Fast variants of the Golub and Welsch algorithm for symmetric weight functions

Gérard Meurant

30 rue du sergent Bauchat, 75012 Paris, France

Alvise Sommariva

University of Padua, Italy.

Abstract

In this paper, we investigate variants of the well-known Golub and Welsch algorithm for computing nodes and weights of Gaussian quadrature rules for symmetric weights w in intervals $(-a, a)$ (not necessarily bounded). The purpose is to reduce the complexity of the Jacobi eigenvalue problem stemming from Wilf's theorem and show the effectiveness of these methods for reducing the computer times. Numerical examples on three test problems show the benefits of these variants.

Key words: Algebraic quadrature, symmetric weights, Golub and Welsch algorithm.

1 Introduction

A classical problem in numerical analysis is determining an approximation of the integral

$$I_w(f) = \int_a^b f(x)w(x)\,dx$$

where f is a continuous function defined on the interval (a, b) and w is a positive weight function. In general, $I_w(f)$ is computed via an N-point weighted

Email addresses: gerard.meurant@gmail.com (Gérard Meurant), alvise@math.unipd.it (Alvise Sommariva).

sum, usually called a *quadrature rule*,

$$Q_n(f) = \sum_{i=1}^{N} w_i f(x_i).$$

The abscissas $\{x_i\}_{i=1,\ldots,n}$ are known as nodes while the quantities $\{w_i\}_{i=1,\ldots,n}$ are known as weights [9], [13], [18]. A common approach is to determine $\{x_i\}_{i=1,\ldots,n}$ and $\{w_i\}_{i=1,\ldots,n}$ so as to achieve the maximum algebraic degree of exactness δ, i.e. the maximum positive integer such that $I_w(p) = Q_n(p)$ for any polynomial p of degree less than or equal to δ. From this point of view, the highest δ that one can achieve for an N-point quadrature is obtained by the Gaussian rule for which $\delta = 2N - 1$. In this case, it is well-known that the nodes are the zeros of an orthogonal polynomial of degree N w.r.t. the weight w and that the weights are determined by $w_k = \int_a^b L_k(x)w(x)dx$ where L_k is the k-th elementary Lagrange polynomial.

In the monograph by Wilf [29], it has been noted that starting from the three-term recurrence of the orthogonal polynomials one can determine a tridiagonal matrix J, usually referred to as a *Jacobi matrix*, whose spectral decomposition delivers the nodes and weights of the desired Gaussian rule. In particular the eigenvalues of J provide the nodes $\{x_k\}$, while from the first component of the eigenvectors one easily gets the weights $\{w_k\}$ (see, e.g., [9, p.118], [13, p.153], [18, p.99] and [29, p.55] for details). The advantage of this approach is that reliable linear algebra tools can be used to determine the nodes and weights of the Gaussian rules [12].

The three-term recurrence is available in several ways. In certain instances it is known analytically while other times only numerically, e.g. via moments computation or modification algorithms (see e.g. [12], [13]). The spectral decomposition is usually performed by computation of eigenvalues and eigenvectors of the matrix J via the QR iterative algorithm. Since only the first components of the eigenvectors of J is needed, a clever technique computing only what is necessary has been introduced by Golub and Welsch in [19] and [20]. However, at first glance, this algorithm does not seem to be well suited for even weight functions on symmetric intervals w.r.t. the origin, i.e. $w : (-a, a) \to \mathbb{R}$ such that $w(x) = w(-x)$. For the sake of simplicity we will call these w *symmetric weight functions*; see, for instance, [24].

Since many w's of practical interest have this form, as in the case of Gegenbauer or Hermite weights, in this paper we consider several variants of the Golub and Welsch algorithm that are suitable for these weight functions and allow better performances. In the first one, we compute an N-point Gaussian rule for the symmetric weight function w by means of a $N/2$-point Gaussian rule for a certain nonsymmetric weight function. In the second variant, we use tools from linear algebra to determine the formula via a smaller spectral

problem.

Matlab [1] routines implementing the Golub and Welsch approach are available in the OPQ suite by Gautschi (see [14], [16]). Our goal is to improve upon these routines by implementing our variants in Matlab. We stress that, in the literature, another approach has been introduced by Glaser, Liu and Rokhlin in [17] to compute with $O(N)$ complexity up to a million nodes and weights. This method is implemented in the Matlab package Chebfun [3]. A fast algorithm for Gauss-Legendre rules has been proposed by Bogaert, Michiels and Foster in [1]. Hale and Townsend recently described in [22] how to compute the more general Gauss-Jacobi rules. For a very large number of nodes this method is today the most efficient one. It is now also implemented in Chebfun.

Since in practical problems the number of nodes is rarely larger than hundreds, we have found useful to show that our variants are in general competitive with some recently proposed fast methods for degrees of exactness in the hundreds or thousands and not for only $\delta \leq 100$ as in the classical approach (see [23]). Moreover, one should note that our methods are valid for any symmetric weight function.

We test these algorithms on two classical problems, the computation of the Gaussian quadrature rules for Gegenbauer and Hermite weights. Finally, we show their benefits in the case of a new weight function proposed in [6], [8], [15].

The paper is organized as follows. In §2 we state some basic properties of the nodes $\{x_k\}_{k=1,\ldots,N}$ and of the weights $\{w_k\}_{k=1,\ldots,N}$ of a Gaussian rule when the weight function w is symmetric. In §3, we use these results to show how $\{x_k\}_{k=1,\ldots,N}$ and $\{w_k\}_{k=1,\ldots,N}$ for even N are related to the nodes $\{\tilde{x}_k\}_{k=1,\ldots,N/2}$ and of the weights $\{\tilde{w}_k\}_{k=1,\ldots,N/2}$ of a certain weight function $\tilde{w}$. In §4, we discuss similarly the case in which N is odd. In §5 we determine the three-term recurrence of orthogonal polynomials w.r.t. $\tilde{w}$, knowing that of w. As an alternative, in §6 an approach based on linear algebra is developed. Finally, in §7, §8, §9, §10 numerical tests of all these Matlab implementations are discussed for three families of weight functions.

