Jim, October 20, 2004

I have read the eight papers you gave me, and they are all interesting. More or less, I derived every equation in every paper, except where the information to do so was not available without a trip to the library. I could of course only do all this since I am in a "lull" on contracting (thank goodness), and it was quite entertaining. *And*, I learned many new things.

Below are a few comments, questions and some suggestions regarding each paper.

If you feel like it, you might insert your responses in-line to my comments in red and send this file back to me,

OK by me to send some or all of this to Nelson Beebe if you want.

Best regards,
-Phil

"Let's see, how do you do calculus again.... "

---

1. Wong and Goto, on fast computation of elementary functions [1994]

I don't really know how things like exp(x) and sin(x) are computed on my PC. I presume is it a combination of software and hardware that someone has hopefully made efficient. The hardware I presume can only do +, -, * and / in FP. Their ATA idea is pretty simple but works only for single-precision and could easily be done in a Xilinx chip.

The basic Xilinx block RAMs these days are 16Kx1 (meaning 14 address lines and one bit stored), but they can be treated in many ways as this chart shows,

Table 17: **Dual-Port Mode Configurations**

Port A	16K x 1	16K x 1	16K x 1	16K x 1	16K x 1	16K x 1
Port B	16K x 1	8K x 2	4K x 4	2K x 9	1K x 18	512 x 36
Port A	8K x 2	8K x 2	8K x 2	8K x 2	8K x 2	
Port B	8K x 2	4K x 4	2K x 9	1K x 18	512 x 36	
Port A	4K x 4	4K x 4	4K x 4	4K x 4		
Port B	4K x 4	2K x 9	1K x 18	512 x 36		
Port A	2K x 9	2K x 9	2K x 9			
Port B	2K x 9	1K x 18	512 x 36			
Port A	1K x 18	1K x 18				
Port B	1K x 18	512 x 36				
Port A	512 x 36					
Port B	512 x 36					

A block RAM can be dual ported (simultaneous read and write at different addresses) with the port size combinations shown above. Often one has one port connected to software, and the other to hardware (but both can go to hardware and the RAM can be inited as a ROM). The (hardware) access time is on the

order of 2 nsec (read or write or both). This is always something to think about. The Wong & Goto application uses 4Kx30 ROMs (6 of them shown on the 4th page block diagram, most less than 30 bits in fact). This would require maybe 8 of the 16Kx1 RAM primitives for each of their tables. The largest Xilinx chips contain about 500 of these RAM primitives.

In addition to fast memories, with each of the above memory blocks Xilinx chips also have a fast 18 bit fixed-point multiplier (with 36 bit output) that can multiply in about 5 nsec. Their intention was DSP applications, but I could imagine these being used in iteration methods to do something interesting. Again, the largest Xilinx chips have about 500 of these multipliers just sitting there waiting to do something. In my designs at L3, they typically go completely unused!

---

2. Automatic Computation of Zeros of Bessel Functions, etc (Jim Ball Aug 2000 SIAM)

This taught me the basic Jacobi Matrix method for finding the zeros of an orthogonal polynomial, something I had not seen before. In this paper, vectors were N+1 in length and matrices were N+1 x N+1. The notation was self-consistent with respect to indices on the recursion coefficients. The idea is to use the EISPACK TQL2 routine to diagonalize the Jacobi matrix to generate the $g(x_i)$ eigenvalues, from which the x_i nodes can be found if g(x) is invertible, which it certainly is in all your cases. The code certainly is concise, and I did not know about the constant .d0 notion in Fortran. I did not go learn details of the QL method for actually handling the tridiagonal matrix, no need.

---

3. Recurrence Relations for Ortho Polys and Efficient Algorithms (Jim Ball, I guess unpublished)

Here I ran into immediate trouble because the recursion formula has the same index convention as paper 2 above, but the indices on the upper left corner of the displayed Jacobi matrix are labeled wrong, subscripts all off by 1 (but correct in the bottom right corner). This did not impact the rest of the paper it turned out. [It is an interesting matrix that has N+1 elements on the diagonal, but N-1 on the adjacent off-diagonal.]

