# Orthogonal Polynomials (in Matlab)

Walter Gautschi

Abstract. A suite of Matlab programs has been developed as part of the book "Orthogonal Polynomials: Computation and Approximation" expected to be published in 2004. The package contains routines for generating orthogonal polynomials as well as routines dealing with applications. In this paper, a brief review is given of the first part of the package, dealing with procedures for generating the three-term recurrence relation for orthogonal polynomials and more general recurrence relations for Sobolev orthogonal polynomials. Moment-based methods and discretization methods, and their implementation in Matlab, are among the principal topics discussed.

*Keywords:* Orthogonal polynomials; recurrence relations; Matlab.

## 1. Introduction

The analytic theory of orthogonal polynomials is well documented in a number of treatises; for classical orthogonal polynomials on the real line as well as on the circle, see [25], for those on the real line also [24]. General orthogonal polynomials are dealt with in [5] and more recently in [22], especially with regard to $n$th-root asymptotics. The text [3] is rooted in continued fraction theory and recurrence relations.

While the theory of orthogonal polynomials is well developed, the practice of orthogonal polynomials — constructive, computational, and software aspects — is still in an early stage of development. An effort in this direction is being made by the author's forthcoming book [13] and the accompanying package *OPQ: a Matlab Suite of Programs for Generating Orthogonal Polynomials and Related Quadrature Rules*, which can be found at the URL

`http://www.cs.purdue.edu/archives/2002/wxg/codes`.

The purpose of the work in [13] is twofold: (i) to present various procedures for generating the coefficients of the recurrence relations satisfied by orthogonal polynomials on the real line and by Sobolev orthogonal polynomials; and (ii) to discuss selected applications of these recurrence relations, including numerical quadrature, least squares and moment-preserving spline approximation, and the summation of slowly convergent series. All is to be implemented in the form of Matlab scripts. In the present article we wish to give a brief account of the first part of [13]: the generation of recurrence coefficients for orthogonal polynomials and related Matlab programs. All Matlab routines mentioned in this paper, and many others, are downloadable individually from the above Web site.

## 2. Orthogonal polynomials

We begin with some basic facts about orthogonal polynomials on the real line and introduce appropriate notation as we go along. Suppose $\mathrm{d}\lambda$ is a positive measure supported on an interval (or a set of disjoint intervals) on the real line such that all moments $\mu_r = \int_{\mathbb{R}} t^r \mathrm{d}\lambda(t)$ exist and are finite. Then the inner product

$$(p,q)_{\mathrm{d}\lambda} = \int_{\mathbb{R}} p(t)q(t)\mathrm{d}\lambda(t) \tag{1}$$

is well defined for any polynomials $p$, $q$ and gives rise to a unique system $\pi_r(t) = t^r + \cdots$, $r = 0, 1, 2, \dots$, of monic orthogonal polynomials

$$\pi_k(\,\cdot\,) = \pi_k(\,\cdot\,;\mathrm{d}\lambda): \quad (\pi_k, \pi_\ell)_{\mathrm{d}\lambda} \begin{cases} = 0,\ k \neq \ell, \\ > 0,\ k = \ell. \end{cases} \tag{2}$$

It is well known that they satisfy a three-term recurrence relation

$$\begin{gathered} \pi_{k+1}(t) = (t - \alpha_k)\pi_k(t) - \beta_k \pi_{k-1}(t), \quad k = 0, 1, 2, \dots, \\ \pi_{-1}(t) = 0, \quad \pi_0(t) = 1, \end{gathered} \tag{3}$$

where $\alpha_k = \alpha_k(\mathrm{d}\lambda)$ and $\beta_k = \beta_k(\mathrm{d}\lambda)$ are real resp. positive constants which depend on the measure $\mathrm{d}\lambda$. For convenience, we define $\beta_0 = \int_{\mathbb{R}} \mathrm{d}\lambda(t)$. Associated with the recurrence relation (3) is the *Jacobi matrix*

$$\boldsymbol{J}(\mathrm{d}\lambda) = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & \boldsymbol{0} \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \alpha_2 & \ddots & \\ & & \ddots & \ddots & \\ \boldsymbol{0} & & & & \end{bmatrix}, \tag{4}$$

a symmetric tridiagonal matrix of infinite order. Its leading principal minor matrix of order $n$ will be denoted by

$$\boldsymbol{J}_n(\mathrm{d}\lambda) = \boldsymbol{J}(\mathrm{d}\lambda)_{[1:n,1:n]}. \tag{5}$$

As already indicated in §1, the basic problem is this: for a given measure $\mathrm{d}\lambda$ and for given integer $n \geq 1$, generate the first $n$ coefficients $\alpha_k(\mathrm{d}\lambda)$, $k =$

$0, 1, 2, \ldots, n-1$, and the first $n$ coefficients $\beta_k(\mathrm{d}\lambda)$, $k = 0, 1, 2, \ldots, n-1$, that is, the Jacobi matrix $\boldsymbol{J}_n(\mathrm{d}\lambda)$ of order $n$ and $\beta_0$.

2.1. *Recurrence coefficients.* Frequently, the measure $\mathrm{d}\lambda$ is absolutely continuous, i.e., representable in the form

$$\mathrm{d}\lambda(t) = w(t)\mathrm{d}t, \tag{6}$$

where $w$ is a nonnegative function, called *weight function*, integrable on the support of $\mathrm{d}\lambda$ and not identically zero. Among the best-known weight functions are the classical weight functions, the more important of which are listed in Table 2.1.

| name | $w(t)$ | supported on |
|---|---|---|
| Jacobi | $(1-t)^\alpha(1+t)^\beta$, $\alpha > -1$, $\beta > -1$ | $[-1, 1]$ |
| Laguerre | $t^\alpha e^{-t}$, $\alpha > -1$ | $[0, \infty]$ |
| Hermite | $\lvert t\rvert^{2\alpha} e^{-t^2}$, $2\alpha > -1$ | $[-\infty, \infty]$ |

Table 2.1. Classical weight functions

For these, the recurrence coefficients are explicitly known. In Matlab, the first $N$ recurrence coefficients are always stored in an $N \times 2$ array `ab` as shown in Fig. 2.1.

$$\begin{array}{||c|c||}
\hline
\alpha_0 & \beta_0 \\
\alpha_1 & \beta_1 \\
\vdots & \vdots \\
\alpha_{N-1} & \beta_{N-1} \\
\hline
\end{array}$$

Figure 2.1. The array `ab` of recurrence coefficients

The Matlab command to compute them has the syntax `ab=r_`*name*(*parameters*), where *name* identifies the weight function, and *parameters* is a list of parameters including $N$. Thus, for example, in the case of the Jacobi weight function, the Matlab command is

```
ab=r_jacobi(N,a,b).
```

Here, `a`, `b` are the Jacobi parameters (denoted by $\alpha$ and $\beta$ in Table 2.1). If $\alpha = \beta$, it suffices to write `ab=r_jacobi(N,a)`, and if $\alpha = \beta = 0$, to write `ab=r_jacobi(N)`.

*Demo*#1. The first ten recurrence coefficients for the Jacobi polynomials with parameters $\alpha = -\frac{1}{2}$, $\beta = \frac{3}{2}$.

