Computation of pairs of related Gauss-type quadrature rules

H. Alqahtani[a], C. F. Borges[b], D. Lj. Djukić[c], R. M. Mutavdžić Djukić[c], L. Reichel[d], M. M. Spalević[c]

[a]*Department of Mathematics, Faculty of Science and Arts, King Abdulaziz University, Rabigh, Saudi Arabia*
[b]*Department of Mathematics, Naval Postgraduate School, Monterey, CA 93943, USA*
[c]*Department of Mathematics, University of Belgrade, Faculty of Mechanical Engineering, Kraljice Marije 16, 11120 Belgrade 35, Serbia*
[d]*Department of Mathematical Sciences, Kent State University, Kent, OH 44242, USA*

Abstract

The evaluation of Gauss-type quadrature rules is an important topic in scientific computing. To determine estimates or bounds for the quadrature error of a Gauss rule often another related quadrature rule is evaluated, such as an associated Gauss-Radau or Gauss-Lobatto rule, an anti-Gauss rule, an averaged rule, an optimal averaged rule, or a Gauss-Kronrod rule when the latter exists. We discuss how pairs of a Gauss rule and a related Gauss-type quadrature rule can be computed efficiently by a divide-and-conquer method.

Keywords: divide-and-conquer method, Gauss rule, Gauss-Radau rule, Gauss-Lobatto rule, averaged Gauss rule, optimal averaged Gauss rule
2010 MSC: 65D30, 65D32

1. Introduction

The need to evaluate Gauss-type quadrature rules arises in many applications in science and engineering. Therefore, the computation of such rules occupies a central place in scientific computing. This paper is concerned with the efficient evaluation of Gauss quadrature rules, as well as of some related rules that help bound or estimate the error in the Gauss rules.

Many algorithms have been developed for the fast evaluation of Gauss rules associated with a non-negative measure with support on (parts of) the real axis. The classical algorithm for this purpose is due to Golub and Welsch [19]; it requires $\mathcal{O}(n^2)$ arithmetic floating point operations (flops) to determine the

Email addresses: `hfalqahtani@kau.edu.sa` (H. Alqahtani), `borges@nps.edu` (C. F. Borges), `ddjukic@mas.bg.ac.rs` (D. Lj. Djukić), `rmutavdzic@mas.bg.ac.rs` (R. M. Mutavdžić Djukić), `reichel@math.kent.edu` (L. Reichel), `mspalevic@mas.bg.ac.rs` (M. M. Spalević)

nodes and weights of an n-node Gauss-rule. More recently, Bogaert [3] as well as Hale and Townsend [21] described algorithms that only require $\mathcal{O}(n)$ flops to compute the nodes and weights of n-node Gauss-Legendre and Gauss-Jacobi quadrature rules. These methods are competitive when n is large, but are restricted to the Legendre measure, $\mathrm{d}w(t) = dt$ for $-1 < t < 1$, see [3], or a Jacobi measure $\mathrm{d}w(t) = (1-t)^{\alpha}(1+t)^{\beta}dt$ for $-1 < t < 1$ and $\alpha, \beta > -1$; see [21]. These algorithms are faster than the algorithm by Glaser et al. [15], which also requires only $\mathcal{O}(n)$ flops, but the latter algorithm can be applied to a larger class of classical measures, including Legendre, Jacobi, Hermite, and Laguerre measures.

It is the purpose of the present paper to revisit the problem of evaluating Gauss-type quadrature rules. We are interested in being able to compute nodes and weights for general non-negative non-classical measures with support on (part of) the real axis. The reasons for our interest are two-fold: there is presently no public-domain implementation of the Golub-Welsch algorithm that requires $\mathcal{O}(n^2)$ flops to a determine the nodes and weights of an n-node Gauss rule. For instance, the implementation provided by Gautschi [13] demands $\mathcal{O}(n^3)$ flops. Moreover, it is often important to be able to determine an estimate for the quadrature error of an n-node Gauss rule in order to assess whether the chosen number of nodes is sufficiently large to yield an approximation of the desired integral of required accuracy. A too small value of n gives a too large quadrature error, while a too large value results in an unnecessarily large computational burden, in particular if the integrand is expensive to evaluate.

Several approaches have been proposed to estimate the quadrature error of an n-node Gauss rule. The classical approach is to evaluate a $(2n+1)$-node Gauss-Kronrod rule that is associated with the n-node Gauss rule. However, Gauss-Kronrod rules are not guaranteed to have all nodes in the convex hull of the support of the measure that defines the Gauss rule; in particular, some Gauss-Kronrod nodes might have pairs of complex conjugate nodes away from the real axis; see Notaris [24] for a nice survey of Gauss-Kronrod rules as well as computed examples in [1]. When the Gauss-Kronrod rule has nodes in the complex plane, the integrand has to be defined in a sufficiently large domain in the complex plane that contains the support of the measure in order for the Gauss-Kronrod rule to be applicable. This limits the applicability of these rules.

This shortcoming of Gauss-Kronrod rules has lead to the development of several alternative approaches to estimate the quadrature error of Gauss rules. These approaches include the evaluations pairs of

(i) an n-node Gauss and an associated $(n+1)$-node Gauss-Radau or an associated $(n+2)$-node Gauss-Lobatto rule. Pairs of Gauss and Gauss-Radau or pairs of Gauss and Gauss-Lobatto quadrature rules give upper and lower bounds for the integral under suitable conditions on the integrand and an appropriate choice of the user-specified Radau node. This follows from the remainder terms for Gauss and Gauss-Radau quadrature; see, e.g., Golub and Meurant [17] as well as Gautschi [14]. Note that Gauss-Radau and Gauss-Lobatto rules associated with an n-node Gauss rule are guaranteed to exist and to have real nodes.

(ii) an n-node Gauss and an associated $(n+1)$-node anti-Gauss rule. Pairs of these rules yield upper and lower bounds for the integral if the integrand is sufficiently smooth; see [8], as well as [2] a for related method. Anti-Gauss rules were introduced by Laurie [22]. They are modifications of Gauss rules. In particular, the nodes of the n-node Gauss rule interlace the nodes of the associated $(n+1)$-node anti-Gauss rule. Anti-Gauss rules always exist and have real nodes, however, for some measures that define the Gauss rule, the associated anti-Gauss rule may have a node or two outside the convex hull of the support of the measure; see [22] for details.