[1] Matlab is a trademark of The Mathworks.

2 On nodes and weights for a symmetric weight function

Let us suppose that $w : (-a, a) \to \mathbb{R}$ is a symmetric weight function and that we need to compute the Gaussian quadrature rule

$$Q_n(f) = \sum_{i=1}^{N} w_i f(x_i) \tag{1}$$

that is exact for all polynomials of degree $\delta_N = 2N - 1$, i.e. such that

$$\int_{-a}^{a} p(x)w(x)\,dx = Q_N(p)$$

for all p belonging to the space $\mathcal{P}_{\delta_N}$ of polynomials of degree δ_N. Here we do not require a to be finite. Typically $a = 1$ in the case of Gauss-Jacobi or $a = \infty$, e.g. for the Gauss-Hermite weight function. The integer δ_N is known as the *algebraic degree of exactness of the rule*.

Since the weight function is symmetric, one can easily prove that

(1) The orthogonal polynomials $\phi_N \in \mathcal{P}_N$ associated to w are respectively even or odd functions if N is even or odd.
(2) Their N simple zeros, i.e. the nodes of the Gaussian rule, belong to the interval $(-a, a)$ and are symmetric w.r.t. the origin. If N is odd then 0 is a node of the Gaussian rule.
(3) Two symmetric nodes of the Gaussian rule share the same weight.

We introduce the quantities x_k^+ and w_k^+ to denote respectively the k-th *strictly positive* node in the set $\{x_i\}_{i=1}^N$ and the corresponding weight. Consequently, the symmetric Gaussian rule (1) can be rewritten for $N = 2L$ as

$$\sum_{i=1}^{N} w_i f(x_i) = \sum_{i=1}^{L} w_i^+ f(-x_i^+) + \sum_{i=1}^{L} w_i^+ f(x_i^+)$$

and for $N = 2L + 1$ as

$$\sum_{i=1}^{N} w_i f(x_i) = \sum_{i=1}^{L} w_i^+ f(-x_i^+) + \sum_{i=1}^{L} w_i^+ f(x_i^+) + w_0^+ f(0).$$

One can also observe that, owing to the symmetry of the nodes, the orthogonal polynomial ϕ_K is such that, if $K = 2J + 1$ then

$$\phi_K(x) = c_K \cdot x \prod_{i=1}^{K-1} (x - x_i) = c_K \cdot x \prod_{i=1}^{J} (x^2 - (x_i^+)^2)$$

while if $K = 2J$ then

$$\phi_K(x) = c_K \cdot \prod_{i=1}^{K}(x - x_i) = c_K \cdot \prod_{i=1}^{J}(x^2 - (x_i^+)^2).$$

We introduce now the polynomial $\tilde{\phi}_J(x) := c_K \cdot \prod_{i=1}^{J}(x-(x_i^+)^2)$. Consequently for $K = 2J + 1$ we have

$$\phi_K(x) = x\tilde{\phi}_J(x^2),$$

while for $K = 2J$ we get

$$\phi_K(x) = \tilde{\phi}_J(x^2).$$

We finally observe that the rule is exact for all polynomials of degree $\delta_N = 2N - 1$ if and only if it is exact on a basis of orthogonal polynomials for the space $\mathcal{P}_{\delta_N}$. The Gaussian rule, being symmetric, integrates exactly all the orthogonal polynomials of odd degree $2J - 1$ since by symmetry

$$\int_{-a}^{a} \phi_{2J-1}(x)w(x)dx = 0.$$

For the quadrature rule, if N is even, say $N = 2L$, we have

$$\begin{aligned}\sum_{i=1}^{N} w_i\phi_{2J-1}(x_i) &= \sum_{i=1}^{L} w_i^+\phi_{2J-1}(-x_i^+) + \sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) \\ &= -\sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) + \sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) = 0. \end{aligned} \tag{2}$$

For odd $N= 2L + 1$, from $\phi_{2J-1}(0) = 0$, we get

$$\begin{aligned}\sum_{i=1}^{N} w_i\phi_{2J-1}(x_i) &= \sum_{i=1}^{L} w_i^+\phi_{2J-1}(-x_i^+) + \sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) + w_0^+\phi_{2J-1}(0) \\ &= -\sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) + \sum_{i=1}^{L} w_i^+\phi_{2J-1}(x_i^+) + 0 = 0.\end{aligned}$$

This shows that the quadrature rule is exact for odd polynomials. The quadrature problem is therefore reduced to determine N nodes and weights of a symmetric rule that is exact only on even orthogonal polynomials ϕ_{2K} (w.r.t. the function w), with $2K \leq \delta_N = 2N - 1$.

3 Gaussian rules for symmetric weight functions

Let $d\lambda(t) = w(t)dt$ be a symmetric weight function on $[-a, a]$, $a > 0$, and

$$\pi_{2k}(t; d\lambda) = \pi_k^+(t^2),\ \pi_{2k+1}(t; d\lambda) = t\pi_k^-(t^2) \tag{3}$$

the respective (monic) orthogonal polynomials. By [13, Theorem 1.18], the $\pi_k^\pm$ are the monic polynomials orthogonal on $[0, a^2]$ with respect to the measure $d\lambda^\pm(t) = t^{\mp 1/2} w(t^{1/2})dt$. Let $\alpha_k^\pm$, $\beta_k^\pm$ be the recursion coefficients of $\{\pi_k^\pm\}$.

