A NONLINEAR OPTIMIZATION PROCEDURE FOR GENERALIZED GAUSSIAN QUADRATURES

JAMES BREMER, ZYDRUNAS GIMBUTAS, AND VLADIMIR ROKHLIN

Abstract. We present a new nonlinear optimization procedure for the computation of generalized Gaussian quadratures for a broad class of functions. While some of the components of this algorithm have been previously published, we present a simple and robust scheme for the determination of a sparse solution to an underdetermined nonlinear optimization problem which replaces the continuation scheme of the previously published works. The performance of the resulting procedure is illustrated with several numerical examples.

1. Introduction

Classical Gaussian quadrature rules are optimal formulas for the evaluation of integrals of the form

$$\int_a^b P(x)\omega(x)\,dx \tag{1.1}$$

where $P(x)$ is a polynomial and $\omega : [a,b] \to \mathbb{R}^+$ is an essentially arbitrary nonnegative weight function; they are optimal in sense that an n-point Gaussian quadrature rule integrates polynomials of degree $n-1$ exactly with respect to the weight function ω. They are extremely efficient as long as functions to be integrated are smooth or of the form $f(x) = g(x)w(x)$ where $g(x)$ is smooth and $w(x)$ is fixed and known *a priori*. However, they perform poorly in a number of instances of great importance in computational mathematics. Of particular interest is the integration of functions f which admit representations of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x), \tag{1.2}$$

where the ϕ_i are oscillatory, singular, or both and known *a priori* but the coefficients α_i are not. Then each of the functions $\phi_i(x)$ can exhibit a different type of singularity or a different frequency of oscillation, and classical Gaussian quadrature rules perform very poorly. Many such integrals arise in computational physics (see, for instance, [4] and [18]).

It has long been known that Gaussian quadratures admit generalization to a fairly broad class of systems of functions (see [11, 12, 15, 16, 13]). We adopt the terminology of [14, 20, 3], and say that a generalized Gaussian quadrature for a collection $\{\phi_1, \ldots, \phi_{2n}\}$ of $2n$ functions and a nonnegative weight function ω is an n-point quadrature rule

$$\sum_{j=1}^{n} \phi(x_j)w_j \sim \int_a^b \phi(x)\omega(x)\,dx \tag{1.3}$$

exactly integrating each of the ϕ_i with respect to the weight function ω.

The constructions found in [11, 12, 15, 16, 13] do not lead readily to numerical algorithms for the design of generalized Gaussian quadrature rules. In [14], a numerical algorithm is introduced for the construction of generalized Gaussian quadrature rules for a fairly broad class of functions. The approach is based on the observation that the nodes x_j and weights w_j of a generalized Gaussian quadrature satisfy a nonlinear system of equations. The procedure of [14] is a variant of Newton's algorithm coupled with a continuation scheme for the generation of a suitable initial point for the modified Newton iterations. In [20] and [3], the continuation scheme of [14] is refined, improving

Date: June 30, 2008.

the stability of the algorithm, and a new preprocessing step is added in [20], greatly expanding its range of applicability.

The present paper introduces a new numerical procedure for obtaining generalized Gaussian quadrature rules. While we also use Newton-type iterations to solve nonlinear systems of equations, our scheme differs from the algorithm of [14, 20, 3] in that the continuation method is abandoned in favor of a simpler, more robust approach.

The paper is structured as follows. Section 2 contains mathematical and numerical preliminaries. In section 3, we describe certain standard numerical tools used by the algorithm. Section 4 describes the algorithm in detail. Section 5 contains several examples of quadratures we have obtained. Finally, in Section 6, we present conclusions and discuss several possible extensions of this work.

2. Mathematical and numerical preliminaries

2.1. Generalized Gaussian and Chebyshev quadratures. The quadrature formulas considered in this paper are of the form

$$\sum_{j=1}^{n} \phi(x_j) w_j. \tag{2.1}$$

where the x_j and ω_j are real numbers. We will refer to the x_j as the nodes and the w_j as the weights of the quadrature formula (2.1). They will be used to approximate integrals of the form

$$\int_a^b \phi(x)\omega(x)\,dx \tag{2.2}$$

where $\omega(x)$ is a nonnegative weight function.

Quadratures are typically chosen so as to make (2.1) exact for some set of functions, commonly polynomials of a fixed order. Classical Gaussian quadrature rules consist of n nodes and n weights which integrate polynomials of order $2n-1$ exactly. In [14], the notion of a Gaussian quadrature was generalized as follows:

DEFINITION 2.1. *A quadrature formula will be referred to as Gaussian with respect to a set of $2n$ functions $\phi_1, \dots, \phi_{2n} : [a,b] \to \mathbb{R}$ and a weight function $\omega : [a,b] \to \mathbb{R}^+$ if it consists of n nodes and n weights and integrates exactly the functions ϕ_i with respect to the weight function ω for all $i = 1, \dots, 2n$. The weights and nodes of a Gaussian quadrature will be referred to as Gaussian weights and nodes, respectively.*

Although Chebyshev quadratures are classical Gaussian quadratures on the interval $[-1,1]$ with respect to the weight function $\omega(x) = (1-x^2)^{-1/2}$, in practice, Chebyshev nodes and weights are often used to integrate functions on $[-1,1]$ with respect to the weight function $\omega(x) = 1$. This practice leads to a $2n$-point quadrature which integrates exactly polynomials of order $2n-1$, and motivates the following definition:

DEFINITION 2.2. *A quadrature formula will be referred to as a Generalized Chebyshev quadrature for a set of $2n$ functions $\phi_1, \dots, \phi_{2n} : [a,b] \to \mathbb{R}$ and a weight function $\omega : [a,b] \to \mathbb{R}^+$ if it consists of $2n$ nodes and $2n$ weights and integrates exactly the functions ϕ_i with respect to the weight function ω for all $i = 1, \dots, 2n$. The weights and nodes of a Chebyshev quadrature will be referred to as Chebyshev weights and nodes, respectively.*

2.2. Quadrature rules and optimization. Let m, n be integers with $m \le 2n$ and let $x_1, \dots, x_n$, $w_1, \dots, w_n$ be a quadrature rule

$$\int_a^b f(x)\omega(x)\,dx \sim \sum_{j=1}^{n} f(x_j) w_j \tag{2.3}$$

which is exact for the functions $\phi_1, \dots, \phi_m$. Obviously, the weights w_j and nodes x_j of such a quadrature satisfy the (generally underdetermined) nonlinear system of equations

$$\begin{aligned} F_1(x_1, \dots, x_n, w_1, \dots, w_n) &= b_1 \\ F_2(x_1, \dots, x_n, w_1, \dots, w_n) &= b_2 \\ &\vdots \\ F_m(x_1, \dots, x_n, w_1, \dots, w_n) &= b_m, \end{aligned} \tag{2.4}$$

where

$$F_i(x_1, \dots, x_n, w_1, \dots, w_n) = \sum_{j=1}^{n} \phi_i(x_j) w_j, \tag{2.5}$$

and

$$b_i = \int_a^b \phi_i(x)\omega(x)\, dx. \tag{2.6}$$

When $m = 2n$, the nodes x_j and weights w_j in (2.4) form a generalized Gaussian quadrature for the functions $\phi_1, \dots, \phi_{2n}$. In [11] and [13], the existence of a unique solution of (2.4) is proven under certain conditions on the system of functions $\phi_1, \dots, \phi_{2n}$. It is observed in [14, 20, 3] that solutions, or at least approximate solutions, of (2.4) exist for many systems of $2n$ functions not satisfying those conditions.

In this paper, we will encounter systems of the form (2.4) where $m < 2n$. Under these conditions, the system does not admit a unique solution. Non-uniqueness is not, however, an obstruction to the determination of a quadrature rule for the functions $\phi_1, \dots, \phi_m$ since such a rule is given by *any* solution of the system (2.4). Indeed, even in the case when there is no exact solution of (2.4), it is often possible to construct an approximate quadrature for the functions $\phi_1, \dots, \phi_m$.

We close this section with the following definition:

DEFINITION 2.3. *Suppose $\phi_1, \dots, \phi_m : [a, b] \to \mathbb{R}$ are square integrable with respect to the nonnegative integrable weight function ω. We say that a quadrature rule $x_1, \dots, x_n, w_1, \dots, w_n$ integrates $\phi_1, \dots, \phi_m$ with precision $\epsilon > 0$ if*

$$\sum_{i=1}^{m} |F_i(x_1, \dots, x_n, w_1, \dots, w_n) - b_i|^2 \le \epsilon^2, \tag{2.7}$$

where

$$F_i(x_1, \dots, x_n, w_1, \dots, w_n) = \sum_{j=1}^{n} \phi_i(x_j) w_j \tag{2.8}$$

and

$$b_i = \int_a^b \phi_j(x)\omega(x)\, dx \tag{2.9}$$

for all $i = 1, \dots, m$.

2.3. **Quadrature and interpolation.** It is well known that when Chebyschev nodes are used for polynomial interpolation on the interval $[-1, 1]$, the procedure is numerically stable and the convergence properties are close to optimal (see [19] and [6]). In this subsection, we prove that the nodes of *any* Gaussian quadrature and many generalized Gaussian quadratures lead to stable interpolation formulas.

The principal analytic tool of this subsection is the following obvious theorem (see, for example, [20]).

THEOREM 2.1. *Suppose that the weight function $\omega : [a, b] \to \mathbb{R}$ is nonnegative and the functions $\phi_1, \dots, \phi_n : [a, b] \to \mathbb{R}$ are orthonormal with respect to ω. Suppose further that the n-point quadrature* *rule* $x_1, \dots, x_n, w_1, \dots, w_n$ *is exact for all products of the form* $\phi_i(x)\phi_j(x)$ *and* $w_i > 0$ *for all* $1 \le i \le n$. *Then the* $n \times n$ *matrix* A *with entries*

$$A_{ij} = \sqrt{w_j}\phi_i(x_j) \tag{2.10}$$

is orthogonal.

Now let f be a function of the form

$$f(x) = \sum_{j=1}^{n} \alpha_i \phi_i(x). \tag{2.11}$$

We would like to construct an interpolation formula on the interval [a,b] for functions of this form; that is, given the values $f_1, \dots, f_n$ of a function of the form (2.11) at a collection of points $x_1, \dots, x_n$, we would like a formula for determining the coefficients $\alpha_1, \dots, \alpha_n$. Let α be the vector $\alpha = (\alpha_1, \dots, \alpha_n)$, let F be the vector $F = (f_1, f_2, \dots, f_n)$, and finally, let B be the $n \times n$ matrix with entries

$$B_{ij} = \phi_i(x_j). \tag{2.12}$$

Then

$$F = B\alpha, \tag{2.13}$$

and assuming that B is invertible, it follows that

$$\alpha = B^{-1}F. \tag{2.14}$$

If the condition number of B is not too high, then (2.14) is a numerically stable formula for computing the coefficients $\alpha_1, \dots, \alpha_n$. The following observation is the principal observation of this subsection.

Observation 2.1. *Under the conditions of Theorem 2.1, the matrix* B *in the interpolation formula (2.14) is of the form*

$$B = DQ, \tag{2.15}$$

where D *is a diagonal matrix with entries*

$$D_{ii} = \frac{1}{\sqrt{w_i}} \tag{2.16}$$

and Q *is orthogonal. Thus*

$$\alpha = Q^* D^{-1} F. \tag{2.17}$$

In other words, the coefficients α *can be obtained by applying a diagonal matrix followed by an orthogonal matrix.*

2.4. **The damped Gauss-Newton method.**

The damped Gauss-Newton method is a well-known iterative technique for the solution of nonlinear least-squares problems. It converges under very general conditions, and does not require that the Jacobian of the system be nonsingular. Here we give only elementary details; a more thorough treatment can be found in [7].