(iii) Gauss and averaged Gauss rules. The difference between an n-node Gauss rule and the associated $(2n+1)$-node averaged Gauss rule provides an estimate for the quadrature error. The latter rule is the average of the n-node Gauss rule and the associated $(n+1)$-node anti-Gauss rule. These averaged rules were introduced by Laurie [22]. Thus, the computation of an n-node Gauss rule and the associated averaged rule demands the evaluation of the Gauss rule and the associated $(n+1)$-node anti-Gauss rule. The averaged rule might have a node or two outside the convex hull of the support of the measure.

(iv) Gauss and optimal averaged Gauss rules. The difference between an n-node Gauss rule and the associated $(2n+1)$-node optimal averaged Gauss rule provides an estimate for the quadrature error. The latter rules were introduced by Spalević [27] and have for smooth enough integrands a higher degree of precision than the averaged Gauss rule defined by Laurie [22]. Optimal averaged Gauss rules associated with an n-node Gauss rule can be expressed as a weighted sum of the n-node Gauss rule and a related $(n+1)$-node Gauss-type quadrature rule; see [25]. The optimal averaged Gauss rules always exist and have real nodes, but similarly as for the averaged Gauss rules, they may for certain measures have one or two nodes outside the convex hull of the support of the measure that defines the Gauss rule. Analyses of the location of the nodes for optimal averaged rules as well as for averaged rules for a variety of classical and other measures can be found in [10, 11, 12] and references therein.

In all the approaches (i)-(iv), both the n-node Gauss rule and an associated $(n+1)$-node Gauss-type quadrature rule have to be evaluated. This requires two applications of the Golub-Welsch algorithm. We propose to use a divide-and-conquer method to evaluate the n-node Gauss rule and one of the associated Gauss-type quadrature rules mentioned above. The flop count for the evaluation of each one of these quadrature rules by the divide-and-conquer method described is $\mathcal{O}(n^2)$ flops, and the application of this method is faster than the use of the Golub-Welsch algorithm; see Section 5 for timings. Parallel implementation of divide-and-conquer methods is quite straightforward and this makes it possible to use these methods to evaluate quadrature rules with very many nodes.

We will not discuss the evaluation of pairs of an n-node Gauss rule and the associated $(2n+1)$-node Gauss-Kronrod rule because, as mentioned above, the latter rules are not guaranteed to have real nodes. Moreover, Gauss-Kronrod rules are more complicated to compute than the rules listed above; see [1, 7, 23] for algorithms for evaluating Gauss-Kronrod rules.

This paper is organized as follows. Section 2 describes Gauss quadrature rules and how a divide-and-conquer method can be applied to their evaluation. In Section 3, we describe Gauss-Radau, Gauss-Lobatto, anti-Gauss, as well as averaged and optimal averaged Gauss rules, and discuss their efficient computation by a divide-and-conquer method. Section 4 is concerned with the deflation process of the divide-and-conquer method. Computed examples are presented in Section 5 and concluding remarks can be found in Section 6.

2. A divide-and-conquer method applied to the evaluation of Gauss quadrature rules

Let $\mathrm{d}w$ be a non-negative measure on (part of) the real axis with infinitely many points of support and such that all moments $\mu_k = \int t^k \mathrm{d}w(t)$, $k = 0, 1, \dots$, are well defined. For notational simplicity, we assume that $\mu_0 = 1$. We are interested in the approximation of integrals of the form

$$\mathcal{I}(f) = \int f(t)\mathrm{d}w(t) \tag{1}$$

by an n-node Gauss quadrature rule

$$\mathcal{G}_n(f) = \sum_{j=1}^{n} f(t_j)w_j. \tag{2}$$

Gauss rules are related to monic orthogonal polynomials $p_0, p_1, \dots$ determined by the inner product

$$\langle f, g\rangle = \int f(t)g(t)\mathrm{d}w(t).$$

Hence, the polynomials p_j of degree j have leading coefficient one and satisfy

$$\langle p_i, p_j\rangle \begin{cases} > 0, & i = j, \\ = 0, & i \neq j. \end{cases}$$

We have $p_0(x) \equiv 1$, and it is convenient to define $p_{-1}(x) \equiv 0$. It is well known that the polynomials p_k satisfy a three-term recurrence relation of the form

$$p_{j+1}(t) = (t - \alpha_j)p_j(t) - \beta_j p_{j-1}(t), \quad j = 0, 1, \dots ,$$

where the coefficients α_j and β_j are given by

$$\begin{aligned} \alpha_j &= \frac{\langle tp_j, p_j\rangle}{\langle p_j, p_j\rangle}, \quad j = 0, 1, \dots , \\ \beta_j &= \frac{\langle p_j, p_j\rangle}{\langle p_{j-1}, p_{j-1}\rangle}, \quad j = 1, 2, \dots , \end{aligned}$$

and $\beta_0 = 1$. All β_j are known to be positive. The polynomial p_n has n distinct real zeros, all of which are in the convex hull of the support of $\mathrm{d}w$. The nodes $t_1, t_2, \ldots, t_n$ of the Gauss rule (2) are the zeros of p_n and the weights $w_1, w_2, \ldots, w_n$ are positive; see, e.g., [14, 28] for proofs.

Among all interpolatory quadrature rules with n nodes for the approximation of the integral (1), the Gauss rule (2) has maximal degree of precision, $2n-1$, i.e.,

$$\mathcal{G}_n(p) = \mathcal{I}(p), \qquad \forall p \in \mathbb{P}_{2n-1},$$

where $\mathbb{P}_{2n-1}$ denotes the set of all polynomials of degree at most $2n-1$; see [14, 28] for proofs. This makes Gauss quadrature rules well suited for the approximation of many integrals of the form (1).