The discussion on pages 3,4 and 5 seemed just a bit blurry to me, but I was able to clarify it to myself by using more quantum mechanical type notations. The **X** operator has eigenstates and eigenvalues:

$$\mathbf{X}\,|x\rangle = x\,|x\rangle$$

Glue <n| onto this and use $\mathbf{1} = \Sigma_{\mathbf{m}}|m\rangle\langle m|$ completeness (in poly Hilbert space) to say

$$\Sigma_{\mathbf{m}} \langle n|\mathbf{X}|m\rangle\langle m|x\rangle = x\,\langle n|x\rangle \qquad \text{or} \qquad \Sigma_{\mathbf{m}}\, X_{nm}\phi_m(x) = x\ \phi_n(x)$$

Set $x = x_i$, then X_{nm} must be the Jacobi matrix since it is the matrix whose eigenvalues are x_i and whose eigenvectors are the $\phi_n(x_i)$. Also things like

$$\mathbf{1} = \int dx\, w(x)\, |x\rangle\langle x| \qquad \text{// completeness in continuous x space}$$

$$\langle m|n\rangle = \delta_{m,n} = \int dx\, w(x) \langle m|x\rangle\langle x|n\rangle = \int dx\, w(x)\phi_m(x)\phi_n(x)$$

Then there is a different *discrete* set of basis states $|\, x_i\rangle$ (x_i = the nodes) such that

$$\langle x_i \mid x_i\rangle = \Sigma_n \langle x_i\, |n\rangle\langle n|\, x_i\rangle = \Sigma_n\, \phi_n(x_i)^2 \;\equiv\; S_i$$

But then normalize these states to get

$$|i\rangle \equiv (1/\sqrt{S_i}\,)|\, x_i\rangle \qquad \Rightarrow \quad \langle i|i\rangle = 1$$

$$\mathbf{1} = \Sigma_i\, |i\rangle\langle i| \;=\; \Sigma_i\, (1/S_i)\; |x_i\rangle\langle x_i| \qquad\qquad \text{// completeness in i space}$$

where this last shows the analogy between discrete weight $W_i = (1/S_i)$ and continuous weight $w(x)$. And so on and so forth. I guess at least referring to A_{nm} as X_{nm} would have been better for me in (3.1). Connections to your notations are

$$\langle i|n\rangle = v_n^{\,i} = (1/\sqrt{S_i}\,)\langle x_i|n\rangle = (1/\sqrt{S_i}\,)\phi_n(x_i) = (1/\sqrt{S_i}\,)V_n^{\,i}$$

Here was my personal derivation of the important Gaussian Quadrature formula (3.7):

Let our 2N polynomial be written as you suggest, sums to N

$$f(x) = \mathbf{\Sigma}_{m,n}\, F_{mn}\, \phi_n(x)\phi_m(x)$$

If you insert this into the **RHS** of (3.7), you get

$$\Sigma_i\; (1/S_i)\;\; \mathbf{\Sigma}_{m,n}\, F_{mn}\, \phi_n(x_i)\phi_m(x_i) = \mathbf{\Sigma}_{m,n}\, F_{mn}\;\; \Sigma_i\; (1/S_i)\; \phi_n(x_i)\phi_m(x_i)$$

$$= \mathbf{\Sigma}_{m,n}\, F_{mn}\;\; \Sigma_i\; (1/S_i)\, \langle n|x_i\rangle\langle x_i|m\rangle \;=\; \mathbf{\Sigma}_{m,n}\, F_{mn}\;\; \langle n|m\rangle = \mathrm{tr}(F)\ .$$

If you insert this into the **LHS** of (3.7), you get

$$\int dx\, w(x)\, \mathbf{\Sigma}_{m,n}\, F_{mn}\, \phi_n(x)\phi_m(x) \;= \mathbf{\Sigma}_{m,n}\, F_{mn}\, \Big[\, \int dx\, w(x)\;\; \phi_n(x)\phi_m(x)\, \Big]$$

$$= \mathbf{\Sigma}_{m,n}\, F_{mn}\, \delta_{m,n} = \mathrm{tr}(F)\ .$$

So the GQ formula is exact for polys of degree 2N.

In Section 4 you do a solution inside a sphere. My only complaint here is that the A_L of (4.2) is not the same as the A_L of (4.3), the difference being a normalization factor. Also, a certain Figure 1 mentioned on page 7 is missing from my copy of your paper.

This paper just seemed a bit odd to me. The main new thing I learned compared to the previous paper was the fact that the recursion-relation / Jacobi Matrix approach gives you both the eigenvalues x_i (zeros) *and* gives you the Gaussian Quadrature weights $W_i = 1/S_i$ with S_i as shown above. You compute the inverse weight S_i by adding up the magnitudes of the eigenvectors that the TQL2 routine can return. In the previous paper we only worried about the eigenvalues.