Suppose that $R : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable function of the form

$$R(x) = \begin{pmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_m(x) \end{pmatrix} \tag{2.18}$$

and let $J(x)$ be the Jacobian

$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1}(x) & \dots & \frac{\partial r_1}{\partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial r_m}{\partial x_1}(x) & \dots & \frac{\partial r_m}{\partial x_n}(x) \end{pmatrix} \tag{2.19}$$

of R at the point x. The damped Gauss-Newton method is a numerical method for minimizing the function

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} |r_j(x)|^2. \tag{2.20}$$

It belongs to a broad class of numerical optimization methods which proceed from an initial guess x_0 by forming a sequence $x_1, x_2, \ldots$ of iterates via the formula

$$x_{k+1} = x_k + \alpha_k d_k \tag{2.21}$$

where d_k is referred to as the search direction at iteration k and α_k is a carefully chosen step size.

The primary purpose of this section is Theorem 2.2 below, which gives conditions under which an iteration of the form (2.21) converges. We start with the following definition:

DEFINITION 2.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and consider any iteration of the form (2.21). We say that the step length α_k and step direction d_k satisfy the Wolfe conditions at the point x_k if*

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \lambda\alpha_k \nabla f(x_k) \cdot d_k, \tag{2.22}$$

and

$$\nabla f(x_k + \alpha_k d_k) \cdot d_k \geq \beta \nabla f(x_k) \cdot d_k \tag{2.23}$$

for some constants $\lambda \in (0,1)$ and $\beta \in (\lambda, 1)$.

The following theorem can be found in [7]:

THEOREM 2.2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function bounded from below, and assume that there exists $\gamma \geq 0$ such that*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \gamma \|x - y\|_2 \tag{2.24}$$

for every x and y in $\mathbb{R}^n$. Consider any iteration of the form (2.21) such that for each $k = 0, 1, \ldots$ the pair (d_k, α_k) satisfies the Wolfe conditions. If, in addition, one of the conditions

$$\nabla f(x_k) \cdot d_k < 0 \tag{2.25}$$

or

$$\nabla f(x_k) = 0 \quad \textit{and} \quad d_k = 0 \tag{2.26}$$

holds for each $k = 0, 1, \ldots,$ then either

$$\nabla f(x_k) = 0 \textit{ for some } k \geq 0, \tag{2.27}$$

or

$$\lim_{k\to\infty} \frac{\nabla f(x_k) \cdot d_k}{\|d_k\|_2} = 0. \tag{2.28}$$

Remark 2.1. *Theorem 2.2 states that either the sequence x_k converges to a critical point for the function f or the direction d_k become orthogonal to the gradient of f. In practice, it is easy to avoid the later condition, thus ensuring the convergence of $\nabla f(x_k)$ to 0.*

Remark 2.2. *Under mild conditions on the function f, given a sequence of $\{d_k\}$ satisfying condition (2.25), there exists a sequence of α_k satisfying the Wolfe conditions (see, for example,* [7] *Theorem 6.3.2).*

In the case of the damped Gauss-Newton method, the search direction d_k is chosen as a solution to the least-squares problem

$$\min_{d_k} \; ||J(x_k)d_k + R(x_k)||_2, \tag{2.29}$$

which is an affine approximation to the nonlinear least-squares problem

$$\min_{x} \; ||R(x)||_2.$$

Since

$$\nabla f(x_k) = J^*(x_k)R(x_k), \tag{2.30}$$

we have

$$\langle \nabla f(x_k), d_k \rangle = \langle R(x_k), J(x_k) d_k \rangle \,. \tag{2.31}$$

The search direction d_k is chosen so that $J(x_k)d_k$ is the projection of $-R(x_k)$ onto the column space of $J(x_k)$. It follows that

$$\langle \nabla f(x_k), d_k \rangle = \langle R(x_k), J(x_k) d_k \rangle \leq 0. \tag{2.32}$$

If we choose d_k to be 0 in the event that $\langle \nabla f(x_k), d_k \rangle = 0$, then we obtain the following theorem:

THEOREM 2.3. *Suppose that $R : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable function of the form (2.18), $f : \mathbb{R}^n \to \mathbb{R}$ is given by (2.20), and the Jacobian, $J(x)$, of R is given by (2.19). Further suppose that $\|J(x)\|_2$ is bounded on $\mathbb{R}^n$ and there exists a constant $\gamma > 0$ such that*

$$\|J(x) - J(y)\|_2 \leq \gamma \|x - y\|_2$$

for all x and y in $\mathbb{R}^n$. If x_k is defined by the iteration

$$x_{k+1} = x_k + \alpha_k d_k$$

where, for each $k = 1, 2, \ldots$, d_k is a solution of the least-squares problem

$$||J(x_k)d_k + R(x_k)||_2 = \min_v \; ||J(x_k)v + R(x_k)||_2,$$

and the sequence $\{\alpha_k\}$ is chosen so that the pairs (d_k, α_k) satisfy the Wolfe conditions for $k = 0, 1, \ldots$, then either

$$\nabla f(x_k) = 0 \text{ for some } k \geq 0$$

or

$$\lim_{k\to\infty} \frac{\nabla f(x_k) \cdot d_k}{\|d_k\|_2} = 0.$$

2.5. **Singular value decomposition.** The SVD is a ubiquitous tool in numerical analysis (see, for instance, [8]). Here we discuss it in the case of real matrices.

LEMMA 2.1. *(SVD). For any $n \times m$ real matrix A, there exist, for some integer k, an $n \times k$ real matrix U with orthonormal columns, an $m \times k$ real matrix V with orthonormal columns, and a $k \times k$ real diagonal matrix Σ with positive diagonal entries $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k > 0$, such that*

$$A = U \cdot \Sigma \cdot V^*. \tag{2.33}$$

The diagonal entries of Σ are called singular values, the columns of the matrix U are called the left singular vectors, and the columns of the matrix V are called right singular vectors.

One of the most common applications of the SVD is the approximation of matrices; if we let Σ_p denote the diagonal matrix with entries

$$D_{ii} = \begin{cases} \sigma_i & \text{if } i \leq p \\ 0 & \text{otherwise} \end{cases},$$

then

$$\|A - U\Sigma_p V^*\|_2 = \sigma_{p+1}. \tag{2.34}$$

The matrix $U\Sigma_p V^*$ is, in fact, the optimal rank p approximation of the matrix A (see, for instance, [8]); that is,

$$\min_{A_k} \|A - A_k\|_2 = \sigma_{p+1} \tag{2.35}$$

where A_k ranges over the set of all $n \times m$ matrices of rank k.

2.6. **QR decompositions.** The singular value decomposition provides the means to construct an optimal rank k approximation to a given matrix; however, the SVD can be expensive to form, and other less computationally expensive matrix factorizations can be used to compress matrices in lieu of the SVD.

By a slight abuse of terminology, we will refer to the factorization in the following obvious lemma as a "QR decomposition."

LEMMA 2.2. *For any $n \times m$ real matrix A, there exist an integer k, an $n \times k$ real matrix Q with orthonormal columns, an $m \times m$ permutation matrix Π, and a $k \times m$ real matrix R with the block form*

$$R = \begin{pmatrix} T & M \end{pmatrix},$$

where T is a $k \times k$ upper triangular matrix with positive diagonal entries and M is a general $k \times (m-k)$ matrix, such that

$$A\Pi = QR. \tag{2.36}$$

Moreover, there is a one-to-one correspondence between $m \times m$ permutation matrices Π and decompositions of the form (2.36).

The following theorem can be found (in a stronger form) in [10].

THEOREM 2.4. *Suppose that A is a real $m \times n$ matrix with singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. For any integer p, $0 < p \leq r$, there exists an $m \times m$ permutation matrix Π such that the QR decomposition uniquely determined by Π is of the form*

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{2.37}$$

where R_{11} is a $p \times p$ upper triangular matrix with positive diagonal entries, and

$$\|R_{22}\|_2 \leq \sqrt{1 + (n-k)}\sigma_{p+1}. \tag{2.38}$$

Remark 2.3. *Theorem 2.4 implies that given any real $m \times n$ matrix A with singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ and any integer $0 < k < r$, there is a permutation matrix Π such that the well-known Gram-Schmidt orthogonalization procedure (see, for example, [8]) applied to the matrix $A\Pi$ results in the approximate matrix factorization*

$$\|A - QS\|_2 \leq c\sigma_{k+1}, \tag{2.39}$$

where Q is an $m \times k$ matrix with orthonormal columns and S is a $k \times n$ matrix S. Note that the error bound (2.39) is close to the optimal error for rank k approximations to A.

Remark 2.4. *When properly implemented, the modified Gram-Schmidt algorithm with reorthogonalization is a reliable method for obtaining a QR decomposition (see, for instance, [2]). While there are no guaranteed bounds of the form (2.39) for this algorithm, it does well in practice. In [10], robust, provably stable algorithms are introduced for producing QR decompositions that satisfy bounds of the form (2.39).*

2.7. **Analogs of matrix factorizations for finite sequences of functions.** Here we introduce analogs of the matrix factorizations of the preceding two subsections for finite sequences of functions. We begin with the following obvious generalization of Lemma 2.2.

LEMMA 2.3. *For any finite sequence $\phi_1, \ldots, \phi_m : [a,b] \to \mathbb{R}$ of square integrable functions, there exists an integer $k \leq m$, a permutation $\Pi \in S^m$, a collection of orthonormal functions $q_1, \ldots, q_k : [a,b] \to \mathbb{R}$, and an $k \times m$ real matrix $R = (r_{ij})$ with the block form*

$$\begin{pmatrix} T & M \end{pmatrix}$$

where T is a $k \times k$ upper triangular matrix with positive diagonal entries and M is a general $k \times (m-k)$ matrix, such that

$$\phi_{\Pi(j)}(x) = \sum_{i=1}^{\max(j,k)} r_{ij} q_i(x) \tag{2.40}$$

for all $j = 1, \ldots, m$. Moreover, there is a one-to-one correspondence between decompositions of this form and permutations $\Pi \in S^m$.

By analogy with the finite dimensional case, we will refer to decompositions of this type as QR decompositions. The functions $q_i(x)$ will be referred to as QR functions and we will call the diagonal entries r_{ii} of R the normalizing factors of the decomposition.

As in the case of matrices, a common application of expansions of the form (2.40) is the "compression" of the functions ϕ_j; that is, often permutations $\Pi \in S^m$ can be found for which the truncated series

$$\sum_{j=1}^{p} a_{ij} q_i(x), \tag{2.41}$$

with $p < k$, provide good approximations to the functions $\phi_{\Pi(j)}$.

Now, we restate a result found in [3], generalizing the SVD to the case of a finite sequence of functions.

THEOREM 2.5. *Suppose that the functions $\phi_1, \ldots, \phi_m : [a, b] \to \mathbb{R}$ are square integrable. Then there exist an integer k, a finite orthonormal sequence of functions $u_1, \ldots, u_k : [a, b] \to \mathbb{R}$, an $m \times k$ matrix $V = (v_{ij})$ with orthonormal columns, and a sequence $s_1 \geq s_2 \geq \cdots \geq s_k > 0 \in \mathbb{R}$ such that*

$$\phi_j(x) = \sum_{i=1}^{k} u_i(x) s_i v_{ji} \tag{2.42}$$

for all $x \in [a, b]$ and all $j = 1, \ldots, m$. The sequence $s_1, \ldots, s_k$ is uniquely determined by k.

By analogy with the finite-dimensional case, we will refer to this decomposition as the SVD of a finite sequence of functions. We call the functions u_i the singular functions, the columns of V the singular vectors, and the values s_i the singular values. The SVD is clearly a useful tool for the compression of the sequence $\phi_1, \ldots, \phi_m$: if we let $\tilde{\phi_j}(x)$ denote the p-term truncation

$$\tilde{\phi_j}(x) = \sum_{i=1}^{p} u_i(x) s_i v_{ji} \tag{2.43}$$

of the sum (2.42), then

$$\|\tilde{\phi_j}(x) - \phi_j(x)\| \leq s_{p+1} \tag{2.44}$$

for $j = 1, \ldots, m$.

