Electronic Transactions on Numerical Analysis.
Volume 61, pp. 121–136, 2024.
Copyright © 2024, Kent State University.
ISSN 1068–9613.
DOI: 10.1553/etna_vol61s121

COMPUTATION OF GAUSS-TYPE QUADRATURE RULES[*]

CARLOS F. BORGES[†] AND LOTHAR REICHEL[‡]

Dedicated to Giuseppe Rodriguez on the occasion of his 60th birthday.

Abstract. Many problems in scientific computing require the evaluation of Gauss quadrature rules. It is important to be able to estimate the quadrature error in these rules. Error estimates or error bounds often can be computed by evaluating an additional related Gauss-type formula such as a Gauss-Radau, Gauss-Lobatto, anti-Gauss, averaged Gauss, or optimal averaged Gauss rule. This paper presents software for both the evaluation of a single Gauss quadrature rule and the calculation of a pair of a Gauss rule and a related Gauss-type rule. The software is based on a divide-and-conquer method. This method is compared to both an available and a new implementation of the Golub-Welsch algorithm, which is the classical approach to evaluate a single Gauss quadrature rule. Timings on a laptop computer show the divide-and-conquer method to be competitive except for the computation of a single quadrature rule with very few nodes.

Key words. quadrature, Gauss rule, Gauss-Radau rule, Gauss-Lobatto rule, averaged Gauss rule, optimal averaged Gauss rule quadrature, divide-and-conquer method, Golub-Welsch algorithm

AMS subject classifications. 65D30, 65D32

1. Introduction.

The need to approximate integrals over an interval on the real axis arises in many applications in mathematics, science, and engineering; see, e.g., Gautschi [23] and Golub and Meurant [27] for discussions of many applications. It is common to approximate such integrals by a Gauss quadrature rule. This paper describes software for the computation of Gauss-type quadrature rules associated with a fairly general real measure with support on the real axis or part thereof. We assume that the recurrence coefficients of the three-term recurrence relation for monic orthogonal polynomials determined by the measure are available and discuss the efficient computation of a Gauss quadrature rule or of a pair of quadrature rules made up of a Gauss rule and a related Gauss-type quadrature rule from the recursion coefficients. The Gauss-type rules considered include Gauss-Radau, Gauss-Lobatto, anti-Gauss, averaged Gauss, and optimal averaged Gauss rules. We note that for classical measures, as well as for many other measures, the recursion coefficients of the three-term recurrence relation of orthogonal polynomials are explicitly known; see Gautschi [23] as well as [15, 16, 17, 18] for examples. This is also the case in many applications in linear algebra, where the recursion coefficients are generated by the Lanczos algorithm; see [4, 11, 27, 38] for some illustrations. In the event that the recursion coefficients are not available, we refer to Gautschi [23, Section 2.2] for a discussion on how they can be computed from the measure that determines the orthogonal polynomials.

The classical algorithm for computing Gauss-type quadrature rules from recursion coefficients that are associated with a fairly general measure with support on the real axis is due to Golub and Welsch [29]; it requires $\mathcal{O}(n^2)$ arithmetic floating point operations (flops) to determine the nodes and weights of an n-node Gauss rule from the zeroth moment and the first $2n-1$ recursion coefficients. We will compare software based on a divide-and-conquer method to two implementations of the Golub-Welsch algorithm, one provided by Meurant [35] and one based on a QR algorithm described by Gates and Gragg [21]. The implementation provided by Gautschi [22] does not exploit the structure of the problem and demands $\mathcal{O}(n^3)$

[*]Received May 13, 2024. Accepted August 12, 2024. Published online on October 18, 2024. Recommended by A. Buccini.

[†]Department of Mathematics, Naval Postgraduate School, Monterey, CA 93943, USA (borges@nps.edu).
[‡]Department of Mathematical Sciences, Kent State University, Kent, OH 44242, USA (reichel@math.kent.edu).

flops to calculate the nodes and weights of an n-node Gauss rule. Therefore, it is not part of our comparison.

We remark that faster schemes than the Golub-Welsch algorithm are available for the computation of Gauss quadrature rules associated with certain classical measures. Bogaert [5], as well as Hale and Townsend [31], describe algorithms that only demand $\mathcal{O}(n)$ flops to compute the nodes and weights of n-node Gauss-Legendre and Gauss-Jacobi quadrature rules. The latter algorithms are competitive with the Golub-Welsch algorithm with respect to CPU-time when n is large, and they are faster than the algorithm by Glaser et al. [25], which also requires $\mathcal{O}(n)$ flops and can be applied to a larger class of classical measures including Legendre, Jacobi, Hermite, and Laguerre measures.

It often is important to estimate the quadrature error when applying an n-node Gauss rule to determine whether the number of nodes chosen yields an approximation of an integral of desired accuracy; a rule with too few nodes gives a too large quadrature error, while using too many nodes results in an unnecessarily large computational burden, in particular if the integrand is cumbersome to evaluate. Methods for estimating the quadrature error of Gauss rules therefore have received considerable attention. Recent discussions on error estimation can be found in, e.g., [14, 32, 37, 42]; see also the paper by Gautschi and Varga [24].

The classical approach to estimate the quadrature error of an n-node Gauss rule associated with a fairly general measure is to evaluate a related $(2n + 1)$-node Gauss-Kronrod rule and use the difference of the values of the Gauss-Kronrod and Gauss rules as an estimate of the error in the Gauss rule. However, differently from Gauss rules, 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 fact for some measures some nodes may be complex-valued. This limits the applicability Gauss-Kronrod rules, because the integrand might not be defined at all Gauss-Kronrod nodes; see Notaris [36] for a nice survey of Gauss-Kronrod rules and [1] for some computed examples.

This shortcoming of Gauss-Kronrod rules has prompted the development of several alternative techniques to estimate the quadrature error of Gauss rules. They include the evaluation of pairs of an n-node Gauss and

(i) an associated $(n + 1)$-node Gauss-Radau rule or an associated $(n + 1)$-node Gauss-Lobatto rule. Pairs of a Gauss rule and a suitable Gauss-Radau rule, or of a Gauss rule and an appropriate Gauss-Lobatto rule, give upper and lower bounds for the integral under certain conditions on the integrand. This follows from the remainder formula for Gauss, Gauss-Radau, and Gauss-Lobatto quadrature rules; see, e.g., Golub and Meurant [26, 27] or Gautschi [23].