The Gauss rule (2) can be associated with the symmetric tridiagonal matrix

$$T_n = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\beta_{n-2}} & \alpha_{n-2} & \sqrt{\beta_{n-1}} \\ 0 & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{3}$$

which is unreduced, i.e., all subdiagonal entries are positive. Therefore, its eigenvalues are real and distinct; they are the nodes $t_1, t_2, \ldots, t_n$ of the Gauss rule. Moreover, the weights $w_1, w_2, \ldots, w_n$ of the Gauss rule are the square of the first component of normalized eigenvectors.

This section discusses the computation of the nodes and weights of the Gauss quadrature rule (2) by a divide-and-conquer method applied to the matrix (3). Algorithms for divide-and-conquer methods have been described by Borges and Gragg [6] as well as by Gu and Eisenstat [20]. They consist of a divide phase and a conquer phase. We will outline these phases in order and use the notation in [6], where further details can be found. Thus, introduce the spectral factorization of the matrix (3),

$$T_n = U_n \Lambda_n U_n^T, \tag{4}$$

where the matrix $U_n \in \mathbb{R}^{n \times n}$ is orthogonal; its columns are eigenvectors of T_n and the superscript ^{T} denotes transposition. The diagonal entries of

$$\Lambda_n = \mathrm{diag}[\lambda_1, \lambda_2, \ldots, \lambda_n] \in \mathbb{R}^{n \times n}$$

are the eigenvalues of (3). We would like to compute these eigenvalues, which are the nodes of the rule (2), and the first components of the eigenvectors, from which the weights w_j in (2) can be determined. The divide-and-conquer method to be described yields the eigenvalues, and the first and last components of normalized eigenvectors. We will refer to a decomposition of the matrix T_n that consists of the eigenvalues, and first and last components of the eigenvectors, as a *partial spectral factorization* of T_n.

Let $e_k = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ denote the kth axis vector of appropriate dimension for $k = 1, 2, \ldots$, and introduce the split index s for some $1 \leq s \leq n$. Consider the following block form of the matrix (3),

$$T_n = \begin{bmatrix} \breve{T}_1 & e_{s-1}\sqrt{\beta_{s-1}} & 0 \\ e_{s-1}^T\sqrt{\beta_{s-1}} & \alpha_{s-1} & e_1^T\sqrt{\beta_s} \\ 0 & e_1\sqrt{\beta_s} & \breve{T}_2 \end{bmatrix},$$

where $\breve{T}_1$ is the leading principal $(s-1)\times(s-1)$ submatrix of T_n and $\breve{T}_2$ is the trailing principal $(n-s)\times(n-s)$ submatrix. Our discussion below assumes that $1 < s < n$, but the values $s = 1$ and $s = n$ are permitted; in the former case the matrix $\breve{T}_1$ is empty, and in the latter case $\breve{T}_2$ is empty.

Assume for the moment that the smaller eigenvalue problems

$$\breve{T}_k = \breve{U}_k\breve{\Lambda}_k\breve{U}_k^T, \qquad k = 1, 2, \tag{5}$$

have been solved. Here $\breve{U}_k$ is an orthogonal matrix whose columns are eigenvectors of $\breve{T}_k$, and $\breve{\Lambda}_k$ is a diagonal matrix whose diagonal entries are the eigenvalues. Define the orthogonal block diagonal matrix

$$\widehat{U}_n = \begin{bmatrix} \breve{U}_1 & & 0 \\ & 1 & \\ 0 & & \breve{U}_2 \end{bmatrix} \in \mathbb{R}^{n\times n},$$

where "1" denotes the scalar one. Letting $u_1 = \breve{U}_1^T e_s \in \mathbb{R}^{s-1}$ and $u_2 = \breve{U}_2^T e_1 \in \mathbb{R}^{n-s}$, we obtain

$$\widehat{U}_n^T T_n \widehat{U}_n = \begin{bmatrix} \breve{\Lambda}_1 & u_1\sqrt{\beta_{s-1}} & 0 \\ u_1^T\sqrt{\beta_{s-1}} & \alpha_{s-1} & u_2^T\sqrt{\beta_s} \\ 0 & u_2\sqrt{\beta_s} & \breve{\Lambda}_2 \end{bmatrix}. \tag{6}$$

Note that only the last row of the matrix $\breve{U}_1$ and the first row of $\breve{U}_2$ are required to determine the vectors u_1 and u_2. This observation is important for the fast computation of the nodes and weights of the quadrature rule (2). We will return to the computation of this quadrature rule below.

The matrix (6) is the sum of a diagonal matrix and a "cross". It is convenient to permute the rows and columns symmetrically so that the "cross" is moved to the last row and column. Thus, define the permutation matrix

$$\widehat{P}_n = [e_1, e_2, \ldots, e_{s-1}, e_{s+1}, e_{s+2}, \ldots, e_n, e_s] \in \mathbb{R}^{n\times n}.$$

Then

$$\widehat{P}_n^T\widehat{U}_n^T T_n \widehat{U}_n \widehat{P}_n = \begin{bmatrix} \breve{\Lambda}_1 & 0 & u_1\sqrt{\beta_{s-1}} \\ 0 & \breve{\Lambda}_2 & u_2\sqrt{\beta_s} \\ u_1^T\sqrt{\beta_{s-1}} & u_2^T\sqrt{\beta_s} & \alpha_{s-1} \end{bmatrix} \tag{7}$$

is an arrow matrix. There are efficient methods for computing the spectrum of an arrow matrix. Such methods are discussed by Borges and Gragg [6], and Gu and Eisenstat [20]. We will discuss the former method in more detail below. Here we only note that the eigenvectors of an arrow matrix have a simple form. For notational simplicity, consider the arrow matrix

$$A = \begin{bmatrix} D & c \\ c^T & \alpha \end{bmatrix}, \tag{8}$$

where $D = \mathrm{diag}[d_1, d_2, \ldots, d_m] \in \mathbb{R}^{m\times m}$ and $c = [\gamma_1, \gamma_2, \ldots, \gamma_m]^T \in \mathbb{R}^m$. If any entry γ_j vanishes, then e_j is an eigenvector and we may set $\lambda_j = d_j$. Borges and Gragg [6] refer to this as γ-deflation. We comment on γ-deflation further in Section 4. Assume that all possible deflations of this kind have been carried out and let (8) be the resulting matrix. Thus, all the entries γ_j of this matrix are non-vanishing. Let λ_j be an eigenvalue of (8). Then

$$v_j = \begin{bmatrix} (\lambda_j I - D)^{-1} c \\ 1 \end{bmatrix} \tag{9}$$

is an eigenvector of A; see [6]. Normalization yields orthonormal eigenvectors.

