NUMERICAL RECIPES

Orthogonal Polynomials And Gaussian Quadrature With Nonclassical Weight Functions

William H. Press and Saul A. Teukolsky

The subject of orthogonal polynomials crops up in a variety of different applications in both analytic and numerical work. Every different weight function (defined below) implies a different set of orthogonal polynomials. For the few most common, "classical" weight functions, the resulting polynomials are tabulated in standard references. In this column we will show you how to construct your own orthogonal polynomials when the weight function in your problem is not one of the classical forms. As an application, we will show how you can use a set of such orthogonal polynomials to carry out an otherwise intractable Gaussian quadrature.

To fix our notation, let us consider an interval (a,b) on the real axis. We define the "scalar product of two functions f and g over a weight function W" as

$$\langle f|g\rangle \equiv \int_a^b W(x)f(x)g(x)dx. \tag{1}$$

The scalar product is a number, not a function of x. Two functions are said to be *orthogonal* if their scalar product is zero. A function is said to be *normalized* if its scalar product with itself is unity. A set of functions that are all mutually orthogonal and also all individually normalized is called an *orthonormal* set.

We can find a set of polynomials (i) that includes exactly one polynomial of order j, called $p_j(x)$, for each $j=0,1,2,\ldots$, and (ii) all of which are mutually orthogonal over the specified weight function $W(x)$. A constructive procedure for finding such a set is the recurrence relation

$$\begin{aligned} &p_{-1}(x)\equiv 0,\\ &p_0(x)\equiv 1,\\ &p_{j+1}(x) = (x-a_j)p_j(x) - b_j p_{j-1}(x),\\ &\qquad\qquad j=0,1,2,\ldots, \end{aligned} \tag{2}$$

where

$$\begin{aligned} a_j &= \langle xp_j|p_j\rangle/\langle p_j|p_j\rangle, \quad j=0,1,2,\ldots,\\ b_j &= \langle p_j|p_j\rangle/\langle p_{j-1}|p_{j-1}\rangle, \quad j=1,2,\ldots. \end{aligned} \tag{3}$$

The coefficient b_0 is arbitrary; we can take it to be zero.

The polynomials defined by (2) are *monic*, i.e., the coefficient of their leading term [x^j for $p_j(x)$] is unity. If we divide each $p_j(x)$ by the constant $[\langle p_j|p_j\rangle]^{1/2}$ we can render the set of polynomials orthonormal. One also encounters orthogonal polynomials with various other normalizations. You can convert from a given normalization to monic polynomials if you know that the coefficient of x^j in p_j is λ_j, say; then the monic polynomials are obtained by dividing each p_j by λ_j. Note that the coefficients in a recurrence relation like (2) depend on the adopted normalization.

For the "classical" orthogonal polynomials, the coefficients a_j and b_j have been worked out long ago, with formulas for them given in standard references. In such cases, the recurrence relation (2) provides a stable means of generating the polynomials. The classical cases include the Legendre polynomials [$W(x)=1$], Chebyshev polynomials [$W(x)=(1-x^2)^{-1/2}$], Jacobi polynomials [$W(x)=(1-x)^\alpha(1+x)^\beta$], which are all defined on the interval $(-1,1)$. On the interval $(0,\infty)$ there are the Laguerre polynomials [$W(x)=x^\alpha e^{-x}$], while on $(-\infty,\infty)$ there are the Hermite polynomials [$W(x)=e^{-x^2}$].

By contrast, if you are confronted with a nonclassical weight function $W(x)$, the construction of the associated set of orthogonal polynomials is not trivial. The *procedure of Stieltjes*, outlined above, is to compute a_0 from (3), then $p_1(x)$ from (2). Knowing p_0 and p_1, we can compute a_1 and b_1 from (3), and so on. But how are we to compute the inner products in (3)?

The textbook approach is to represent each $p_j(x)$ explicitly as a polynomial in x and to compute the inner products by multiplying out term by term. This will be feasible if we know the first $2N$ moments of the weight function,

$$\mu_j = \int_a^b x^j W(x)dx, \quad j=0,1,\ldots,2N-1. \tag{4}$$

However, the solution of the resulting set of algebraic equations for the coefficients a_j and b_j in terms of the moments μ_j is in general *extremely* ill-conditioned. Even in double precision, it is not unusual to lose all accuracy by the time $N=12$. We must thus reject any procedure based on the moments (4).

