Floating-Point Indoctrination: A Summation

During May-July 1988, Prof. W. Kahan of the University of California presented a lecture course on *Computer System Support for Scientific and Engineering Computation* at Sun Microsystems in Mountain View, CA. To summarize this course, Prof. Kahan will present a final lecture, at 7:30 PM on Thursday, 28 July 1988, at Apple Computer's DeAnza-3 Building, 10500 N DeAnza Boulevard, Cupertino, CA. Enter from the south side.

This final lecture is free to the public. Please publicize to interested colleagues.

ABSTRACT

Most scientific and engineering computer users consider irrelevant the details of floating-point hardware implementation, compiler code generation, and system exception handling, until some anomalous behavior confronts them and prevents the satisfactory completion of a computational task. Some of these confrontations are inevitable consequences of the use of finite-precision floating-point arithmetic; others are gratuitous results of hardware and software designs diminished by the designers' well-intentioned corner-cutting. Distinguishing the intrinsic from the gratuitous is no simple matter; such chastened computer users are not sure what they might reasonably demand of computer system purveyors.

The novice's impression that there is no rhyme nor reason to the dark mysteries of floating-point computation is sometimes superseded by a euphoric discovery that there is a good deal that can be axiomatized and proven about floating point; later experience may temper such a discovery by indicating that not everything that can be axiomatized or proven is worth the trouble. Furthermore, what would be worth knowing is often surprisingly difficult to encapsulate and refractory to prove; even when each subproblem of a realistic application permits a satisfactory error analysis, the overall problem may admit no such analysis. The proofs of simple statements about algorithms or programs often require machinery from other parts of mathematics far more elaborate than expected. Thus some of the mathematically inclined who become involved in these studies, out of external necessity, then become permanently sidetracked by intricate mathematical issues. To remain relevant, a sense of engineering economy must guide such studies, in order to distinguish the things that are worth doing, and therefore worth doing well, from those that aren't.

Over the nearly twenty years since this lecture course was first presented, the software environment has gradually deteriorated despite that hardware has improved. The software deterioration may be attributable to the establishment of Computer Science as a separate academic discipline, whose graduates need have little acquaintance with scientific computation. The hardware improvement can be principally attributed to the advent and acceptance, for most microcomputers, of the ANSI/IEEE Standards 754 and 854 for floating-point arithmetic. But some of the potential benefits of those standards are lost because so much software was and is written to exploit only those few worthwhile features common to almost all commercially significant existing systems. In fact, much portable mathematical software, created with funding directly or indirectly from American taxpayers, is crippled by a misguided quest for performance on the fastest existing supercomputers regardless of detriment to far more numerous mini- and microcomputers.

Well-intentioned attempts by language architects and standardizing bodies to ameliorate some of the difficulties encountered in floating-point computation have too often exacerbated them and, in some instances, spread over them a fog caused by ostensibly insignificant variations in the definitions of words with otherwise familiar connotations. What we need now is a measure of consensus on language-independent definitions of needed functionality, even if we must sacrifice some compatibility with past practice to achieve intellectual economy in the future. Alas, few professionals will pay the present costs of incompatibility with past errors to achieve gains promised for an indeterminate future. The computing world has too many broken promises rusting in its basement.

One of the anticipated outcomes of this course is that lecture notes will eventually be published reflecting current thinking on some of these issues. In addition a group of students has undertaken to improve the implementation of certain elementary transcendental functions to a better standard than has been customary.

Computer System Support for Scientific and Engineering Computation

Lecture 27 - July 28, 1988 (notes revised June 14, 1990)

This is the final lecture of the floating-point indoctrination lecture series, and it is mostly a summary of the materials presented in the course.

1 Is Floating-Point Arithmetic a Moral Issue?

Floating-point arithmetic has often been viewed by many as a moral issue instead of a technical issue because there many analogies between them. Some of the analogies, listed in the order of importance, are :

- Due allowance for consequences.
- Consideration for the interests of others.
- Deference to custom.
- Deference to authority.
- Informed choice.

Any decisions which make no *allowance for consequences* cannot be considered as moral. In the floating-point arena, estimating the consequences of our deeds can be more complicated than we expect. Although this task appears to be quite straight-forward, as floating-point arithmetic is often perceived as merely a matter of mathematics, it is far from the truth. It would be simple if we knew which mathematics is germane, but this is not so because computers vary so much, and for the same computer, compilers differ too. Consequently, predicting the consequences of our deeds in floating-point arithmetic is immensely difficult.

The next most important moral issue in floating-point arithmetic has to be *consideration for the interests of others.* In the world of floating-point the interest of others tends to have been entrenched for so long and to be so diverse that the issue is really a matter of *deference to custom.* In the computing world, this is most often referred to as compatibility.

Another anology between floating-point arithmetic and moral issues is the *deference to authority.* In the arena of floating-point, it is difficult to identify who the authorities are. Suppose we can somehow identify them and we solicit their opinion on any interesting floating-point issue, there'll probably be more opinions than there are people. As it is

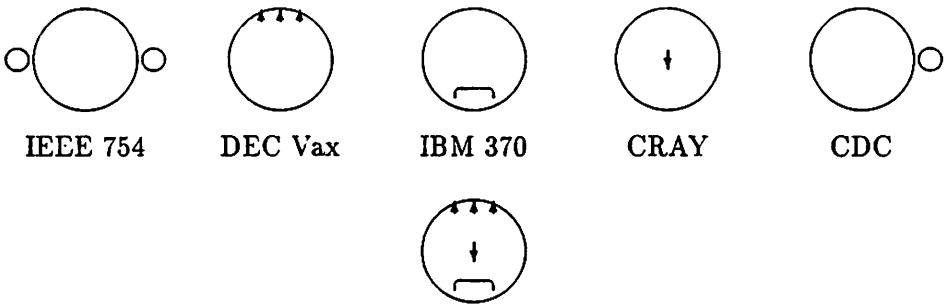


Figure 1: Models of floating-point arithmetic.

so difficult to accept anybody's opinion, there is no reason why a person should keep his opinion constant over a reasonable period of time.

The bottom line of this discussion is really a matter of *informed choice* because each of us should be able to make proper decisions, whether on technical or moral issues, without any deference to authority except our own intelligence, but with due regard for deference to customs. Deference to customs is important because it determines what people expect. We must bear in mind that our ultimate goal is *reliability*, not moral rectitude. To be reliable, we ought to do things in accordance to people's expectations.

2 Due Allowance for Consequences.

2.1 Models of Floating-Point Arithmetic.

How can we estimate the consequences of our deeds? We have discussed the various models of floating-point arithmetic in previous lectures. Stan Brown's model, which has been adopted by ADA and ANSI C, was discussed in Lecture 26 and Kulisch-Miranker axioms were discussed in Kulisch's lecture. The principles of operations of various commercial computers, such as the IBM 370, the DEC Vax and the CRAY, have been discussed in preceeding lectures.

In the past, a small number of computer companies dominated the floating-point world, so floating-point arithmetic was defined by these few companies. At present there is a standard more widely adopted than any single design; in fact, it may be more widely adopted than all the other designs combined. The widespread adoption of the IEEE standard 754 came as a pleasent surprise to many people, especially to those people involved in mathematical software, because the standard is quite difficult to implement.

The IEEE standard is difficult to implement because it presents such a clear face (see Figure 1) to the user, with very few anomalies, and has a number of features not available in most other machines : infinities and denormalized numbers (each feature represented by a "ear" in the figure). On a Vax, infinities and denormalized numbers are not available and it has some exponent limitations; besides these descrepencies, the Vax has very good arithmetic. The IBM 370 is quite peculiar mainly because it uses hexadecimal radix. On a CRAY, the arithmetic is appalling: the lack of a guard digit results in a slowdown in

double precision arithmetic by a factor of about 50 instead of about 5 to 10 in most other reasonable machines. CDC also lacks a guard digit, but in a different manner, such that there are anomalies in its denormalized numbers; it has infinities, though.

If we have a model, such as Stan Brown's model, which is supposed to explain what a machine will do in mathematical terms that are sufficiently general to encompass all machines of commercial significance, then we have a model that describes, in some sense intrinsically, the things we cannot do in general. The reason is that if we use a particular machine and encounter any of its anomalies, something bad will happen to our program.

Stan Brown's model is designed to prove that a program is correct so that we can predict the consequences of our deeds. We can only do such a prediction if we avoid the union of all the anomalies of the machines encompassed in the model, as well as some other anomalies. So Stan Brown's model, designed as a tool to aid us, turns out to be the principle of operations for a machine worse than any that has been built. It is particularly sad that this model has been incorporated into ADA and ANSI C by people who sincerely believe that it'll lead to portability of code when, in reality, it'll result in our inability to prove if a code is correct because we don't have a categorical description of the mathematics used.