We turn to the computation of the Gauss rule (2). The following proposition shows that only the first row of the matrix $\breve{U}_1$, and the eigenvalues of (6), part of the matrix made up of normalized eigenvectors defined by (9), are required to determine the nodes and weights of the Gauss rule (2).

Proposition 1. *Only the first row of the matrix $\breve{U}_1$ and the first s components of each normalized eigenvector* (9) *are needed to compute the nodes and weights of the quadrature rule* (2).

Proof. Consider the spectral factorization of the matrix (7). We have

$$\widehat{P}_n^T \widehat{U}_n^T T_n \widehat{U}_n \widehat{P}_n = \widehat{W}_n \Lambda_n \widehat{W}_n^T, \tag{10}$$

where the matrix $W_n \in \mathbb{R}^{n\times n}$ is orthogonal; it is obtained by normalizing the eigenvectors (9). It follows from (4) and (10) that

$$U_n = \widehat{U}_n \widehat{P}_n \widehat{W}_n.$$

Therefore,

$$e_1^T U_n = [e_1^T \breve{U}_1, 0_{n-s+1}^T] \widehat{P}_n \widehat{W}_n = [e_1^T \breve{U}_1, 0_{n-s+1}^T] \widehat{W}_n, \tag{11}$$

where $0_j \in \mathbb{R}^j$ denotes the zero vector. The right-hand side of (11) can be evaluated by computing the first $s-1$ components of the normalized eigenvectors (9). □

The proposition relies of the fact that only the first row of the matrix U_n is required to determine the Gaussian weights. A more detailed analysis shows that only the first and last components of the eigenvector matrices $\breve{U}_1$ and $\breve{U}_2$, as well as the eigenvalues of the matrices $\breve{T}_1$ and $\breve{T}_2$, and of their subdivisions, are required to carry out the computations with the divide-and-conquer method and, in particular, to compute the nodes and weights of the Gauss rule (2). In detail, each eigenvector can be evaluated in $\mathcal{O}(n)$ flops. Therefore, the left-hand side of (11) can be computed in $\mathcal{O}(n^2)$ flops by evaluating $e_1^T \breve{U}_1$ first. The divide-and-conquer method splits the eigenproblems for the matrices (5) similarly as we split the eigenproblem for (4). Splittings are applied recursively until only small eigenproblems (whose sizes are independent of n) remain. When the splitting is carried out so that each subproblem is of roughly the same small size, e.g., the subproblems may be of order one, two, or three, and only partial spectral factoriztions are computed, one obtains an algorithm that demands only $\mathcal{O}(n^2)$ flops to compute the nodes and weights of the Gauss rule (2); see [6] for details on the divide-and-conquer method.

3. Divide-and-conquer methods for pairs of related Gauss-type quadrature rules

This section describes how the divide-and-conquer method outlined in Section 2 can be applied to evaluate pairs of two related quadrature rules, one of which is the Gauss rule (2).

3.1. Computation of pairs of Gauss and Gauss-Radau rules

Assume that the integrand f in (1) is $2n+1$ times continuously differentiable and that the support of the measure $\mathrm{d}w$ is contained in the bounded real interval $[a, b]$. If the $2n$th and $(2n+1)$st derivatives of f are of constant sign, then the user-specified node $\widetilde{t}_0$ in an $(n+1)$-nodes Gauss-Radau rule

$$\mathcal{R}_{n+1}(f) = \sum_{j=0}^{n} f(\widetilde{t}_j)\widetilde{w}_j \tag{12}$$

can be chosen so that the quadrature errors of the Gauss and Gauss-Radau rules are of opposite sign. The values of the Gauss rule (2) and the associated Gauss-Radau rule (12) then bracket the value of the integral (1). This follows from the remainder terms for Gauss and Gauss-Radau quadrature rules; see Golub and Meurant [17] for details. It is therefore of interest to compute the nodes and weights of both the rules (2) and (12). The latter rule can be associated with the symmetric tridiagonal matrix

$$\widetilde{T}_{n+1} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \sqrt{\beta_{n-2}} & \alpha_{n-2} & \sqrt{\beta_{n-1}} & & \\ & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{\beta_n} \\ 0 & & & & \sqrt{\beta_n} & \widetilde{\alpha}_n \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \tag{13}$$

The entry $\widetilde{\alpha}_n$ is chosen so that the Radau node $\widetilde{t}_0$ is allocated at the desired location, which usually is $\widetilde{t}_0 = a$ or $\widetilde{t}_0 = b$; expressions for $\widetilde{\alpha}_n$ can be found in [16, 17]. Then the eigenvalues of (13) are the nodes $\widetilde{t}_0, \widetilde{t}_1, \ldots, \widetilde{t}_n$ of the Gauss-Radau rule (12) and the weights $\widetilde{w}_0, \widetilde{w}_1, \ldots, \widetilde{w}_n$ are the square of the first component of normalized eigenvectors of the matrix (13).

We apply the divide-and conquer method to the matrix (13) as follows: first the method is applied to the leading principal submatrix (3) as outlined above. This yields the Gauss rule (2). Then using (4), we obtain

$$\begin{aligned} \widetilde{T}_{n+1} &= \begin{bmatrix} T_n & e_n\sqrt{\beta_n} \\ e_n^T\sqrt{\beta_n} & \widetilde{\alpha}_n \end{bmatrix} \\ &= \begin{bmatrix} U_n & 0_n \\ 0_n^T & 1 \end{bmatrix} \begin{bmatrix} \Lambda_n & \widetilde{u}_n\sqrt{\beta_n} \\ \widetilde{u}_n^T\sqrt{\beta_n} & \widetilde{\alpha_n} \end{bmatrix} \begin{bmatrix} U_n^T & 0_n \\ 0_n^T & 1 \end{bmatrix}, \end{aligned} \tag{14}$$

where $\widetilde{u}_n = U_n^T e_n$. The formation of this vector only requires knowledge of the last row of U_n, which we assume to be available. The middle matrix in (14) is an arrow matrix. The calculation of its partial spectral factorization demands $\mathcal{O}(n^2)$ flops. These quantities allow us to compute the first row of the eigenvector matrix (the square root of the Gauss-Radau weights) and the Gauss-Radau nodes (the eigenvalues of the middle matrix). It follows that the computations of the nodes and weights of the Gauss-Radau rule demand $\mathcal{O}(n^2)$ flops.