However it is obtained, using an approximate representation

$$\phi_j(x) \sim \sum_{i=1}^{p} \alpha_i q_i(x), \tag{2.45}$$

integrals of the form

$$\int_a^b \phi_j(x) \omega(x)\, dx \tag{2.46}$$

can be approximated as

$$\begin{aligned} \int_a^b \phi_j(x)\omega(x)\, dx &\approx \int_a^b \left(\sum_{i=1}^{p} \alpha_i q_i(x) \right) \omega(x)\, dx \\ &= \sum_{i=1}^{p} \alpha_i \int_a^b q_i(x) \omega(x)\, dx; \end{aligned} \tag{2.47}$$

thus, a quadrature which is exact for the integrals

$$\int_a^b q_i(x) \omega(x)\, dx \tag{2.48}$$

for $i = 1, \ldots, p$, can be used as an approximate quadrature for the integrals

$$\int_a^b \phi_j(x)\omega(x)\,dx \tag{2.49}$$

for $j = 1, \ldots, m$.

2.8. **Underdetermined systems.** The purpose of this short subsection is the statement of two lemmas on the existence and behavior of sparse solutions of underdetermined systems.

We begin with the following result on the existence of well-behaved solutions to underdetermined least-squares problems, whose proof is an elementary exercise in linear algebra.

LEMMA 2.4. *Suppose that A is an $n \times m$ matrix, $n < m$, with left singular vectors $u_1, \ldots, u_k$ and singular values $\sigma_1 \geq \ldots \geq \sigma_k > 0$. For $0 < p \leq k$, let U_p be the subspace of $\mathbb{R}^n$ spanned by $u_1, \ldots, u_p$, and let $Proj_p : \mathbb{R}^n \to U_p$ denote the projection operator $\mathbb{R}^n \to U_p$. Then, given any vector $b \in \mathbb{R}^n$ and any integer $0 < p \leq k$, there exists a vector $x \in \mathbb{R}^m$ with no more than p nonzero entries such that*

$$\|Ax - b\|_2 = \|Proj_p b - b\|_2 \tag{2.50}$$

and

$$\|x\|_2 \leq \frac{\sigma_1}{\sigma_p}\|b\|_2. \tag{2.51}$$

Let A be an $n \times m$ matrix with $n < m$ and consider the least-squares problem

$$\min_x \|Ax - b\|_2, \tag{2.52}$$

where b is a given vector in $\mathbb{R}^n$. The problem is underdetermined and therefore the condition number of A is infinite. However, Lemma 2.4 implies that if b is in the span of a small number of singular vectors of A, then a well-behaved approximate solution x to the least-squares problem (2.52) can still be found.

The second lemma is a stronger, but more specialized result, whose proof is an easy corollary of the following theorem, which appears as Theorem 2 in [17].

THEOREM 2.6. *Suppose that S is an arbitrary set, n is a positive integer, $f_1, \ldots, f_n$ are bounded complex-valued functions on S, and ϵ is a positive real number such that*

$$\epsilon \leq 1. \tag{2.53}$$

Then, there exist n points $x_1, \ldots, x_n$ in S and n functions $g_1, \ldots, g_n$ on S such that

$$|g_k(x)| \leq 1 + \epsilon \tag{2.54}$$

for all x in S and $k = 1, 2, \ldots, n$, and

$$f(x) = \sum_{k=1}^{n} f(x_k) g_k(x) \tag{2.55}$$

for all x in S and any function f defined on S via the formula

$$f(x) = \sum_{k=1}^{n} c_k f_k(x). \tag{2.56}$$

Moreover, if the set S is finite, then $g_1, \ldots, g_n$ can be chosen so that (2.54) holds with $\epsilon = 0$.

LEMMA 2.5. *If*

$$Ax = b, \tag{2.57}$$

where A is an $m \times n$ matrix of rank m, then there exists a vector $\tilde{x} \in \mathbb{R}^n$ with no more then m nonzero entries such that

$$A\tilde{x} = b,$$

and

$$\|\tilde{x}\|_1 \leq m\|x\|_1.$$

Proof: By Theorem 2.6, there exists an $m \times n$ matrix G whose entries are bounded by 1 and an $m \times m$ matrix of $\tilde{A}$ consisting of m columns $A_{i_1}, \ldots, A_{i_m}$ of A such that

$$A = \tilde{A}G. \tag{2.58}$$

Since $Ax = b$, it follows that

$$\tilde{A}Gx = b. \tag{2.59}$$

If we let $y = Gx \in \mathbb{R}^m$ and define $\tilde{x}$ by the formula

$$\tilde{x_j} = \begin{cases} y_k & \text{if } j = i_k, \\ 0 & \text{otherwise} \end{cases}, \tag{2.60}$$

then $\tilde{x}$ has at most m nonzero entries, $A\tilde{x} = b$, and

$$\|\tilde{x}\|_1 = \sum_{i=1}^{m} |\tilde{x_i}| \tag{2.61}$$

$$= \sum_{i=1}^{m} \left| \sum_{j=1}^{n} g_{ij} x_j \right| \tag{2.62}$$

$$\leq \sum_{i=1}^{m} \sum_{j=1}^{n} |x_j| \tag{2.63}$$

$$\leq m \|x\|_1, \tag{2.64}$$

as desired.

Remark 2.5. *For sets S which are finite, the proof of Theorem 2.6 given in* [17] *is constructive, but the procedure is computationally infeasible. To wit, in the special case of Lemma 2.5, the scheme of* [17] *amounts to the choice of a submatrix $\tilde{A}$ consisting of a set of columns $A_{i_1}, \ldots, A_{i_m}$ of A which maximize the determinant*

$$\det \begin{pmatrix} A_{i_1} & A_{i_2} & \ldots & A_{i_m} \end{pmatrix} \tag{2.65}$$

over the collection of all sets of m columns of the matrix A. The existence of the matrix G follows since

$$\det(\tilde{A}) \neq 0$$

by construction, and the bound on the entries of G follows from Cramer's rule.

Remark 2.6. *In practice, the modified Gram-Schmidt procedure with reorthogonalization can be used to find a set of m columns $A_{i_1}, \ldots, A_{i_m}$ of the $m \times n$ rank m matrix A such that*

$$\det \begin{pmatrix} A_{i_1} & A_{i_2} & \ldots & A_{i_m} \end{pmatrix} \tag{2.66}$$

is comparable to the supremum

$$\sup_{j_1, \ldots, j_m} \det \begin{pmatrix} A_{j_1} & A_{j_2} & \ldots & A_{j_m} \end{pmatrix} \tag{2.67}$$

over all collections of m columns of A. Indeed, the modified Gram-Schmidt procedure is nothing more than a "greedy" algorithm for finding an approximate solution to the optimization problem

$$\operatorname*{argmax}_{j_1, \ldots, j_m} \det \begin{pmatrix} A_{j_1} & A_{j_2} & \ldots & A_{j_m} \end{pmatrix}. \tag{2.68}$$

An obvious modification of the argument in [17] *(which is sketched above in Remark 2.5) shows that once such a set of columns $i_1, \ldots, i_m$ has been found, the matrix A can factored as*

$$A = \tilde{A}G,$$

where $\tilde{A}$ is an $m \times m$ submatrix of A and G is an $m \times n$ matrix whose entries are bounded in absolute value, but not necessarily by 1.

Remark 2.7. *Strong rank revealing QR factorizations (see* [10] *and* [2], *for instance) identify a spanning set* $A_{i_1}, \dots, A_{i_m}$ *of columns of an* $m \times n$ *matrix* A *of rank* m *for which the ratio of*

$$\det \begin{pmatrix} A_{i_1} & A_{i_2} & \dots & A_{i_m} \end{pmatrix}$$

to

$$\sup_{j_1,\dots,j_m} \det \begin{pmatrix} A_{j_1} & A_{j_2} & \dots & A_{j_m} \end{pmatrix}$$

is guaranteed to satisfy a lower bound, thus ensuring that a stable factorization of the form

$$A = \tilde{A}G,$$

where $\tilde{A}$ *is an* $m \times m$ *submatrix of* A*, can be found.*

2.9. The Sherman-Morrison-Woodbury formula.

The Sherman-Morrison-Woodbury formula gives an expression for the rank-k update

$$(A + UV^t)^{-1}$$

of the inverse of a matrix A. The following Lemma can be found, for instance, in [8]:

LEMMA 2.6. *Suppose that* A *is an invertible* $n \times n$ *matrix,* U *is an* $n \times k$ *matrix, and* V *is an* $k \times n$ *matrix. If the rank-k update*

$$(A + UV^t) \tag{2.69}$$

of the matrix A *is invertible, then its inverse is*

$$(A + UV^t)^{-1} = A^{-1} - A^{-1}U(I + V^tA^{-1}U)^{-1}V^tA^{-1}.$$

In this paper, given an $m \times n$ real matrix A, we will utilize the Sherman-Morrison-Woodbury formula to perform a specific type of update to the inverse of the $m \times m$ matrix AA^t. In particular, we wish to update the inverse of AA^t in order to form the inverse of the matrix BB^t, where B is obtained from A by deleting its j^{th} column. That is,

$$B = A - uv^t,$$

where u is the $m \times 1$ vector which is the j^{th} column of the matrix A and v is the $n \times 1$ vector with entries

$$v_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.70}$$

A simple calculation shows that

$$\left(A - uv^t\right)\left(A - uv^t\right)^t = AA^t - uv^tA^t - Au^tv + uv^tvu^t \tag{2.71}$$

$$= AA^t - uu^t. \tag{2.72}$$

In other words, if we form the matrix B by deleting the j^{th} column of the matrix A, then BB^t can be formed from AA^t by a rank-1 update. The Sherman-Morrison-Woodbury formula then implies that the inverse of BB^t can be computed from $(AA^t)^{-1}$ via a rank-1 update.

3. Numerical apparatus

3.1. Nested Legendre discretizations of sequences of functions.

In this paper, we are confronted with sequences $\phi_1, \dots, \phi_m$ of functions defined on the interval $[a, b]$ for which we wish to construct a generalized Gaussian quadrature rule. The collection of functions ϕ_j has the following properties:

- Each function ϕ_j is integrable on $[a, b]$ and analytic except at a small number of points,
- The total number functions is quite large (e.g., $m = 10,000$),
- The rank of the set $\phi_1, \dots, \phi_m$ is low (e.g., 40) to high precision.

In order to construct an efficient quadrature rule, we will take advantage of rank deficiency to "compress" the sequence $\phi_1, \dots, \phi_m$. This means, more specifically, that we first form a set $u_1, \dots, u_k$ of orthonormal functions on $[a, b]$ such that each ϕ_i can be approximated by a sum of the form

$$\phi_i(x) \sim \sum_{j=1}^{k} \alpha_{ij} u_j(x), \tag{3.1}$$

and each u_j is defined by a sum

$$u_j(x) = \sum_{i=1}^{m} \beta_{ij} \phi_i(x). \tag{3.2}$$

We then build a generalized Gaussian quadrature rule for the functions $u_1, \dots, u_k$. In the course of constructing that quadrature, it will be necessary to evaluate the functions u_j at arbitrary points on the interval $[a, b]$. If the sums (3.2) involve a large number of terms, or if the evaluation of the ϕ_i is expensive, then it is impractical to use formula (3.2) to evaluate the u_j. It will therefore be necessary to represent the function u_j in a manner which allows for their rapid evaluation.

An obvious alternative to evaluating sums of the form (3.2) directly is to represent the functions u_j via polynomial interpolation. Let $x_1, \dots, x_n$ be a mesh of interpolation points on the interval $[a, b]$ and suppose that the Lagrange polynomials interpolating $u_1, \dots, u_k$ at these mesh points represent the functions u_j to a given precision. Then, clearly, the Lagrange polynomial interpolating the function

$$\psi_i = \sum_{j=1}^{n} \alpha_{ij} u_j(x) \tag{3.3}$$

at $x_1, \dots, x_n$ approximates ϕ_i with controllable precision. As was discussed in Subsection 2.3, if Gaussian nodes are used as interpolating points, then the resulting procedure is numerically stable. However, when the functions ϕ_j are not smooth, polynomial interpolation becomes inefficient and can fail completely for sufficiently singular functions.

In such cases, where the functions to be interpolated are singular, it is customary to use an adaptive interpolation scheme instead. That is, the interval $[a, b]$ is subdivided into a collection of subintervals such that each function ϕ_j is accurately interpolated by a low order polynomial on each subinterval.

