Algorithm 726: ORTHPOL—A Package of Routines for Generating Orthogonal Polynomials and Gauss-Type Quadrature Rules

WALTER GAUTSCHI
Purdue University

A collection of subroutines and examples of their uses, as well as the underlying numerical methods, are described for generating orthogonal polynomials relative to arbitrary weight functions. The object of these routines is to produce the coefficients in the three-term recurrence relation satisfied by the orthogonal polynomials. Once these are known, additional data can be generated, such as zeros of orthogonal polynomials and Gauss-type quadrature rules, for which routines are also provided.

Categories and Subject Descriptors: G.1.2 [**Numerical Analysis**]: Approximation; G.1.4 [**Numerical Analysis**]: Quadrature and Numerical Differentiation; G.4 [**Mathematical Software**]

General Terms: Algorithms

Additional Key Words and Phrases: Gauss-type quadrature rules, orthogonal polynomials

1. INTRODUCTION

Classical orthogonal polynomials, such as those of Legendre, Chebyshev, Laguerre, and Hermite, have been used for purposes of approximation in widely different disciplines and over a long period of time. Their popularity is due in part to the ease with which they can be employed and in part to the wealth of analytic results known for them. Widespread use of nonclassical orthogonal polynomials, in contrast, has been impeded by a lack of effective and generally applicable constructive methods. The present set of computer routines has been developed over the past 10 years in the hope of remedying this impediment and of encouraging the use of nonstandard orthogonal polynomials. A number of applications indeed have already been made, for

This work was supported in part by National Research Foundation grants, most recently by grant DMS-9023403.
Author's address: Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398.

example, to numerical quadrature (Cauchy principal value integrals with coth-kernel [Gautschi et al. 1987], Hilbert transform of Jacobi weight functions [Gautschi and Wimp 1987], integration over half-infinite intervals [Gautschi 1991c], rational Gauss-type quadrature [Gautschi 1993a; 1993b]), to moment-preserving spline approximation [Gautschi 1984a; Gautschi and Milovanović 1986; Frontini et al. 1987], to the summation of slowly convergent series [Gautschi 1991a, 1991b], and, perhaps most notably, to the proof of the Bieberbach conjecture [Gautschi 1986b].

In most applications, orthogonality is with respect to a positive weight function, w, on a given interval or union of intervals, or with respect to positive weights, w_ι, concentrated on a discrete set of points, $\{x_\iota\}$, or a combination of both. For convenience of notation, we subsume all of these cases under the notion of a positive measure, $d\lambda$, on the real line $\mathbb{R}$. That is, the respective inner product is written as a Riemann–Stieltjes integral,

$$(u, v) = \int_{\mathbb{R}} u(t)v(t)\, d\lambda(t), \tag{1.1}$$

where the function $\lambda(t)$ is the indefinite integral of w for the continuous part, and a step function with jumps w_ι at x_ι for the discrete part. We assume that (1.1) is meaningful whenever u, v are polynomials. There is then defined a unique set of (monic) orthogonal polynomials,

$$\begin{aligned} &\pi_k(t) = t^k + \text{lower-degree terms}, \qquad k = 0, 1, 2, \ldots, \\ &(\pi_k, \pi_\ell) = 0 \qquad \text{if} \quad k \neq \ell. \end{aligned} \tag{1.2}$$

We speak of "continuous" orthogonal polynomials if the support of $d\lambda$ is an interval or a union of intervals, of "discrete" orthogonal polynomials if the support of $d\lambda$ consists of a discrete set of points, and of orthogonal polynomials of "mixed type" if the support of $d\lambda$ has both a continuous and discrete part. In the first and last cases, there are infinitely many orthogonal polynomials, one for each degree, whereas in the second case, there are exactly N orthogonal polynomials, $\pi_0, \pi_1, \ldots, \pi_{N-1}$, where N is the number of support points. In all cases, we denote the polynomials by $\pi_k(\cdot) = \pi_k(\cdot; d\lambda)$, or $\pi_k(\cdot; w)$, if we want to indicate their dependence on the measure $d\lambda$ or weight function w, and use similar notations for other quantities depending on $d\lambda$ or w.

It is a distinctive feature of orthogonal polynomials, compared to other orthogonal systems, that they satisfy a three-term recurrence relation,

$$\begin{aligned} &\pi_{k+1}(t) = (t - \alpha_k)\pi_k(t) - \beta_k \pi_{k-1}(t), \qquad k = 0, 1, 2, \ldots, \\ &\pi_0(t) = 1, \qquad \pi_{-1}(t) = 0, \end{aligned} \tag{1.3}$$

with coefficients $\alpha_k = \alpha_k(d\lambda) \in \mathbb{R}$, $\beta_k = \beta_k(d\lambda) > 0$ that are uniquely determined by the measure $d\lambda$. By convention, the coefficient β_0, which multiplies

$\pi_{-1} = 0$ in (1.3), is defined by

$$\beta_0 = \beta_0(d\lambda) = \int_{\mathbb{R}} d\lambda(t). \tag{1.4}$$

The knowledge of these coefficients is absolutely indispensable for any sound computational use and application of orthogonal polynomials [Gautschi 1982a, 1990]. One of the principal objectives of the present package is precisely to provide routines for generating these coefficients. Routines for related quantities are also provided, such as Gauss-type quadrature weights and nodes and, hence, also zeros of orthogonal polynomials.

Occasionally (e.g., in Gautschi [1984a], Gautschi and Milovanović [1986], Frontini et al. [1987], and Gautschi [1993a; 1993b]), one needs to deal with indefinite (i.e., sign-changing) measures $d\lambda$. The positivity of the β_k is then no longer guaranteed, indeed not even the existence of all orthogonal polynomials. Nevertheless, our methods can still be formally applied, albeit at the risk of possible breakdowns or instabilities.

There are basically four methods used here to generate recursion coefficients: (1) *Methods based on explicit formulas*. These relate to classical orthogonal polynomials and are implemented in the routine **recur** of Section 2. (2) *Methods based on moment information*. These are dealt with in Section 3 and are represented by a single routine, **cheb**. Its origin can be traced back to work of Chebyshev on discrete least squares approximation. (3) *Bootstrap methods based on inner product formulas for the coefficients, and orthogonal reduction methods*. We have attributed the idea for the former method to Stieltjes, and referred to it in Gautschi [1982a] as the *Stieltjes procedure*. The prototype is the routine **sti** in Section 4, applicable for discrete orthogonal polynomials. An alternative routine is **lancz**, which accomplishes the same purpose, but uses the *method of Lanczos*. Either of these routines can be used in **mcdis**, which applies to continuous as well as to mixed-type orthogonal polynomials. In contrast to all previous routines, **mcdis** uses a discretization process and, thus, furnishes only approximate answers whose accuracies can be controlled by the user. The routine, however, is by far the most sophisticated and flexible routine in this package, one that requires, or can greatly benefit from, ingenuity of the user. The same kind of discretization is also applicable to moment-related methods, yielding the routine **mccheb**. (4) *Modification algorithms*. These are routines generating recursion coefficients for measures modified by a rational factor, utilizing the recursion coefficients of the original measure, which are assumed to be known. They can be thought of as algorithmic implementations of the Christoffel, or generalized Christoffel, theorem and are incorporated in the routines **chri** and **gchri** of Section 5. An important application of all of these routines is made in Section 6, where routines are provided that generate the weights and nodes of quadrature rules of Gauss, Gauss–Radau, and Gauss–Lobatto types.

Each routine has a single-precision and double-precision version with similar names, except for the prefix **d** in double-precision procedures. The latter are generally a straightforward translation of the former. An exception

is the routine **dlga** used in **drecur** for computing the logarithm of the gamma function, which employs a different method than the single-precision companion routine **alga**.

All routines of the package have been checked for ANSI conformance and tested on two computers: the Cyber 205 and a Sun 4/670 MP workstation. The former has machine precisions $\epsilon^s \approx 7.11 \times 10^{-15}$, $\epsilon^d \approx 5.05 \times 10^{-29}$ in single and double precision, respectively, while the latter has $\epsilon^s \approx 5.96 \times 10^{-8}$, $\epsilon^d \approx 1.11 \times 10^{-16}$. The Cyber 205 has a large floating-point exponent range, extending from approximately -8617 to $+8645$ in single as well as in double precision, whereas the Sun 4/670 has the rather limited exponent range $[-38, 38]$ in single precision, but a larger range $[-308, 308]$ in double precision. All output cited relates to work on the Cyber 205.

The package is organized as follows: Section 0 contains (slightly amended) *netlib* routines, namely, **r1mach** and **d1mach**, providing basic machine constants for a variety of computers. Section 1 contains all of the driver routines, named **test1**, **test2**, etc., which are used (and described in the body of this paper) to test the subroutines of the package. The complete output of each test is listed immediately after the driver. Sections 2–6 constitute the core of the package: The single- and double-precision subroutines described in the equally numbered sections of this paper. All single-precision routines are provided with comments and instructions for their use. These, of course, apply to the double-precision routines as well.

2. CLASSICAL WEIGHT FUNCTIONS

Among the most frequently used orthogonal polynomials are the Jacobi polynomials, generalized Laguerre polynomials, and Hermite polynomials, supported, respectively, on a finite interval, half-infinite interval, and the whole real line. The respective weight functions are

$$w(t) = w^{(\alpha,\beta)}(t) = (1-t)^{\alpha}(1+t)^{\beta} \quad \text{on } (-1,1),\ \alpha > -1,\ \beta > -1\text{: Jacobi;} \tag{2.1}$$

$$w(t) = w^{(\alpha)}(t) = t^{\alpha}e^{-t} \quad \text{on } (0,\infty),\ \alpha > -1\text{: Generalized Laguerre;} \tag{2.2}$$

$$w(t) = e^{-t^2} \quad \text{on } (-\infty,\infty)\text{: Hermite.} \tag{2.3}$$

Special cases of the Jacobi polynomials are the Legendre polynomials ($\alpha = \beta = 0$); the Chebyshev polynomials of the first ($\alpha = \beta = -\frac{1}{2}$), second ($\alpha = \beta = \frac{1}{2}$), third ($\alpha = -\beta = -\frac{1}{2}$), and fourth ($\alpha = -\beta = \frac{1}{2}$) kinds; and the Gegenbauer polynomials ($\alpha = \beta = \lambda - \frac{1}{2}$). The Laguerre polynomials are the special case $\alpha = 0$ of the generalized Laguerre polynomials.

For each of these polynomials, the corresponding recursion coefficients $\alpha_k = \alpha_k(w)$, $\beta_k = \beta_k(w)$ are explicitly known (see, e.g., Chihara [1978, pp. 217–221] and are generated in single precision by the routine **recur**. Its calling sequence is

recur(n, ipoly, al, be, a, b, ierr).

On entry,

n is the number of recursion coefficients desired; type integer.

ipoly is an integer identifying the polynomial as follows:

1 = Legendre polynomial on $(-1, 1)$;
2 = Legendre polynomial on $(0, 1)$;
3 = Chebyshev polynomial of the first kind;
4 = Chebyshev polynomial of the second kind;
5 = Chebyshev polynomial of the third kind;
6 = Jacobi polynomial with parameters **al**, **be**;
7 = generalized Laguerre polynomial with parameter **al**; and
8 = Hermite polynomial.

al, **be** are the input parameters α, β for Jacobi and generalized Laguerre polynomials; type real; they are only used if **ipoly** = 6 or 7, and in the latter case, only **al** is used.

On return,

a, **b** are real arrays of dimension **n** with $\mathbf{a}(k), \mathbf{b}(k)$ containing the coefficients $\alpha_{k-1}, \beta_{k-1}$, respectively, $k = 1, 2, \ldots, \mathbf{n}$.

ierr is an error flag, where

ierr = 0 on normal return,
ierr = 1 if either **al** or **be** is out of range when **ipoly** = 6 or **ipoly** = 7,
ierr = 2 if there is potential overflow in the evaluation of β_0 when **ipoly** = 6 or **ipoly** = 7; in this case, β_0 is set equal to the largest machine-representable number,
ierr = 3 if **n** is out of range, and
ierr = 4 if **ipoly** is not one of the admissible integers.

No provision has been made for Chebyshev polynomials of the fourth kind, since their recursion coefficients are obtained from those for the third-kind Chebyshev polynomials simply by changing the sign of the α's (and leaving the β's unchanged).

The corresponding double-precision routine is **drecur**; it has the same calling sequence, except for real data types now being double precision.

In the cases of Jacobi polynomials (**ipoly** = 6) and generalized Laguerre polynomials (**ipoly** = 7), the recursion coefficient β_0 (and only this one) involves the gamma function Γ. Accordingly, a function routine, **alga**, is provided that computes the logarithm $\ln \Gamma$ of the gamma function, and a separate routine, **gamma**, computing the gamma function by exponentiating its logarithm. Their calling sequences are

function alga(x)
function gamma(x, ierr),

where **ierr** is an output variable set equal to 2 or 0 depending on whether the gamma function does, or does not, overflow, respectively. The corresponding

double-precision routines have the names **dlga** and **dgamma**. All of these routines require machine-dependent constants for reasons explained below.

The routine **alga** is based on a rational approximation valid on the interval $[\frac{1}{2}, \frac{3}{2}]$. Outside this interval, the argument x is written as

$$x = x_e + m,$$

where

$$x_e = \begin{cases} x - \lfloor x \rfloor + 1 & \text{if } \ x - \lfloor x \rfloor \le \frac{1}{2}, \\ x - \lfloor x \rfloor & \text{otherwise} \end{cases}$$

is in the interval $(\frac{1}{2}, \frac{3}{2}]$ and where $m \ge -1$ is an integer. If $m = -1$ (i.e., $0 < x \le \frac{1}{2}$), then $\ln \Gamma(x) = \ln \Gamma(x_e) - \ln x$, while for $m > 0$, one computes $\ln \Gamma(x) = \ln \Gamma(x_e) + \ln p$, where $p = x_e(x_e + 1)\cdots(x_e + m - 1)$. If m is so large, say, $m \ge m_0$, that the product p would overflow, then $\ln p$ is computed (at a price!) as $\ln p = \ln x_e + \ln(x_e + 1) + \cdots + \ln(x_e + m - 1)$. It is here where a machine-dependent integer is required, namely, m_0 = smallest integer m such that $1 \cdot 3 \cdot 5 \cdots (2m + 1)/2^m$ is greater than or equal to the largest machine-representable number, R. By Stirling's formula, the integer m_0 is seen to be the smallest integer m satisfying $((m + 1)/e)\ln((m + 1)/e) \ge (\ln R - \frac{1}{2}\ln 8)/e$, hence, equal to $\lfloor e \cdot t((\ln R - \frac{1}{2}\ln 8)/e) \rfloor$, where $t(y)$ is the inverse function of $y = t \ln t$. For our purposes, the low-accuracy approximation of $t(y)$, given in Gautschi [1967b, pp. 51–52], and implemented in the routine **t**, is adequate.

The rational approximation chosen on $[\frac{1}{2}, \frac{3}{2}]$ is one due to W. J. Cody and K. E. Hillstrom, namely, the one labeled $n = 7$ in Table II of Cody and Hillstrom [1967]. It is designed to yield about 16 correct decimal digits (cf. Table I of Cody and Hillstrom [1967]), but because of numerical cancellation furnishes only about 13–14 correct decimal digits.