2.2 Surprising Consequences Abound.

Even if we have a categorical description of the mathematics for our arithmetic model, we can still encounter various surprises. It is possible to be astonished by simple computations, two of which are shown below.

2.2.1 A Stripline Model.

A pulse is introduced in a medium (a string, for instance) and as this pulse travels along the medium, it broadens due to dispersion. Let's consider the simplest model, that is, one where there is no dispersion. This pulse can be modeled by the partial differential equation

$$\frac{\partial^2 E}{\partial t^2} = c^2 \frac{\partial^2 E}{\partial x^2}$$

which when discretized yields (let $\frac{\Delta x}{\Delta t} \approx 10c$)

$$E_x^{t+1} - 2E_x^t + E_x^{t-1} = (0.1)^2(E_{x+1}^t - 2E_x^t + E_{x-1}^t).$$

The various ways to compute $E_{\cdots}^{t+1}$ are :

Formula 1 : $E_x^{t+1} = 2 \cdot (1 - (0.1)^2)E_x^t - E_x^{t-1} + (0.1)^2(E_{x+1}^t + E_{x-1}^t)$
Formula 2 : $E_x^{t+1} = 2E_x^t - E_x^{t-1} + (0.1)^2(E_{x+1}^t - 2E_x^t + E_{x-1}^t)$
Formula 3 : $E_x^{t+1} = E_x^t(E_x^t - E_x^{t-1}) + (0.1)^2((E_{x+1}^t - E_x^t) - (E_x^t - E_{x-1}^t))$
Formula 4 : Formula 3 with compensated summation

Formula 1 appears to be economical because the constants can be computed in advance and, therefore, the amount of arithmetic is minimized. Minimizing the amount of arithmetic would lead one to conclude that the amount of rounding errors is minimized too, but this is not true. Formula 2, which is found in some textbooks, is a rearrangement of Formula 1, but Formula 3 is actually superior to Formula 2 because when two numbers are close together and one is subtracted from the other, no rounding error is introduced (on

reasonable machines). [1] Since the values at a certain time and the next (like E_x^{t-1} and E_x^t) and those of neighboring points (like E_{x+1}^t and E_x^t or E_x^t and E_{x-1}^t) are quite small, no rounding error is introduced in computing $E_x^t - E_x^{t-1}$, $E_{x+1}^t - E_x^t$ and $E_x^t - E_{x-1}^t$. The first add in Formula 3 is the operation which introduces the most rounding errors; compensated summation (see lecture 9) can be used to circumvent this problem, and it is implemented in Formula 4.

It takes a certain amount of experience and expertise in floating-point arithmetic to determine where rounding errors will occur in the four formulae; what most people see are simply algebra. The ability to predict the consequences of one's deeds requires the expertise to see these invisible rounding errors.

The results produced by the four formulae are quite astonishing; the experiments were performed on an IBM 3090. When a positive pulse (pulse with positive amplitude) was introduced into the medium, Formulae 1-3 produced negative pulses; the pulse for Formula 1 was more negative than the one for Formula 2, which in turn was more negative than the one for Formula 3. Consequently, one would say that Formula 2 is better than Formula 1, but if one does not know the correct answer apriori, how does one know that one is better than the other and they are all wrong? Afterall, the pulses seemed nice and smooth. The difference in the pulses is a result of the difference in patterns of rounding errors. Formula 4 produced a pulse which did not have a negative amplitude, which is correct. For a person who is not acquainted with the physics of pulses there is no reason for him to suspect that Formulae 1-3 produced incorrect solutions.

An interesting observation is that if the computations were performed to the resolution of the graph, all four formulae produced the correct solution. This implies that rounding on the IBM 3090 was biased. For machines which are truly unbiased, like machines which conform to the IEEE standard (Apple Macintosh, Sun), or which are almost unbiased (DEC Vax) the law of averages is in their favor, irrespective of the formula used. An important point about the IEEE standard is that, on the whole, we tend to get results which are at least as good as those obtained from other machines of comparable capacity (precision, memory, ...), yet the cost incurred is not that great.

2.2.2 Monotonicity.

Monotonicity simply means that if the data are altered, the results change in the correct direction, if they are supposed to change. This property can actually persevere even if the results are not what we expect, that is, even if the number of correct figures in the results is less than desired, monotonicity is preserved.

Monotonicity is a very important property because it is transitive on composition. Consider

$$h(x) = g(f(x)).$$

If $f()$ and $g()$ are both monotonic then $h()$ is also monotonic. If $f()$ and $g()$ are both accurate, $h()$ is not necessary accurate because the result of $f()$ may cause the input of $g()$ to change by very little, but sufficiently large to destroy the accuracy of the final answer. Therefore, accuracy is not a transitive property. The fact that accuracy do not persevere when programs, which produces accurate results separately, are composed is one of the reasons why the ACRITH system does not work as well as desired.

[1] Reasonable machines do not include the CRAY or the CDC.

In practice, the situation is more interesting than we would like. The function

$$f(x) = x - sin(x)$$

is monotonic since

$$\begin{aligned} f'(x) &= 1 - cos(x) \\ &= 2sin^2(x/2) \\ &\geq 0 \end{aligned}$$

which implies that $f(x)$ increases monotonically with x. Let $F(X)$ be the computed value of $f(x)$ when $x = X$. Assume that the trigonometric function sin(x) is correctly rounded for all x. Is $F(X)$ monotonic? The result is quite surprising :

Number of significant digits (decimal)	Is it monotonic?
4	Yes, no exceptions
5	one exception at 0.010000
6	one exception at 0.100167

How can we anticipate such counter-intuitive results? Interested readers should try to prove the results and find out what happens in binary. This example is an instance where our intuitions do not survive a few rounding errors.

If we have a program which works correctly in theory and in practice, it must be treasured. Such a program is very valuable and should be usable by others without having the proof for correctness destroyed by somebody who would rather save a few milliseconds by cutting corners thereby sacrificing accuracy. These correct codes correspond to commodities that an engineer can buy with the greatest confidence that they'll work as expected.

In the world of floating-point, even the smallest probability of failure should not be tolerated (see Lecture 1). If floating-point operations were to fail on one occasion in a million, then with current computers, we'll have a failure every second. Therefore, we must subject ourselves to an extremely high standard of reliability; otherwise people will waste their time debugging our software.

3 Consideration for the Interests of Others.

Even though many people consider floating-point arithmetic to be a matter of mathematics and should therefore be predictable, they fail to see that mathematics is a game where mathematicians play with whatever rules they like. There is no mathematical law that obliges you to stick with any other mathematical law. He who plays the game may choose and change its rules.

When we have to work together in a community, since we depend on the work of others, we must be considerate and choose mathematical rules with due regard to what others expect. Otherwise, what may seems like a change of mathematical rule to us becomes a capricious change that introduces an element of unreliability for others.

	$j < 0$	$j = 0$	$j > 0$
x in X	$(1/x)^{-j}$	1	x^j
x is NaN	x		x

Table 1: Exponentiation x^y with y = integer j.

	$y = -\infty$	$-\infty < y < 0$	$0 < y < +\infty$	$y = +\infty$
$x < -1$	$+0$	NaN *		$+\infty$
$x = -1$				
$-1 < x < 0$	$+\infty$	$+\infty$ *		$+0$
$x = 0$				
$0 < x < 1$				
$x = 1$	NaN *	$\exp(y \cdot \ln(x))$		NaN *
$1 < x < +\infty$	$+0$			$+\infty$
$x = +\infty$				
x is NaN	x			

Table 2: Exponentiation x^y with $y \neq$ integer.

3.1 Exponentiation x^y.

We had a lengthy discussion in lecture 24 on what the values of x^y should be. Should $0^0 = 1$? What about NaN^0? Tables 1 and 2 are intellectually economical specifications for exponentiations x^y.

In the specifications, much care was taken to ensure that the results of exponentiation can be extended smoothly to the complex domain, even though we are working with real numbers. Consequently, a NaN is obtained when a negative number is raised to a fractional power because if we had obtained a real result we would have encountered problems when we deal with complex numbers.

Although Table 2 appears complicated, the code to implement it isn't. Whenever we cross a line in the table, we cross a singularity : a singularity is a boundary between domains of analyticity where everything on one side follows a set of rules while those on the other side follows another. Fortunately, most of the boundaries are not reflected in the code by the usual tests and branches because they are taken care of by exp() and ln(). All entries in the table, except $(x < 0)^{\pm\infty}$, $0^{(y>0)}$ and $0^{-\infty}$ are produced automatically, including signal marked by *, by the expression $\exp(y \cdot \ln(x))$ provided it is evaluated in a way analogous to the specifications of the IEEE standards. Thus, despite the need to handle infinities and NaNs, the code needed to implement x^y is actually quite simple, while other models of arithmetic require the users to handle exceptional values by themselves.