We required for notational simplicity that the support of the measure $\mathrm{d}w$ is (part of) a bounded real interval $[a, b]$. This requirement easily can be relaxed to allow the support of the measure to live in a semi-infinite interval $[a, \infty[$, with $a \in \mathbb{R}$ finite, if the integrand $f(t)$ converges to zero sufficiently quickly when $t \to \infty$; one can handle the situation when the support of the measure lives in the semi-infinite interval $]-\infty, b]$ with $b \in \mathbb{R}$ finite in a similar manner.

3.2. Computation of pairs of Gauss and Gauss-Lobatto rules

Assume that the integrand f is $2n+2$ times continuously differentiable on the bounded interval $[a, b]$ that contains the support of the measure $\mathrm{d}w$. Let the $2n$th and $(2n+2)$nd derivatives of f be of constant and opposite signs on $[a, b]$. The user-specified nodes in the Gauss-Lobatto rule

$$\mathcal{L}_{n+2}(f) = \sum_{j=0}^{n+1} f(\widetilde{t}_j)\widetilde{w}_j \tag{15}$$

are commonly chosen as $\widetilde{t}_0 = a$ and $\widetilde{t}_{n+1} = b$. Then the quadrature errors of the Gauss and Gauss-Lobatto rules are of opposite sign. This follows from the remainder terms for Gauss and Gauss-Lobatto quadrature rules; see Golub and Meurant [17] for details. Thus, the Gauss and Gauss-Lobatto rules bracket the value of the integral (1).

The symmetric tridiagonal matrix associated with the $(n+2)$-node Gauss-

Lobatto rule (15) is of the form

$$\widetilde{T}_{n+2} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \sqrt{\beta_{n-2}} & \alpha_{n-2} & \sqrt{\beta_{n-1}} & & & \\ & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{\beta_n} & & \\ & & & & \sqrt{\beta_n} & \alpha_n & \sqrt{\widetilde{\beta}_{n+1}} & \\ 0 & & & & & \sqrt{\widetilde{\beta}_{n+1}} & \widetilde{\alpha}_{n+1} & \end{bmatrix} \in \mathbb{R}^{(n+2)\times(n+2)}, \tag{16}$$

where the entries $\widetilde{\alpha}_{n+1}$ and $\widetilde{\beta}_{n+1}$ are determined so that the nodes $\widetilde{t}_0$ and $\widetilde{t}_{n+1}$ of rule (15) are allocated at specified points, say $\widetilde{t}_0 = a$ and $\widetilde{t}_{n+1} = b$; see [16, 17] for formulas for $\widetilde{\alpha}_{n+1}$ and $\widetilde{\beta}_{n+1}$.

Assume that the partial spectral factorization of the matrix (3) is known. This factorization suffices to determine the Gauss rule (2). Then apply the divide-and-conquer method with $s = n$ to the matrix (16) to determine the Gauss-Lobatto rule, where we use the fact that the first and last rows of the eigenvector matrix of the spectral factorization (4) are known. The flop count for computing both the Gauss and Gauss-Lobatto rules is $\mathcal{O}(n^2)$.

3.3. Computation of pairs of Gauss and anti-Gauss rules

The anti-Gauss rule

$$\mathcal{A}_{n+1}(f) = \sum_{j=1}^{n+1} f(\widetilde{t}_j)\widetilde{w}_j \tag{17}$$

associated with the Gauss rule (2) is characterized by

$$(\mathcal{I} - \mathcal{A}_{n+1})(f) = -(\mathcal{I} - \mathcal{G}_n)(f) \qquad \forall\ \mathbb{P}_{2n+1};$$

see Laurie [22]. Anti-Gauss rules can be applied to estimate the quadrature error. Under suitable conditions, pairs of Gauss and the associated anti-Gauss rule bracket the exact value of the integral (1); see [8].

The tridiagonal matrix associated with the anti-Gauss rule (17) is given by

$$\widetilde{T}_{n+1} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \sqrt{\beta_{n-2}} & \alpha_{n-2} & \sqrt{\beta_{n-1}} & & \\ & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{2}\sqrt{\beta_n} & \\ 0 & & & & \sqrt{2}\sqrt{\beta_n} & \widetilde{\alpha}_n & \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \tag{18}$$

It follows that pairs of Gauss and anti-Gauss rules can be determined similarly as pairs of Gauss and Gauss-Radau rules. A stability analysis of anti-Gauss rules is presented by Díaz de Alba et al. [9]. When the entry $\widetilde{\alpha}_n$ is expensive to compute, simplified anti-Gauss rules can be applied; see [2]. The symmetric tridiagonal matrix associated with an $(n+1)$-node simplified anti-Gauss rule is of the same form as (18). Hence, pairs of Gauss rules and anti-Gauss rules or pairs of Gauss rules and simplified anti-Gauss rules can be evaluated similarly as pairs of Gauss and Gauss-Radau rules. As mentioned above, the average of a Gauss and an associated anti-Gauss rule yields an averaged Gauss rule, which also can be applied to estimate the error in the Gauss rule. Similarly, the average of a Gauss rule and an associated simplified anti-Gauss rule gives a simplified averaged rule, which can be applied to estimate the error in the Gauss rule.

3.4. Computation of pairs of Gauss and optimal averaged Gauss rules

The difference between the values of the Gauss rule (2) and the associated $(2n+1)$-node optimal averaged Gauss quadrature rule

$$S_{2n+1}(f) = \sum_{j=1}^{2n+1} f(\widetilde{t}_j)\widetilde{w}_j \tag{19}$$

generally provides quite accurate estimates of the quadrature error of the Gauss rule; see [25, 26] for illustrations. This suggests that optimal averaged Gauss quadrature rules can be applied to estimate the quadrature error in the Gauss rule.