Since rational approximations for $\ln \Gamma$ having sufficient accuracies for double-precision computation do not seem to be available in the literature, we use a different approach for the routine **dlga**, namely, the asymptotic approximation (cf. eq. 6.1.42 of Abramowitz and Stegun [1964], where the constants B_{2m} are Bernoulli numbers)

$$\ln \Gamma(y) = (y - \tfrac{1}{2})\ln y - y + \tfrac{1}{2}\ln(2\pi) + \sum_{m=1}^{n} \frac{B_{2m}}{2m(2m-1)} y^{-(2m-1)} + R_n(y) \tag{2.4}$$

for values of $y > 0$ large enough to have

$$|R_n(y)| \le \frac{1}{2} 10^{-d}, \tag{2.5}$$

where d is the number of decimal digits carried in double-precision arithmetic, another machine-dependent real number. If (2.5) holds for $y \ge y_0$ and if $x \ge y_0$, we compute $\ln \Gamma(x)$ from the asymptotic expression (2.4) (where

$y = x$ and the remainder term is neglected). Otherwise, we let k_0 be the smallest positive integer k such that $x + k \geq y_0$, and use

$$\ln \Gamma(x) = \ln \Gamma(x + k_0) - \ln(x(x + 1) \cdots (x + k_0 - 1)), \tag{2.6}$$

where the first term on the right is computed from (2.4) (with $y = x + k_0$). Since, for $y > 0$,

$$|R_n(y)| \leq \frac{|B_{2n+2}|}{(2n + 2)(2n + 1)} y^{-(2n+1)}$$

(cf. Abramowitz and Stegun [1964, eq. 6.1.42]), the inequality (2.5) is satisfied if

$$y \geq \exp\left\{ \frac{1}{2n + 1} \left[d \ln 10 + \ln \frac{2|B_{2n+2}|}{(2n + 2)(2n + 1)} \right] \right\}. \tag{2.7}$$

In our routine **dlga**, we have selected $n = 9$. For double-precision accuracy on the Cyber 205, we have $d \approx 28.3$, for which (2.7) then gives $y \geq \exp\{.121188 \cdots d + .053905 \cdots\} \approx 32.6$.

For single-precision calculation, we selected the method of rational approximation, rather than the asymptotic formula (2.4) and (2.6), since we found that the former is generally more accurate, by a factor, on the average, of about 20 and as large as 300. Neither method yields full machine accuracy. The former, as already mentioned, loses accuracy in the evaluation of the approximation. The latter suffers loss of accuracy because of cancellation occurring in (2.6), which typically amounts to a loss of 2–5 significant decimal digits in the gamma function itself.

Since these errors affect only the coefficient β_0 (and only if **ipoly** = 6 or 7), they are of no consequence unless the output of the routine **recur** serves as input to another routine, such as **gauss** (cf. Section 6), which makes essential use of β_0. In this case, for maximum single-precision accuracy, it is recommended that β_0 be first obtained in double precision by means of **drecur** with **n** = 1 and then converted to single precision.

3. MOMENT-RELATED METHODS

It is a well-known fact that the first n recursion coefficients $\alpha_k(d\lambda)$, $\beta_k(d\lambda)$, $k = 0, 1, \ldots, n - 1$ (cf. (1.3)), are uniquely determined by the first $2n$ moments μ_k of the measure $d\lambda$,

$$\mu_k = \mu_k(d\lambda) = \int_{\mathbb{R}} t^k \, d\lambda(t), \qquad k = 0, 1, 2, \ldots, 2n - 1. \tag{3.1}$$

Formulas are known, for example, that express the α's and β's in terms of Hankel determinants in these moments. The trouble is that these formulas become increasingly sensitive to small errors as n becomes large. There is an inherent reason for this: The underlying (nonlinear) map $K_n: \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ has

a condition number, cond K_n, that grows exponentially with n (cf. Gautschi [1982a, sect. 3.2]). Any method that attempts to compute the desired coefficients from the moments in (3.1), therefore, is doomed to fail, unless n is quite small or extended precision is being employed. That goes, in particular, for an otherwise elegant method due to Chebyshev (who developed it for the case of discrete measures $d\lambda$) that generates the α's and β's directly from the moments (3.1), bypassing determinants altogether (cf. Chebyshev [1859] and Gautschi [1982a, sect. 2.3]).

Variants of Chebyshev's algorithm with more satisfactory stability properties have been developed by Sack and Donovan [1972] and by Wheeler [1974] (independently of Chebyshev's work). The idea is to forgo the moments (3.1) as input data and instead depart from so-called *modified moments*. These are defined by replacing the power t^k in (3.1) by an appropriate polynomial $p_k(t)$ of degree k,

$$\nu_k = \nu_k(d\lambda) = \int_{\mathbb{R}} p_k(t)\, d\lambda(t), \qquad k = 0, 1, 2, \ldots, 2n-1. \tag{3.2}$$

For example, p_k could be one of the classical orthogonal polynomials. More generally, we shall assume that $\{p_k\}$ are monic polynomials satisfying a three-term recurrence relation similar to the one in (1.3),

$$\begin{aligned} p_{k+1}(t) &= (t - a_k)p_k(t) - b_k p_{k-1}(t), \qquad k = 0, 1, 2, \ldots, \\ p_0(t) &= 1, \qquad p_{-1}(t) = 0, \end{aligned} \tag{3.3}$$

with coefficients $a_k \in \mathbb{R}$, $b_k \geq 0$ that are known. (In the special case $a_k = 0$, $b_k = 0$, we are led back to powers and ordinary moments.) There now exists an algorithm, called the *modified Chebyshev algorithm* in Gautschi [1982a, sect. 2.4], which takes the $2n$ modified moments in (3.2) and the $2n-1$ coefficients $\{a_k\}_{k=0}^{2n-2}$, $\{b_k\}_{k=0}^{2n-2}$ in (3.3), and from them generates the n desired coefficients $\alpha_k(d\lambda)$, $\beta_k(d\lambda)$, $k = 0, 1, \ldots, n-1$. It generalizes Chebyshev's algorithm, which can be recovered (if need be) by putting $a_k = b_k = 0$. The modified Chebyshev algorithm is embodied in the subroutine **cheb**, which has the calling sequence

```
cheb(n, a, b, fnu, alpha, beta, s, ierr, s0, s1, s2)
dimension a(*), b(*), fnu(*), alpha(n), beta(n), s(n),
  s0(*), s1(*), s2(*)
```

On entry,

n is the number of recursion coefficients desired; type integer.

a, b are arrays of dimension $2 \times \mathbf{n} - 1$ holding the coefficients $\mathbf{a}(k) = a_{k-1}$, $\mathbf{b}(k) = b_{k-1}$, $k = 1, 2, \ldots, 2n-1$.

fnu is an array of dimension $2 \times \mathbf{n}$ holding the modified moments $\mathbf{fnu}(k) = \nu_{k-1}$, $k = 1, 2, \ldots, 2 \times \mathbf{n}$.

On return,

alpha, beta are arrays of dimension **n** containing the desired recursion coefficients $\mathbf{alpha}(k) = \alpha_{k-1}, \mathbf{beta}(k) = \beta_{k-1}, k = 1, 2, \ldots, \mathbf{n}$.

s is an array of dimension **n** containing the numbers $\mathbf{s}(k) = \int_{\mathbb{R}} \pi_{k-1}^2 \, d\lambda, k = 1, 2, \ldots, \mathbf{n}$.

ierr is an error flag, equal to 0 on normal return, equal to 1 if $|\nu_0|$ is less than the machine zero, equal to 2 if **n** is out of range, equal to $-(k-1)$ if $\mathbf{s}(k), k = 1, 2, \ldots, \mathbf{n}$, is about to underflow, and equal to $+(k-1)$ if it is about to overflow.

The arrays **s0, s1, s2** of dimension $2 \times \mathbf{n}$ are needed for working space.

There is again a map $K_n: \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ underlying the modified Chebyshev algorithm, namely, the map taking the $2n$ modified moments into the n pairs of recursion coefficients. The condition of the map K_n (actually of a somewhat different, but closely related, map) has been studied in [Gautschi 1982a, sect. 3.3; 1986a] in the important case where the polynomials p_k defining the modified moments are themselves orthogonal polynomials, $p_k(\cdot) = p_k(\cdot; d\mu)$, with respect to a measure $d\mu$ (e.g., one of the classical ones) for which the recursion coefficients a_k, b_k are known. The upshot of the analysis then is that the condition of K_n is characterized by a certain positive polynomial $g_n(\cdot; d\lambda)$ of degree $4n - 2$, depending only on the target measure $d\lambda$, in the sense that

$$\operatorname{cond} K_n = \int_{\mathbb{R}} g_n(t; d\lambda) \, d\mu(t). \tag{3.4}$$

Thus, the numerical stability of the modified Chebyshev algorithm is determined by the magnitude of g_n on the support of $d\mu$.

The occurrence of underflow (overflow) in the computation of the α's and β's, especially on computers with limited exponent range, can often be avoided by multiplying all modified moments by a sufficiently large (small) scaling factor before entering the routine. On exit, the coefficient β_0 (and only this one!) then has to be divided by the same scaling factor. (There may occur harmless underflow of auxiliary quantities in the routine **cheb**, which is difficult to avoid since some of these quantities actually are expected to be zero.)

Example 3.1 $d\lambda_\omega(t) = [(1 - \omega^2 t^2)(1 - t^2)]^{-1/2} \, dt$ on $(-1, 1)$, $0 \le \omega < 1$. This example is of some historical interest, in that it has already been considered by Christoffel [1877, example 6]; see also Rees [1945]. Computationally, the example is of interest as there are empirical reasons to believe that for the choice $d\mu(t) = (1 - t^2)^{-1/2} \, dt$ on $(-1, 1)$, which appears rather natural, the modified Chebyshev algorithm is exceptionally stable, uniformly in n, in the sense that in (3.4) one has $g_n \le 1$ on $\operatorname{supp} d\mu$ for all n (cf. Gautschi [1984b, example 5.2]). With the above choice of $d\mu$, the polynomials p_k are clearly the Chebyshev polynomials of the first kind, $p_0 = T_0, p_k =$

$2^{-(k-1)}T_k$, $k \geq 1$, and the modified moments are given by

$$\nu_0 = \int_{-1}^{1} d\lambda_\omega(t), \qquad \nu_k = \frac{1}{2^{k-1}} \int_{-1}^{1} T_k(t)\, d\lambda_\omega(t), \qquad k = 1,2,3,\ldots. \tag{3.5}$$

They are expressible in terms of the Fourier coefficients $C_r(\omega^2)$ in

$$(1 - \omega^2 \sin^2\theta)^{-1/2} = C_0(\omega^2) + 2\sum_{r=1}^{\infty} C_r(\omega^2)\cos 2r\theta \tag{3.6}$$

by means of (cf. Gautschi [1982a, example 3.3])

$$\left.\begin{aligned} \nu_0 &= \pi C_0(\omega^2), \\ \nu_{2m} &= \frac{(-1)^m \pi}{2^{2m-1}} C_m(\omega^2) \\ \nu_{2m-1} &= 0 \end{aligned}\right\} \qquad m = 1,2,3,\ldots. \tag{3.7}$$

The Fourier coefficients $\{C_r(\omega^2)\}$, in turn, can be accurately computed as the minimal solution of a certain three-term recurrence relation (see Gautschi [1982a, pp. 310–311]).

The ordinary moments

$$\mu_0 = \nu_0, \qquad \mu_k = \int_{-1}^{1} t^k\, d\lambda_\omega(t), \qquad k = 1,2,3,\ldots, \tag{3.8}$$

likewise can be expressed in terms of the Fourier coefficients $C_r(\omega^2)$ by writing t^{2m} as a linear combination of Chebyshev polynomials $T_0, T_2, \ldots, T_{2m}$ (cf. Luke [1975, Eq. 22, p. 454]). The result is

$$\left.\begin{aligned} \mu_{2m} &= \frac{(-1)^m \pi}{2^{2m-1}} \sum_{r=0}^{m} (-1)^r \gamma_r^{(m)} C_{m-r}(\omega^2) \\ \mu_{2m-1} &= 0 \end{aligned}\right\} \qquad m = 1,2,3,\ldots, \tag{3.9}$$

where

$$\begin{aligned} \gamma_0^{(m)} &= 1, \\ \gamma_r^{(m)} &= \frac{2m+1-r}{r}\gamma_{r-1}^{(m)}, \qquad r = 1,2,\ldots,m-1, \\ \gamma_m^{(m)} &= \frac{m+1}{2m}\gamma_{m-1}^{(m)}. \end{aligned} \tag{3.10}$$

The driver routine **test1** (in Section 1 of the package) generates for $\omega^2 = .1(.2).9, .99, .999$ the first n recurrence coefficients $\beta_k(d\lambda_\omega)$ (all $\alpha_k = 0$), both in single and double precision, using modified moments if **modmom** = **.true.** and ordinary moments otherwise. In the former case, $n = 80$; in the latter, $n = 20$. It prints the double-precision values of β_k, together with the relative errors of the single-precision values (computed as the difference of

Table I. Selected Output from **test1** in the Case of Modified Moments

ω^2	k	β_k^{double}	err β_k^{single}
.100	0	3.22488269744043879645983 2725	1.433(−14)
	1	0.50658408063826844751584957 27	1.187(−14)
	5	0.24999995389003190188102826 7	1.109(−14)
	11	0.24999999999999999963650485 40	1.454(−18)
	18	0.25000000000000000000000000	0.000
.500	0	3.70814935460274383686770069 4	9.005(−15)
	1	0.54305341895553637462503337 73	2.431(−14)
	8	0.24999998464317232960837794 80	4.109(−15)
	20	0.24999999999999999978894635 84	8.442(−18)
	35	0.25000000000000000000000000	0.000
.900	0	5.15618422669634637640514154 3	6.950(−15)
	1	0.63497316614524587116224926 13	7.920(−15)
	19	0.24999999569259500946295028 30	1.820(−14)
	43	0.24999999999999982821041008 96	6.872(−16)
	79	0.24999999999999999999999999 62	1.525(−26)
.999	0	9.68226512110059406067820825 7	1.194(−13)
	1	0.79378214213851769655317195 71	6.311(−14)
	19	0.24990638943982092000474525 37	1.026(−14)
	43	0.24999558226336808258597500 68	8.282(−15)
	79	0.24999984176881578761530692 11	1.548(−15)

the double-precision and single-precision values divided by the double-precision value). In **test1**, as well as in all subsequent drivers, not all error flags are interrogated for possible malfunction. The user is urged, however, to do so as a matter of principle.

The routine

fmm(n, eps, modmom, om2, fnu, ierr, f, f0, rr)

used by the driver computes the first $2 \times \mathbf{n}$ modified (ordinary) moments for $\omega^2 = \mathbf{om2}$, to a relative accuracy **eps** if **modmom** = **.true.** (**.false.**). The results are stored in the array **fnu**. The arrays **f**, **f0**, and **rr** are internal working arrays of dimension **n**, and **ierr** is an error flag. On normal return, **ierr** = 0; otherwise, **ierr** = 1, indicating lack of convergence (within a prescribed number of iterations) of the backward recurrence algorithm for computing the minimal solution $\{C_r(\omega^2)\}$. The latter is likely to occur if ω^2 is too close to 1. The routine **fmm**, as well as its double-precision version **dmm**, is listed immediately after the routine **test1**.

Table I shows selected results from the output of **test1**, when **modmom** = **.true.** (Complete results are given in the package immediately after **test1**.) The values for $k = 0$ are expressible in terms of the complete elliptic integral, $\beta_0 = 2K(\omega^2)$, and were checked, where possible, against the 16S-values in Abramowitz and Stegun [1964, Table 17.1]. In all cases, there was agreement to all 16 digits. The largest relative error observed was 2.43×10^{-13} for $\omega^2 = .999$ and $k = 2$. When $\omega^2 \leq .99$, the error was always less than 2.64×10^{-14}, which confirms the extreme stability of the modified Chebyshev algo-

Table II. Selected Output from **test1** in the Case of Ordinary Moments

ω^2	k	err β_k	ω^2	k	err β_k
.100	1	1.187(−14)	.900	1	3.270(−15)
	7	2.603(−10)		7	4.819(−10)
	13	9.663(−6)		13	1.841(−5)
	19	4.251(−1)		19	6.272(−1)
.500	1	2.431(−14)	.999	1	6.311(−14)
	7	5.571(−10)		7	1.745(−9)
	13	9.307(−6)		13	8.589(−5)
	19	5.798(−1)		19	4.808(0)

rithm in this example. It can be seen (as was to be expected) that for ω^2 not too close to 1, the coefficients converge rapidly to $\frac{1}{4}$.

