SIAM J. SCI. COMPUT.
Vol. 0, No. 0, pp. 000–000

AN EFFICIENT ALGORITHM FOR THE CLASSICAL LEAST SQUARES APPROXIMATION[*]

DIMITAR K. DIMITROV[†] AND LOURENÇO L. PEIXOTO[‡]

Abstract. We explore the computational issues concerning a new algorithm for the classical least-squares approximation of N samples by an algebraic polynomial of degree at most n when the number N of the samples is very large. The algorithm is based on a recent idea about accurate numerical approximations of sums with large numbers of terms. For a fixed n, the complexity of our algorithm in double precision accuracy is $\mathcal{O}(1)$. It is faster and more precise than the standard algorithm in MATLAB.

Key words. least squares approximation, Gaussian quadrature, orthogonal Gram polynomials, WDDK method, Newton–Raphson method, Golub–Welsch algorithm

AMS subject classifications. 65D10, 65D30, 65H04

DOI. 10.1137/19M1259936

1. Introduction.

There are various methods for computing reasonable polynomial fits to equally spaced data. These include Hoffman and Reddy's mock-Chebyshev interpolation [22], Boyd's Tikhonov regularization approach [8], and Boyd and Ong's multi-interval scheme [9]. For a comprehensive list of methods for approximating analytic functions from equispaced samples see [25].

The present paper is the final part of a project aimed at designing an efficient algorithm for the classical least squares approximation problem (CLSAP) for very large data sets. Given equally spaced mesh points x_j, $j = 1, \dots, N$, and real-valued data $y_j = f(x_j)$, one must determine an algebraic polynomial $p_n(x; f)$ of degree not exceeding n, where n is much smaller than N, which minimizes the corresponding l^2 norm. More precisely, if $\mathbb{P}_n$ denotes the space of real algebraic polynomials of degree at most n, the CLSAP is the problem of computing $p_n(x; f) \in \mathbb{P}_n$, such that

$$\|f - p_n(\cdot\,; f)\|^2 = \min_{p\in\mathbb{P}_n} \|f - p\|^2 = \min_{p\in\mathbb{P}_n} \sum_{j=1}^{N} \left(f(x_j) - p(x_j)\right)^2 . \tag{1.1}$$

Since $p_n(x; f)$ is the projection of f onto $\mathbb{P}_n$, it exists and is unique. Moreover, the best representation of the solution p_n is in terms of its Fourier series with respect to the corresponding orthogonal basis (see [21, Chapter 7]). The latter basis, up to an affine transformation of the variable x, is given by the orthogonal Gram polynomials. Without loss of generality, we assume that

$$x_j(N) = -1 + \frac{2j-1}{N} \tag{1.2}$$

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section May 6, 2019; accepted for publication (in revised form) July 29, 2020; published electronically DATE.
https://doi.org/10.1137/19M1259936

Funding: This work was supported by the Brazilian foundations FAPESP under grant 2016/09906-0, by CNPq under grant 306136/2017-1, by CAPES under grant 23038010617201/33-4, and a research leave from IFMG.

[†]Departamento de Matemática, IBILCE, Universidade Estadual Paulista, 15054-000 São José do Rio Preto, SP, Brazil (d_k_dimitrov@yahoo.com).

[‡]Departamento de Matemática, Instituto Federal de Minas Gerais, 36415-000 Congonhas, MG, Brazil (lourenco.peixoto@ifmg.edu.br).

and define the inner product

$$\langle f, g\rangle_N = \frac{1}{N}\sum_{j=1}^{N} f(x_j(N))\, g(x_j(N)), \tag{1.3}$$

which, in turn, induces the norm

$$\|f\|_N = \left(\frac{1}{N}\sum_{j=1}^{N} f^2(x_j(N))\right)^{1/2}. \tag{1.4}$$

The polynomials $\mathcal{G}_k(x;N)$, $k = 0,\dots,N-1$, orthonormal with respect to the above inner product, are known as the discrete Chebyshev polynomials, or even more commonly, the Gram polynomials [6, p. 113]. They are expressed in terms of a hypergeometric function as (see [1, Chapter 2])

$$\begin{aligned}\mathcal{G}_k(x;N) &= (-1)^k \sqrt{\frac{(2k+1)(N-k)_k}{(N+1)_k}}\; {}_3F_2\left(\begin{matrix} -k, k+1, (1-N-Nx)/2 \\ 1, 1-N \end{matrix}\;\middle|\; 1\right) \\ &\quad (-1)^k \sqrt{\frac{(2k+1)(N-k)_k}{(N+1)_k}} \sum_{\nu=0}^{k} \frac{(-k)_\nu\,(k+1)_\nu\,((1-N-Nx)/2)_\nu}{(\nu!)^2\,(1-N)_\nu},\end{aligned} \tag{1.5}$$

where the Pochhammer symbol is defined by $(A)_\nu = A(A+1)\cdot\dots\cdot(A+\nu-1)$, $\nu \geq 1$, and $(A)_0 := 1$. Using the above notation, the solution of the CLSAP is explicitly written in the form

$$p_n(x;f) = \sum_{k=0}^{n} a_k\, \mathcal{G}_k(x;N), \tag{1.6}$$

where the Fourier coefficients a_k are given by

$$a_k = \langle f, \mathcal{G}_k(\cdot\,;N)\rangle_N = \frac{1}{N}\sum_{j=1}^{N} f(x_j(N))\; \mathcal{G}_k(x_j(N);N). \tag{1.7}$$