The Matlab command, followed by the output, is shown in the box below.

```
>> ab=r_jacobi(10,-.5,1.5)
ab =
        6.666666666666666e-01  4.712388980384690e+00
        1.333333333333333e-01  1.388888888888889e-01
        5.714285714285714e-02  2.100000000000000e-01
        3.174603174603174e-02  2.295918367346939e-01
        2.020202020202020e-02  2.376543209876543e-01
        1.398601398601399e-02  2.417355371900826e-01
        1.025641025641026e-02  2.440828402366864e-01
        7.843137254901961e-03  2.455555555555556e-01
        6.191950464396285e-03  2.465397923875433e-01
        5.012531328320802e-03  2.472299168975069e-01
```

Classical weight functions are not the only ones for which the recurrence coefficients are explicitly known. For example, the logistic weight function

$$w(t) = \frac{e^{-t}}{(1+e^{-t})^2}\,, \quad t \in \mathbb{R},$$

of interest in statistics, has all coefficients $\alpha_k = 0$ (by symmetry) and $\beta_0 = 1$, $\beta_k = k^4\pi^2/(4k^2-1)$, $k \geq 1$ ([3, Eq. (8.7) where $\lambda = 0$, $x = t/\pi$]). The corresponding Matlab routine is `r_logistic.m`. Other examples are measures occurring in the diatomic linear chain model, which are supported on two disjoint intervals; cf. [10].

Many nonclassical weight functions and measures, however, are such that their recurrence relations are not explicitly known. In these cases, numerical techniques must be used, some of which are to be described in the next four subsections.

2.2. *Modified Chebyshev algorithm.* In principle, the desired recurrence coefficients can be computed from well-known formulae expressing them in terms of Hankel-type determinants involving the moments $\mu_r$ of the given measure $\mathrm{d}\lambda$. The problems with this are: excessive complexity and, more seriously, extreme numerical instability. To avoid these problems, one can attempt to use *modified moments*

$$m_r = \int_{\mathbb{R}} p_r(t)\mathrm{d}\lambda(t), \quad r = 0, 1, 2, \ldots\,, \tag{7}$$

where $p_r$ are monic polynomials of degree $r$ "close" in some sense to the desired polynomials $\pi_r$. In particular, they are assumed to also satisfy a three-term recurrence relation

$$\begin{gathered} p_{k+1}(t) = (t-a_k)p_k(t) - b_k p_{k-1}(t), \quad k = 0, 1, 2, \ldots\,, \\ p_{-1}(t) = 0, \quad p_0(t) = 1, \end{gathered} \tag{8}$$

but this time with *known* coefficients $a_k \in \mathbb{R}$, $b_k \geq 0$. (We allow for zero coefficients $b_k$, since $a_k = b_k = 0$ yields the ordinary moments.) There is then a unique map

$$\mathbb{R}^{2n} \mapsto \mathbb{R}^{2n}: \quad [m_k]_{k=0}^{2n-1} \mapsto [\alpha_k, \beta_k]_{k=0}^{n-1} \tag{9}$$

that takes the first $2n$ modified moments into the desired $n$ recurrence coefficients $\alpha_k$ and $\beta_k$. An algorithm implementing this map has been developed by Sack and Donovan [21], and in more definitive form, by Wheeler [26]. In the case of ordinary moments ($a_k = b_k = 0$), it reduces to an algorithm alrady developed (for discrete measures) by Chebyshev [2]. We called it, therefore, the *Modified Chebyshev Algorithm*. It is implemented in the Matlab procedure

```
ab=chebyshev(N,mom,abm),
```

where `N` is the number $n$ in (9), `mom` the $1 \times 2N$ array of modified moments, and `abm` the $(2N-1) \times 2$ array of the first $2N-1$ recurrence coefficients $a_k$, $b_k$ in (8). If `abm` is omitted from the list of input parameters, the routine assumes `abm=zeros(2*N-1,2)`, that is, ordinary moments.

In view of the highly ill-conditioned nature of the map (9) when $m_r = \mu_r$ are ordinary moments, the conditioning of the modified moment map is an important question that has been studied already in [7], and more definitively in [9]. There are examples where the map is entirely well conditioned, but also others, especially when the measure $\mathrm{d}\lambda$ has unbounded support, in which the map is almost as ill conditioned as for ordinary moments.

*Demo*#2. The weight function

$$w(t) = [(1-\omega^2 t^2)(1-t^2)]^{-1/2} \quad \text{on } [-1,1], \quad 0 \leq \omega < 1,$$

of the "elliptic orthogonal polynomials".

Since the weight function reduces to the Chebyshev weight function when $\omega = 0$, it seems natural to use as modified moments those relative to the monic Chebyshev polynomials,

$$m_0 = \int_{-1}^{1} w(t)\mathrm{d}t, \quad m_k = \frac{1}{2^{k-1}} \int_{-1}^{1} T_k(t) w(t)\mathrm{d}t, \quad k \geq 1.$$

Their computation, though not trivial by any means, can be accomplished in a very stable fashion [9, Example 3.3]. The first $2N$ of them are generated in the Matlab routine `mm_ell.m`. The following box shows the Matlab script required to generate elliptic polynomials.

```
function ab=r_elliptic(N,om2)
abm=r_jacobi(2*N-1,-1/2);
mom=mm_ell(N,om2);
ab=chebyshev(N,mom,abm);
```

The routine works well even for $\omega^2$ quite close to 1, as is shown by the output below (displayed only partially) for `N`=40, `om2`=.999.

```
ab =
         0   9.682265121100620e+00
         0   7.937821421385184e-01
         0   1.198676724605757e-01
         0   2.270401183698990e-01
         0   2.410608787266061e-01
         0   2.454285325203698e-01
       ...       .................
         0   2.499915376529289e-01
         0   2.499924312667191e-01
         0   2.499932210069769e-01
```

All coefficients are accurate to machine precision.

2.3. *Discrete Stieltjes and Lanczos algorithm.* Partly in preparation for the next subsection, we now consider a *discrete N-point measure*

$$d\lambda_N(t) = \sum_{k=1}^{N} w_k \delta(t - x_k), \quad w_k > 0, \tag{10}$$

where $\delta$ is the Dirac delta function. Thus, the measure is supported on $N$ distinct points $x_k$ on the real axis, where it has positive jumps $w_k$. The corresponding inner product is a finite sum,

$$(p, q)_N = \int_{\mathbb{R}} p(t)q(t)d\lambda_N(t) = \sum_{k=1}^{N} w_k p(x_k)q(x_k). \tag{11}$$

There are now only a finite number, $N$, of recurrence coefficients $\alpha_k = \alpha_k(d\lambda_N)$, $\beta_k = \beta_k(d\lambda_N)$, which can be computed by either of two algorithms, one mentioned briefly by Stieltjes [23], and a more recent one based on ideas of Lanczos [18].