---

4. Contact Plate slowly converging series Gautschi 1991

I was impressed by the result of this paper, although I don't know how these series arise in structural mechanics where (I guess) a plate presses against some object. I can see that, if p = 2, and z near 1, you would have to add up about 5 billion terms of the series to get e-20 accuracy, whereas his "trick" does it no problem. He transforms the sum in question into an integral with a certain interval and weight function w(t), which of course then has some weird ortho polys associated with it. These polys have a recursion formula. If you knew the α and β coefficients, you could define an iteration whose limit is the integral you want. Unfortunately, *why* this is so is not explained in this paper, but in another. Similarly, he quotes the method for finding the α and β, but that too is in some other paper. At least he puts the values in an appendix. A good feature of the paper is that there are lots of experimental results showing how well the method actually works with closed-form test sums. I was happy to see similar experimental results appearing in later Jim papers. I presume this guy Gautschi is a world guru on ortho polys, maybe the Szego of his era.

---

5. Gaussian Quadrature for Two Classes of Log functions. Jim, unpublished

Unfortunately, I read this before I realized that the next paper on my list was a cleanup of this one. As I read it, I marked up quite a few errors. First, we had the same confusion with the indices of the recursion coefficients in the displayed Jacobi matrix, which caused trouble down the road in this paper because the Laguerre coefficients quoted were then one off in index [as in (3.3)], and so on. And suddenly things are N x N instead of N+1 x N+1.

I was happy of course to see that all the errors I found were fixed in the next paper. As I ground my way through this paper, I did read some very interesting stuff I never knew in my second volume of Erdelyi, more comments on this below. [9. and 10.]

---

6. Ball & Beebe Paper #1: The Theory behind doing the log integrals by GQ

This paper was a significant improvement over the draft paper #5, and here are my reasons for saying that:

- the recursion relation index labeling is correctly stated and consistent with the matrix
- all the other errors I found before have been fixed
- explicit references are given for all obscure results, such as A&S integrals
- some references use dates like [1884] which I find very interesting in this kind of paper

- the idea that there are two methods is made clearer (really 3 methods, 1 not so good)
- some experimental test results have been added.
- there is more comment on the work of other authors (I suspect referee induced)
- probably some reviewer suggested fewer numbered equations ??

On this paper I have a few questions for Jim.

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.

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.

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?

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).

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.

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 $\text{ULPsDIFF} = x - x_{exact} = (x - x_{exact})/ x_{exact} * x_{exact} = \varepsilon * x_{exact}$, 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?

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.

---

7. Beebe & Ball, Paper #2: The code package

It sounds like this is the Beebe paper and the other is the Ball paper. This paper certainly is highly detailed. I was unaware that Fortran to C converters exist and actually work. I am amazed to hear that you have to keep routine names to 6 letters to be portable in Fortran, what a ball and chain that is.

For this paper, I have some comments:

(1) Well, I guess I have to say this. As an "everyman" ordinary Citizen, I (like almost every other person who would possibly use the quadlog package) have a Windows PC in front of me, with a whopping fast CPU, in my case Athlon. It would sure be nice to see an entry in Table 1 on page 4 in the Intel x86 section which says something like " Windows XP/2000/NT with GNU/Cygwin/BASH shell and some kind of compiler", etc. In any event, it would be nice to make some kind of Windows comment at least. I could imagine readers maybe downloading the package and wanting to try it out "at home", people such as Jim Ball or me -- without having to create a Linux partition and all that bother. Also, I suspect that the days of R6000 UNIX computers on professor desks may be numbered. But maybe I am wrong on this. At L3, only the "old guys" have both UNIX and PC in their cubes. If nothing else, it is an interesting topic.

(2) I had to look very hard to find a location where I could actually download the package. I found it buried in a Beebe reference on page 26 as http://www.math.utah.edu/~beebe/software/mdiff. But then that web page does not exist. It seems that something like this would be "put up" on some stable web server that will last many years, not on someone's personal page. (People do move, etc.) And I think I would put the link somewhere at the start of the paper. Just comments...

(3) I don't *think* anyone would ever want to do quadlog in Java.