3.1 Case I: N even

Let us consider first the case of even N, say $N = 2L$. Then, suppose that

$$\int_0^{a^2} f(t)d\lambda^+(t) = \int_0^{a^2} f(t)\frac{w(t^{1/2})}{t^{1/2}}dt = \sum_{s=1}^{L} \lambda_s^G f(t_s^G),\ f \in \mathcal{P}_{2L-1} \tag{4}$$

is the L-point Gauss quadrature formula for $d\lambda^+$. Then, putting $t = x^2$, we get

$$2\int_0^a f(x^2)\frac{w(x)}{x}x\,dx = \sum_{s=1}^{L} \lambda_s^G f(t_s^G),\ f \in \mathcal{P}_{2L-1},$$

that is, in view of the symmetry of the weight function w,

$$\int_{-a}^a f(x^2)w(x)dx = \sum_{s=1}^{L} \lambda_s^G f(t_s^G),\ f \in \mathcal{P}_{2L-1}.$$

Put $q(x) \equiv f(x^2)$, $q \in \mathcal{P}_{4L-2} = \mathcal{P}_{2N-2}$. Then

$$\int_a^a q(x)w(x)dx = \sum_{s=1}^{L} \lambda_s^G q(\sqrt{t_s^G}) = \frac{1}{2}\sum_{s=1}^{N/2} \lambda_s^G \left(q(\sqrt{t_s^G}) + q(-\sqrt{t_s^G})\right).$$

This is an N-point quadrature rule exact for all even polynomials of degree $\leq 2N-2$. It is trivially exact for all odd polynomials. Hence it is exact for all polynomials of degree $\leq 2N-1$, i.e., it is the Gauss formula for $d\lambda$ when N is even.

3.2 Case II: N odd

Let us now consider the case of odd N, say $N = 2L+1$. Then, suppose that

$$\int_0^{a^2} f(t)d\lambda^+(t) = \lambda_0^R f(0) + \sum_{s=1}^{L} \lambda_s^R f(t_s^R),\ f \in \mathcal{P}_{2L}, \tag{5}$$

is the $(L+1)$-point Gauss-Radau formula for $d\lambda^+$. Putting $t = x^2$, we get

$$2\int_0^a f(x^2)w(x)\,dx = \lambda_0^R f(0) + \sum_{s=1}^{L} \lambda_s^R f(t_s^R),\ f \in \mathcal{P}_{2L}$$

or, setting $q(x) \equiv f(x^2)$, $q \in \mathcal{P}_{4L} = \mathcal{P}_{2N-2}$,

$$\begin{aligned}\int_{-a}^a q(x)w(x)dx &= \lambda_0^R q(0) + \sum_{s=1}^{L} \lambda_s^R q(\sqrt{t_s^R}) \\ &= \lambda_0^R q(0) + \frac{1}{2}\sum_{s=1}^{(N-1)/2} \lambda_s^R (q(\sqrt{t_s^R}) + q(-\sqrt{t_s^R})).\end{aligned} \tag{6}$$

By the same argument as in Case I, this is the N-point Gauss formula for $d\lambda$ when N is odd.

4 Computational issues

The L-point Gauss formula (4) can be computed by the Golub and Welsch algorithm applied to the Jacobi matrix of order L,

$$\mathbf{J}_L^G(d\lambda^+) = \begin{bmatrix} \alpha_0^+ & \beta_1^+ & & 0 \\ \beta_1^+ & \alpha_1^+ & \ddots & 0 \\ & \ddots & \ddots & \beta_{L-1}^+ \\ 0 & \dots & \beta_{L-1}^+ & \alpha_{L-1}^+ \end{bmatrix} \tag{7}$$

that is, $\{t_s^G\}$ are the eigenvalues of (7), and $\{\lambda_s^G\}$ given by $\lambda_s^G = \beta_0^+ \mathbf{v_{s,1}}$ where $\beta_0^+ = \int_0^{a^2} d\lambda^+(t)$ and $\mathbf{v_{s,1}}$ is the first component of the normalized eigenvector $\mathbf{v}_s$ belonging to the eigenvalue t_s^G (cf. [13, Theorem 3.1]). This is implemented in `gauss.m` in Gautschi's OPQ Matlab toolbox [14]. Similarly, the $(L+1)$-point Gauss-Radau formula (5) can be computed by the same algorithm applied to the Jacobi-Radau matrix of order $L+1$,

$$\mathbf{J}_{L+1}^R(d\lambda^+) = \begin{bmatrix} \mathbf{J}_L^R(d\lambda^+) & \sqrt{\beta_L^+}\mathbf{e}_L \\ \sqrt{\beta_L^+}\mathbf{e}_L & \alpha_L^R \end{bmatrix},\ \mathbf{e}_L^T = [0,0,\dots,1] \in \mathbb{R}^L \tag{8}$$

where $\alpha_L^R = -\beta_L^+ \pi_{L-1}(0; d\lambda^+)/\pi_L(0; d\lambda^+)$ (cf.[13, Theorem 3.2]). This is implemented in `radau.m` in OPQ. Incidentally, the nodes $\{t_s^R\}$ are the zeros of π_L^- (see [13, 3.1.1.2]). It is also known ([5, Chapters 8-9]) that the recursion

coefficients of $d\lambda$ are $\alpha_k = 0$, and

$$\left.\begin{array}{l} \beta_1 = \alpha_0^+, \\ \end{array}\begin{array}{l} \beta_{2k} = \beta_k^+ / \beta_{2k-1} \\ \beta_{2k+1} = \alpha_k^+ - \beta_{2k} \end{array}\right\} \; k = 1, 2, 3, \ldots \tag{9}$$

These equalities allow to compute the necessary recursion coefficients α_k^+, β_k^+ of the monic orthogonal polynomials w.r.t. $d\lambda^+$ from the correspective α_k, β_k of $d\lambda$. If β_k has a finite limit as $k \to \infty$, then the nonlinear recursion (9) is numerically stable (cf. [10, p. 477]).