The former combines Darboux's formulae for the recurrence coefficients,

$$\begin{cases} \alpha_k = \dfrac{(t\pi_k, \pi_k)_N}{(\pi_k, \pi_k)_N}, \quad k = 0, 1, \ldots, n-1, \\ \beta_k = \dfrac{(\pi_k, \pi_k)_N}{(\pi_{k-1}, \pi_{k-1})_N}, \quad k = 1, 2, \ldots, n-1, \end{cases} \tag{12}$$

with the recurrence relation (3). In (12), the $\pi_k$ are the (as yet unknown) discrete orthogonal polynomials $\pi_k(\,\cdot\,; d\lambda_N)$. *Stieltjes's Procedure* consists in starting with $k = 0$ and successively increasing $k$ by 1 until $k = n - 1$. Thus, when $k = 0$, we have $\pi_0 = 1$, so that $\alpha_0$ can be computed by the top relation in

(12) with $k = 0$ and $\beta_0$ by $\beta_0 = \sum_{k=1}^{N} w_k$. With $\alpha_0$, $\beta_0$ at hand, we can go into (3) with $k = 0$ and compute $\pi_1(x_k)$ for all the support points $x_k$. This then in turn allows us to reapply (12) with $k = 1$ and compute $\alpha_1$ and $\beta_1$. Going back to (3) with $k = 1$, we compute $\pi_2(x_k)$, whereupon (12) with $k = 2$ yields $\alpha_2$, $\beta_2$, etc. In this manner we continue until $\alpha_{n-1}$, $\beta_{n-1}$ have been computed. Here $n \leq N$.

The second algorithm is based on the existence of an orthogonal similarity transformation

$$\boldsymbol{Q}^T \begin{bmatrix} 1 & \sqrt{w_1} & \sqrt{w_2} & \cdots & \sqrt{w_N} \\ \sqrt{w_1} & x_1 & 0 & \cdots & 0 \\ \sqrt{w_2} & 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sqrt{w_N} & 0 & 0 & \cdots & x_N \end{bmatrix} \boldsymbol{Q} = \begin{bmatrix} 1 & \sqrt{\beta_0} & 0 & \cdots & 0 \\ \sqrt{\beta_0} & \alpha_0 & \sqrt{\beta_1} & \cdots & 0 \\ 0 & \sqrt{\beta_1} & \alpha_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_{N-1} \end{bmatrix},$$

where $\boldsymbol{Q}$ is an orthogonal matrix of order $N + 1$ having the first coordinate vector $\boldsymbol{e}_1 \in \mathbb{R}^{N+1}$ as its first column. *Lanczos's Algorithm* [18] carries out this transformation and thus, since the $w_k$ and $x_k$ are given, determines the recurrence coefficients $\alpha_k$, $\beta_k$. The algorithm, unfortunately, is unstable, but can be stabilized by using ideas of Rutishauser [20]; see [16].

In Matlab, the two algorithms are implemented in the routines

```
ab=stieltjes(n,xw)
ab=lanczos(n,xw)
```

$$\left.\begin{array}{l} \texttt{ab=stieltjes(n,xw)} \\ \texttt{ab=lanczos(n,xw)} \end{array}\right\} \quad n \leq N,$$

where `xw` is the $N \times 2$ array of the support points and weights of the given discrete measure (10); see Fig. 2.2.

| | |
|---|---|
| $x_1$ | $w_1$ |
| $x_2$ | $w_2$ |
| $\vdots$ | $\vdots$ |
| $x_N$ | $w_N$ |

Figure 2.2. The array `xw` of support points and weights

The first routine is generally the one to be preferred, although as $n$ approaches $N$, it may gradually become unstable. If such is the case, and values of $n$ near $N$ are indeed required, the second routine is preferable but is considerably more time-consuming than the first.

2.4. *Discretization methods.* The basic idea, first advanced in [7] and more fully developed in [9], is very simple: One first approximates the given measure $\mathrm{d}\lambda$ by a discrete $N$-point measure,

$$\mathrm{d}\lambda(t) \approx \mathrm{d}\lambda_N(t), \tag{13}$$

typically by applying some appropriate quadrature scheme. Thereafter, the desired recurrence coefficients are approximated by those of the discrete measure,

$$\begin{cases} \alpha_k(\mathrm{d}\lambda) \approx \alpha_k(\mathrm{d}\lambda_N), \\ \beta_k(\mathrm{d}\lambda) \approx \beta_k(\mathrm{d}\lambda_N). \end{cases} \tag{14}$$

If necessary, the integer $N$ is increased to improve the approximation. For each $N$, the approximate recurrence coefficients on the right of (14) are computed by one of the methods described in §2.3. To come up with a good discretization (13) that yields fast convergence as $N \to \infty$ may require skill and inventiveness on the part of the user. But if implemented intelligently, the method is one of the most effective ones for generating orthogonal polynomials.

The seemingly complicated constructions of multicomponent discretizations to be described further on will first be motivated by a simple example.

*Example* 2.1. The weight function

$$w(t) = (1-t^2)^{-1/2} + c \text{ on } [-1,1], \quad c > 0.$$

When $c = 0$, this is the Chebyshev weight, and as $c \to \infty$, one expects to recover the Legendre polynomials. Thus, in a sense, the polynomials orthogonal with respect to $w$ "interpolate" between the Legendre and Chebyshev polynomials.

It would be very difficult to find a single quadrature scheme that would adequately approximate an integral with respect to the weight function $w$ by a finite sum. However, by considering $w$ as a 2-component weight function, the first component consisting of the Chebyshev weight, and the second of a constant weight function, a natural discretization is obtained by applying Gauss-Chebyshev quadrature to the first component, and Gauss-Legendre quadrature to the second. Thus, the inner product with respect to the weight function $w$ is approximated by

$$\begin{aligned} (p,q)_w &= \int_{-1}^{1} p(t)q(t)(1-t^2)^{-1/2}\mathrm{d}t + c\int_{-1}^{1} p(t)q(t)\mathrm{d}t \\ &\approx \sum_{k=1}^{M} w_k^{Ch} p(x_k^{Ch}) q(x_k^{Ch}) + c\sum_{k=1}^{M} w_k^{L} p(x_k^{L}) q(x_k^{L}), \end{aligned} \tag{15}$$

where $x_k^{Ch}$, $w_k^{Ch}$ are the nodes and weights of the $M$-point Gauss-Chebyshev quadrature formula, and $x_k^L$, $w_k^L$ those of the $M$-point Gauss-Legendre quadrature formula. This in effect approximates the measure $\mathrm{d}\lambda(t) = w(t)\mathrm{d}t$ by a discrete $N$-point measure $\mathrm{d}\lambda_N$, where $N = 2M$. Since $M$-point Gauss quadrature integrates polynomials of degree $2M-1$ exactly and all inner products in the Darboux formulae (12) involve polynomials of degree at most $2n-1$, the choice $M = n$ will insure that $\alpha_k(\mathrm{d}\lambda) = \alpha_k(\mathrm{d}\lambda_N)$ for all $k \le n-1$, and similarly for the $\beta_k$. Thus, Stieltjes's procedure, and therefore also Lanczos's algorithm, produces exact results. There is no need to increase $N$ any further.