The representation of the solution $p_n(x;f)$ via (1.6) and (1.7) is well known (see Chapter 7 of Hildebrand's book [21]). The idea of constructing an efficient algorithm for the CLASP by calculating first the coefficients a_k from (1.7) and then the solution $p_n(x;f)$ from (1.6) was developed recently in [2, 3]. It is especially efficient in the most natural situation when the degree n of $p_n(x;f)$ is much smaller than N. The algorithm developed in [3] is reasonably efficient. However, the emphasis in [3] is on the analysis of the algorithm from a theoretical point of view. Furthermore, the computations in [3] are done in Mathematica with variable precision. In the present work we suggest various improvements of the algorithm from [3] and build and develop a new one, which can be easily implemented in a large variety of programming languages because we work with double precision. Our routines are written in MATLAB.

The sum (1.7) usually contains a very large number of terms, so [2, 3] used a "Gaussian type quadrature formula" of the form

$$\frac{1}{N}\sum_{j=1}^{N} F(x_j) \approx \sum_{k=1}^{m} B_{m,k}\, F(g_{m,k}(N)) =: Q_m(F), \tag{1.8}$$

where the nodes $g_{m,k}(N)$ coincide with the zeros of $\mathcal{G}_m(x;N)$. However, there are significant challenges in developing an efficient algorithm for (1.8). Approximating the zeros of $\mathcal{G}_m(x;N)$ is a difficult task. It requires both precise estimates, obtained in [2], and proving the convergence of the Weierstrass–Dochev–Durand–Kerner (WDDK) method, established in [3].

In the present note we show that the WDDK method is extremely accurate in double precision. This is done in section 2, where we analyze and compare various methods for calculating the nodes as well as the weights of the quadrature formula.

Key computational issues concerning the construction of the approximating polynomial $p_n(x;f)$ itself are discussed in section 3. A special emphasis is given on a procedure aimed at minimizing the effects of the roundoff error.

In section 4 we report the results of a comparison between the new method and the following "standard" algorithm in MATLAB:[1]

```
function c = standard(x,vals,n)
N=numel(vals);
V=zeros(N, n+1);
V(:,1)=ones(N,1,class(x));
for j = 2:n+1
    V(:,j)=x.*V(:,j-1);
end
c = V\vals;
end
```

This algorithm, referred to as the STANDARD in the rest of the paper, constructs the Vandermonde matrix V of the mesh points $x_j(N)$ and then uses the operator \ in MATLAB to solve the CLSAP via the QR decomposition of V (see [6, section 5.7]). It turns out that STANDARD is a bit more efficient than the `polyfit()` command because lines 65–67 in `polyfit()` explicitly compute the Q and R factors before calling \ and there is also error checking as well as global calls to turn warnings on/off.

We use the following errors in the comparisons:

$$\text{maximum absolute error} = \max_{k=1,\ldots,m} \left| v_k - v_k^{\text{true}} \right|,$$

$$\text{maximum relative error} = \max_{k=1,\ldots,m} \left| \frac{v_k - v_k^{\text{true}}}{v_k^{\text{true}}} \right|,$$

$$\text{relative maximum error} = \frac{\max_k |v_k - v_k^{\text{true}}|}{\max_k |v_k^{\text{true}}|}.$$

2. The Gaussian type quadrature for large sums.

We briefly review the ideas in [2, 3] concerning the above-mentioned Gauss type quadrature rule (1.8). It is exact for all $F \in \mathbb{P}_{2m-1}$ and its nodes coincide with the zeros $g_{m,k}(N)$ of the orthonormal polynomial $\mathcal{G}_m(x;N)$. Let us emphasize that in [3] the number of nodes of (1.8) was chosen to be equal to the degree n of the polynomial best approximation while here we analyze the more general situation when (1.8) possesses m nodes and then we choose m appropriately.

It was proved in [3] that all the zeros $g_{m,k}(N)$, $k = 1,\ldots,m$, can be calculated simultaneously using the WDDK method [29, 15, 16, 23] with initial approximations being the nodes of the classical Gaussian quadrature formula, that is, the zeros of Legendre polynomials. Here we adopt a more straightforward approach and use as

[1]We thank one of the referees for suggesting this algorithm as the main competitor.