In contrast, Table II shows selected results (for complete results, see the package) in the case of ordinary moments (**modmom** = **.false.**) and demonstrates the severe instability of the Chebyshev algorithm. Note that the moments themselves are all accurate to essentially machine precision, as has been verified by additional computations.

The next example deals with another weight function for which the modified Chebyshev algorithm performs rather well.

Example 3.2 $d\lambda_\sigma(t) = t^\sigma \ln(1/t)\,dt$ on $(0,1]$, $\sigma > -1$.

What is nice about this example is that both modified and ordinary moments of $d\lambda_\sigma$ are known in closed form. The latter are obviously given by

$$\mu_k(d\lambda_\sigma) = \frac{1}{(\sigma+1+k)^2} \qquad k = 0,1,2,\ldots, \tag{3.11}$$

whereas the former, relative to shifted monic Legendre polynomials (**ipoly** = 2 in **recur**), are (cf. Gautschi [1979])

$$\frac{(2k)!}{k!^2}\nu_k(d\lambda_\sigma) = \begin{cases} (-1)^{k-\sigma}\dfrac{\sigma!^2(k-\sigma-1)!}{(k+\sigma+1)!}, & 0 \le \sigma < k, \\ & \sigma \text{ an integer}, \\ \dfrac{1}{\sigma+1}\left\{\dfrac{1}{\sigma+1} + \displaystyle\sum_{r=1}^{k}\left(\dfrac{1}{\sigma+1+r} - \dfrac{1}{\sigma+1-r}\right)\right\} & \\ \cdot \displaystyle\prod_{r=1}^{k}\dfrac{\sigma+1-r}{\sigma+1+r}, & \text{otherwise}. \end{cases} \tag{3.12}$$

The routines **fmm** and **dmm** appended to **test2** in Section 1 of the package, similarly as the corresponding routines in Example 3.1, generate the first

Table III. Selected Output from **test2** in the Case of Modified Moments

σ	k	α_k	β_k
− .5	0	.111111111111111111111111	4.000000000000000000000000
	12	.499497191609463856624220 2	0.062312770828774884775638 86
	24	.499866291232421894380159 2	0.062453725573422426004572 26
	48	.499965263548544580066196 9	0.062488567177486847424336 18
	99	.499991618402435627167078 9	0.062497338230518216369371 56
0	0	.250000000000000000000000	1.000000000000000000000000
	12	.499283180215736131027262 5	0.062383568359535711235603 30
	24	.499806283948614639850153 2	0.062471000844691110016391 28
	48	.499949408379702387935642 4	0.062492812681109674623738 89
	99	.499987799201590328304791 9	0.062498326706169259262048 96
.5	0	.360000000000000000000000	0.444444444444444444444444
	12	.499375573291755564420326 7	0.062370827382807526119608 87
	24	.499832449770639448872272 5	0.062465810119454968835430 89
	48	.499956727522377172779152 1	0.062491153327110271766959 32
	99	.499989693184178978188767 4	0.062497872512816829738256 35

$2 \times \mathbf{n}$ modified moments $\nu_0, \nu_1, \ldots, \nu_{2n-1}$ if **modmom** = **.true.** and the first $2 \times \mathbf{n}$ ordinary moments otherwise. The calling sequence of **fmm** is

fmm(n, modmom, intexp, sigma, fnu).

The logical variable **intexp** is to be set **.true.** if σ is an integer and **.false.** otherwise. In either case, the input variable **sigma** is assumed to be of type **real**.

The routine **test2** generates the first **n** recursion coefficients $\alpha_k(d\lambda_\sigma)$, $\beta_k(d\lambda_\sigma)$ in single and double precision for $\sigma = -\frac{1}{2}, 0, \frac{1}{2}$, where $\mathbf{n} = 100$ for the modified Chebyshev algorithm (**modmom** = **.true.**) and $\mathbf{n} = 12$ for the classical Chebyshev algorithm (**modmom** = **.false.**). Selected double-precision results to 25 significant digits, when modified moments are used, are shown in Table III. (The complete results are given in the package after **test2**.)

The largest relative errors observed, over all $k = 0, 1, \ldots, 99$, were, respectively, 6.211×10^{-11}, 2.237×10^{-12}, and 1.370×10^{-12} for the α's and 1.235×10^{-10}, 4.446×10^{-12}, and 2.724×10^{-12} for the β's, attained consistently at $k = 99$. The accuracy achieved is slightly less than in Example 3.1, for reasons explained in Gautschi [1984b, Example 5.3].

The complete results for $\sigma = -\frac{1}{2}$ are also available in Gautschi [1991b, Appendix, Table 1]. (They differ occasionally by one unit in the last decimal place from those produced here, probably because of a slightly different computation of the modified moments.) The results for $\sigma = 0$ can be checked up to $k = 15$ against the 30S-values given in Stroud and Secrest [1966, p. 92], and for $16 \le k \le 19$ against 12S-values in Danloy [1973, Table 3]. There is complete agreement to all 25 digits in the former case and agreement to 12 digits in the latter, although there are occasional end-figure discrepancies of one unit. These are believed to be due to rounding errors committed in Danloy [1973], since similar discrepancies occur also in the range $k \le 15$. We

Table IV. Selected Output from **test2** in the Case of Ordinary Moments

k	σ	err α_k	err β_k	σ	err α_k	err β_k	σ	err α_k	err β_k
2	−.5	1.8(−13)	7.7(−14)	0	4.2(−13)	7.6(−13)	.5	1.6(−12)	2.6(−13)
5		2.2(−9)	1.2(−9)		4.2(−9)	1.2(−10)		1.3(−8)	6.6(−9)
8		1.1(−5)	5.5(−6)		4.3(−6)	3.8(−6)		6.0(−5)	5.2(−6)
11		2.5(−1)	1.7(−1)		1.3(0)	3.2(−1)		2.2(0)	4.7(−1)

do not know of any tables for $\sigma = \frac{1}{2}$, but a test is given in Section 5, Example 5.1.

The use of ordinary moments (**modmom** = **.false.**) produces predictably worse results, the relative errors of which are shown in Table IV.

4. STIELTJES, ORTHOGONAL REDUCTION, AND DISCRETIZATION PROCEDURES

4.1 The Stieltjes Procedure

It is well known that the coefficients $\alpha_k(d\lambda)$, $\beta_k(d\lambda)$ in the basic recurrence relation (1.3) can be expressed in terms of the orthogonal polynomials (1.2) and the inner product (1.1) as follows:

$$\alpha_k(d\lambda) = \frac{(t\pi_k, \pi_k)}{(\pi_k, \pi_k)}, \qquad k \geq 0;$$

$$\beta_0(d\lambda) = (\pi_0, \pi_0), \qquad \beta_k(d\lambda) = \frac{(\pi_k, \pi_k)}{(\pi_{k-1}, \pi_{k-1})}, \qquad k \geq 1. \tag{4.1}$$

Provided that the inner product can be readily calculated, (4.1) suggests the following "bootstrap" procedure: Compute α_0 and β_0 by the first relations in (4.1) for $k = 0$. Then use the recurrence relation (1.3) for $k = 0$ to obtain π_1. With π_0 and π_1 known, apply (4.1) for $k = 1$ to get α_1, β_1, then again apply (1.3) to obtain π_2, and so on. In this way, alternating between (4.1) and (1.3), we can bootstrap ourselves up to as many of the coefficients α_k, β_k as are desired. We attributed this procedure to Stieltjes and called it *Stieltjes's procedure* in Gautschi [1982a].

In the case of discrete orthogonal polynomials, that is, for inner products of the form

$$(u, v) = \sum_{k=1}^{N} w_k u(x_k) v(x_k), \qquad w_k > 0, \tag{4.2}$$

Stieltjes's procedure is easily implemented; the resulting routine is called **sti** and has the calling sequence

sti(n, ncap, x, w, alpha, beta, ierr, p0, p1, p2).

On entry,

n is the number of recursion coefficients desired; type integer.

ncap is the number of terms, N, in the discrete inner product; type integer.

x, w are arrays of dimension **ncap** holding the abscissas $\mathbf{x}(k) = x_k$ and weights $\mathbf{w}(k) = w_k$, $k = 1, 2, \ldots,$ **ncap**, of the discrete inner product.

On return,

alpha, beta are arrays of dimension **n** containing the desired recursion coefficients $\mathbf{alpha}(k) = \alpha_{k-1}$, $\mathbf{beta}(k) = \beta_{k-1}$, $k = 1, 2, \ldots,$ **n**.

ierr is an error flag having the value 0 on normal return and the value 1 if **n** is not in the proper range $1 \le n \le N$; if during the computation of a recursion coefficient with index k there is impending underflow or overflow, **ierr** will have the value $-k$ in case of underflow and the value $+k$ in case of overflow. (No error flag is set in case of harmless underflow.)

The arrays **p0, p1, p2** are working arrays of dimension **ncap**. The double-precision routine has the name **dsti**.

Occurrence of underflow (overflow) can be forestalled by multiplying all weights w_k by a sufficiently large (small) scaling factor prior to entering the routine. Upon return, the coefficient β_0 will then have to be readjusted by dividing it by the same scaling factor.

4.2 Orthogonal Reduction Method

Another approach to producing the recursion coefficients α_k, β_k from the quantities x_k, w_k defining the inner product (4.2) is based on the observation (cf. Boley and Golub [1987] and Gautschi [1991d, sect. 7]) that the symmetric tridiagonal matrix of order $N + 1$,

$$J(d\lambda_N) = \begin{bmatrix} 1 & \sqrt{\beta_0} & & & 0 \\ \sqrt{\beta_0} & \alpha_0 & \sqrt{\beta_1} & & \\ & \sqrt{\beta_1} & \alpha_1 & \ddots & \\ & & \ddots & \ddots & \sqrt{\beta_{N-1}} \\ 0 & & & \sqrt{\beta_{N-1}} & \alpha_{N-1} \end{bmatrix} \tag{4.3}$$

(the "extended Jacobi matrix" for the discrete measure $d\lambda_N$ implied in (4.2)), is orthogonally similar to the matrix

$$\begin{bmatrix} 1 & \sqrt{w}^T \\ \sqrt{w} & D_x \end{bmatrix}, \qquad \sqrt{w} = \begin{bmatrix} \sqrt{w_1} \\ \vdots \\ \sqrt{w_N} \end{bmatrix}, \qquad D_x = \begin{bmatrix} x_1 & & 0 \\ & \ddots & \\ 0 & & x_N \end{bmatrix}. \tag{4.4}$$

Hence, the desired matrix $J(d\lambda_N)$ can be obtained by applying Lanczos's algorithm to the matrix (4.4). This is implemented in the routine

lancz(n, ncap, x, w, alpha, beta, ierr, p0, p1),

which uses a judiciously constructed sequence of Givens transformations to accomplish the orthogonal similarity transformation (cf. Rutishauser [1963], de Boor and Golub [1978], Gragg and Harrod [1984], and Boley and Golub [1987]; the routine **lancz** is adapted from the routine RKPW in Gragg and Harrod [1984, p. 328]). The input and output parameters of the routine **lancz** have the same meaning as in the routine **sti**, except that **ierr** can only have the value 0 or 1, while **p0**, **p1** are again working arrays of dimension **ncap**. The double-precision version of the routine is named **dlancz**.

The routine **lancz** is generally superior to the routine **sti**: The procedure used in **sti** may develop numerical instability from some point on and therefore give inaccurate results for larger values of **n**. It furthermore is subject to underflow and overflow conditions. None of these shortcomings is shared by the routine **lancz**. On the other hand, there are cases where **sti** does better than **lancz** (cf. Example 4.5).

We illustrate the phenomenon of instability (which is explained in Gautschi [1993c]) in the case of the "discrete Chebyshev" polynomials.

Example 4.1 The inner product (4.2) with $x_k = -1 + 2(k-1)/(N-1)$, $w_k = 2/N$, $k = 1, 2, \ldots, N$.

This generates discrete analogues of the Legendre polynomials, which they indeed approach as $N \to \infty$. The recursion coefficients are explicitly known:

$$\alpha_k = 0, \qquad k = 0, 1, \ldots, N-1;$$

$$\beta_0 = 2, \qquad \beta_k = \left(1 + \frac{1}{N-1}\right)^2 \left(1 - \left(\frac{k}{N}\right)^2\right)\left(4 - \frac{1}{k^2}\right)^{-1}, \qquad k = 1, 2, \ldots, N-1. \tag{4.5}$$

To find out how well the routines **sti** and **lancz** generate them (in single precision), when $N = 40$, 80, 160, and 320, we wrote the driver **test3**, which computes the respective absolute errors for the α's and relative errors for the β's.

Selected results for Stieltjes's algorithm are shown in Table V. The gradual deterioration, after some point (depending on N), is clearly visible. Lanczos's method, in contrast, preserves essentially full accuracy; the largest error in the α's is 1.42(−13), 2.27(−13), 4.83(−13), and 8.74(−13) for $N = 40$, 80, 160, and 320, respectively, and 3.38(−13), 6.63(−13), 2.17(−12), and 5.76(−12) for the β's.

4.3 Multiple-Component Discretization Procedure

We now assume a measure $d\lambda$ of the form

$$d\lambda(t) = w(t)\,dt + \sum_{j=1}^{p} y_j \delta(t - x_j)\,dt, \qquad p \geq 0, \tag{4.6}$$

Table V. Errors in the Recursion Coefficients α_k, β_k of (4.5) Computed by Stieltjes's Procedure

N	n	err α	err β	N	n	err α	err β
40	≤ 35	$\leq 1.91(-13)$	$\leq 7.78(-13)$	160	≤ 76	$\leq 2.98(-13)$	$\leq 7.61(-13)$
	36	$3.01(-12)$	$1.48(-11)$		85	$1.61(-9)$	$1.57(-8)$
	37	$6.93(-11)$	$3.55(-10)$		94	$1.25(-4)$	$1.17(-3)$
	38	$2.57(-9)$	$1.30(-8)$		103	$2.64(-3)$	$1.51(-1)$
	39	$1.93(-7)$	$9.58(-7)$		112	$2.35(-3)$	1.16(0)
80	≤ 53	$\leq 2.04(-13)$	$\leq 6.92(-13)$	320	≤ 106	$\leq 8.65(-13)$	$\leq 7.39(-13)$
	57	$2.04(-10)$	$5.13(-10)$		117	$3.96(-10)$	$7.73(-10)$
	61	$3.84(-7)$	$9.35(-7)$		128	$2.46(-6)$	$4.67(-6)$
	65	$1.94(-3)$	$4.61(-3)$		139	$2.94(-2)$	$6.27(-2)$
	69	$1.87(-1)$	6.14(0)		150	$1.15(-3)$	$2.18(-2)$

consisting of a continuous part, $w(t)\,dt$, and (if $p > 0$) a discrete part written in terms of the Dirac δ-function. The support of the continuous part is assumed to be an interval or a finite union of disjoint intervals, some of which may extend to infinity. In the discrete part, the abscissas x_j are assumed pairwise distinct, and the weights positive, $y_j > 0$. The inner product (1.1), therefore, has the form

$$(u, v) = \int_{\mathbb{R}} u(t)v(t)w(t)\,dt + \sum_{j=1}^{p} y_j u(x_j)v(x_j). \tag{4.7}$$

The basic idea of the discretization procedure is rather simple: One approximates the continuous part of the inner product, that is, the integral in (4.7), by a sum, using a suitable quadrature scheme. If the latter involves N terms, this replaces the inner product (4.7) by a discrete inner product $(\cdot, \cdot)_{N+p}$ consisting of $N + p$ terms, the N "quadrature terms," and the p original terms. In effect, the measure $d\lambda$ in (4.6) is approximated by a discrete $(N + p)$-point measure $d\lambda_{N+p}$. We then compute the desired recursion coefficients from the formulas (4.1), in which the inner product $(\cdot, \cdot)$ is replaced, throughout, by $(\cdot, \cdot)_{N+p}$. Thus, in effect, we approximate

$$\alpha_k(d\lambda) \approx \alpha_k(d\lambda_{N+p}), \qquad \beta_k(d\lambda) \approx \beta_k(d\lambda_{N+p}). \tag{4.8}$$

The quantities on the right can be computed by the methods in Section 4.1 or 4.2, that is, employing the routines **sti** or **lancz**.