5 A linear algebra approach for symmetric weight functions

The previous sections have identified the orthogonal polynomials for the symmetric problem. In this section we show that the nodes and weights of the quadrature rule for a symmetric weight function can also be obtained from a purely algebraic point of view. We start from the three-term recurrence satisfied by the orthonormal polynomials associated to the weight function w,

$$\gamma_j p_j(\lambda) = (\lambda - \omega_j) p_{j-1}(\lambda) - \gamma_{j-1} p_{j-2}(\lambda), \quad j = 1, 2, \ldots, N \tag{10}$$

$$p_{-1}(\lambda) \equiv 0, \quad p_0(\lambda) \equiv 1/\gamma_1.$$

By symmetry of the weight function w the coefficients ω_j, $j = 1, \ldots, N$, are all zero. The nodes of the Gauss quadrature rule are the eigenvalues of a symmetric tridiagonal matrix with a zero diagonal. From (10) the eigenvalue problem can be written as

$$Jx = \begin{bmatrix} 0 & \gamma_1 & & & & \\ \gamma_1 & 0 & \gamma_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \gamma_{N-2} & 0 & \gamma_{N-1} & \\ & & & \gamma_{N-1} & 0 & \end{bmatrix} = \lambda x. \tag{11}$$

The weights, up to a constant factor, are given by the squares of the the first components of the normalized eigenvectors. Now we use an old trick. We reorder the equations and the unknowns x_i. We first take the odd numbered equations and then the even numbered ones. For the unknowns we take first the even components and then the odd ones. Let x_O (resp. x_E) be the vector of the odd (resp. even) components of the eigenvector x. Then, the eigenvalue

problem can be written as

$$\begin{pmatrix} F & 0 \\ 0 & F^T \end{pmatrix} \begin{pmatrix} x_E \\ x_O \end{pmatrix} = \lambda \begin{pmatrix} x_O \\ x_E \end{pmatrix}.$$

We can eliminate x_E from these equations to obtain

$$FF^T x_O = \mu x_O, \quad \mu = \lambda^2. \tag{12}$$

The matrix F is lower bidiagonal. The number of rows (resp. columns) is the number of even (resp. odd) components of x. The matrix F is square if N is even and has one more row than columns if N is odd. Nevertheless, it is straightforward to see that the product FF^T is a square tridiagonal matrix (with a nonzero diagonal). Hence we have reduced the eigenvalue problem (11) of order N to the problem (12) of order almost $N/2$.

The positive nodes of the quadrature rule are the square roots of the eigenvalues of FF^T. When N is odd this matrix is singular and we have a zero eigenvalue. The matrix FF^T being semi-positive definite, all the other eigenvalues are strictly positive. The squares of the first components of x can be obtained by rescaling the eigenvectors of (12). If N is even, the first components of the normalized eigenvectors of (12) have to be divided by $\sqrt{2}$. If N is odd we do the same except for the eigenvector corresponding to the zero eigenvalue. The weights are then obtained by squaring the first components.

6 Implementations details

Starting in the 1960s the nodes and weights of quadrature rules were computed by brute force by solving systems of nonlinear equations stating the degree of exactness of the rule. However, the method of choice today is to compute the nodes and weights using the tridiagonal Jacobi matrix corresponding to the orthogonal polynomials associated with the given measure and interval of integration.

The result saying that the nodes are given by the eigenvalues of the Jacobi matrix and the weights are given by the squares of the first components of the normalized eigenvectors was already known at the beginning of the 1960s; see for instance Wilf [29]. Golub and Welsch [19], [20] used these results and devised an algorithm based on a QR iteration tailored for tridiagonal matrices with a Wilkinson-like shift to compute the nodes and weights. It is constructed in such a way that only the first components of the eigenvectors are computed. Moreover, as an addendum to [20] there was a microfiche containing the Algol procedures implementing their algorithm (note that there is a typographical

error in the part of the code symmetrizing the tridiagonal matrix). This paper and the contents of the microfiche have been reprinted in [2].

In the Fortran package ORTHPOL [11], Gautschi implemented the Golub and Welsch algorithm as well as many other codes for determining Gaussian rules. Later on, in [14], he provided a software package for Matlab, called OPQ, implementing the computation of the nodes and weights. In this classical approach once the three-term recurrence of the monic orthogonal polynomials has been computed, the code `gauss` calls the Matlab built-in function `eig` (which is quite fast) to compute eigenvalues and eigenvectors of the tridiagonal Jacobi matrix, thus determining the nodes and weights of the Gaussian quadrature rule via Wilf's theorem [29, Thm. 5, p.55]. However, all components of the eigenvectors are computed contrary to the genuine Golub and Welsch approach. Moreover, apparently, Matlab does not take into account the fact that the matrix is tridiagonal. This does not really matter if the number of nodes N is small. Such a method is sometimes confused with the Golub and Welsch algorithm (see, e.g. [23]). However, we will see that computing only the first components can make a big difference for large values of N. A Matlab (almost verbatim) translation of the Golub and Welsch routines was provided by Meurant [25] as a toolbox accompanying the book [18]. We denote this code by `GW`. It turns out that the Matlab implementation of the Golub and Welsch algorithm can be optimized by vectorizing most of the statements. This code is denoted as `GWo`.

A possible alternative consists in computing only the quadrature nodes $\{x_j\}$ as eigenvalues of the Jacobi matrix, obtaining the weights $\{w_j\}$ via evaluation of orthogonal polynomials. In fact, for a general weight function w having p_k as normalized orthogonal polynomial of degree k, if $\{x_j\}_{j=1,\ldots,N}$ are the nodes of the N-point Gaussian rule then the weights satisfy the equality

$$1/w_j = \sum_{k=0}^{N} (p_k(x_j))^2, \quad j = 1, \ldots, N \tag{13}$$

(see [9, p.119] for details and [26]). We also take this variant (denoted as `Ew`) into account. However, we must warn the reader that computing the weights using the three-term recurrence may not be always a stable process. It is sometimes better to normalize the polynomials in a different way. This is the case, for instance, for the Hermite weight function.