In this subsection, we describe a numerical procedure for the approximation of a sequence of functions via nested Gauss-Legendre polynomial interpolation. The procedure is introduced in [20], but we reproduce it here since it is an integral part of the algorithm. The input to this procedure is a collection $\phi_1, \dots, \phi_m$ of functions integrable on $[a, b]$, a precision $\epsilon > 0$, and a reasonably large integer k which controls the number of interpolation nodes used on each subinterval (for the computations in this paper, we used $k = 30$). The algorithm proceeds in three stages.

Stage 1. In the first stage, the following procedure is used to discretize each of the ϕ_i separately. That is, the following sequence of steps is repeated for $i = 1, \dots, m$:

1. Construct the $2k$ Legendre nodes $x_1, \dots, x_{2k}$ on the interval $[a, b]$.

2. Let $P(x)$ denote the Lagrange polynomial of order $2k - 1$ interpolating ϕ_i at the mesh points $x_1, \dots, x_{2k}$. Construct the coefficients $\alpha_1, \dots, \alpha_{2k}$ of $P(x)$ in the expansion

$$P(x) = \sum_{j=0}^{2k-1} \alpha_j L_j(x)$$

where $L_j(x)$ is the j^{th} order Legendre polynomial.

3. If the inequality

$$\sum_{j=k+1}^{2k} |\alpha_j|^2 < \epsilon \tag{3.4}$$

is satisfied, then we conclude that the order k Legendre expansion for ϕ_i on the interval $[a, b]$ is sufficient. Otherwise, we split the interval $[a, b]$ into two subintervals, $[a, (a+b)/2]$ and $[(a+b)/2, b)]$, and repeat the procedure recursively for each of the subintervals.

Stage 2. Store the endpoints of each subinterval constructed in Stage 1 in an array. Sort the array and eliminate multiple elements. The resulting array of points on the interval $[a, b]$ is the array of endpoints of the subintervals of the final subdivision.

Stage 3. Construct the k point Legendre discretization on each the subintervals obtained in Stage 2 for each of the functions $\phi_i(x)$.

Remark 3.1. *The scheme of this subsection is a reasonably reliable mechanism for the discretization of sets of functions with singularities. The stopping condition (3.4) is analogous to that usually used to terminate an adaptive quadrature procedure. Just as any such quadrature procedure can fail for carefully designed counterexamples, so too can the procedure of this section fail under certain circumstances. The problem, however, is not encountered in the examples of this paper and whenever the authors have encountered it in practice, it has been easy to rectify.*

3.2. **Compression of a finite sequence of functions.**

This subsection describes a numerical procedure for compressing a finite sequence of functions by approximating either the singular value decomposition or a QR decomposition of the sequence.

We begin with two definitions, the second of which is adapted from [20] and [3]. In what follows, $PC([a, b])$ refers to the vector space of piecewise continuous functions on the interval $[a, b]$.

DEFINITION 3.1. *A k-point linear interpolation scheme on the interval $[a, b]$ is a collection of linearly independent functions $\phi_1, \dots \phi_k$ in $PC([a, b])$ and a set of distinct nodes $x_1, \dots, x_k$ in $[a, b]$ together with a linear mapping $T : PC([a, b]) \to span\{\phi_1, \dots, \phi_k\}$ such that*

$$Tf(x_j) = f(x_j) \tag{3.5}$$

for all $j = 1, \dots, k$.

We call the $x_1, \dots, x_k$ interpolation nodes, the mapping T the interpolation mapping, and the ϕ_j interpolating functions. We also say that the coefficients $\alpha_1, \dots, \alpha_k$ of $Tf(x)$ with respect to the basis $\{\phi_1, \dots, \phi_k\}$ are the interpolation coefficients for the function f.

Associated with every k-point linear interpolation scheme is an invertible linear transformation $\mathbb{R}^k \to \mathbb{R}^k$ which takes the values $f(x_1), \dots, f(x_k)$ of a function f at the interpolation nodes to the k interpolation coefficients for the function. The image Tf of f under the interpolation mapping is, of course, determined by these interpolation coefficients. We will often refer to Tf as the function defined by the interpolation scheme and the values $f(x_1), \dots, f(x_k)$.

Remark 3.2. *It is generally possible to extend the definition of interpolation scheme to encompass interpolating functions ϕ not in $PC([a, b])$, as well as interpolation mappings T defined on larger spaces. We must keep in mind, however, that equation (3.5) requires that both $Tf(x)$ and $f(x)$ be defined pointwise.*

DEFINITION 3.2. *We say that the the combination of a k-point linear interpolation scheme on $[a, b]$ with nodes $y_1, \dots, y_k \in [a, b]$ and interpolating functions $\phi_1, \dots, \phi_k$, and a k-point quadrature rule with nodes $x_1, \dots, x_k \in [a, b]$ and weights $w_1, \dots, w_k$ preserves inner products if*

- *The nodes $x_1, \dots, x_k$ of the quadrature rule coincide with the nodes $y_1, \dots, y_k$ of the interpolation scheme, and*
- *The quadrature rule is exact for all products $\phi_i(x)\phi_j(x)$ of the interpolating functions.*

Remark 3.3. *The second condition in definition 3.2 together with the assumption of the linearity of the interpolation scheme ensures that for any two piecewise continuous functions $f(x)$ and $g(x)$ the quadrature rule is exact for the integral*

$$\int_a^b Tf(x)Tg(x)\, dx. \tag{3.6}$$

The following are examples of quadrature and interpolation schemes which preserve inner products:

Example 3.1. *The combination of a classical Gaussian quadrature and Lagrange interpolation at the same Gaussian nodes preserves inner products, since polynomials interpolation on n nodes produces an interpolating polynomial of order $n-1$ and the product of any two such polynomials is exactly integrated by an n point Gaussian quadrature.*

Example 3.2. *Nested Gaussian interpolation — like that of the preceding section — coupled with corresponding nested Gaussian quadrature formulas also preserve inner products, since on each subinterval Gaussian interpolation is coupled with a Gaussian quadrature formula.*

Note that the interpolating functions in this example are not continuous, but rather piecewise continuous.

The following theorem, which is a reformulation of Theorem 4.5 in [20], will be used in approximating the singular value decomposition (or a QR decomposition) of a sequence of functions.

THEOREM 3.1. *Suppose that the combination of a quadrature rule with nodes $x_1, \ldots, x_n$ and weights $w_1, \ldots, w_n$, and a linear interpolation scheme with interpolating functions $\psi_1, \ldots, \psi_n$ and interpolation mapping T preserves inner products on $[a, b]$. Suppose further that $\phi_1, \ldots, \phi_m$ is a collection of piecewise continuous functions on $[a, b]$, $U = (u_{ij})$ is an $n \times k$ matrix with orthonormal columns, and $R = (r_{ij})$ is a $k \times m$ matrix such that*

$$\phi_j(x_i)\sqrt{w_i} = \sum_{l=1}^{k} u_{il} r_{lj} \tag{3.7}$$

for all $j = 1, \ldots, m$ and $i = 1, \ldots, n$. If the functions $u_j(x)$ are defined for all $j = 1, \ldots, k$ via the interpolation scheme and the values

$$u_j(x_i) = \frac{u_{ij}}{\sqrt{w_i}}, \tag{3.8}$$

then:

1. *The functions $u_j(x)$ are orthonormal; that is,*
$$\int_a^b u_i(x) u_j(x)\, dx = \delta_{ij}$$
for all $i, j = 1, \ldots, k$.
2. *The sequence of functions $\tilde{\phi}_1, \ldots, \tilde{\phi}_m$ defined by the formula*
$$\tilde{\phi}_j(x) = \sum_{i=1}^{k} u_i(x) r_{ij}$$
is identical to the sequence of functions produced by sampling the functions $\phi_1, \ldots, \phi_m$ at the points $\{x_i\}$ and then interpolating with the interpolation scheme on $[a, b]$.

The algorithm described below uses as input a sequence of functions $\phi_1, \ldots, \phi_m : [a, b] \to \mathbb{R}$ and a quadrature and interpolation scheme which preserves inner products. The nodes and weights of the quadrature will be denoted by $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$, respectively.

The output of the algorithm depends on whether the SVD or a QR decomposition is used for compression. In either case, the output is a sequence $u_1, \ldots, u_k$ of orthonormal vectors and a sequence $\sigma_1, \ldots, \sigma_k$ of positive real numbers. When the SVD is used, the orthonormal vectors $u_1, \ldots, u_k$ approximate the singular vectors for the sequence $\phi_1, \ldots, \phi_m$ and the $\sigma_1, \ldots, \sigma_k$ approximate the corresponding singular values. In the case of the QR decomposition, the orthonormal vectors $u_1, \ldots, u_k$ approximate a collection of QR vectors for $\phi_1, \ldots, \phi_m$ and the σ_j are the corresponding normalizing factors.

Description of the algorithm.

1. Construct the $n \times m$ matrix A with entries
$$A_{ij} = \phi_j(x_i)\sqrt{w_i}.$$

2. Compute either the SVD of the matrix A or a QR decomposition for the matrix A. In the first case, produce the factorization
$$A = U \Sigma V^*,$$

where $U = (u_{ij})$ is an $n \times k$ matrix with orthonormal columns, $V = (v_{ij})$ is a n $m \times k$ matrix with orthonormal columns, and Σ is a diagonal matrix whose j^{th} diagonal entry is σ_j. In the second case, produce the factorization

$$A\Pi = UR,$$

where $U = (u_{ij})$ is an $n \times k$ matrix with orthonormal columns and R is an $k \times m$ trapezoidal matrix with diagonal entries $\sigma_1, \ldots, \sigma_k$.

3. Construct the $n \times k$ values $u_j(x_i)$ defined by the formula

$$u_j(x_i) = \frac{u_{ij}}{\sqrt{w_i}}. \tag{3.9}$$

4. For any desired point $x \in [a, b]$, evaluate the functions $u_i : [a, b] \to \mathbb{R}$ using the interpolation scheme on $[a, b]$.

Remark 3.4. *Theorem 3.1 ensures that the accuracy of the approximation produced by the algorithm of this section depends primarily on the accuracy of the underlying interpolation scheme.*

3.3. **Construction of Chebyshev quadratures.** In this subsection, we describe a numerical algorithm for the construction of a Chebyshev quadrature for a finite sequence of functions. Given a pre-existing n-point quadrature formula $x_1, x_2, \ldots, x_n, w_1, w_1, \ldots, w_n$, where $n > k$, which exactly integrates the $u_1, \ldots, u_k$, it is straightforward to construct a Chebyshev quadrature for $u_1, \ldots, u_k$. Because the pre-existing quadrature rule exactly integrates the input functions $u_1, \ldots, u_k$, the matrix equation

$$\begin{pmatrix} u_1(x_1) & u_1(x_2) & \cdots & u_1(x_n) \\ u_2(x_1) & u_2(x_2) & \cdots & u_2(x_n) \\ \vdots & & \cdots & \vdots \\ u_k(x_1) & u_k(x_2) & \cdots & u_k(x_n) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{pmatrix}, \tag{3.10}$$

where r_i is defined by

$$r_i = \int_a^b u_j(x)\, dx$$

for all $i = 1, \ldots, k$, is satisfied. By Lemma 2.5, there exist $i_1, \ldots, i_n$ and $\tilde{w_1}, \ldots \tilde{w_k}$ such that

$$\begin{pmatrix} u_1(x_{i_1}) & u_1(x_{i_2}) & \cdots & u_1(x_{i_n}) \\ u_2(x_{i_1}) & u_2(x_{i_2}) & \cdots & u_2(x_{i_n}) \\ \vdots & & \cdots & \vdots \\ u_k(x_{i_1}) & u_k(x_{i_2}) & \cdots & u_k(x_{i_n}) \end{pmatrix} \begin{pmatrix} \tilde{w_1} \\ \tilde{w_2} \\ \vdots \\ \tilde{w_k} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{pmatrix} \tag{3.11}$$

and

$$\sum_{j=1}^{k} |\tilde{w_j}| \leq k \sum_{j=1}^{n} |w_j|. \tag{3.12}$$

In other words, there is by Lemma 2.5 a k-point quadrature formula $x_{i_1}, \ldots, x_{i_k}, \tilde{w_1}, \ldots, \tilde{w_k}$ for the k functions $u_1, \ldots, u_k$. Moreover, the inequality (3.12) ensures that the k-point quadrature formula is numerically stable *assuming* the weights of the original quadrature are reasonably small.

