Ball & Beebe Paper #1 PhL 10.19.04

Efficient High-Accuracy Quadrature for Two Classes of Logarithmic Weight Functions

no date, not yet published

This is a rework of Jim's solo paper of similar title. There are many improvements:

- all the many errors have been fixed, I could find none left
- references are given for all obscure results, such as A&S
- references use dates like [1784] which I find very interesting
- the idea that there are two methods is made clearer (really 3 methods)
- some experimental test results have been added.
- there is more comment on the work of other authors
- probably some reviewer suggested fewer numbered equations

Section 1. So let's try to take a **higher level look** at this thing now. In (1) and (2) we have two integrals we want to do numerically. Why can't you just include the log factor inside the weight function? In the first case, for x<1 the ln(x) weight goes negative. In the second case the same is true, for example ln(1 - .5) = - ln 2. So OK, then why not take the log along with your f(x) function? The problem here I suspect is this. As you consider more nodes x_i, some of them get very close to the problem endpoint (x = 0 for (1), and x = -1 for (2)) where the log diverges (negatively, it happens). For example, ln 10^{-10} = -10 ln10 = -30. Perhaps this relatively large value swamps the quadrature sum somehow. They should make it clearer why there is a problem (maybe they do later in the paper).

We are reminded that if f(x) is a 2N-1 poly, there are 2N free parameters that can be adjusted, and GQ has exactly that number and should be exact for this class of f(x).

The d/dα method causes derivatives f'(x_i) to appear, and a new comments has been added that this can be identified with "Hermitian Interpolation". This is exactly what I read about in Chapter 10 of the Scheid book, the first order osculating polynomial fit to a function with "Hermite's Formula", page 65. This is maybe better presented on page 125 of Chapter 15, where we have the A and B weights to worry about, and the B's are for the f' values. I think the idea is this: you can use any set of x_i and then the 2N coefficients are the A and B, or you can use the nodes for x_i and just the A. That is a bit unclear to me.

The catch 22 is this: in the GQ or Hermite version world, the A and B coefficients require you do do integrals of polys against your weight functions, see for example page 128 Scheid. This is how you project out the weights. But this begs our question, because we are trying to find a method to accurately integrate polys in this manner! I think this is a plug for the Jacobi matrix method.

Section 2. Now a high level look at section 2 where the method is developed. The weight is taken as shown in (4), and this fits both our problems with the right choice of x_0. But (4) is not the integral we want. The integral we want is the d/dα of (4), as shown below (4). We know how to do (4) because it has a pos def weight function without the log. So the idea is to do that one by GQ, then do the d/dα to get the thing we want. You do worry a little about the idea of a derivative causing accuracy problems.

Now regular GQ gives the result (5) and the claim is that this can be done "exactly" for 2N poly f(x). What does that statement really mean? I guess if you have a computer with infinite bit storage for a number, you can do the exact Jacobi solution.

So the big result then is (6), unfortunately split onto two pages. This (6) does require both f and f' at the nodes x_i . It also requires **three other quantities** called W, dW/dα, and dx/dα. So the game here is to write expressions for these three other quantities. The results are shown on page 6. You assume that the

Jacobi method gives you both the x_i eigenvalues, and the $P_n(x_i)$ eigenvectors (being a vector as index n is run from 0 to some large N like N = 100.). The Jacobi method just gives you NUMBERS for all these things. Notice that you must specific a particular value of α to get such numbers, you cannot just leave it a variable.

So in (11) you add up the set of numbers shown to get $S_i = 1/W$. Then (12) shows how to get the needed quantity $dx/d\alpha$. Here we need the recursion A and B and A' and B'. How do you know the A and B recursion coefficients based on your weight function? I know these in terms of the k and k' leading coefficients of the polys. I think Jim does not INTEND to know these in the general case, because he is only going to do real calculations in the specific cases of the next two sections. But IN THEORY for any choice of parameters, there are some A and B recursors that are functions of α. Fine.

Next, in order to get the final quantity $dW/d\alpha$, we need the messy R and Q things. These involve derivatives of the P_n wrt to x and α. Now THIS is certainly hard to come by in a numerical only world or the general case. But in the specific cases, we will have this information!

OK, I now have a better understanding of the idea. There is no intention to try to do the general case for an arbitrary g(x) and arbitrary x_0.

Section 3. Now a high-level look at section 3. Bang, we set $x_0 = 0$, $g(x) = e^{-x}$. For this weight function, the polys are the well-known "generalized Laguerres", but they are normalized versions which makes them different from the book versions. We see the recursor A and B values in (19) and (20) along with the derivatives, very good. Then above (22) we see the dP/dx. But where is the $dP/d\alpha$? I think they should say something about this! I cannot find such a formula anywhere. This is a good question to ask Jim!!! Also, could do the paper for both HG and CHG functions maybe. This missing derivative is mysterious to me!

Now suppose you don't want to use your f', then we move to the second method. Let's take a high-level look at this second method. It is an "add and subtract" method. You find a weight function that is linear in your desired log, but which is non-negative. So (23) gives how this might be done.