In general, the support interval $[a, b]$ of $\mathrm{d}\lambda$ is decomposed into $m$ subintervals

$$[a, b] = \bigcup_{\mu=1}^{m} [a_\mu, b_\mu], \quad m \geq 1,$$

which may or may not be disjoint. The integral of a polynomial $f$ against the measure $\mathrm{d}\lambda(t) = w(t)\mathrm{d}t$ is then represented somehow in the form

$$\int_a^b f(t)w(t)\mathrm{d}t = \sum_{\mu=1}^{m} \int_{a_\mu}^{b_\mu} f_\mu(t)w_\mu(t)\mathrm{d}t, \tag{16}$$

where in the most general case $f_\mu$ will differ from $f$ (and in fact may no longer be a polynomial) and $w_\mu$ is a positive weight function which, too, may be different from $w$. The *Multicomponent Discretization Method* uses (16) with $f(t) = p(t)q(t)$ to approximate the inner product $(p, q)_w$ by applying an appropriate $M$-point quadrature rule to each constituent integral on the right of (16). This yields an approximation $\mathrm{d}\lambda \approx \mathrm{d}\lambda_N$ with $N = mM$. If the given measure $\mathrm{d}\lambda$, in addition to the absolutely continuous component, contains also a discrete $p$-point component, then the latter is simply added to the $(mM)$-point approximation to yield an $N$-point approximation $\mathrm{d}\lambda_N$ with $N = mM + p$. Using either Stieltjes's procedure or Lanczos's algorithm, we then compute the approximations $\alpha_k(\mathrm{d}\lambda_N)$, $\beta_k(\mathrm{d}\lambda_N)$ of $\alpha_k(\mathrm{d}\lambda)$, $\beta_k(\mathrm{d}\lambda)$ for $k = 0, 1, \ldots, n-1$. The integer $M$ (and with it $N$) may be successively increased in an attempt to obtain sufficient accuracy.

In Matlab, the multicomponent discretization method is implemented in the routine

```
[ab,Mcap,kount]=mcdis(n,eps0,quad,Mmax).
```

Here, `n` is the number of recurrence coefficients to be computed, and `eps0` the desired relative accuracy in the $\beta$-coefficients. (The $\alpha$-coefficients, if they are small, or even zero, may be obtained only to an absolute accuracy of `eps0`.) The input parameter `quad` is a quadrature routine that generates the $M$ nodes and weights of the quadrature approximation of the $\mu$th component of $\mathrm{d}\lambda$ for the current discretization parameter $M$. It may be a user-defined routine tailored to the specific problem at hand, or a general-purpose routine provided automatically. The last input parameter `Mmax` is an upper bound for the discretization parameter $M$, which, when exceeded, causes the routine to issue an error message. The output parameter `ab` is the `n`$\times$2 array of the desired recurrence coefficients, `Mcap` the value of $M$ that yields the requested accuracy, and `kount` the number of iterations required to achieve this accuracy.

The details of the discretization must be specified prior to calling the procedure. They are embodied in the following global parameters:

| | |
|---|---|
| `mc` | the number of component intervals |
| `mp` | the number of points in the discrete part of the measure (`mp`=0 if there is none) |
| `iq` | to be set equal to 1 if a user-defined quadrature routine is to be used, and different from 1 otherwise |
| `idelta` | a parameter whose default value is 1, but which is preferably set equal to 2 if `iq`=1 and the user provides Gauss-type quadrature routines |
| `irout` | to be set equal to 1 if Stieltjes's procedure is to be used, and different from 1 otherwise |
| `DM` | if `mp`$>0$ an `mp`$\times 2$ array $[[x_1\ y_1];[x_2\ y_2];\ldots;[x_{\mathtt{mp}}\ y_{\mathtt{mp}}]]$ containing the abscissae and jumps of the discrete component of the measure |
| `AB` | an `mc`$\times 2$ array specifying the component intervals $[[a_1\ b_1];[a_2\ b_2];\ldots;[a_{\mathtt{mc}}\ b_{\mathtt{mc}}]]$. |

*Example* 2.2. Normalized Jacobi weight function plus a discrete measure,

$$d\lambda(t) = [\beta_0^J]^{-1}(1-t)^\alpha(1+t)^\beta dt + \sum_{j=1}^{p} y_j\delta(t-t_j)dt, \quad \alpha>-1,\ \beta>-1,\ y_j>0,$$

where $\beta_0^J = \int_{-1}^{1}(1-t)^\alpha(1+t)^\beta dt$.

Similarly as in Example 2.1, we use the $M$-point Gauss-Jacobi quadrature rule with $M = n$ and Jacobi parameters $\alpha$, $\beta$ to discretize the absolutely continuous component, but now add on the discrete $p$-point measure. As in Example 2.1, this will produce the first $n$ recurrence coefficients exactly. The Matlab routine implementing this is shown in the box below.

```
function ab=r_jacplus(n,alpha,beta,ty)
global mc mp iq idelta irout DM AB
global a b
a=alpha; b=beta;
mc=1; mp=size(ty,1); iq=1; idelta=2; irout=1;
Mmax=n+1; DM=ty; AB=[-1 1]; eps0=1e3*eps;
[ab,Mcap,kount]=mcdis(n,eps0,@quadjp,Mmax);
```

The variables `a` and `b` are declared global since they are used in the quadrature routine `quadjp.m`, which is shown in the next box. Note also the choice `Mmax=n+1`, which is legitimate since the discretization parameter $M = n$ yields exact results.

```
function xw=quadjp(N,mu)
global a b
ab=r_jacobi(N,a,b); ab(1,2)=1;
xw=gauss(N,ab);
```

The integer `mu` in the routine `quadjp` (in the present case `mu`=1) specifies the `mu`th component interval. The call to `gauss(N,ab)` generates the `N`-point Gaussian quadrature rule for the measure identified via the `N`$\times$2 array `ab` of its recurrence coefficients.

*Demo*#3. The first 40 recurrence coefficients of the normalized Jacobi weight function with parameters $\alpha = -\frac{1}{2}$, $\beta = \frac{3}{2}$ and a mass point of strength 2 added at the left endpoint of $[-1, 1]$.

The Matlab program, followed by the output (only partially displayed), is shown in the box below.

```
>> ty=[-1 2];
>> ab=r_jacplus(40,-.5,1.5,ty)
   ab =
         -4.444444444444e-01  3.000000000000e+00
          2.677002583979e-01  6.635802469136e-01
          3.224245925965e-01  8.620335316387e-02
          1.882535273840e-01  1.426676765162e-01
          1.207880431181e-01  1.809505902299e-01
          8.380358927439e-02  2.025747903114e-01
            ................    ................
          2.077921831426e-03  2.489342817850e-01
          1.972710627986e-03  2.489888786295e-01
          1.875292842444e-03  2.490393860403e-01
```

The results can be compared with analytic answers (cf. [11, p. 43]) and are found to be accurate to all digits shown.

*Example* 2.3. A weight function involving the modified Bessel function,

$$w(t) = t^{\alpha} K_0(t) \quad \text{on } [0, \infty], \quad \alpha > -1.$$

This has applications in the asymptotic approximation of oscillatory integral transforms [27].