The following algorithm is a computational procedure for constructing such a quadrature via the modified Gram-Schmidt algorithm with double orthogonalization, or one of its variants. This procedure does not realize the theoretical bound (2.5) guaranteed by Lemma 2.5. In practice, however, it still leads to the construction of stable Chebyshev quadrature formulas for the input functions; see Subsection 2.8 for a more discussion of the numerical stability of the modified Gram-Schmidt algorithm and its variants.

The algorithm uses as input a sequence $u_1, \dots, u_k$ of functions and a pre-existing quadrature $x_1, \dots, x_n, w_1, \dots, w_n$, with $n > k$, which exactly integrates the functions $u_1, \dots, u_k$, and produces as output a k-point quadrature formula consisting of k nodes $\tilde{x_1}, \dots, \tilde{x_k} \in \{x_1, \dots, x_n\}$ and k weights $\tilde{w_1}, \dots, \tilde{w_k}$.

Description of the algorithm.

1. For the vector $r \in \mathbb{R}^k$ whose i^{th} entry is the sum
$$r_i = \sum_{j=1}^{n} u_i(x_j) w_j,$$
which is the value of the integral
$$\int_a^b u_i(x)\, dx.$$
2. Form the $k \times n$ matrix B with entries
$$B_{ij} = u_i(x_j)\sqrt{w_j}.$$
3. Use the pivoted Gram-Schmidt algorithm with double reorthogonalization to select a set $B_{i_1}, \dots, B_{i_k}$ of spanning columns for the matrix B and form the factorization
$$B = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$
where R_{11} is a $k \times k$ upper triangular matrix, Q is a $k \times k$ orthogonal matrix,
$$\begin{pmatrix} B_{i_1} & B_{i_2} & \cdots & B_{i_k} \end{pmatrix} = QR_{11},$$
and $\|R_{22}\|_2$ is small.
4. Use back substitution to construct a solution $z \in \mathbb{R}^k$ to the $k \times k$ system of equations
$$R_{11} z = Q^* r,$$
which is, of course, a least squares solution of the system
$$\begin{pmatrix} B_{i_1} & B_{i_2} & \cdots & B_{i_k} \end{pmatrix} z = r.$$
5. Form the new k-point quadrature $\tilde{x_1}, \dots, \tilde{x_k}$, $\tilde{w_1}, \dots, \tilde{w_k}$ by letting
$$\tilde{x_j} = x_{i_j} \quad \text{and} \quad \tilde{w_j} = z_j \sqrt{w_j}$$
for all $j = 1, \dots, k$.

Remark 3.5. *The columns of the matrix B are scaled by the square roots of the quadrature weights so that the l^2 norms of columns of B computed in Step 3 by the Gram-Schmidt orthonormalization procedure are proportional to the quadrature weight for the corresponding column.*

4. Numerical algorithm

This section describes a numerical algorithm for the construction of the nodes and weights of an approximate quadrature rule for a sequence of functions. The algorithm's input is a sequence of functions

$$\phi_1, \dots, \phi_m : [a, b] \to \mathbb{R} \tag{4.1}$$

and two accuracies, ϵ_{disc} and ϵ_{quad}. The first, ϵ_{disc}, controls the accuracy of the scheme used to discretize and compress the input functions $\phi_1, \dots, \phi_m$, and the second, ϵ_{quad}, is the desired accuracy for the quadrature rule. The algorithm's output is a quadrature rule consisting of a set of nodes $x_1, \dots, x_l$ and a set of weights $w_1, \dots, w_l$ such that

$$\int_a^b \phi_i(x)\, dx \sim \sum_{j=1}^{l} \phi_i(x_j) w_j \tag{4.2}$$

for all $i = 1, \dots, m$. The integer l depends on the numerical rank of the input set $\phi_1, \dots, \phi_m$ and both ϵ_{disc} and ϵ_{quad}.

The algorithm proceeds in three stages. In the first stage, the numerical techniques of the preceding section are used to discretize and compress the functions $\phi_1, \dots, \phi_m$. In the second, a suboptimal quadrature rule is obtained for the compressed collection of functions. Finally, in the third phase, an optimization procedure is used to reduce the number of points needed by the quadrature.

Stage 1: Discretization and compression.

In this stage the following sequence of steps is performed to discretize and compress the input functions.

1. Use the technique of Section 3.1 to discretize the functions $\phi_1, \dots, \phi_m$, so that they are all represented to the precision ϵ_{disc}. Let $\tilde{\phi_1}(x), \dots, \tilde{\phi_m}(x)$ denote the resulting discretizations. Also, let $x_1, \dots, x_n$ denote the discretization nodes and $w_1, \dots, w_n$ the weights of the associated quadrature.
2. Apply the procedure of Subsection 3.2 to compress the sequence $\tilde{\phi_1}, \dots, \tilde{\phi_n}$ via either a QR decomposition or the singular value decomposition. Denote the resulting orthonormal functions by $u_1, \dots, u_p$ and the associated singular values or normalization factors by $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p > 0$.
3. Discard any of the functions $u_1, \dots, u_p$ corresponding to a singular value (or normalization factor) $\lambda_j \leq \epsilon_{quad}$.
4. Denote the functions obtained in this manner by $u_1, \dots, u_k$ and the associated singular values (normalization factors) by $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k > 0$.

Remark 4.1. *It is important that ϵ_{quad} be somewhat larger than ϵ_{disc} (for the examples of this paper, $\epsilon_{quad}/\epsilon_{disc} \geq 100$). Because the functions $\phi_1, \dots, \phi_m$ are discretized to precision ϵ_{disc}, singular vectors u_j corresponding to singular values λ_j comparable to ϵ_{disc} contain little information about the functions $\phi_1, \dots, \phi_m$. This means that constructing a quadrature formula which faithfully integrates such singular functions is pointless, and moreover, since the u_j corresponding to singular values λ_j comparable to ϵ_{svd} are not necessarily smooth, even when the the functions $\phi_1, \dots, \phi_m$ are very smooth, it can cause difficulties with the the algorithm described in Step 3 below.*

Stage 2: Construction of a k-point quadrature rule.

We now apply the procedure of Subsection 3.3 to construct a k-point quadrature formula for $\phi_1, \dots, \phi_m$. We use as inputs to that procedure the functions $u_1, \dots, u_k$ and the nested Gaussian quadrature $x_1, \dots, x_n, w_1, \dots, w_n$ constructed in Stage 1. Note that since the $u_1, \dots, u_k$ are defined via the nested Gaussian interpolation scheme, the quadrature $x_1, \dots, x_n, w_1, \dots, w_n$ is exact for the functions $u_1, \dots, u_k$.

Denote the resulting k quadrature nodes by $\tilde{x_1}, \dots, \tilde{x_k}$ and the k quadrature weights by $\tilde{w_1}, \dots, \tilde{w_k}$. The weights of the nested Gaussian quadrature scheme satisfy the bound

$$\sum_{j=1}^{n} |w_j| \leq (b-a), \tag{4.3}$$

since $w_j > 0$ for all $j = 1, \dots, n$ and the quadrature rule is exact for the function $f(x) = 1$. It follows from the discussion in Subsection 3.3 that the new k-point quadrature formula is numerically stable

Because the discretizations $\tilde{\phi_1}, \dots, \tilde{\phi_m}$ are well approximated by functions in the span of the $u_1, \dots, u_k$, a quadrature rule exact for the functions $u_1, \dots, u_k$ will approximately integrate the functions $\tilde{\phi_1}, \dots, \tilde{\phi_m}$ (and, of course, it follows that such a quadrature rule will also serve for the functions $\phi_j(x)$, assuming the discretizations constructed in Step 1 are sufficiently accurate). So the new k-point quadrature rule $\tilde{x_1}, \dots, \tilde{x_k}, \tilde{w_1}, \dots, \tilde{w_k}$ is an approximate quadrature for the input functions $\phi_1, \dots, \phi_m$.

Stage 3: Point-by-point reduction of the quadrature rule.

In this stage, beginning with an n-point quadrature rule $x_1, \dots, x_n$,$w_1, \dots, w_n$, we repeatedly apply the following sequence of steps, which attempt to reduce an n-point quadrature rule for $u_1, \dots, u_k$ to an $(n-1)$-point quadrature rule for $u_1, \dots, u_k$.

To perform these calculations, we use the observation of Subsection 2.2; namely, that the nodes and weights of a quadrature rule exact for a sequence of functions satisfy a system of nonlinear equations. Given one of the n quadrature nodes x_j, we can use this fact to compute an $(n-1)$ point quadrature for the functions $u_1, \dots, u_k$ via the damped Gauss-Newton algorithm described in

Subsection 2.4: we simply form the system of k nonlinear equations in $n-1$ unknowns described in Subsection 2.2 and use as an initial guess for the Gauss-Newton iterations the current quadrature nodes and weights, excluding the chosen point x_j and its corresponding weight w_j.

The only problem with this approach is the selection of the quadrature node x_j to eliminate. There are n possible nonlinear systems, each corresponding to one of the n points which can be omitted. It is computationally expensive to search for a sufficiently accurate $(n-1)$-point quadrature rule by solving each of the n resulting nonlinear systems of equations via the damped Gauss-Newton method. That difficulty is surmounted via the procedure described in Step 1, wherein the direction of the step for the first iteration of the damped Gauss-Newton method is computed for each of the n possible nonlinear systems. The procedure is expedited by using the Sherman-Morrison-Woodbury update formula, which leads to a computationally feasible scheme, and the results are used to determine the order in which to remove quadrature nodes.

Step 1: Rank the remaining nodes.

1. For each $i = 1, \ldots, k$, compute the integrals

$$\int_a^b u_i(x)\, dx$$

using the original nested Legendre quadrature rule formed in Step 1. Form the vector

$$r = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{pmatrix}.$$

2. Form the Jacobian matrix

$$J = \begin{pmatrix} u_1'(x_1)w_1 & u_1'(x_2)w_2 & \ldots & u_1'(x_n)w_n & u_1(x_1) & u_1(x_2) & \ldots & u_1(x_n) \\ u_2'(x_1)w_1 & u_2'(x_2)w_2 & \ldots & u_2'(x_n)w_n & u_2(x_1) & u_2(x_2) & \ldots & u_2(x_n) \\ \vdots & & & \vdots & & & & \vdots \\ u_k'(x_1)w_1 & u_k'(x_2)w_2 & \ldots & u_k'(x_n)w_n & u_k(x_1) & u_k(x_2) & \ldots & u_n(x_n) \end{pmatrix}$$

for the nonlinear system (2.4) in Subsection (2.2), and compute the inverse

$$A = (JJ^t)^{-1}.$$

of the product JJ^t.

3. For each node x_k, first use the Sherman-Morrison-Woodbury formula (see Subsection 2.9) to form the matrix

$$A_k = (J_k {J_k}^t)^{-1}$$

via two rank-1 updates to A, where the matrix J_k is obtained from J by deleting its k^{th} and $(k+n)^{th}$ columns. That is, J_k is obtained from J by deleting the contributions from the node x_k and its corresponding weight w_k.