(ii) an associated $(n + 1)$-node anti-Gauss rule. Pairs of these quadrature rules yield upper and lower bounds for the integral if the coefficients of an expansion of the integrand in terms of orthogonal polynomials associated with the measure converge to zero sufficiently quickly; see [12]. A related method is described in [3]. Anti-Gauss rules were introduced by Laurie [33]. A recent analysis of their properties is provided by Díaz de Alba et al. [13].

(iii) an associated $(2n + 1)$-node averaged Gauss rule. The averaged Gauss rule is the average of the n-node Gauss rule and the associated $(n + 1)$-node anti-Gauss rule. Averaged rules were introduced by Laurie [33]. The difference between the averaged Gauss rule and the Gauss rule provides an estimate of the quadrature error for the Gauss rule. Some computed examples can be found in [40].

(iv) an associated $(2n + 1)$-node optimal averaged Gauss rules. The latter rules were introduced by Spalević [41] and have a higher degree of precision than the averaged Gauss rule with the same number of nodes. The $(2n + 1)$-node optimal averaged

Gauss rules can be written as a weighted sum of the n-node Gauss rule and a related $(n+1)$-node Gauss-type quadrature rule; see [39]. The difference between the optimal averaged Gauss rule and the Gauss rule gives an estimate for the quadrature error for the Gauss rule. Applications to error estimation are described in [40]. Analyses of properties of optimal averaged Gauss rules can be found in [15, 16, 17, 18, 20].

In all the error estimation approaches (i)–(iv), both the n-node Gauss rule and an associated $(n+1)$-node Gauss-type quadrature rule have to be evaluated. Alqahtani et al. [2] describe how such pairs of quadrature rules can be calculated efficiently by a divide-and-conquer method. This paper presents software for a divide-and-conquer method that is based on an algorithm described by Borges and Gragg [9]. The performance of the software is compared to two structure-respecting implementations of the Golub-Welsch algorithm.

The computation of pairs of Gauss-type quadrature rules by the Golub-Welsch algorithm demands two applications of this algorithm, one for each quadrature rule. Timings reported by Alqahtani et al. [2], based on computations on a laptop computer, show the divide-and-conquer method to require less CPU-time than two applications of the structure-ignoring implementation of the Golub-Welsch algorithm available in [22] when the quadrature rules have 100 or more nodes. We have polished the code for the divide-and-conquer method, and our present implementation demands less CPU-time for computing pairs of Gauss-type quadrature rules than a structure respecting implementation based on a tridiagonal QR algorithm by Gates and Gragg [21]. Moreover, the divide-and-conquer method is competitive with regard to CPU-time also when computing a single Gauss quadrature rule with 32 or more nodes. The accuracy in the nodes and weights achieved with the divide-and-conquer method typically is higher than the accuracy obtained with the structure-respecting implementations of the Golub-Welsch algorithm used for in our comparison. This is illustrated in Section 6. We also note that divide-and-conquer methods lend themselves well to implementation on parallel computers. This makes it possible to use a divide-and-conquer algorithm to evaluate quadrature rules with very many nodes.

This paper is organized as follows. Section 2 describes Gauss quadrature rules and Section 3 introduces the concept of simply nested pairs of quadrature rules. This concept is used when computing pairs of quadrature rules, one of which is a Gauss rule and the other one may be a Gauss-Radau, Gauss-Lobatto, or anti-Gauss rule. The computation of these rules is based on calculating the partial spectral factorization of tridiagonal matrices. This is discussed in Section 4. Section 5 defines nested pairs of quadrature rules, which generalizes the concept of simply nested pairs. Computed examples and timings are presented in Section 6, and concluding remarks can be found in Section 7.

We remark that this paper does not discuss the evaluation of pairs of an n-node Gauss rule and an associated $(2n+1)$-node Gauss-Kronrod rule. The reasons for this is that Gauss-Kronrod rules are not guaranteed to have real nodes and they are more complicated to compute than the Gauss-type rules considered in this paper; see [1, 10, 34] for algorithms for the evaluation of Gauss-Kronrod rules.

2. Gauss quadrature rules.

Let $\mathrm{d}w$ be a non-negative measure on the real axis with infinitely many points of support and such that all moments $\mu_k = \int t^k \mathrm{d}w(t)$, $k = 0, 1, \ldots$, are well defined. Gauss quadrature rules are well suited to approximate integrals of the form

$$\mathcal{I}(f) = \int f(t)\mathrm{d}w(t). \tag{2.1}$$

The n-node Gauss quadrature rule can be expressed as

$$\mathcal{G}_n(f) = \sum_{j=1}^{n} f(t_j)w_j. \tag{2.2}$$

The nodes t_j are known to be real, distinct, and live in the convex hull of the support of the measure $\mathrm{d}w$; the weights w_j are positive. We refer to Gautschi [23] for discussions and proofs of many properties of Gauss quadrature rules.

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

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

Thus, the polynomial p_j is of degree j, has leading coefficient one, and the polynomials satisfy

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

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

$$p_{j+1}(t) = (t - \alpha_j)p_j(t) - \beta_j p_{j-1}(t), \quad j = 0, 1, \ldots, \tag{2.3}$$

where the coefficients α_j and β_j are given by

$$\alpha_j = \frac{\langle tp_j, p_j \rangle}{\langle p_j, p_j \rangle}, \qquad j = 0, 1, \ldots,$$
$$\beta_j = \frac{\langle p_j, p_j \rangle}{\langle p_{j-1}, p_{j-1} \rangle}, \quad j = 1, 2, \ldots,$$

and β_0 is arbitrary. In our code, we set $\beta_0 = \mu_0$. The nodes $t_1, t_2, \ldots, t_n$ of the Gauss rule (2.2) are the zeros of p_n; see, e.g., [23] for a proof.

The Gauss rule (2.2) can be associated with the symmetric tridiagonal matrix

$$T_n = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \ddots & \\ & \ddots & \ddots & \sqrt{\beta_{n-1}} \\ 0 & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{2.4}$$

with positive subdiagonal entries. Its eigenvalues are the nodes $t_1, t_2, \ldots, t_n$ of the Gauss rule and the weights $w_1, w_2, \ldots, w_n$ of the Gauss rule are the squares of the first components of normalized eigenvectors, scaled by the zeroth moment, μ_0, of the measure.