For people who do not have the perserverence to handle all the details of what to do in each domain of analyticity, *rectrospective diagnostic* is a useful tool (see lecture 25). Rectrospective diagnostic allows the users to provide whatever values they feel are reasonable for exceptional cases, but record these exceptional events in a rectrospective log. Since these are exceptional events, usually there won't be any entry in the log, but when exceptional events occur, the users can investigate further.

3.2 Maximun $\max(x, y)$.

This is an example simpler than exponentiation to illustrate some of the difficulties which cannot be solved by simply waving a mathematical wand. It is hard to imagine that there'll be difficulty defining the maximun of two numbers, but it's more complicated than most of us would expect.

One way to implement $\max(x, y)$ is by

$$\max(x, y) = \frac{(x + y) + |x - y|}{2}$$

which is short and simple, and is often used. This implementation works well if there're no rounding error or over/underflow. But even for integers, it can malfunction since integers can overflow. For floating-point numbers, it can malfunction in lots of interesting ways. For instance, if $x = -10^{30}$ and $y = 5$ and double precision numbers are used,

$$x + y = -10^{30} \quad \text{and} \quad |x - y| = +10^{30}$$

so $\max(x, y) = 0$! The formula malfunctioned because of rounding errors.

Another implementation of $\max(x, y)$ is

$$\begin{aligned} \max(x, y) = \ & \text{if } x > y \quad \text{then } x \\ & \qquad\qquad \text{else } y \end{aligned}$$

but it does not work correctly for certain operands :

$$\begin{aligned} \max(\text{NaN}, 5) &= 5 \\ \max(5, \text{NaN}) &= \text{NaN} \end{aligned}$$

This is a rather bizzare situation because a different result is produced when the two operands are swapped. The reason for the strange behavior is because any predicate which involves $<$, $\leq$, $>$ or $\geq$ is false when either of the operands, or both, is a NaN. [2]

The code below functions properly for all operands :

$$\begin{aligned} &\max(x, y) = \text{if } x \neq x \text{ then } y \\ &\text{else if } x? > y \text{ then } x \\ &\qquad\qquad \text{else } y \end{aligned}$$

The predicate "$x? > y$" is true when $x > y$ or when either x or y, or both, is a NaN, and the predicate "$x \neq x$" is false only when x is a NaN. Now

$$\max(\text{NaN}, 5) = \max(5, \text{NaN}) = 5$$

which is acceptable by most people, although some people argue that if NaN is not a number (literally), how can we say that 5 is greater than NaN? The point is that the result can be either one, but it must be consistent. There is no proof that 5 (or NaN) is correct, but experience (such as windowing applications) indicates that 5 may be a better choice.

Another problem spot for $\max(x, y)$ is when the arguments are $+0$ and -0. The values of $\max(+0, -0)$ and $\max(-0, +0)$ are arithmetically equal, but they may be represented by different bit strings. It would be a good idea to have the maximun function implemented in hardware, which uses lexical comparison of bit strings, such that $+0 > -0$ lexically. In fact, both $\max(x, y)$ and $\min(x, y)$ should be implemented in hardware since it is much faster than software implementations.

[2] The invalid operation flag is signaled, so presumably the user can use this flag to circumvent the problem.

4 Inevitable Conflict : Speed Versus Reliability.

Speed and Reliability are two qualities most commonly used to determined the performance of a computer system, but unfortunately they are often conflicting properties. Most people value reliability no higher than the price of the insurance premium paid to insure against the consequences of unreliability. On the other hand, there are some people who feel that reliability is so important that they make sure others pay attention to it by instilling fear in unreliability.

Actually numerical computations are extremely reliable, all things considered, and much of the reliability come about because they really do not matter, that is, a large fraction of numerical computations, correct or incorrect, are discarded unread. Until there is an error serious enough to be noticeable will the users be aware of the unreliability. Consequently, unreliable computations very often do not upset us because they do not sufficiently hurt us or their existance passed by unnoticed.

The issue of speed versus reliability should be dealt with in a rational, though not necessary uniform, manner so that we can predict the consequences of the computations. These decisions should be made with intellectual economy, economy of effort and humane use of human talent in mind.

Unfortunately, if something is unreliable, then somebody will be blamed for it. This is one of the reasons why some people consider the speed versus reliability issue a moral issue because they believe morality is a question of deciding whom to blame. This is a common, but incorrect, view. It is so common because if we can determine who to blame, we know who is responsible for changing things.

Programmers usually bear most of the blame, but this is usually unjustified because the environment in which he live is so intellectually unmanageable and is full of anomalous diversity, incoherent standards, non-uniform implementations, "optimizing" compilers, feeble diagnostic aids (especially at run-time), cumbersome text-processing, and chaotic I/O and graphics. Things have gotten increasing complicated to the point where programmers would have to be much better educated than they have been or will be in the near future. There is such a deversity that if programmers were really professionals they would have to know about the arithmetic on several machines. How many programmers do you know who is knowledgeable in the arithmetic of more than one machine?

At present, there is such diversity in languages, arithmetic and exception handling among commercial computers, so it is not reasonable to blame the programmers if something do go wrong. Read lectures 21-23 to determine for yourselves how some current commercial machines deal with exception handling.

4.1 Horrors Programmers Encounter.

In lecture 26 we discussed how programming costs are exaburated when a programmer most conform to Stan Brown's model. In particular, we saw this example :

$$\begin{aligned} f(x) = \text{if } x > 1 \text{ then -arctan}(\ln(x))/\text{arccosh}(x)^2 \\ \text{else if } x = 1 \text{ then } -0.5 \\ \text{else arctan}(\ln(x))/\text{arccosh}(x)^2 \end{aligned}$$

This seems like a simple task but if it is written in the obvious way, we'll encounter anomalies on certain machines. If this code, transcribed in any language, is executed on machines like the IBM 370, the DEC Vax or machines which conform to the IEEE standard 754, the

relative error bound is small for all arguments. If we want to predict the error bound using Stan Brown's model, the bound is very pessimistic; in fact, for certain arguments, like those near $x = 1$, the bound approaches infinity because this model makes allowance for machines like the CRAY, the Cyber or the UNIVAC. On these machines something bad could happen near $x = 1$ as a result of cancellation errors.

If a programmer wants to write a portable code for the above task and if he is restricted to using Stan Brown's model, then he must figure out, at great time expense, a special Taylor series for $f(x)$, so that bad things do not happen near $x = 1$. If in the vacinity of $x = 1$

$$f(x) = -\frac{1}{2} + \frac{(x-1)}{6} - \frac{(x-1)^2}{20} - \frac{124(x-1)^3}{945} + \cdots$$

is used instead of the usual formula, then the error bound near $x = 1$ are suppresed. Consequently, the error bound for Stan Brown's model is more manageable now, although still very pessimistic.

This example illustrates a situation where the reward for the diligent extra efford on the part of the programmer is a program which is less accurate than the original code executed on IEEE machines. If the modified code is executed on an IEEE machine, the best error bound is actually worse than that of the original code executed on a similar machine. This means that in the process of writing a portable code, the performance on reasonable machines is penalized.

4.2 Effect of the IEEE 754 Standard on Reliability

The IEEE standard 754 is now the most widely adopted design for a family of floating-point arithmetic computers. Last year (1987) there were over 8 million machines which conform to the standard, but the count is probably much higher now because there are more chips conforming to the standard.

The standard is appreciated most by people who used to devote a part of their youthful exuberance dealing with the lack of arithmetic uniformity among the various machines. Fortunately now there's a surprising degree of uniformity as a result of the standard. Among the chips which conform to the standard are :

Intel 8087, 80287, 80387	(IBM PC, XT, AT, ...)
Motorola 68881, 68882	(Sun III, new Macintosh, ...)
AT&T WE 32106	(AT&T and Zilog systems)
National Semiconductor 32081	(IBM RT-PC, ...)
Weitek 1164/5	(Sun III Floating-Point Accelerator, ...)

Because of the widespread adoption of the standard, arithmetic is much more uniform and the consequences of our deeds are more readily predictable.

4.3 What does the IEEE 754 Standard Confer upon You?

Figure 2 shows a consequence of the clear-face presented by the IEEE standard, which has so few anomalies. Any numerical software, portable to any two of the three machine families (IBM 370, DEC Vax and CDC Cyber), will almost certainly run at least as well on any machine that conforms to IEEE 754 with comparable capacity (speed, memory and precision). No existing arithmetic design can make this claim, though DEC Vax F-G-H formats can come close with appropriate software support.

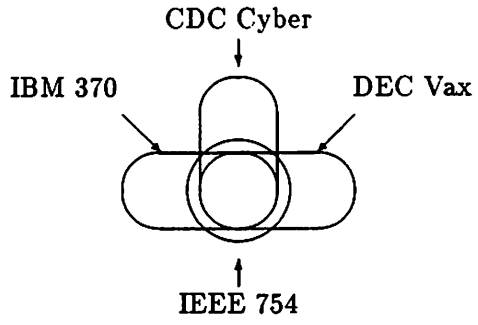


Figure 2: Importability of "portable" software.

