

Let the diagonal matrix $\Lambda = \mathrm{diag}(1/\lambda, \lambda, \lambda)$ and define $T = \Lambda M \Lambda$. This amounts to scaling the rows and columns as if the units used for variables had been changed from, say, millimeters to miles, except that the scale factors in Λ are all powers of 16 to avoid rounding errors on a hexadecimal IBM machine. Consequently $T^{-1} = \Lambda^{-1} M^{-1} \Lambda^{-1}$ is obtained by the reverse scaling of rows and columns and should be computed as such despite roundoff. Indeed, T is as well-conditioned as M in the sense that varying the elements of T each in its twelfth significant decimal will vary the elements of T^{-1} each in its eleventh or beyond, but not in its tenth.

Therefore T is well-conditioned by ACRITH's criterion quoted above, and this should have been confirmed when we submitted the interval matrix $[(1 - 16^{-10})T, (1 + 16^{-9})T]$ to DIINV. Instead of intervals with endpoints agreeing to eleven significant decimals, as the data deserves, DIINV produced grotesque intervals with endpoints in all cases over five times too big and with opposite signs! No warning message. Not so accurate as was promised in the quotations above. What has gone wrong?

We think ACRITH's failure here was caused by three bugs, all founded upon a misunderstanding of the matrix norms. Ideally, the norm apt for any specific situation should have roughly the same value for all perturbations regarded as about equally (in)consequential. The usual norms, like the root-sum-of squares norm, have this property for most endfigure perturbations of M, but not for similar perturbations of

$$T = \begin{pmatrix} 2 & -1 & 1 \\ -1 & -\lambda^2 & -\lambda^2 \\ 1 & -\lambda^2 & 0 \end{pmatrix}.$$

In those norms, endfigure perturbations of T's lower right corner 2×2 submatrix look negligible compared with endfigure perturbations of the first row and column. To make the usual norms measure perturbations aptly, T must be rescaled until it more nearly resembles M; but ACRITH does not rescale correctly, if at all. Consequently, ACRITH selects the element "2" as a pivot during Gaussian elimination, causing rounding errors that wipe out λ^2; this is the first bug. Wiping out λ^2 makes T look singular, or very ill-conditioned, so iterative refinement converges too slowly and terminates prematurely with no warning; this is the second bug. The third bug is the omission, from ACRITH's documentation and claims quoted above, of any warning that scaling problems are handled no better by ACRITH than by most conventional linear equation solvers.

A Polynomial Equation

Bad examples make bad generalizations. Many of the examples, published to show where conventional floating-point arithmetic is inferior to ACRITH and the Kulisch-Miranker methodology, come with comparisons of results obtained from simple programs versus results obtained for the elaborate programs that implement that methodology [**7**, pg.30-46, 53, 66-68, 73, 79, 134-136; **9**, pg.7-9; **8**, pg.14-18; **12**]. At times the comparisons resemble an attempt to infer, from the observation that a Diesel locomotive can pull a heavier train than a coolie, that the coolie must be weaker than the locomotive's engineer. But two of the cited publications, [**8**, pg.15] and [**12**, pg.168], invite comparisons with handheld calculators. That sounds like a sporting proposition.

$$P(x) = 1594323x^{13} - 6908733x^{12} + 13817466x^{11} - 16888014x^{10} + 14073345x^9 - 8444007x^8 + 3752892x^7 - 1250964x^6 + 312741x^5 - 57915x^4 + 7722x^3 - 702x^2 + 39x - 1$$

is a polynomial that reverses sign at $x = 1/3$ and vanishes nowhere else. ACRITH has a program to locate x:

> "The subroutine DPZERO computes bounds ZL, ZR of high accuracy for a real zero of the polynomial P which is near a specified approximation ZETA. That means that P(ZL) and P(ZR) are not of equal sign and that the difference between ZL and ZR is small. ... If the zeros of P are extremely ill-conditioned (if two or more zeros are between two successive floating-point numbers) the inclusion may fail." [**10**, pg.54]

(DPZERO issues IER=3 to signal that "Inclusion failed" whenever it cannot find values ZL and ZR that straddle a place x where $P(x)$ reverses sign.)

