All new notes on Ball/Beebe #1 PhL 1.19.05

These notes are "by the page", but I also tried to make them by the section as well.

page 2

The integrals of interest are (1) and (2), in which I recognize the Laguerre and Jacobi weight functions. Authors claim that the ln(x) factor causes trouble when you try to do direct GQ on these integrals. We get evidence of this fact later on in Tables 1 and 2.

It is claimed that to integrate degree 2N-1 polys by quadrature, you must have 2N parameters. I know from a certain Theorem 1 (see Accuracy of GQ in ortho poly binder) that in general you will need an n+1 point GQ to handle a poly of degree n, so for degree n = 2N-1 you would need n+1 = 2N points, that is true. In this general GQ, you pick your x_i perhaps as equally spaced stupid values, and then you will need 2N A_i weight values. In this method, all 2N parameters are weights. But we know if we pick x_i as the zeros, then we only need N weights. So OK, you can regard your 2N parameters as $\{ x_i, A_i \}$.

The next claim is that is that you can compute the $\{ x_i, A_i \}$ from the recurrence relation. Well, I now know that you can in fact compute the polys $p_n(x)$ from the recurrence coefficients. [In fact, given (a,b) and w(x), we can compute both those coeffs and the polys!] Given these polys, we then know how to compute the zeros x_i of $p_N(x_i)$. I also know that certain sums of these give the weights A_i , so yes, I agree with this claim. [Again, if you just do the Wilf Jacobi matrix thing, you get $\{ x_i, A_i \}$, where $1/A_i$ is a finite sum of the eigenvector components squared.]

Some reviewer complained that B&B were merely doing **Hermite interpolation**, so why write a paper on such a well known subject? Their response is that yes, there is such a thing as HI, but in that method the weights are given as integrals of polys against the weight function (see eg page 128 Scheid, where A_i and B_i are the weights, and yes, the $L_i(x)$ and other factors are all polys). But if you try to use HI to do GQ on polys of degree 2N-1, you need all these weights, but to do the weights, you need to integrate polys against w(x), and yes, it is indeed "Catch 22".

page 3

We start with section 2 here, skipping the summary-of-the-paper remarks. All equations on this page have been derived by me in Jim's earlier paper. The basic idea is that we do conventional N-point GC on the integral shown in (5), getting weights W and zeros x_i that we can regard as functions of α. Then we differentiate this integral wrt α to get the integral we are really after which has the log sitting in it. We end up with the **main result (6)** which unfortunately is split onto two pages. You can see why someone might say it looks like Hermite interpolation.

page 4

We now have a statement of the matrix form of the recursion relation, where we see our Jacobi matrix. But now we have "yet another" choice of symmetric recursion coefficient labels. Comparing to our previous Jim form, we would say that $\alpha_n = A_{n+1}$ **and** $\beta_n = B_n$, we he has offset the index on A by one unit. This does cause the matrix to at least look reasonable. The orthonormal eigenvectors are called P_n.

As I show in my Hilbert Spaces notes, you can write the GQ weights as shown in (9) where you do a sum of your P_n^2 having N terms. Yes you can use the C-D formula to get the second part of (9), but they won't be using this fact, they will do the sum. Notice that **P' is defined here** and will be used throughout as dP/dx, so we won't use this notation for the α derivatives.