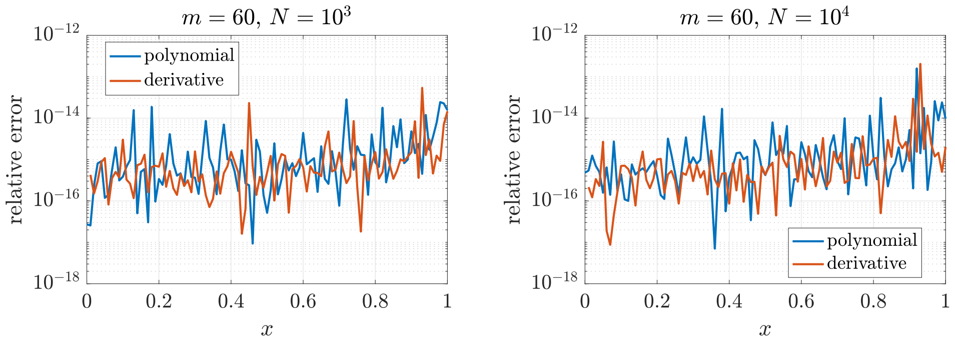

FIG. 1. *The errors of the calculation of Gram's polynomial and its derivative via the recurrence relations for $m = 60$ with $N = 10^3$ (left) and 10^4 (right).*

initial approximations for the WDDK method the explicit values suggested by Förster and Petras [17, Theorem 1]. It turns out that the choice of the WDDK method is appropriate because the resulting method is better than both the Newton–Raphson method with the same initial approximations and the celebrated Golub–Welsch (GW) algorithm [18], adapted for (1.8). Section 2.2 contains all the details concerning the algorithm for the zeros $g_{m,k}(N)$. The calculation of the weights $B_{m,k}$ is discussed in section 2.3, where we compare different ways to compute them. The quadrature is analyzed in section 2.4. All the methods for calculating the nodes and the weights of the Gaussian quadrature require calculating the values of $\mathcal{G}_m$ and $\mathcal{G}'_m$ with a high precision. We analyze such calculations in section 2.1.

2.1. Evaluation of the orthonormal Gram polynomials. It is known that the orthonormal Gram polynomials are generated by the following recurrence relation:

$$\begin{aligned} &\mathcal{G}_0(x;N) = 1, \quad \mathcal{G}_1(x;N) = 2\,\alpha_0\,x, \\ &\mathcal{G}_m(x;N) = 2\,\alpha_{m-1}\,x\,\mathcal{G}_{m-1}(x;N) - \frac{\alpha_{m-1}}{\alpha_{m-2}}\,\mathcal{G}_{m-2}(x;N), \quad m \ge 2, \end{aligned} \tag{2.1}$$

where

$$\alpha_{m-1} = \frac{N}{m}\left(\frac{m^2 - \frac{1}{4}}{N^2 - m^2}\right)^{\frac{1}{2}}, \quad m \ge 1. \tag{2.2}$$

It is easily obtained from the recurrence relation for the corresponding monic polynomials [2, 3]. Unlike the classical continuous orthogonal polynomials [10, 27], the Gram polynomials do not satisfy a straightforward relation with their derivatives. Instead, $\mathcal{G}'_m(x;N)$ is computed by differentiating the recurrence relation above:

$$\mathcal{G}'_m(x;N) = 2\alpha_{m-1}\left(\mathcal{G}_{m-1}(x;N) + x\mathcal{G}'_{m-1}(x;N)\right) - \frac{\alpha_{m-1}}{\alpha_{m-2}}\mathcal{G}'_{m-2}(x;N) \tag{2.3}$$

for $m \ge 2$ with $\mathcal{G}'_1(x;N) = 2\alpha_0$ and $\mathcal{G}'_0(x;N) = 0$.

Figure 1 shows the error in computing the orthonormal Gram polynomials and their derivatives via the recurrence relations (2.1) and (2.3). The graphs show that the results are precise within 14–16 significant digits, both for the values of the polynomial and its derivative, with a slight loss of accuracy near the end-point 1. That is why we calculate Gram's polynomials via the recurrence relation (2.1). We point out that the values of $\mathcal{G}_m$ and $\mathcal{G}'_m$ in [3] were obtained via a version of Horner's algorithm [3,

TABLE 1
Values of $\|I - QQ^T\|$. *Orthonormality does hold when the orthonormal Gram polynomials are calculated via the recurrence relation* (2.1).

	$N = 10^3$	$N = 10^4$	$N = 10^6$
$m = 10$	4.4409e-16	1.3323e-15	5.5511e-15
$m = 60$	1.3087e-15	1.9984e-15	5.7732e-15
$m = 100$	8.2808e-15	4.4409e-15	8.1046e-15

TABLE 2
Number of iterations required by WDDK method for $m = \min\{\lfloor 2.5\sqrt{N}\rfloor, 100\}$.

N	250	500	750	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
# iterations	7	7	7	7	4	3	3	3	3	3	3

Algorithm 1]. As mentioned in the introduction, the calculations reported in [3] were performed using symbolic software which uses arbitrary-precision numbers. Unfortunately, the so modified Horner algorithm accumulates roundoff errors, especially close to one of the end-points ± 1, when the calculations are performed in double-precision floating-point numbers.

In order to test numerically the orthonormality of the Gram basis obtained in this manner, we calculate the values of $\|I - QQ^T\|$, where

$$Q = \frac{1}{\sqrt{N}} \begin{bmatrix} \mathcal{G}_0(x_1; N) & \dots & \mathcal{G}_0(x_N; N) \\ \vdots & \ddots & \vdots \\ \mathcal{G}_m(x_1; N) & \dots & \mathcal{G}_m(x_N; N) \end{bmatrix} \tag{2.4}$$

and the norm $\|\cdot\|$ is the largest absolute value of the entries of the corresponding matrix. See Table 1.

