Thacher Comments on Floating Point Indoctrination 4/18/88

COMMENTS ON FLOATING-POINT INDOCTRINATION SYLLABUS

p. 5. Project Suggestions. These additional topics, about suitable for a master's project, might be of interest. I can supply additional details and preliminary results to anyone interested.

a. Static representation of floating-point numerals. MACHAR and PARANOIA characterize floating-point by analyzing the results of floating-point arithmetic operations. It is also important for some purposes to know the representation of numerals in memory. This can be accomplished by manipulating the memory representation using integer or logical operations. I have already gotten the wordlength, base, and scaling factor, as well as the bits allocated to the exponent and significand. More things need to be worked out.

b. Testing Functions of Several Variables. Most protocols for testing routines for testing functions of a single variable include tests at a set of random values of the variable. For functions of two or more variables, the number of values rapidly becomes exorbitant. In quadrature, where the same explosion in data requirements is found, considerable attention has been paid to using "equidistributed" point sets (c.f. Stroud, Approximate Calculation of Multiple Integrals, Chap. 6). How well would these techniques apply to estimating at least the RMS error of a function routine? They should, of course, be supplemented by tests at critical points.

2. Comparisons of floating-point arithmetics in computers. I assume you are aware of Yohe's MRC reports on this topic.

I hope you will blast the manufacturers who advertise that BCD arithmetic is free of rounding errors.

Don't forget to mention the problem with early IBM 360's which they had to field modify because of no guard digit for multiplication in double precision.

What effect will RISC architectures have on floating-point calculations? Coprocessors driven by a RISC cpu? Microprogrammed floating-point operations? Interpreters?

Kathy Ward wrote a balanced ternary floating-point simulator as a M.S. thesis for me at the University of Kentucky in 1983. It includes C listings for the arithmetic and conversion routines. I can make my copy available for copying by anyone who wants to experiment with this system.

It may be of interest to note that the EDSAC 1, probably the first machine to have even programmed floating point, used a binary significand, with a decimal scaling base. Fortunately, they normalized the significand so that $1 < |f| \leq 10$, so that the integers could be represented exactly. Tom Kurtz, in the interpreter for his SCALP Algol processor for the LGP-30, chose the

normalization $1/2 \leq |f| < 1$, so that only multiples of 5 were exact.

Metropolis pushed unnormalized arithmetic on MANIAC III, as expounded in several papers by Aschenhurst. Argonne (Gray and Harrison, IEEE paper early 60's) proposed using an index of significance (i.e. a count of the number of left normalizing shifts) and built the floating-point processor FLIP which was attached to the old George machine. Unfortunately, by the time it was built, there was nobody around who was interested in using it for research.

On multi-precision schemes, Bob Gregory pushed a scheme very hard. It is described in a little book he published, my copy of which evaporated with some student. You can also find something about it in Young and Gregory's two-volume numerical analysis book. The scheme required too much programming for me to experiment with. I translated Hill's ACM Algol procedures into FORTRAN, and used them quite successfully for special tasks, e.g. computing the continued fraction for the exponential integral and sine and cosine integrals, and the zeros of the latter, and the coefficients of the Gram series for the Riemann zeta function.

Shouldn't conversion of precisions be included among the algebraic operations? This would emphasize the danger of assuming that a variable as stored is the same as the value remaining in an extra-length register.

Is Pichat as good a reference on precision doubling as Linnainmaa's 1981 TOMS paper? I transcribed it for Borland's Turbopascal 2.0, and spent quite a while discovering that their programmed floating-point did not use a guard digit in subtraction. Naturally, my complaint fell on deaf ears, and I upgraded to Turbo-87. I do hope they send a representative to these lectures. I haven't checked the emulators on their more recent releases. Another project, perhaps.

3. Models of floating-point arithmetic for programmers. Wasn't Dekker's paper one of the first to attempt to define what the programmer should be able to expect?

Should the model, and preferably a usable one, such as the IEEE standards, be an essential part of the language specification? As I remember, ADA tried this with the Brown model, but that is a bit hard to work with. I told Kurtz of the IEEE work when they were producing the new BASIC standard, but they didn't take advantage of that effort.

To what extent should accuracy checks be included in mathematical software? For example, Forsythe, Malcolm, and Moler's approximation to the condition number in DECOMP and SOLVE, or regenerating a polynomial from its computed zeros.

5. Real elementary transcendental functions. The key to most function routines is a nonanalytic operation: exponent manipulation, extraction of significand, and so on.

Relative error is a manageable, but inexact way of accounting for variations in the size of the result. The goal should be

accuracies which match the wobbling precision of the particular floating-point representation.

"Best" approximation is a game played by approximators. We should look for the cheapest good enough approximation. A successful approximation depends far more on range selection and the use of an effective auxiliary function than on tedious leveling of the error curve by adjusting the parameters. Many high quality numerical tables, including some in AMS 55, use good auxiliary functions to permit easy interpolation. With proper auxiliary functions, truncated Chebyshey series are usually good enough.

Not elementary functions, but pertinent to testing: Along with checks in critical regions where isolated extreme errors may be found, most testing protocols include tests at a fairly large number (2000-5000 say) of random arguments, preferably constrained to provide fairly even distribution over the domain. For functions of several variables, the number of evaluations to provide equal coverage of the domain becomes 2000^(number of variables). The project may reduce this somewhat.

6.Complex arithmetic. I have been looking at T-polynomials on regular polygons in the complex plane, because triangles, squares and hexagons cover the plane without overlapping as do circles for which the polynomials are the powers of z. The work is not complete, but they seem promising. I can tell you more if you are interested.

8.Numerical lapses in standard programming languages. Many of the problems we see may derive from the poor Algol 60 choice of real to indicate floating-point numerals. Floating-point numerals are all rational, at least. Furthermore they are enumerable. Each except for the infinities has a successor and a predecessor. Providing these functions would allow easy monitoring of precision at a particular value.

Ignoring integer overflow can conceal some errors, but warnings should be optional. Otherwise, random number generators are harder to write.

Most ABS functions I have seen return a negative value for the most negative representable integer.

Your objections to Pascal seem to have been eliminated in the latest version of Turbopascal 4.0.

I remember your complaining about the conversion of constants including the question of when it should be done, and that it should be done well, and conformable with the I/O conversion routines, but I did not find it in the syllabus.