The difficult part of this approach is to find a discretization that results in rapid convergence, as $N \to \infty$, of the approximations on the right of (4.8) to the exact values on the left, even in cases where the weight function w in (4.6) exhibits singular behavior. (The speed of convergence, of course, is unaffected by the discrete part of the inner product (4.7).) To be successful in this endeavor often requires considerable inventiveness on the part of the user. Our routines, **mcdis** and **dmcdis**, which implement this idea in single (resp., double) precision, however, are designed to be flexible enough to promote the use of effective discretization procedures.

Indeed, if the support of the weight function w in (4.7) is contained in the (finite or infinite) interval (a, b), it is often useful to first decompose that

interval into a finite number of subintervals,

$$\operatorname{supp} w \subset [a,b] = \bigcup_{i=1}^{m} [a_i, b_i], \qquad m \geq 1, \tag{4.9}$$

and to approximate the inner product separately on each subinterval $[a_i, b_i]$, using an appropriate weighted quadrature rule. Thus, the integral in (4.7) is written as

$$\int_{\mathbb{R}} u(t)v(t)w(t)\,dt = \sum_{i=1}^{m} \int_{a_i}^{b_i} u(t)v(t)w_i(t)\,dt, \tag{4.10}$$

where w_i is an appropriate weight function on $[a_i, b_i]$. The intervals $[a_i, b_i]$ are not necessarily disjoint. For example, the weight function w may be the sum $w = w_1 + w_2$ of two weight functions on $[a, b]$, which we may want to treat individually (cf. Example 4.2). In that case, one would take $[a_1, b_1] = [a_2, b_2] = [a, b]$ and w_1 on the first interval, and w_2 on the other. Alternatively, we may simply want to use a composite quadrature rule to approximate the integral, in which case (4.9) is a partition of $[a, b]$ and $w_i(t) = w(t)$ for each i. Still another example is a weight function w that is already supported on a union of disjoint intervals; in this case, (4.9) would be the same union, or possibly a refined union where some of the subintervals are further partitioned.

In whichever way (4.9) and (4.10) are constructed, each integral on the right of (4.10) is now approximated by an appropriate quadrature rule,

$$\int_{a_i}^{b_i} u(t)v(t)w_i(t)\,dt \approx Q_i(uv), \tag{4.11}$$

where

$$Q_i f = \sum_{r=1}^{N_i} w_{r,i} f(x_{r,i}). \tag{4.12}$$

This gives rise to the approximate inner product

$$\begin{aligned} (u,v)_{N+p} &= \sum_{i=1}^{m} \sum_{r=1}^{N_i} w_{r,i} u(x_{r,i}) v(x_{r,i}) + \sum_{j=1}^{p} y_j u(x_j) v(x_j), \\ N &= \sum_{i=1}^{m} N_i. \end{aligned} \tag{4.13}$$

In our routine **mcdis**, we have chosen, for simplicity, all N_i to be the same integer N_0,

$$N_i = N_0, \qquad i = 1, 2, \ldots, m, \tag{4.14}$$

so that $N = mN_0$. Furthermore, if n is the number of α_k and the number of β_k desired, we have used the following iterative procedure to determine the coefficients α_k, β_k to a prescribed (relative) accuracy ϵ: Let N_0 be increased through a sequence $\{N_0^{[s]}\}_{s=0,1,2,}$ of integers, for each s use Stieltjes's (or

Lanczos's) algorithm to compute $\alpha_k^{[s]} = \alpha_k(d\lambda_{mN_0^{[s]}+p})$, $\beta_k^{[s]} = \beta_k(d\lambda_{mN_0^{[s]}+p})$, $k = 0, 1, \ldots, n-1$, and stop the iteration for the first $s \geq 1$ for which all inequalities

$$|\beta_k^{[s]} - \beta_k^{[s-1]}| \leq \epsilon\beta_k^{[s]}, \qquad k = 0, 1, \ldots, n-1, \tag{4.15}$$

are satisfied. An error flag is provided if within a preset range $N_0^{[s]} \leq N_0^{\max}$ the stopping criterion (4.15) cannot be satisfied. Note that the latter is based solely on the β-coefficients. This is because, unlike the α's, they are known to be always positive, so that it makes sense to insist on relative accuracy. (In our routine we actually replaced $\beta_k^{[s]}$ on the right of (4.15) by its absolute value to ensure proper termination in cases of sign-changing measures $d\lambda$.)

In view of formulas (4.1), it is reasonable to expect, and indeed has been observed in practice, that satisfaction of (4.15) entails sufficient absolute accuracy for the α's if they are zero or small, and relative accuracy otherwise.

Through a bit of experimentation, we have settled on the following sequence of integers $N_0^{[s]}$:

$$\begin{aligned} N_0^{[0]} &= 2n, & N_0^{[s]} &= N_0^{[s-1]} + \Delta s, & s &= 1, 2, \ldots, \\ \Delta_1 &= 1, & \Delta_s &= 2^{\lfloor s/5 \rfloor} \cdot n, & s &= 2, 3 \ldots . \end{aligned} \tag{4.16}$$

Note that if the quadrature formula (4.11) is exact for each i, whenever $u \cdot v$ is a polynomial of degree $\leq 2n-1$ (which is the maximum degree occurring in the inner products of (4.1), when $k \leq n-1$), then our procedure converges after the very first iteration step! Therefore, if each quadrature rule Q_i has (algebraic) degree of exactness $\geq d(N_0)$ and if $d(N_0)/N_0 = \delta + O(N_0^{-1})$ as $N_0 \to \infty$, then we let $N_0^{[0]} = 1 + \lfloor (2n-1)/\delta \rfloor$ in an attempt to get exact answers after one iteration. Normally, $\delta = 1$ (for interpolatory rules) or $\delta = 2$ (for Gauss-type rules).

The calling sequence of the multiple-component discretization routine is as follows:

```
mcdis(n, ncapm, mc, mp, xp, yp, quad, eps, iq, idelta, irout,
  finl, finr, endl, endr, xfer, wfer, alpha, beta, ncap,
  kount, ierr, ie, be, x, w, xm, wm, p0, p1, p2)
dimension xp(*), yp(*), endl(mc), endr(mc), xfer(ncapm),
  wfer(ncapm), alpha(n), beta(n), be(n), x(ncapm),
  w(ncapm), xm(*), wm(*), p0(*), p1(*), p2(*)
logical finl, finr
```

On entry,

n is the number of recursion coefficients desired; type integer.

ncapm is the integer $N_0^{\max}$ above, that is, the maximum integer N_0 allowed (**ncapm** = 500 will usually be satisfactory).

mc is the number of component intervals in the continuous part of the spectrum; type integer.

mp is the number of points in the discrete part of the spectrum; type integer; if the measure has no discrete part, set **mp** = 0.

xp, yp are arrays of dimension **mp** containing the abscissas and the jumps of the point spectrum.

quad is a subroutine determining the discretization of the inner product on each component interval, or a dummy routine if **iq** $\neq 1$ (see below); specifically, **quad(n, x, w, i, ierr)** produces the abscissas $\mathbf{x}(r) = x_{r,i}$ and weights $\mathbf{w}(r) = w_{r,i}$, $r = 1, 2, \ldots, n$, of the n-point discretization of the inner product on the interval $[a_i, b_i]$ (cf. (4.13)); an error flag **ierr** is provided to signal the occurrence of an error condition in the quadrature process.

eps is the desired relative accuracy of the nonzero recursion coefficients; type real.

iq is an integer selecting a user-supplied quadrature routine **quad** if **iq** = 1 or the ORTHPOL routine **qgp** (see below) otherwise.

idelta is a nonzero integer, typically 1 or 2, inducing fast convergence in the case of special quadrature routines; the default value is **idelta** = 1.

irout is an integer selecting the routine for generating the recursion coefficients from the discrete inner product; specifically, **irout** = 1 selects the routine **sti**, and **irout** $\neq 1$ selects the routine **lancz**.

The logical variables **finl, finr** and the arrays **endl, endr, xfer, wfer** are input variables to the subroutine **qgp** and are used (and, hence, need to be properly dimensioned) only if **iq** $\neq 1$.

On return,

alpha, beta are arrays of dimension **n** holding the desired recursion coefficients **alpha**$(k) = \alpha_{k-1}$, **beta**$(k) = \beta_{k-1}$, $k = 1, 2, \ldots,$ **n**.

ncap is the integer N_0 yielding convergence.

kount is the number of iterations required to achieve convergence.

ierr is an error flag, equal to 0 on normal return, equal to -1 if **n** is not in the proper range, equal to i if there is an error condition in the discretization on the ith interval, and equal to **ncapm** if the discretized Stieltjes procedure does not converge within the discretization resolution specified by **ncapm**.

ie is an error flag inherited from the routine **sti** or **lancz** (whichever is used).

The arrays **be, x, w, xm, wm, p0, p1, p2** are used for working space, the last five having dimension **mc** $\times$ **ncapm** + **mp**.

A general-purpose quadrature routine, **qgp**, is provided for cases in which it may be difficult to develop special discretizations that take advantage of the structural properties of the weight function w at hand. The routine

assumes the same setup (4.9)–(4.14) used in **mcdis**, with *disjoint* intervals $[a_i, b_i]$, and provides for Q_i in (4.12) the Fejér quadrature rule, suitably transformed to the interval $[a_i, b_i]$, with the same number $N_i = N_0$ of points for each i. Recall that the N-point Fejér rule on the standard interval $[-1, 1]$ is the interpolatory quadrature formula

$$Q_N^F f = \sum_{r=1}^{N} w_r^F f(x_r^F), \tag{4.17}$$

where $x_r^F = \cos((2r-1)\pi/2N)$ are the Chebyshev points. The weights are all positive and can be computed explicitly in terms of trigonometric functions (cf., e.g., Gautschi [1967a]). The rule (4.17) is now applied to the integral in (4.11) by transforming the interval $[-1, 1]$ to $[a_i, b_i]$ via some monotone function ϕ_i (a linear function if $[a_i, b_i]$ is finite) and letting $f = uvw_i$:

$$\begin{aligned}\int_{a_i}^{b_i} u(t)v(t)w_i(t)\,dt &= \int_{-1}^{1} u(\phi_i(\tau))v(\phi_i(\tau))w_i(\phi_i(\tau))\phi_i'(\tau)\,d\tau \\ &\approx \sum_{r=1}^{N} w_r^F w_i\big(\phi_i(x_r^F)\big)\phi_i'(x_r^F)\cdot u\big(\phi_i(x_r^F)\big)v\big(\phi_i(x_r^F)\big).\end{aligned}$$

Thus, in effect, we take in (4.13)

$$x_{r,i} = \phi_i(x_r^F), \qquad w_{r,i} = w_r^F w_i\big(\phi_i(x_r^F)\big)\phi_i'(x_r^F), \qquad i = 1, 2, \ldots, m. \tag{4.18}$$

If the interval $[a_i, b_i]$ is half-infinite, say, of the form $[0, \infty]$, we use $\phi_i(t) = (1+t)/(1-t)$, and similarly for intervals of the form $[-\infty, b]$ and $[a, \infty]$. If $[a_i, b_i] = [-\infty, \infty]$, we use $\phi_i(t) = t/(1-t^2)$.

The routine **qgp** has the following calling sequence:

```
subroutine qgp(n, x, w, i, ierr, mc, finl, finr, endl, endr, xfer, wfer)
dimension x(n), w(n), endl(mc), endr(mc), xfer(*), wfer(*)
logical finl, finr
```

On entry,

n is the number of terms in the Fejér quadrature rule.

i indexes the interval $[a_i, b_i]$ for which the quadrature rule is desired; an interval that extends to $-\infty$ has to be indexed by 1, and one that extends to $+\infty$ by **mc**.

mc is the number of component intervals; type integer.

finl is a logical variable to be set **.true.** if the extreme left interval is finite and **.false.** otherwise.

finr is a logical variable to be set **.true.** if the extreme right interval is finite and **.false.** otherwise.

endl is an array of dimension **mc** containing the left endpoints of the component intervals; if the first of these extends to $-\infty$, **endl**(1) is not being used by the routine.

endr is an array of dimension **mc** containing the right endpoints of the component intervals; if the last of these extends to $+\infty$, **endr(mc)** is not being used by the routine.

xfer, wfer are working arrays holding the standard Fejér nodes and weights, respectively; they are dimensioned in the routine **mcdis**.

On return,

x, w are arrays of dimension **n** holding the abscissas and weights (4.18) of the discretized inner product for the ith component interval.

ierr has the integer value 0.

The routine calls on the subroutines **fejer**, **symtr** and **tr**, which are appended to the routine **qgp** in Section 4 of the package. The first generates the Fejér quadrature rule; the others perform variable transformations. The user has to provide his or her own function routine **wf(x, i)** to calculate the weight function $w_i(x)$ on the ith component interval.

Example 4.2 Chebyshev weight plus a constant: $w^c(t) = (1 - t^2)^{-1/2} + c$, $c > 0$, $-1 < t < 1$.

It would be difficult here to find a single quadrature rule for discretizing the inner product and to obtain fast convergence. However, using in (4.9) $m = 2$, $[a_1, b_1] = [a_2, b_2] = [-1, 1]$, and $w_1(t) = (1 - t^2)^{-1/2}$, $w_2(t) = c$ in (4.11), and taking for Q_1 the Gauss–Chebyshev, and for Q_2 the Gauss–Legendre n-point rule (the latter multiplied by c), yield convergence to $\alpha_k(w^c)$, $\beta_k(w^c)$, $k = 0, 1, \ldots, n - 1$, in one iteration (provided δ is set equal to 2)! Actually, we need $N_0 = n + 1$, in order to test for convergence; cf. (4.15). The driver **test4** implements this technique and calculates the first $n = 80$ beta-coefficients to a relative accuracy of $5000 \times \epsilon^s$ for $c = 1, 10, 100$. (All α_k are zero.) Attached to the driver is the quadrature routine **qchle** used in this example. It, in turn, calls for the Gauss quadrature routine **gauss**, to be described in Section 6. Anticipating convergence after one iteration, we put **ncapm** = 81.

The weight function of Example 4.2 provides a continuous link between the Chebyshev polynomials ($c = 0$) and the Legendre polynomials ($c = \infty$); the recursion coefficients $\beta_k(w^c)$ indeed converge (except for $k = 0$) to those of the Legendre polynomials, as $c \to \infty$.

Selected results of **test4** (where **irout** in **mcdis** can be arbitrary) are shown in Table VI. The output variable **kount** is 1 in each case, confirming convergence after one iteration. The coefficients $\beta_0(w^c)$ are easily seen to be $\pi + 2c$.

Example 4.3 Jacobi weight with one mass point at the left endpoint: $w^{(\alpha,\beta)}(t; y) = [\mu_0^{(\alpha,\beta)}]^{-1}(1 - t)^\alpha(1 + t)^\beta + y\delta(t + 1)$ on $(-1, 1)$, $\mu_0^{(\alpha,\beta)} = 2^{\alpha+\beta+1}\Gamma(\alpha + 1)\Gamma(\beta + 1)/\Gamma(\alpha + \beta + 2)$, $\alpha > -1$, $\beta > -1$, $y > 0$.