(4) I am not familiar with FP hardware on most computers, but I am a little familiar with FP in the PC world. The FP unit does 80-bit FP math and has several 80-bit internal registers, but of course bulk RAM registers are 64-bits. So any time you add up a sum of lots of terms (as you indeed must do in quadlog), you want to accumulate that sum in a FP register and only take out the 64-bit answer at the end. I don't know how you tell a compiler to do things this way, but maybe it happens by default. I suspect the dvsum() routine mentioned on page 8 is a software method for minimizing sum errors. It is probably a bit

slow, and would not be necessary on 95% of the world's computers. Probably this 80-bit stuff is just as good as 128-bit software quad precision if you are just trying to get a properly rounded 64-bit sum of a few hundred terms. { The 80-bits is mentioned in passing on page 16 } { and the shorter memory word issue is mentioned on page 19 }

(5) By the way, I feel that Fortran statements like A = B/C/D are not good practice because they add uncertainty in the mind of a multi-lingual code reader who may not be doing Fortran for a living. I am sure it is always evaluated as A = B/(C*D). Why make something unclear when it is trivial to make it clear? I guess I am showing my lack of Fortran experience here. [You do this in your "automatic computation of zeros" code in paper 2.]

(6) I like the ndiff program idea as a "soft diff". [I also like the freeware csdiff]

(7) I agree that J(α,β) should be B(α,β) on page 24. J certainly makes you think Bessel. Instead of saying "we start with the analytic result", you might say " we start with this well-known integral representation of the Beta function", or something like that. Maybe give a more standard reference, like A&S Dover page 258 6.2.1 and 6.2.2, since people may not happen to own the 1978 version of CRC, etc etc etc. (whine whine whine)

(8) On page 19 at the start of section 9, the words might be changed to: "the smallest number that can be added to one such that the sum differs from one".

(9) I was a bit surprised to hear that you were able to run off the end of the dynamic range of double-precision numbers which is 10^{308} with N ~ 100 (page 25)

All in all, these two Ball and Beebe papers seem quite solid and well-written to me. I am not sure why you are having publication problems. Some journals might feel that #2 is "wordy" with too much detail, but on-line publishing (if you were to use it) does not care about length, and the reader benefits from having the extra information available. I hate to see info eliminated when hard work was needed to develop that info and some readers might find it very useful.

---

8. Half-range Generalized Hermite polys, use in GQ

As I derived the various equations shown, I again found the QM or scalar product notation useful and compact. For example,

"The recursion relation implies

$$(\phi_r, \phi_{n+1}) = (x\phi_r, \phi_n) - \alpha_n (\phi_r, \phi_n) - \beta_n (\phi_r, \phi_{n-1}) \qquad \text{where } (\phi_r, \phi_r) = T_r$$

Setting r = n gives

$$0 = (x\phi_n, \phi_n) - \alpha_n T_n = S_n - \alpha_n T_n \quad \Rightarrow \quad \alpha_n = S_n / T_n \qquad (2.5)$$

where $S_n \equiv (x\phi_n, \phi_n)$, and setting r = n-1 gives

$$0 = (x\phi_{n-1}, \phi_n) - \beta_n T_{n-1} = T_n - \beta_n T_{n-1} \quad \Rightarrow \quad \beta_n = T_n / T_{n-1} \qquad (2.4)$$

In the last step above, since the ϕ_n are monic, we know $\phi_{n-1} = x^{n-1} +$ rest. Thus, $x\phi_{n-1} = x^n$ + rest'. Thus, we can write $x\phi_{n-1} = \phi_n$ + lincom of other ϕ, so $(x\phi_{n-1}, \phi_n) = T_n$ by orthogonality. "

I spent a lot of minutes deriving these little equations when I could have been watching Red Sox! I guess it is always an issue of how much you want to help the reader follow your path.

I presume I have to go read Shizgal to understand results (2.9) and (2.11). Is that correct, or did I miss something here? I would think there is probably some simple derivation.

In any event, I understand the 14x/stage error-magnification problem encountered in the iteration, and how you got around it by two methods:

(1) pick good starting values for the g_n . Think of them like beads on a string, and run your little recursion-relation tool over that string maybe 10 times to get the beads positioned correctly. The endpoints have to stay fixed, so the g_N error does not go away.

(2) Adjust the entire string of beads at once, globally, using the matrix iteration (3.16). This converges much faster because ALL information is used at once rather than in piecemeal fashion. Also, I presume this lets g_N be adjusted along with the other beads. [By the way, I think this notion of global use of information is why the Jacobi matrix method for GQ works so well numerically.]