3. Computation with simply nested pairs of Gauss rules.

We will say that a pair of related Gauss quadrature rules are *simply nested* if they are related in the following way:

- The nodes and weights of the first rule are the eigenvalues and scaled squares of the first elements of the eigenvectors of T_n.
- The nodes and weights of the second rule are the eigenvalues and scaled squares of the first elements of the eigenvectors of

$$T_{n+1} = \begin{bmatrix} T_n & \gamma \mathbf{e}_n \\ \gamma \mathbf{e}_n^T & \omega \end{bmatrix}$$

  for some choices of the real parameters γ and ω. Throughout this paper $\mathbf{e}_n$ denotes the nth column of an identity matrix of suitable order and the superscript ^{T} denotes transposition.

We are interested in calculating the nodes and weights of both rules as accurately and efficiently as possible. Historically, the best way of doing this was to use the Golub-Welsch algorithm twice. No benefit is derived from the first calculation that might simplify the second. We will take a different approach. Assume that the spectral factorization of the matrix (2.4) has been computed,

$$T_n = U_n \Lambda_n U_n^T, \tag{3.1}$$

where U_n is a unitary matrix whose columns are the eigenvectors of T_n, Λ_n is a diagonal matrix whose diagonal entries are the eigenvalues. Observe that the following unitary similarity transformation

$$\begin{bmatrix} U_n & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} T_n & \gamma \mathbf{e}_n \\ \gamma \mathbf{e}_n^T & \omega \end{bmatrix} \begin{bmatrix} U_n & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \Lambda_n & \gamma U_n^T \mathbf{e}_n \\ \gamma \mathbf{e}_n^T U_n & \omega \end{bmatrix}$$

yields a symmetric arrow matrix. Let $\mathbf{u}^T = \mathbf{e}_1^T U_n$ and $\mathbf{v}^T = \mathbf{e}_n^T U_n$ denote, respectively, the first and last rows of U_n. Then the resulting symmetric arrow matrix can be written as

$$A_{n+1} = \begin{bmatrix} \Lambda_n & \gamma \mathbf{v} \\ \gamma \mathbf{v}^T & \omega \end{bmatrix}. \tag{3.2}$$

We note that solving the eigenproblem for a symmetric arrow matrix is much faster on parallel computers than doing so for a symmetric tridiagonal matrix of the same size, since the eigenvalues and eigenvectors can all be computed simultaneously as the problem possesses perfect parallelism. Moreover, the problem is conducive to vectorized/pipelined operations. This makes the computations efficient even without exploiting parallelism.

The codes that accompany this paper[1] follow the development in Borges and Gragg [9], including the use of the zero-finder described therein, which has monotonic global cubic convergence. We note that careful deflation is critically important.

The code `eig_arrow.m` provided in the software package that accompanies this paper implements this method. For reasons of efficiency this code assumes, without validation, that the shaft elements of the arrow matrix are sorted and that the barb elements are non-negative. To make the code widely applicable, the code is serial and does not take advantage of the perfect parallelism of the eigenvalue problem for the symmetric arrow matrix. Moreover, we have deliberately limited the vectorization in the code to improve readability at the cost of speed. In particular, although we vectorize the opening step of the zero-finder (evaluating the secular function at the center of each interior interval), we do not vectorize subsequent steps as this would result in delicate and confusing code.

We will refer to a partial decomposition of the matrix T_n that consists of the eigenvalues, and the first and last components of the normalized eigenvectors, as a *partial spectral factorization* (PSF) of T_n. We observe that the PSF of T_n provides us with sufficient information for computing the n-point Gauss rule (the eigenvalues and the elements of **u**), as well as sufficient information (the eigenvalues and elements of **v**), when paired with the recurrence coefficients γ and ω, to compute the $(n+1)$-point Gauss rule by solving a symmetric arrow eigenvalue problem instead of a symmetric tridiagonal eigenvalue problem.

Let the spectral factorization

$$A_{n+1} = U_{n+1} \Lambda_{n+1} U_{n+1}^T$$

[1] See the supplementary material:
https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol61/addition/p121.php.

be known. Then, the nodes of the $(n+1)$-point Gauss rule are given by the diagonal elements of Λ_{n+1}, and the weights (up to the scaling by μ_0) are given by the elements of the first row of

$$\begin{bmatrix} U_n & 0 \\ 0 & 1 \end{bmatrix} U_{n+1},$$

which are given by $\begin{bmatrix} \mathbf{u}^T & 0 \end{bmatrix} U_{n+1}$. We see that both rules can be evaluated by first computing the PSF of T_n and then solving a symmetric arrow eigenvalue problem. This leads to Algorithm 1.

Algorithm 1 Computing a simply nested pair of Gauss rules

```
1. Compute the PSF of $T_n$, given by $\Lambda_n$, $\mathbf{u}$, and $\mathbf{v}$.
2. Compute the first Gauss-rule using the diagonal elements of $\Lambda_n$ and the squared
   elements of $\mathbf{u}$.
3. Compute $\gamma$ and $\omega$, and let
       $A_{n+1} = \begin{bmatrix} \Lambda_n & \gamma\mathbf{v} \\ \gamma\mathbf{v}^T & \omega \end{bmatrix}.$
4. Compute the spectral factorization of $A_{n+1} = U_{n+1}\Lambda_{n+1}U_{n+1}^T$.
5. Compute the second Gauss-rule using the diagonal elements of $\Lambda_{n+1}$ and the squared
   elements of $\begin{bmatrix} \mathbf{u}^T & 0 \end{bmatrix} U_{n+1}$.
```

3.1. Some specific simply nested pairs of Gauss-type rules.

This section outlines how Algorithm 1 can be applied to evaluate several known and widely used simply nested pairs of quadrature rules. Since Algorithm 1 can solve any problem of this type, all that remains to be discussed are the differences in the computations for specific values of γ and ω in step 3 of the algorithm. We shall see that in some cases we can further leverage the PSF to evaluate γ and ω in a very efficient manner.