Table VI. Selected Recursion Coefficients $\beta_k(w^c)$ for $c = 1, 10, 100$

k	$\beta_k(w^1)$	$\beta_k(w^{10})$	$\beta_k(w^{100})$
0	5.1415926540	23.14159265	203.1415927
1	0.4351692451	0.3559592080	0.3359108398
5	0.2510395775	0.2535184776	0.2528129500
12	0.2500610870	0.2504824840	0.2505324193
25	0.2500060034	0.2500682357	0.2501336338
51	0.2500006590	0.2500082010	0.2500326887
79	0.2500001724	0.2500021136	0.2500127264

The recursion coefficients α_k, β_k are known explicitly (see Chihara [1985, Eqs. 6.23, 3.5][1]) and can be expressed, with some effort, in terms of the recursion coefficients α_k^J, β_k^J for the Jacobi weight $w^{(\alpha,\beta)}(\cdot) = w^{(\alpha,\beta)}(\cdot; 0)$. The formulas are

$$\alpha_0 = \frac{\alpha_0^J - y}{1+y}, \qquad \beta_0 = \beta_0^J + y,$$

$$\alpha_k = \alpha_k^J + \frac{2k(\alpha+k)}{(\alpha+\beta+2k)(\alpha+\beta+2k+1)}(c_k - 1) + \frac{2(\beta+k+1)(\alpha+\beta+k+1)}{(\alpha+\beta+2k+1)(\alpha+\beta+2k+2)}\left(\frac{1}{c_k} - 1\right),$$

$$\beta_k = \frac{c_k}{c_{k-1}}\beta_k^J, \qquad k = 1, 2, 3, \ldots, \tag{4.19}$$

where

$$c_0 = 1 + y, \qquad c_k = \frac{1 + \dfrac{(\beta+k+1)(\alpha+\beta+k+1)}{k(\alpha+k)} y d_k}{1 + y d_k}, \qquad k = 1, 2, \ldots, \tag{4.20}$$

and

$$d_1 = 1,$$
$$d_k = \frac{(\beta+k)(\alpha+\beta+k)}{(\alpha+k-1)(k-1)} d_{k-1}, \qquad k = 2, 3, \ldots. \tag{4.21}$$

Again, it is straightforward with **mcdis** to get exact results (modulo rounding) after one iteration, by using the Gauss–Jacobi quadrature rule (see **gauss** in Section 6) to discretize the continuous part of the measure. The driver **test5** generates in this manner the first $n = 40$ recursion coefficients α_k, β_k, $k = 0, 1, \ldots, n-1$, to a relative accuracy of $5000 \times \epsilon^s$, for $y = \frac{1}{2}, 1, 2,$

[1] In Chihara [1985] the interval is taken to be $[0, 2]$, rather than $[-1, 1]$. There is a typographical error in the first formula of (6.23), which should have the numerator $2\beta + 2$ instead of $2\beta + 1$.

4, and 8. For each $\alpha = -.8(.2)1.$ and $\beta = -.8(.2)1.$, it computes the maximum relative errors (absolute error, if $\alpha_k \approx 0$) of the α_k, β_k by comparing them with the exact coefficients. These have been computed in double precision by a straightforward implementation of formulas (4.19)–(4.21).

As expected, the output of **test5** reveals convergence after one iteration, the variable **kount** having consistently the value 1. The maximum relative error in the α_k is found to lie generally between 2×10^{-8} and 3×10^{-8}, the one in the β_k between 7.5×10^{-12} and 8×10^{-12}; they are attained for k at or near 39. The discrepancy between the errors in the α_k and those in the β_k is due to the α_k being considerably smaller than the β_k, by 3–4 orders of magnitude. Replacing the routine **sti** in **mcdis** by **lancz** yields very much the same error picture.

It is interesting to note that the addition of a second mass point at the other endpoint makes an analytic determination of the recursion coefficients intractable (cf. Chihara [1985, p. 713]). Numerically, however, it makes no difference whether there are two or more mass points and whether they are located inside, outside, or on the boundary of the support interval. It was observed, however, that if at least one mass point is located outside the interval $[-1, 1]$ the procedure **sti** used in **mcdis** becomes severely unstable[2] and *must* be replaced by **lancz**.

Example 4.4 Logistic density function: $w(t) = e^{-t}/(1 + e^{-t})^2$ on $(-\infty, \infty)$. In this example we illustrate a slight variation of the discretization procedure (4.9)–(4.13), which ends up with a discrete inner product of the same type as in (4.13) (and thus implementable by the routine **mcdis**), but derived in a somewhat different manner. The idea is to integrate functions with respect to the density w by splitting the integral into two parts, one from $-\infty$ to 0 and the other from 0 to ∞, changing variables in the first part, and thus obtaining

$$\int_{-\infty}^{\infty} f(t)w(t)\,dt = \int_0^{\infty} f(-t)\frac{e^{-t}}{(1+e^{-t})^2}\,dt + \int_0^{\infty} f(t)\frac{e^{-t}}{(1+e^{-t})^2}\,dt. \quad (4.22)$$

Since $(1 + e^{-t})^{-2}$ quickly tends to 1 as $t \to \infty$, a natural discretization of both integrals is provided by the Gauss–Laguerre quadrature rule applied to the product $f(\pm t) \cdot (1 + e^{-t})^{-2}$. This amounts to taking, in (4.13), $m = 2$ and

$$x_{r,1} = -x_r^L, \qquad x_{r,2} = x_r^L; \qquad w_{r,1} = w_{r,2} = \frac{w_r^L}{(1+e^{-x_r^L})^2},$$
$$r = 1, 2, \ldots, N,$$

where x_r^L, w_r^L are the Gauss–Laguerre N-point quadrature nodes and weights.

[2]This has also been observed in a similar example [Gautschi 1982a, Example 4.8], but was incorrectly attributed to a phenomenon of ill-conditioning. Indeed, the statement made at the end of Example 4.8 can now be retracted: Stable methods *do* exist, namely, the method embodied by the routine **mcdis** in combination with **lancz**.

Table VII. Selected Output from **test6**

k	β_k	err α_k	err β_k
0	1.00000000000000000000000	4.572(−13)	1.918(−13)
1	3.28986813369645287294483O	1.682(−13)	5.641(−13)
6	89.44760352315950188817832	2.187(−12)	2.190(−12)
15	555.7827839879296775066697	1.732(−13)	2.915(−12)
26	1668.580222268668421827788	3.772(−12)	4.112(−12)
39	3753.534025194898387722354	2.482(−11)	4.533(−12)

The driver **test6** incorporates this discretization into the routines **mcdis** and **dmcdis**, runs them for $n = 40$ with error tolerances $5000 \times \epsilon^s$ and $1000 \times \epsilon^d$, respectively, and prints the absolute errors in the α's ($\alpha_k = 0$, in theory) and the relative errors in the β's. (We used the default value $\delta = 1$.) Also printed are the number of iterations #it (= **kount**) in (4.15) and the corresponding final value N_0^f(= **ncap**). In single precision we found that #it = 1, $N_0^f = 81$, and in double precision, #it = 5, $N_0^f = 281$. Both routines returned with the error flags equal to 0, indicating a normal course of events. A few selected double-precision values[3] of the coefficients β_k along with absolute errors in the α's and relative errors in the β's are shown in Table VII. The results are essentially the same no matter whether **sti** or **lancz** is used in **mcdis**. The maximum errors observed are 2.482×10^{-11} for the α's and 4.939×10^{-12} for the β's, which are well within the single-precision tolerance $\epsilon = 5000 \times \epsilon^s$.

On computers with limited exponent range, convergence difficulties may arise, both with **sti** and **lancz**, owing to underflow in many of the Laguerre quadrature weights. This seems to perturb the problem significantly enough to prevent the discretization procedure from converging.

Example 4.5 Half-range Hermite measure: $w(t) = e^{-t^2}$ on $(0, \infty)$.

This is an example of a measure for which there do not seem to exist natural discretizations other than those based on composite quadrature rules. Therefore, we applied our general-purpose routine **qgp** (and its double-precision companion **dqgp**), using, after some experimentation, the partition $[0, \infty] = [0, 3] \cup [3, 6] \cup [6, 9] \cup [9, \infty]$. The driver **test7** implements this, with $n = 40$ and an error tolerance $50 \times \epsilon^s$ in single precision, and $1000 \times \epsilon^d$ in double precision.

The single-precision routine **mcdis** (using the default value $\delta = 1$) converged after one iteration, returning **ncap** = 81, whereas the double-precision routine **dmcdis** took four iterations to converge and returned **ncapd** = 201. Selected results (where err α_k and err β_k both denote relative errors) are shown in Table VIII. The maximum error err α_k occurred at $k = 10$ and had the value 1.038×10^{-12}, whereas $\max_k$ err $\beta_k = 3.180 \times 10^{-13}$ is attained at $k = 0$. The latter is within the error tolerance ϵ, the former only slightly

[3] *Note added in proof:* Alphonse Magnus, in an email message, May 5, 1993, kindly pointed out to the author that the β-coefficients are known explicitly: $\beta_k = k^4\pi^2/(4k^2 - 1)$, $k = 1, 2, \ldots$.

Table VIII. Selected Output from **test7**

k	α_k and err α_k	β_k and err β_k
0	0.56418958354775628694807 95	0.88622692545275801364908 37
	1.096(−13)	3.180(−13)
1	0.98842539284680028548706 34	0 18169011381620932846223 25
	1.514(−13)	7.741(−14)
6	2.08062033640083322481762 2	1.00234785101101084222453 8
	1.328(−13)	5.801(−14)
15	3.21427063607112822744891 4	2.50092791713370266995432 1
	2.402(−14)	8.186(−14)
26	4.20304857887200195266027 7	4.33386790122995044360443 0
	1.415(−13)	7.878(−14)
39	5.13153288689429651931969 2	6.50035623770713293803515 5
	6.712(−13)	1.820(−14)

larger. Comparison of the double-precision results with Table I on the microfiche supplement to Galant [1969] revealed agreement to all 20 decimal digits given there, for all k in the range $0 \le k \le 19$. Interestingly, the routine **sti** in **mcdis** did consistently better than **lancz** on the β's, by a factor as large as 235 (for $k = 33$), and is comparable with **lancz** (sometimes better, sometimes worse) on the α's.

Without composition, that is, using **mc** = 1 in **mcdis**, it takes 8 iterations ($N_0^f = 521$) in single precision and 10 iterations ($N_0^f = 761$) in double precision to satisfy the much weaker error tolerances $\epsilon = \frac{1}{2}\,10^{-6}$ and $\epsilon^d = \frac{1}{2}\,10^{-12}$, respectively. All single-precision results, however, turn out to be accurate to about 12 decimal places. (This is because of the relatively large final increment $\Delta_8 = 2n = 80$ in N_0 (cf. (4.16)) that forces convergence.)

4.4 Discretized Modified Chebyshev Algorithm

The whole apparatus of discretization (cf. (4.9)–(4.14)) can also be employed in connection with the modified Chebyshev algorithm (cf. Section 3), if one discretizes modified moments rather than inner products. Thus, one approximates (cf. (4.14), (4.16))

$$\nu_k(d\lambda) \approx \nu_k(d\lambda_{mN_0^{[s]}+p}) \tag{4.23}$$

and iterates the modified Chebyshev algorithm with $s = 0, 1, 2, \ldots$ until the convergence criterion (4.15) is satisfied. (It would be unwise to test convergence on the modified moments, for reasons explained in Gautschi [1982a, sect. 2.5].) This is implemented in the routine **mccheb**, whose calling sequence is as follows:

mccheb(n, ncapm, mc, mp, xp, yp, quad, eps, iq, idelta, finl, finr, endl, endr, xfer, wfer, a, b, fnu, alpha, beta, ncap, kount, ierr, be, x, w, xm, wm, s, s0, s1, s2)

Its input and output parameters have the same meaning as in the routine **mcdis**. In addition, the arrays **a**, **b** of dimension $2 \times \mathbf{n} - 1$ are to be supplied

with the recursion coefficients $\mathbf{a}(k) = a_{k-1}, \mathbf{b}(k) = b_{k-1}, k = 1, 2, \ldots, 2 \times \mathbf{n} - 1$, defining the modified moments. The arrays **be, x, w, xm, wm, s, s0, s1, s2** are used for working space. The double-precision version of the routine has the name **dmcheb**.

The discretized modified Chebyshev algorithm must be expected to behave similarly as its close relative, the modified Chebyshev algorithm. In particular, if the latter suffers from ill-conditioning, so does the former.

Example 4.6 (Example 3.1, revisited).

We recompute the $n = 40$ first recursion coefficients α_k, β_k of Example 3.1 to an accuracy of $100 \times \epsilon^s$ in single precision, using the routine **mccheb** instead of the routine **cheb**. For the discretization of the modified moments, we employed the Gauss–Chebyshev quadrature rule:

$$\int_{-1}^{1} f(t)(1 - \omega^2 t^2)^{-1/2}(1 - t^2)^{-1/2}\, dt \approx \frac{\pi}{N} \sum_{r=1}^{N} f(x_r)(1 - \omega^2 x_r^2)^{-1/2}, \tag{4.24}$$

where $x_r = \cos((2r - 1)\pi/2N)$ are the Chebyshev points. This is accomplished by the driver **test8**. The results of this test (shown in the package) agree to all 10 decimal places with those of **test1**. The routine **mccheb** converged in one iteration, with **ncap** = 81, for $\omega^2 = .1, .3, .5, .7, .9$; in 4 iterations, with **ncap** = 201, for $\omega^2 = .99$; and in 8 iterations, with **ncap** = 521, for $\omega^2 = .999$. A double-precision version of **test8** was also run with $\epsilon = \frac{1}{2} \times 10^{-20}$ (not shown in the package) and produced correct results to 20 decimals in one iteration (**ncap** = 81) for $\omega^2 = .1, .3, .5, .7$; in 3 iterations (**ncap** = 161) for $\omega^2 = .9$; in 6 iterations (**ncap** = 361) for $\omega^2 = .99$; and in 11 iterations (**ncap** = 921) for $\omega^2 = .999$.

5. MODIFICATION ALGORITHMS

Given a positive measure $d\lambda(t)$ supported on the real line, and two polynomials $u(t) = \pm \prod_{\rho=1}^{r}(t - u_\rho)$, $v(t) = \prod_{\sigma=1}^{s}(t - v_\sigma)$ whose ratio is finite on the support of $d\lambda$, we may ask for the recursion coefficients $\hat{\alpha}_k = \alpha_k(d\hat{\lambda})$, $\hat{\beta}_k = \beta_k(d\hat{\lambda})$ of the modified measure

$$d\hat{\lambda}(t) = \frac{u(t)}{v(t)}\, d\lambda(t), \qquad t \in \operatorname{supp}(d\lambda), \tag{5.1}$$

assuming known the recursion coefficients $\alpha_k = \alpha_k(d\lambda)$, $\beta_k = \beta_k(d\lambda)$ of the given measure. Methods that accomplish the passage from the α's and β's to the $\hat{\alpha}$'s and $\hat{\beta}$'s are called *modification algorithms*. The simplest case $s = 0$ (i.e., $v(t) \equiv 1$) and u positive on $\operatorname{supp}(d\lambda)$ has already been considered by Christoffel [1858], who represented the polynomial $u(\cdot)\hat{\pi}_k(\cdot) = u(\cdot)\pi_k(\cdot; d\hat{\lambda})$ in determinantal form in terms of the polynomials $\pi_j(\cdot) = \pi_j(\cdot; d\lambda)$, $j = k, k + 1, \ldots, k + r$. This is now known as *Christoffel's theorem*. Christoffel, however, did not address the problem of how to generate the new coefficients $\hat{\alpha}_k$, $\hat{\beta}_k$ in terms of the old ones. For the more general modification (5.1), Christoffel's theorem has been generalized by Uvarov [1959; 1969]. The coefficient prob-

lem stated above, in this general case, has been treated in Gautschi [1982b], and previously by Galant [1971] in the special case $v(t) \equiv 1$.

The passage from $d\lambda$ to $d\hat{\lambda}$ can be carried out in a sequence of elementary steps involving real linear factors $t - x$ or real quadratic factors $(t - x)^2 + y^2$, either in $u(t)$ or in $v(t)$. The corresponding elementary steps in the passage from the α's and β's to the $\hat{\alpha}$'s and $\hat{\beta}$'s can all be performed by means of certain nonlinear recurrences. Some of these, however, when divisions of the measure $d\lambda$ are involved, are liable to instabilities. An alternative method can then be used, which appeals to the modified Chebyshev algorithm supplied with appropriate modified moments. These latter are of independent interest and find application, for example, in evaluating the kernel in the contour integral representation of the Gauss quadrature remainder term.

