Electronic Transactions on Numerical Analysis.
Volume 28, pp. 168-173, 2008.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

VARIABLE-PRECISION ARITHMETIC CONSIDERED PERILOUS — A DETECTIVE STORY*

DIRK LAURIE†

In memory of Gene Golub

The scene of the crime. In 2002, I was interested in computations to very high precision, having been challenged by Nick Trefethen to compute a number he called τ to 10 000 digits. This number is the sum of ten other numbers, each of which was defined by a difficult computational problem. The story is told in [1].

Before barging in to use unfamiliar software on those really hard problems, I needed to acquire faith in the chosen multiple-precision package, Pari-GP [12], by using it on some easier computational problems that I knew well.

I started by implementing a variant of the Golub-Welsch algorithm [7]. This has become the standard procedure for the calculation of Gaussian quadrature formulas from their Jacobi matrices, canonized in Gautschi's software package ORTHPOL [6]. Moreover, I already had working code in Octave [3], an open-source Matlab clone. Of course, this is just a toy problem to test the software — an actual computation of an integral to 10 000 digits would not employ Gaussian quadrature. See [1, Chapters 1, 3 and 9] for a discussion of methods better suited to very high precision.

In the version of Pari-GP I used, the floating-point support consists of a high-level interface to the GNU multiprecision package [4].

The body of the victim. All went well for the standard quadrature formulas. Next, I tried computing a seven-point Gauss-Kronrod formula [9] by the method described in [10]. Three of the points in this formula also belong to the three-point Gaussian rule.

Using Pari-GP's default precision (96 bits, about 29 digits) the non-negative nodes and their weights came out as follows:

j	x_j	w_j
0	`5.331911574339452454E-29`	`0.4509165386584741424`
1	`0.4342437493468025580020715029`	`0.4013974147759622229050518188`
2	`0.7745966692414833770358530799`	`0.2684880897`
3	`0.9604912687080202834235070925`	`0.1046562260264672651938238563`

> *In the case of the nodes x_0 and x_2 that are also nodes of the 3-point Gaussian formula, the weights w_0 and w_2 have been grievously mutilated, with respectively only 64-bit and 32-bit accuracy still being recognizable.*

The prime suspect. Unlike IEEE arithmetic, which has a fixed-length mantissa, Pari-GP (like most algebraic packages that offer high-precision floating point) works with variable-precision arithmetic. This is usually considered to be a good thing. Here is a small demonstration, with the internal representation printed out in hexadecimal below each number, omitting the first word, which contains exponent and sign information. (The actual arithmetic is genuine binary floating point, though.) Note that even though only 29 decimal digits of `cos(100*Pi)` are printed, the internal representation of its mantissa contains seven words:

*Received January 29, 2007. Accepted for publication November 18, 2007. Recommended by L. N. Trefethen.

†Department of Mathematical Sciences, University of Stellenbosch, Stellenbosch, South Africa (`dlaurie@na-net.ornl.gov`).

the precision has automatically been increased. Note, too, that `cos(100*Pi)` is not a perfectly accurate value of $\cos(100\pi)$, since `100*Pi` is only a three-word representation of 100π.

```
100*Pi
  314.15926535897932384626433 83
  9d1462ce  aa19d7b9  39bafcfd
cos(100*Pi)
  1.00000000000000000000000000
  ffffffff  ffffffff  ffffffff  ffffffff
  ffffffff  fffe0000  00000000
1-cos(100*Pi)
  2.088097429759527848 E-53
  80000000  00000000
1.-cos(100*Pi)
  2.524354897 E-29
  80000000
```

The interesting part is what happens in the two subtractions. The integer `1` is taken to have the same precision as `cos(100*Pi)`, and the first five words of the difference is now zero. By the principles of variable-precision arithmetic, only a two-word mantissa remains. When subtracted from `1.`, which is assumed to have the default three-word precision, only a one-word mantissa (the shortest possible) is retained. These shortened mantissas show up in the decimal printout too.

The investigating officer immediately suspected that exactly this culprit was at work in mutilating the weights w_0 and w_2.