We submitted P and various guesses ZETA to DPZERO; from values like 0.333...3333 and 0.333...3334 for ZETA we got always an interval like $[1/9, 1/\sqrt{3}]$ for [ZL, ZR]. Although that interval does contain the zero $x = 1/3$, its endpoints are inexplicably farther from the zero than the first guess ZETA. No error was signaled; DPZERO set IER=0 to indicate "Normal end."

We must expect to lose some accuracy because $P(x) = (3x - 1)^{13}$. Therefore, if roundoff causes $P(x) \pm \epsilon$ to be computed in place of $P(x)$, we should expect estimates like $(1 \pm \epsilon^{1/13})/3$ to turn up for the zero. In effect, we expect to get only about one thirteenth as many correct significant digits for a thirteenfold zero as are carried during the calculation of $P(x)$. If the super-accumulator's accuracy were unbounded there would be no loss of accuracy when the zero of P was calculated. However, values of $|P|$ below the underflow threshold $\epsilon = 16^{-65}$ are flushed to 0, so we expected estimates like $(1 \pm \epsilon^{1/13})/3 = (1 \pm 16^{-5})/3$ for the zero from DPZERO. We do not know what bug caused DPZERO to deliver $ZL = 0.1111\cdots$ and $ZR = 0.5773\cdots$ instead. Neither can DPZERO say anything about how many times $P(x)$ reverses sign in $ZL < x < ZR$ except that the number must be odd.

The Hewlett-Packard HP-71B handheld calculator [**15**] carries twelve significant decimals for its floating-point variables, and with its HP 82480 Math Pac ROM (part no. 5061-7226) it conforms to the proposed IEEE standard p854 [**13**] for floating-point arithmetic. For a start we submitted the coefficients of P to a program PROOT in the Math Pac to obtain estimates of all 13 (complex) zeros of P. As expected, the estimates all agreed with $x = 1/3$ to at least 12/13 of a significant decimal. Next, a BASIC program that computes $P(x)$ from its coefficients was submitted to a program FNROOT in the Math Pac, and the real zero of $P(x)$ was estimated from various starting guesses analogous to ZETA. Since directed roundings (OPTION ROUND POS and NEG) are built into the calculator, we used

them during the calculation of $P(x)$ to get lower bounds ZL ranging from about 0.26 to 0.27 and upper bounds ZR from about 0.42 to 0.43 for the zero, depending on the starting guesses. Thus have we verified by a computational mathematical proof that $P(x)$ reverses sign an odd number of times between ZL and ZR. For this polynomial $P(x)$ the HP-71B calculator's bounds are tighter than ACRITH's; we don't think that implies the calculator is the more powerful engine.

Rational Expression Evaluation

"The subroutines FTRANS, FEVAL and FDELET allow the evaluation of arithmetic expressions with high accuracy. ... This holds as long as no over- or underflow occurs." [**10**, pg.47] Any rational expression in FORTRAN syntax can be submitted to ACRITH's FTRANS procedure to be transformed into what is essentially a ratio of two multivariate polynomials for subsequent evaluation by FEVAL. Each polynomial is evaluated as the solution of a triangular system of formally linear equations ostensibly solvable to arbitrarily high accuracy by iterative refinement using a super-accumulator to calculate scalar products exactly. After the two polynomials have been computed accurately enough, their quotient may be obtained almost equally accurately with one division. Details may be found in [**4**]. This scheme underlies the claim [**8**, pg.49] that the Kulisch-Miranker methodology is applicable universally; "... polynomials and then arbitrary arithmetic expressions can be evaluated with maximum accuracy (the validation step included) ..." and then equations can be solved to evaluate algebraic functions, and then transcendental functions can be approximated as usual by algebraic functions. The obvious defect is that the same scalar products have to recalculated in the super-accumulator again and again during iterative refinement, but this defect is mild and avoidable by renegade algorithms like our *ABC* multiplication above that accessed the super-accumulator directly. Less obvious is the potential for overflow latent in the transformations. Here are examples.