5.1 Nonlinear Recurrence Algorithms

The routine that carries out the elementary modification steps is called **chri** and has the calling sequence

chri(n,iopt,a,b,x,y,hr,hi,alpha,beta,ierr).

On entry,

n is the number of recursion coefficients desired; type integer.

iopt is an integer identifying the type of modification as follows:
(1) $d\hat{\lambda}(t) = (t - x)\, d\lambda(t)$.
(2) $d\hat{\lambda}(t) = ((t - x)^2 + y^2)\, d\lambda(t)$, $y > 0$.
(3) $d\hat{\lambda}(t) = (t^2 + y^2)\, d\lambda(t)$ with $d\lambda(t)$ and $\text{supp}(d\lambda)$ assumed symmetric with respect to the origin and $y > 0$.
(4) $d\hat{\lambda}(t) = d\lambda(t)/(t - x)$.
(5) $d\hat{\lambda}(t) = d\lambda(t)/((t - x)^2 + y^2)$, $y > 0$.
(6) $d\hat{\lambda}(t) = d\lambda(t)/(t^2 + y^2)$ with $d\lambda(t)$ and $\text{supp}(d\lambda)$ assumed symmetric with respect to the origin and $y > 0$.
(7) $d\hat{\lambda}(t) = (t - x)^2 d\lambda(t)$.

a, b are arrays of dimension **n** + 1 holding the recursion coefficients $\mathbf{a}(k) = \alpha_{k-1}(d\lambda)$, $\mathbf{b}(k) = \beta_{k-1}(d\lambda)$, $k = 1, 2, \ldots, \mathbf{n} + 1$.

x, y are real parameters defining the linear and quadratic factors (or divisors) of $d\lambda$.

hr, hi are the real and imaginary part, respectively, of $\int_{\mathbb{R}} d\lambda(t)/(z - t)$, where $z = x + iy$; the parameter **hr** is used only if **iopt** = 4 or 5, and the parameter **hi** only if **iopt** = 5 or 6.

On return,

alpha, beta are arrays of dimension **n** containing the desired recursion coefficients $\mathbf{alpha}(k) = \alpha_{k-1}(d\hat{\lambda})$, $\mathbf{beta}(k) = \beta_{k-1}(d\hat{\lambda})$, $k = 1, 2, \ldots, \mathbf{n}$.

ierr is an error flag, equal to 0 on normal return, equal to 1 if $\mathbf{n} \leq 1$ (the routine assumes that **n** is larger than or equal to 2), and equal to 2 if the integer **iopt** is inadmissible.

It should be noted that in the cases **iopt** = 1 and **iopt** = 4, the modified measure $d\hat{\lambda}$ is positive (negative) definite if x is to the left (right) of the support of $d\lambda$, but indefinite otherwise. Nevertheless, it is permissible to have x inside the support of $d\lambda$ (or inside its convex hull), provided the resulting measure $d\hat{\lambda}$ is still quasi-definite (cf. Gautschi [1982b]).

For **iopt** = $1, 2, \ldots, 6$, the methods used in **chri** are straightforward implementations of the nonlinear recurrence algorithms, respectively, in Eqs. (3.7), (4.7), (4.8), (5.1), (5.8), and (5.9) of Gautschi [1982b]. The only minor modification required concerns $\hat{\beta}_0 = \beta_0(d\hat{\lambda})$. In Gautschi [1982b] this constant was taken to be 0, whereas here it is defined to be $\hat{\beta}_0 = \int_{\mathbb{R}} d\hat{\lambda}(t)$. Thus, for example, if **iopt** = 2,

$$\hat{\beta}_0 = \int_{\mathbb{R}} \left((t-x)^2 + y^2\right) d\lambda(t) = \int_{\mathbb{R}} \left((t-\alpha_0+\alpha_0-x)^2 + y^2\right) d\lambda(t)$$

$$= \int_{\mathbb{R}} \left((t-\alpha_0)^2 + (\alpha_0-x)^2 + y^2\right) d\lambda(t),$$

since $\int_{\mathbb{R}}(t-\alpha_0)\,d\lambda(t) = \int_{\mathbb{R}} \pi_1(t)\,d\lambda(t) = 0$. Furthermore (cf. (4.1)),

$$\int_{\mathbb{R}} (t-\alpha_0)^2\, d\lambda(t) = \beta_0\,\beta_1,$$

so that the formula to be used for $\hat{\beta}_0$ is

$$\hat{\beta}_0 = \beta_0\left(\beta_1 + (\alpha_0 - x)^2 + y^2\right) \qquad (\textbf{iopt} = 2).$$

Similar calculations need to be made in the other cases.

The case **iopt** = 7 incorporates a QR step with shift x, following Kautsky and Golub [1983], and uses an adaptation of the algorithm in Wilkinson [1965, Eq. 67.11, p. 567], to carry out the QR step. The most significant modification made is the replacement of the test $c \neq 0$ by $|c| > \epsilon$, where $\epsilon = 5 \times \epsilon^s$ is a quantity close to, but slightly larger than, the machine precision. (Without this modification, the algorithm could fail.)

The methods used in **chri** are believed to be quite stable when the measure $d\lambda$ is modified multiplicatively (**iopt** = 1, 2, 3, and 7). When divisions are involved (**iopt** = 4, 5, and 6), however, the algorithms rapidly become unstable as the point $z = x + iy \in \mathbb{C}$ moves away from the support interval of $d\lambda$. (The reason for this instability is not well understood at present; see, however, Galant [1992].) For such cases there is an alternative routine, **gchri** (see Section 5.2), that can be used.

Example 5.1 Checking the results (for $\sigma = \frac{1}{2}$) of **test2**.

We apply **chri** (and the corresponding double-precision routine **dchri**) with **iopt** = 1, $x = 0$, to $d\lambda_\sigma(t) = t^\sigma \ln(1/t)$ on $(0, 1)$ with $\sigma = -\frac{1}{2}$, to recompute the results of **test2** for $\sigma = \frac{1}{2}$. This can be done by a minor modification, named **test9**, of **test2**. Selected results from it, showing the relative discrepancies between the single-precision values α_k, β_k (resp. double-precision values α_k^d, β_k^d), computed by the modified Chebyshev algorithm and the modification algorithm, are shown in Table IX (cf. Table III). The maximum errors occur consistently for the last value of $k(= 98)$.

Table IX. Comparison between Modified Chebyshev Algorithm and Modification Algorithm in Example 5.1 (cf. Example 3.2)

σ	k	err α_k	err β_k	err α_k^d	err β_k^d
.5	0	7.895(−14)	4.796(−14)	2.805(−28)	7.952(−28)
	12	3.280(−12)	6.195(−12)	8.958(−26)	1.731(−25)
	24	7.648(−12)	1.478(−11)	2.065(−25)	3.985(−25)
	48	2.076(−11)	4.088(−11)	5.683(−25)	1.121(−24)
	98	6.042(−11)	1.201(−10)	1.504(−24)	2.987(−24)

Example 5.2 Induced orthogonal polynomials.

Given an orthogonal polynomial $\pi_m(\cdot; d\lambda)$ of fixed degree $m \geq 1$, the sequence of orthogonal polynomials $\hat{\pi}_{k,m}(\cdot) = \pi_k(\cdot; \pi_m^2\, d\lambda)$, $k = 0, 1, 2, \ldots$, has been termed *induced orthogonal polynomials* in Gautschi and Li [1993]. Since their measure $d\hat{\lambda}_m$ modifies the measure $d\lambda$ by a product of quadratic factors,

$$d\hat{\lambda}_m(t) = \prod_{\mu=1}^{m} (t - x_\mu)^2 \cdot d\lambda(t), \tag{5.2}$$

where x_μ are the zeros of π_m, we can apply the routine **chri** (with **iopt** = 7) m times to compute the n recursion coefficients $\hat{\alpha}_{k,m} = \alpha_k(d\hat{\lambda}_m)$, $\hat{\beta}_{k,m} = \beta_k(d\hat{\lambda}_m)$, $k = 0, 1, \ldots, n-1$, from the $n + m$ coefficients $\alpha_k = \alpha_k(d\lambda)$, $\beta_k = \beta_k(d\lambda)$, $k = 0, 1, \ldots, n - 1 + m$. The subroutines **indp** and **dindp** in the driver **test10** implement this procedure in single (resp., double) precision. The driver itself uses them to compute the first $n = 20$ recursion coefficients of the induced Legendre polynomials with $m = 0, 1, \ldots, 11$. It also computes the maximum absolute errors in the $\hat{\alpha}$'s ($\hat{\alpha}_{k,m} = 0$ for all m) and the maximum relative errors in the $\hat{\beta}$'s by comparing single-precision with double-precision results.

An excerpt of the output of **test10** is shown in Table X. It already suggests a high degree of stability of the procedure employed by **indp**. This is reinforced by an additional test (not shown in the package) generating $n = 320$ recursion coefficients $\hat{\alpha}_{k,m}$, $\hat{\beta}_{k,m}$, $0 \leq k \leq 319$, for $m = 40$, 80, 160, 320 and $d\lambda$ being the Legendre, the first-kind Chebyshev, the Laguerre, and the Hermite measure. Table XI shows the maximum absolute error in the $\hat{\alpha}_{k,m}$, $0 \leq k \leq 319$ (relative error in the Laguerre case), and the maximum relative error in the $\hat{\beta}_{k,m}$, $0 \leq k \leq 319$.

5.2 Methods Based on the Modified Chebyshev Algorithm

As was noted earlier, the procedure **chri** becomes unstable for modified measures involving division of $d\lambda(t)$ by $t - x$ or $(t - x)^2 + y^2$ as $z = x + iy \in \mathbb{C}$ moves away from the "support interval" of $d\lambda$, that is, from the smallest interval containing the support of $d\lambda$. We now develop a procedure that works better the further away z is from that interval.

The idea is to use modified moments of $d\hat{\lambda}$ relative to the polynomials $\pi_k(\cdot; d\lambda)$ to generate the desired recursion coefficients $\hat{\alpha}_k$, $\hat{\beta}_k$ via the modified Chebyshev algorithm (cf. Section 3). The modified moments in question

Table X. Induced Legendre Polynomials

k	$m = 0, \hat{\beta}_{k,m}$	$m = 2, \hat{\beta}_{k,m}$	$m = 6, \hat{\beta}_{k,m}$	$m = 11, \hat{\beta}_{k,m}$
0	2.0000000000	0.1777777778	0.0007380787	0.0000007329
1	0.3333333333	0.5238095238	0.5030303030	0.5009523810
6	0.2517482517	0.1650550769	0.2947959861	0.2509913424
12	0.2504347826	0.2467060415	0.2521022519	0.1111727541
19	0.2501732502	0.2214990335	0.2274818789	0.2509466619
err $\hat{\alpha}$	0.000(0)	1.350(−13)	9.450(−13)	1.357(−12)
err $\hat{\beta}$	1.737(−14)	2.032(−13)	2.055(−12)	3.748(−12)

Table XI. Accuracy of the Recursion Coefficients for Some Classical Induced Polynomials

	Legendre		Chebyshev		Laguerre		Hermite	
m	err $\hat{\alpha}$	err $\hat{\beta}$	err $\hat{\alpha}$	err $\hat{\beta}$	err $\hat{\alpha}$	err $\hat{\beta}$	err $\hat{\alpha}$	err $\hat{\beta}$
40	3.4(−11)	1.5(−10)	1.9(−9)	7.9(−10)	3.0(−10)	6.0(−10)	1.8(−9)	2.7(−10)
80	1.4(−10)	5.4(−10)	2.1(−9)	2.2(−9)	5.8(−10)	9.2(−10)	7.9(−9)	9.2(−10)
160	1.5(−9)	5.1(−9)	9.5(−9)	1.1(−8)	7.8(−10)	1.4(−9)	1.1(−8)	6.8(−10)
320	3.3(−9)	2.1(−8)	9.6(−9)	2.1(−8)	3.9(−9)	7.2(−9)	2.5(−8)	1.1(−9)

are

$$\nu_k = \nu_k(x; d\lambda) = \int_{\mathbb{R}} \frac{\pi_k(t; d\lambda)}{t - x}\, d\lambda(t), \qquad k = 0, 1, 2, \ldots, \tag{5.3}$$

for linear divisors and

$$\nu_k = \nu_k(x, y; d\lambda) = \int_{\mathbb{R}} \frac{\pi_k(t; d\lambda)}{(t - x)^2 + y^2}\, d\lambda(t), \qquad k = 0, 1, 2, \ldots, \tag{5.4}$$

for quadratic divisors. Both can be expressed in terms of the integrals

$$\rho_k = \rho_k(z; d\lambda) = \int_{\mathbb{R}} \frac{\pi_k(t; d\lambda)}{z - t}\, d\lambda(t), \qquad z \in \mathbb{C} \setminus \operatorname{supp}(d\lambda), \quad k = 0, 1, 2, \ldots, \tag{5.5}$$

the first by means of

$$\nu_k(x; d\lambda) = -\rho_k(z; d\lambda), \qquad z = x, \tag{5.6}$$

and the others by means of

$$\nu_k(x, y; d\lambda) = -\frac{\operatorname{Im} \rho_k(z; d\lambda)}{\operatorname{Im} z}, \qquad z = x + iy. \tag{5.7}$$

The point to observe is that $\{\rho_k(z; d\lambda)\}$ is a minimal solution of the basic recurrence relation (1.3) for the orthogonal polynomials $\{\pi_k(\cdot; d\lambda)\}$ (cf. Gautschi [1981]). The quantities $\rho_k(z; d\lambda)$, $k = 0, 1, \ldots, n$, therefore, can be computed accurately by a backward recurrence algorithm [Gautschi 1981, sect. 5], which, for $\nu > n$, produces approximations $\rho_k^{[\nu]}(z; d\lambda)$ converging to $\rho_k(z; d\lambda)$ when $\nu \to \infty$, for any fixed k,

$$\rho_k^{[\nu]}(z; d\lambda) \to \rho_k(z; d\lambda), \qquad \nu \to \infty. \tag{5.8}$$

The procedure is implemented in the routine

knum(n, nu0, numax, z, eps, a, b, rho, nu, ierr, rold),

which computes $\rho_k(z; d\lambda)$ for $k = 0, 1, \ldots,$ **n** to a relative precision **eps**. The results are stored as $\mathbf{rho}(k) = \rho_{k-1}(z; d\lambda)$, $k = 1, 2, \ldots,$ **n** + 1, in the complex array **rho** of dimension $n + 1$. The user has to provide a starting index **nu0** $= \nu_0 > n$ for the backward recursion, which the routine then increments by units of 5 until convergence to within **eps** is achieved. If the requested accuracy **eps** cannot be realized for some $\nu \leq$ **numax**, the routine exits with **ierr** = **numax**. Likewise, if $\nu_0 >$ **numax**, the routine exits immediately, with the error flag **ierr** set equal to **nu0**. Otherwise, the value of ν for which convergence is obtained is returned as output variable **nu**. The arrays **a**, **b** of dimension **numax** are to hold the recursion coefficients $\mathbf{a}(k) = \alpha_{k-1}(d\lambda)$, $\mathbf{b}(k) = \beta_{k-1}(d\lambda)$, $k = 1, 2, \ldots,$ **numax**, for the given measure $d\lambda$. The complex array **rold** of dimension $n + 1$ is used for working space. In the interest of rapid convergence, the routine should be provided with a realistic estimate of ν_0. For classical measures, such estimates are known (cf. Gautschi [1981, sect. 5]) and are implemented here by the function routines

nu0jac(n, z, eps), nu0lag(n, z, al, eps), nu0her(n, z, eps).

The first is for Jacobi measures, the second is for generalized Laguerre measures with parameter **al** $= \alpha > -1$, and the last is for the Hermite measure. Note that ν_0 for Jacobi measures does not depend on the weight parameters α, β, in contrast to ν_0 for the generalized Laguerre measure.