Introduce the reverse matrix

$$T'_n = \begin{bmatrix} \alpha_{n-1} & \sqrt{\beta_{n-1}} & & & 0 \\ \sqrt{\beta_{n-1}} & \alpha_{n-2} & \sqrt{\beta_{n-2}} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\beta_2} & \alpha_1 & \sqrt{\beta_1} \\ 0 & & & \sqrt{\beta_1} & \alpha_0 \end{bmatrix} \in \mathbb{R}^{n\times n},$$

which is obtained by reversing the order of the rows and columns of the matrix (3). The nodes and weights of the optimal averaged Gauss rule (19) are the eigenvalues and the squared first component of normalized eigenvectors, respectively, of the concatenated symmetric tridiagonal matrix

$$\widehat{T}_{2n+1} = \begin{bmatrix} T_n & \sqrt{\beta_n}e_n & 0 \\ \sqrt{\beta_n}e_n^T & \alpha_n & \sqrt{\beta_{n+1}}e_1^T \\ 0 & \sqrt{\beta_{n+1}}e_1 & T'_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times(2n+1)}; \tag{20}$$

see [27] for details. This matrix can be split into two matrices, one of them being a multiple of the matrix (3); see [25]. However, when using a divide-and-conquer method to determine the nodes and weights of the rule (19), it is beneficial to apply the method to the matrix (20). Introduce the skew-identity $J_n \in \mathbb{R}^{n\times n}$, which has the entries one on the skew-diagonal and all other entries equal to zero. Define the orthogonal block matrix

$$\widehat{U}_{2n+1} = \begin{bmatrix} U_n & & 0 \\ & 1 & \\ 0 & & J_n U_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times(nk+1)},$$

where "1" is a scalar and introduce the permutation matrix

$$\widehat{P}_{2n+1} = [e_1, e_2, \dots, e_n, e_{n+2}, e_{n+3}, \dots, e_{2n+1}, e_{n+1}] \in \mathbb{R}^{(2n+1)\times(2n+1)}.$$

Then

$$\widehat{P}^T_{2n+1}\widehat{U}^T_{2n+1}\widehat{T}_{2n+1}\widehat{U}_{2n+1}\widehat{P}_{2n+1} = \begin{bmatrix} \Lambda_n & 0 & u\sqrt{\beta_n} \\ 0 & \Lambda_n & u\sqrt{\beta_{n+1}} \\ u^T\sqrt{\beta_n} & u^T\sqrt{\beta_{n+1}} & \alpha_n \end{bmatrix}$$

with $u = U_n^T e_n$. We now can annihilate the vector $u\sqrt{\beta_n}$ by orthogonal similarity transformation as follows. This annihilation is known as combo-deflation; see Section 4 below. Define the block Givens rotation

$$\widehat{G}_{2n+1} = \begin{bmatrix} cI_n & sI_n & \\ -sI_n & cI_n & \\ & & 1 \end{bmatrix} \in \mathbb{R}^{(2n+1)\times(2n+1)},$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the identity matrix and

$$c = \frac{\sqrt{\beta_{n+1}}}{\sqrt{\beta_n + \beta_{n+1}}}, \qquad s = \frac{\sqrt{\beta_n}}{\sqrt{\beta_n + \beta_{n+1}}}.$$

Then

$$\widehat{G}^T_{2n+1}\widehat{P}^T_{2n+1}\widehat{U}^T_{2n+1}\widehat{T}_{2n+1}\widehat{U}_{2n+1}\widehat{P}_{2n+1}\widehat{G}_{2n+1} = \begin{bmatrix} \Lambda_n & 0_n & 0_n \\ 0_n^T & \Lambda_n & u\sqrt{\widetilde{\beta}_n} \\ 0_n^T & u^T\sqrt{\widetilde{\beta}_n} & \alpha_n, \end{bmatrix},$$

where $\widetilde{\beta}_n = \beta_n + \beta_{n+1}$. This shows that the eigenproblem for the matrix $\widehat{T}_{2n+1}$ splits into two eigenproblems: the eigenproblems for T_n and the eigenproblem for the trailing $(n+1)\times(n+1)$ arrow submatrix

$$\widetilde{M}_{n+1} = \begin{bmatrix} \Lambda_n & u\sqrt{\widetilde{\beta}_n} \\ u^T\sqrt{\widetilde{\beta}_n} & \alpha_n \end{bmatrix}. \tag{21}$$

Thus, we first compute the partial spectral factorization of the matrix (4) by a divide-and-conquer method, and then determine the eigenvalues and first components of the eigenvectors of the matrix (21) as described in Section 2. This yields the quadrature rules (2) and (19) in $\mathcal{O}(n^2)$ flops.

4. Deflation in the divide-and-conquer method

The computation of the spectral factorization or partial spectral factorization of the matrix (7) may require deflation. Two kinds of deflation will be considered: γ-deflation and combo-deflation.

As mentioned above, the symmetric tridiagonal matrices $\breve{T}_1$ and $\breve{T}_2$, defined by (5), are unreduced, i.e., all the super- and sub-diagonal elements are non-vanishing. We will show that the divide-and-conquer algorithm outlined above therefore does not require γ-deflation at any step of the algorithm when applied to the symmetric arrow matrix (7).

We can apply γ-deflation to the arrow matrix (7) if one of the elements in the last row or column vanishes. These elements are made up of scaled elements from the first row of $\breve{U}_2$ and from the last row of $\breve{U}_1$. In particular, they are non-vanishing. This follows from the fact that the first and last elements of every eigenvector of an unreduced symmetric tridiagonal matrix are non-vanishing. We conclude that γ-deflation cannot take place as it would imply that an element in either the first row of $\breve{U}_2$ or the last row of $\breve{U}_1$ was zero.

Eigenvalues of the arrow matrix (7) can be determined with combo-deflation if any of the shaft elements (that is, the diagonal elements of the matrices $\breve{\Lambda}_1$ and $\breve{\Lambda}_2$) are repeated. We note that since the matrices $\breve{T}_1$ and $\breve{T}_2$ are unreduced, neither matrix has repeated eigenvalues. Therefore, any repeated elements on the shaft of the arrow occur in pairs. We already noted in Subsection 3.4 that combo-deflations may occur.