page 5

Our formula (6) requires W_i, $dW_i/d\alpha$, $dx_i/d\alpha$ and finally f(x) and f'(x) at the x_i. This certainly looks foreboding and messy. B&B develop these quantities one at a time. The first act is to think of varying the α parameter a bit so you can think about derivatives relative to α. You then end up with $dx_i/d\alpha$ being in the usual perturbation result which here is $V^T\ \delta T\ V\ /\ S_i$ (see Hilbert Space notes). The matrix of $\delta T = T'$ simply has A and B in the T matrix converted to A' and B' wrt α, and this is all written out. So we have an expression for $dx_i/d\alpha$ as shown. For $dW/d\alpha$ we just differentiate our sum rule for S_i and now we have to worry about the TOTAL derivative $dP/d\alpha$, which is then broken down as shown into its two parts. To avoid confusion, we use $\partial P/\partial\alpha = \phi$. Now when all the dust settles, we get the results shown on the next page, but there is another important issue here. At the bottom of the page we see recursions for P' and for ϕ. The ϕ recursion is found simply by differentiating the P one wrt α and has three extra terms. Everything looks fine here. The thing to understand is that you can in theory use these recursions (10) to bootstrap expressions for the P' and ϕ AND for the recursion coefficients, all at the same time, using something like the Stieltjes procedure. This is the only way that this paper gives the reader for computation of the ϕ, for example.

page 6

On the top half of the page we just assemble all the pieces already computed on page 5. Nothing is new here, and our formula (6) now appears as (17).

Laguerre Case. Now we move at once to the Laguerre case. We have to first get normalized versions as in (18).

page 7

The recursion coefficients are just quoted from the Laguerre literature as in (19) and (21), and we can do the diff wrt α explicitly as shown. We then roll out the now-famous "diff formula" which exists for all classical polys to get P'= dP/dx. Now authors mention that we have to use some recurrence formulas. In particular, we can put in all the known stuff and then we have to create the ϕ_n by bootstrap from (10) second equation. At this point we are supposed to "turn the crank" and compute the "2N coefficients" which appear in (17), and this refers to δW_i and δx_i shown there. For some reason, they don't count the x_i here as things you have to compute, but we know that you do that from the Jacobi matrix. So you might then want to say there are 3N parameters needed in all.

Now where I have drawn the line at mid page we have a **sudden change of topic**. Suppose you cannot compute $f'(x_i)$ for some reason. They are going to develop a whole new method to handle this situation.

The no-f'(x) method. The first step in this method is to consider the weight shown in (23) and to note that it is positive definite as a good w(x) should always be.

page 8

Look at (24) with our new weight function. We can write the -lnx part as - RHS(17), and this then explains the last two negative terms on RHS(24). We then treat (x-1)f(x) as F(x) and do conventional GQ on that (with the original w(x)) , call this GQ#1, and this gives the first term W(x-1) of RHS(24). Then we take the entire LHS(24) and do it as well by conventional GQ as shown in (32) on page 9. However, since this has the new weight function, the weights are Z_i instead of W_i, and the nodes are y_i instead of x_i. Call this one GQ#2. If we can do all these things, then the integral we want is :

{ stuff we want } = GQ#1 - GQ#2

Since neither #1 nor #2 requires evaluating f'(x) anywhere, we will have achieved the desired goal.

Now most of page 8 is a discussion of how to do GQ#2 with the new oddball weight function. We don't know what the ortho polys are for this w(x)! It is no longer just the Laguerre weight function. So this page describes the Stieltjes bootstrap method (which I have written up in "Addendum"), and it all makes perfect sense. The ortho polys are called Λ_n and the recursion coeffs are called script $\mathcal{A}$ and $\mathcal{B}$. The paper then outlines my 6b method for computing the recursion coeffs in the "Jim" basis [as summarized in equations (25) and (26)], with Stieltjes' name attached. Yes, once you have the recursion coeffs, you can then generate the ortho polys as well. [Instead of doing bootstrap, you could have the Jacobi matrix routine return the eigenvectors, but that is no good because we need them analytically so we can integrate them as part of the method, as in (28).]

But then we don't use the 6b method due to the subtraction error complaint, and we move instead to the monic method (my 6c) and this is outlined in (27) through (31). The monics are called M instead of Λ, and we now have the Gaustchi coefficients a and b, and this is what causes those square roots to appear in the Jacobi equation -- all that stuff is explained in great detail in my Addendum to Erdelyi item 3.