We will use the algorithms described in sections 2 to 5 taking advantage of symmetry in several different ways and we will compare their results to the Matlab function `gauss` from OPQ. The algorithms we consider are:

- `SymmAG`, the algorithm of Section 3, using `gauss` (that is, computing all the components of the eigenvectors) for the reduced system,
- `SymmAGWo`, the same algorithm but using the optimized Golub and Welsch

`GWo` code for the reduced system,
- `SymmAw`, the algorithm of Section 3 using Matlab's `eig` for the eigenvalues and the three-term recurrence of the polynomials for the weights, as in (13),
- `SymmMGWo`, the matrix algorithm of Section 5 with `GWo` for the reduced system,
- `SymmMw`, the algorithm of Section 5 computing the nodes with `eig` and the three-term recurrence of the polynomials for the weights.

As we said above, new algorithms that compute nodes and weights of classical Gaussian rules for some weight functions in $O(n)$ operations have been suggested in [17], [22] and accordingly to this strategy, Matlab routines for the computation of nodes and weights have been proposed by the Chebfun team in [3] (see also [21], [28]).

7 Numerical experiments with Gegenbauer weight functions

As a first numerical example we consider the computation of Gaussian rules for Gegenbauer weights (also known as *ultraspherical weights*), i.e. nodes and weights of the quadrature formula

$$\int_{-1}^{1} f(x)(1-x^2)^{\mu-1}dx \approx \sum_{i=1}^{N} w_k f(x_k), \quad \mu > 0$$

whose degree of exactness is $\delta_N = 2N - 1$.

The weight function $w(x) = (1-x^2)^{\mu-1}$ is of course symmetric in the interval $(-1, 1)$ and corresponds to a Jacobi weight $w_{\alpha,\beta}(x) = (1-x)^{\alpha}(1+x)^{\beta}$ for $\alpha = \beta = \mu - 1 > -1$. In the numerical experiments, we take $\mu = 7/4$.

The computations were done on a PC using an Intel core i5-2430M CPU at 2.40 Ghz with 6 GB of RAM and Matlab R2010b. We chose to display ratios of computing times relative to `gauss` from the OPQ toolbox since this is generally considered as the "standard" Matlab implementation of the Golub and Welsch algorithm.

Let us first compare the functions `gauss`, `GW`, `GWo` and `Ew` which compute the nodes with `eig` (requiring only the eigenvalues) and the weights with the three-term recurrence. The speed-ups relative to `gauss` are given in Figure 1. To obtain reliable time measurements the tests are repeated 100 times when the maximum value of N is 200 and 10 times when the maximum value is 2000. Starting around $N = 150$, `GW` is faster than `gauss`. The optimized version `GWo` is faster than `GW` and its advantage increases with N. `Ew` is faster than the other methods for N smaller than 700. This can be explained by the fact that `eig` is a built-in function. Figure 2 shows the $\log_{10}$ of the constants C, assuming that

the computer times are CN^γ with $\gamma = 3$ for `gauss` and $\gamma = 2$ for `GW`, `GWo` and `Ew`. Although the constant is much smaller for `gauss`, but, as it is well known, the computing time is proportional to N^3 while it is proportional to N^2 for the Golub and Welsch algorithm. The constant for the optimized version is a little smaller than for the "genuine" `GW` version. Although the computing times of `GW` and `GWo` are proportional to N for one step of the algorithm, the number of iterations needed to obtain the eigenvalues is also increasing linearly with N. This explains the $O(N^2)$ computing times. As we can see, `Ew` is not really $O(N^2)$. This explains why it is slower than `GW` and `GWo` for N large. From now on we will use the optimized version `GWo` as the Golub and Welsch algorithm, instead of `GW`.

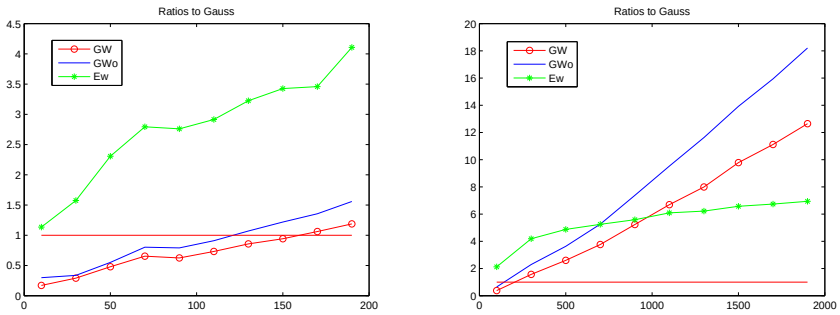

Fig. 1. Comparison of the speed-ups of the rules `GW`, `GWo` and `Ew` relative to `gauss`. On the left figure, $N = 10, 30, \ldots, 200$ while on the right, $N = 100, 300, \ldots 2000$.

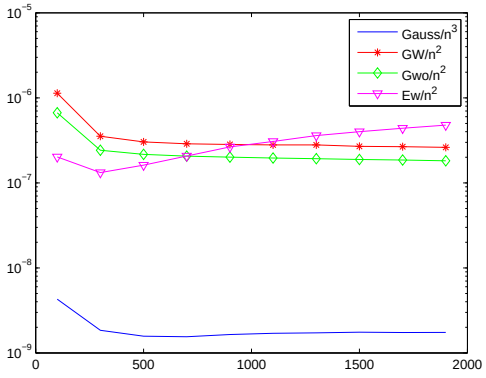

Fig. 2. Computer times divided by N^3 for `gauss` and divided by N^2 for `GW`, `GWo` and `Ew`

Figure 3 displays the ratios of the computer times relative to `gauss`. The `GLR` algorithm is from Chebfun [3] (without a switch to the Golub and Welsch algorithm for N small) and the algorithm proposed in [22], that we will name `Hale-T`, is from the package `quadpts` [21]. Note that to be fair to these algorithms, we have taken into account the times for computing the coefficients of the orthogonal polynomials for the weight function for all methods. This is done using the function `r_jacobi` in the package OPQ from Gautschi [16]. The speedups are the average over 100 tests, while in the second test for large

