MATHEMATICS OF COMPUTATION
Volume 75, Number 256, October 2006, Pages 1891–1903
S 0025-5718(06)01855-2
Article electronically published on May 23, 2006

# ORTHOGONAL POLYNOMIALS FOR REFINABLE LINEAR FUNCTIONALS

DIRK LAURIE AND JOHAN DE VILLIERS

Abstract. A refinable linear functional is one that can be expressed as a convex combination and defined by a finite number of mask coefficients of certain stretched and shifted replicas of itself. The notion generalizes an integral weighted by a refinable function. The key to calculating a Gaussian quadrature formula for such a functional is to find the three-term recursion coefficients for the polynomials orthogonal with respect to that functional. We show how to obtain the recursion coefficients by using only the mask coefficients, and without the aid of modified moments. Our result implies the existence of the corresponding refinable functional whenever the mask coefficients are nonnegative, even when the same mask does not define a refinable function. The algorithm requires $O(n^2)$ rational operations and, thus, can in principle deliver exact results. Numerical evidence suggests that it is also effective in floating-point arithmetic.

## 1. Refinable linear functionals

Let $N$ be a positive integer and $\mathcal{P}$ be the linear space of all polynomials with real coeffients. Denote by $E_j : \mathcal{P} \to \mathcal{P}$ the stretch-shift operator defined by

$$E_j f(x) = f\left(\frac{x+j}{2}\right), \qquad j = 0, 1, \ldots, N. \tag{1.1}$$

We say that the linear functional $L : \mathcal{P} \to \mathbb{R}$ is *refinable* if there exists an $N$-tuple

$$\boldsymbol{\gamma} = (\gamma_0, \gamma_1, \ldots, \gamma_N)$$

of real numbers, called a *mask*, such that

$$L[f] = \frac{1}{2}\sum_{j=0}^{N} \gamma_j L[E_j f]. \tag{1.2}$$

We consider only the case where $L[e_0]$ is nonzero, where $e_0$ denotes the constant function $e_0(x) = 1$, in which case one may assume without loss of generality that

$$L[e_0] = 1.$$

Clearly (put $f = e_0$) the mask coefficients satisfy

$$\sum_{j=0}^{N} \gamma_j = 2. \tag{1.3}$$

Received by the editor October 22, 2004 and, in revised form, May 3, 2005.
2000 *Mathematics Subject Classification.* Primary 65D30, 42C40; Secondary 42C05, 65D07.
*Key words and phrases.* Gaussian quadrature, refinable, orthogonal polynomials.

We assume that $\gamma_0$ and $\gamma_N$ are nonzero, and that all $\gamma_j \geqslant 0$, which implies that (1.2) can be interpreted as saying that $L$ is a convex combination of certain shifted and stretched replicas of itself.

The most obvious refinable functionals are integrals of the form

$$L[f] = \int_{-\infty}^{\infty} f(x)\phi(x)\,\mathrm{d}x, \tag{1.4}$$

where $\phi$ is a *refinable function*; i.e., one that satisfies

$$\phi(x) = \sum_{j=0}^{N} \gamma_j \phi(2x-j), \quad x \in \mathbb{R}; \tag{1.5}$$

$$\int_{-\infty}^{\infty} \phi(x)\,\mathrm{d}x = 1. \tag{1.6}$$

The best-known refinable weight functions are the cardinal B-splines, although most of the literature on B-splines (see, e.g., [5]) does not exploit their refinability. The theory of refinable weight functions is central in wavelet analysis, where the calculation of integrals of the form (1.4) is important. The reader is warned that the term "refinable function" as usually defined (see, e.g., [12]) involves certain constraints on the mask, implying additional properties (existence, continuity, smoothness) of $\phi$. We do not require those constraints here, mainly because $\phi$ is never actually evaluated, and need not even exist: everything depends only on the property (1.2) of $L$.

The refinable functional defined by a mask may exist even when the corresponding refinable function does not. For example, when $N = 1$, $\gamma_0 = \gamma$, $\gamma_1 = 2 - \gamma$, with $\gamma \neq 1$, it can easily be proved that there is no nonzero function $\phi$ which is right-continuous at 0, continuous at $\frac{m}{n}$, $m = 1, \ldots, n-1$ for some natural number $n$, and left-continuous at 1, and satisfies (1.5). Yet the corresponding functional does exist, as we shall show at the end of §3.

In general, when there is no refinable function over $\mathbb{R}$ corresponding to a given mask $\boldsymbol{\gamma}$, one can find a constant $c$ such that when $\boldsymbol{\gamma}$ is replaced by $c\boldsymbol{\gamma}$, there exists a refinable function over the set $\mathbb{D}$ of dyadic points, i.e., points that have a terminating binary expansion. In other words, for the adjusted mask there exists $\phi : \mathbb{D} \to \mathbb{R}$ that satisfies (1.5) for $x \in \mathbb{D}$. The closer $c$ is to 1, the more the graph of $\phi$ appears to be continuous.

For example, in our earlier paper [11] we devoted a lot of attention to the mask

$$\boldsymbol{\gamma} = \left(\frac{2}{7}, \frac{4}{7}, \frac{6}{7}, \frac{2}{7}\right).$$

In this case, $c = \frac{7}{2}\left(\frac{1}{2} - \frac{1}{10}\sqrt{5}\right) \doteq 0.9673762$. The graph of $\phi$ in the latter case [11, Fig. 4] looks like noise superimposed on a rather jagged continuous function, with the amplitude of the noise being greatest near the centre of the interval $[0, 3]$.