I guess that Steen published a table in which most of the digits are wrong! (your page 9).

I have a comment/question about computing the α and β here, but it is embedded in the next sections. I guess it seems that these things could be gotten in closed form and just evaluated with some gamma functions.

---

9. Comment on F.B. Hildebrand "Intro to Numerical Analysis" Dover 2nd Ed 1974

Things learned from this book:

(1) It is known [Szego 1967] than an interval (a,b) and a non-negative weight function w(x) define a unique set of orthonormal polys provided $\int_a^b dx\ w(x)\ x^k$ is finite for all k, even if a or b or both are infinity. (p 326)

(2) If the weight function is allowed to have negative regions, then the norm $||\phi_n||^2 = (\phi_n, \phi_n)$ can be negative, and then you are SOL (I guess no longer a Hilbert space or something like that).

(3) You can write a very simple looking ODE with trivial boundary conditions for U_r where $D^r U_r = w(x)\phi_r(x)$. This ODE is Rodrigues-like, but is not Rodrigues. Solve for U_r, then you know the ϕ_r. (p 327) This is how (1) above is known to be true. (The ODE is this: $D^{r+1} [(1/w) D^r U_r] = 0$.)

(4) If you try to fit a function f(x) with a linear combination of orthogonal polynomials $p(x) = \Sigma\, A_r \phi_r(x)$ corresponding to some w(x) and (a,b), then the coefficients A_r which minimize the weighted integral over (a,b) of $[\, f(x) - p(x)\,]^2$ are precisely the weights obtained in Gaussian Quadrature. So the GQ weights are those which give a least squares fit to f(x) on (a,b). Of course the fit is exact if f(x) is a poly is degree N. But as shown earlier, GQ is also exact if f(x) is order 2N (or maybe it is 2N-1, I forget).

---

10. Comments on Erdelyi Vol 2 "Special Functions" (Bateman project) , Chapter 12 on Ortho Polys

Things learned from this book:

(1) Szego 1939 was the "standard textbook" on this subject in 1953 when the Bateman was done.

(2) Let $c_n \equiv \int_a^b dx\; w(x)\; x^n$. These are the things that must be finite, as noted above. Then apart from normalization, the exact explicit orthogonal polynomials are given by expression (4) on page 158. This is a determinant of a matrix whose elements are all c_i but whose last row is 1, x, x^2 ... x^n. Thus, if the c_n integrals can be done in closed form, you have a closed form for the orthog polynomials.

Example: Half-Range Generalized Hermites,

$c_n = \int_a^b dx\; w(x)\; x^n = \int_0^\infty dx\; x^{\gamma+n} \exp(-x^2) = \Gamma\,[\, (n+\gamma+1)/2\,] / 2$ // says Maple

I would think one could then write a closed form for the $\phi_n(x)$ using Erdelyi's determinant with $\varepsilon_{ijk..}$ cofactor notation, etc etc. In particular, one could write down the highest order two coefficients which Erdelyi likes to call k_n and k'_n. The normalization factors $h_n = (\phi_n, \phi_n)$ can also be written in terms of the c_n as in (6) p 158.

(3) The recursion relation is stated in (7) page 158 as

$$p_{n+1}(x) = (A_n x + B_n)\, p_n(x) - C_n\, p_{n-1}(x)$$

Then in a few lines on the top of page 159 he *derives* this recursion relation, so I finally know why it is that the factor g(x) = x appears in the thing. [There is a sign typo, it should say $\gamma_1 = -\, C_n$ in mid page, and this is not in the list of errata at the start of my book.] The coefficients are derived to be

$A_n = k_{n+1}/k_n$

$B_n = A_n\, [\; k_{n+1}' / k_{n+1} \; - \; k_n' / k_n\,]$

$C_n = (\, A_n/A_{n-1}\,) * (\, h_n/h_{n-1}\,)$ // $= (\, A_n/A_{n-1}\,)$ for normalized polys