N, 10 experiments have been considered. The left part of the figure is for small values of N. We see that `SymmAG` is much faster than `gauss` and has a better ratio than `GW` and `GWo` because it uses a built-in function. The right part of the figure shows that this is true until $N = 200$. However, for $N < 1200$ `SymmAw` and `SymmMw` are much faster than the other methods, based on the Golub-Welsch algorithm. For N larger than 200, the variants `SymmAGW` and `SymmAGWo` are much faster than `SymmAG` whose performance is stagnating. We note also that `SymmAGWo` is a little faster than the matrix approach `SymmMGwo`. After $N = 1200$ `SymmAw` and `SymmMw` are levelling off and `SymmAGW` and `SymmAGWo` become faster. The "pure" `GLR` (without a switch to the methods similar to `gauss`) function is slower than `gauss` for $N < 160$. For small N the symmetric variants of Golub and Welsch are faster than `GLR`. But, `GLR` becomes faster than the other Golub-Welsch based methods for $N > 1300$ as can be seen on the right part of the figure. On the other side the new method by Hale and Townsend provides a better performance for $N > 850$. We do not compare our algorithms with those introduced by Bogaert, Michiels and Foster since they exclusively compute Gauss-Legendre rules.

Nevertheless, the ratios of `SymmAGwo` and `SymmMGWo` are, of course, still growing because their complexity is better than that of `gauss`.

For $N = 100$, the speedups for `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmAw`, `SymmMw`, `GLR` and `Hale-T` are respectively, 3.22, 2.30, 2.31, 8.32, 8.07, 0.43 and 0.58. For $N = 1000$, the speed-ups are respectively, 9.05, 38.15, 31.09, 39.76, 40.29, 27.45 and 58.74.

The maximum norm of the differences of the nodes and weights with those computed by `gauss` are of the order 10^{-14}. Thus, the fastest algorithms for $N < 800$ are obtained by using the algorithms of sections 3 and 5, computing the nodes with `eig` and the weights using the three-term recurrence. The Golub and Welsch algorithm, coupled with the symmetrizations of the rule, is also competitive with `quadpts` for computing Gauss-Gegenbauer rules with algebraic degree of exactness $\delta_N < 1700$.

It is important to observe that the results may vary from computer to computer, as well as for different Matlab versions due to possible changes in the implementation of the Matlab built-in function `eig`. This example shows that for the set of algorithms we considered, there is no method which is the best for all values of N. Note also that the methods using `eig` have an advantage since this is a Matlab built-in function, but when our variants are faster the conclusions will be the same with Fortran or C implementations of the algorithms.

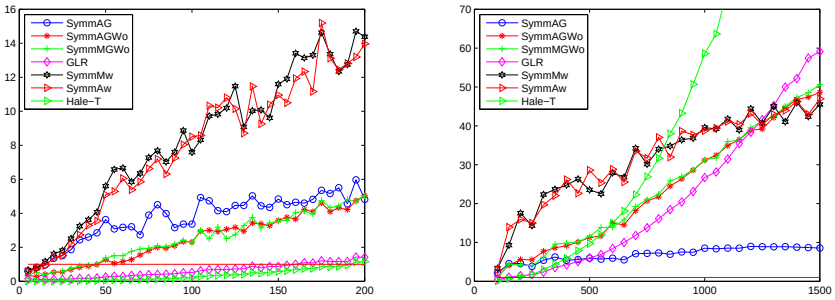

Fig. 3. Comparison of the speed-ups of the rules w.r.t. `gauss`, for the Gegenbauer weight function $w(x) = (1-x^2)^{3/4}$, in the interval $(-1, 1)$. On the left part, $N = 5, 10, \ldots, 200$ while on the right, $N = 100, 150, \ldots, 1500$.

8 Numerical experiments with Hermite weight functions

In this section, in order to illustrate the behavior of our algorithms on unbounded intervals, we consider the problem of the computation of Gaussian rules for the Hermite weight function, i.e. formulas

$$\int_{-\infty}^{\infty} f(x) \cdot \exp(-x^2)\, dx \approx \sum_{i=1}^{N} w_k f(x_k),$$

whose degree of precision is $\delta_N = 2N - 1$.

Then the algorithm denoted as `gauss` uses the function `r_hermite` from Gautschi's package, followed by `eig` to compute the nodes $\{x_k\}$ and weights $\{w_k\}$. Alternatively, they can be obtained by `hermpts` provided by the Chebfun team [4] that uses the Glaser-Liu-Rokhlin algorithm.

As in the previous section, we use `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmAw` and `SymmMw` to determine the nodes and the weights of the Gaussian rule, this time w.r.t. the Hermite weight.

We observe that, introducing the generalized Laguerre weight function $L(x, \mu) = x^{\mu} \cdot \exp(-x)$, then $\tilde{w}(x) = \frac{\exp(-x)}{2\sqrt{x}} = 0.5 \cdot L(x, -0.5)$. Thus one can compute the three-term recurrence of the monic $\tilde{w}$-orthogonal polynomials via the Matlab routine `r_Laguerre` by Gautschi and the nodes and weights by the Golub and Welsch algorithm. The routine `SymmLGWo` implements these ideas.

In Figure 4, we compare the speed-ups of `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmLGWo`, `SymmMw`, `SymmAw` and `hermpts` w.r.t. the classical algorithm `gauss`. For small values of N (left part of the figure) the conclusions are more or less the same than for the Gegenbauer weight function. The algorithms `SymmAGWo`, `SymmMGWo` become faster than `gauss` for N between 50 and 60 as for the Gegenbauer weight but `hermpts` also, contrary to the previous example. The fastest algo-

rithms for small N are `SymmAw` and `SymmMw`.