A plea of "not guilty". As can be seen clearly from the above demonstration, because of cancellation, fewer significant bits remain after the subtraction. Shortening the mantissa is justified by the fact that there is no information available about the bits that would have followed. In fixed-precision arithmetic, and in some other implementations of variable precision, zeros would have been appended to make the mantissa length the same as that of the operands. There is no reason to suppose that the lost bits should have been all zeros. When later multiplications in fixed-precison arithmetic create extra nonzero bits, those bits are spurious.

Thus, this implementation of variable-precision arithmetic is a responsible, public-spirited citizen who by shortening the mantissa warns you about the precision loss.

Cross-examination by the prosecution. If the argument of the previous section is valid, then the numbers calculated by IEEE fixed-precision 53-bit floating point should be inaccurate, because the same cancellation would occur. These are:

j	x_j	w_j
0	0.000000000000000	0.450916538658474
1	0.434243749346803	0.401397414775962
2	0.774596669241483	0.268488089868334
3	0.960491268708020	0.104656226026468

In particular, we know from the variable-precision computation that an intermediate result in the computation of w_2 contained no more than 32 significant bits.

However, w_2 happens to be an exact rational number,

$$w_2 = \frac{12500}{46557} \doteq 0.26848808986833344072856928 07,$$

showing that all fifteen decimal digits shown of the 53-bit result are in fact correct.

The prosecution submits that variable precision, whether maliciously or by neglect, needlessly squandered valuable correct digits.

An accomplice.

```
Name:  Rational implicitly shifted QL algorithm
Known aliases:  TQL, PWK, GG.
```

An n-point Gaussian quadrature rule is intimately related to an $n \times n$ symmetric triangular matrix T, known as the Jacobi matrix of the Gaussian rule in question. Here T has diagonal elements $a_k,\ k = 1, 2, \ldots, n$, and subdiagonal elements $\sqrt{b_k},\ k = 1, 2, \ldots, n-1$.

A Kronrod rule also has a Jacobi matrix, in which the numbers a_k and b_k are rational numbers that can be found by the process described in [10].

The Golub-Welsch algorithm essentially consists of applying the implicitly shifted QR algorithm [13] to T. The standard implementation of this algorithm involves at each step the transformation of T by an orthogonal matrix of the form

$$Q_1 = G_{1,2} G_{2,3} \cdots G_{n-1,n},$$

where $G_{i,i-1}$ is a rotation matrix in the $(i, i-1)$ plane. The great contribution of Golub and Welsch was to observe that if these rotations are simultaneously applied to the first row of the identity matrix, thus obtaining the first row of the normalized eigenvector matrix of T, then this row equals

$$[\pm c\sqrt{w_1}, \pm c\sqrt{w_2}, \ldots, \pm c\sqrt{w_n}],$$

where c is a constant.

In the QR-based weight computation, one forms $\mathbf{e}_1^\mathsf{T} Q_1$, where $\mathbf{e}_1^\mathsf{T} = [1, 0, \ldots, 0]$. Thus the rotations are applied to $\mathbf{e}_1^\mathsf{T}$ from its left to its right, creating nonzeros as far as it goes. If the QL algorithm is used instead, the transformation matrix has the form

$$Q_2 = G_{n,n-1} G_{n-1,n-2} \cdots G_{2,1}$$

(not the same rotation matrices $G_{i,i-1}$). In forming $\mathbf{e}_1^\mathsf{T} Q_2$, the rotations are applied from the right of $\mathbf{e}_1^\mathsf{T}$ to its left. Only $G_{2,1}$ has any effect on the row; the other rotations operate on zeros. Thus $\mathbf{e}_1^\mathsf{T} Q_2 = \mathbf{e}_1^\mathsf{T} G_{2,1}$. As noted in [11], if an eigenvalue x_j is used as shift, the first element will then be $c\sqrt{w_j} = \cos\theta_1$, with the same c as before, and $\cos\theta_k$ the $(1,1)$ element of $G_{k+1,k}$.