The name **knum** comes from the fact that $\rho_n(z; d\lambda)$ in (5.5) is the numerator in the kernel

$$K_n(z; d\lambda) = \frac{\rho_n(z; d\lambda)}{\pi_n(z; d\lambda)} \tag{5.9}$$

of the remainder term of the n-point Gaussian quadrature rule for analytic functions (cf., e.g., Gautschi and Varga [1983]). For the sequence of kernels $K_0, K_1, \ldots, K_n$, we have the following routine:

```
subroutine kern(n, nu0, numax, z, eps, a, b, ker, nu, ierr, rold)
complex z, ker, rold, p0, p, pm1
dimension a(numax), b(numax), ker(*), rold(*)
call knum(n, nu0, numax, z, eps, a, b, ker, nu, ierr, rold)
if(ierr.ne.0) return
p0 = (0., 0.)
p = (1., 0.)
do 10 k = 1, n
  pm1 = p0
  p0 = p
  p = (z - a(k))*p0 - b(k)*pm1
  ker(k + 1) = ker(k + 1) / p
10 continue
return
end
```

The meaning of the input and output parameters is the same as in **knum**. The double-precision version of the routine is named **dkern**.

All of the ingredients are now in place to describe the workings of **gchri**, the alternative routine to **chri** when the latter is unstable. First, the routine **knum** is used to produce the first $2n$ modified moments $\nu_k(x; d\lambda)$ (resp., $\nu_k(x, y; d\lambda)$), $k = 0, 1, \ldots, 2n - 1$. These are then supplied to the routine **cheb** along with the recursion coefficients $\alpha_k(d\lambda)$, $\beta_k(d\lambda)$ (needed anyhow for the computation of the ν_k), which produces the desired coefficients $\alpha_k(d\hat{\lambda})$, $\beta_k(d\hat{\lambda})$, $k = 0, 1, \ldots, n - 1$. The routine has the following calling sequence:

gchri(n, iopt, nu0, numax, eps, a, b, x, y, alpha, beta, nu, ierr, ierrc, fnu, rho, rold, s, s0, s1, s2).

On entry,

n is the number of recursion coefficients desired; type integer.

iopt is an integer identifying the type of modification as follows:
(1) $d\hat{\lambda}(t) = d\lambda(t)/(t - x)$, where x is assumed to be outside of the smallest interval containing $\text{supp}(d\lambda)$.
(2) $d\hat{\lambda}(t) = d\lambda(t)/((t - x)^2 + y^2)$, $y > 0$.

nu0 is an integer $\nu_0 \geq 2n$ estimating the starting index for the backward recursion to compute the modified moments; if no other choices are available, take **nu0** $= 3 \times$ **n**.

numax is an integer used to terminate backward recursion in case of nonconvergence; a conservative choice is **numax** = 500.

eps is a relative error tolerance; type real.

a, b are arrays of dimension **numax** to be supplied with the recursion coefficients $\mathbf{a}(k) = \alpha_{k-1}(d\lambda)$, $\mathbf{b}(k) = \beta_{k-1}(d\lambda)$, $k = 1, 2, \ldots,$ **numax**.

x, y are real parameters defining the linear and quadratic divisors of $d\lambda$.

On return,

alpha, beta are arrays of dimension **n** containing the desired recursion coefficients **alpha**$(k) = \hat{\alpha}_{k-1}$, **beta**$(k) = \hat{\beta}_{k-1}$, $k = 1, 2, \ldots,$ **n**.

nu is the index ν for which the error tolerance **eps** is satisfied for the first time; if it is never satisfied, **nu** will have the value **numax**.

ierr is an error flag, where
ierr = 0 on normal return,
ierr = 1 if **iopt** is inadmissible,
ierr = **nu0** if **nu0** > **numax**,
ierr = **numax** if the backward recurrence algorithm does not converge, and
ierr = −1 if **n** is not in range.

ierrc is an error flag inherited from the routine **cheb**.

The real arrays **fnu,s,s0,s1,s2** are working space, all of dimension $2 \times \mathbf{n}$, except **s**, which has dimension **n**. The complex arrays **rho**, **rold** are also working space, both of dimension $2n$. The routine calls on the subroutines **knum** and **cheb**. The double-precision version of **gchri** has the name **dgchri**.

Since the routine **gchri** is based on the modified Chebyshev algorithm, it shares with the latter its proneness to ill-conditioning, particularly in cases of measures supported on an infinite interval. On finitely supported measures, however, it can be quite effective, as seen in the next example.

Example 5.3 The performance of **chri** and **gchri**.

To illustrate the severe limitations of the routine **chri** in situations where divisions of the measure $d\lambda$ are involved, and at the same time to document the effectiveness of **gchri**, we ran both routines with $n = 40$ for Jacobi measures $d\lambda^{(\alpha,\beta)}$ with parameters $\alpha, \beta = -.8(.4).8, \beta \geq \alpha$. This is done in **test11**.

The routine **test11** first tests division by $t - x$, where $x = -1.001, -1.01, -1.04, -1.07$, and -1.1. Both routines **chri** and **gchri** are run in single and double precision, the latter with $\epsilon = 10 \times \epsilon^s$ and $\epsilon = 100 \times \epsilon^d$, respectively. The double-precision results are used to determine the absolute errors in the $\hat{\alpha}$'s and the relative errors in the $\hat{\beta}$'s for each routine. The required coefficients α_k, β_k, $0 \leq k \leq \nu_{\max} - 1$ ($\nu_{\max} = 500$ for single precision and 800 for double precision) are supplied by **recur** and **drecur** with **ipoly** = 6. The routine **nu0jac** is used to provide the starting recurrence index ν_0 (resp., ν_0^d). In Tables XII and XIII, relating, respectively, to linear and quadratic divisors, we give only the results for the Legendre measure ($\alpha = \beta = 0$). The first line in each three-line block of Table XII shows x, ν_0, ν_0^d, and the maximum (over k, $0 \leq k \leq 39$) errors in the $\hat{\alpha}_k$ and $\hat{\beta}_k$ for **gchri**, followed by the analogous information (except the ν_0's) for **chri**. The recurrence index ν yielding convergence was found (not shown in **test11**) to be $\nu = \nu_0 + 5$ and $\nu^d = \nu_0^d + 5$, without exception.

It can be seen from the leading lines in Table XII that **chri** rapidly loses accuracy as x moves away from the interval $[-1, 1]$, all single-precision accuracy being gone by the time x reaches -1.1. Similar, if not more rapid, erosion of accuracy is observed for the other parameter values of α, β. The next two lines in each three-line block show "reconstruction errors," that is, the maximum errors in the α's and β's if the $\hat{\alpha}$'s and $\hat{\beta}$'s produced by **gchri**, **chri** and **dgchri**, **dchri** are fed back to the routines **chri** and **dchri** with **iopt** = 1 to recover the original recursion coefficients in single and double precision. The first of these two lines shows the errors in reconstructing these coefficients from the output of **gchri** (resp., **dgchri**), and the second from the output of **chri** (resp., **dchri**). Rather remarkably, the coefficients are recovered to essentially full accuracy, even when the input coefficients (produced by **chri** and **dchri**) are very inaccurate! This is certainly a phenomenon that deserves further study. It can also be seen from Table XII (and the more complete results in Section 1 of the package) that **gchri** consistently produces accurate results, some slight deterioration occurring only very close to $x = -1$, where the routine has to work harder.

Table XII. Performance of **gchri** and **chri** for Elementary Divisors $t - x$ of the Legendre Measure $d\lambda(t)$

			gchri		**chri**	
x	ν_0	ν_0^d	err $\hat{\alpha}$	err $\hat{\beta}$	err $\hat{\alpha}$	err $\hat{\beta}$
−1.001	418	757	8.000(−14)	1.559(−13)	1.013(−13)	1.647(−13)
			8.527(−14)*	1.705(−13)	1.010(−27)	2.423(−27)
			1.421(−14)*	5.329(−14)	2.019(−28)	1.211(−27)
−1.010	187	294	4.016(−14)	6.907(−14)	1.396(−10)	2.424(−10)
			3.553(−14)	9.946(−14)	6.058(−28)	1.211(−27)
			7.105(−15)	4.262(−14)	1.515(−28)	9.080(−28)
−1.040	133	187	3.590(−14)	4.759(−14)	5.944(−6)	8.970(−6)
			2.842(−14)	7.103(−14)	5.554(−28)	1.312(−27)
			7.105(−15)	4.263(−14)	1.010(−28)	9.080(−28)
−1.070	120	161	2.194(−14)	4.850(−14)	5.334(−3)	7.460(−3)
			2.842(−14)	7.104(−14)	6.058(−28)	1.211(−27)
			7.105(−15)	4.263(−14)	1.010(−28)	7.062(−28)
−1.100	114	148	2.238(−14)	4.359(−14)	4.163(0)	4.959(+1)
			2.132(−14)	5.683(−14)	3.534(−28)	1.009(−27)
			1.549(−12)	1.833(−12)	1.010(−28)	6.057(−28)

*The second two lines of each three-line block show reconstruction errors.

Table XIII. Performance of **gchri** and **chri** for Elementary Divisors $(t - x)^2 + y^2$ of the Legendre Measure $d\lambda(t)$ with $z = x + iy$ on $\mathscr{E}_\rho$

			gchri		**chri**	
ρ	$\bar{\nu}_0$	$\bar{\nu}_0^d$	$\overline{\text{err}}\ \hat{\alpha}$	$\overline{\text{err}}\ \hat{\beta}$	$\overline{\text{err}}\ \hat{\alpha}$	$\overline{\text{err}}\ \hat{\beta}$
1.050	390	700	7.879(−13)	1.440(−12)	7.685(−14)	1.556(−13)
			7.814(−13)*	1.433(−12)	1.786(−26)	3.042(−26)
			2.024(−14)*	8.442(−14)	3.016(−28)	1.742(−27)
1.275	142	204	6.252(−14)	1.287(−13)	4.562(−7)	6.162(−7)
			6.554(−14)	1.279(−13)	1.541(−27)	3.061(−27)
			2.295(−14)	8.970(−14)	3.579(−28)	1.646(−27)
1.500	117	154	3.991(−14)	7.966(−14)	4.906(−1)	2.339(0)
			4.207(−14)	9.064(−14)	6.932(−28)	1.676(−27)
			3.805(−14)	8.971(−14)	4.351(−28)	1.744(−27)

*The second two lines of each three-line block show reconstruction errors.

The second half of **test11** tests division by $(t - x)^2 + y^2$, where $z = x + iy$ is taken along the upper half of the ellipse

$$\mathscr{E}_\rho = \left\{ z \in \mathbb{C}: z = \frac{1}{2}\left(\rho e^{i\vartheta} + \frac{1}{\rho} e^{-i\vartheta} \right), 0 \le \vartheta \le 2\pi \right\}, \qquad \rho > 1, \quad (5.10)$$

which has foci ± 1 and sum of the semiaxes equal to ρ. (These ellipses are contours of constant ν_0 for Jacobi measures.) We generated information analogous to the one in Table XII, for $\rho = 1.05, 1.1625, 1.275, 1.3875$, and 1.5,

except that all quantities are averaged over 19 equally spaced points on $\mathscr{E}_\rho$ corresponding to $\vartheta = j\pi/20$, $j = 1, 2, \ldots, 19$. Selected results (bars indicate averaging), again for the Legendre case, are shown in Table XIII. They reveal a behavior very similar to the one in Table XII for linear divisors.

6. GAUSS-TYPE QUADRATURE RULES

One of the important uses of orthogonal polynomials is in the approximation of integrals involving a positive measure $d\lambda$ by quadrature rules of maximum, or nearly maximum, algebraic degree of exactness. In this context, it is indispensable to know the recursion coefficients for the respective orthogonal polynomials $\{\pi_k(\cdot; d\lambda)\}$, since they allow us to generate the desired quadrature rules accurately and effectively via eigenvalue techniques. The software developed in the previous sections thus finds here a vast area of application.

6.1 Gaussian Quadrature

Given the (positive) measure $d\lambda$ (having an infinite number of support points), there exists, for each $n \in \mathbb{N}$, a quadrature rule

$$\int_{\mathbb{R}} f(t)\, d\lambda(t) = \sum_{k=1}^{n} w_k f(x_k) + R_n(f) \tag{6.1}$$

having algebraic degree of exactness $2n - 1$, that is, zero error, $R_n(f) = 0$, whenever f is a polynomial of degree $\leq 2n - 1$. The nodes x_k indeed are the zeros of the nth-degree orthogonal polynomial $\pi_n(\cdot; d\lambda)$, and the weights w_k, which are all positive, are also expressible in terms of the same orthogonal polynomials. Alternatively, and more importantly for computational purposes, the nodes x_k are the eigenvalues of the nth-order Jacobi matrix

$$J_n(d\lambda) = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \ddots & \ddots & \\ & & \ddots & & \sqrt{\beta_{n-1}} \\ 0 & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix}, \tag{6.2}$$

where $\alpha_k = \alpha_k(d\lambda)$, $\beta_k = \beta_k(d\lambda)$ are the recurrence coefficients for the (monic) orthogonal polynomials $\{\pi_k(\cdot; d\lambda)\}$, and the weights w_k are expressible in terms of the associated eigenvectors. Specifically, if

$$J_n(d\lambda)v_k = x_k v_k, \qquad v_k^T v_k = 1, \qquad k = 1, 2, \ldots, n, \tag{6.3}$$

that is, if v_k is the normalized eigenvector of $J_n(d\lambda)$ corresponding to the eigenvalue x_k, then

$$w_k = \beta_0 v_{k,1}^2, \qquad k = 1, 2, \ldots, n, \tag{6.4}$$

where $\beta_0 = \beta_0(d\lambda)$ is defined in (1.4) and $v_{k,1}$ is the first component of v_k (cf. Golub and Welsch [1969]). There are well-known and efficient algorithms, such as the QR algorithm, to compute eigenvalues and (part of the) eigenvectors of symmetric tridiagonal matrices. These are used in the routine **gauss**,[4] whose calling sequence is as follows:

gauss(n, alpha, beta, eps, zero, weight, ierr, e).

On entry,

n is the number of terms in the Gauss formula; type integer.

alpha, beta are arrays of dimension **n** assumed to hold the recursion coefficients **alpha**$(k) = \alpha_{k-1}$, **beta**$(k) = \beta_{k-1}$, $k = 1, 2, \ldots,$ **n**.

eps is a relative error tolerance, for example, the machine precision.

On return,

zero, weight are arrays of dimension **n** containing the nodes (in increasing order) and the corresponding weights of the Gauss formula, **zero**$(k) = x_k$, **weight**$(k) = w_k$, $k = 1, 2, \ldots,$ **n**.

ierr is an error flag equal to 0 on normal return, equal to i if the QR algorithm does not converge within 30 iterations on evaluating the ith eigenvalue, equal to -1 if **n** is not in range, and equal to -2 if one of the β's is negative.

The array **e** of dimension **n** is used for working space. The double-precision routine has the name **dgauss**.

We refrain here from giving numerical examples, since the use of the routine **gauss** and the routines yet to be described is straightforward. Some use of **gauss** and **dgauss** has already been made in Examples 4.2–4.4 and 5.2.