Suppose we rewrite the recursion formula in "Ball standard form" [as in B&B#1 (7)]

$$p_{n+1}(x) = (A_n x + B_n)\, p_n(x) - C_n\, p_{n-1}(x) \Rightarrow$$

$$A_n\, x\, p_n(x) = p_{n+1}(x) - B_n\, p_n(x) + C_n\, p_{n-1}(x)$$

$$x\, p_n(x) = (1/A_n) p_{n+1}(x) - (B_n/A_n)\, p_n(x) + (C_n/A_n)\, p_{n-1}(x)$$

$$\equiv a_{n+1}\, p_{n+1}(x) + b_n\, p_n(x) + c_n\, p_{n-1}(x)$$

Then we get $a_n = (1/A_{n-1})$ so

$$a_n = k_{n-1}/k_n$$

$$b_n = [\, k_n' / k_n - k_{n+1}' / k_{n+1} \,]$$

$$c_n = (a_n / a_{n+1}) * (h_n/h_{n-1}) = (k_{n-1}\, k_{n+1} / k_n^2) * (h_n/h_{n-1})$$

So I guess my point is that you ought to be able to write down the recursion coefficients directly without any need for iterative formulas and such, but I guess it might be a little messy with the various determinant forms. Still, it seems that a fully closed form result for the recursion coefficients would be possible, such as for those half-range Hermites. (But maybe such a closed form is not good for numerical computation...)

(4) If in addition to the weight w(x) and the interval (a,b), you add *any one* of the three conditions shown here (p 164), then the orthonormal polynomials are called "classical" ortho polys. Those conditions are:

(i) the $\phi_n'(x)$ derivatives also form a set of ortho polys

(ii) the $\phi_n(x)$ satisfy an ODE of the form: $A(x)\, y'' + B(x)\, y' + \lambda_n\, y = 0$

(iii) the $\phi_n(x)$ can be written as: $\phi_n(x) = \text{constant} * (1/w)\, D^n [\, w\, X^n \,]$
where X is a poly of degree 0, 1 or 2 only. This is Rodrigues' Formula.

And any *one* of these three conditions implies the other two. So this cleared up the mystery of why Erdelyi was not writing a generalized Rodriguez' formula in his general discussion of ortho polys based on w(x) and (a,b). And it shows why we always have a Rodrigues for our familiar functions -- they all satisfy an ODE which is why they came to our attention in the first place.

(5) The Christoffel-Darboux formulas are shown on page 159, and Erdelyi says they are "easily derived" from the recursion relation, but I had to look at Hildebrand page 342 to see how this works. I have learned from you that when n=N (or N+1) in the sum and $x = x_i$, one of the terms goes away since x_i are nodes, and you get a simple closed form result for the GQ weight.

---

11. Comments on Word. One final comment on a different subject. My Word normal.dot file knows most of the Greek letters using the ALT key, and I have two kinds of key-controlled sub and superscripts. The small ones are hard to read, but don't affect my line spacing, as in $T^{\mu\upsilon} = x_i + \gamma^2$. The large ones are easier to read, but do affect line spacing, as in $T^{\mu\upsilon} = x_i + \gamma^2$. These are all done with fast keystrokes. So all this makes it pretty fast to type equations in line or separate, as I have done in this document above. For integrals and sums and radicals and things, I have pre-built objects made from the EQ field thing. I then open them with my F4 key to make changes (F4 is also in my normal.dot, to avoid a right click to Toggle Field Code). It is all very fast. (I hate using the "equation editor", but I do it for matrices). See end of this document.

In Word, to add new keystrokes, there are two things you can do:

(1) for something like a Greek letter, do Insert/Symbol and set a shortcut key with the button shown.
I use things like ALT g for γ, etc. (and of course SHIFT ALT g for Γ).

(2) for menu items, do Tools/Customize then hit the keyboard button and then you can set keystrokes as you like. I think all Word commands are available here.

An easier way is to just get hold of my normal.dot file.

I have a link to the following file (cut and pasted in below) in my Word Work menu so I can always get to it quickly. Here is the file. (The integral signs are little sleazy, but I have learned to like them.)