We illustrate details of combo-deflation. Let

$$J = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Permutation gives the matrix

$$P^T J P = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Reduction to an arrow via the divide-and-conquer method yields

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & 2 & 0 & 1/\sqrt{2} \\ 0 & 0 & 0 & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} & -1/\sqrt{2} & 1 \end{bmatrix}.$$

Another permutations gives the matrix

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 & 1/\sqrt{2} \\ 0 & 2 & 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & 0 & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} & -1/\sqrt{2} & 1 \end{bmatrix}.$$

It is easy to see that one gets two combo-deflations since the shaft elements are made up of a pair of zeros and a pair of twos. Combo-deflations are easy to deal with since the eigenvector corresponding to the eigenvalue represented by the repeated shaft element can be constructed very simply as follows: take the last column of the arrow and zero out everything except for the barb elements that correspond to the repeated shaft elements. Swap the two non-zero elements and negate one. This gives an unnnormalized eigenvector without additional rounding error. The rest of the algorithm is undisturbed by the combo-deflation.

The divide-phase is carried out repeatedly until only 3×3, 2×2, or 1×1 matrices remain. Details on how to compute the partial spectral factorization of these matrices accurately is addressed by Borges [4, 5]. The former work discusses the accurate spectral factorization of symmetric 3×3 matrices. First these matrices are transformed to arrow matrices, whose eigenvalues are computed by applying a zero-finder to a spectral function. A cubically convergent zero-finder described in [4, 6] can be used; the fact that the zero-finder converges cubically follows from [6, Eq. (5)]. Alternatively, we can apply a Newton method to compute the zeros. Newton's method can be shown to determine iterates that converge monotonically to the extreme zeros; see Section 4 for details. The iterations with both zero-finders are terminated as soon as two consecutive approximations of the zero are numerically identical. The "conquer phase" of the divide-and-conquer method yields larger and larger eigenproblems for symmetric arrow matrices until the nodes and weights of the Gauss rule (2) have been determined.

In [5] Borges discusses the computation of the spectral factorization of a 2×2 symmetric matrix by the application of a similarity transformation determined by a Givens rotation. Borges advocates the use of the MATLAB function hypot when evaluating the Givens rotations. This is done in the software used for the computed examples of the following section.

5. Computed examples

This section presents timings for the divide-and-conquer method applied to the Gauss, Gauss-Radau, Gauss-Lobatto, and optimal averaged Gauss quadrature rules as described in Section 3, and compares them to timings of the Golub-Welsch algorithm applied to the computation of these quadrature rules. We use the implementation of the Golub-Welsch algorithm furnished by the code gauss.m by Gautschi [13], and recall that this code does not implement the Golub-Welsch algorithm as described in [19]; to compute the nodes and weights of an n-node quadrature rule, the code gauss.m calculates the spectral factorization of an $n \times n$ matrix with the aid of the MATLAB function eig. This approach ignores the structure of the problem and requires $\mathcal{O}(n^3)$ flops, while the structure exploiting algorithm by Golub and Welsch described in [19] only demands $\mathcal{O}(n^2)$. Also computations with the divide-and-conquer algorithm only requires $\mathcal{O}(n^2)$ flops. However, the timings presented below show the code gauss.m to demand less CPU time than computations with the divide-and-conquer algorithm for the evaluation of quadrature rules with few to moderately many nodes. This depends on that the MATLAB function eig is implemented very efficiently; for quadrature rules with 100 nodes or more, the use of the divide-and-conquer algorithm is competitive with regard to CPU time. We note that the code gauss.m and, therefore, the function eig are called twice to evaluate pairs of Gauss and Gauss-Radau rules, pairs of Gauss and Gauss-Lobatto rules, or pairs of Gauss and optimal averaged Gauss rules. We remark that similarly as the divide-and-conquer algorithm, a user-implementation of the Golub-Welsch algorithm as described in [19], would require more CPU time than gauss.m for the computation of quadrature rules with few or moderately many nodes.

The accuracy of the computed nodes and weights determined by using the divide-and-conquer algorithm and the code gauss.m is about the same. We note that the structure of the former algorithm is well suited for implementation in a parallel computing environment. Such an implementation would make it possible to compute the nodes and weights of quadrature rules with very many nodes. The timings reported below are carried out on a laptop computer.

Specifically, the timings reported in Tables 1-4 are carried out in MATLAB version R2023a on a MacBook Pro laptop computer with an M1 processor and 8 GB of RAM. The computations are performed with about 15 significant decimal digits. Timings of the same computations carried out in different runs typically differ. The tables therefore report mean values over 1000 runs.

Table 1: Ratios of average CPU times for computing Gauss rules by the Golub-Welsch and the divide-and-conquer algorithms for several numbers of nodes. The table shows averages over 1000 runs.

	Gauss rules with k nodes.
k	$\frac{\text{Golub-Welsch algorithm}}{\text{divide-and-conquer algorithm}}$
200	4.733
100	1.669
50	0.614
20	0.459
10	0.876

Tables 1-4 show the divide-and-conquer algorithm to be competitive with the Golub-Welsch algorithm as implemented by the code gauss.m for 100 or more nodes.

Table 2: Ratios of average CPU times for computing Gauss and Gauss-Radau rules by the Golub-Welsch and the divide-and-conquer algorithms for several numbers of nodes. The table shows averages over 1000 runs.

Gauss rules with k nodes and Gauss-Radau rules with $k+1$ nodes.

k	$\frac{\text{Golub-Welsch algorithm}}{\text{divide-and-conquer algorithm}}$
200	4.565
100	1.583
50	0.560
20	0.266
10	0.359

Table 3: Ratios of average CPU times for computing Gauss and Gauss-Lobatto rules by the Golub-Welsch and the divide-and-conquer algorithms for several numbers of nodes. The table shows averages over 1000 runs.