3.1.1. Computing a simply nested pair of Gauss and Gauss-Radau rules.

Let the measure dw be contained in the bounded real interval $[a, b]$, and let the integrand f in (2.1) be $2n + 1$ times continuously differentiable in this interval. Assume that the $2n$th and $(2n + 1)$st derivatives of f are of constant sign in this interval. It then follows from the remainder terms for Gauss and Gauss-Radau quadrature rules that the user-specified node $\widetilde{t}_0 \in \{a, b\}$ of the $(n + 1)$-node Gauss-Radau rule

$$\mathcal{R}_{n+1}(f) = \sum_{j=0}^{n} f(\widetilde{t}_j)\widetilde{w}_j \tag{3.3}$$

can be chosen so that the quadrature errors of the Gauss rule (2.2) and the Gauss-Radau rule (3.3) are of opposite sign. Hence, these quadrature rules bracket the value of the integral (2.1); see, e.g., Golub and Meurant [27] for details. It is therefore of interest to compute the nodes and weights for both the rules (2.2) and (3.3). The latter rule can be associated with the symmetric tridiagonal matrix

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

where the entry $\widetilde{\alpha}_n$ is chosen so that the Radau node $\widetilde{t}_0$ is at the desired location.

One can determine $\widetilde{\alpha}_n$ to allocate the node $\widetilde{t}_0$ at the desired point by combining [26, eqs (7.5) and (7.6)]. This leads to the formula

$$\widetilde{\alpha}_n = \widetilde{t}_0 + \beta_n \mathbf{e}_n^T (T_n - \widetilde{t}_0 I)^{-1} \mathbf{e}_n.$$

Invoking the spectral factorization (3.1), this becomes

$$\widetilde{\alpha}_n = \widetilde{t}_0 + \beta_n \mathbf{e}_n^T U_n (\Lambda_n - \widetilde{t}_0 I)^{-1} U_n^T \mathbf{e}_n.$$

Since $\mathbf{v}^T = [v_1, v_2, \dots, v_n] = \mathbf{e}_n^T U_n$, we can leverage the PSF to get

$$\widetilde{\alpha}_n = \beta_n \sum_{k=1}^{n} \frac{v_k^2}{\lambda_k - \widetilde{t}_0}. \tag{3.4}$$

The eigenvalues of T_n live in the largest open interval whose closure is the closed convex hull of the measure $\mathrm{d}\omega$. Therefore, the Gauss-Radau rule exits for $\widetilde{t}_0 \in \{a, b\}$.

To evaluate the Gauss-Radau rule (3.3), we set $\gamma = \sqrt{\beta_n}$, where β_n is a recursion coefficient for the orthogonal polynomials (2.3), and let $\omega = \widetilde{\alpha}_n$ in step 3 of Algorithm 1 using the formula (3.4). The code `GaussPlusGaussRadau.m` included in the program package provided with this paper implements this method.

3.1.2. Computing a simply nested pair of Gauss and Gauss-Lobatto rules. Let the support of the measure $\mathrm{d}w$ be contained in the bounded real interval $[a, b]$, and let the integrand f in (2.1) be $2n$ times continuously differentiable in this interval. Assume that the $2n$th derivative of f is of constant sign in this interval. The user-specified Lobatto nodes are often chosen to be $\widetilde{t}_0 = a$ and $\widetilde{t}_n = b$. Then, the quadrature error of the $(n+1)$-node Gauss-Lobatto rule

$$\mathcal{L}_{n+1}(f) = \sum_{j=0}^{n} f(\widetilde{t}_j)\widetilde{w}_j \tag{3.5}$$

is of opposite sign as the quadrature error of the Gauss rule. This follows from the remainder terms of Gauss and Gauss-Lobatto quadrature rules; see Golub [26], Golub and Meurant [27], or Gautschi [23].

The key is choosing the parameters γ and ω in the matrix (3.2) to force the Lobatto nodes to occur at the locations a and b. Following a construction similar to the one used in Section 3.1.1, we can once again leverage the PSF to show that

$$\gamma = \frac{a - b}{S_b - S_a}, \qquad \omega = a + \gamma S_a,$$

where

$$S_a = \sum_{k=1}^{n} \frac{v_k^2}{\lambda_k - a}, \qquad S_b = \sum_{k=1}^{n} \frac{v_k^2}{\lambda_k - b}.$$

Here we use the same notation as in (3.4); see [23, 26, 27] for details. In particular, the Gauss-Lobatto rule (3.5) exists for the choice of Lobatto nodes of this section. The algorithm `GaussPlusGaussLobatto.m` provided with this paper implements the method of the present section.

3.1.3. Computing a simply nested pair of Gauss and anti-Gauss rules. The anti-Gauss rule

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

associated with the Gauss rule (2.2) is characterized by

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

see Laurie [33]. The tridiagonal matrix associated with this anti-Gauss rule is given by

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

We see that $\gamma = \sqrt{2\beta_n}$ and $\omega = \alpha_n$ in (3.2). The code `GaussPlusAntiGauss.m` that accompanies this paper implements the method of this section.

4. Computing the PSF.

The critical step when using simply nested pairs of quadrature rules is computing the PSF of the symmetric tridiagonal matrix (2.4). We describe two algorithms for doing so.

4.1. Computing the PSF using the QR algorithm. The classical Golub-Welsch algorithm [29] is easy to understand conceptually. The algorithm computes the eigenvalues and first elements of the normalized eigenvectors of a symmetric tridiagonal matrix $T_n \in \mathbb{R}^{n\times n}$. To do this, one can apply any standard QR algorithm for symmetric tridiagonal matrices and instead of simply applying the sequence of rotations Q_k generated by the algorithm to the tridiagonal matrix as in

$$Q_m^T Q_{m-1}^T \cdots Q_1^T T_n Q_1 \cdots Q_{m-1} Q_m \rightarrow \Lambda_n, \tag{4.1}$$

one simultaneously captures the first row of the eigenvector matrix by applying these rotations to the transpose of the first axis vector

$$\mathbf{u}^T = \mathbf{e}_1^T Q_1 \cdots Q_{m-1} Q_m.$$

The right arrow in (4.1) signifies that the rotations Q_k are chosen so that the left-hand side converges to the diagonal matrix Λ_n as the number of rotations applied increases.

One can compute the PSF by also applying the rotations to the transpose of the last axis vector,

$$\mathbf{v}^T = \mathbf{e}_n^T Q_1 \cdots Q_{m-1} Q_m.$$

The determination of the vectors **u** and **v** requires just a few additional flops per iteration, compared to the QR algorithm when applied to only compute the eigenvalues of a symmetric tridiagonal matrix, and the total flop count remains $O(n^2)$ for the matrix T_n.