So to summarize this page, they are going to use the "monic Gautschi method" to compute the recursion coefficients a_n and b_n, and the monics M (if they are needed). All these things go with the new weight function which we have for GQ#2.

page 9

As part of the Gautschi monic bootstrap method (my 6c), you must use a computed ortho at one value of n to compute the recursion coefficients for the next level up. You have to do integrals like those shown in (28) and (29). But we won't have closed forms for these messy things (weight is messy), so B&B actually do these integrals with more GQ. They find that you should use N+1 or N+2 points doing these to maintain accuracy. Since the f(x) being integrated are polys, this GQ is "exact" . Oh yeah, you do have to go off and compute the nodes y_i for these polys, and that will come by diagonalizing the Jacobi matrix with the b and $\sqrt{a}$ coeffs sitting in it as noted bottom of page 8.

In a digression, authors quote Gautschi saying that the method they are proposing to use should not work due to accuracy problems. Authors then counter with an explanation of why they don't have this problem in what they are doing. One suspects this arose from some referee on the paper.

So let's assume that the ortho polys M have been computed, and we get the weight Z by doing the sum in (9), and of course we get the y_i from Jacobi diag. We end up with (32), which we already took note of in the previous pages notes.

Finally, they comment on the GQ#1 I mentioned on the last page which is the (x-1) part of LHS(24). Here you see more directly that you will subtract (33) which is GQ#1 from (32) which is GQ#2 to get your final answer (with no need to compute f'(x) anywhere).

In the last paragraph, they note that you have to do N+1 evaluations of $f(x_i)$ to do (33), and you need only N evaluations of $f(y_i)$ to do (32). I am not quite clear why we don't use N-points for each one.

page 10

So, our two methods for doing the integral of interest are summarized in (34) and (35), where (34) is the originally derived method which requires $f'(x_i)$, and where (34) is the method we have just been discussing. And (36) just reminds us of the problem we start with before we diff wrt α to get that log integral. I was wrong thinking that f(x) had been redefined, it is the same f(x).

Jacobi Case. Here we get undersay in parallel with the Laguerre case. Write (37) for normalized Jacobis, then look up the recursions and their β-derivatives in the literature.

page 11

Get dP/dx from the diff formula (42). Then we have to go off and do our recursion to get the ϕ_n. So the outline of the plan here is same as before. The f + f ' form is shown in (46).

And as before, suppose you don't want to involve f '(x). Then we use the fancy but posdef weight shown in (44) and we arrive at (45). The LHS we will do directly with the fancy theory and this will be GQ#2 as shown later in (48). The ln(1+x) part is the desired final integral, so this time it appears as two + terms on RHS. The negative term comes from the ln(2). Now, the full fancy f and f ' result is shown in (46). Authors' wording is strange because (46) is really the "first method", but fine. We will do the ln(2) using GQ#1, and this is shown in (47).

page 12

So we end up with (49) as the no-f ' method. Mercifully, we are spared all the details of how we bootstrap up the polys and recursor coeffs for this fancy weight (this was done in detail in the Laguerre case to show the method).

Now as I well know by now, this **Jacobi** case includes a lot of other cases, as discussed in the Erdelyi classical overview notes. **Hypergeometric Polynomials** F(a,b;c;z) , then $\alpha = \beta$, to get **Gegenbauers** $C^{\lambda}_n(x)$, aka **ultrasphericals.** Then $\alpha = \beta = 0$ gives w(x) = 1 and you have the **Legendres** aka **sphericals**. Another special case is when $\alpha = \beta = -1/2$, you get the **Chebyshev** polys. So they are covering all these cases, but they don't mention the hypergeometrics for some reason, maybe not a popular poly name.