2.2. The nodes. The formulae for calculating the nodes of the quadrature via the iterative WDDK method are given by [3, eq. (3.1)]. Here we apply them with the orthonormal polynomials in place of the monic ones.

The convergence of the WDDK method for the zeros of $\mathcal{G}_m(x; N)$ was established rigorously in [3, Theorem 3.1] in the case when the initial approximations are the zeros of the Legendre polynomials $P_m(x)$. We have tested the convergence of the WDDK method with the initial conditions, used by Förster and Petras [17, Theorem 1] to approximate the zeros of $P_m(x)$. It converges when $m \leq \min\{\lfloor 2.5\sqrt{N}\rfloor, 100\}$ and N runs from 10 to 10^{10}. Existing results (see [5] and [6, p. 114]) recommend that $m \leq 2.5\sqrt{N}$. Moreover, for any fixed m, the zeros of $\mathcal{G}_m(x; N)$ converge to the zeros of $P_m(x)$, as $N \to \infty$, so those initial approximations are quite efficient indeed. Therefore, we use the initial approximations from [17, Theorem 1] for the WDDK method.

Table 2 shows the number of iterations required by WDDK. In all cases the zeros are obtained with high accuracy in double precision.

We have compared the WDDK method with the Newton–Raphson algorithm and with the classical method of Golub and Welsch [18]. The Newton–Raphson fails to calculate the zeros of the Gram polynomials even with the extremely precise initial approximations used in WDDK. We believe that the reasons that Newton–Raphson fails are the oscillations of the polynomial with high frequencies and large amplitudes near the endpoints $[-1, 1]$. Indeed, this phenomenon causes the tangent line to the graph of $\mathcal{G}_m(x; N)$ to cross the real line at a point which is too far from the zero

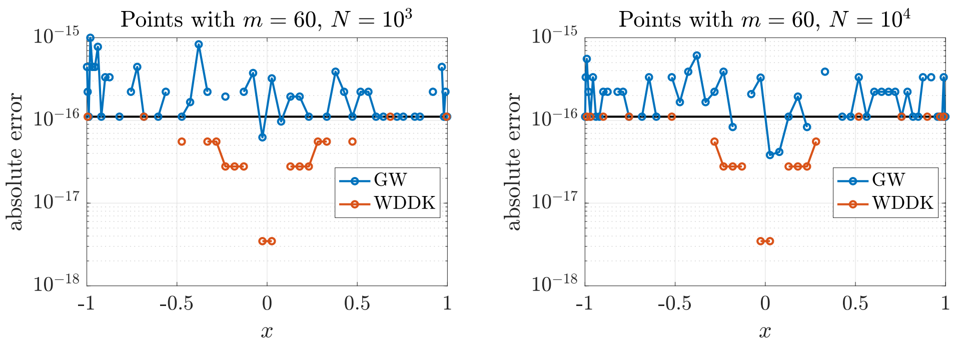

FIG. 2. *The absolute error in computing the zeros of* $\mathcal{G}_{60}(x;N)$ *for* $N = 10^3$ (*left*) *and* 10^4 (*right*), *by the GW algorithm and the WDDK method. The black line corresponds to the value* `eps/2`. *The points not exhibited correspond to values with all digits correct in double precision accuracy.*

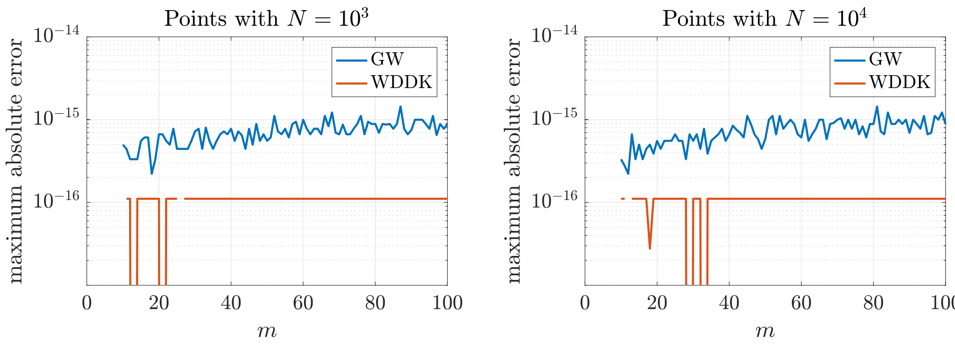

FIG. 3. *The maximum absolute errors in computing the zeros of* $\mathcal{G}_m(x;N)$ *for* $N = 10^3$ (*left*) *and* 10^4 (*right*), *by the version of the GW algorithm, described above, and the WDDK method. The gaps in the red line correspond to values which are correct in double precision accuracy.*

which is supposed to be approximated. Therefore, the further iterations obtained by the method may converge to another zero.

In order to compare our algorithm with GW, adapted for the Gram polynomials, we run the latter one, implemented in a routine for FORTRAN [19]. We do so because the analysis of Hale and Townsend [20] shows that the implementation of GW in MATLAB has a high complexity. In all of our comparisons, the exact values of $g_{m,k}(N)$ and $B_{m,k}$ are considered to be those computed by this version of GW but with quadruple precision.