The code that we provide with this paper (`tqrPSF.m`) is based on the TQR algorithm described by Gates and Gragg [21]. If a user supplies two output parameters, then the code performs a true Golub-Welsch algorithm to generate the eigenvalues and the positive first elements of the normalized eigenvectors; if a user gives three output parameters, then the code delivers the PSF as just described.

To avoid unnecessary overflows/underflows every normalization that is required to evaluate the rotations is computed with a call to the function `hypot(a,b)`. This is generally more accurate than the commonly used approach described, e.g., in [28, Algorithm 5.13]. The use of this function is particularly attractive in computer languages, such as Julia, that implement a function `hypot(a,b)` with correct rounding. Such a function is described in [7]. More accurate algorithms for constructing Givens rotations, such as the one described in [8], can yield further improvement.

4.2. Computing the PSF by a divide-and-conquer method. We outline the computation of the PSF of T_n by using the divide-and-conquer by extension method described in [9, 30]. Algorithms for divide-and-conquer methods consist of a divide phase and a conquer phase. We outline these phases and use the notation in [9], where further details can be found.

Consider the spectral factorization (3.1) of the matrix T_n defined by (2.4). We would like to compute the PSF of T_n and for this purpose introduce the split index s for some $1 \leq s \leq n$. Regard the block form of the matrix T_n,

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

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.

We begin by solving the smaller eigenvalue problems

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

in what is known as the *divide phase* or *split phase*. Here $\breve{U}_k$ is an orthogonal matrix whose columns are the 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 $\mathbf{v}_1 = \breve{U}_1^T \mathbf{e}_s \in \mathbb{R}^{s-1}$ and $\mathbf{u}_2 = \breve{U}_2^T \mathbf{e}_1 \in \mathbb{R}^{n-s}$, we obtain

$$\widehat{U}_n^T T_n \widehat{U}_n = \begin{bmatrix} \breve{\Lambda}_1 & \sqrt{\beta_{s-1}}\mathbf{v}_1 & 0 \\ \sqrt{\beta_{s-1}}\mathbf{v}_1^T & \alpha_{s-1} & \sqrt{\beta_s}\mathbf{u}_2^T \\ 0 & \sqrt{\beta_s}\mathbf{u}_2 & \breve{\Lambda}_2 \end{bmatrix}. \tag{4.2}$$

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 $\mathbf{v}_1$ and $\mathbf{u}_2$.

The matrix (4.2) 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. Define the permutation matrix

$$\widehat{P}_n = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{s-1}, \mathbf{e}_{s+1}, \mathbf{e}_{s+2}, \ldots, \mathbf{e}_n, \mathbf{e}_s] \in \mathbb{R}^{n\times n}.$$

Then,

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

is an arrow matrix, which we shall denote by A. There are efficient methods for computing the PSF of an arrow matrix. Such methods are discussed by Borges and Gragg [9] as well as by Gu and Eisenstat [30]. We refer to these references for details.

Assume that we have computed the spectral factorization of $A = U^T \Lambda U$. Then in the *conquer* phase of the algorithm, we obtain

$$U^T \widehat{P}_n^T \widehat{U}_n^T T_n \widehat{U}_n \widehat{P}_n U = \Lambda.$$

The eigenvector matrix of T_n is given by $\widehat{U}_n \widehat{P}_n U$, and we can recover its first and last rows as follows

$$\mathbf{e}_1^T \widehat{U}_n \widehat{P}_n U = \begin{bmatrix} \mathbf{u}_1^T & 0 & \ldots & 0 \end{bmatrix} U$$

and

$$\mathbf{e}_n^T \widehat{U}_n \widehat{P}_n U = \begin{bmatrix} 0 & \ldots. & 0 & \mathbf{v}_2^T & 0 \end{bmatrix} U.$$

This yields the PSF of T_n. Clearly, we need only compute the PSF of each of the subproblems to construct the PSF for the original problem.

We have described one application of the split phase. Divide-and-conquer codes apply the splitting repeatedly until only 1×1 or 2×2 matrices remain. In the code that accompanies this paper (`dandcPSF.m`), the eigenvalue problems for the 2×2 matrices are solved by using a modification of the code in Borges [6] that restricts itself to real symmetric 2×2 matrices with a *non-negative* off-diagonal element. The code `dandcPSF.m` calls the standard math library function `hypot()` to prevent unnecessary overflow/underflow and gives demonstrably more accurate results over a wider range of inputs than other common approaches.

We note that `dandcPSF.m` is a recursive serial implementation of the algorithm. The code is deliberately written to be easy to use and modify. A non-recursive and/or fully parallel version would demand less CPU-time.

5. Computation of nested pairs of Gauss rules.

This section extends the approach described above to a more general setting. We say that a pair of related Gauss quadrature rules are *nested* if they are related in the following way:

- The nodes and weights of the first rule are the eigenvalues and squares of the first components of the eigenvectors of T_n.
- The nodes and weights of the second rule are the eigenvalues and squares of the first components of the eigenvectors of

$$T_{n+1} = \begin{bmatrix} T_n & \gamma_1 \mathbf{e}_n & 0 \\ \gamma_1 \mathbf{e}_n^T & \omega & \gamma_2 \mathbf{e}_1^T \\ 0 & \gamma_2 \mathbf{e}_1 & \bar{T} \end{bmatrix},$$

  where $\bar{T}$ is a symmetric tridiagonal matrix and γ_1, γ_2, and ω are scalars.

Algorithm 2 Computing a nested pair of Gauss rules

```
1. Compute the PSF of $T_n$, given by $\Lambda_n$, $\mathbf{u}_1$, and $\mathbf{v}_1$.
2. Compute the first Gauss rule using the diagonal elements of $\Lambda_n$ and the squared elements of $\mathbf{u}_1$.
3. Compute the PSF of $\bar{T}$, given by $\bar{\Lambda}$, $\mathbf{u}_2$, and $\mathbf{v}_2$.
4. Compute $\gamma_1, \gamma_2$, and $\omega$, and let
   $$A = \begin{bmatrix} \Lambda_n & 0 & \gamma_1\mathbf{v}_1 \\ 0 & \bar{\Lambda} & \gamma_2\mathbf{u}_2 \\ \gamma_1\mathbf{v}_1^T & \gamma_2\mathbf{u}_2^T & \omega \end{bmatrix}.$$
5. Compute the spectral factorization of $A = U\Lambda U^T$.
6. Compute the second Gauss-rule using the diagonal elements of $\Lambda$ and the squared elements of $\begin{bmatrix}\mathbf{u}_1^T & 0 & \dots & 0\end{bmatrix} U$.
```

Now recall the derivation in Section 4.2 with the substitutions $\breve{T}_1 = T_n$ and $\breve{T}_2 = \bar{T}$. It is straightforward to verify that we can compute the nested pair of rules with Algorithm 2. In the degenerate case, when the matrix $\bar{T}$ is void (and therefore also γ_2), Algorithm 2 simplifies to Algorithm 1.

