Ball & Beebe Paper 2 PhL 10.18.04

Algorithm xxx: QUADLOG -- A package of routines for doing the log stuff.
Beebe and Ball (author order reversed from paper #1).

This paper outlines a software package for implementing the ideas presented in paper #1. I get the feeling that it was written 100% by Beebe, and it is just a bit long winded.

1. Introduction. They are interested in the same two integrals of the form (1) and (2).

2. Implementation and Portability. The author is concerned that this code run with ALL Fortran compilers, and there are now about 4 different year-specs: 1977, 1990, 1995, High Perf. He is concerned about character sets and the number of letters in routine names (hard to imagine that as an issue but I guess it is). Not sure how he gets TEX into Fortran comments.

The B&B code is Fortran, they did not code it in C or Java or anything else. There is a free f2c converter, which is news to me, and a commercial one too from Cobalt Blue. Claim is F2C really works and they tested it a bit.

They have tested portability by using 40 different systems, I guess the math department has ways to do that, or maybe a grad student unnamed did the tests.

3. Packaging Issues. They plan (no location given) to distribute two tar files, one is their quadlog, the other is gampsi used to compute Γ and ψ functions to high accuracy. This section talks UNIX stuff about how to install all this stuff. By the way, nothing they do runs in Windows as far as I can tell, so it is thus not accessible to me. Maybe they just did not test it with Windows.

4. The Laguerre Case. Recall that we have three ways to compute things, here (1) and (2) and (3).

Call: glqfd : Inputs = nquad = N and alpha = α N < 1024 in their implementation
Outputs = x, W, δx and δW and ierr

These are the things you need to evaluate the form (1) or (3). Once you get all these numbers back, you have to then do the little addition shown on page 8, and using the dvsum thing is a little more accurate, I can imagine it stores more bits in the intermediate sum.

Call: glqf: Inputs same as above
Outputs x,y, W, Z, W(x-1) and ierr

Obviously this is used with form (2) where you don't need f'(x). Again, you add things up after doing this call.

Call: glqrc Same as above.
Outputs = a,b,s,t (I am unclear on this algorithm)

All the above routines have quad precision versions that begin with the extra letter q.

5. The Jacobi Case. Everything is the same here, but replace letter "l" with letter "j" in the calls. Yes, the sums are slightly different, eg, ln2 appears on page 11.

6. Overview of Testing and Evaluation. Reasons for the need to test are stated and I agree with them. He notes that you cannot get the same LS bits from different machines due to order of evaluation and such. Page 13 claims Pentium is 1 ULP and IA-64 is maybe .6 ULP due to rounding. Mention of the ndiff program to compare things to some preset level of accuracy.

7. Test coverage analysis. This is just the issue of testing all your lines of code at least once.

8. Testing the Γ and ψ primitives. This is a very long and wordy section. I know that both these functions must be evaluated for the test cases, but maybe they are needed inside the algorithm as well, perhaps only Γ in that case. The table on page 17 for Γ shows pretty huge ulps for certain normal system implementations! By the way, notice in this table that when quad is used, the answer is presented as 1.1e-16 which is exactly half my LSB $\Delta = 2^{-52}$, so this is just what I would expect! (Some are 1.08, reason unknown). B&B are doing another paper where they describe their special quad-based Γ and ψ routines. If they round these results, then they will be 0 ulp, so to speak.

9. Testing the machine-ε primitives. Talks about code used to measure the smallest Δ you can have, which should be .000.....0001 and talks about problems encoutered!

10. Testing the eigenvalue solver. The EISPACK is more self-contained and compact compared to the newer LAPACK. They used TQL1. Comment on pythag routine to improve square roots calcs that are involved inside.

11. Testing the Gauss case. Equation (9) shows a test case -- ie, the integral can be done exactly. They have test routines to do this and display the ULPS your system is getting. Test 1 does (9). Test 2 does (10). Test 3 (page 23) continues test 2 using the recursion for different p. Test 4 tries to integrate a sine which is not a poly. (the no log case). There are also tests 5 and 6, and they all can be run in quad versions. This section , however, does not quote any results!

12. Testing the Jacobi case. Again, the test program compares the quadlog routine against exact results for one or more simple integrals of the Jacobi class.

13. Conclusion. OK.

Appendix: What are ULPS ?

Think of the closest that two numbers can be in 64-bit FP. First, if you had a 2-bit mantissa after the point, you would say .11 and .10 are as close as you can get, so $\Delta = .01 = 1/4$. Your quantized result could be as much as 1/4 from the exact infinite-bit answer, barring rounding. So in 64-bit, there is a 52 bit mantissa and an extra 1 sign bit. So the $\Delta = 2^{-52} = 2.22 \times 10^{-16} = .0000\ 0000\ 0000\ 0002\ 2$.

Now Jim in paper #1 is plotting this:

$\varepsilon = (x2 - x1)/x1$

For example, he has a bolded value of 9.91e-15 = ε. Now, if the answer itself were x1 ~ 1, then you would say that (x2 - x1) = ε * x1 = 9.91e-15. If you talk in terms of 16 decimal places, then your error is 99.1e-16, and you would say you had 99.1 ulps. Here imagine x1 is a computed answer and x2 is an exact answer, so the ulps is how far they differ "in the last decimal places". But, suppose you had x1 ~ .8. Then you would have (x2 - x1) = ε * x1 = 9.91e-15 * 0.8 = 79.3, and you would have only 79 upls. In general, we know that the mantissa x1 value is between 0 = .0000000 and 1 = .1111111. Thus, you see how the ULPS can be less than the relative error.

What would Jim's results look like with a perfect algorithm? The ULPS would be 1 (or 0.5 with rounding). So with 91 ULPS, he does have some significant error. This means he is OK to about 14 places, not 16 places.

In quad, his results are still with this same UPLS range, but of course the relative error is much smaller, he does not have a table of this.