In our opinion there is much interesting work to be done on the generalization of the concept of refinability to masks that does not meet the requirements for the existence of a refinable function. Our notion of a refinable functional is a first step in that direction.

However, the thrust of this paper is not the investigation of nonclassical cases, but rather the presentation of a simple and elegant computational algorithm for computing the corresponding orthogonal polynomials.

## 2. Orthogonal polynomials with respect to a refinable linear functional

The theory of orthogonal polynomials is most often presented [7] in terms of inner products associated with weighted integrals, i.e.,

$$\langle f, g\rangle = L[fg], \tag{2.1}$$

with certain conditions imposed on the weight function to ensure that $\langle\cdot,\cdot\rangle$ defines an inner product over suitable function spaces, in particular $\mathcal{P}$. In some cases, it is useful to think of $\langle\cdot,\cdot\rangle$ only as a bilinear form. Quadrature formulas are subsumed in this framework in one of two ways: either by allowing the weight function to be a linear combination of Dirac deltas, or by Stieltjes integration, where $\phi(x)\,dx$ is replaced by $d\sigma(x)$ for a suitable measure $\sigma$.

The inner product notation is a little too general, even when regarded as a bilinear form, in the sense that it obscures the crucial associative property of (2.1), namely that $\langle fg, h\rangle = \langle f, gh\rangle$. The existence of a three-term recursion formula depends on this property holding for the case $g(x) = x$. For this reason, we prefer to formulate the theory as that of polynomials orthogonal with respect to the linear functional $L$. Although it is trivial that $\langle f, g\rangle = L[fg]$, the latter notation exposes the associative property. It also points the way to a fruitful generalization: as pointed out by Chihara [4, 7], a system of orthogonal polynomials up to degree $n$ exists if $L$ is quasi-definite over $\mathcal{P}_{n-1}$; i.e., $L[p^2] \neq 0$ for all nonzero $p \in \mathcal{P}_{n-1}$. Thus, by working with linear functionals, we avoid the need to know anything about $L$ beyond its quasi-definiteness.

The monic orthogonal polynomials $p_k$ with respect to $L$ are defined by $L[p_k p_l] = 0,\ k \neq l$, and can be generated by the classical *Stieltjes algorithm*:

**Initialize:** $p_{-1}(x) = 0,\ p_0(x) = 1.$

**Iterate:** For $k = 0, 1, 2, \ldots$:

$$b_k = \begin{cases} 1, & k = 0, \\ L[p_k^2]/L[p_{k-1}^2], & k > 0; \end{cases} \tag{2.2}$$

$$a_k = L[xp_k^2]/L[p_k^2], \tag{2.3}$$

$$p_{k+1}(x) = (x - a_k)p_k(x) - b_k p_{k-1}(x). \tag{2.4}$$

It is well known [16] that knowledge of the recursion coefficients is tantamount to knowledge of the Gauss–Christoffel quadrature formula for the functional $L$, which can be computed by the Golub–Welsch algorithm [9].

The Stieltjes algorithm presupposes the ability to evaluate $L[f]$ when $f$ is a polynomial. The case where $L$ is given by (1.4) has been considered by several authors. Evaluation of $L[f]$ is straightforward when $\phi$ is a B-spline, which is piecewise polynomial with discontinuities at $0, 1, \ldots, N$. The integrals in the Stieltjes algorithm can then be calculated exactly except for a roundoff error by the compound Gauss–Legendre quadrature. This approach was taken by Phillips and Hanson [14], but clearly does not generalize to arbitrary masks. For reasonably smooth masks, Gautschi et al. [8] succeeded in computing the integrals in 2.2, (2.3) by a quadrature formula, involving the evaluation of $\phi$ at thousands of dyadic points.

The need to evaluate $\phi$ is avoided by Laurie and De Villiers [11], generalizing an idea of Sweldens and Piessens [17]. This is that having chosen an *auxiliary basis* $\{\pi_l,\ l = 0, 1, 2, \ldots\}$ for the linear space $\mathcal{P}_n$ of polynomials of degree not more than

$n$, one can evaluate the modified moments

$$\mu_l = I[\pi_l]$$

directly from the mask coefficients $\gamma_j$, without resorting to quadrature. Once the modified moments are known, the Sack–Donovan algorithm [15] can be used to find the recursion coefficients. Modified moments for refinable functions are also used in [1, 2, 10].

All the above algorithms make use of the property

$$\phi(x) = 0, \quad x \notin [0, N]; \tag{2.5}$$

that refinable functions have under additional constraints on the mask. In [14] and [8] these properties are essential; in [17] and [11] they influence the choice of an auxiliary basis.

In [17], Chebyshev polynomials of the first kind over $[0, N]$ are used as an auxiliary basis, and the derivation exploits some special properties of those polynomials. In [11], algorithms are derived which require only that the auxiliary basis be generated by a three-term recursion similar to (2.2)–(2.4)—no properties of the auxiliary basis beyond the numerical values of the recursion coefficients are required.

Three possibilities for the auxiliary basis are considered in [11]. Algorithm 1 uses the monomials (which trivially satisfy a three-term recursion with all coefficients zero), working in rational arithmetic because of the notorious ill-conditioning [6] of the map from the monomials to the orthogonal polynomials. Since the option of rational arithmetic is not available when the mask coefficients themselves are irrational, a numerically stable algorithm is needed. Algorithm 2 uses Chebyshev polynomials of the second kind over $[0, N]$; the rationale is that their weight function $\sqrt{x(N-x)}$ is zero at the endpoints, as $\phi$ typically is. Even these polynomials have been observed to suffer from noticeable if slight ill-conditioning, which is addressed in Algorithm 3 by the use of a floating-point approximation to the orthogonal polynomials $p_k$, as computed by Algorithm 2. Algorithm 3 still requires some coefficients past those approximations, for which those of the Chebyshev polynomials of the second kind are retained.