5.1. Computation of pairs of Gauss and optimal averaged Gauss rules. The difference between the values of the Gauss rule (2.2) and the associated $(2n+1)$-node optimal averaged Gauss quadrature rule

$$S_{2n+1}(f) = \sum_{j=1}^{2n+1} f(\widetilde{t}_j)\widetilde{w}_j \tag{5.1}$$

often provides an accurate estimate of the quadrature error of the Gauss rule; see [39, 40] for illustrations.

It is well known that the nodes and weights of the optimal averaged Gauss rule (5.1) are the eigenvalues and the squared first components of the normalized eigenvectors, respectively, of the concatenated symmetric tridiagonal matrix

$$\widehat{T}_{2n+1} = \begin{bmatrix} T_n & \sqrt{\beta_n}e_n & 0 \\ \sqrt{\beta_n}e_n^T & \alpha_n & \sqrt{\beta_{n+1}}e_1^T \\ 0 & \sqrt{\beta_{n+1}}e_1 & JT_nJ \end{bmatrix} \in \mathbb{R}^{(2n+1)\times(2n+1)}, \tag{5.2}$$

where $J \in \mathbb{R}^{n\times n}$ is the counter-identity; see, e.g., [2, 41]. This construction shows that the Gauss rule and the optimal averaged Gauss rule are a nested pair, and, hence, we can compute both using Algorithm 2.

There are significant advantages to be had by exploiting the structure of the matrix (5.2). First, we note that the PSF of JT_nJ can be determined from the PSF of the matrix T_n without arithmetic cost by swapping the roles of **u** and **v** from the PSF of T_n, and second, we can exploit the massive deflation in the arrow matrix that results from this symmetry. In particular, the matrices JT_nJ and T_n have the same spectra. Because of this we will not pursue solving this problem by Algorithm 2 directly but instead refer the reader to the method discussed in [2] that fully exploits the structure. The code `GaussPlusOptimalAveragedGauss.m` that is supplied with this paper implements a structure-exploiting divide-and-conquer method.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

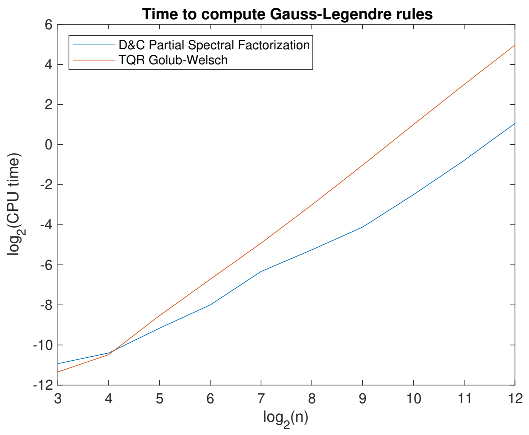

FIG. 6.1. *A log-log plot of the average CPU times for computing Gauss rules by the PSF D&C and TQR G-W algorithms across a range of sizes.*

6. Some computations with the software described. This section examines the performance of the codes in the software package that accompanies this paper. Further examples and scripts that illustrate how the functions of the package can be called are provided in separate files.

All testing is done in MATLAB version R2023b on a Dell Precision desktop computer with an Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz with 16.0 GB of RAM. The computations are carried out in MATLAB with about 15 significant decimal digits. Some also compare with functions in the Chebfun program package [19]. This package, which is implemented in MATLAB, replaces functions by accurate expansions in terms of piece-wise Chebyshev polynomials and allow a user to carry out accurate computations with functions. Chebfun implements the function `jacpts` for computing the nodes and weights of Gauss quadrature rules with a Jacobi weight function

$$w(t) = (1-t)^{\alpha}(1+t)^{\beta}, \qquad -1 < t < 1, \qquad \alpha, \beta > -1. \tag{6.1}$$

Both accuracy and execution time (CPU-time) of the algorithms are illustrated. All timings are done with the MATLAB command `timeit()`. We refer to the divide-and-conquer algorithm applied to the computation of the partial spectral resolution as the `PSF D&C` algorithm. Our implementation of the Golub-Welsch algorithm [29] based on the tridiagonal QR algorithm described by Gates and Gragg [21] is referred to as `TQR G-W`. We compare the accuracy of this method to an implementation of the Golub-Welsch algorithm made available by Meurant [35] and refer to the latter implementation as `MMQ G-W`. The MATLAB function `eig` can be applied to the tridiagonal matrix (2.4) to compute the nodes and weights of the n-node quadrature rule determined by this matrix. This approach ignores the tridiagonal structure of T_n and requires $\mathcal{O}(n^3)$ flops. Finally, we also compare the accuracy with that of the Chebfun function `jacpts`.

We begin by comparing timings of the `PSF D&C` algorithm and the `TQR G-W` algorithm when generating one Gauss-Legendre rule for a varying number of nodes. Figure 6.1 shows the `PSF D&C` algorithm to be faster for all quadrature rules with $n \geq 32$ nodes. Timings are reported for $n = 2^k$ and $k = 3, 4, \ldots$ The reported times are average CPU-times in seconds over several runs.