What are the main differences between this Jacobi and the previous Laguerre case?
(1) two parameters α, β instead of just α, and this time they do derivatives wrt β
(2) different simple and then different fancy weight function

5. Sample Tests. If you select $f(x) = x^n$ in our two examples, you get known integrals as in (50), (51) for Laguerre and Jacobi, so we can TEST our methods! So we have a separate test for each integer "n", and they will be testing n = 0,1,2....39.

page 13

Error sources? One is adding up terms in (9) to get the W_i. Other sources seem routine to me. B&B#2 paper will provide double and quad routines.

Table 1 (N = 20 for all tests) is the Laguerre (50) test with $\alpha = -15/16 \sim -1$. This causes the $w(x) = x^{\alpha} e^{-x}$ to blow up at the lower endpoint of (a,b) = $(0,\infty)$, to challenge the method. Even though this gets multiplied by x^n, we are still doing all our work really with w(x) and it is blowing up.

Now, if you just try to do the result directly using (36) where you put the log inside f(x) and use this w(x), you get very bad results as shown for N, see the (36) column where things finally get OK around n = 13, because eventually x^n helps you out.

If you use the "original method" with both f and f ', that gives the (34) column and results are uniformly very good for all n values including n = 0. This is the best column. If we use the no-f ' method, we get the (35) result which is 2-3 times worse than the original (34) method, but still is very good. Maybe a whole decimal place difference.

Table 2 is a similar test for the Jacobi case with $\alpha = \beta = -15/16$, so really doing Gegenbauers. Instead of x^n we really have $(1-x)^n$ and this does nothing to fix up the problem at x = -1 as n increases. For this reason, the "do it with blind regular GQ" method of column (47) gives completely useless results for all values of n ! The other two methods work pretty well, and again the f + f ' method is better by about 1 decimal place.

Comments: the 0's in the columns indicate places where accuracy was "perfect" to machine precision. Computing either side of (50) gave the same answer. So all in all, this looks like a solution for people who need to do this kind of integral. In 64-bit math, there are 52 mantissa bits and I think the truncation limit is then about 2.2×10^{-16}, with rounding cutting this in half. (see previous notes on this paper).

Related Work. Two Russians did the cases shown bottom page 13, two of which are special cases of B&B. Danloy 1973 did something in the Jacobi area. However, his work was $O(N^3)$ and B&B claim to be $O(N^2)$.

We then come to Gautschi 1994 (a "landmark article") who created a dynamite ORTHPOL Fortran package for doing lots of fancy ortho poly things, but nothing except one example with a log, and for this example, BB and he basically agree.

Web quote: " Walter Gautschi published a Fortran package ORTHPOL in TOMS, recently followed by a Matlab version (90 functions) called OPQ.

http://www.cs.purdue.edu/archives/2002/wxg/codes/OPQ.html // it is all there!

This set of Matlab codes is a companion piece to the book ``Orthogonal Polynomials: Computation and Approximation", Clarendon Press, Oxford, 2004. The routines, among others, implement all computational procedures discussed therein and provide code for the examples, tables, and figures. The book is is referenced below as ``OPCA".

I note that B&B do not reference this new book. (August 30, 2004, on Amazon, $130, 300p)

page 15

7. Conclusion. We messing with the Jacobi, they realized they could do [w(x) ln w(x)] as well, where w(x) is the Jacobi weight. Maybe do higher order perturbation theory to get higher powers of logs. Not clear where you would go next with this idea. Getting out a software package would be good.

page 16, 17

I read through the references and added some red lining. So, I think I finally have a solid understanding of this paper! Why did they not do the Hermite case I wonder? (the third classical $w = e^{-xx}$)

Concerning my list of questions to Jim

Here I will quote everything from that first notes document, and I will answer my own questions:

Question #1: One of the ingredients is a quantity Q, which is a sum of terms including a factor called ϕ_n which is defined as $\partial P_n^{\alpha}/\partial\alpha$. This goes into the expression for $\delta x_i/\delta\alpha$. I notice that in both the Laguerre and Jacobi discussions, there is no expression given for $\partial P_n^{\alpha}/\partial\alpha$. Nor could I find one for Laguerre in a brief search, nor could Maple tell me the answer (but my Maple skills are weak). How did you handle this? What did you use for $\partial P_n^{\alpha}/\partial\alpha$? Readers might be interested, and may see this as a strange omission.