The use of $\pi_k = p_k$ as auxiliary basis may well be the best conditioned of all, but to approximate it iteratively by using a different basis first is not elegant. The main purpose of this paper is to derive an algorithm that bypasses that step and, moreover, makes no use at all of the finite support property (2.5). Our approach is algebraic and algorithmic, with no appeal to analytical properties of $\phi$. We shall say nothing further about the interplay of properties of $\boldsymbol{\gamma}$ with those of $\phi$, such as necessary and sufficient conditions for the existence, continuity, smoothness, etc., of $\phi$ when $\boldsymbol{\gamma}$ is given. There is a rich literature on that topic, some highlights of which are summarized in [11].

Our algorithm produces the three-term coefficients by means of a finite number of rational operations on the mask coefficients $\boldsymbol{\gamma}$. The only essential property of $\boldsymbol{\gamma}$ is that $\sum_{j=0}^{N} \gamma_j = 2$. The property $\gamma_j \geqslant 0,\ j = 0, 1, \ldots n$, guarantees that the algorithms do not break down, since we shall show that it implies that the functional $L$ is positive definite; i.e., $L[f] > 0$ whenever $f \in \mathcal{P}$ is nonnegative everywhere and positive on a set of measure greater than 0. The algebraic identities underlying the algorithm remain true even when some $\gamma_j$ are negative, provided that no zero divisor occurs during the execution of the algorithm.

## 3. Derivation of the algorithm

We recapitulate the argument given in [17] and generalized in [11]. The crucial fact is that when $f \in \mathcal{P}_n$, then $E_j f \in \mathcal{P}_n$. Thus, there exist coefficients $c_{j,k,l}$ such that

$$E_j \pi_k = \sum_{l=0}^{k} c_{j,k,l} \pi_l. \tag{3.1}$$

Substituting $f = \pi_k$ and (3.1) into (1.2), and recalling that $\sum_{j=0}^{N} \gamma_j = 2$, one obtains

$$\mu_k = \frac{1}{2} \sum_{j=0}^{N} \sum_{l=0}^{k} \gamma_j c_{j,k,l} \mu_l,$$

and thus

$$\mu_k = \frac{1}{2(1 - c_{j,k,k})} \sum_{j=0}^{N} \sum_{l=0}^{k-1} \gamma_j c_{j,k,l} \mu_l, \tag{3.2}$$

which can be used to compute $\mu_k$ recursively starting from $\mu_0 = 1$. The recursion is well defined since it follows from (1.1) that

$$c_{j,k,k} = 2^{-k}. \tag{3.3}$$

In [11] the algorithms involve the generation of $c_{j,k,l}$ up to $k = 2n-1$, from which $\mu_k$ up to $k = 2n-1$ are computed by (3.2), after which the recursion coefficients $a_k, b_k$ up to $k = n-1$ are computed by the Sack–Donovan algorithm.

Now comes the new idea. Put $\pi_l = p_l$ : the algorithms in [11] can no longer be used because the auxiliary basis is as yet unknown. But we shall show that the information that $\pi_l = p_l$ allows a direct realization of the Stieltjes algorithm. Let $p_{j,k} = E_j p_k$; then by (2.4), (1.1) we find that

$$p_{j,k+1}(x) = ((e_1 + j)/2 - a_k) p_{j,k} - b_k p_{j,k-1}. \tag{3.4}$$

The notation $e_1$ denotes the identity function $e_1(t) = t$. The bare-bones algorithm is

**Initialize:** $p_{j,-1}(x) = 0,\ p_{j,0}(x) = 1,\ j = 0, 1, \ldots, N.$

**Iterate:** For $k = 0, 1, 2, \ldots$ :

$$b_k = \begin{cases} 1, & k = 0, \\ L[p_k^2]/L[p_{k-1}^2], & k > 0; \end{cases}$$

$$a_k = L[e_1 p_k^2]/L[p_k^2];$$

Calculate $p_{j,k},\ j = 0, 1, \ldots, N$, by (3.4).

The difficulty is, of course, how do we evaluate the required integrals? Since $p_{j,k}$ is known as an expansion in terms of the orthogonal polynomials $p_l,\ l = 0, 1, \ldots, k$, we appeal to a well-known general procedure in such cases. If

$$\begin{aligned} f &= f_0 p_0 + f_1 p_1 + \cdots + f_m p_m, \\ g &= g_0 p_0 + g_1 p_1 + \cdots + g_n p_n, \end{aligned} \tag{3.5}$$

then we define a bilinear form by

$$\langle f,g\rangle_k = \sum_{l=0}^{k} f_l g_l L[p_l^2]. \tag{3.6}$$

This gives

$$L[fg] = \langle f,g\rangle_{\min(m,n)}. \tag{3.7}$$

The snag is that at the start of the inductive step, we know $L[p_l^2]$ only for $l \leqslant k-1$, whereas $\min(m,n) = k$ when $L[p_k^2]$ is to be evaluated by (3.7).