As more machines conform to IEEE 754, more software will be designed specifically for them. Therefore, we will have access to a growing body of superb software designed to run on IEEE 754 machines, but difficult or impossible to adapt to other machines. Examples of such softwares are statistical packages that exploit IEEE 754's NaN for missing or uninitialized data.

4.4 Computer Software Support.

Computer software support are essential for the development of more sophisticated softwares, but unfortunately the software support situation is not satisfactory on some machines.

4.4.1 Signed Zero.

In lecture 24 we discussed the situation where some machines do not support signed zero. It is important to have both $+0$ and -0, otherwise strange things will happen. When both $+0$ and -0 exist,

$$+\infty = \frac{1}{(+0)} \neq \frac{1}{(-0)} = -\infty.$$

This relation, which is not true if -0 is represented as $+0$, must hold or the program for comformal maps of slitted domain (discussed in lecture 24) will not work. When we attempt to plot the graph of conformal maps on an IEEE machine (has signed zero) everything works well, but on non-IEEE machines without signed zero part of the slot's boundary goes astray.

It is a shame that so few implementations of computer arithmetic have signed zero, especially when we are in an era when one of the main application using complex arithmetic deals with conformal maps.

4.4.2 You Don't Always Get All You Bought.

There are things which we don't have, though we've paid for them. For instance, we still do not have good extended precision capabilities. Borland's PASCAL compiler on the

IBM PC has facilities to access the Intel 80-bit extended precision format, but most other compilers, such as Microsoft's, do not provide such capabilities. Therefore, very often we are denied access to extended precision capabilities even though we've paid for them. Extended precision numbers are very useful for matrix operations like iterative improvement.

Another two features missing in most compilers are convenient exception handling capabilities and tools for providing directed rounding. Exception handling and directed rounding are useful in the development of some software like those for stability analysis.

All the features described above are accessible on the Macintosh because of Apple's diligent supervision of higher-level language implementations. Convenient exception handling and directed rounding are also available on the Sun.

5 Major Tasks Remain.

There are still several major tasks remaining which need to be addressed, and some of them are :

1. Computer scientists, especially compiler writers, need to be made aware of the issues they affect by better educating them on floating-point issues.

2. Good exception handling capabilities need to be implemented: implement rectrospective diagnostic to ease debugging of purchased code; implement presubstitution to alleviate dependence upon precise interrupts; and permit vast exponent range extension in certain very special contexts.

3. The provision of benchmark with diagnostic capabilities. When performing benchmarks, this capability aids us to determine the things about our systems the benchmarks do not like.

Problems 2-3 are technical problems where solutions are now in sight, but problem 1 is political in nature, and therefore refractory.

So far most of the work have been done by very few individuals. We need to devote more attention to floating-point research, but funding agencies are more interested in "sexy" machines, such as supercomputers and parallel machines, many of which have very perverse arithmetic.

Is Floating-Point Arithmetic A MORAL ISSUE ?

- Due allowance for Consequences.

- Consideration for the Interests of Others.

- Deference to Custom

- Deference to Authority.

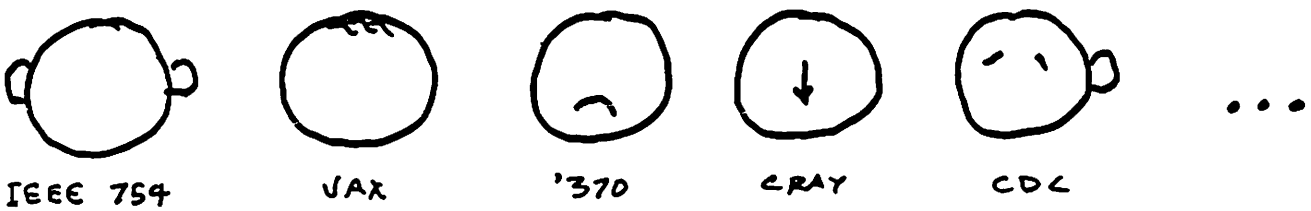
- INFORMED CHOICE

W. Kahan
24 Jul. 89

Due Allowance for Consequences :

HOW CAN YOU TELL
WHAT THEY WILL BE ?

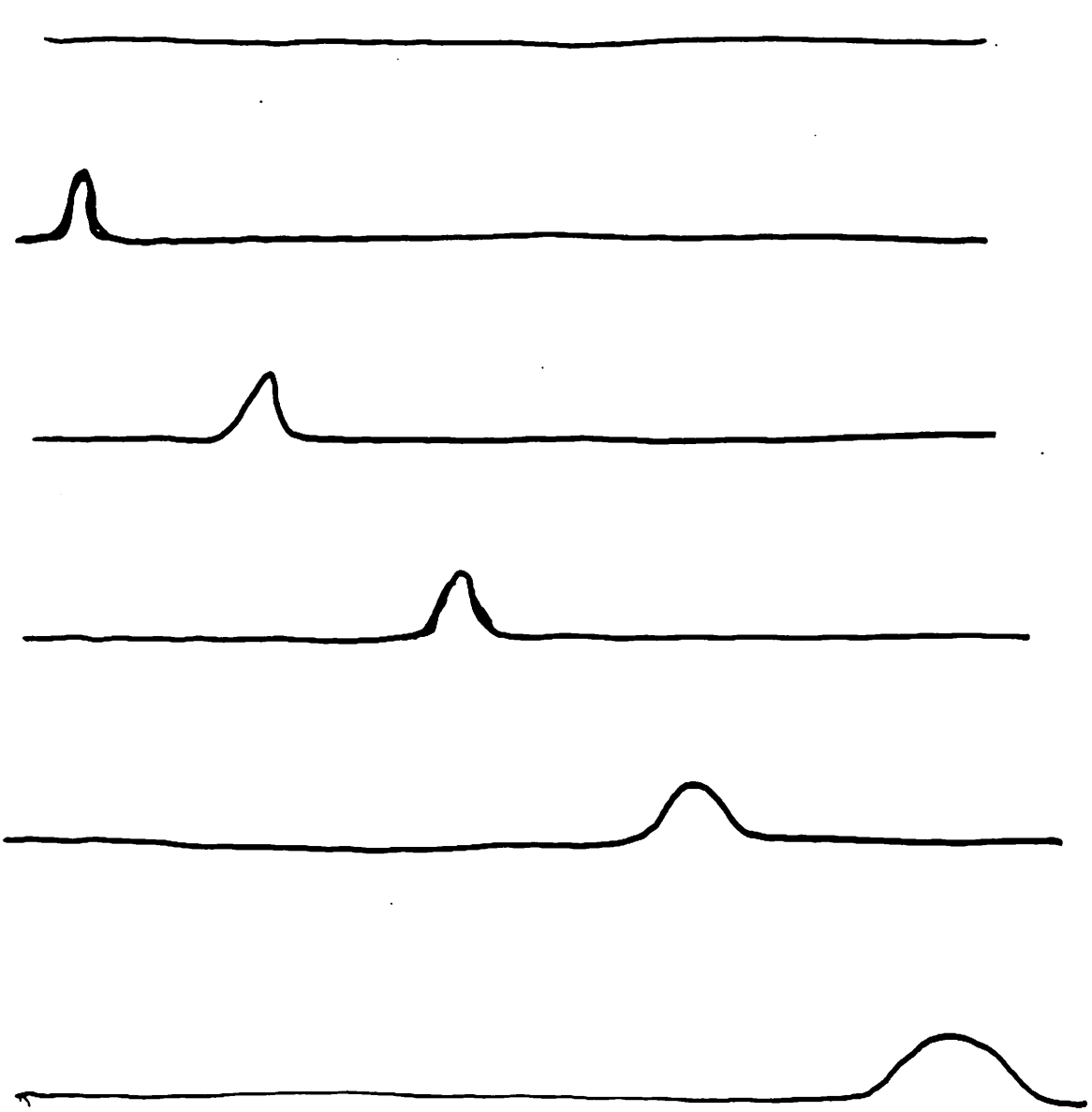
- Stan Brown's Model
  ... cf. ADA, ANSI C, ...
- Kulisch-Miranker Axioms
- "Principles of Operation of ...."
  IBM '370 DEC VAX CRAY ....
- IEEE standard 754



Brown's Model

SURPRISING CONSEQUENCES ABOUND

e.g. #1 A stripline model ...