Figure 2 shows the absolute errors of the results obtained by GW and WDDK for all zeros of the Gram polynomial of degree 60 with $N = 10^3$ and 10^4. The WDDK method computes all zeros with precision smaller than `eps/2` in MATLAB. Here, as usual, the floating-point relative accuracy `eps` is 2^{-52}.

Figure 3 demonstrates the maximum absolute error of the results for $N = 10^3$ and 10^4 and $10 \leq m \leq 100$. Observe that this error does not exceed `eps/2` for the WDDK method. Furthermore, as m grows, a slight and gradual increase occurs in the error of the GW method.

2.3. The weights. A common way to calculate the weights of the Gaussian quadrature is to use an explicit formula which involves the values of the corresponding orthogonal polynomials and their derivatives at the nodes [21, p. 390, eq. (8.4.17)]).

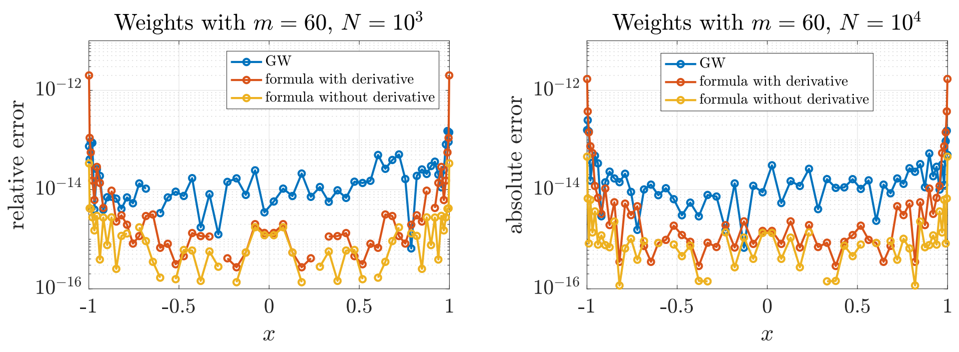

Fig. 4. *The relative error in computing the weights of $\mathcal{G}_{60}(x;N)$ for $N = 10^3$ (left) and 10^4 (right), by the GW algorithm and the formulae with and without derivative. The points not exhibited correspond to values which are precise to double precision accuracy.*

Applying it to (1.8), one obtains

$$B_{m,k} = \frac{2\alpha_{m-1}}{\mathcal{G}_{m-1}(g_{m,k}(N);N)\mathcal{G}'_m(g_{m,k}(N);N)}. \tag{2.5}$$

Another expression can be obtained using the Christoffel–Darboux formula for orthogonal polynomials, which, in our setting, reads as (see [21, p. 390, eq. (8.4.18)])

$$\frac{1}{B_{m,k}} = \sum_{j=0}^{m-1} \left(\mathcal{G}_j(g_{m,k}(N);N)\right)^2. \tag{2.6}$$

We emphasize that Lether [24] verified that the corresponding formula in the case of Gauss–Legendre quadrature is stable.

The graphs in Figure 4 show a comparison of the relative error in computing the weights by the adapted version of GW with our algorithm, which is based on (2.5) and (2.6), for $m = 60$ with $N = 10^3$ and 10^4. It shows that the calculation via formula (2.6), without derivatives, turns out to be more accurate than the others. Also, it reveals that the calculation of the weights which correspond to nodes close to the end points ± 1 is less precise, but again the use of the formula without derivatives provides the best results.

The maximum relative error and the relative maximum one, for $N = 10^3$ and $m = 10, 11, \ldots, 100$, are shown in Figure 5. Comparisons, carried out with other values of N, revealed a very similar trend for the weights. Therefore, we choose formula (2.6) to evaluate the weights.

2.4. The quadrature. Having discussed the best choices of methods for computing the nodes and the weights, we now address some additional issues concerning the quadrature formula (1.8). If F is given explicitly in the whole interval $[-1,1]$, then the values of F at $g_{m,k}(N)$ are known. Moreover, if F sufficiently smooth and the norms of its derivatives are uniformly bounded in $[-1,1]$, then (1.8) provides very precise results. Indeed, as was shown in [3], when $F \in C^{2m}[-1,1]$, the error of (1.8) depends on the norm of $F^{(2m)}(x)$. Here we illustrate that this feature of (1.8) persists in the present situation when we use double precision. As is seen in Figure 6, the error declines rapidly when m increases.

In the case when only the samples $F(x_j(N))$ are known, in order to apply (1.8) we need to approximate $F(g_{m,k}(N))$ in terms of the data $\{F(x_j(N))\}$. Smolyak's

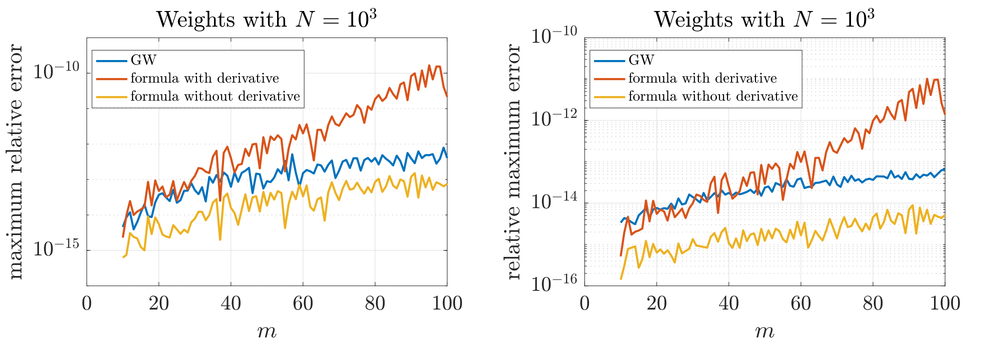

FIG. 5. *The maximum relative* (*left*) *and relative maximum* (*right*) *errors of calculating the weights for $N = 10^3$, by the GW algorithm and by formulae* (2.5) *and* (2.6)*.*