Before proceeding, let us take another look at the derivation of (3.2). Define the functions $s : \mathbb{R} \to \mathbb{R}$ and $\hat{s} : \mathbb{R} \to \mathbb{R}$ by

$$s(\mu_k) = \frac{1}{2}\sum_{j=0}^{N}\sum_{l=0}^{k} \gamma_j c_{j,k,l}\mu_l, \qquad \hat{s}(\mu_k) = s(\mu_k) - 2^{-k}\mu_k;$$

then $\mu_k = \hat{s}(\mu_k)/(1-2^{-k})$. The point is that $\hat{s}$, despite the form of the equation in which it is defined, is a constant function, since the term that contains $\mu_k$ has been subtracted from $s(\mu_k)$. Therefore $\hat{s}(\mu_k) = \hat{s}(0) = s(0)$, and indeed (3.2) simply says $\mu_k = s(0)/(1-2^{-k})$.

Now apply a similar idea to the calculation of $L[p_k^2]$. Define

$$\hat{p}_{j,k} = p_{j,k} - 2^{-k}p_k. \tag{3.8}$$

Despite the form of (3.8), $\hat{p}_{j,k}$ does not really depend on $p_k$: in the light of (3.3), it is a polynomial of degree at most $k-1$. Using (3.1), (3.3), we obtain

$$\begin{aligned}
L[p_k^2] &= \frac{1}{2}\sum_{j=0}^{N}\gamma_j L[E_j p_k^2] = \frac{1}{2}\sum_{j=0}^{N}\gamma_j L[p_{j,k}p_{j,k}] \\
&= \frac{1}{2}\sum_{j=0}^{N}\gamma_j L[p_{j,k}(2^{-k}p_k + \hat{p}_{j,k})] \\
&= 2^{-k-1}L[p_k p_{j,k}]\sum_{j=0}^{N}\gamma_j + \frac{1}{2}\sum_{j=0}^{N}\gamma_j L[p_{j,k}\hat{p}_{j,k}] \\
&= 2^{-k}L[p_k p_{j,k}] + \frac{1}{2}\sum_{j=0}^{N}\gamma_j L[p_{j,k}\hat{p}_{j,k}].
\end{aligned}$$

By (3.7), (3.6), $L[p_k p_{j,k}] = c_{j,k,k}L[p_k^2] = 2^{-k}L[p_k^2]$. Thus we can solve for $L[p_k^2]$, obtaining, with the aid of (1.3), that

$$L[p_k^2] = \frac{1}{2(1-2^{-2k})}\sum_{j=0}^{N}\gamma_j L[p_{j,k}\hat{p}_{j,k}]. \tag{3.9}$$

Since $\hat{p}_{j,k}$ is of degree at most $k-1$, we have

$$L[p_{j,k}\hat{p}_{j,k}] = \langle p_{j,k},\hat{p}_{j,k}\rangle_k = \langle p_{j,k},\hat{p}_{j,k}\rangle_{k-1} = \langle p_{j,k},p_{j,k}\rangle_{k-1}. \tag{3.10}$$

Thus the right-hand side of (3.9) involves only known quantities. Once we know $L[p_k^2]$, we can calculate $b_k$ by (2.2).

At this point we have enough information to allow us to apply (3.7) also when $\min(m,n) \leqslant k$, so one might think that $L[e_1 p_k^2]$ is now possible:

$$L[e_1 p_k^2] = \frac{1}{2}\sum_{j=0}^{N} \gamma_j L[E_j e_1 p_k^2] = \frac{1}{4}\sum_{j=0}^{N} \gamma_j L[(e_1 + j) p_{j,k}^2],$$

in which one of the factors has degree $k$. The snag lies in the calculation of $(e_1 + j)p_{j,k}$ : if we know the expansion (3.5), the way to calculate the expansion of $e_1 f$ is

$$e_1 f = \sum_{l=0}^{m} f_l e_1 p_l = \sum_{l=0}^{m} f_l (p_{l+1} + a_l p_l + b_l p_{l-1}). \tag{3.11}$$

Thus, we require the value of $a_k$, and we do not yet have it.

Once again the way forward is to subtract explicitly the term involving the unknown quantity. Let the coefficients $f_l$ in (3.11) refer to the expansion of $p_{j,k}$ (to avoid clutter, the explicit dependence on $j$ and $k$ will be restored only after the derivation) and define the function $q : \mathbb{R} \to \mathcal{P}_{k+1}$ by

$$q(\alpha) = \sum_{l=0}^{k-1} f_l (p_{l+1} + a_l p_l + b_l p_{l-1}) + f_k (p_{k+1} + \alpha p_k + b_k p_{k-1}) + j p_{j,k} - \alpha f_k p_k.$$

Then $q(a_k) = (e_1 + j)p_{j,k} - a_k f_k p_k$. But clearly $q(\alpha)$ is independent of $\alpha$, so the equation $(e_1 + j)p_{j,k} = q(\alpha) + a_k f_k p_k$ holds for any value of $\alpha$; a particularly convenient choice is $\alpha = 0$. Defining $\tilde{p}_{j,k+1} = q(0)$ and recalling that $f_k = c_{j,k,k} = 2^{-k}$, we obtain

$$(e_1 + j)p_{j,k} = \tilde{p}_{j,k+1} + a_k 2^{-k} p_k,$$

an expression in which all the quantities are known. We can now proceed to calculate $L[e_1 p_k^2]$ as

$$\begin{aligned} L[e_1 p_k^2] &= \frac{1}{4}\sum_{j=0}^{N} \gamma_j L[(\tilde{p}_{j,k+1} + a_k 2^{-k} p_k) p_{j,k}] \\ &= \frac{1}{4}\sum_{j=0}^{N} \gamma_j L[\tilde{p}_{j,k+1} p_{j,k}] + \frac{1}{4}\sum_{j=0}^{N} \gamma_j a_k 2^{-k} L[p_k p_{j,k}] \\ &= \frac{1}{4}\sum_{j=0}^{N} \gamma_j L[\tilde{p}_{j,k+1} p_{j,k}] + 2^{-2k-1} a_k L[p_k^2]. \end{aligned}$$