This observation allows a square-root-free version [11] of the Golub-Welsch algorithm. We start from a square-root-free version `TQL` of the tridiagonal QL algorithm, such as those by Pal, Walker and Kahan (PWK) [13, p.169] or by Gates and Gragg (GG) [5]. These algorithms recursively generate

$$C_k = \cos^2\theta_k,\ S_k = \sin^2\theta_k, \quad k = n-1, n-2, \ldots, 1,$$

using rational operations only. The suggested procedure is:

- First find all the nodes by `TQL` using the usual shoft strategies [13, §8-14].
- For each node x_j in turn, apply `TQL` with shift x_j, then $C_1 = c^2 w_j$.

Can it be the case that this version of the TQL algorithm conspires with variable precision to yield an unreliable result?

Character evidence in favour of the accomplice. In the inner loop of the PWK algorithm, the following statement appears:

```
if C=0 then P=oldC*BB else P=gamma^2/C
```

The GG algorithm also contains the same statement.

Parlett goes on to say [13, p.168–9] that this formula, testing as it does whether a floating-point number is exactly 0, “appears to invite disaster” but “this is not the case”.

> *The PWK algorithm . . . avoids tolerances without sacrificing either stability or elegance.*
> Beresford N. Parlett, *The Symmetric Eigenvalue Problem*, [13, p.164]

Forensic evidence. When x is a Gaussian node in a $(2m + 1)$-point Kronrod formula, then it is shown in [10] that x is an eigenvalue of the trailing $m \times m$ submatrix of the Jacobi matrix T_{2m+1} of the Kronrod rule. In that case, the TQL algorithm with shift x inevitably transforms T_{2m+1} to a tridiagonal matrix with a zero element near the middle of the codiagonal.

Maybe this is a good place to note that another of Gene Golub’s contributions to this area, as part of the formidable team responsible for [2], is the invention of a method that implicitly makes use of the existence of T_{2m+1} without ever actually forming it. That method, based on arrowhead divide-and-conquer, is reported in [2] to be in many cases more accurate than the method in [10]; it may well be less sensitive to the vagaries of variable precision.

The zero element in the transformed T_{2m+1} shows up as a value γ_k which should be zero, but because of roundoff usually is instead a very small number. Moreover, γ_k is produced by subtraction and has very few significant bits: in variable precision, γ_k is truncated to single precision.

Since γ_k is not exactly zero, the branch $P = \gamma_k^2/C_{k+1}$ is taken in the TQL algorithm, so that all further quantities depend on γ_k and thus have single precision only. The divisor C_k can be shown to contain the factor γ_k^2, and appears to be a very, very small number indeed.

However, γ_{k-1} has the form $aC_k + b\gamma_k$, and thus contains the factor γ_k. In the expression γ_{k-1}^2/C_k, the numerator and denominator both contain the factor γ_k^2. When computed in floating-point arithmetic, they are both formed by multiplications involving the same imprecise value of γ_k, which therefore cancels when the quotient is formed.

> *This is the reason why the PWK algorithm does not, in fact, invite disaster.*

Saving the life of the victim. When a small γ_k arises, it should be promoted to full precision. Even though the extra zero bits are spurious, this does not matter because γ_k cancels out when γ_{k-1}^2/C_k is formed. The promotion causes later calculations to be done to full precision.

The TQL algorithm, modified in this way, gives in 96-bit variable-precision arithmetic a value $\tilde{w}_2$ such that

$$12500/46557 - \tilde{w}_2 \doteq \texttt{-3.997586568E-29}.$$

The verdict. When the numerator and denominator are both small, they contain the same inaccurate small factor, which cancels to leave an accurate quotient. This fact is established only by careful analysis, and variable-precision floating point cannot be blamed for being ignorant of it.

> *Accordingly, the defendant is found Not Guilty. However, this Court is of the opinion that variable-precision floating point is perilous, and its use should be restricted to qualified professionals.*

Appeal to the High Court. When the same algorithm is programmed in Maple, a Maple 7 run gives the results to full working precision. It is tempting to point the finger at Pari/GP as the profligate squanderer of useful digits, whereas Maple conserves them frugally. That would be totally misleading, as shown by another example, adapted from one given by Henrici [8, §1.4].