The discretization of the measure $\mathrm{d}\lambda(t) = w(t)\mathrm{d}t$ should be done with due regard to the properties of the weight function, especially its behavior for small and large $t$. This behavior is determined by

$$K_0(t) = \begin{cases} R(t) + I_0(t)\ln(1/t) & \text{if } 0 < t \le 1, \\ t^{-1/2}e^{-t}S(t) & \text{if } 1 \le t < \infty, \end{cases}$$

where $I_0$ is the "regular" modified Bessel function and $R$, $S$ are smooth functions for which good rational approximations are known [19]. This suggests the decomposition $[0,\infty] = [0,1] \cup [0,1] \cup [0,\infty]$ and the representation

$$\begin{aligned} \int_0^\infty f(t)w(t)\mathrm{d}t = \int_0^1 [R(t)f(t)]t^\alpha \mathrm{d}t + \int_0^1 [I_0(t)f(t)]t^\alpha \ln(1/t)\mathrm{d}t \\ + e^{-1}\int_0^\infty [(1+t)^{\alpha-1/2}S(1+t)f(1+t)]e^{-t}\mathrm{d}t. \end{aligned} \tag{17}$$

Thus, in the notation of (16),

$$f_1(t) = R(t)f(t), \quad w_1(t) = t^\alpha \text{ on } [0,1],$$

$$f_2(t) = I_0(t)f(t), \quad w_2(t) = t^\alpha \ln(1/t) \text{ on } [0,1],$$

$$f_3(t) = e^{-1}(1+t)^{\alpha-1/2}S(1+t)f(1+t), \quad w_3(t) = e^{-t} \text{ on } [0,\infty].$$

The appropriate discretization of (17), therefore, involves Gauss-Jacobi quadrature (with parameters 0 and $\alpha$) for the first integral, Gauss quadrature relative to the weight function $w_2$ on $[0,1]$ for the second integral, and Gauss-Laguerre quadrature for the third integral. The Gaussian quadrature rules required are readily generated, the first and third by classical means, and the second by using the routine `r_jaclog.m` for generating the recurrence coefficients for the weight function $w_2$ followed by an application of the routine `gauss.m`. This is implemented for arbitrary $\alpha > -1$ in the routine `r_modbess.m` shown in the next box. The routine `r_jacobi01.m` called in the sixth line generates the recurrence coefficients for the shifted Jacobi polynomials (supported on the interval $[0,1]$). The variables `abjac`, `abjaclog`, `ablag`, declared global, are used in the quadrature routine `quadbess.m`, which also incorporates one of the rational approximations of [19] for computing $R$, $S$.

```
function ab=r_modbess(N,a,Mmax,eps0)
global mc mp iq idelta irout AB
global abjac abjaclog ablag
mc=3; mp=0; iq=1; idelta=2; irout=1;
AB=[[0 1];[0 1];[0 Inf]];
abjac=r_jacobi01(Mmax,0,a);
abjaclog=r_jaclog(Mmax,a);
ablag=r_laguerre(Nmax);
ab=mcdis(N,eps0,@quadbess,Mmax);
```

*Demo*#4. Compute

$$\int_0^\infty e^{-t} t^\alpha K_0(t)\mathrm{d}t = \frac{\sqrt{\pi}}{2^{\alpha+1}} \frac{\Gamma^2(\alpha+1)}{\Gamma(\alpha+3/2)}\,.$$

The routine in the box below applies $n$-point Gauss quadrature of $e^{-t}$ relative to the weight function $w(t) = t^\alpha K_0(t)$ and determines the smallest $n$ for which the relative error is less than `eps0`.

```
>> global a
>> a=-1/2; N=20; Mmax=200; eps0=1e4*eps;
>> exact=sqrt(pi)*(gamma(a+1))^2/(2^(a+1)*gamma(a+3/2));
>> ab=r_modbess(N,a,Mmax,eps0); s=0; n=0;
>> while abs(s-exact)>abs(exact)*eps0
     n=n+1;
     xw=gauss(n,ab);
     s=sum(xw(:,2).*exp(-xw(:,1)));
   end
>> n, s, abs(s-exact)/abs(exact)
```

For the choices made of `a`, `N`, `Mmax`, and `eps0`$=2.22\times10^{-12}$, the routine yields $n = 12$, $s = 3.937402486427721$, with a relative error of $7.32 \times 10^{-13}$.

2.5. *Modification algorithms.* The problem to be considered here is the following: Given the recurrence coefficients of $\mathrm{d}\lambda$, generate those of the modified measure

$$\mathrm{d}\lambda_{\mathrm{mod}}(t) = r(t)\mathrm{d}\lambda(t),\ \ r \text{ rational } \geq 0 \text{ on supp}(\mathrm{d}\lambda).$$

The problem can be reduced to the one in which $r$ is either a real linear, or a real quadratic factor or divisor, since any general real $r$ can be written as a product of such factors and divisors. For these special cases, the problem has been solved in [8]. (Other approaches have been taken in [17] and [4]; see also [12, §3].) We briefly discuss the case of a linear factor, already solved by Galant [6].

*Example* 2.4. Modification by a liner factor,

$$r(t) = s(t-c), \quad c \in \mathbb{R}\backslash\text{supp}(\mathrm{d}\lambda),$$

where $s = \pm1$ is chosen such that $r$ is nonnegative on the support of $\mathrm{d}\lambda$.

The solution given by Galant is most elegantly described in linear algebra terms. It consists in applying one step of the (symmetric) shifted LR algorithm to the Jacobi matrix of the measure $\mathrm{d}\lambda$. Specifically, the matrix $s[\boldsymbol{J}_{n+1}(\mathrm{d}\lambda) - c\boldsymbol{I}]$, which by assumption is positive definite, is first Cholesky decomposed,

$$s[\boldsymbol{J}_{n+1}(\mathrm{d}\lambda) - c\boldsymbol{I}] = \boldsymbol{L}\boldsymbol{L}^T,$$

whereupon the factors on the right are interchanged and the shift $c\boldsymbol{I}$ added back. Discarding the last row and column of the resulting matrix yields the desired Jacobi matrix of order $n$,

$$\boldsymbol{J}_n(\mathrm{d}\lambda_{\mathrm{mod}}) = (\boldsymbol{L}^T\boldsymbol{L} + c\boldsymbol{I})_{[1:n,1:n]}.$$

The solution can also be described in terms of a nonlinear recurrence algorithm, which in Matlab is implemented by the routine

```
ab=chri1(N,ab0,c),
```

where `ab0` contains the first $N+1$ recurrence coefficients of $\mathrm{d}\lambda$ and `c` is the shift parameter.

Our package includes seven additional routines `chri2.m`, `chri3.m`, …, `chri8.m` corresponding to quadratic factors of various types, linear divisors, and quadratic divisors of different kinds. The routine `chri7.m`, for example, deals with a quadratic factor of the form $r(t) = (t-x)^2$ with $x \in \mathbb{R}$. It would be tempting to apply the routine `chri1.m` for the linear factor $t - x$ twice in succession, but this may be risky if $x$ is inside the support of $\mathrm{d}\lambda$. There is, however, an algorithm similar to Galant's algorithm, which applies one step of the shifted QR algorithm to the Jacobi matrix $\boldsymbol{J}_{n+2}(\mathrm{d}\lambda)$ and discards the last two rows and columns of the result to obtain $\boldsymbol{J}_n(r\mathrm{d}\lambda)$ (cf. [12, §3.3]).

*Example* 2.5. Induced orthogonal polynomials ([14]).