> **Answer #1:** As discussed above, in general you have to bootstrap ϕ_n up from nothing using the recursion relations (10) on page 5. We know how to do this just fine. A simpler alternative might be to apply $\delta/\delta\beta$ to a form of the Jacobi's such as the series (12) on page 169 Erdelyi, or apply $\delta/\delta\alpha$ on (7) on page 188 for the Laguerre case.

Question #2: At the bottom of page 8, you are talking about doing the Jacobi Matrix trick with some a and b coefficients, tridiagonal and all that. Why do you have $\sqrt{a}$ instead of just a on the off diagonal? I must be missing something here.

> **Answer #2:** It is because we are using the "Gautschi form" a,b in (27), and not the "Jim form" as in the equation prior to (25). See detailed notes elsewhere on this subject.

Question #3: In Table 1, why are three of the relative errors shown in the first two data columns equal to 0.00e+00. These seems a bit odd.... did the code blow up?

> **Answer #3:** The *error was really zero*, both ways to do the integral gave the same answer. Notice that although the exponent is 0, so is the mantissa part.

Question #4: In Table 1 (for example), there is an Eq. (36) column which has lousy results for small n. I *assume* that these results come from using equation (36) with $f(x) = \ln(x)\, x^n$. That is to say, since you don't show ln(x) as part of the weight function in (36), it has presumably been moved into f(x). The other columns I presume just have $f(x) = x^n$ for their respective integrals. I guess I got a little confused about f(x) having these two different meanings, but I understand it just means "some function". Sort of like the A_L confusion I mentioned earlier (paper 2 above at the end).

> **Answer #4:** The stated assumption is correct for this table.

Comment: I suppose a reader might casually wonder why you don't just include ln(x) directly in the weight function as is. Yes it is singular at the endpoint 0, but so are other things in other weight functions, perhaps even more so since the log is soft. The reason is that ln(x) is not positive definite in the range $(0,\infty)$, and that screws up the ortho poly theory, invalidating the method.

Comment on the tables. Comments are really on the words in the caption of the tables.

(1) I guess your readers are more expert than me in this whole subject, but maybe they should still be reminded that the double-precision truncation error is 2^{-52} = 2.22e-16 [52 bits after the binary point] which is I guess is one ULP, and that therefore your results being ~ e-15 are PDG (pretty damn good). Having said that, I wonder how the 90 ULPs of error you have in worst case break down into the three categories you give on pages 12 and 13 -- and then how this error might be further reduced.

No answer.

PS. My impression is that if ε is your relative error, and x is your computed answer in the mantissa, then you would say that ULPsDIFF = $x - x_{exact} = (x - x_{exact})/ x_{exact} * x_{exact} = \varepsilon * x_{exact}$, and x_{exact} is in the range of .5 to 1, meaning .100000... to .111111.... This explains to me why you can have ε = 99e-16 but ULPs = 90. Is that right?

No answer.

PPS. I guess if you have rounding, you can say your representation accuracy can be 1.11e-16. More later on this. I guess you would call that ULPs = 0.5 or maybe 0. I notice this 1.1e-16 appearing in the second paper Table 2 on page 17 on the gamma function. Maybe you really call this ULPs = 0 since there is then really no error at all in the last place, so to speak.

(2) You frequently mention that other techniques require X thousand "function evaluations" to yield results that are in fact much less accurate than yours. I don't really have a feel for how SLOW these other routines are compared to yours. Is it dramatic? Are you 1000 times faster for the same accuracy?? All I see is a bunch of order N and order N^2 claims. Clarifying this might add some zap to your paper. I presume these many functional evaluations of the other methods arise from some kind of Newton-like iteration/adaptive process.

No answer.