Since GLR through `hermpts` is faster than it was before, it becomes faster than `SymmAGWo` and `SymmMGWo` for $N > 400$. It becomes faster than `SymmMw` and `SymmAw` for $N > 700$.

For $N = 100$, the speed-ups for `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmLGWo`, `SymmMw`, `SymmAw` and `hermpts` are respectively, $4.21, 1.76, 1.38, 1.98, 7.55, 6.67$ and 1. For $N = 1000$, the speed-ups are respectively, $9.90, 35.08, 29.09, 34.35, 49.79, 49.11$ and 61.90.

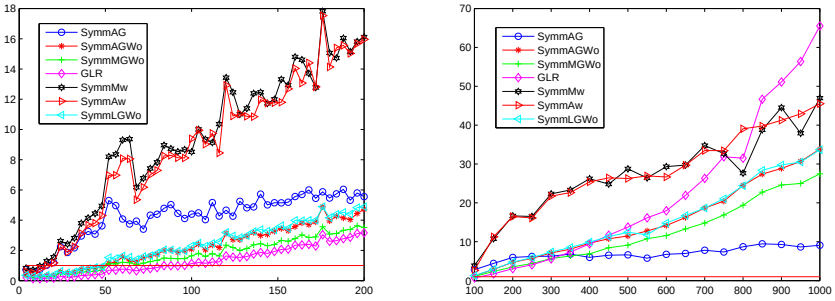

Fig. 4. Comparison of the speed-ups of the rules w.r.t. `gauss`, for the Hermite weight function $w(x) = \exp(-x^2)$, in the interval $(-\infty, \infty)$. On the left part, $N = 5, 9, \ldots, 200$ while on the right, $N = 100, 150, \ldots, 1000$.

9 Numerical experiments with Trigonometric Gaussian cubature

In some recent papers [6], [7], [8], [15], the authors have considered the problem of constructing a quadrature formula with $n+1$ angles and positive weights, exact in the $(2n+1)$-dimensional space of trigonometric polynomials of degree $\leq n$ on intervals with length smaller than 2π. In particular, the following result has been proven in [6].

Theorem 1 *Let $\{(\xi_j, \lambda_j)\}_{1 \leq j \leq n+1}$, be the nodes and positive weights of the algebraic Gaussian quadrature formula for the weight function*

$$w(x) = \frac{2\sin(\omega/2)}{\sqrt{1 - \sin^2(\omega/2)x^2}}, \quad x \in (-1, 1). \tag{14}$$

Then, denoting by $\mathbb{T}_n([-\omega, \omega]) = span\{1, \cos(k\theta), \sin(k\theta), 1 \leq k \leq n, \theta \in [-\omega, \omega]\}$, the $(2n+1)$-dimensional space of trigonometric polynomials on $[-\omega, -\omega]$, we have

$$\int_{-\omega}^{\omega} f(\theta)d\theta = \sum_{j=1}^{n+1} \lambda_j f(\phi_j), \quad \forall f \in \mathbb{T}_n([-\omega, -\omega]), \quad 0 < \omega \leq \pi$$

where

$$\phi_j = 2\arcsin\left(\sin(\omega/2)\xi_j\right) \in (-\omega, \omega), \quad j = 1, 2, \ldots, n+1.$$

The weight function w in (14) is of course symmetric, but at this time it is not known whether any second order differential equation exists that determines the orthogonal polynomials and would allow the application of Glaser-Liu-Rokhlin techniques. On the other hand, in [6], [8], [15], the authors have been able to compute in different ways the three-term recursion of the symmetric weight w described in (14). Note that, without the factor $2\sin(\omega/2)$, the weight function is something in-between the Legendre weight function ($\omega = 0$) and the Chebyshev weight function ($\omega = \pi$).

The purpose of these rules, as shown in [7], [8], is to provide multidimensional product rules on elliptical sectors and spherical domains. This entails the computation of many Gaussian quadrature rules with different values of ω, requiring the efficient computation of the basic univariate rule for the weight function w introduced in (14).

In Figure 5, we compare the speed-ups of `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmMw`, `SymmAw` w.r.t. the classical algorithm `gauss` for $\omega = \pi/2$. We note that numerical underflow problems due to the algorithms proposed in [6], [8], [15], do not allow the computation of the three-term recurrence for $N > 510$. Thus, we limit our experiments to $N = 500$. `SymmMw` and `SymmAw` are the fastest algorithms but the acceleration factor is smaller than what it was for the Gegenbauer and Hermite weight functions. All variants are faster than `gauss` for $N > 50$.

For $N = 100$, the speed-ups for `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmMw` and `SymmAw` are respectively, $2.07, 1.58, 1.39, 2.32$ and 2.25. For $N = 500$, the speed-ups are respectively, $4.36, 5.76, 5.45, 8.29$ and 8.21.

10 Conclusion

In this paper we have considered some variants of the well-known Golub and Welsch algorithm for the computation of Gaussian quadrature rules. We have shown that, depending on the degree of precision, the performance of Matlab implementations can be enhanced by vectorizing the original code or computing the weights by mean of the three-term recurrence.

Furthermore, we have analyzed the case of symmetric weight functions, introducing some new methods, based on the three-term recurrence of the orthogonal polynomials. Some of them are based on exploiting the nodes and the

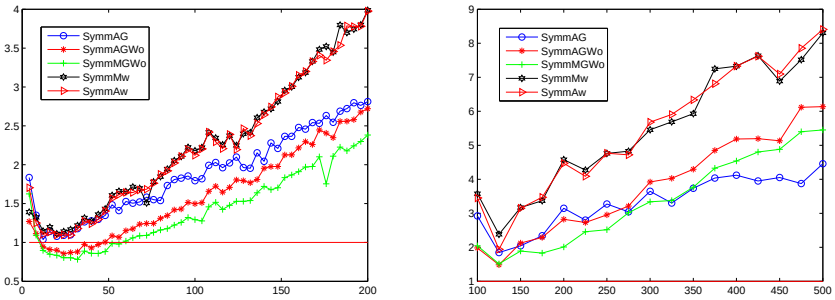

Fig. 5. Comparison of the speed-ups of the rules `SymmAG`, `SymmAGWo`, `SymmMGWo`, `SymmMw`, `SymmAw` w.r.t. `gauss`, for the weight function $w(x) = \frac{2\sin(\omega/2)}{\sqrt{1-\sin^2(\omega/2)x^2}}$, in the interval $(-1,1)$, for $\omega = \pi/2$. On the left part, $N = 5, 9, \ldots 100$ while on the right, $N = 100, 125, \ldots, 500$.

weights of a Gaussian rule for a different weight function $\tilde{w}$ and then deriving those for w, some others use a purely linear algebra approach.