What is equation (24) trying to say? The second two terms represent the lnx portion as in (17). And the first term represents the (x-1) portion. But why should the weights for the (x-1) part be the same as for the second part? The weights as in (11) are entirely determined by the polys and the nodes of the polys, so the weights are the same no matter WHAT function you apply GQ to, duh. In Scheid remember that the weights are integrals of the weight function times the Lagrange multiplier poly. So yes, (24) makes perfect sense to me, but I don't think they intend to use it since it contains the f' values.

Their plan is to treat the LHS of (24) as a normal GQ problem with weight function as in (23). They have a method of finding the recursor A and B coefficients by doing integrals, but how do they then do those integrals?

A new part is added here now. Instead of working with the Λ polys which are normalized, they are going to use instead some M polys which are monic normalized. Then (27) is the new recursion relation and the a and b are the recursor coefficients which you develop iteratively from the t and s integrals shown. But still, how do you DO these integrals? I guess by the Jacobi method! Notice that the t_n are just the usual h_n normalization numbers, while the s_n have an extra x in there. This reminds me of the Erdelyi method where you have all those c's.

OK, things are messy now. In order to get the s and t, they must do their numeric quadrature trick. They now have the a and b recursion coefficients for the next Jacobi application. The previous Jacobi application gives them the nodes now called y_i of the Λ function, and top of page 9 gives the Λ values. You then do the standard GQ Jacobi method to get the Z shown in (32).

So here are the steps:

(1) do Jacobi using recursion (27) to compute the t,s,a,b,M,Λ numbers. But something is fishy here! In order to diagonalize the Jacobi matrix, you need to know the values of the a and b. But these are the numbers you are trying to compute by doing the integrals! Maybe some kind of Newton method??

(2) Once you have the nodes y_i and the poly numbers Λ, you compute the sum to get S hence Z and that is where (32) comes from.

So I was wrong, there is only one Jacobi process going on here, not two or three.
OK, things are still hazy in this section!

Section 4. Again we have specific values $x_0 = 1$ and $g(x) = (1-x)^{\alpha}$. The polys are stated, the A and B are stated along with their derivatives, we get the dP/dx derivative, and again the dP/dα derivative is not mentioned. Everything goes along in exact analogy. For example, the pos def weight is (44). The statement (24) has its analog in (45), with some confusion about minus sign but all is OK. This is the formula that seems useless to me, but it is correct. Equation (46) is analog of (34). And (47) is like (36). And then (48) is like (32). And (49) is like (35)

My only criticism of this section is that it should be more parallel to the previous section. The equations are presented in a different order in this section. Maybe "analogous to (36)" would help.

Section 5. First, consider Table 1. I think when Jim refers to equation (36), he implies that we are supposed to be thinking that $f(x) = \ln(x)x^n$ and then this finally shows why regular GQ is poor on this for small n values at least where the singular end is enhanced. (the small n, small x problem area). He should state what the perfect single and quad precision results would be if only quantization was considered. They compare to a certain "adaptive" CG routine. They talk about "error in the last place" which confuses me. The table itself contains relative error ε I presume. OK. the bolded items are in fact the largest ε as n is varied for the three methods. What does he mean by "21 units in the last place" ?

For 64-bit FP, the mantissa is 52 bits plus 1 sign bit. Excel site says you get 15 decimal places accuracy. Can I confirm that? Error with mantissa = 2 bits would be what? .11xxxxxx error could be about as much as the last place, or 1/4 or 2^{-2}. So I guess error with 52 places is about $2^{-52} = 2.2 \times 10^{-16}$. This then would be the "intrinsic error" without rounding. With rounding, maybe cut this in half. What does this mean in terms of decimal places? The error is ε = .0000 0000 0000 0002 2 ~ .0000 0000 0000 0002. It is about 2 in the 16th decimal place. Now, Jim's error 9.91×10^{-15} would be .0000 0000 0000 0099 so you could say this is 90 units in the "last place". I am confused by Jim's comments, maybe the other paper will clear things up.

Table 2 shows that including the log in f(x) is VERY bad, the (47) column. Again, confusion about units in the last place.

Section 6: Related Work. This is where the referee probably made them add stuff. Danloy, Krylov, and Gautschi are mentioned.

OK, conclusion read and I am now done with this baby. Let's go look now at the second paper and see if it answers some of my questions.

Note added on first order perturbation theory.

$TV_i = x_i V_i \qquad (\delta T)V_i + T(\delta V_i) = (\delta x_i)V_i + x_i(\delta V_i)$

Then apply V_i^T from the left to get:

$$V_i^T (\delta T) V_i + V_i^T T(\delta V_i) = (\delta x_i) V_i^T V_i + x_i V_i^T (\delta V_i)$$

If I ignore the (δV_i) terms and use $V_i^T V_i = S_i$ and set $(\delta T) = T'$ then I get your result. Why can we ignore the δV terms? Here is why. They are

$$V_i^T T(\delta V_i) - x_i V_i^T (\delta V_i) = [\, V_i^T T - x_i V_i^T \,] (\delta V_i) = [\, 0 \,] (\delta V_i) = 0$$

because T is symmetric and $TV_i = x_i V_i \Rightarrow V_i^T T = x_i V_i^T$. There you have it!!!