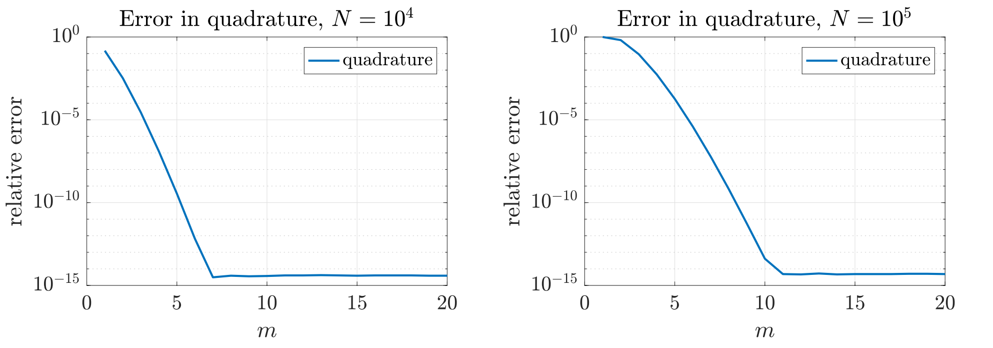

FIG. 6. *The relative error of the quadrature for $F(x_j) = e^{x_j}$* (*left*) *and $F(x_j) = x_j \sin 3x_j$* (*right*)*. The accuracy in double precision are attained.*

TABLE 3
Relative error in quadrature ($m = 30$) *using different methods to approximate $F(g_{m,k}(10^4))$ for the data $F(x_j) = e^{x_j} + 0.01\delta_{10^4}(j)$. Here δ_N =* `randn(1,N)` *returns a vector with N normally distributed pseudorandom numbers by* MATLAB.

Method	Error
Linear interpolant	8.9709e-04
Splines (4 pts.)	9.6824e-04
Splines (6 pts.)	1.0386e-03
Splines (8 pts.)	1.0550e-03
Splines (10 pts.)	1.0587e-03

lemma [26] in the theory of optimal recovery states that among all methods for recovery of a linear functional via a fixed set of data, there exists a linear method for recovery (see also [4] and [7, p. 75, Lemma 5.7]). In [3] each value $F(g_{m,k}(N))$ was approximated by a local natural cubic interpolating spline. Based on a large number of comparisons of various linear methods, we have concluded that the most efficient one is the linear interpolation, that is, the approximation by a piecewise linear function which interpolates F at the mesh points. We call this algorithm “the linear interpolant.” It adequately suits a large variety of data, including some generated almost randomly. Since it is impossible to report the entire work, we show in Table 3 only the comparison of the linear interpolant with local natural cubic spline approximation with $4, 6, 8,$ and 10 points, closest to $g_{m,k}(N)$. We observe that the

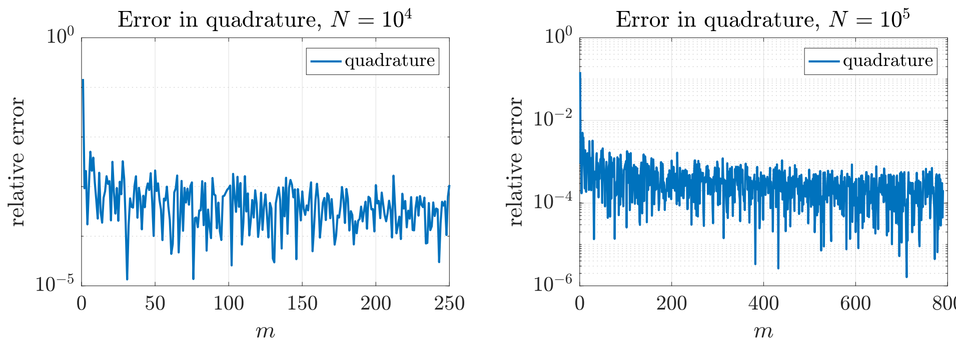

FIG. 7. *The relative error in quadrature for* $F(x_j) = e^{x_j} + 0.01\delta_{10^4}(j)$ *(left) and* $F(x_j) = e^{x_j} + 0.01\delta_{10^5}(j)$ *(right). The graphs show all values of* $m \leq \lfloor 2.5\sqrt{N} \rfloor$.

linear interpolant is of the least computational complexity because it uses only $4m$ additions, $2m$ multiplications, and m divisions while the spline approximation with s points uses $(14s + 4)m$ additions, $(5s + 3)$ multiplications, and $(3s + 3)m$ divisions.