Next, compute the damped Gauss-Newton step direction Δx_k for the nonlinear system obtained by omitting the point x_k; i.e., find a solution Δx_k to the least squares problem

$$\operatorname*{argmin}_x \|J_k x - r\|_2,$$

where r is the vector of definite integrals computed above. The solution to the minimization problem is found by solving the normal equations

$$\Delta x_k = (J_k J_k^t)^{-1} J_k^t r = A_k J_k^t r.$$

4. For each $k = 1, \ldots, n$, let η_k denote the l^2 norm of the solution vector Δx_k. We will refer to the value η_k as the significance of the node x_k.
5. Renumber the nodes and weights of the quadrature so that $\{x_1, \ldots, x_n\}$ are arranged in order of increasing η_j.

x_i	w_i
0.2768757118897219E-21	0.4385873207003101E-20
0.6728999118496260E-16	0.6670167705957102E-15
0.2913313729367070E-12	0.2065563299999408E-11
0.1363525818876240E-09	0.7260286603891295E-09
0.1466530079834641E-07	0.6011573165242795E-07
0.5526176228089556E-06	0.1770124736858168E-05
0.9556438762298892E-05	0.2415867272187855E-04
0.9137352164364451E-04	0.1836327943155820E-03
0.5528197186769772E-03	0.8884287232428108E-03
0.2340613778507800E-02	0.3023731195124369E-02
0.7487402952102819E-02	0.7813101428951048E-02
0.1919273618673196E-01	0.1624899441619337E-01
0.4125089438263980E-01	0.2844094097856474E-01
0.7699431299223189E-01	0.4336382757029246E-01
0.1282821075952860E+00	0.5919402249047975E-01
0.1950313100451387E+00	0.7398735448703726E-01
0.2754007895160087E+00	0.8623391772361065E-01
0.3663421175354336E+00	0.9502522703162401E-01
0.4641496390531796E+00	0.9991966301938738E-01
0.5648167175087882E+00	0.1007221934294668E+00
0.6641918632800308E+00	0.9731824512000341E-01
0.7580165934350541E+00	0.8960557633305366E-01
0.8419459185715879E+00	0.7752702387328704E-01
0.9116491627096593E+00	0.6119598511874399E-01
0.9630711719694888E+00	0.4108284432044861E-01
0.9928824356753315E+00	0.1819930030488772E-01

x_i	w_i
0.1056470082726196E-21	0.1711096457804046E-20
0.3448276636263963E-16	0.3503308273337532E-15
0.1844497556672273E-12	0.1340813788691161E-11
0.1004613396825143E-09	0.5475770404764035E-09
0.1199010226476309E-07	0.5015698242641016E-07
0.4834665505524823E-06	0.1574374049854322E-05
0.8721446282278398E-05	0.2233377157238088E-04
0.8558255329196107E-04	0.1737239626794526E-03
0.5262332881183370E-03	0.8524600446470783E-03
0.2251655341009627E-02	0.2928241357781042E-02
0.7255657724649352E-02	0.7616225404410970E-02
0.1869906517557028E-01	0.1591836338093090E-01
0.4035483211529235E-01	0.2796728481689857E-01
0.7555760069233022E-01	0.4275972822079389E-01
0.1261876307279175E+00	0.5848899820837419E-01
0.1922011617225488E+00	0.7323146146527266E-01
0.2718197580245527E+00	0.8550269331832136E-01
0.3620819036055107E+00	0.9441537793180152E-01
0.4593842035808489E+00	0.9953654606755140E-01
0.5598231960985849E+00	0.1006639600889222E+00
0.6593340785103139E+00	0.9765633871558868E-01
0.7537053806233217E+00	0.9035707472894443E-01
0.8385666970462145E+00	0.7861734429197797E-01
0.9094519100536209E+00	0.6242511779052747E-01
0.9620432996182802E+00	0.4213323000112514E-01
0.9926707956133087E+00	0.1873187135192821E-01

TABLE 1. Quadrature formulas for the functions of the form (5.2) with $\alpha \in [-.6, 1.0]$ and $b = 20$. The 26-point rule on the left was generated with the QR variant of the algorithm, and the 26-point rule on the right with the SVD variant. Both achieve full double precision accuracy.

Step 2: First pass through the nodes.

For each $j = 1, \dots, n$, perform the following sequence of steps:

1. Form an initial guess $x_1, \dots, \hat{x_j}, \dots, x_n$ and $w_1, \dots, \hat{w_j}, \dots, w_n$ for the damped Gauss-Newton method, which excludes the node x_j and its corresponding weight w_j.
2. Perform a small number (e.g., 4) of damped Gauss-Newton iterations to form a $(n-1)$-point quadrature rule $\tilde{x_1}, \dots, \tilde{x_{n-1}}, \tilde{w_1}, \dots, \tilde{w_{n-1}}$ for the functions $u_1, \dots, u_k$.
3. Measure the approximation error

$$\epsilon_j = \sum_{i=1}^{k} \left| \sum_{j=1}^{n-1} u_i(\tilde{x}_j)\tilde{w}_j - r_i \right|^2$$

   for this new quadrature rule.
4. If the error ϵ_j for the quadrature rule is sufficiently small (i.e., $\epsilon_j \leq \epsilon_{quad}$), then we accept this $(n-1)$-point quadrature rule and go to Step 4.

If this sequence of steps completes without finding a quadrature rule with acceptable accuracy, then we renumber the points $x_1, \dots, x_n$ and weights $w_1, \dots, w_n$ so that

$$\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_n,$$

and move onto Step 3.

Step 3: Second pass through the nodes.

We arrive at this stage only if we were unable to find a satisfactory $(n-1)$-point quadrature rule by taking a small number of damped Gauss-Newton steps. For each $j = 1, \dots, n$ we perform the following sequence of steps:

1. Form an initial guess $x_1, \dots, \hat{x_j}, \dots, x_n$ and $w_1, \dots, \hat{w_j}, \dots, w_n$ for the damped Gauss-Newton method, which excludes the node x_j and its corresponding weight w_j.
2. Use the damped Gauss-Newton algorithm to form an $(n-1)$-point quadrature rule $\tilde{x_1}, \dots, \tilde{x_{n-1}}$, $\tilde{w_1}, \dots, \tilde{w_{n-1}}$ for the functions $u_1, \dots, u_k$. In this step, the limit m on the number of iterations should be large (for the examples of this paper, $m = 30$).

x_i	w_i
0.7142868061585990E-22	0.1160152107855531E-20
0.2456099244659448E-16	0.2509065273801540E-15
0.1388771056068208E-12	0.1016875670063657E-11
0.7923942786393112E-10	0.4349600879799666E-09
0.9728977588843004E-08	0.4086717652474466E-07
0.3945858602101462E-06	0.1282739831111961E-05
0.7004999008550828E-05	0.1775840695185165E-04
0.6647847640796673E-04	0.1322763958769828E-03
0.3909094695073146E-03	0.6144256207665815E-03
0.1591555788166976E-02	0.1989634887947442E-02
0.4880565012011741E-02	0.4888053092794360E-02
0.1201508322521625E-01	0.9711002899742405E-02
0.2491818304126502E-01	0.1636280699646038E-01
0.4515299730588538E-01	0.2424264956181601E-01
0.7352628756922541E-01	0.3249392541983736E-01
0.1099997234334442E+00	0.4033170405949942E-01
0.1538773279072512E+00	0.4724229693878507E-01
0.2040969860767333E+00	0.5299744130306683E-01
0.2594754054419947E+00	0.5756526955130879E-01
0.3188532789291890E+00	0.6101116645139500E-01
0.3811550534088713E+00	0.6342859582873963E-01
0.4453957796025275E+00	0.6490097796110366E-01
0.5106608382341754E+00	0.6548363166007094E-01
0.5760733409165308E+00	0.6519584582093280E-01
0.6407559864297044E+00	0.6401701018681496E-01
0.7037897986529295E+00	0.6188400467055524E-01
0.7641706741669453E+00	0.5868958877875049E-01
0.8207656867459757E+00	0.5428419177206927E-01
0.8722752374352830E+00	0.4848722081318521E-01
0.9172156595977674E+00	0.4111890543212653E-01
0.9539498396044303E+00	0.3206665104837302E-01
0.9808044766847136E+00	0.2139086878064731E-01
0.9963048714773390E+00	0.9450771617397615E-02

x_i	w_i
0.9545672035004698E-22	0.1529382502435904E-20
0.2782141914841101E-16	0.2808039205209724E-15
0.1433442445848540E-12	0.1039911929764398E-11
0.7793572117442273E-10	0.4253421034031052E-09
0.9385234534215939E-08	0.3933760334981154E-07
0.3797811590248750E-06	0.1235803705030886E-05
0.6783814687534227E-05	0.1725353179365193E-04
0.6495463184473910E-04	0.1298334366282352E-03
0.3852594648391139E-03	0.6086396169406588E-03
0.1580066444838872E-02	0.1985699293680274E-02
0.4874206864113175E-02	0.4907602643213011E-02
0.1205644750230908E-01	0.9794765851446592E-02
0.2509445953623319E-01	0.1655504783796611E-01
0.4558557083460269E-01	0.2456292098104041E-01
0.7433570351228120E-01	0.3292041369959096E-01
0.1112672046075921E+00	0.4081110237571449E-01
0.1556251204014568E+00	0.4771399178998336E-01
0.2062913928215772E+00	0.5341223120206637E-01
0.2620423475006598E+00	0.5789177660270919E-01
0.3216956048993508E+00	0.6123393859233695E-01
0.3841661902355933E+00	0.6354343717872484E-01
0.4484687328581966E+00	0.6491062297564529E-01
0.5136935077033405E+00	0.6539475441359519E-01
0.5789715342131883E+00	0.6501741217619580E-01
0.6434353506653515E+00	0.6375971643567762E-01
0.7061776301391591E+00	0.6156055973389270E-01
0.7662083501992056E+00	0.5831564380342947E-01
0.8224122548819472E+00	0.5387967213251687E-01
0.8735123565945287E+00	0.4807758967228032E-01
0.9180534258335345E+00	0.4073548782515963E-01
0.9544319708711901E+00	0.3174482830959985E-01
0.9810102430752318E+00	0.2116533404178935E-01
0.9963450010115773E+00	0.9348448278692109E-02

TABLE 2. Quadrature formulas for the functions of the form (5.2) with $\alpha \in [-.6, 1.0]$ and $b = 50$. The 33-point rule on the left was generated with the QR variant of the algorithm, and the 33-point rule on the right with the SVD variant. Both achieve full double precision accuracy.

3. Measure the approximation error

$$\epsilon_j = \sum_{i=1}^{k} \left| \sum_{j=1}^{n-1} u_i(\tilde{x}_j)\tilde{w}_j - \int_a^b u_i(x)\,dx \right|^2$$

for this new quadrature rule.

4. If the error ϵ_j for the quadrature rule is sufficiently small (i.e., $\epsilon_j \leq \epsilon_{quad}$), then we accept this $(n-1)$-point quadrature rule and go to Step 4.

Step 4: Form the $(n-1)$-point quadrature.

If an $(n-1)$ point quadrature rule with sufficient precision has been found, then we accept this rule and repeat the procedure of this stage for the newly formed $(n-1)$ point quadrature, beginning with Step 1. Otherwise, we accept the n-point quadrature rule which was the input to this stage and the algorithm terminates.

Remark 4.2. *The process terminates when an n-point quadrature rule cannot be reduced to an $(n-1)$-point quadrature rule without an unacceptable loss of precision.*

Remark 4.3. *We would like to reiterate that the quadrature rules obtained by this algorithm are not exact for the functions $\phi_1, \ldots, \phi_m$, but rather depend on the precision of the approximation used and the pointwise properties of the ϕ_j.*

5. Numerical Examples

We have implemented the algorithm of this paper for the computation of efficient quadrature rules and tested it on a number of examples. Below we present several of these examples in order to demonstrate the performance of the algorithm. It was implemented in Fortran 77, compiled with the Lahey-Fujitsu FORTRAN 95, and all timings were measured on a 2.0 GHz Intel Core 2 Duo processor with 2GB of RAM (no parallelization was utilized). Where possible, computations were performed in double precision (FORTRAN REAL*8) arithmetic; however, in order to achieve double precision accuracy for quadrature rules, it is necessary to perform the computations in extended precision (FORTRAN REAL*16). Because the Intel Core 2 Duo processor does not support extended precision arithmetic in hardware, the running times for these computations are quite long.