Sack and Donovan[1] discovered that the numerical stability is greatly improved if, instead of using powers of x as a set of basis functions to represent the p_j's, one uses some other known set of orthogonal polynomials $\pi_j(x)$, say. Roughly speaking, the improved stability occurs

William H. Press is a professor of astronomy and physics at Harvard University. Saul A. Teukolsky is a professor of physics and astronomy at Cornell University.

because the polynomial basis "samples" the interval (a,b) better than the power basis when the inner product integrals are evaluated, especially if its weight function resembles $W(x)$.

So assume that we know the *modified moments*

$$\nu_j = \int_a^b \pi_j(x) W(x) dx, \quad j = 0,1,...,2N-1, \tag{5}$$

where the π_j's satisfy a recurrence relation analogous to (2),

$$\begin{aligned} &\pi_{-1}(x) \equiv 0, \\ &\pi_0(x) \equiv 1, \\ &\pi_{j+1}(x) = (x - \alpha_j)\pi_j(x) - \beta_j \pi_{j-1}(x), \\ &\qquad\qquad j = 0,1,2,..., \end{aligned} \tag{6}$$

and the coefficients α_j, β_j are known explicitly. Then Wheeler[2] has given an efficient $O(N^2)$ algorithm equivalent to that of Sack and Donovan for finding a_j and b_j via intermediate quantities

$$\sigma_{k,l} = \langle p_k | \pi_l \rangle, \quad k,l \geqslant -1. \tag{7}$$

Initialize

$$\begin{aligned} &\sigma_{-1,l} = 0, \quad l = 1,2,...,2N-2, \\ &\sigma_{0,l} = \nu_l, \quad l = 0,1,...,2N-1, \\ &a_0 = \alpha_0 + \nu_1/\nu_0, \\ &b_0 = 0. \end{aligned} \tag{8}$$

Then, for $k = 1,2,...,N-1$, compute

$$\begin{aligned} \sigma_{k,l} &= \sigma_{k-1,l-1} - (a_{k-1} - \alpha_l)\sigma_{k-1,l} - b_{k-1}\sigma_{k-2,l} \\ &\quad + \beta_l \sigma_{k-1,l-1}, \\ &\qquad\qquad l = k,k+1,...,2N-k-1, \\ a_k &= \alpha_k - \sigma_{k-1,k}/\sigma_{k-1,k-1} + \sigma_{k,k+1}/\sigma_{k,k}, \\ b_k &= \sigma_{k,k}/\sigma_{k-1,k-1}. \end{aligned} \tag{9}$$

Note that the normalization factors can also easily be computed if needed:

$$\begin{aligned} &\langle p_0 | p_0 \rangle = \nu_0, \\ &\langle p_j | p_j \rangle = b_j \langle p_{j-1} | p_{j-1} \rangle, \quad j = 1,2,... . \end{aligned} \tag{10}$$

You can find a derivation of the above algorithm in Ref. 3.

Wheeler's algorithm requires that the modified moments (5) be accurately computed. In practical cases there is often a closed form, or else recurrence relations can be used. The algorithm is extremely successful for *finite* intervals (a,b). For infinite intervals, the algorithm does not completely remove the ill-conditioning. In this case, Gautschi[4] recommends reducing the interval to a finite interval by a change of variable, and then using a suitable discretization procedure to compute the inner products. You will have to consult the reference for details.

Box 1.

```
SUBROUTINE orthog(n,anu,alpha,beta,a,b)
INTEGER n,NMAX
REAL a(n),alpha(2*n-1),anu(2*n),b(n),beta(2*n-1)
PARAMETER(NMAX=64)
```