But by (2.3) $a_k L[p_k^2] = L[e_1 p_k^2]$, and therefore

$$L[e_1 p_k^2] = \frac{1}{4(1 - 2^{-2k-1})}\sum_{j=0}^{N} \gamma_j L[\tilde{p}_{j,k+1} p_{j,k}]. \tag{3.12}$$

*Remark.* Our algorithm implies the existence of the functional $L$, and that it is positive definite by the following argument. The shifted polynomials $p_{j,k}$ are not identically equal to $c_{j,k,k} p_k$; therefore, (3.9), (3.10) implies that $L[p_k^2] > 0$, and as many recursion coefficients as we like can be computed. Thus $L[f]$ can be computed for any polynomial $f$. The question of extending $L[f]$ to $\mathcal{F} = \mathcal{L}_2([0, N])$, the space of square-integrable functions over $[0, N]$, is a little more delicate. The polynomials are dense in $\mathcal{F}$, and the natural definition $L[f]$ for $f \in \mathcal{F}$ is $\lim_{n\to\infty} L[p_n]$, where

$p_n$ converges uniformly to $f$. This definition, however, requires uniform continuity of $L$ over $\mathcal{P}$, which we have not established in the case where the mask does not define a refinable function.

## 4. Algorithmic implementation

One can program the whole algorithm in a transparent way with the aid of two subroutines:

`mulx(f,a,b)`: Calculate $e_1 f$.
*Input.* Vector `f` of coefficients in an orthogonal expansion of the form (3.5); vectors `a`,`b` of recursion coefficients.
*Output.* Vector of expansion coefficients of $e_1 f$, calculated using (3.11). This vector has one component more than `f`.

`dot(f1,f2,nu)`: Evaluate the "dot product" $L[f_1 f_2]$.
*Input.* Vectors `f1`, `f2` of coefficients in an orthogonal expansion; vector `nu` of "norms" $L[p_j^2]$.
*Output.* $L[f_1 f_2]$, calculated using (3.6), (3.7).

In the case where the mask is symmetric (i.e., $\gamma_j = \gamma_{N-j},\ j = 0, 1, \ldots, N$), the algorithm can be simplified in two ways:

- The polynomials $p_{j,k}$ need not be generated when $j > N - j$, since in the symmetric case $L[p_{j,k} p_{j,k}] = L[p_{N-j,k} p_{N-j,k}]$.
- The computation of $L[p_k^2]$ can be bypassed, since $a_k = N/2$.

Our implementation in the algorithmic language of the open-source package Pari-GP [13] is given in Figure 5.1. Although Pari-GP has an uncluttered syntax and is very close to standard mathematical notation, there are some aspects that may make the code difficult to read for nonexperts, so at the risk of tautology we describe the algorithm below in more familiar notation.

`ab_refinable(n,g)`: Compute recursion coefficients from a mask.
*Input.* Mask coefficients `g` $= (\gamma_0, \gamma_1, \ldots, \gamma_N)$; number `n` of pairs of recursion coefficients desired.
*Output.* `a` $= (a_0, a_1, \ldots, a_{n-1})$ and `b` $= (b_0, b_1, \ldots, b_{n-1})$.

*Working variables.*

| | |
|---|---|
| `symm` | Boolean variable denoting whether `g` is symmetric. |
| `M` | Number of polynomials $p_{j,k}$ actually needed. |
| `nu` | $\nu = (\nu_0, \nu_1, \ldots, \nu_{n-1})$, with $\nu_k = L[p_k, p_k]$. |
| `p0` | Vector of polynomials $p_{j,k-1},\ j = 0, 1, \ldots, M$. |
| `p1` | Vector of polynomials $p_{j,k},\ j = 0, 1, \ldots, M$. |
| `p2` | Vector of polynomials $p_{j,k+1},\ j = 0, 1, \ldots, M$. |
| `ckk` | Current value of $c_{j,k,k}$. |

*Outline of algorithm.* Readers who wish to compare the following description with the code should bear in mind that Pari-GP vectors all start at index 1, whereas the mathematical description would correspond more closely to vectors that start at index 0. Thus `j` $= j + 1$, `k` $= k + 1$, etc.

(1) $a$, $b$ and $\nu$ are initialized to all zeros; the mask is normalized to ensure that its coefficients add up to 2; a test for symmetry is made; $p_{j,-1}$ and $p_{j,0}$ are initialized to 0 and 1, respectively; $c_{j,k,k}$ is initialized to 1.
The following steps are then carried out for $k = 0, 1, \ldots, n-1$. During each iteration, `ckk` $= 2^{-k}$.
(2) If $k = 0$, $\nu_0 = b_0 = 1$. Otherwise, $\nu_k$ is computed by (3.9), exploiting symmetry if present, and $b_k$ by (2.2).
(3) $\tilde{p}_{j,k+1} = q(0)$ is computed using (3.11)—this step involves `mulx`—and is stored in `p2`.
(4) Depending on the presence of symmetry, $a_k$ is either simply assigned, or $L[\tilde{p}_{j,k+1}p_k]$ is computed by (3.12) and used without being stored to compute $a_k$ by (2.3). Quit if all required coefficients have now been computed.
(5) $(e_1 - j)p_k$ is calculated by adding $2^{-k}a_k p_k$ to `p2`.
(6) $p_{j,k+1}$ is calculated using (3.4).
(7) `ckk` is halved.