Throughout this section, we will denote by $J_n(z)$ the n^{th} order Bessel function of the first kind, by $Y_n(z)$ the n^{th} order Bessel function of the second kind, and by $H_n(z)$ the n^{th} order Hankel function of the first kind,

$$H_n(z) = J_n(z) + iY_n(z). \tag{5.1}$$

5.1. **Functions which are both oscillatory and singular.** Classical quadrature techniques, like Gaussian quadratures, perform poorly when the functions to be integrated exhibit more than one kind of oscillatory or singular behavior. In this example, we present quadrature rules for functions $[0,1] \to \mathbb{R}$ of the form

$$\sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{i,j} \cos(\beta_j x) + b_{i,j} \sin(\beta_j x) \right) x^{\alpha_i}, \tag{5.2}$$

where the α_i are arbitrary real numbers on the interval $[-.6, 1.0]$, and the β_j are arbitrary real numbers on an interval $[0, b]$.

To construct such quadratures, we first chose fairly large integers m and n, and then construct n Legendre nodes α_i on the interval $[-.6, 1.0]$ and m Legendre nodes β_j on the interval $[0, b]$. We then take all functions of the form

$$x^{\alpha_i} \cos(\beta_j x) \quad \text{and} \quad x^{\alpha_i} \sin(\beta_j x) \tag{5.3}$$

x_i	w_i
0.1229785612931944E-21	0.1975656354324724E-20
0.3552652759524009E-16	0.3567391763467614E-15
0.1686329315315300E-12	0.1205245279219683E-11
0.8100212080727273E-10	0.4310646162233048E-09
0.8480396519577635E-08	0.3435473776420015E-07
0.2999419813536928E-06	0.9380681263322151E-06
0.4775750160241328E-05	0.1165688301296981E-04
0.4182340572497058E-04	0.8041357440032319E-04
0.2326791577716382E-03	0.3553400421394265E-03
0.9141444687391752E-03	0.1117568231732080E-02
0.2744584412742149E-02	0.2706368315522836E-02
0.6682130398844386E-02	0.5351823901681276E-02
0.1379497029777539E-01	0.9027505156251501E-02
0.2498028781481084E-01	0.1342713453729264E-01
0.4073593719986401E-01	0.1808816226118706E-01
0.6109753716986285E-01	0.2257750666507361E-01
0.8573787689934898E-01	0.2661334339415909E-01
0.1141325013227837E+00	0.3007780005914209E-01
0.1457010079575668E+00	0.3296666645566729E-01
0.1798914175653836E+00	0.3533282633911356E-01
0.2162162828182287E+00	0.3724775994063266E-01
0.2542599995624801E+00	0.3878153866772767E-01
0.2936726106825575E+00	0.3999467106048757E-01
0.3341588668019299E+00	0.4093600316907737E-01
0.3754666793859390E+00	0.4164319265743018E-01
0.4173765534810784E+00	0.4214400183613144E-01
0.4596923705410667E+00	0.4245762431761676E-01
0.5022333728490615E+00	0.4259573188716399E-01
0.5448270050049624E+00	0.4256313223505469E-01
0.5873022022823650E+00	0.4235800106227075E-01
0.6294826883836380E+00	0.4197166388463288E-01
0.6711798141862015E+00	0.4138788759007980E-01
0.7121844171914087E+00	0.4058161770611706E-01
0.7522571077315104E+00	0.3951708131079590E-01
0.7911163140906269E+00	0.3814519845080301E-01
0.8284234137301751E+00	0.3640037679024763E-01
0.8637645183766196E+00	0.3419715966891961E-01
0.8966293685769854E+00	0.3142816445555609E-01
0.9263901328780540E+00	0.2796677541655260E-01
0.9522879806764252E+00	0.2368149162837343E-01
0.9734441058145912E+00	0.1847233461780559E-01
0.9889218194127054E+00	0.1233680051765303E-01
0.9978661596654697E+00	0.5456702457360249E-02

x_i	w_i
0.7962322346848315E-22	0.1255518558516502E-20
0.1830884530860187E-16	0.1807251002083772E-15
0.7683378387780577E-13	0.5430995592381254E-12
0.3548822655627213E-10	0.1887682739748173E-09
0.3820064929774288E-08	0.1568277392295615E-07
0.1453085581405940E-06	0.4671196468523313E-06
0.2542069046177732E-05	0.6453818186891716E-05
0.2452467963854058E-04	0.4945575220250773E-04
0.1490989514657686E-03	0.2400742808398220E-03
0.6325800496319742E-03	0.8181326443575235E-03
0.2027015096036673E-02	0.2119810413961939E-02
0.5213355986400703E-02	0.4437537936998829E-02
0.1127123616180105E-01	0.7852211174980829E-02
0.2121486403535344E-01	0.1215005865408980E-01
0.3571945099051861E-01	0.1689083767930460E-01
0.5498138040746230E-01	0.2159081784351869E-01
0.7876554840715572E-01	0.2589061574717723E-01
0.1065681257681138E+00	0.2961183182803991E-01
0.1377839750139660E+00	0.3271991578613246E-01
0.1718183483671372E+00	0.3526008081919886E-01
0.2081399215101480E+00	0.3730762085807319E-01
0.2462954187930989E+00	0.3894030217190292E-01
0.2859051752770733E+00	0.4022648983646625E-01
0.3266515434452085E+00	0.4122168685192172E-01
0.3682659628410250E+00	0.4196875223655011E-01
0.4105169996086324E+00	0.4249935402437477E-01
0.4531999641092431E+00	0.4283556050595227E-01
0.4961279927664242E+00	0.4299113717974382E-01
0.5391242133546008E+00	0.4297240498182741E-01
0.5820145309561993E+00	0.4277862015965356E-01
0.6246205472012950E+00	0.4240185727864892E-01
0.6667521011409383E+00	0.4182636166562592E-01
0.7081988727858907E+00	0.4102730967187124E-01
0.7487204161980902E+00	0.3996889290357553E-01
0.7880339071872188E+00	0.3860165671827315E-01
0.8257988675123893E+00	0.3685914346939630E-01
0.8615983375780399E+00	0.3465427456539984E-01
0.8949168202476468E+00	0.3187687491511273E-01
0.9251176490208435E+00	0.2839583726541435E-01
0.9514276420773934E+00	0.2407310551022889E-01
0.9729462427273851E+00	0.1880072022809006E-01
0.9887074670174289E+00	0.1256986040620092E-01
0.9978240533043199E+00	0.5563859224973946E-02

TABLE 3. Quadrature formulas for the functions of the form (5.2) with $\alpha \in [-.6, 1.0]$ and $b = 100$. The 43-point rule on the left was generated with the QR variant of the algorithm, while the 43-point rule on the right was generated with the SVD variant. Both achieve full double precision accuracy.

as the input functions ϕ_i to the algorithm of Section 5. It is clear the for sufficiently large n and m, the obtained quadrature will work for all functions of the form (5.2).

In Tables 1, 2, and 3, we list the quadrature nodes and weights for $b = 20, 50$, and 100. In each case, two quadrature rules were computed, one using the QR decomposition variant of the algorithm and one using the SVD variant. The parameters m and n were chosen to be $m = 900$ and $n = 100$. In Table 4, we report the time spent computing each of these quadratures as well as the number of quadrature nodes required. These computations were performed in extended precision arithmetic in order to ensure double precision accuracy for the resulting quadrature rule.

We also computed single precision (10^{-8}) quadrature rules for $b = 20, 50$, and 100. These computations were performed in double precision arithmetic and the results are reported in Table 5.

	$b=20$		$b=50$		$b=100$	
	QR	SVD	QR	SVD	QR	SVD
t_{cheb}	208.11	213.21	447.01	457.93	894.71	933.67
t_{newt}	196.23	195.03	509.25	634.77	1277.34	1315.04
t_{tot}	404.94	408.00	956.14	1091.22	2171.12	2248.01
# *nodes*	26	26	33	33	43	43

Table 4. Execution times (in seconds) for the computation of the double precision accuracy quadratures of Example 5.1, as well as the number of quadrature nodes. Results for both the SVD and QR variants of the algorithm are provided. The CPU time taken by the first two stages of the algorithm is reported as t_{cheb} while the CPU time for Stage 3 is reported as t_{newt}.

	$b=20$		$b=50$		$b=100$	
	QR	SVD	QR	SVD	QR	SVD
t_{cheb}	1.15	1.18	1.88	1.94	3.28	3.26
t_{newt}	6.78	6.43	9.33	9.86	15.0	22.3
t_{tot}	7.93	7.61	11.2	11.8	18.2	25.5
# *nodes*	15	15	21	21	30	30

Table 5. Execution times (in seconds) for the computation of the single precision accuracy quadratures of Example 5.1, as well as the number of quadrature nodes. Results for both the SVD and QR variants of the algorithm are provided. The CPU time taken by the first two stages of the algorithm is reported as t_{cheb} while the CPU time for Stage 3 is reported as t_{newt}.

5.2. **Plane wave expansions.** The Green's function for the Helmholtz equation in $\mathbb{R}^3$ satisfies the following identity, valid for $z > 0$:

$$\frac{e^{i\omega r}}{r} = \int_0^\infty e^{-z\sqrt{\xi^2-\omega^2}} J_0(\xi\sqrt{x^2+y^2}) \frac{\xi}{\sqrt{\xi^2-\omega^2}}\, d\xi, \tag{5.4}$$

where $r = \sqrt{x^2+y^2+z^2}$. This formula can be derived by applying the Fourier Inversion Theorem followed by contour integration. In [9], a scheme for accelerating fast multipole methods for the Helmholtz equations at low frequency was introduced. It operates via discretizations of formula (5.4) which hold for x, y, and z satisfying

$$L \le z \le 4L$$

$$-4L \le x, y \le 4L. \tag{5.5}$$

The appropriate quadrature rule ostensively depends on the two parameters ω and L. In fact, by making the substitutions

$$\lambda = \frac{\xi}{L} \quad \text{and} \quad \omega = L\omega_0$$

in (5.4), we obtain the equivalent representation

$$\frac{e^{i\omega r}}{r} = L \int_0^\infty e^{-(zL)\sqrt{\lambda^2-\omega_0^2}} J_0(\lambda\sqrt{(xL)^2+(yL)^2}) \frac{\lambda}{\sqrt{\lambda^2-\omega_0^2}}\, d\lambda, \tag{5.6}$$

which shows that up to rescaling factors, the quadrature rule depends only on the product ωL, which is the size of the box (5.5) in wavelengths. Moreover, since $J_0(z)$ satisfies the well-known identity

$$J_0(z) = \frac{1}{\pi}\int_0^\pi \cos(z\sin(\theta))\, d\theta, \tag{5.7}$$

which can be found as formula (9.1.18) in [1], in order to generate a discretization of (5.4) which holds for

$$a \le \omega L \le b,$$

it suffices to take as input to the algorithm of Section 5 functions of the form

$$e^{-x\sqrt{\lambda^2-\omega^2}} \frac{\lambda\cos(y\lambda)}{\sqrt{\lambda^2-\omega^2}}, \tag{5.8}$$

where x, y, and ω are allowed to vary as

$$\begin{aligned} 1 &\le x \le 4 \\ 0 &\le y \le 4b\sqrt{2}, \\ a &\le \omega \le b. \end{aligned} \tag{5.9}$$

ωL	Expansion order							
	$\epsilon = 10^{-3}$		$\epsilon = 10^{-6}$		$\epsilon = 10^{-9}$		$\epsilon = 10^{-15}$	
	QR	SVD	QR	SVD	QR	SVD	QR	SVD
.5	13	13	30	29	44	44	72	72
5	18	19	33	33	46	46	80	79
10	22	24	36	35	48	49	84	84
25	44	44	53	52	65	63	94	94

TABLE 6. Orders of the plane wave expansions of Example 5.2 as a function of box size and accuracy.

The algorithm of this paper was applied to functions of the form (5.8) in order to generate discretizations of formula (5.4) of the form

$$\frac{e^{i\omega r}}{r} \simeq \sum_j e^{-z\sqrt{\xi_j^2-\omega^2}} J_0(\xi_j\sqrt{x^2+y^2}) \frac{\xi_j}{\sqrt{\xi_j^2-\omega^2}} w_j \tag{5.10}$$

for boxes of varying sizes. The Stage 1 and Stage 2 discretization steps were performed twice for each choice of ωL and variant (QR or SVD) of the algorithm, once using extended precision arithmetic and once using double precision arithmetic. The Stage 3 algorithm was then used repeatedly to generate quadratures of varying accuracies for each of the boxes. The number of terms in the plane wave expansion is given in Table 6 as a function of the size of the box in wavelengths, the required accuracy, and which variant (QR or SVD) of the algorithm is used. Here accuracy is measured as the largest absolute error occurring in formula (5.10).