Computes the coefficients a_j and b_j, $j = 0,\ldots N-1$, of the recurrence relation for monic orthogonal polynomials with weight function $W(x)$ by Wheeler's algorithm. On input, `alpha(1:2*n-1)` and `beta(1:2*n-1)` are the coefficients α_j and β_j, $j = 0,\ldots 2N-2$, of the recurrence relation for the chosen basis of orthogonal polynomials. The modified moments ν_j are input in `anu(1:2*n)`. The first `n` coefficients are returned in `a(1:n)` and `b(1:n)`.

```
INTEGER k,l
REAL sig(2*NMAX+1,2*NMAX+1)
do 11 l=3,2*n                         Initialization, Equation (8).
    sig(1,l)=0.
enddo 11
do 12 l=2,2*n+1
    sig(2,l)=anu(l-1)
enddo 12
a(1)=alpha(1)+anu(2)/anu(1)
b(1)=0.
do 14 k=3,n+1                         Equation (9).
    do 13 l=k,2*n-k+3
        sig(k,l)=sig(k-1,l+1)+(alpha(l-1)-a(k-2))*sig(k-1,l)-
*           b(k-2)*sig(k-2,l)+beta(l-1)*sig(k-1,l-1)
    enddo 13
    a(k-1)=alpha(k-1)+sig(k,k+1)/sig(k,k)-sig(k-1,k)/sig(k-1,k-1)
    b(k-1)=sig(k,k)/sig(k-1,k-1)
enddo 14
return
END
```

Box 1 lists a routine, orthog, for generating the coefficients a_j and b_j by Wheeler's algorithm, given the coefficients α_j and β_j, and the modified moments ν_j. To conform to the usual FORTRAN convention for dimensioning subscripts, the indices of the σ matrix are increased by 2, i.e., sig(k,l) $= \sigma_{k-1,l-1}$, while the indices of the vectors α, β, a, and b are increased by 1.

As an example of the use of orthog, consider the problem[3] of generating orthogonal polynomials with the weight function $W(x) = -\log x$ on the interval $(0,1)$. A suitable set of π_j's is the shifted Legendre polynomials

$$\pi_j = [(j!)^2/(2j)!] P_j(2x-1). \tag{11}$$

The factor in front of P_j makes the polynomials monic. The coefficients in the recurrence relation (6) are

$$\begin{aligned} &\alpha_j = \tfrac{1}{2}, &&j = 0,1,..., \\ &\beta_j = 1/[4(4 - j^{-2})], &&j = 1,2,..., \end{aligned} \tag{12}$$

while the modified moments (calculated analytically) are

$$\nu_j = \begin{cases} 1, & j = 0, \\ (-1)^j (j!)^2/[j(j+1)(2j)!], & j \geqslant 1. \end{cases} \tag{13}$$

A call to orthog with this input allows one to generate the required polynomials to machine accuracy for very large N. Before Sack and Donovan's observation, this seemingly simple problem was essentially intractable.

Let us turn now from the construction of orthogonal polynomials to one of their principal applications, namely, the construction of Gaussian quadrature formulas. The simplest quadrature formulas (trapezoidal rule, Simpson's rule, and their higher-order Newton–Cotes generalizations) approximate the integral of a function by the sum of its functional values at a set of equally spaced points, multiplied by certain aptly chosen weighting coefficients. The idea of Gaussian quadratures is to give ourselves the freedom to choose not only the weighting coefficients, but also the location of the abscissas at which the function is to be evaluated: They will no longer be equally spaced. Thus we will have *twice* the number of degrees of freedom at our disposal; it turns out that we can achieve Gaussian quadrature formulas whose order is, essentially, twice that of the Newton–Cotes formula with the same number of function evaluations.

Does this sound too good to be true? Well, in a sense it is. The catch is a familiar one, which cannot be overemphasized: High order is not the same as high accuracy. High order translates to high accuracy only when the integrand is very smooth, in the sense of being "well approximated by a polynomial."

There is, however, one additional feature of Gaussian quadrature formulas that adds to their usefulness: We can arrange the choice of weights and abscissas to make the integral exact for a class of integrands "polynomials times some known function $W(x)$" rather than for the usual class of integrands "polynomials." The function $W(x)$ can then be chosen to remove integrable singularities from the desired integral. Given $W(x)$, in other words, and given an integer N, we can find a set of weights w_j and abscissas x_j such that the approximation.

$$\int_a^b W(x)f(x) \approx \sum_{j=1}^{N} w_j f(x_j) \tag{14}$$

is exact if $f(x)$ is a polynomial.

The theory behind Gaussian quadratures goes back to Gauss in 1814, who used continued fractions to develop the subject. In 1826 Jacobi rederived Gauss's results by means of orthogonal polynomials. The systematic treatment of arbitrary weight functions $W(x)$ using orthogonal polynomials is largely due to Christoffel in 1877. The fundamental theorem of Gaussian quadratures that lets you find the abscissas for any particular case is the following: The abscissas of the N-point Gaussian quadrature formula (14) with weight function $W(x)$ in the interval (a,b) are precisely the roots of the orthogonal polynomial $p_N(x)$ for the same interval and weight function.