TYPICAL TROUBLE WITH COMPUTER ARITHMETIC

Thurs. Sept. 24, 1987

JIAYUAN FANG, grad. student, Elect. Eng. & Computer Sci.
U.C. Berkeley

Microwave strip-lines .... wave equation $\frac{\partial^2 E}{\partial t^2} = c^2 \frac{\partial^2 E}{\partial x^2}$

DISCRETIZED: $t \to 0, \Delta t, 2\Delta t, 3\Delta t, \ldots$; $x \to 0, \Delta x, 2\Delta x, 3\Delta x, \ldots$ } $\frac{\Delta x}{\Delta t} \doteq 10c$, say

$$E_x^{t+1} - 2E_x^t + E_x^{t-1} = (0.1)^2 \left(E_{x+1}^t - 2E_x^t + E_{x-1}^t \right)$$

To compute $E_{\ldots}^{t+1}$ from $E_{\ldots}^{t}$ and $E_{\ldots}^{t-1}$:

different ALGEBRAICALLY EQUIVALENT formulas

Formula #1:

$$E_x^{t+1} := 2\cdot(1-(0.1)^2)\,E_x^t - E_x^{t-1} + (0.1)^2 \left(E_{x+1}^t + E_{x-1}^t \right)$$

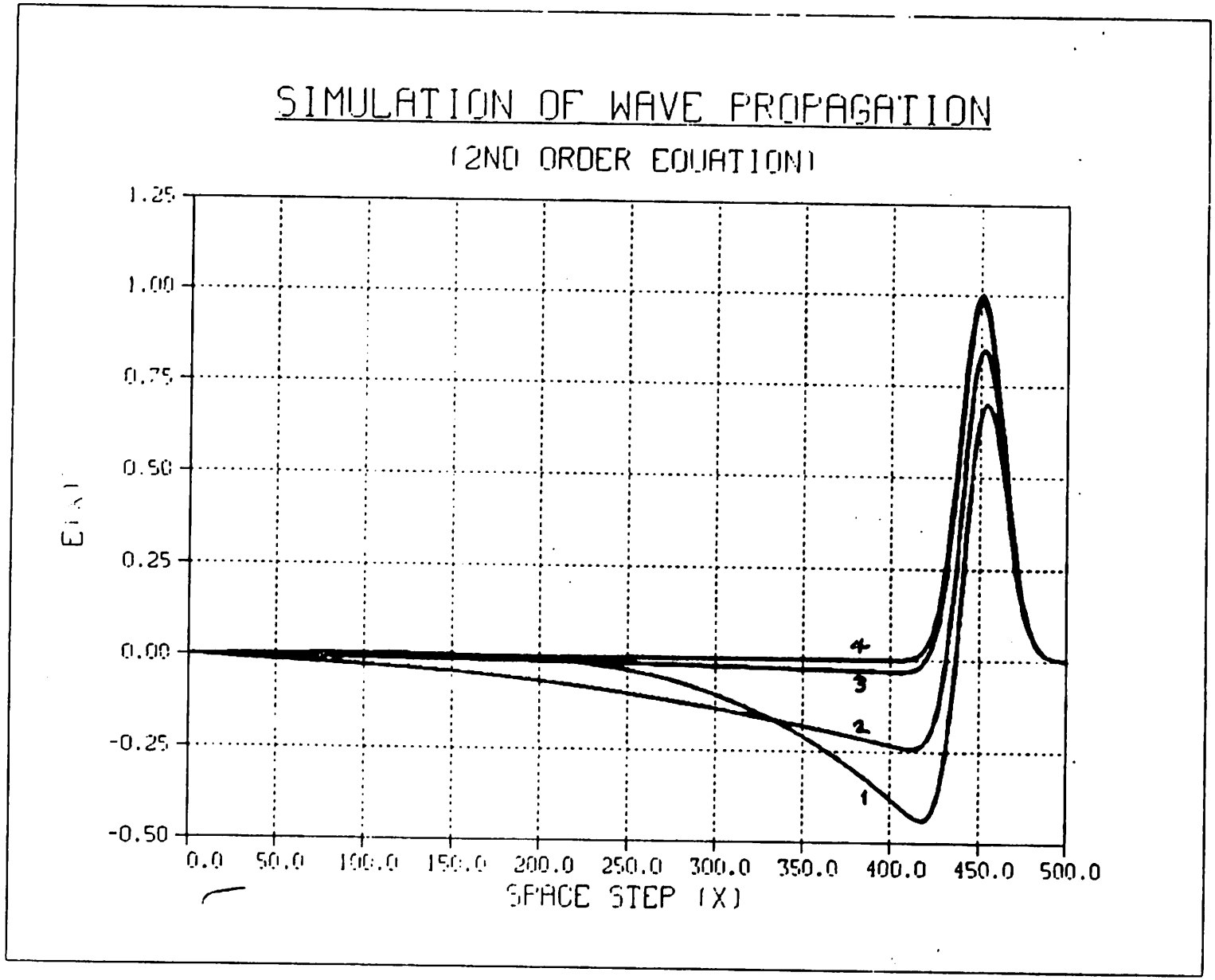
Formula #2:

$$E_x^{t+1} := 2E_x^t - E_x^{t-1} + (0.1)^2 \left(E_{x+1}^t - 2E_x^t + E_{x-1}^t \right)$$

Formula #3:

$$E_x^{t+1} := E_x^t + \left(E_x^t - E_x^{t-1} \right) + (0.1)^2 \left((E_{x+1}^t - E_x^t) - (E_x^t - E_{x-1}^t) \right)$$

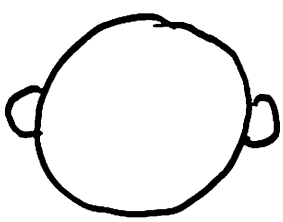
Formula #4: ... W.K.'s rearrangement ...



1-4 on DEC VAX, SUN, APPLE MAC, ... ≡ 1-3 on IBM 3090
4 on IBM 3090

The moral of that story:

IEEE 754 arithmetic does
at least about as well
as any other of
comparable capacity
(precision, speed, memory, ...)



SURPRISING CONSEQUENCES ABOUND

eg. #2 MONOTONICITY

This is one of the few properties TRANSITIVE under composition of functions:

If $h(x) := g(f(x))$,

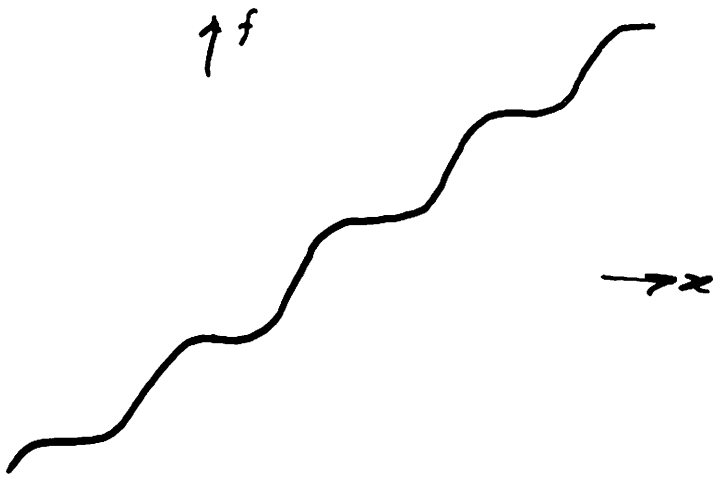
and if

	f	is	monotonic	accurate
and	g	is	monotonic	accurate
then	h	is	monotonic	~~accurate~~

✓ ✗

MONOTONICITY IS OFTEN PRESUMED.

COUNTER-INTUITIVE EXAMPLE:



$f(x) := x - \sin(x)$

MONOTONIC:

$f'(x) = 1 - \cos(x)$

$= 2\sin^2(x/2) \geq 0.$

$F(X) := X - SIN(X)$

↑ assume CORRECTLY ROUNDED

Is $F(X)$ monotonic ?

6	sig dec:	1 exception @ 0.100167
5	sig dec:	1 exception @ 0.010000
4	sig dec:	YES, no exceptions.

CAN YOU PROVE THIS ?

WHAT ABOUT BINARY ?

Consideration for the interests of others.

THERE IS NO MATHEMATICAL LAW THAT OBLIGES YOU TO STICK WITH ANY OTHER MATHEMATICAL LAW. HE WHO PAYS FOR THE GAME MAY CHOOSE AND CHANGE ITS RULES.

But it may be inconsiderate of others to change the rules capriciously.

INTELLECTUALLY ECONOMICAL

Specifications for Exponentiation x^y over the Extended Reals X :

W. Kahan

EXPONENTIATION x^y with y = integer j

	j < 0	j = 0	j > 0
x in X	$(1/x)^{-j}$	1	x^j
x is NaN	x		x

EXPONENTIATION x^y with y ≠ integer

	y = −∞	−∞ < y < 0	0 < y < +∞	y = +∞
x < −1	+0	NaN *		+∞
x = −1				
−1 < x < 0				
x = 0	+∞	+∞ *		+0
0 < x < 1		exp(y*ln(x))		
x = 1	NaN *			NaN *
1 < x < +∞				
x = +∞	+0			+∞
x is NaN	x			

All entries in this table except $(x<0)^{\pm\infty}$, $0^{(y>0)}$ and $0^{-\infty}$ are produced automatically, including the signals where marked by an * , by the expression exp(y*ln(x)) provided it is evaluated in a way analogous to the specifications of the IEEE standards, and then the expression exp(NaN*ln(x)) quietly produces NaN for x^{NaN} too. In the previous table 1/0 signals DIVBZ .