Let $r(x)$ be the root nearest 1 of the cubic equation $(\ r - 1\ )(\ r - 1 + \ x)^2 = 1$. When x is big that root $r(x)$ is closely approximated from above and below respectively by the two continued fractions

$$f(x) = 1 + 1/(x + 1/(x + 1/(x + 1/(x + 1/x^2)^2)^2)^2)^2$$

and

$$g(x) = 1 + 1/(x + 1/(x + 1/(x + 1/(x + 1/(x + 1/x^2)^2)^2)^2)^2)^2.$$

Their values at $x = 5$ and $x = 17$ are displayed below.

Formula	at $x = 5$	at $x = 17$
HP-71B's $f(x)$	1.03937 732815	1.00345 88002
HP-71B's $g(x)$	1.03937 732811	1.00345 88002
$r(x)$	1.03937 73281139..	1.00345 880002251..
ACRITH's $f(x)$	1.03937 7328150982	Exponent Overflow
ACRITH's $g(x)$	15 Megabytes = Insufficient Virtual Storage	

The HP-71B evaluates $f(x)$ and $g(x)$ satisfactorily throughout $0 \le x \le \infty$, including values x like 10^{-499} and 10^{499}, and does so in less than a second for all but the most extreme x's. FTRANS took over twelve seconds on the IBM 3081K to translate $f(x)$ into a form that took FEVAL about two seconds to evaluate when it was not thwarted by overflow beyond $16^{63} = 7.237 * 10^{75}$ in some intermediate expression. Overflow would have thwarted FEVAL at $g(5)$ too had not FTRANS first exhausted 15 Megabytes and almost six minutes working on it.

To explain why such tame expressions turn so ferocious inside ACRITH, we need merely exhibit them as ratios of polynomials. $f(x) = 1 + Nf(x)/Df(x)$ and $g(x) = 1 + Ng(x)/Dg(x)$ where

$$Nf(x) = x^{60} + 28x^{57} + 354x^{54} + \cdots + 513096x^{30} \cdots + 198x^6 + 20x^3 + 1\ ;$$

$$Df(x) = x^{62} + 30x^{59} + 407x^{56} + \cdots + 1020523x^{32} + \cdots + 243x^8 + 22x^5 + x^2\ ;$$

$$Ng(x) = x^{124} + 60x^{121} + 1714x^{118} + \cdots + 3800176888923x^{64} + \cdots + 970x^{10} + 44x^7 + x^4\ ;$$

$$Dg(x) = x^{126} + 62x^{123} + 1831x^{120} + \cdots + 8277163248848x^{63} + \cdots + 881x^6 + 42x^3 + 1\ .$$

Obviously FEVAL overflows because $17^{62} > 10^{76} > 16^{63} < 10^{88} < 5^{126}$. But we have no idea of how FTRANS overran 15 megabytes.

Were examples like these hard to find, the defect they expose in ACRITH's methodology might be tolerable. We fear such examples may be abundant. Consider a familiar Chebyshev polynomial:

$$\begin{aligned} T_{64}(x) &= \cos(64\,arccos\ x) \\ &= \cosh(64\,arccosh\ x) \\ &= -1 + 2(-1 + 2(-1 + 2(-1 + 2(-1 + 2(-1 + 2x^2)^2)^2)^2)^2)^2\ . \end{aligned}$$

A handfull of keystrokes can calculate $T_{64}(x)$ on a handheld calculator, but only the last expression can be fed to ACRITH, and when that is done FTRANS munches for almost seven minutes on an IBM 3081K and then overflows 15 Megabytes' memory again. Naive programmers might feel less bewildered by the event if they saw

$$T_{64}(x) = 9223372036854775808x^{64} - 14757395258967641292 8x^{62} + \cdots + 6456334894356662059008x^{32} - \cdots - 2048x^2 + 1\ ,$$

but they would be no less appalled. And if overflow does not stop the computation, a programmer might wonder how to explain why FEVAL can take perhaps a million times longer than conventional quadruple-precision arithmetic to get a result no better for expressions like these.