A subtle point about the programming is that it is not necessary to clutter the code by explicitly carrying out the subtraction in (3.8). Since $\nu_k$ has been initialized to 0, the first call to `dot` computes $\langle p_k, p_k\rangle_{k-1}$, not $\langle p_k, p_k\rangle_k$. Similarly, since $a_k$ has been initialized to 0, the call to `mulx` leads directly to the computation of $q(0)$.

The complexity of the algorithm is $O(Mn^2)$, which for a fixed mask reduces to $O(n^2)$. This can be seen from the facts that steps 2 to 7 are carried out $n$ times, and that equations (3.9), (3.11), (3.12), (3.4) all involve $O(n)$ operations per mask coefficient.

## 5. Numerical examples

*The purpose of computing is insight, not numbers.* — Richard Hamming

In this section we present a few numerical experiments, in order to give the reader a feel of how the algorithm performs and to suggest what future work on the numerical stability of the algorithm is required.

5.1. **The simplest possible case.** When $n = 1$, the mask has the form $[\gamma, 2-\gamma]$. The case $\gamma = 1$ corresponds to the Legendre weight shifted to $[0, 1]$, i.e., $L[f] = \int_0^1 f(x)\,\mathrm{d}x$. We give this example to illustrate that our algorithm is a finite rational algorithm, capable in principle of giving exact results.

Here is a Pari-GP session, omitting the functions defined in Figure 5.1. We first leave $\gamma$ as an algebraic monomial, calculating only three coefficients in order not to overwhelm the reader with many lines of output. Although only the case $\gamma = 1$ corresponds to a functional of the form (1.4), there is no problem in computing the recursion coefficients for any $\gamma \in (0, 2)$.

Next we put $\gamma = 1$ and calculate a few more coefficients, so the the well-known recursion coefficients of the Legendre polynomials, shifted to $[0, 1]$, can be seen to make their appearance.

```
? ab_refinable(3,[g,2-g])
%2 = [[-1/2*g + 1, -1/14*g + 4/7,
(68339*g^3+92056*g^2-409072*g-144744)/
(594146*g^2-1188292*g-192696)],
 [1, -1/12*g^2 + 1/6*g, -37/735*g^2 + 74/735*g + 4/245]]
? ab_refinable(5,[1,1])
%3 = [[1/2, 1/2, 1/2, 1/2, 1/2], [1, 1/12, 1/15, 9/140, 4/63]]
```

```
{mulx(f,a,b)= local(n); n=length(f);
 vector(n+1,k, if(k>1,f[k-1]) + if(k<n,b[k+1]*f[k+1]) +
               if(k<=n,a[k]*f[k]))}

dot(f1,f2,nu)= sum(k=1,min(length(f1),length(f2)),f1[k]*f2[k]*nu[k])

{ab_refinable(n,g)= local(a,b,ckk,p0,p1,p2,nu,N,M,symm);
 N=length(g); a=b=nu=vector(n); g=2*g/sum(j=1,N, g[j]);
 symm=1; for(i=1,floor(N/2), if(g[i]!=g[N+1-i], symm=0; break));
 M=if(symm, floor((N+1)/2), N);
 p1=vector(M,j,[]); p2=vector(M,j,[1]); ckk=1;
 for(k=1,n, p0=p1; p1=p2;
   if(k==1,  nu[k]=b[k]=1, \\ else
     nu[k]=sum(j=1,M,
       (1+(symm&&j<=N-j))*g[j]*dot(p1[j],p1[j],nu))/(2*(1-ckk^2));
     b[k]=nu[k]/nu[k-1] );
   for(j=1,M, p2[j]=mulx(p1[j],a,b)+(j-1)*concat(p1[j],0) );
   a[k]=if(symm, (N-1)/2, \\ else
     sum(j=1,M, g[j]*dot(p1[j],p2[j],nu))/(4*nu[k]*(1-ckk^2/2)));
   if(k==n, break);
   for(j=1,M, p2[j][k]=p2[j][k]+ckk*a[k];
     p2[j] = p2[j]/2 - a[k]*concat(p1[j],0)
             - b[k]*concat(p0[j],[0,0]) );
   ckk=ckk/2 );
 [a,b]}

{addhelp(mulx,"mulx(f,a,b): multiply orthogonal expansion f"
  " with recursion coefficients [a,b] by x");}
{addhelp(dot, "dot(f1,f2,nu): dot product of f1 and f2 with"
  " respect to weights nu");}
{addhelp(ab_refinable, "ab_refinable(n,g): n recursion coefficients"
  " [a,b] for the refinable linear functional with mask g.");}
```

FIGURE 5.1. Pari-GP code for the two-term recursion coefficients generated by the mask of a refinable linear functional.

5.2. **Experimental investigation of numerical stability.** A proof of numerical stability is outside the scope of the present article, which is concerned only with the algebra of the algorithm. A straightforward approach, based on bounding the condition of the map from the system $\{p_k,\ k = 0, 1, \ldots, n\}$ to each system $\{p_{j,k},\ k = 0, 1, \ldots, n\}$ separately will not suffice to prove stability, since those maps are exponentially ill-conditioned [3] because the two systems have different intervals of support. Before a delicate proof, taking into account the overall map, is attempted, one would like to have some assurance that the computational method presented here has been observed to be stable.