Table 7 reports the total time taken by the Stage 1 and Stage 2 computations. Finally, Table 8 gives the time taken by the Stage 3 procedure. Note that all Stage 3 computations were performed in double precision arithmetic, except for the quadrature rules with accuracy 10^{-15}; those computations were performed in extended precision.

	$\omega L = .5$		$\omega L = 5$		$\omega L = 10$		$\omega L = 25$	
	QR	SVD	QR	SVD	QR	SVD	QR	SVD
t_{double}	8.84	9.03	8.23	8.60	7.17	8.10	9.14	9.84
t_{quad}	1858.96	1954.31	2046.89	2192.72	2079.71	2227.13	2486.69	2644.03

Table 7. Time (in seconds) taken by the Stage 1 and Stage 2 procedures for quadrature rules of Example 5.2; t_{double} gives the CPU time for the double precision computations and t_{quad} gives the total CPU time for the extended precision computations.

ωL	CPU Time (seconds)							
	$\epsilon = 10^{-3}$		$\epsilon = 10^{-6}$		$\epsilon = 10^{-9}$		$\epsilon = 10^{-15}$	
	QR	SVD	QR	SVD	QR	SVD	QR	SVD
.5	0.58	0.61	3.19	2.81	15.01	10.33	12055.45	12398.62
5	0.80	0.86	3.95	3.58	12.01	11.32	18468.92	17513.46
10	1.25	1.31	5.03	4.59	14.31	14.21	22018.44	22662.14
25	7.61	8.01	17.22	1.57	38.32	35.21	34646.45	36221.12

Table 8. CPU time (in seconds) required to perform the Stage 3 computations for the quadratures of Example 5.2.

5.3. **Integral representations for H-expansions.**

Let (p, θ) denote the polar coordinate system defined by

$$x = p\cos(\theta) \quad \text{and} \quad y = p\sin(\theta).$$

As is well known (see, for instance, [18]), if a function $\phi : \mathbb{R}^2 \to \mathbb{C}$ satisfies the Helmholtz equation

$$\nabla^2 \phi + \omega \phi = 0$$

outside of a disc D of radius R centered at 0 and the radiation condition

$$\lim_{t \to \infty} \phi(tx) e^{-ikt|x|} \sqrt{t} = c$$

at ∞, then $\phi(x)$ can be uniquely represented outside the disc D via the H-expansion

$$\phi(x) = \sum_{n=-\infty}^{\infty} \beta_n H_n(\omega p) e^{in\theta}. \tag{5.11}$$

Moreover, once $N > |\omega| R$, the error in the approximation

$$\phi(x) \sim \sum_{n=-N}^{N} \beta_n H_n(\omega p) e^{in\theta}. \tag{5.12}$$

for $|x| = R_1 > R$ decays as $(R_1/R)^N$ (see, for instance, [18]).

The formula

$$H_n(\omega\rho)e^{in\theta} = \frac{(-1)^n}{\pi}\int_{-\infty}^{\infty} \frac{e^{iy\sqrt{\omega^2-\xi^2}}}{\sqrt{\omega^2-\xi^2}} \left(\frac{i\xi - \sqrt{\omega^2-\xi^2}}{\omega}\right)^n e^{i\xi x}\, d\xi, \tag{5.13}$$

valid for $y > 0$, can be obtained via manipulation of the well-known representation

$$H_n(z) = \frac{(-1)^n}{\pi}\int_C e^{iz\cos(w)+inw}\, dw, \tag{5.14}$$

where C is a properly chosen contour in the complex plane (see [5] for a simple derivation of (5.14)). In this example, we construct quadratures for formula (5.13) which hold for x and y satisfying

$$\begin{aligned} 2L \le y \le 4L \\ -L \le x \le L, \end{aligned} \tag{5.15}$$

and for $n = 0, 1, \ldots, M$. Since

$$H_{-n}(z) = (-1)^n H_n(z),$$

which is formula (9.1.6) in [1], an L-point quadrature rule for functions of this form allows us to approximate an H-expansion

$$\psi = \sum_{n=-M}^{M} a_n H_n(\omega p) e^{in\theta} \tag{5.16}$$

via a sum

$$\psi \simeq \sum_{j=1}^{L} b_j(\omega) e^{i\lambda_j x + \sqrt{\omega^2-\lambda_j^2}y}. \tag{5.17}$$

ωL	M=2		M=4		M=8	
	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$
1	23	57	27	63	32	66
5	29	60	29	64	33	68
10	33	66	35	67	36	72
15	39	70	39	73	41	79

TABLE 9. Orders of the expansions (5.17).

If we make the substitutions

$$\lambda = \xi/L \quad \text{and} \quad \omega = L\omega_0 \tag{5.18}$$

into formula (5.14), then we obtain the equivalent representation

$$H_n(\omega\rho)e^{in\theta} = \frac{(-1)^n}{\pi}\int_{-\infty}^{\infty} \frac{e^{i\lambda(xL)+i(yL)\sqrt{\omega_0^2-\lambda^2}}}{\sqrt{\omega_0^2-\lambda^2}} \left(\frac{i\lambda - \sqrt{\omega_0^2-\lambda^2}}{\omega_0}\right)^n d\lambda, \tag{5.19}$$

which shows that as with the last example, the appropriate discretization for (5.13) depends only on ωL.

The QR variant of the algorithm was used to compute quadrature rules for function of the form (5.16) for various values of ωL and M. Table 9 gives the number of terms in the expansion (5.17) necessary to obtain precisions 10^{-7} and 10^{-15} for boxes of different sizes and for different values of M. Table 10 gives the time required to perform the Stage 1 and Stage 2 computations for these quadrature rules, while Table 11 gives the time taken by the Stage 3 computations. Note that, as usual, the computations for quadrature rules with accuracy 10^{-7} were performed in double precision arithmetic while those for quadrature rules with 10^{-15} were performed in extended precision arithmetic.

ωL	M=2		M=4		M=8	
	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$
1	18.50	1999.02	24.16	2796.14	45.12	4274.44
2	24.65	2872.72	32.96	3811.83	55.87	5863.14
5	29.47	3724.73	41.62	4693.70	66.33	7762.44
15	37.92	5234.68	50.98	6359.60	78.56	8702.26

Table 10. Time (in seconds) required by the Stage 1 and Stage 2 computations for the quadrature rules of Example 5.3.

ωL	M=2		M=4		M=8	
	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-15}$
1	1.01	4238.30	1.55	6302.25	2.85	8411.11
5	2.08	5694.78	2.17	6011.20	3.26	8302.36
10	3.38	7604.56	3.76	6540.80	4.36	9221.55
15	6.39	9881.15	6.86	10861.93	6.94	12322.11

Table 11. Time (in seconds) taken by the Stage 3 computations for the quadrature rules of Example 5.3.

6. Generalizations and Conclusions

We have presented a simple and robust scheme for the computation of efficient quadrature rules for a wide class of functions. We demonstrated this algorithm by generating efficient quadrature rules for several different classes of functions, including systems of functions exhibiting different kinds of singular and oscillatory behavior.

We close with a number of conclusions and possible generalizations of this work:

1. The results of this paper are purely experimental. While there is a framework for proving (under certain conditions) the existence of generalized Gaussian quadratures (see [11, 12, 15, 16, 13]), it does not apply to many of the examples of this paper. Indeed, the numerical experiments of this paper and those of [14, 3, 20] seem to indicate that such quadratures exist under very general conditions.
2. Stage 3 of the algorithm of Section 4, wherein quadrature nodes are eliminated one-by-one, is applicable to a wide range of problems. The method applies wherever a sparse solution for an underdetermined nonlinear system of equations is being sought.
3. The quadrature weights generated by the algorithm of this paper are generally, but not necessarily, positive. In many applications, positive quadrature weights are desirable. A modification of the algorithm to ensure that the weights of the resulting quadrature formulae are positive might prove useful.

   The Chebyshev quadrature procedure of Stage 2 of the algorithm could be modified to produce positive weights by replacing the least squares minimization problem with a linear program. That program can be solved via the simplex method, which would result in a Chebyshev quadrature. The Stage 3 Newton iterations could be modified in a number of ways (e.g. barrier methods) to ensure, or at least encourage, positive quadrature weights.
4. Although there are several difficulties that must be overcome, there is no fundamental barrier to generalizing the procedure of this paper to higher dimensions. A procedure for generating efficient quadrature rules for collections of functions defined on domains in $\mathbb{R}^2$ would have many applications in numerical analysis. This topic is being vigorously investigated by the authors.

5. The scheme of this paper has obvious application to the interpolation of functions. In particular, the observations of Subsection 2.3 and the algorithm of this paper allow for the construction of stable, efficient interpolation formulas for highly singular and oscillatory functions.
6. There are a number of applications of this work to the discretization of integral equations. The ability to produce efficient quadratures for broad classes of collections of functions is by itself useful in the discretization of integral equations, and the general framework of this paper for "downsampling" a quadrature formula should allow for the reduction of the complexity of discretizations of integral equations.

References

[1] M. Abramowitz and I. Stegun(editors), *Handbook of Mathematical Functions*, National Bureau of Standard, 1964.

[2] A. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadephia, 1996.

[3] H. Cheng, V. Rokhlin, and N. Yarvin, *Nonlinear optimization, quadrature, and interpolation*, SIAM J. Optim, 9 (1999), pp. 901–923.

[4] W. Chew, E. Michielssen, J. Song, and J. Jin, *Fast and Efficient Algorithms in Computational Electrodynamics*, Artech House, Norwood, MA, 2001.

[5] R. Courant and D. Hilbert, *Methods of Mathematical Physics, Vol. 1*, John Wiley & Sons, New York, 1991.

[6] G. Dahlquist and A. Björck, *Numerical Methods*, Dover Publications, Mineola, New York, 2003.

[7] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[8] G. Golub and C. V. Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1983.

[9] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, *Accelerating fast multipole methods for the helmholtz equation at low frequencies*, IEEE Comput. Sci. Eng, 5 (1998), pp. 32–38.

[10] M. Gu and S. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

[11] S. Karlin, *The existence of eigenvalues for integral operators*, Trans. Amer. Math Soc., 113 (1964), pp. 1–17.

[12] S. Karlin and W. Studden, *Tchebycheff systems with applications in Analysis and Statistics*, Wiley-Interscience, New York, 1966.

[13] M. Krein, *The Ideas of P.L. Chebyshev and A.A. Markov in the Theory of Limiting Values of Integrals, Amer. Math. Soc. Transl. Ser. 2, 12*, AMS, Providence, R.I., 1959.

[14] J. Ma, V. Rokhlin, and S. Wandzura, *Generalized Gaussian quadrature rules for systems of arbitrary functions*, SIAM J. Numer. Anal., 33 (1996), pp. 971–996.

[15] A. Markov, *On the limiting value of integrals in connection with interpolation*, Zap. Imp. Akad. Nauk. Fix-Mat. Otd., 6 (1898). (in Russian).

[16] ———, *Selected Papers on Continued Fractions and the Theory of Functions Deviating Least from Zero*, OGIZ, Moscow, Leningrad, 1948. (in Russian).

[17] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *On interpolation and integration in finite-dimensional spaces of bounded functions*, Communications in Applied Mathematics and Computational Science, 1 (2006), pp. 133–142.

[18] P. Morse and H. Feshbach, *Methods of Mathematical Physics*, Feshbach Publishing, Minneapolis, 1981.

[19] Tyrtyshnikov, *A Brief Introduction to Numerical Analysis*, Birkhäuser, Boston, 1997.

[20] N. Yarvin and V. Rokhlin, *Generalized Gaussian quadratures and singular value decompositions of integral operators*, SIAM J. Sci. Comput., 20 (1998), pp. 699–718.