****

Collection of Symbols for Cut and Paste in E&M PhL 1.21.99

1. Integral sign: Σ $\int$ $\int_a^b$ $\int_a^b$ $\sum_{n=0}^{3}$ $\prod_{n=0}^{3}$ $\sum_{n=0}^{3}$

$\sum_{n=0}^{3}$ $\int f(x)\,dx$ $\int a$ $\int_{-\infty}^{\infty} f(x)dx$ $\Big|_b^a$

2. Unit vectors: $\hat{\mathbf{r}}$ $\hat{\mathbf{r}}_a$ $\hat{\boldsymbol{\varepsilon}}_t$ $\hat{\boldsymbol{\varepsilon}}_1$
$\hat{\boldsymbol{\varepsilon}}_2$ $\hat{\boldsymbol{\phi}}$ $\hat{\mathbf{n}}$ $\hat{\boldsymbol{\theta}}$ $\hat{x}$

3. Fractions: $\frac{a}{b}$ $\frac{\partial a}{\partial b}$ $\frac{e^{i\mathbf{k}\bullet\mathbf{r}}}{r}$ $\frac{e^{ikR}}{R}$ $\frac{\exp(ikR)}{R}$ $\sqrt{\frac{a}{b}}$ $\begin{Bmatrix} a \\ b \\ c \end{Bmatrix}$

4. exponentials: $e^{i\mathbf{k}\bullet\mathbf{r}}$ e^{ikR} $e^{i(\mathbf{k}\bullet\mathbf{r} - \omega t)}$ $e^{i\omega t}$

4a. better versions: $e^{i\omega t}$ // here you can set the superscript text size as you like

5. Square roots: $\sqrt{a}$ $\sqrt{2}$ $\sqrt{\mu\varepsilon}$ $\begin{matrix} a \\ b \end{matrix}$ $\left(\begin{matrix} a \\ b \end{matrix}\right)$ $\left(\begin{matrix} a \\ b \end{matrix}\right)$

$\begin{cases} \text{hello} \\ \text{bob} \end{cases}$

6. Time derivatives: $\dot{\mathbf{r}}$ $\ddot{\mathbf{r}}$ $\dot{x}$ $\ddot{r}$ $\dot{\mathbf{A}}$ $\ddot{A}$
(caps have higher dots)

7. Other: □ $\vec{\mathbf{H}}$ $\overleftrightarrow{\mathbf{T}}$ $\vec{\mathbf{h}}$ $\mathcal{P}$ $\mp$ $\hbar$ $\lambda\!\!\!^{-}$ $\bar{x}$

8. Using the ADVANCE control. To see all the advance fields below, select the whole object and hit the F4 key. Sometimes otherwise you don't get to see ALL the advance fields. You have to adjust the left-shift of the lower characters depending on how many you want to put there. Don't forget the final adjust on exit to get the line back to the right vertical level.

$h_{\ell}^{(1)}(kr)$

This is an in-line $\mathbf{M}_{e\ell n}^{(1)}(x,y,z)$ and here we continue the text.

$\mathbf{M}_{o1x}^{(1)}$ mmmmmmXXX

General Notes

The small help you get from the EQ field insertion does not tell the whole story, you have to look up Help on EQ and you will get much more detail. For example, the \I field not only makes integral signs, but summation and anything else you might want. Here is the help on \I:

\I field

Integral: \i(,,)

Creates an integral, using the specified symbol or default symbol and three elements. The first element is the lower limit, the second is the upper limit, and the third is the integrand. You can use the options listed in the following table to modify the \i switch.

Option Description

\su Changes the symbol to a capital **sigma** and creates a summation.

\pr Changes the symbol to a capital **pi** and creates a product.

\in Creates the **inline format** with the limits displayed to the right of the symbol instead of above and below it.

\fc\c Substitutes a fixed-height character specified by c for the symbol.

\vc\c Substitutes a variable-height character specified by c for the symbol. The symbol matches the height of the third element.

Example { EQ \i \su(n=0,3,)} displays $\sum_{n=0}^{3}$ $\sum_{n=0}^{3} k$ $\prod_{n=0}^{3}$ $\sum_{n=0}^{3}$

In this example, you have to at least have a space in the third argument to get it to work.

Subscripts and Superscripts

Superscript or Subscript: \s()

Places an element or elements as superscript or subscript characters. Each \s code can have one or more elements; separate the elements with commas. If more than one element is specified, the elements are stacked and left-aligned. You can use the options listed in the following table to place single elements after the \s switch.

Option Does this

\ain () Adds space above a line in a paragraph by the number of points specified by n.

\upn () Moves a single element above the adjacent text by the number of points specified by n. The default is 2 points.

\din () Adds space below a line in a paragraph by the number of points specified by n.

\don () Moves a single element below the adjacent text by the number of points specified by n. The default is 2 points.

Example:

{ EQ \s\up8(UB)\s\do8(2) } displays: ssss $^{UB}_{2}$