Once you know the abscissas $x_1,...,x_N$, you need to find the weights $w_j, j = 1,...,N$. The most useful computational formula for the weights is

$$w_j = \langle p_{N-1} | p_{N-1} \rangle / [p_{N-1}(x_j) p'_N(x_j)], \tag{15}$$

since the derivative p'_N can be computed efficiently by the derivative of (2) in the general case, or by special relations for the classical polynomials. Note that (15) is valid as written only for monic polynomials; for other normalizations, there is an extra factor of λ_N/λ_{N-1}, where λ_N is the coefficient of x^N in p_N.

What is the best way to solve for all the abscissas and weights for some Gaussian quadrature formula? Except in some special cases, the best way is *not* to use a root-finding method like Newton's method on $p_N(x)$. It is generally faster to use the Golub–Welsch[5] algorithm, which is based on a result of Wilf.[6] This algorithm notes that if you bring the term xp_j to the left-hand side of (2) and the term p_{j+1} to the right-hand side, the recurrence relation can be written in matrix form as

$$x\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-2} \\ p_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 & 1 & & & \\ b_1 & a_1 & 1 & & \\ & \vdots & \vdots & & \\ & & b_{N-2} & a_{N-2} & 1 \\ & & & b_{N-1} & a_{N-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-2} \\ p_{N-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ p_N \end{bmatrix}$$

or

$$x\mathbf{p} = \mathbf{T}\cdot\mathbf{p} + p_N \mathbf{e}_{N-1}. \tag{16}$$

Here, $\mathbf{T}$ is a tridiagonal matrix, $\mathbf{p}$ is a column vector of $p_0, p_1, ..., p_{N-1}$, and $\mathbf{e}_{N-1}$ is a unit vector with a 1 in the $(N-1)$th (last) position and zeros elsewhere. The matrix $\mathbf{T}$ can be symmetrized by a diagonal similarity transformation $\mathbf{D}$ to give

$$\mathbf{J} = \mathbf{D}\mathbf{T}\mathbf{D}^{-1} = \begin{bmatrix} a_0 & \sqrt{b_1} & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & \\ & \vdots & \vdots & & \\ & & \sqrt{b_{N-2}} & a_{N-2} & \sqrt{b_{N-1}} \\ & & & \sqrt{b_{N-1}} & a_{N-1} \end{bmatrix}. \tag{17}$$

The matrix $\mathbf{J}$ is called the *Jacobi matrix* (not to be confused with other matrices named after Jacobi that arise in completely different problems!). Now we see from (16) that $p_N(x_j) = 0$ is equivalent to x_j being an eigenvalue of $\mathbf{T}$. Since eigenvalues are preserved by a similarity transformation, x_j is an eigenvalue of the symmetric tridiagonal matrix $\mathbf{J}$. Moreover, Wilf[6] shows from the Christoffel–Darboux identity that if $\mathbf{v}_j$ is the eigenvector corresponding to the eigenvalue x_j, normalized so that $\mathbf{v}\cdot\mathbf{v} = 1$, then

$$w_j = \mu_0 v_{j,1}^2, \tag{18}$$

where $v_{j,1}$ is the first component of $\mathbf{v}$. Now finding all eigenvalues and eigenvectors of a symmetric tridiagonal matrix is a relatively efficient and well-conditioned procedure.[7–9]

Box 2 lists the routine gaucof, for finding the abscissas and weights, given the coefficients a_j and b_j. Remember that if you know the recurrence relation for

Box 2.

```
SUBROUTINE gaucof(n,a,b,amu0,x,w)
INTEGER n,NMAX
REAL amu0,a(n),b(n),w(n),x(n)
PARAMETER (NMAX=64)
USES eigsrt,tqli
  Computes the abscissas and weights for a Gaussian quadrature formula from the Jacobi matrix.
  On input, a(1:n) and b(1:n) are the coefficients of the recurrence relation for the set
  of monic orthogonal polynomials. The quantity μ0 ≡ ∫_a^b W(x) dx is input as amu0. The
  abscissas and weights are returned in descending order in x(1:n) and w(1:n). The array b
  is modified. Execution can be speeded up by modifying tqli and eigsrt to compute only
  the first component of each eigenvector.
INTEGER i,j
REAL z(NMAX,NMAX)
do 13 i=1,n
    if(i.ne.1)b(i)=sqrt(b(i))        Set up superdiagonal of Jacobi matrix.
    do 11 j=1,n                      Set up identity matrix for tqli to compute eigenvec-
        if(i.eq.j)then                   tors.
            z(i,j)=1.
        else
            z(i,j)=0.
        endif
    enddo 11
enddo 13
call tqli(a,b,n,NMAX,z)
call eigsrt(a,z,n,NMAX)              Sort eigenvalues into descending order.
do 13 i=1,n
    x(i)=a(i)
    w(i)=amu0*z(1,i)**2              Equation (18).
enddo 13
return
END
```