We have implemented these routines in Matlab and compared them on Gegenbauer and Hermite weight functions, showing that the methods are competitive with the Glaser-Liu-Rokhlin algorithms implemented in Chebfun for degrees of exactness respectively $\delta_N < 2600$ and $\delta_N < 1600$ and provide a certain speed-up w.r.t. the commonly used codes in the OPQ suite. We have also tested the new algorithm by Hale and Townsend on a Gegenbauer weight function and seen that our routines `SymmMw` and `SymmAw` are competitive for a degree of exactness $\delta_N < 1700$.

A test on a sub-range Chebyshev weight function has also been considered, exhibiting smaller computer times than the codes previously known. In all the examples, better performances are outlined even for moderate degrees of exactness.

Finally we stress that our algorithms can be used for any symmetric weight function, as demonstrated in the third example in section 9, contrary to some other fast methods for which one needs to know the associated differential equation or some asymptotic developments.

Matlab codes implementing our algorithms are available at the homepage [27].

Acknowledgements

The authors thank W. Gautschi and the referees for some useful comments that improved and simplified the paper. We also thank N. Hale for providing some insights on his routines.

References

[1] I. Bogaert, B. Michiels and J. Foster, $\mathcal{O}(1)$ *computation of Legendre polynomials and Gauss-Legendre nodes and weights for parallel computing*, SIAM J. Sci. Comput, 34 No.2 (2012), pp. C83–C101.

[2] R.H. Chan, C. Greif and D.P. O' Leary Eds., *Milestones in matrix computation: the selected works of Gene H. Golub with commentaries*, Oxford University Press, Oxford (2007).

[3] Chebfun Team, Jacpts.m, (http://www.mathworks.com/matlabcentral/fileexchange/23972-chebfun/content/ chebfun/jacpts.m).

[4] Chebfun Team, Hermpts.m, (http://mathworks.com/matlabcentral/fileexchange/23972-chebfun/content/ chebfun/hermpts.m).

[5] T.S. Chihara, *An introduction to orthogonal polynomials*, Gordon and Breach, New York, (1978).

[6] G. Da Fies and M. Vianello, *Trigonometric Gaussian quadrature on subintervals of the period*, Electron. Trans. Numer. Anal., 39 (2012), pp. 102–112.

[7] G. Da Fies and M. Vianello, *Algebraic cubature on planar lenses and bubbles*, Dolomites Res. Notes Approx., 5 (2012), pp. 7–12.

[8] G. Da Fies, A. Sommariva and M. Vianello, *Algebraic cubature on generalized elliptical sectors*, Appl. Numer. Math., 74 (2013), pp. 49–61.

[9] P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Second edition. Dover Publications, Inc. (1984).

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

[11] W. Gautschi, *Algorithm 726: ORTHPOL. A package of routines for generating orthogonal polynomials and Gauss-type quadrature rules*, ACM Transactions on Mathematical Software, Volume 20, Issue 1, (1994), pp. 21–62.

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

[13] W. Gautschi, *Orthogonal Polynomials. Computation and Approximation.* Oxford Science Publications, (2004).

[14] W. Gautschi, *Orthogonal polynomials (in Matlab)*, J. Comput. Appl. Math., 178 1-2 (2005), pp. 215–234.

[15] W. Gautschi, *Sub-range Jacobi polynomials*, Numer. Algorithms 61 (2012), no. 4, pp. 649–657.

[16] W. Gautschi, OPQ suite,
(http://www.cs.purdue.edu/archives/2002/wxg/codes).

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

[18] G.H. Golub and G. Meurant, *Matrices, Moments and Quadrature with Applications*, Princeton Series in Applied Mathematics, 2010.

[19] G.H. Golub and J.H. Welsch, *Calculation of Gauss quadrature rules*. Technical report no. CS 81, November 3, 1967. Computer Science Department, School of Humanities and Sciences, Stanford University.

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

[21] N. Hale, quadpts,
(https://github.com/nickhale/quadpts/).

[22] N. Hale and A. Townsend, *Fast and accurate computation of Gauss-Legendre and Gauss-Jacobi nodes and weights*, SIAM J. Sci. Comput, 35 No.2 (2013), pp. A652–A674.

[23] N. Hale and L.N. Trefethen, *Chebfun and Numerical quadrature*, Sci. China Math. 55 (2012), no. 9, pp. 1749–1760.

[24] G. Mastroianni and G.V. Milovanovic, *Interpolation Processes, Basic Theory and Applications*, Springer, (2008), pp. 102–103.

[25] G. Meurant, Personal homepage,
(http://gerard.meurant.pagesperso-orange.fr/).

[26] G.V. Milovanovic and A.S. Cvetkovic, *Note on a construction of weights in Gauss-type quadrature rule*, Facta Univ. Ser. Math. Inform., 15 (2000), pp. 69–83.

[27] A. Sommariva, homepage,
(http://www.math.unipd.it/ alvise/software.html).

[28] L.N. Trefethen, *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics (2013), Philadelphia, PA, 2013.

[29] H.S. Wilf, *Mathematics for the Physical Sciences*, John Wiley, New York, (1962). [Reprinted in 1978 by Dover Publications, New York.]