6.2 Gauss–Radau Quadrature

We now assume that $d\lambda$ is a measure whose support is either bounded from below, bounded from above, or both. Let x_0 be either the infimum or the supremum of supp $d\lambda$, so long as it is finite. (Typically, if supp $d\lambda = [-1, 1]$, then x_0 could be either -1 or $+1$; if supp $d\lambda = [0, \infty]$, then x_0 would have to be 0; etc.). By *Gauss–Radau quadrature* we then mean a quadrature rule of maximum degree of exactness that contains among the nodes the point x_0. It thus has the form

$$\int_{\mathbb{R}} f(t)\, d\lambda(t) = w_0 f(x_0) + \sum_{k=1}^{n} w_k f(x_k) + R_n(f) \tag{6.5}$$

[4]This routine was kindly supplied to the author by Professor G. H. Golub.

and, as is well known, can be made to have a degree of exactness $2n$, that is, $R_n(f) = 0$ for all polynomials of degree $\leq 2n$. Interestingly, all nodes $x_0, x_1, \ldots, x_n$ and weights $w_0, w_1, \ldots, w_n$ can again be interpreted in terms of eigenvalues and eigenvectors, exactly as in the case of Gaussian quadrature rules, but now relative to the matrix (cf. Golub [1973])

$$J_{n+1}^*(d\lambda) = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \ddots & & \\ & \ddots & \ddots & \sqrt{\beta_{n-1}} & \\ & & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{\beta_n} \\ 0 & & & \sqrt{\beta_n} & \alpha_n^* \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}, \quad (6.6)$$

where $\alpha_k = \alpha_k(d\lambda)(0 \leq k \leq n-1)$, $\beta_k = \beta_k(d\lambda)(1 \leq k \leq n)$ as before, but

$$\alpha_n^* = \alpha_n^*(d\lambda) = x_0 - \beta_n \frac{\pi_{n-1}(x_0; d\lambda)}{\pi_n(x_0; d\lambda)}. \quad (6.7)$$

Hence, we can use the routine **gauss** to generate the Gauss–Radau formula. This is done in the following subroutine:

```
      subroutine radau(n, alpha, beta, end, zero, weight, ierr, e, a, b)
      dimension alpha(*), beta(*), zero(*), weight(*), e(*), a(*), b(*)
c
c  The arrays alpha, beta, zero, weight, e, a, b are assumed to have
c  dimension n + 1.
c
      epsma = r1mach(3)
c
c  epsma is the machine single precision.
c
      np1 = n + 1
      do 10 k = 1, np1
        a(k) = alpha(k)
        b(k) = beta(k)
   10 continue
      p0 = 0.
      p1 = 1.
      do 20 k = 1, n
        pm1 = p0
        p0 = p1
        p1 = (end - a(k))*p0 - b(k)*pm1
   20 continue
      a(np1) = end - b(np1)*p0 / p1
      call gauss(np1, a, b, epsma, zero, weight, ierr, e)
      return
      end
```

The input variables are **n**, **alpha**, **beta**, and **end**, representing, respectively, n; two arrays of dimension $n+1$ containing the $\alpha_k(d\lambda)$, $\beta_k(d\lambda)$, $k = 0, 1, 2, \ldots, n$; and the "endpoint" x_0. The nodes (in increasing order) of the

Gauss–Radau formula are returned in the array **zero**, and the corresponding weights in the array **weight**. The arrays **e**, **a**, **b** are working space, and **ierr** is an error flag inherited from the routine **gauss**. The double-precision routine has the name **dradau**.

We remark that x_0 could also be outside the support of $d\lambda$, in which case the routine would generate a "Christoffel-type" quadrature rule.

6.3 Gauss – Lobatto Quadrature

Assuming now the support of $d\lambda$ bounded on either side, we let $x_0 = \inf \operatorname{supp}(d\lambda)$ and $x_{n+1} = \sup \operatorname{supp}(d\lambda)$ and consider a quadrature rule of the type

$$\int_{\mathbb{R}} f(t)\, d\lambda(t) = w_0 f(x_0) + \sum_{k=1}^{n} w_k f(x_k) + w_{n+1} f(x_{n+1}) + R_n(f) \quad (6.8)$$

having maximum degree of exactness $2n + 1$. This is called the *Gauss–Lobatto quadrature rule*. Its nodes $x_0, x_1, \ldots, x_{n+1}$ and weights $w_0, w_1, \ldots, w_{n+1}$ again admit the same spectral representation as in the case of the Gauss and Gauss–Radau formulas, only this time the matrix in question has order $n + 2$ and is given by (cf. Golub [1973])

$$J^*_{n+2}(d\lambda) = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & & \\ & \sqrt{\beta_2} & \ddots & & & \\ & & \ddots & \ddots & \sqrt{\beta_n} & \\ & & & \sqrt{\beta_n} & \alpha_n & \sqrt{\beta^*_{n+1}} \\ 0 & & & & \sqrt{\beta^*_{n+1}} & \alpha^*_{n+1} \end{bmatrix}. \quad (6.9)$$

Here, as before, $\alpha_k = \alpha_k(d\lambda)(0 \le k \le n)$, $\beta_k = \beta_k(d\lambda)(1 \le k \le n)$, and α^*_{n+1}, β^*_{n+1} are the unique solution of the linear 2×2 system

$$\begin{bmatrix} \pi_{n+1}(x_0; d\lambda) & \pi_n(x_0; d\lambda) \\ \pi_{n+1}(x_{n+1}; d\lambda) & \pi_n(x_{n+1}; d\lambda) \end{bmatrix} \begin{bmatrix} \alpha^*_{n+1} \\ \beta^*_{n+1} \end{bmatrix} = \begin{bmatrix} x_0 \pi_{n+1}(x_0; d\lambda) \\ x_{n+1} \pi_{n+1}(x_{n+1}; d\lambda) \end{bmatrix}. \quad (6.10)$$

Hence, we have the following routine for generating the Gauss–Lobatto formulas:

```
      subroutine lob(n, alpha, beta, aleft, right, zero, weight, ierr, e, a, b)
      dimension alpha(*), beta(*), zero(*), weight(*), e(*), a(*), b(*)
c
c  The arrays alpha, beta, zero, weight, e, a, b are assumed to have
c  dimension n + 2.
c
      epsma = r1mach(3)
c
c  epsma is the machine single precision.
c
```

```
      np1 = n + 1
      np2 = n + 2
      do 10 k = 1,np2
        a(k) = alpha(k)
        b(k) = beta(k)
   10 continue
      p0l = 0.
      p0r = 0.
      p1l = 1.
      p1r = 1.
      do 20 k = 1,np1
        pm1l = p0l
        p0l = p1l
        pm1r = p0r
        p0r = p1r
        p1l = (aleft - a(k))*p0l - b(k)*pm1l
        p1r = (right - a(k))*p0r - b(k)*pm1r
   20 continue
      det = p1l*p0r - p1r*p0l
      a(np2) = (aleft*p1l*p0r - right*p1r*p0l) / det
      b(np2) = (right - aleft)*p1l*p1r / det
      call gauss(np2,a,b,epsma,zero,weight,ierr,e)
      return
      end
```

The meaning of the input and output variables is as in the routine **radau**, the variable **aleft** standing for x_0 and **right** for x_{n+1}. The double-precision routine is named **dlob**.

A remark analogous to the one after the routine **radau** applies to the routine **lob**.

ACKNOWLEDGMENTS

The author gratefully acknowledges a number of suggestions from two anonymous referees and from the associate editor, Dr. Ronald F. Boisvert, for improving the code of the package.

REFERENCES

ABRAMOWITZ, M., AND STEGUN, I. A., EDS. 1964. Handbook of mathematical functions. NBS Appl. Math. Ser. 55, U.S. Government Printing Office, Washington, D.C.

BOLEY, D., AND GOLUB, G. H. 1987. A survey of matrix inverse eigenvalue problems. *Inverse Problems 3*, 4, 595–622.

CHEBYSHEV, P. L. 1859. Sur l'interpolation par la méthode des moindres carrés. *Mém. Acad. Impér. Sci. St. Pétersbourg* (7) *1*, 15, 1–24. (*Oeuvres I*, pp. 473–498.)

CHIHARA, T. S. 1978. *An Introduction to Orthogonal Polynomials*. Gordon and Breach, New York.

CHIHARA, T. S. 1985. Orthogonal polynomials and measures with end point masses. *Rocky Mountain J. Math. 15*, 3, 705–719.

CHRISTOFFEL, E. B. 1858. Über die Gaußische Quadratur und eine Verallgemeinerung derselben. *J. Reine Angew. Math 55*, 61–82. (*Ges. Math. Abhandlungen I*, pp. 65–87.)

CHRISTOFFEL, E. B. 1877. Sur une classe particulière de fonctions entières et de fractions continues. *Ann. Mat. Pura Appl. Ser. 2*, vol. 8, 1–10. (*Ges. Math. Abhandlungen II*, pp. 42–50.)

CODY, W. J., AND HILLSTROM, K. E. 1967. Chebyshev approximations for the natural logarithm of the gamma function. *Math. Comput 21*, 98 (Apr.), 198–203.

DANLOY, B. 1973. Numerical construction of Gaussian quadrature formulas for $\int_0^1(-\mathrm{Log}\ x)\cdot x^\alpha\cdot f(x)\cdot dx$ and $\int_0^\infty E_m(x)\cdot f(x)\cdot dx$. *Math. Comput. 27*, 124 (Oct.), 861–869.

DE BOOR, C., AND GOLUB, G. H. 1978. The numerically stable reconstruction of a Jacobi matrix from spectral data. *Linear Algebra Appl. 21*, 3 (Sept.), 245–260.

FRONTINI, M., GAUTSCHI, W., AND MILOVANOVIĆ, G. V. 1987. Moment-preserving spline approximation on finite intervals. *Numer. Math. 50*, 5 (Mar.), 503–518.

GALANT, D. 1969. Gauss quadrature rules for the evaluation of $2\pi^{-1/2}\int_0^\infty \exp(-x^2)f(x)\,dx$, review 42. *Math. Comput. 23*, 107 (July), 676–677. (Loose microfiche suppl. E.)

GALANT, D. 1971. An implementation of Christoffel's theorem in the theory of orthogonal polynomials. *Math. Comput. 25*, 113 (Jan.), 111–113.

GALANT, D. 1992. Algebraic methods for modified orthogonal polynomials. *Math. Comput. 59*, 200 (Oct.), 541–546.

GAUTSCHI, W. 1967a. Numerical quadrature in the presence of a singularity. *SIAM J. Numer. Anal. 4*, 3 (Sept.), 357–362.

GAUTSCHI, W. 1967b. Computational aspects of three-term recurrence relations. *SIAM Rev. 9*, 1 (Jan.), 24–82.

GAUTSCHI, W. 1979. On the preceding paper "A Legendre polynomial integral" by James L. Blue. *Math. Comput. 33*, 146 (Apr.), 742–743.

GAUTSCHI, W. 1981. Minimal solutions of three-term recurrence relations and orthogonal polynomials. *Math. Comput. 36*, 154 (Apr.), 547–554.

GAUTSCHI, W. 1982a. On generating orthogonal polynomials. *SIAM J. Sci. Stat. Comput. 3*, 3 (Sept.), 289–317.

GAUTSCHI, W. 1982b. An algorithmic implementation of the generalized Christoffel theorem. In *Numerical Integration*, G. Hämmerlin, Ed. International Series of Numerical Mathematics, vol. 57. Birkhäuser, Basel, pp. 89–106.

GAUTSCHI, W. 1984a. Discrete approximations to spherically symmetric distributions. *Numer. Math. 44*, 1 (June), 53–60.

GAUTSCHI, W. 1984b. Questions of numerical condition related to polynomials. In *Studies in Mathematics*, vol. 24. G. H. Golub, Ed. Studies in Numerical Analysis. Mathematical Association of America, Washington, D.C., pp. 140–177.

GAUTSCHI, W. 1986a. On the sensitivity of orthogonal polynomials to perturbations in the moments. *Numer. Math. 48*, 4 (Apr.), 369–382.

GAUTSCHI, W. 1986b. Reminiscences of my involvement in de Branges's proof of the Bieberbach conjecture. In *The Bieberbach Conjecture*. Mathematical Surveys and Monographs, no. 21. American Mathematical Society, Providence, R.I., pp. 205–211.

GAUTSCHI, W. 1990. Computational aspects of orthogonal polynomials. In *Orthogonal Polynomials—Theory and Practice*, P. Nevai, Ed. NATO ASI Series, Series C: Mathematical and Physical Sciences, vol. 294. Kluwer, Dordrecht, pp. 181–216.

GAUTSCHI, W. 1991a. A class of slowly convergent series and their summation by Gaussian quadrature. *Math. Comput. 57*, 195 (July), 309–324.

GAUTSCHI, W. 1991b. On certain slowly convergent series occurring in plate contact problems. *Math. Comput. 57*, 195 (July), 325–338.

GAUTSCHI, W. 1991c. Quadrature formulae on half-infinite intervals. *BIT 31*, 3, 438–446.

GAUTSCHI, W. 1991d. Computational problems and applications of orthogonal polynomials. In *Orthogonal Polynomials and Their Applications*. IMACS Annals on Computing and Applied Mathematics, vol. 9. Baltzer, Basel, pp. 61–71.

GAUTSCHI, W. 1993a. Gauss-type quadrature rules for rational functions. In *Numerical Integration IV*, H. Brass and G. Hämmerlin, Eds. International Series of Numerical Mathematics, vol. 112. Birhäuser, Basel, 111–130.

GAUTSCHI, W. 1993b. On the computation of generalized Fermi–Dirac and Bose–Einstein integrals. *Comput. Phys. Commun. 74*, 2 (Feb.), 233–238.

GAUTSCHI, W. 1993c. Is the recurrence relation for orthogonal polynomials always stable? *BIT 33*, 2, 277–284.

GAUTSCHI, W., AND LI, S. 1993. A set of orthogonal polynomials induced by a given orthogonal polynomial. *Aequationes Math. 46*, 1/2 (Aug.), 174–198.

GAUTSCHI, W., AND MILOVANOVIĆ, G. V. 1985. Gaussian quadrature involving Einstein and Fermi functions with an application to summation of series. *Math Comput. 44*, 169 (Jan.), 177–190.

GAUTSCHI, W., AND MILOVANOVIĆ, G. V. 1986. Spline approximations to spherically symmetric distributions. *Numer. Math. 49*, 2/3 (July), 111–121.

GAUTSCHI, W., AND VARGA, R. S. 1983. Error bounds for Gaussian quadrature of analytic functions. *SIAM J. Numer. Anal. 20*, 6 (Dec.), 1170–1186.

GAUTSCHI, W., AND WIMP, J. 1987. Computing the Hilbert transform of a Jacobi weight function. *BIT 27*, 2, 203–215.

GAUTSCHI, W., KOVAČEVIĆ, M. A., AND MILOVANOVIĆ, G. V. 1987. The numerical evaluation of singular integrals with coth-kernel. *BIT 27*, 3, 389–402.

GOLUB, G. H. 1973. Some modified matrix eigenvalue problems. *SIAM Rev. 15*, 2 (Apr.), 318–334.

GOLUB, G. H., AND WELSCH, J. H. 1969. Calculation of Gauss quadrature rules. *Math. Comput. 23*, 106 (Apr.), 221–230.

GRAGG, W. B., AND HARROD, W. J. 1984. The numerically stable reconstruction of Jacobi matrices from spectral data. *Numer. Math. 44*, 3 (Sept.), 317–335.

KAUTSKY, J., AND GOLUB, G. H. 1983. On the calculation of Jacobi matrices. *Linear Algebra Appl. 52-53* (July), 439–455.

LUKE, Y. L. 1975. *Mathematical Functions and Their Approximations*. Academic Press, New York.

REES, C. J. 1945. Elliptic orthogonal polynomials. *Duke Math J. 12*, 173–187.

RUTISHAUSER, H. 1963. On Jacobi rotation patterns. In *Experimental Arithmetic, High Speed Computing and Mathematics*. Proceedings of Symposia in Applied Mathematics, vol. *15*. American Mathematical Society, Providence, R.I., pp. 219–239.

SACK, R. A., AND DONOVAN, A. F. 1972. An algorithm for Gaussian quadrature given modified moments. *Numer. Math. 18*, 5 (Mar.), 465–478.

STROUD, A. H., AND SECREST, D. 1966. *Gaussian Quadrature Formulas* Prentice-Hall, Englewood Cliffs, N.J.

UVAROV, V. B. 1959. Relation between polynomials orthogonal with different weights. *Dokl. Akad. Nauk SSSR 126*, 1, 33–36. (In Russian.)

UVAROV, V. B. 1969. The connection between systems of polynomials that are orthogonal with respect to different distribution functions. *Ž Vyčisl. Mat. i Mat. Fiz. 9*, 6, 1253–1262. (In Russian.)

WHEELER, J. C. 1974. Modified moments and Gaussian quadrature. *Rocky Mountain J. Math. 4*, 2, 287–296.

WILKINSON, J. H. 1965. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford.

Received February 1992; revised December 1992 and March 1993; accepted March 1993