IEEE 754 does at least about as well as ...

"Everything should be as simple as possible, but no simpler." ... Einstein

$$\max\{x, y\} := \ ?$$

Try

$$\max\{x, y\} := \frac{(x+y) + |x-y|}{2} .$$

$x = -10^{30}$, $y = +5$ } $x+y \to -10^{30}$, $|x-y| \to +10^{30}$ } $\max \to 0$ OOPS!

Try

$$\max\{x, y\} := \text{if } x > y \text{ then } x \text{ else } y .$$

$\max\{NaN, 5\} \longrightarrow 5$

$\max\{5, NaN\} \longrightarrow NaN$

LOST TRICHOTOMY!

BOTH SIGNAL INVLD.

Try

$\max\{x, y\} :=$ if $x \neq x$ then y
else if $x ?> y$ then x
else y.

"$x ?> y$" is true when $x > y$ or either is NaN, and NO SIGNAL

Now $\max\{NaN, 5\} = \max\{5, NaN\}$
$= 5.$

Is this a good idea? cf. WINDOWING.

But $\max\{+0, -0\}$ is not necessarily the same bit-string as $\max\{-0, +0\}$.

NEED (hardware implemented)
LEXICAL COMPARISON
so $+0 \gtrdot -0$ lexically.

INEVITABLE CONFLICT

Most people value RELIABILITY no higher than the price of the insurance premium paid to insure against the consequences of UNRELIABILITY.

cf. Interval Arithmetic : it costs 2 to 10 times as much as the computation whose correctness it would ensure, and all too often merely cries "WOLF".

How to Justify Concern for RELIABILITY: INSTIGATE FEAR.

cf. Interval Arithmetic, Karlsruhe scheme (ACRITH), ...

Actually, best justification for Interval Arithmetic is its use in SEARCHING PROGRAMS that search for Zeros or Extrema of multi-variable functions.
(Eldon Hanson)

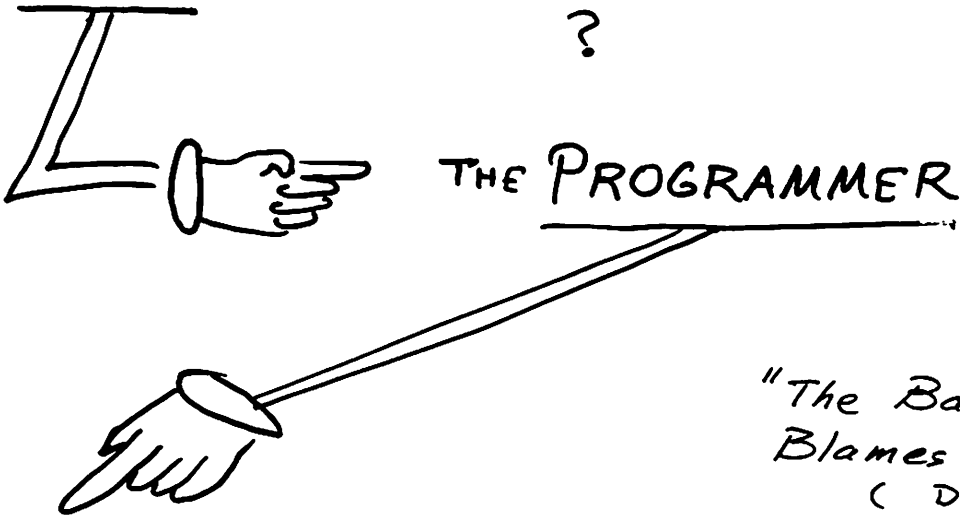
I prefer to argue for ECONOMY :
... intellectual economy,
... economy of effort,
... humane use of human talent.

RELIABILITY = Conformity to Reasonable Expectations

= NEVER having to say "SORRY".

WHO gets BLAMED for *Unreliable Numerical Software*

?



THE PROGRAMMER

"The Bad Blacksmith Blames the Iron"
(Dante)

WHAT IS TO BLAME FOR

UNRELIABLE NUMERICAL SOFTWARE

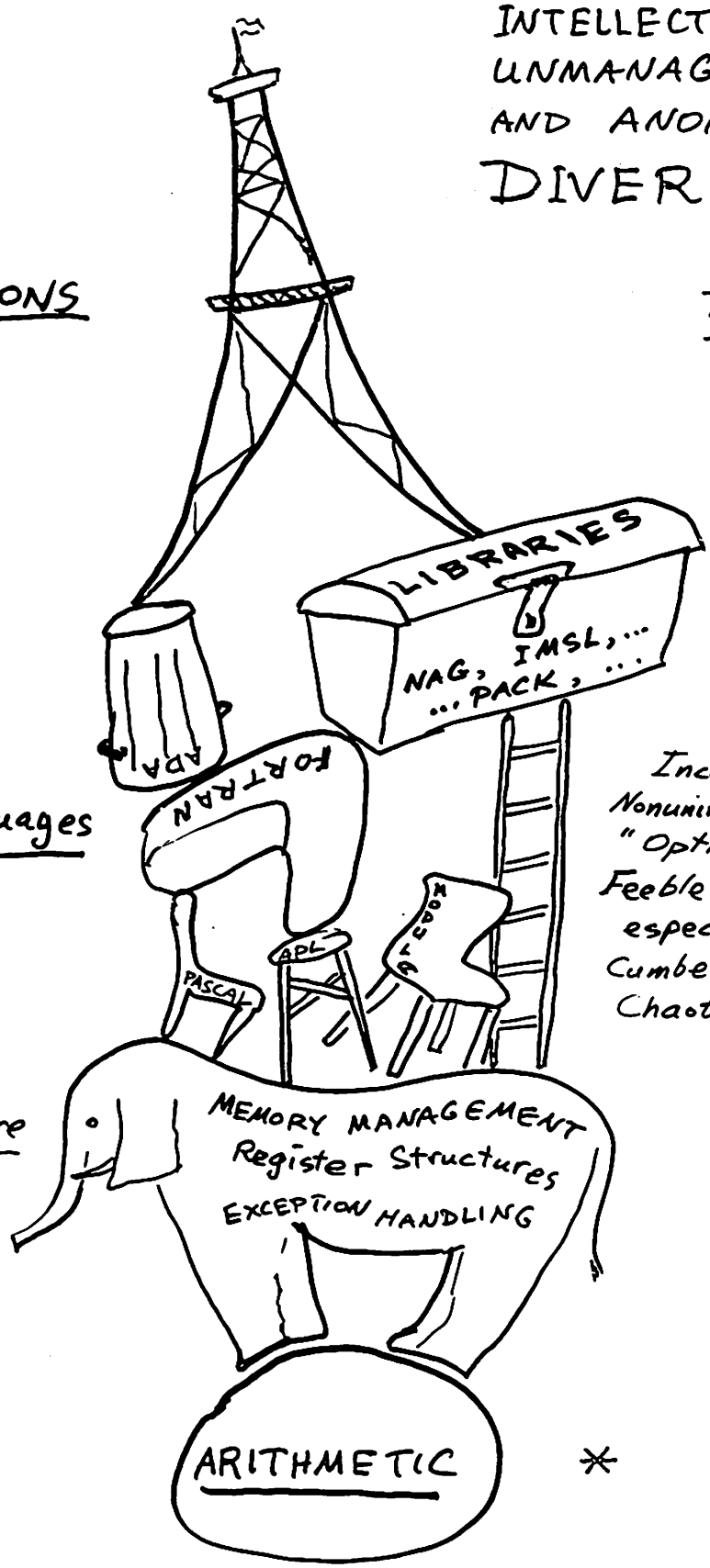
?

BLAME?... WHAT SHOULD BE CHANGED ?

INTELLECTUALLY UNMANAGEABLE AND ANOMALOUS DIVERSITY.

APPLICATIONS

The modern Tower of Babel



Languages

Incoherent Standards.
Nonuniform Implementations.
"Optimizing" compilers.
Feeble Diagnostic Aids, especially at Run-Time.
Cumbersome Text-processing.
Chaotic I/O, graphics.

Hardware

*

Jean-Michel Muller's Example

From his forthcoming book ARITHMETIQUE DES ORDINATEURS :

$$a_0 := 11/2$$

$$a_1 := 61/11$$

$$a_{n+1} := 111 - (1130 - 3000/a_{n-1})/a_n$$

This should generate a sequence { a_n } that converges slowly to 6 , but when computed in floating-point on any computer with only finite precision the computed sequence converges rapidly to 100 instead.