(We obtained the explicit polynomial representation of the foregoing functions in two ways. We ran Prof. D. Stoutemeyer's muMATH-83, obtainable form the Soft Warehouse in Honolulu or from Microsoft in Bellevue WA, on an IBM PC; several minutes sufficed for each function. We also ran the local symbol manipulator VAXIMA on a DEC VAX 11/780 for a few seconds to confirm the results.)

A Timing Anomaly

How predictable are the times that ACRITH spends in its own programs? We tested the matrix inversion procedure DINV on two $n \times n$ matrices A and B whose inverses differ in what we thought is an inconsequential way, but it isn't. Let $A_{ij} := min(i,j)$ and $B_{ij} := n+1-max(i,j)$; each is obtained from the other by reflection in its skew diagonal thus, shown for $n=5$:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 4 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Their inverses are tridiagonal matrices related the same way:

$$A^{-1} = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}, \quad B^{-1} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

These inverses differ only in that their first and last diagonal entries are exchanged. However, the results computed by DINV differed in two other ways. The times DINV took to invert the matrices on an IBM 3081K are displayed here in seconds for $n=20$ and $n=40$:

Times for Inversion		
n	A^{-1}	B^{-1}
20	0.27	2.99
40	1.44	18.44

Why does ACRITH take over ten times as long over B as over A? Part of the reason becomes clear when the second way in which the inverses differ is noticed. A^{-1} is computed exactly, presumably because all intermediate results were integers and no rounding error was generated during Gaussian elimination. Therefore no iterative refinement was needed to clean up. But during the inversion of B Gaussian elimination generated rational numbers that had to be rounded off. Consequently the first estimate of B^{-1} was only approximate, and iterative refinement had to be carried out to get refined estimates. The final results are all tiny intervals, the tiniest possible, and quite satisfactory. Unfortunately, iterative refinement had to be carried to great lengths to push the off-diagonal elements down into underflow to achieve such near perfection. It is a pity that ACRITH could not know that we would not have cared in this instance had iterative refinement stopped much sooner.

In general, different users may have different notions of what accuracy is adequate. ACRITH cannot read their minds, so it is obliged to go for maximum accuracy every time. This matters a lot only when some component vanishes in an array of results, because then maximum accuracy is limited not by a modest number of significant digits to be displayed, but by the underflow threshold. Here is one aspect of accuracy management that cannot be brushed away; the problem is not just to find a way to calculate something as accurately as might be needed, but also to know when to stop.

Conclusions

ACRITH's perceived reliability, and therefore its utility, are jeopardized because malfunctions like overflow, inaccurate results, error warnings or extremely slow execution cannot be attributed correctly to some blameworthy pathology in the data. To say that ACRITH is correct in the sense usually understood for crude Interval Arithmetic, which is to say that when a result appears it is always an interval that contains the desired result, is not good enough; such a claim could mean no more than that a machine was wired to print out only the interval "[$-\infty$, ∞]". ACRITH promises more than that, more than could be claimed for crude quadruple-precision interval arithmetic, but too often delivers less.

ACRITH is not at all typical of IBM products but seems instead first to have escaped prematurely from a research project and then to have evaded quality controls. We think its bugs can probably be repaired, but most likely not without abandoning restrictions upon access to the super-accumulator imposed by a doctrine that forbids explicit mention of extra-precise variables. And then, partly because of experiments performed for us by P. Tang, we expect that ACRITH's goals will turn out to be achievable more economically with the aid of something like quadruple-precision interval arithmetic than with a super-accumulator. If that happens it will vindicate our judgment that ACRITH's methodology is generally not a good way to manage extra-precise arithmetic.