Gauss rules with k nodes and Gauss-Lobatto rules with $k+2$ nodes.

k	$\frac{\text{Golub-Welsch algorithm}}{\text{divide-and-conquer algorithm}}$
200	4.685
100	1.825
50	0.714
20	0.473
10	0.733

Table 4: Ratios of average CPU times for computing Gauss and optimal averaged Gauss rules by the Golub-Welsch and the divide-and-conquer algorithms for several numbers of nodes. The table shows averages over 1000 runs.

Gauss rules with k nodes and optimal averaged Gauss rules with $2k+1$ nodes.

k	$\frac{\text{Golub-Welsch algorithm}}{\text{divide-and-conquer method}}$
200	4.984
100	1.728
50	0.719
20	0.318
10	0.106

6. Conclusion

The paper describes the calculation of the nodes and weights of several Gauss-type quadrature rules by using the divide-and-conquer algorithm presented in [6]. This approach to compute quadrature rules is found to be competitive with the available implementation `gauss.m` of the Golub-Welsch algorithm for quadrature rules with 100 or more nodes.

Acknowledgment

The research of D. Lj. Djukić, R. M. Mutavdžić Djukić and M. M. Spalević is supported in part by the Serbian Ministry of Science, Technological Development, and Innovations, according to Contract 451-03-47/2023-01/200105 dated on February 3, 2023.

The authors would like to thank the referees for comments that led to clarifications of the presentation.

References

[1] G. S. Ammar, D. Calvetti, and L. Reichel, *Computation of Gauss-Kronrod quadrature rules with non-positive weights*, Electron. Trans. Numer. Anal., 9 (1999), pp. 26–38.

[2] H. Alqahtani and L. Reichel, *Simplified anti-Gauss quadrature rules with applications in linear algebra*, Numer. Algorithms, 77 (2018), pp. 577–602.

[3] I. Bogaert, *Iteration-free computation of Gauss-Legendre nodes and weights*, SIAM J. Sci. Comput., 36 (2014), pp. A1008–A1026.

[4] C. F. Borges, *Solving the* 3×3 *symmetric eigenproblem*, arXiv: 1806.06698, 2018.

[5] C. F. Borges, *An improved formula for Jacobi rotations*, arXiv: 1806.07876v1, 2018.

[6] C. F. Borges and W. B. Gragg, *A parallel divide and conquer algorithm for the generalized symmetric definite tridiagonal eigenvalue problem*, in Numerical Linear Algebra, eds. L. Reichel, A. Ruttan, and R. S. Varga, de Gruyter, Berlin, 1993, pp. 11–29.

[7] D. Calvetti, G. H. Golub, W. B. Gragg, and L. Reichel, *Computation of Gauss-Kronrod rules*, Math. Comp., 69 (2000) 1035–1052.

[8] D. Calvetti, L. Reichel, and F. Sgallari, *Application of anti-Gauss quadrature rules in linear algebra*, in Applications and Computation of Orthogonal Polynomials, W. Gautschi, G. H. Golub, and G. Opfer, eds., Birkhäuser, Basel, 1999, pp. 41–56.

[9] P. Díaz de Alba, L. Fermo, and G. Rodriguez, *Solution of second kind Fredholm integral equations by means of Gauss and anti-Gauss quadrature rules*, Numer. Math., 146 (2020), pp. 699–728.

[10] D. Lj. Djukić, L. Reichel, and M. M. Spalević, and J. D. Tomanović, *Internality of generalized averaged Gauss rules and their truncations for Bernstein-Szegő weights*, Electron. Trans. Numer. Anal., 45 (2016), pp. 405–419.

[11] D. Lj. Djukić, R. M. Mutavdžić Djukić, L. Reichel, and M. M. Spalević, *Internality of generalized averaged Gauss quadrature rules and truncated variants for modified Chebyshev measures of the first kind*, J. Comput. Appl. Math., 398 (2021), Art. 113696.

[12] D. Lj. Djukić, R. M. Mutavdžić Djukić, L. Reichel, and M. M. Spalević, *Weighted averaged Gaussian quadrature rules for modified Chebyshev measures*, Appl. Numer. Math., (2023), ISSN 0168-9274, https://doi.org/10.1016/j.apnum.2023.05.014.

[13] W. Gautschi, *OPQ: A Matlab suite of programs for generating orthogonal polynomials and related quadrature rules*, available at https://www.cs.purdue.edu/archives/2002/wxg/codes/OPQ.htm

[14] W. Gautschi, *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, Oxford, 2004.

[15] A. Glaser, X. Liu, and V. Rokhlin, *A fast algorithm for calculating the roots of special functions*, SIAM J. Sci. Comput., 29 (2007), pp. 1420–1438.

[16] G. H. Golub, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.

[17] G. H. Golub and G. Meurant, *Matrices, Moments and Quadrature with Applications*, Princeton University Press, Princeton, 2010.

[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, 2013.

[19] G. H. Golub and J. H. Welsch, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[20] M. Gu and S. C. Eisenstat, *A divide-and-conquer method for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.

[21] N. Hale and A. Townsend, *Fast and accurate computation of Gauss-Legendre and Gauss-Jacobi quadrature nodes and weights*, SIAM J. Sci. Comput., 35 (2013), pp. A652–A674.

[22] D. P. Laurie, *Anti-Gaussian quadrature formulas*, Math. Comp., 65 (1996), pp. 739–747.

[23] D. P. Laurie, *Calculation of Gauss-Kronrod quadrature rules*, Math. Comp., 66 (1997), pp. 1133–1145.

[24] S. Notaris, *Gauss-Kronrod quadrature formulae - a survey of fifty years of research*, Electron. Trans. Numer. Anal., 45 (2016), pp. 371–404.

[25] L. Reichel and M. M. Spalević. *A new representation of generalized averaged Gauss quadrature rules*, Appl. Numer. Math., 165 (2021) pp. 614–619.

[26] L. Reichel and M. M. Spalević. *Averaged Gauss quadrature formulas: Properties and applications*, J. Comput. Appl. Math., 410 (2022), Art. 114232.

[27] M. M. Spalević, *On generalized averaged Gaussian formulas*, Math. Comp., 76 (2007), pp. 1483–1492.

[28] G. Szegő, *Orthogonal Polynomials*, 4th ed., Amer. Math. Soc., Providence, 1975.