****

W. K.'s example:

An example of a singularity removable numerically only if the computer's arithmetic carries a guard digit in subtraction:

```
Real function  f( real z )  :=
        if z < 0 or z > 1  then  0/0
          else if  z = 1  then  -0.5
                 else  arctan( ln(z) )/arccos(z)² .
```

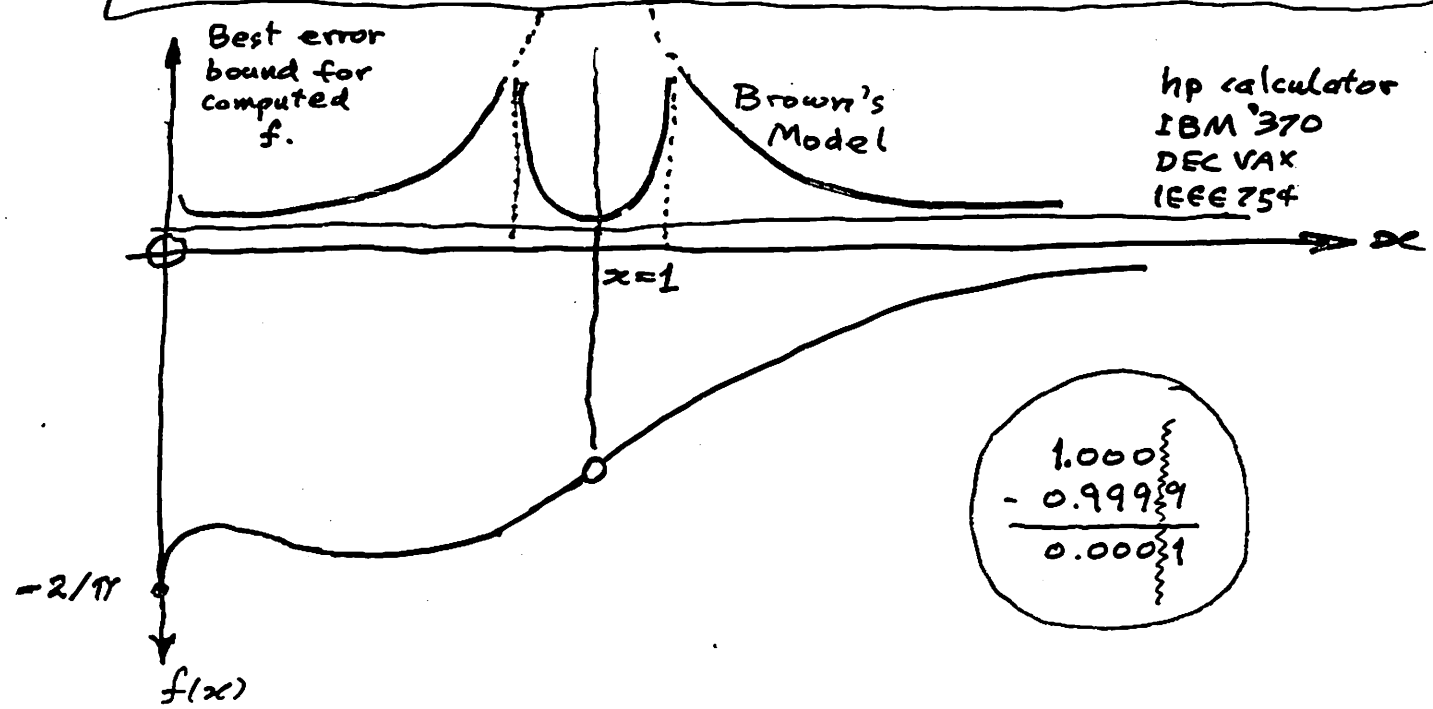
The function has a power series expansion near z = 1 :

$$f(z) = -0.5 + (z-1)/6 - (z-1)^2/20 - 124(z-1)^3/945 + \ldots .$$

EXAMPLE (OVERSIMPLE but TYPICAL)

to show how programming costs are exacerbated when a programmer MUST conform to Brown's Model:

```
f(x) := if x > 1 then -arctan(ln(x))/arccosh(x)^2
        else if x = 1 then -0.5
        else arctan(ln(x))/arccos(x)^2 .
```



For { CDC Cyber 205 / Brown's Model, / Olver's sli arith. } we must re-code f(x) thus:

```
T := ... Threshold dependent upon roundoff level ...
f(x) := if x > 1+T then -arctan(ln(x))/arccosh(x)^2
        else if x > 1-T then
            -1/2 + (x-1)/6 - (x-1)^2/20 - 124·(x-1)^3/945...
        else if x > 0 then -arctan(ln(x))/arccos(x)^2
        else if x = 0 then -2/π
        else ... "ERROR".
```

"ln(x)" → ln(x + δ₁x)

"arccos(x)" → arccos(x + δ₂x)

} δ₁x ≠ δ₂x

IEEE 754 is now the

SINGLE MOST WIDELY ADOPTED DESIGN

for a family of floating-point arithmetics in computers:

CHIPS:

Intel 8087, 80287, 80387 in IBM PC-XT, AT, ...
Motorola 68881, 68882, in SUN III, New MACINTOSH, ...
AT&T WE 32106 in AT&T, Zilog systems
National Semi. 32081, in IBM RT-PC, ...
Weitek 1164/5, in SUN III fpa, ...
Analog Devices, AMD, BIT, Fairchild "Clipper", ...
(INMOS T800, nearly)

COMPUTER FAMILIES:

APPLE MACINTOSH, APPLE II with S.A.N.E
ELXSI 6400
HEWLETT-PACKARD "SPECTRUM"

BINARY

4 byte	SINGLE	(24 sig bit.,	$10^{\pm 38}$)
8 byte	DOUBLE	(53 sig. bit.,	$10^{\pm 308}$)
optional ≥ 10 byte	"EXTENDED"	(≥ 64 sig. bit,	$10^{\pm 4900}$)

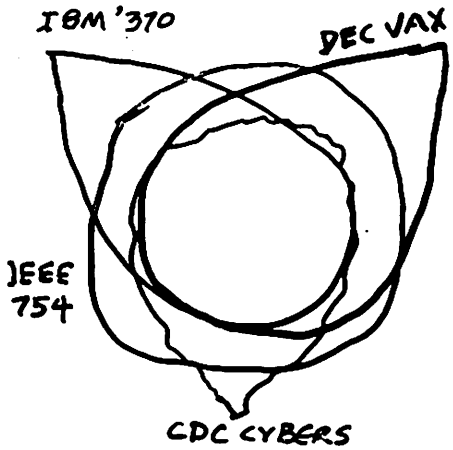
IEEE 854 DECIMAL: hp-71B (hand-held BASIC)
(hp-18C, hp-28C shirt-pocket, INTERNALLY)
(12 sig. dec., $10^{\pm 499}$)

---

cf. W. J. Cody et al. "A Proposed ... Standard ... Arithmetic"
in IEEE MICRO, Aug. 1984 pp. 86–100

WHAT DOES IEEE 754 CONFER UPON YOU?

1. IMPORTABILITY of "PORTABLE" SOFTWARE



ANY numerical software, portable to any two of the three machine families IBM '370, DEC VAX, CDC Cybers, must almost certainly run at least about as well on any machine that conforms to IEEE 754 with comparable capacity (speed, memory, precision(s)).

NO OTHER EXISTING ARITHMETIC DESIGN CAN MAKE THIS CLAIM, though DEC VAX F-G-H formats can come close with appropriate (non-DEC) software support.

2. Access to a growing body of superb software designed to run on IEEE 754 but difficult or impossible to adapt to other machines.

e.g. – Statistical packages that exploit IEEE 754's NaN for missing or uninitialized data

– Robust equation-solver (1 real v'ble) built into hp-18C, hp-28C; soon to appear elsewhere.

– 4.3 BSD Berkeley UNIX Math. library of Elementary Transcendental Functions, one library for VAX D, one library for IEEE 754 Double

– Stable versions of "Unstable" Algorithms.

$+0 = -0$; ... in IEEE 754/854

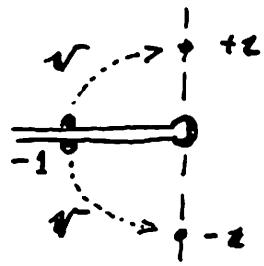
But $+\infty = 1/(+0) \neq 1/(-0) = -\infty$.

How can this be useful?