We calculated the recursion coefficients up to $k = 49$ for $\boldsymbol{\gamma}$ proportional to $[1, 3, 3, 1]$ (this corresponds to integration with a cubic cardinal B-spline as weight) in two ways: exact rational arithmetic and the default floating-point arithmetic

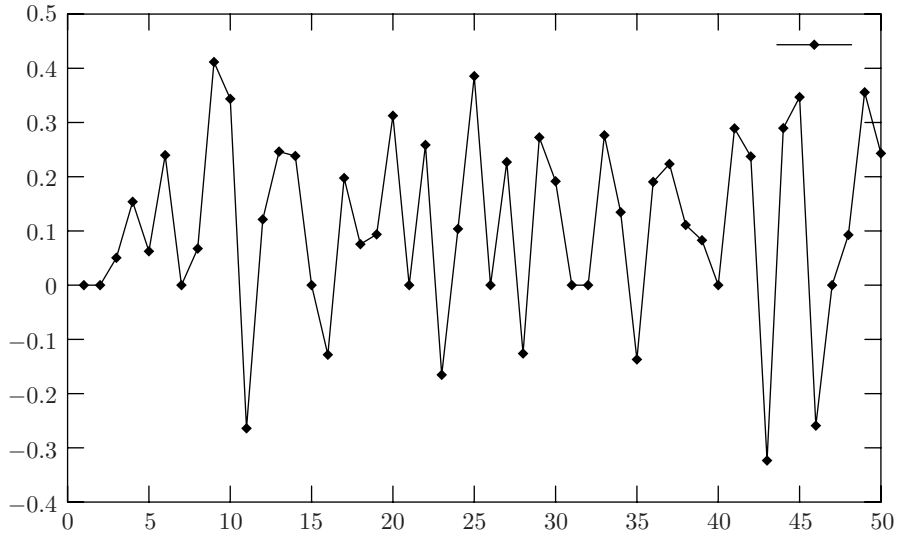


FIGURE 5.2. Error, multiplied by $10^{28}$ in the floating-point computation of the coefficients $b_{k-1}$, plotted against $k$, for the case of a symmetric mask.

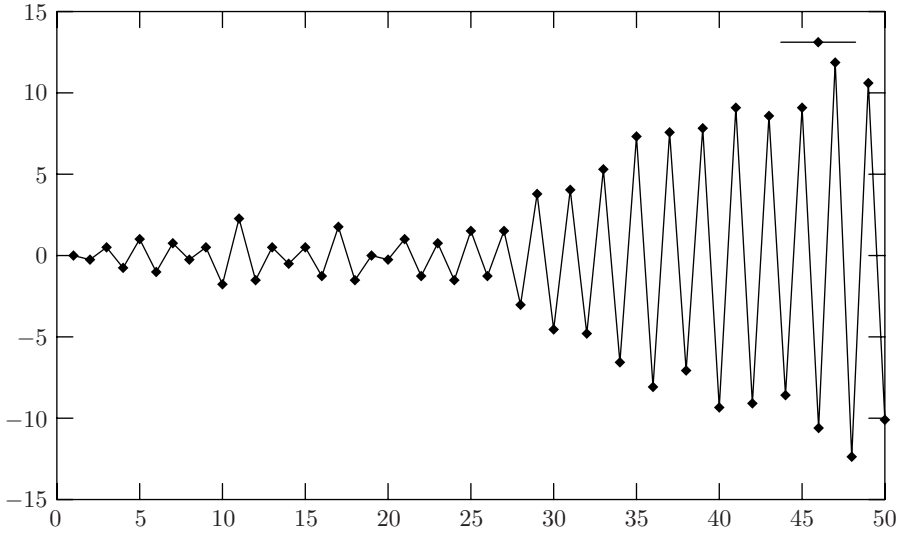


FIGURE 5.3. Error, multiplied by $10^{28}$, in the floating-point computation of the coefficients $a_{k-1}$, plotted against $k$, for the case of an unsymmetric mask.

(three mantissa words, approximately 28 digits of precision) of Pari-GP. The $a_k$ values of this symmetric mask are of course exact. The absolute error in the calculated $b_k$ values is plotted in Figure 5.2. There seems to be no tendency for the error to increase with $k$.

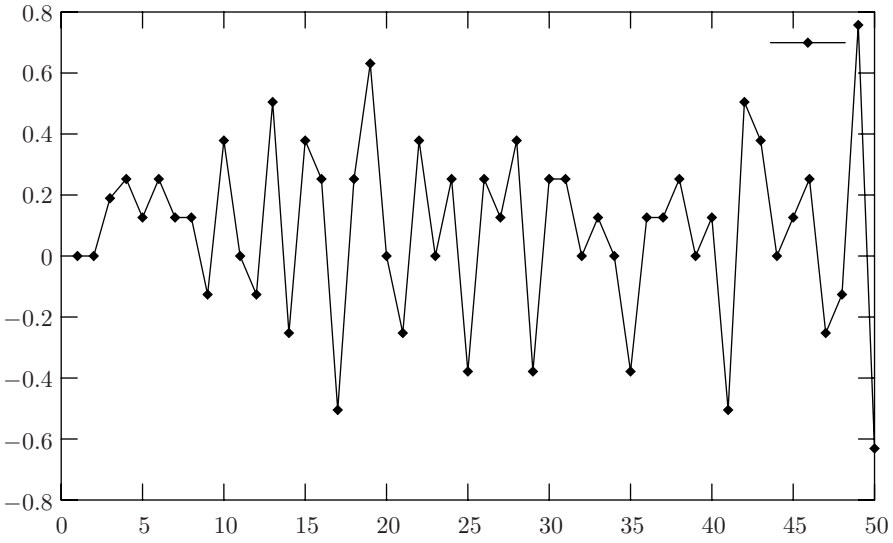


FIGURE 5.4. Error, multiplied by $10^{28}$, in the floating-point computation of the coefficients $b_{k-1}$, plotted against $k$, for the case of an unsymmetric mask.