Given an orthogonal polynomial $\pi_m(\,\cdot\,;\mathrm{d}\lambda)$ of fixed degree $m$, the *induced orthogonal polynomial* of degree $k$ is orthogonal with respect to the weight function $w(t) = \pi_m^2(t)\mathrm{d}\lambda(t)$.

Here,

$$r(t) = \prod_{\mu=1}^{m}(t - x_\mu)^2,$$

where $x_\mu$ are the zeros of $\pi_m$. This calls for $m$ successive applications of the routine `chri7.m` with $x = x_\mu$, $\mu = 1, 2, \ldots, m$. The routine `indop.m` shown in the box below implements this.

```
function ab=indop(N,m,ab0)
N0=size(ab0,1);
if N0<N+m, error('input array ab0 too short'), end
ab=ab0;
if m==0, return, end
zw=gauss(m,ab0);
for imu=1:m
   mi=N+m-imu;
   for n=1:mi+1
     ab1(n,1)=ab(n,1);
     ab1(n,2)=ab(n,2);
   end
   x=zw(imu,1);
   ab=chri7(mi,ab1,x);
end
```

*Demo*#5. Induced Legendre polynomials.

The routine shown in the next box generates the first 20 recurrence coefficients of selected induced orthogonal polynomials when d$\lambda$ is the Legendre measure.

```
>> N=20; M=11;
>> ab0=r_jacobi(N+M);
>> for m=[0 2 6 11]
     ab=indop(N,m,ab0)
   end
```

By symmetry, all the $\alpha$-coefficients are zero. Selected values of the $\beta$-coefficients returned by the routine (rounded to 10 decimal places) are shown in Table 2.2.

| $k$ | $\beta_{k,0}$ | $\beta_{k,2}$ | $\beta_{k,6}$ | $\beta_{k,11}$ |
|---|---|---|---|---|
| 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 |

Table 2.2. $\beta$-coefficients of induced Legendre polynomials

The procedure is remarkably stable, not only for the Legendre measure, but also for other classical measures, and for $n$ and $m$ as large as 320; see [11, Tables X and XI].

## 3. Sobolev orthogonal polynomials

These are polynomials orthogonal with respect to an inner product that involves derivatives in addition to function values, each derivative having associated with it its own (positive) measure. Thus,

$$(p,q)_S = \int_{\mathbb{R}} p(t)q(t)\mathrm{d}\lambda_0(t) + \int_{\mathbb{R}} p'(t)q'(t)\mathrm{d}\lambda_1(t) + \cdots + \int_{\mathbb{R}} p^{(s)}(t)q^{(s)}(t)\mathrm{d}\lambda_s(t). \tag{18}$$

The Sobolev polynomials $\{\pi_k(\,\cdot\,;S)\}$ are monic polynomials of degree $k$ orthogonal with respect to the inner product of (18),

$$(\pi_k, \pi_\ell)_S \begin{cases} = 0, \ k \neq \ell, \\ > 0, \ k = \ell. \end{cases} \tag{19}$$

These polynomials no longer satisfy a three-term recurrence relation, but like any other system of monic polynomials whose degrees increase by 1 from one polynomial to the next, they must satisfy a recurrence relation of the extended form

$$\pi_{k+1}(t) = t\pi_k(t) - \sum_{j=0}^{k} \beta_j^k \pi_{k-j}(t), \quad k = 0, 1, 2, \ldots\ . \tag{20}$$

In place of the Jacobi matrix, we now have an upper Hessenberg matrix of recurrence coefficients,

$$\boldsymbol{H}_n = \begin{bmatrix} \beta_0^0 & \beta_1^1 & \beta_2^2 & \cdots & \beta_{n-2}^{n-2} & \beta_{n-1}^{n-1} \\ 1 & \beta_0^1 & \beta_1^2 & \cdots & \beta_{n-3}^{n-2} & \beta_{n-2}^{n-1} \\ 0 & 1 & \beta_0^2 & \cdots & \beta_{n-4}^{n-2} & \beta_{n-3}^{n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \beta_0^{n-2} & \beta_1^{n-1} \\ 0 & 0 & 0 & \cdots & 1 & \beta_0^{n-1} \end{bmatrix}. \tag{21}$$

In the case $s = 0$ corresponding to ordinary orthogonal polynomials, one has $\beta_j^k = 0$ for $j > 0$, and the matrix $\boldsymbol{H}_n$ is tridiagonal. It can be symmetrized by a (real) diagonal similarity transformation and then becomes the Jacobi matrix $\boldsymbol{J}_n(\mathrm{d}\lambda_0)$ (cf. (4)). When $s > 0$, symmetrization is no longer possible, since some of the eigenvalues of $\boldsymbol{H}_n$ may well be complex.

3.1. *Moment-based algorithms.* We define modified moments similarly as in (7), but now a separate set of them for each measure $\mathrm{d}\lambda_\sigma$,

$$m_k^{(\sigma)} = \int_{\mathbb{R}} p_k(t)\mathrm{d}\lambda_\sigma(t), \quad k = 0, 1, 2, \ldots; \quad \sigma = 0, 1, \ldots, s. \tag{22}$$

For simplicity, we use the same set of polynomials $\{p_k\}$ for each measure and assume, as in (8), that they satisfy a three-term recurrence relation. In analogy to (9), there is now a unique map that takes the first $2n$ modified moments of all the measures $d\lambda_\sigma$ into the recurrence coefficients $\beta_j^k$,

$$[m_k^{(\sigma)}]_{k=0}^{2n-1},\ \sigma=0,1,\ldots,s\ \mapsto [\beta_j^k],\ k=0,1,\ldots n-1; j=0,1,\ldots,k. \quad (23)$$

The conditioning of this map has been studied in [28], and an algorithm, analogous to the modified Chebyshev algorithm, developed (for $s=1$) in [15]. The corresponding routine in Matlab is

```
[B,normsq]=chebyshev_sob(N,mom,abm).
```

Here, `N` is the $n$ in (23), `mom` the $2\times 2N$ array of the first $2N$ modified moments corresponding to $d\lambda_0$ and $d\lambda_1$, and `abm` the $(2N-1)\times 2$ array of the recurrence coefficients in (8). The output variable `B` is the $N\times N$ matrix of the recurrence coefficients $\beta_j^k$, $k=0,1,\ldots,N-1$, $0\le j\le k$, where $\beta_j^k$ occupies the position `B`$(j+1,k+1)$ of the matrix `B`; all remaining elements of `B` are zero. The routine also returns the optional $N$-vector `normsq` of the squared norms $\|\pi_k\|_S^2$ of the Sobolev orthogonal polynomials. If `abm` is absent in the list of input parameters, then ordinary moments are assumed ($a_k=b_k=0$).

*Example* 3.1. The polynomials of Althammer [1].

These are the Sobolev orthogonal polynomials with $s=1$ and $d\lambda_0(t)=dt$, $d\lambda_1(t)=\gamma dt$ on $[-1,1]$, where $\gamma>0$. There is a fairly obvious choice of the polynomials $\{p_k\}$ for defining the modified moments, namely the monic Legendre polynomials. All modified moments in this case, by orthogonality, are zero except for

$$m_0^{(0)}=2,\quad m_0^{(1)}=2\gamma.$$

In Matlab, the recurrence matrix `B` for the Althammer polynomials is generated as shown in the box below (where `N`$=n$ and `g`$=\gamma$).

```
>> N=20; g=1;
>> %g=0;
>> mom=zeros(2,2*N);
>> mom(1,1)=2; mom(2,1)=2*g;
>> abm=r_jacobi(2*N-1);
>> B=chebyshev_sob(N,mom,abm);
```

*Demo*#6. Legendre vs Althammer polynomials.