We calculate

$$f(x) = 10^{10}x\left(\sum_{k=0}^{100}\frac{x^{2k}}{(2k)!} - \sum_{k=0}^{100}\frac{x^{2k+1}}{(2k+1)!}\right)$$

when $x = 25.13274122871834590770114707$ exactly. (That is 8π rounded to 28 significant digits.)

First Maple 7.

```
> Digits:=28;
                                Digits := 28

> x:=2513274122871834590770114707066/100000000000000000000000000000.;
                      x := 25.13274122871834590770114707

> fx:=1e10*x*(sum('(-x)^(2*k)/(2*k)!','k'=0..100)+\
     sum('(-x)^(2*k+1)/(2*k+1)!','k'=0..100));
                      fx := 3.056528377627073124903438081
```

Twenty-eight digits bid and made — impressive.

Next Pari-GP.

```
? \p28
   realprecision = 28 significant digits
? x=2513274122871834590770114707066/100000000000000000000000000000.
%64 = 25.13274122871834590770114707
? fx=1e10*x*(sum(k=0,100,(-x)^(2*k)/(2*k)!)+\
    sum(k=0,100,(-x)^(2*k+1)/(2*k+1)!))
%65 = 3.056532957
```

A meagre ten digits only.

But are those Maple digits really correct? The two sums add up to a partial sum of the alternating series for e^{-x}, with the first neglected term being $x^{202}/202! < 2.3\,10^{-97}$. Thus we know that

$$f(x) = 10^{10}xe^{-x} \doteq 3.056532577159675451799 2069,$$

a figure on which both packages agree. Thus only the first five of the Maple digits, and the first seven of the Pari-GP digits, are in fact correct.

Before we start applauding Pari-GP for its well-founded caution, let us calculate $f(x)$ as a single sum instead of the difference of two sums.

```
? fx=1e10*x*(sum(k=0,201,(-x)^(k)/(k)!))
%70 = 3.056532589740303653188996 6825
```

Now Pari-GP has also been unable to detect the precision loss. For this insidious cause of inaccuracy, Henrici coined the term 'smearing'.

The point is not that Pari-GP is good and Maple bad or vice versa. The point is that in any variable-precision package, a decision is made on how to treat numbers given as data, or arising in intermediate results, which are represented in floating-point format to a precision lower than working precision. Do we promote them to full membership of the high-precision club, or do we treat them and all their associates as second-class citizens? Sometimes the first course is proper, sometimes the second, and it takes careful analysis to tell which. The decision of the Low Court is upheld.

REFERENCES

[1] F. BORNEMANN, D. LAURIE, S. WAGON, AND J. WALDVOGEL, *The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing*, SIAM, Philadelphia, 2004.

[2] D. CALVETTI, G. H. GOLUB, W. B. GRAGG, AND L. REICHEL, *Computation of Gauss-Kronrod quadrature rules*, Math. Comp., 69 (2000), pp. 1035–1052.

[3] J. W. EATON, *GNU Octave*. http://www.octave.org.

[4] F. S. FOUNDATION, *The GNU MP Bignum Library*. http://www.swox.com/gmp.

[5] K. GATES AND W. B. GRAGG, *Notes on TQR algorithms*, J. Comput. Appl. Math., 86 (1997), pp. 195–203.

[6] W. GAUTSCHI, *Algorithm 726: ORTHPOL – a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules*, ACM Trans. Math. Software, 20 (1994), pp. 21–62.

[7] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[8] P. HENRICI, *Essentials of Numerical Analysis*, Wiley, New York, 1982.

[9] A. S. KRONROD, *Nodes and Weights of Quadrature Formulas*, Consultants Bureau, New York, 1965.

[10] D. P. LAURIE, *Calculation of Gauss-Kronrod quadrature formulas*, Math. Comp., 66 (1997), pp. 1133–1145.

[11] ———, *Accurate recovery of recursion coefficients from Gaussian quadrature formulas*, J. Comput. Appl. Math., 112 (1999), pp. 165–180.

[12] *Pari/GP*, http://pari.math.u-bordeaux.fr. (An interactive programming environment for doing formal computations on recursive types, including rational and multiprecision floating-point numbers, polynomials and truncated power series.)

[13] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, 1980.