Next, we repeated the experiment for $\boldsymbol{\gamma}$ proportional to $[1, 1, 3, 3]$. Exact rational arithmetic is in this case not feasible for so many coefficients, since the integers in the numerator and denominator grow very fast. For example, the numerator of $b_{19}$ has 3336 digits. Instead, we made two floating-point calculations: once as before with three mantissa words, and once with six mantissa words, approximately 57 digits of precision. The absolute difference between the calculated $a_k$ and $b_k$ values is plotted in Figures 5.3 and 5.4. The $b_k$ values seem to behave almost as well as in the symmetric case, but the error in the $a_k$ values appears to increase linearly with $k$, and what is more disconcerting, obviously oscillates.

## 6. Conclusions

We have presented an algorithm for obtaining in $O(n^2)$ rational operations the three-recursion coefficients of a refinable function by using only the mask coefficients, and without the aid of modified moments. Our result implies the existence of the corresponding refinable functional over the space of all polynomials whenever the mask coefficients are nonnegative, even when the same mask does not define a refinable function. The algorithm can, in principle, deliver exact results, but numerical evidence indicates that it is also effective in floating-point arithmetic, although an error in the computed $a_k$ values that appears to be roughly proportional to $(-1)^k k$ has been observed when the mask is unsymmetric.

Further investigation is needed on the stability of the algorithm. The numerical experiments reported here are suggestive but not conclusive, and some solid theoretical work is necessary. It would be interesting to know whether the observed mild instability in the unsymmetric case is an artifact of the algorithm, or whether

it arises because the condition number of the problem of computing $n$ pairs of recursion coefficients from the mask is itself proportional to $n$. Also, since the oscillating error behaves so regularly, there may be a way of damping it.

## Acknowledgments

The comments of two anonymous referees led to improvements in the presentation of our results.

## References

1. A. Barinka, T. Barsch, S. Dahlke, and M. Konik. Some remarks on quadrature formulas for refinable functions and wavelets. *Z. Angew. Math. Mech.*, 81(12):839–855, 2001. MR1872770 (2002j:65034)
2. Arne Barinka, Titus Barsch, Stephan Dahlke, Mario Mommer, and Michael Konik. Quadrature formulas for refinable functions and wavelets. II. Error analysis. *J. Comput. Anal. Appl.*, 4(4):339–361, 2002. MR1933926 (2003h:65024)
3. B. Beckermann and E. Bourreau. How to choose modified moments? *J. Computat. Appl. Math*, 98:81–98, 1998. MR1656990 (99g:65024)
4. T. S. Chihara. *An introduction to orthogonal polynomials*. Gordon and Breach Science Publishers, New York, 1978. Mathematics and its Applications, Vol. 13. MR481884 (58:1979)
5. Carl de Boor. *A Practical Guide to Splines*. Springer, New York, 1978. MR0507062 (80a:65027)
6. Walter Gautschi. Questions of numerical condition related to polynomials. In Gene H. Golub, editor, *Studies in Numerical Analysis*, pages 140–177. Math. Assoc. Amer., 1984. MR0925213
7. Walter Gautschi. *Orthogonal polynomials: computation and approximation*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2004. Oxford Science Publications. MR2061539 (2005e:42001)
8. Walter Gautschi, Laura Gori, and Francesca Pitolli. Gauss quadrature for refinable weight functions. *Applied and Computational Harmonic Analysis*, 8:249–257, 2000. MR1754926 (2002a:65049)
9. G.H. Golub and J.H. Welsch. Calculation of Gauss quadrature rules. *Math. Comput.*, 23:221–230, 1969. MR1754926 (2002a:65049)
10. Daan Huybrechs and Stefan Vandewalle. Composite quadrature formulae for the approximation of wavelet coefficients of piecewise smooth and singular functions. *J. Comput. Appl. Math.*, 180:119–135, 2005. MR2141488 (2006c:65030)
11. Dirk Laurie and Johan de Villiers. Orthogonal polynomials and Gaussian quadrature for refinable weight functions. *Applied and Computational Harmonic Analysis*, 17:241–248, 2004. MR2097078 (2005g:42060)
12. C. A. Micchelli. *Mathematical Aspects of Geometric Modeling*. Number 65 in CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, Philadelphia, 1995. MR1308048 (95i:65036)
13. Pari-GP. URL=`http://pari.math.u-bordeaux.fr`. An interactive programming environment for doing formal computations on recursive types, including rational and multiprecision floating-point numbers, polynomials and truncated power series.
14. James L. Phillips and Richard J. Hanson. Gauss quadrature rules with B-spline weight functions. *Math. Comp.*, 28, 1974, loose microfiche suppl., A1–C4. MR0343551 (49:8292)
15. R.A. Sack and A.F. Donovan. An algorithm for Gaussian quadrature given modified moments. *Numer. Math.*, 18:465–478, 1972. MR0303693 (46:2829)
16. A. H. Stroud and D. Secrest. *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, 1966. MR0202312 (34:2185)
17. Wim Sweldens and Robert Piessens. Quadrature formulae and asymptotic error estimates for wavelet approximations of smooth functions. *SIAM J. Numer. Anal.*, 31:1240–1264, 1994. MR1286226 (95e:42043)

Department of Mathematics, University of Stellenbosch, South Africa
*E-mail address*: `dpl@sun.ac.za`

Department of Mathematics, University of Stellenbosch, South Africa
*E-mail address*: `jmdv@sun.ac.za`