Figure 6.2 displays timings for the `PSF D&C` algorithm and the `TQR G-W` algorithm when applied to compute pairs of an n-node Gauss and an associated $(n + 1)$-node anti-Gauss

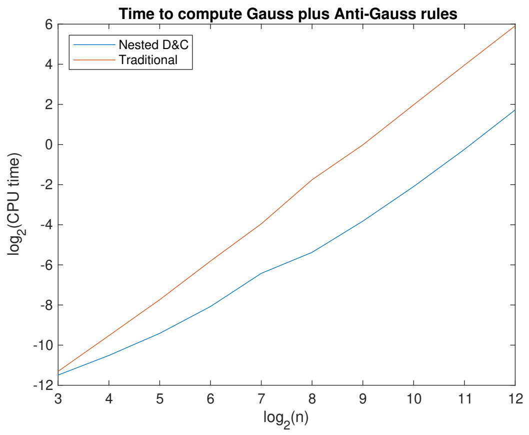

FIG. 6.2. *A log-log plot of the average CPU-times for computing a nested pair of Gauss and anti-Gauss rules using both the nested D&C approach and by using a traditional approach where the n-node Gauss rule and the $(n+1)$-node anti-Gauss rule are both computed with independent applications of the TQR G-W algorithm.*

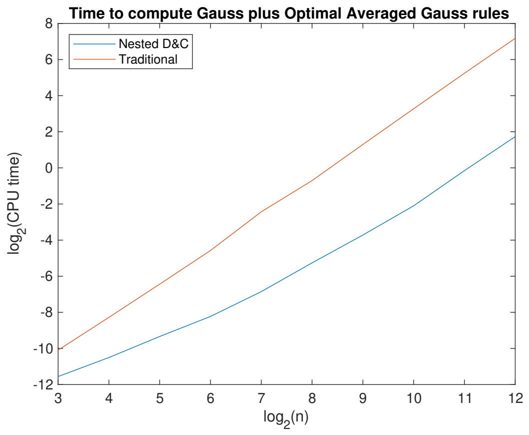

FIG. 6.3. *A log-log plot of the average CPU-times for computing a nested pair of Gauss and optimal averaged Gauss rules using both the nested D&C approach and by using a traditional approach where the n-node Gauss rule and the $(2n+1)$-node optimal averaged Gauss rule are both computed with independent applications of the TQR G-W algorithm.*

quadrature rule for the Legendre measure and $n = 2^k$ with $k = 3, 4, \dots$ The `PSF D&C` algorithm is referred to as "nested", since it computes the Gauss and anti-Gauss rules together for each value of n. The `TQR G-W` algorithm is called twice for each value of n. This approach is referred to as "traditional." The traditional approach can be seen to be slower than the nested D&C approach for all values of n. The timings reported are average CPU-times in seconds over several runs.

Figure 6.3 differs from Figure 6.2 only in that the $(n+1)$-node anti-Gauss rules are replaced by the $(2n+1)$-node optimal averaged rules. The reduction in CPU-time that is achieved by using the `PSF D&C` algorithm can be seen to be even larger than in Figure 6.2.

The following graphs compare the accuracy achieved with the algorithms considered in this paper. The nodes for the n-node Gauss rule with a Jacobi weight function (6.1) with

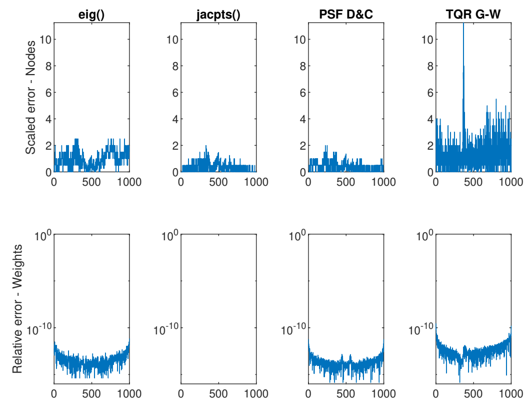

FIG. 6.4. *Errors in the nodes and weights determined by several of the algorithms considered. The TQR G-W method yields the worst accuracy, while the accuracy achieved with PSF D&C is competitive.*

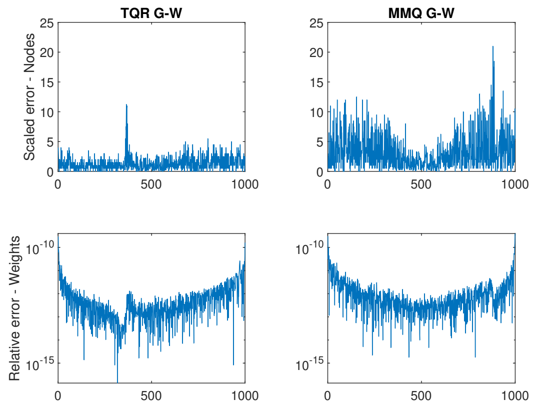

FIG. 6.5. *Errors in the nodes and weights determined by the TQR G-W algorithm and the MMQ G-W algorithm [35].*

$\alpha = \beta = -1/2$ are explicitly known, $t_j = \cos(\frac{2j-1}{2j}\pi)$, $j = 1, 2, \ldots, n$. We compare these nodes to the nodes $\widetilde{t}_j$ computed by the algorithms discussed and plot $|t_j - \widetilde{t}_j|/\epsilon$ where machine epsilon is given by $\epsilon = 2^{-52}$. The top panels of Figure 6.4 display these errors for several methods. The MATLAB function `eig`, the function `jacpts` of Chebfun, and the `PSF D&C` algorithm all yield about the same accuracy, while the errors in the nodes evaluated with the `TQR G-W` method generally are larger.

The bottom panels of Figure 6.4 show the relative errors in the computed weights. This is a semi-log plot in the y-axis to better resolve the differences. The exact weights of the n-node Gauss rule defined above are $w_j = \pi/n$, $j = 1, 2, \ldots, n$. Denote the computed weights by $\widetilde{w}_j$. The panels show the value $|w_j - \widetilde{w}_j|/|w_j|$. The function `jacpts` is seen to determine the weights exactly,[2] while the errors in the weights determined by the `PSF D&C` algorithm are of about the same size as those computed by `eig`; the `TQR G-W` method yields weights with less accuracy.

[2]This is because the `jacpts` code explicitly traps this case and forces a correct answer.

Figure 6.5 performs the same computations but only to compare the two TQR Golub-Welsch approaches. We can see that the Golub-Welsch algorithm presented in this paper is more accurate. Further examples can be found in the files Accuracy_Tests.pdf and Performance_Tests.pdf.

7. Conclusion.

The paper provides software for the application of the divide-and-conquer algorithm presented by Borges and Gragg [9] to the calculation of the nodes and weights a Gauss quadrature rule (2.2) or of pairs of a Gauss rule and a related Gauss-type quadrature rule. Also software for the Golub-Welsch algorithm [29] is made available. Computations show the divide-and-conquer method to require less CPU time on a laptop computer than the Golub-Welsch algorithm for quadrature rules with more than about 32 nodes and typically yields higher accuracy. When pairs of Gauss-type quadrature rules are desired, the divide-and-conquer algorithm requires less CPU time for any number of nodes.