The routine in the box below generates and plots the Sobolev polynomial of degree $N=20$ corresponding to $s=1$ and $\gamma=0$ (Legendre polynomial) resp. $\gamma=1$ (Althammer polynomial). It is assumed that the matrix `B` has already been generated by the routine for Althammer polynomials shown above with `N`=20 and `g`=0 resp. `g`=1.

```
>> N=20;
>> pi=zeros(N+1,1); np=500; y=zeros(np+1,1);
>> for it=0:np
   t=-1+2*it/np;
   pi(1)=1;
   for k=1:N
     temp=0;
     for l=1:k
       temp=temp+B(l,k)*pi(k-l+1);
     end
     pi(k+1)=t*pi(k)-temp;
   end
   y(it+1)=pi(N+1);
end
>> x=linspace(-1,1,np+1);
>> hold on
>> plot(x',y)
>> plot([-1 1],[0 0],'--')
>> hold off
```

The plot for the Legendre polynomial is shown in Fig. 3.1 in the left frame, and the one for the Althammer polynomial in the right frame.

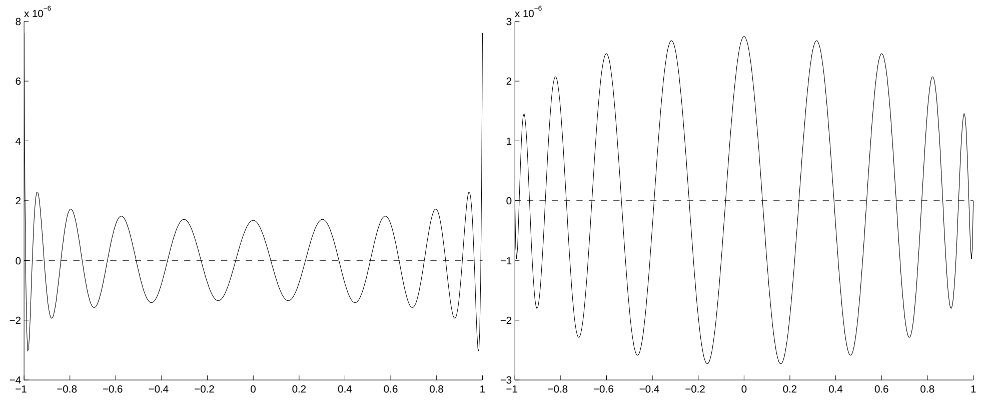

Figure 3.1. Legendre vs Althammer polynomial

Interestingly, for the Legendre polynomial the envelope of the extreme points is convex on top and concave at the bottom, whereas for the Althammer polynomial it is the other way around. Note also that $\pi_{20}(\pm 1) = .7607 \times 10^{-5}$ for the Legendre, and $\pi_{20}(\pm 1) = 0$ for the Althammer polynomial.

3.2. *Discretization algorithm.* The analogue for Sobolev orthogonal polynomials of the Darboux formulae (12) is

$$\beta_j^k = \frac{(t\pi_k, \pi_{k-j})_S}{(\pi_{k-j}, \pi_{k-j})_S}, \quad j = 0, 1, \ldots, k, \tag{24}$$

with the inner product $(\,\cdot\,,\,\cdot\,)_S$ defined as in (18). The *Discretized Stieltjes Algorithm*, similarly as for ordinary orthogonal polynomials, consists in combining the formulae (24) with the recurrence relation (20), discretizing the inner products in (24) by suitable quadrature schemes. We chose to approximate the absolutely continuous component of each measure $\mathrm{d}\lambda_\sigma$ by a Gauss-type quadrature rule,

$$(p, q)_{\mathrm{d}\lambda_\sigma} \approx \sum_{k=1}^{n_\sigma} w_k^{(\sigma)} p(x_k^{(\sigma)}) q(x_k^{(\sigma)}), \quad \sigma = 0, 1, \ldots, s, \tag{25}$$

and to add on any discrete component of $\mathrm{d}\lambda_\sigma$ if present. In Matlab, the quadrature schemes are identified by an $md \times 2(s+1)$ array `xw`,

$$\texttt{xw=}\left\|\begin{array}{ccc|ccc} x_1^{(0)} & \cdots & x_1^{(s)} & w_1^{(0)} & \cdots & w_1^{(s)} \\ x_2^{(0)} & \cdots & x_2^{(s)} & w_2^{(0)} & \cdots & w_2^{(s)} \\ \vdots & & \vdots & \vdots & & \vdots \\ x_{\mathrm{md}}^{(0)} & \cdots & x_{\mathrm{md}}^{(s)} & w_{\mathrm{md}}^{(0)} & \cdots & w_{\mathrm{md}}^{(s)} \end{array}\right\|$$

where `md`=max$(n_\sigma)$. In each column of `xw` the entries after $x_{n_\sigma}^{(\sigma)}$ resp. $w_{n_\sigma}^{(\sigma)}$ (if any) are ignored by the routine. The routine itself has the form

```
B=stieltjes_sob(N,s,nd,xw,a0,same),
```

where `nd`=$[n_0, n_1, \ldots, n_s]$, `a0`=$\alpha_0(\mathrm{d}\lambda_0)$, and `same` is a logical variable to be set equal to 1 if all quadrature rules have the same nodes, and equal to 0 otherwise. If `same`=1, the routine takes advantage of significant simplifications that are possible and reduce running time.

*Example* 3.2. The Althammer polynomials, revisited.

The box below shows the generation of the recurrence matrix `B` for the Althammer polynomials using the routine `stieltjes_sob.m`.

```
>> N=20; g=1;
>> nd=[N N]; s=1; a0=0; same=1;
>> ab=r_jacobi(N);
>> zw=gauss(N,ab);
>> xw=[zw(:,1) zw(:,1) zw(:,2) g*zw(:,2)];
>> B=stieltjes_sob(N,s,nd,xw,a0,same);
```

The results are identical with those produced by the routine `chebyshev_sob.m`. There is no restriction, however, on the parameter $s$ when using the routine `stieltjes_sob.m`.

3.3. *Zeros.* If $\boldsymbol{\pi}(t)$ is the vector of the first $n$ Sobolev orthogonal polynomials,

$$\boldsymbol{\pi}^T(t) = [\pi_0(t), \pi_1(t), \ldots, \pi_{n-1}(t)],$$

then the recurrence relation (20) can be written in matrix form as follows,

$$t\boldsymbol{\pi}^T(t) = \boldsymbol{\pi}^T(t)\boldsymbol{H}_n + \pi_n(t)\boldsymbol{e}_n^T,$$

where $\boldsymbol{e}_n$ is the last coordinate vector in $\mathbb{R}^n$. If $t = \tau_\nu$ is a zero of $\pi_n$, the last term vanishes, implying that $\tau_\nu$ is an eigenvalue of the matrix $\boldsymbol{H}_n$ and $\boldsymbol{\pi}^T(\tau_\nu)$ a corresponding (left) eigenvector. Thus, the zeros of Sobolev orthogonal polynomials can be computed as eigenvalues of an upper Hessenberg matrix. In Matlab, this is done by the routine `sobzeros.m` shown in the box below.

```
function z=sobzeros(n,N,B)
H=zeros(n);
for i=1:n
   for j=1:n
     if i==1
       H(i,j)=B(j,j);
     elseif j==i-1
       H(i,j)=1;
     elseif j>=i
       H(i,j)=B(j-i+1,j);
     end
   end
end
z=sort(eig(H));
```

Here `B` is the recurrence matrix of order $N$ for the Sobolev orthogonal polynomials, and `n`$\leq$`N`. The zeros are arranged in increasing order.