orthogonal polynomials that are not normalized to be monic, you can easily convert it to monic form by means of the quantities λ_j.

So, to summarize, there are three routes that can get you to the abscissas and weights of a desired Gaussian quadrature. Which route you take depends on how standard your weight funciton is.

(1) If your weight function is "very" standard, then the fastest procedure (by a factor of 3 to 5) is to use direct root finding on $p_N(x)$ for the abscissas, by Newton's method. The reason is that good approximations are available[10] as starting guesses for the zeros of $p_N(x)$, enabling Newton's method to converge very rapidly. Newton's method requires the derivative $p'_N(x)$, which is evaluated by standard relations in terms of p_N and p_{N-1}. The weights are then evaluated from the formula (15).

(2) If your weight function is not standard enough to have known good starting guesses (or if you are too lazy to track them down), but *is* standard enough to have known recurrence coefficients a_j and b_j, e.g., tabulated in standard references,[11] then use gaucof to get the desired weights and abscissas. Remember that you need the a_j's and b_j's for the *monic* polynomials. Note that gaucof uses the Numerical Recipes[7–9] routines tqli and eigsrt. For high accuracy, it is straightforward to convert all three routines to double precision.

(3) If your weight function is completely nonclassical, then you must use orthog to get the coefficients a_j and b_j. To do so, you will need to know the recurrence coefficients for a classical set of orthogonal polynomials on the same interval [the α's and β's in equation (6)], and you will need to calculate the modified moments of equation (5) for that classical polynomial—analytically if possible, numerically (to high precision) if you must.

References

1. R. A. Sack and A. F. Donovan, Num. Math. **18**, 465 (1971/72).
2. J. C. Wheeler, Rocky Mount. J. Math. **4**, 287 (1974).
3. W. Gautschi, in *Recent Advances in Numerial Analysis*, edited by C. de Boor and G. H. Golub (Academic, New York, 1978), p. 45.
4. W. Gautschi, in *E. B. Christoffel*, edited by P. L. Butzer and F. Fehér (Birkhauser, Basel, 1981), p. 72.
5. G. H. Golub and J. H. Welsch, Math. Comput. **23**, 221, A1 (1969).
6. H. S. Wilf, *Mathematics for the Physical Sciences* (Wiley, New York, 1962), Problem 9, p. 80.
7. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge U. P., New York, 1986).
8. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge U. P., New York, 1988).
9. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Pascal: The Art of Scientific Computing* (Cambridge U. P., New York, 1989).
10. A. H. Stroud and D. Secrest, *Gaussian Quadrature Formulas* (Prentice-Hall, Englewood Cliffs, NJ, 1966).
11. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (National Bureau of Standards, Washington, 1964; reprinted by Dover Publications, New York, 1968).

In our next column: Fredholm and Volterra Integral Equations of the Second Kind.

Correction

The column on "Recursive Stratified Sampling for Multidimensional Monte Carlo Integration" (March/April, 1990), by William H. Press and Glennys R. Farrar, incorrectly characterized the performance of the program VEGAS by G.P. Lepage. Our copy of VEGAS was corrupt, with modifications not sanctioned by Professor Lepage. Also, there was a bug in our testing program.

In fact, VEGAS' performance is substantially better than that of our routine miser on the first two test problems discussed, and comparable on the third test problem. *In all cases, VEGAS' reported variances are in fact accurate.* It is possible to find test problems where miser outperforms VEGAS, but the examples given are not good cases of this.

We regret the errors. VEGAS is a program that is widely and successfully used. Because it is important that uncorrupt source code of VEGAS be generally available, Professor Lepage has agreed to provide an up-to-date discussion of the program, including a complete listing. This will appear in a future issue.