Supplementary material. The accompanying software is available at
https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol61/addition/p121.php
in form of a compressed file entitled GaussRules.zip. Installation details are discussed in the README.md file. Licence information are in License.md.

Acknowledgment. We would like to thank Gérard Meurant for a pointer to his implementation of the Golub-Welsch algorithm.

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.
https://etna.ricam.oeaw.ac.at/vol.9.1999/pp26-38.dir/pp26-38.pdf.

[2] H. Alqahtani, C. F. Borges, D. Lj. Djukić, R. M. Mutavdzić Djukić, L. Reichel, and M. M. Spalević, *Computation of pairs of related Gauss-type quadrature rules*, Appl. Numer. Math., in press.

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

[4] A. H. Bentbib, M. El Guide, K. Jbilou, and L. Reichel, *A global Lanczos method for image restoration*, J. Comput. Appl. Math., 300 (2016), pp. 233–244.

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

[6] C. F. Borges, *An improved formula for Jacobi rotations*, Preprint on arXiv, 2018.
https://arxiv.org/abs/1806.07876v1

[7] ———, *Algorithm 1014: An improved algorithm for hypot(x,y)*, ACM Trans. Math. Software, 47 (2021), Art. 9, 12 pages.

[8] ———, *Fast compensated algorithms for the reciprocal square root, the reciprocal hypotenuse, and Givens rotations*, Preprint on arXiv, 2021. https://arxiv.org/abs/2103.08694

[9] 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.

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

[11] D. Calvetti and L. Reichel, *Tikhonov regularization of large linear problems*, BIT Numer. Math., 43 (2003), pp. 263–283.

[12] 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.

[13] 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.

[14] D. Lj. Djukić, R. M. Mutavdžić Djukić, A. V. Pejčev, and M. M. Spalević, *Error estimates of Gaussian-type quadrature formulae for analytic functions on ellipses - a survey of recent results*, Electron. Trans. Numer. Anal., 53 (2020), pp. 352–382.
https://etna.ricam.oeaw.ac.at/vol.53.2020/pp352-382.dir/pp352-382.pdf

[15] 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, 11 pages.
[16] ———, *Internality of generalized averaged Gauss quadrature rules and truncated variants for modified Chebyshev measures of the third and fourth kinds*, Numer. Algorithms, 92 (2023), pp. 523–544.
[17] D. LJ. DJUKIĆ, L. REICHEL, AND M. M. SPALEVIĆ, *Internality of generalized averaged Gaussian quadrature rules and truncated variants for measures induced by Chebyshev polynomials*, Appl. Numer. Math., 142 (2019), pp. 190–205.
[18] ———, *Internality of generalized averaged Gauss rules and their truncations for Bernstein-Szegő weights*, Electron. Trans. Numer. Anal., 45 (2016), pp. 405–419. https://etna.ricam.oeaw.ac.at/vol.45.2016/pp405-419.dir/pp405-419.pdf
[19] T. A. DRISCOLL, N. HALE, AND L. N. TREFETHEN, *Chebfun Guide*, Pafnuty Publications, Oxford, 2014.
[20] L. FERMO, L. REICHEL, G. RODRIGUEZ, AND M. M. SPALEVIĆ, *Averaged Nyström interpolants for the solution of Fredholm integral equations of the second kind*, Appl. Math. Comput., 467 (2024), Art. 128482, 20 pages.
[21] K. GATES AND W. B. GRAGG, *Notes on TQR algorithms*, J. Comput. Appl. Math., 86 (1997), pp. 195–203.
[22] 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/.
[23] ———, *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, Oxford, 2004.
[24] W. GAUTSCHI AND R. S. VARGA, *Error bounds for Gaussian quadrature of analytic functions*, SIAM J. Numer. Anal., 20 (1983), pp. 1170–1186.
[25] 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.
[26] G. H. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.
[27] G. H. GOLUB AND G. MEURANT, *Matrices, Moments and Quadrature with Applications*, Princeton University Press, Princeton, 2010.
[28] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, 2013.
[29] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.
[30] 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.
[31] 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.
[32] D. R. JANDRLIĆ, DJ. M. KRTINIĆ, LJ. V. MIHIĆ, A. V. PEJČEV, AND M. M. SPALEVIĆ, *Error bounds of Gaussian quadrature formulae with Legendre weight function for analytic integrands*, Electron. Trans. Numer. Anal., 55 (2022), pp. 424–437. https://etna.ricam.oeaw.ac.at/vol.55.2022/pp424-437.dir/pp424-437.pdf
[33] D. P. LAURIE, *Anti-Gaussian quadrature formulas*, Math. Comp., 65 (1996), pp. 739–747.
[34] ———, *Calculation of Gauss-Kronrod quadrature rules*, Math. Comp., 66 (1997), pp. 1133–1145.
[35] G. MEURANT, home page: https://gerard-meurant.fr/.
[36] S. NOTARIS, *Gauss-Kronrod quadrature formulae - a survey of fifty years of research*, Electron. Trans. Numer. Anal., 45 (2016), pp. 371–404. https://etna.ricam.oeaw.ac.at/vol.45.2016/pp371-404.dir/pp371-404.pdf
[37] A. V. PEJČEV, *A note on "Error bounds of Gaussian quadrature formulae with Legendre weight function for analytic integrands" by M. M. Spalević et al.*, Electron. Trans. Numer. Anal., 59 (2023), pp. 89–98. https://etna.ricam.oeaw.ac.at/vol.59.2023/pp89-98.dir/pp89-98.pdf.
[38] L. REICHEL, H. SADOK, AND A. SHYSHKOV, *Greedy Tikhonov regularization for large linear ill-posed problems*, Int. J. Comput. Math., 84 (2007), pp. 1151–1166.
[39] L. REICHEL AND M. M. SPALEVIĆ, *A new representation of generalized averaged Gauss quadrature rules*, Appl. Numer. Math., 165 (2021) pp. 614–619.
[40] ———, *Averaged Gauss quadrature formulas: Properties and applications*, J. Comput. Appl. Math., 410 (2022), Art. 114232, 18 pages.
[41] M. M. SPALEVIĆ, *On generalized averaged Gaussian formulas*, Math. Comp., 76 (2007), pp. 1483–1492.
[42] H. SUGIURA AND T. HASEGAWA, *Error bounds for Gauss-Jacobi quadrature of analytic functions on an ellipse*, Math. Comp., 94 (2025), pp. 359–379.