Figure 7 shows the relative error in calculating the quadrature (1.8), performed for different values of m, for two sets of data, both involving random fluctuations. This kind of sample is notorious for being the worst type of data for any kind of approximation. Despite the fact that the quadrature is not accurate in double precision any more, its error is of order roughly 10^{-4} compared to 10^{-15}. Observe that the error plateaus out beyond a relatively small value of m. Because of these observations we choose to calculate initially the Fourier coefficients a_k, defined in (1.7) via the Gaussian quadrature (1.8) with $m = \min\{100, \lfloor 2.5\sqrt{N} \rfloor\}$ nodes.

In the next section we will discuss an approach for minimizing the roundoff error which arises in calculating the Taylor coefficients of the approximating polynomial $p_n(x; f)$. That approach requires calculating $\|f\|_N^2$, the square of the norm of the data, defined in (1.4), via the quadrature formula (1.8). Initially we calculate $Q_m(f^2)$, with $m = \min\{100, \lfloor 2.5\sqrt{N} \rfloor\}$ and the relative error

$$r_m = \frac{|Q_m(f^2) - Q_{m-5}(f^2)|}{Q_m(f^2)}.$$

If $r_m \leq 10^{-5}$, then we consider $Q_m(f^2)$ as a precise approximation of $\|f\|_N^2$. Otherwise, we increase m until either r_m reaches the limit 10^{-5} or m becomes $\min\{700, \lfloor 2.5\sqrt{N} \rfloor\}$.

Summarizing, the Gaussian type quadrature approximates very well sums with a very large number of terms for a vast variety of data that arise in the CLSAP. Despite some "pathological" exceptions such as the one in Example 2 in [3], where the calculation of the Riemann zeta function turns out to be incorrect, it is a key ingredient in constructing a very precise approximation of the polynomial $p_n(x; f)$, which solves CLSAP.

3. The least squares approximating polynomial.

There are two issues regarding the solution of CLSAP, namely, the evaluation of $p_n(x; f)$ at a point x and the computation of its Maclaurin coefficients. In both cases the results are obtained from the representation

$$p_n(x;f) = \sum_{k=0}^{n} a_k\,\mathcal{G}_k(x;N),$$

where the Fourier coefficients are obtained by the quadrature (1.8).

For calculating the value of $p_n(x;f)$ at a fixed point x, we invoke Clenshaw's algorithm [11, 13]. Since the Gram polynomials obey a three term recurrence relation (2.1), this algorithm is very suitable because $p_n(x;f)$ is a linear combination of Gram polynomials.

Calculating the Maclaurin coefficients of $p_n(x;f)$, that is, its representation in the monomial basis, requires an addition theoretical tool to avoid roundoff errors caused by multiplication. Indeed, formally we calculate $c_{n,k}$ in

$$p_n(x;f) = c_{n,n}x^n + c_{n,n-1}x^{n-1} + \cdots + c_{n,0} \tag{3.1}$$

in the following way. First we obtain the Maclaurin coefficients $b_{k,j}$ in the monomial expansion of the Gram polynomials

$$\mathcal{G}_k(x;N) = b_{k,k}x^k + b_{k,k-1}x^{k-1} + \cdots + b_{k,0}, \quad k = 0,\ldots,n, \tag{3.2}$$

form their representation

$$\mathcal{G}_k(x;N) = \alpha_{k-1}\left[2^k\prod_{j=0}^{k-2}\alpha_j\,x^k + \sum_{j=1}^{\lfloor k/2\rfloor}\left(2b_{k-1,k-1-2j} - \frac{1}{\alpha_{k-2}}b_{k-2,k-2j}\right)x^{k-2j}\right], \tag{3.3}$$

where $k \geq 2$, $\mathcal{G}_1(x;N) = 2\alpha_0 x$, $\mathcal{G}_0(x;N) = 1$ and α_j are given in (2.2). The formula (3.3) is easily obtained by induction and evaluating at zero successive derivatives of the recurrence relation (2.1). Tests that we have performed in MATLAB confirm that all coefficients obtained by (3.3) are accurately calculated in double precision.

Then (3.1) and (3.2) imply

$$p_n(x;f) = \sum_{k=0}^{n} a_k \sum_{j=0}^{k} b_{k,j}\,x^j = \sum_{j=0}^{n}\left(\sum_{k=j}^{n} a_k b_{k,j}\right)x^j,$$

which yields

$$c_{n,j} = \sum_{k=j}^{n} a_k b_{k,j}, \qquad j = 0,\ldots,n. \tag{3.4}$$

The roundoff error occurs exactly when one performs the latter calculations. Indeed, some of the coefficients $b_{k,j}$ are very large numbers, especially when the degree k of the Gram polynomial and the value of N themselves are large. For example, the leading coefficient $b_{30,30}$ of $\mathcal{G}_{30}(\cdot\,;10^5)$ is equal to `8.602406331394768e+08`. On the other hand, sometimes some of the Fourier coefficients a_k may be equal to zero, because of the nature of the problem. However, when they are calculated by the quadrature method, the numerical result is usually a very small nonzero number. That is why relatively large roundoff errors occur. A typical example is when the samples are the values at the mesh points $x_j(N)$ of an algebraic polynomial of a small degree, say, s, and one performs the algorithm to calculate the best least squares approximation