*Demo*#7. The zeros of the Althammer polynomial of degree 20 with $\gamma = 1$.

Assuming that the matrix `B` has already been generated by either the modified

Chebyshev algorithm or the Stieltjes procedure as described in §§3.1 and 3.2, the box below shows the Matlab commands and output (only the positive zeros are shown, rounded to 12 decimals).

```
<< N=20; z=sobzeros(N,N,B)
z =
     8.05392515636e-02
     2.39532838077e-01
     3.92325438959e-01
     5.34960935873e-01
     6.63745343244e-01
     7.75342384688e-01
     8.66859942239e-01
     9.35924777578e-01
     9.80740571465e-01
     1.00000000000e-01
```

Judging from how well the symmetry of the roots is satisfied, the results appear to be accurate to all digits shown except the last, which may be in error by one or two units. Generating the matrix `B` by the modified Chebyshev algorithm or Stieltjes's procedure produces the same results to this accuracy, but the Stieltjes procedure is considerably slower (by a factor of about 14) than the modified Chebyshev algorithm.

## References

[1] Althammer, P. 1962. Eine Erweiterung des Orthogonalitätsbegriffes bei Polynomen und deren Anwendung auf die beste Approximation. *J. Reine Angew. Math. 211*, pp. 192–204.

[2] Chebyshev, P. L. 1859. Sur l'interpolation par la méthode des moindres carrés. *Mem. Acad. Impér. Sci. St. Petersbourg (7)1(15)*, pp. 1–24. Also in *Œuvres I*, pp. 473–498.

[3] Chihara, T. S. 1978. *An introduction to orthogonal polynomials.* Mathematics and Its Applications **13**, Gordon and Breach, New York.

[4] Fischer, Bernd and Golub, Gene H. 1992. How to generate unknown orthogonal polynomials out of known orthogonal polynomials. *J. Comput. Appl. Math. 43*, pp. 99–115.

[5] Freud, Géza 1971. *Orthogonal polynomials.* Pergamon Press, New York. (English translation of *Orthogonale Polynome*, Birkhäuser, Basel, 1969.)

[6] Galant, David 1971. An implementation of Christoffel's theorem in the theory of orthogonal polynomials. *Math. Comp. 25*, pp. 111–113.

[7] Gautschi, Walter 1968. Construction of Gauss-Christoffel quadrature formulas. *Math. Comp. 22*, pp. 251–270.

[8] Gautschi, Walter 1982. An algorithmic implementation of the generalized Christoffel theorem. In: *Numerical integration* (G. Hämmerlin, ed.), *Internat. Ser. Numer. Math. 57*, pp. 89–106, Birkhäuser, Basel.

[9] Gautschi, Walter 1982. On generating orthogonal polynomials. *SIAM J. Sci. Statist. Comput. 3*, pp. 289–317.

[10] Gautschi, Walter 1984. On some orthogonal polynomials of interest in theoretical chemistry. *BIT 24*, pp. 473–483.

[11] Gautschi, Walter 1994. Algorithm 726: ORTHPOL — a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules. *ACM Trans. Math. Software 20*, pp. 21–62.

[12] Gautschi, Walter 2002. The interplay between classical analysis and (numerical) linear algebra — a tribute to Gene H. Golub. *Electron. Trans. Numer. Anal. 13*, pp. 119–147 (electronic).

[13] Gautschi, Walter 2004. *Orthogonal polynomials: computation and approximation.* Clarendon Press, Oxford, to appear.

[14] Gautschi, Walter and Li, Shikang 1993. A set of orthogonal polynomials induced by a given orthogonal polynomial. *Aequationes Math. 46*, pp. 174–198.

[15] Gautschi, Walter and Zhang, Minda 1995. Computing orthogonal polynomials in Sobolev spaces. *Numer. Math. 71*, pp. 159–183.

[16] Gragg, William B. and Harrod, William J. 1984. The numerically stable reconstruction of Jacobi matrices from spectral data. *Numer. Math. 44*, pp. 317–335.

[17] Kautsky, J. and Golub, G. H. 1983. On the calculation of Jacobi matrices. *Linear Algebra Appl. 52/53*, pp. 439–455.

[18] Lanczos, Cornelius 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Research Nat. Bur. Standards 45*, pp. 255–282. Also in *Collected Published Papers with Commentaries*, Vol. V, pp. 3-9–3-36.

[19] Russon, A. E. and Blair, J. M. 1969. Rational function minimax approximations for the Bessel functions $K_0(x)$ and $K_1(x)$. *Rep. AECL-3461*, Atomic Energy of Canada Limited, Chalk River, Ontario.

[20] Rutishauser, H. 1963. On Jacobi rotation patterns. In: *Experimental arithmetics, high speed computing and mathematics* (N. C. Metropolis, A. H. Taub, J. Todd, and C. B. Tompkins, eds.), *Proc. Sympos. Appl. Math. 15*, pp. 219–239, American Mathematical Society, Providence, RI.

[21] Sack, R. A. and Donovan, A. F. 1972. An algorithm for Gaussian quadrature given modified moments. *Numer. Math. 18*, pp. 465–478.

[22] Stahl, Herbert and Totik, Vilmos 1992. *General orthogonal polynomials.* Encyclopedia of Mathematics and Its Applications **43**, Cambridge University Press, Cambridge.

[23] Stieltjes, T. J. 1884. Quelques recherches sur la théorie des quadratures dites mécaniques. *Ann. Sci. École Norm. Paris (3) 1*, pp. 409–426. Also in *Œuvres I*, pp. 377–396.

[24] Suetin, P. K. 1979. *Classical orthogonal polynomials* (Second ed.). "Nauka", Moscow. (Russian)

[25] Szegö, Gabor 1975. *Orthogonal polynomials* (Fourth ed.). AMS Colloquium Publications **23**, American Mathematical Society, Providence, RI.

[26] Wheeler, John C. 1974. Modified moments and Gaussian quadratures. *Rocky Mountain J. Math. 4*, pp. 287–296.

[27] Wong, R. 1982. Quadrature formulas for oscillatory integral transforms. *Numer. Math. 39*, pp. 351–360.

[28] Zhang, Minda 1994. Sensitivity analysis for computing orthogonal polynomials of Sobolev type. In: *Approximation and computation* (R.V.M. Zahar, ed.), *Internat. Ser. Numer. Math. 119*, pp. 563–576, Birkhäuser Boston, Boston, MA.

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1389
USA

*E-mail address:* `wxg@cs.purdue.edu` (W. Gautschi)