Conformal Maps of Slitted Domains

see ch. 7 of "The State of the Art in Numerical Analysis" (1987)
ed. by A. Iserles & M.J.D. Powell, Oxford Univ. Press.

e.g. $(-1 + 0i)^{1/2} = +0 + i$
$(-1 - 0i)^{1/2} = +0 - i$

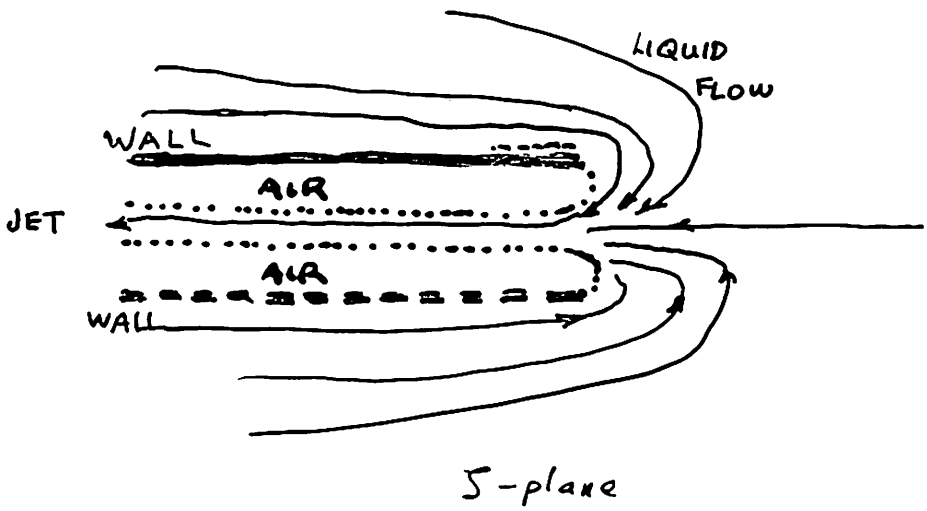


$i^2 = -1$

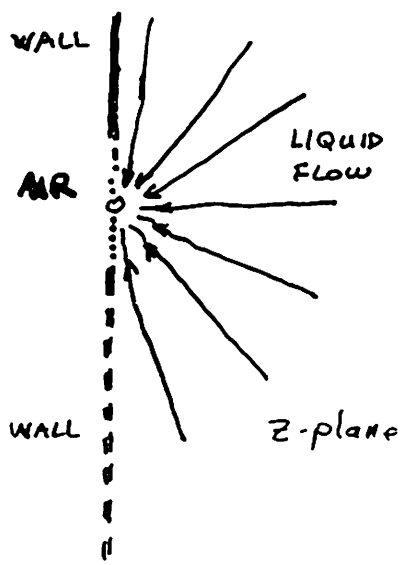
$\therefore\ (z^*)^{1/2} = (z^{1/2})^*$ is valid for $z < 0$ too!

But only for IEEE 754/854

Application example: $\zeta = 1 + z^2 + z\cdot(1+z^2)^{1/2} + \ln\left(z^2 + z\cdot(1+z^2)^{1/2}\right)$



ζ-plane



Without signed zero, lower wall disappears!

What you do NOT have,

THOUGH YOU PAID FOR THEM,

in your IBM PC-XT with: 8087
IBM PC-AT i 80287
SUN III μ 68881
...

are ways to access through FORTRAN, C, PASCAL, ...

1) EXTENDED PRECISION FORMAT (64 sig. bits, $10^{\pm 4900}$)
available on-the-chip, useful for
- MATRIX OPERATIONS like ITERATIVE REFINEMENT
(cf. PC-MATLAB)
- TURNKEY SYSTEM implementations
(cf. hp calculators)

2) CONVENIENT EXCEPTION-HANDLING
- FLAGS to signal INVALID OPERATION (NaN)
"DIVIDE-BY-ZERO" } ±∞
OVERFLOW
UNDERFLOW
INEXACT
- MODES to change response to exceptions

3) DIRECTED ROUNDINGS

BUT THESE ARE ACCESSIBLE ON MACINTOSH
thanks to APPLE's diligent supervision of higher-language implementations

MAJOR TASKS REMAIN:

1) INDOCTRINATION / EDUCATION / BRAIN-WASHING of Computer Scientists, especially Compiler Writers, to make them aware of issues they affect.

2) REFORM of Exception-Handling ...

- Retrospective Diagnostics ... to ease debugging of purchased codes.
- Presubstitution ... to alleviate dependence upon PRECISE INTERRUPTS
- Permit Vast Exponent-Range Extension in certain very special contexts ... avoid kluges in product/quotient loops

cf. DAVID BARNETT'S work on DEC VAX and SUN for Berkeley UNIX (1987)

Problem 2) is a TECHNICAL problem whose solution is now in sight.

But problem 1) is POLITICAL, and refractory.

1977-1987: The IEEE Standard 754-1985 / 854-1987
for Floating-Point Arithmetic (Binary) (Binary & Decimal)

Crucial contributors:

W. J. Cody	(Argonne N.L.)
J. T. Coonen	(UCB, Zilog, Apple)
J. W. Demmel	(UCB, NYU)
D. Hough	(UCB, Tektronix, Apple, Sun)
R. Navé	(Intel)
J. F. Palmer	(Intel)
F. N. Ris	(IBM)
D. Stevenson	(Zilog, ...)
R. Stewart	
H. Stone	(Amherst, ...)
G. Taylor	(UCB, ELXSI, MIPS)
J. H. Wilkinson	

BN Parlett >

cf. Wilkinson's letter to R. Delp, ≈ 1979-80

Hardware that conforms to IEEE 754:

Chips

Intel 8087, 80287, 80387 (used in IBM-PC, -XT, AT, clones)
Motorola 68881, 68882 (used in SUN III, new APPLE Macintosh)
WEITEK 1064/5 (used in SUN III, ...)
National Semi 32081 (used in IBM RT-PC)
AT&T WE 32106 (used by AT&T, Zilog)
Analog Devices, AMD, ELT, Fairchild "Clipper",
INMOS T800 (ready)
...

Computers: Macintosh & Apple II with S.A.N.E.
ELXSI 6400 Super Mini
Hewlett-Packard SPECTRUM series.
...

IEEE 754 : the single most widely adopted design for floating-point arithmetic in new computers.

IEEE 854 Decimal: hp-71B (standard basic)
hp-15C, hp-28C (internally or

ooo TRICKS ? NO !

UNPUBLISHED THEOREM

Rounding the way IEEE 754 or VAX does

(or, with a little more effort,

Rounding the way it is done on IBM '370
hp 3000
B6500
...)

imposes NO LIMIT except over/underflow

upon accuracy of computation using

fully portable (FORTRAN , PASCAL, ...) codes.

(Dekker 1971, Pichat 1972, ...)

This cannot be true for Brown's model .

(nor Olver's)

To do as well for CDC Cybers or CRAYs or UNIVACs

requires either CUMBERSOME

or NON-PORTABLE

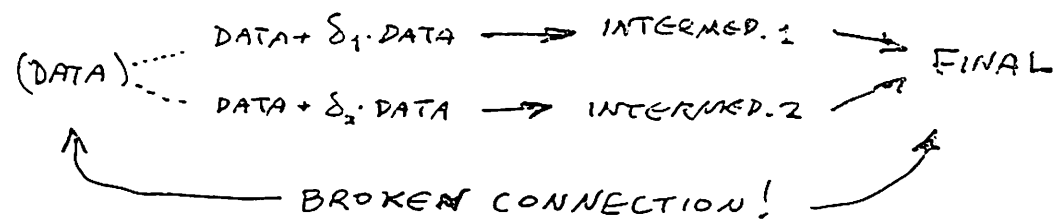
(or both)

programming methods.

e.g. $\dfrac{\text{Time for CRAY double prec'n}}{\text{Time for CRAY single prec'n}} > 50$

(D. Bailey, NASA).

DO NOT BE BLINDED BY MISCONCEPTIONS ABOUT "STABILITY" and "BACKWARD ERROR ANALYSIS" !



CRUCIAL PROPERTY of floating-point arithmetic on B6500, IBM'370, DECVAX, hp Calculators, IEEE 754/854:

Th'm on EXACT CANCELLATION during subtraction

> If floating-point numbers p and q satisfy
> $$1/2 \le p/q \le 2 ,$$
> then $p-q$ is another floating-point number, and will be so computed EXACTLY.

(unless $p-q$ underflows ... this CANNOT HAPPEN to IEEE 754/854)

THIS Th'm IS FALSIFIED BY
- Brown's Model (& by CDC, CRAY, UNIVAC, TI)
- Olver's level-index arithmetic

When the Th'm is satisfied by floating-point arithmetic, it permits ...

- Calculation of Area & Angles in a Triangle, given its sides, despite nearness to Needle-shape.
- PRECONDITIONING of ill-conditioned ...
  - linear systems $Ax = b \rightarrow PAx = Pb$ exactly
    $\kappa(A) >> \kappa(PA)$
  - Polynomial equations, esp. Quadratics & Cubics.
- Suppression of growth of roundoff like
  - $O(1/\text{timestep})$ or $O(1/\text{timestep})^2$ in TRAJECTORY problems, celestial mechanics
  - $O(1/\Delta x)^4$ in BEAM calculations $y^{iv} = f(y)$...

...

- ARE THESE JUST TRICKS?