To prevent misunderstanding, we repeat that the management of extra-precise arithmetic is at the core of our disagreement with ACRITH's doctrine. Except for that, we share most of ACRITH's goals and much of its overall strategy:

- Iterative refinement using contractive mappings to make a good guess better;
- Extra-precise calculation as needed to get acceptably accurate results;
- Interval arithmetic to know for sure when results are accurate enough.

We think ACRITH's defects are consequences mostly of an unnecessarily complicated approach to its goals.

Our conclusions contrast starkly with the doctrine promulgated by Kulisch, Miranker and their disciples, and accoutred in a vast panoply of algebraic terminology and theorems. If we are right, where is the flaw in their theory? They have proved that their methodology is in principle ***sufficient*** to compute everything computable, but not that it is ***necessary*** nor that it is ***efficient*** nor that it is intellectually more economical than the lore in the vaster literature that precedes and surrounds theirs.

Their theory suffers from what might be called *Algebraic Intransigency*, a lack of a kind of transitivity; specifically, scalar products of scalar products of scalar products ... induce first numerical constipation in the one super-accumulator their doctrine allows, then symbolic constipation in the algebraic transformations their doctrine demands.

Acknowledgments

We have benefitted greatly by conversations with Dr. J. W. Demmel (now at New York University), Dr. K.-C. Ng, Prof. B. N. Parlett, and P. Tang at the University of California in Berkeley, and with Dr. A. DuBrulle, Dr. F. N. Ris and Dr. S. Rump of IBM. We would also like to thank the IBM Corporation Ltd. and the Digital Equipment Corporation for contracts and grants which partially supported this work.

References

Interval Arithmetic

[1] Alefeld G. and Herzberger, J.: "*Introduction to Interval Computations.*" Academic Press, New York (1974).

[2] Moore, R.E.: "*Methods and Applications of Interval Analysis.*" SIAM Studies in Applied Mathematics, SIAM, Philadelphia (1979).

[3] Rokne J. and Ratschek, H.: "*Computer Methods for the Range of Functions.*" Halstead Press, New York (1984).

ACRITH Methods

[4] Böhm Harald: "*Evaluation of Arithmetic Expressions with Maximum Accuracy,*" in [7] pp. 121-137.

[5] Kaucher E. and Rump S.M.: "*E-Methods for Fixed Point Equations* $f(x) = x$," Computing, Vol. 38, pp. 31-42, (1982).

[6] Kulisch U. and Miranker W.L.: "*Computer Arithmetic in Theory and Practice.*" Academic Press, New York (1981).

[7] Kulisch U. and Miranker W.L., eds.: "*A New Approach to Scientific Computation.*" Academic Press, New York (1983).

[8] Kulisch U. and Miranker W.L.: "*The Arithmetic of the Digital Computer.*" IBM Research Report RC 10580 (#47356) 6/15/84 (1984).

[9] High-Accuracy Arithmetic, Subroutine Library, General Information Manual, IBM Program Number 5664-185 (1983).

[10] High-Accuracy Arithmetic, Subroutine Library, Program Description and User's Guide, IBM Program Number 5664-185 (1983).

[11] Rump S.M.: "*Solving Algebraic Problems with High Accuracy,*" in [7] pp. 51-120.

[12] Rump S.M.: "*Wie zuverlässig sind die Ergebnisse unserer Rechenanlagen?,*" Jahrbuch Überblicke Mathematik, pp. 163-169, (1983).

Miscellaneous

[13] Cody W. J. *et al*: "*A Proposed Radix- and Word-length-independent Standard for Floating-Point Arithmetic,*" IEEE MICRO 4, no. 4 pp. 86-100, (Aug. 1984).

[14] Dongarra, J.J., Bunch, J.R., Moler, C.B. and Stewart, G.W.: "*LINPACK Users' Guide.*" SIAM, Philadelphia (1979).

[15] HP Journal 35, no.7, (July 1984).

[16] Kahan W.: "*Interval Arithmetic Options in the Proposed IEEE Floating Point Arithmetic Standard,*" Proceedings of a Symposium on Interval Mathematics, pp. 99-128, Academic Press, New York (1980).