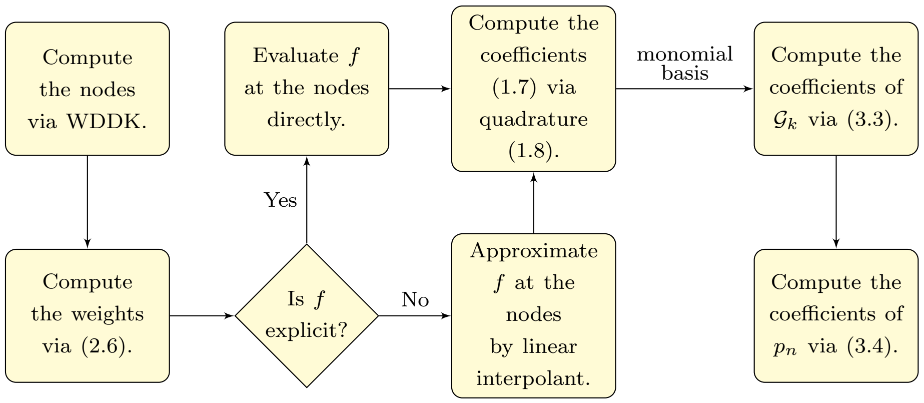
Compute the nodes via WDDK.
Evaluate f at the nodes directly.
Compute the coefficients (1.7) via quadrature (1.8).
monomial basis
Compute the coefficients of $\mathcal{G}_k$ via (3.3).
Compute the weights via (2.6).
Yes
Is f explicit?
No
Approximate f at the nodes by linear interpolant.
Compute the coefficients of p_n via (3.4).

FIG. 8. *The new algorithm to solve the CLSAP.*

$p_n(x; f)$ of degree n larger than s. Then the coefficients a_k for $k = s+1, \ldots, n$ must vanish but numerically the quadrature rule produces nonzero, though very small, numbers.

For this reason we perform a procedure which "recognizes" and discards very small Fourier coefficients. It is inspired and justified by the Pythagorean theorem, or equivalently, the Parseval identity for the ℓ^2 norm of the data (see [14, Chapter 1])

$$\|f\|_N^2 = \sum_{k=0}^{N-1} a_k^2,$$

the obvious fact that

$$\|p_n(\cdot; f)\|_N^2 = \sum_{k=0}^{n} a_k^2,$$

and the triangular inequality $\|f\|_N - \|p_n(\cdot; f)\|_N \leq \|f - p_n(\cdot; f)\|_N$. We choose a small ε, say, $\varepsilon = 5(r_m + 2\texttt{eps})$ and, if

$$\frac{|a_k|}{\|f\|_N} < \varepsilon$$

for some of a_k, $k = 0, \ldots, n$, we set $a_k = 0$.

Then we calculate the Maclaurin coefficients $c_{n,j}$ via (3.4).

The whole algorithm introduced in this paper is performed according to the flowchart in Figure 8.

4. Comparisons and conclusions. In this section we compare our algorithm, called the NEW one, with the STANDARD algorithm in MATLAB. The comparison is done when NEW calculates the polynomial $p_n(x; f)$ either expanded in the Gram basis as in (1.6), or in the basis of monomials, as in (3.1), while the STANDARD furnishes $p_n(x; f)$ only in the form (3.1).

First we compare the performance of the algorithms when the data is taken to be the values of an algebraic polynomial of a small degree. In Figures 9, 10, and 11 we demonstrate the results of this comparison when $f(x_j) = x_j^3 - \pi x_j^2 - 1$ for various values of n and N. Observe that the expansion of the polynomial $f(x) = x^3 - \pi x^2 - 1$

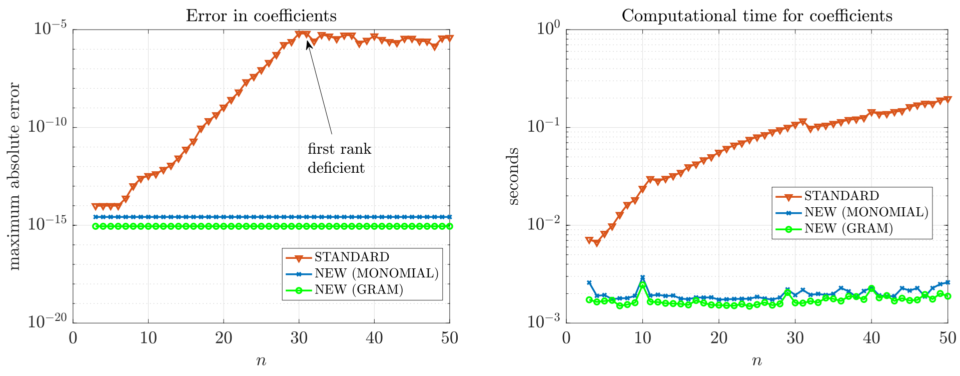

FIG. 9. *Maximum absolute error in the coefficients* (*left*) *and the computational time* (*right*) *when* $f(x_j) = x_j^3 - \pi x_j^2 - 1$ *with* $N = 10^5$.

in terms of the Gram basis, when $N = 10^5$, is

$$f(x) = a_0\, \mathcal{G}_0(x; 10^5) + a_1\, \mathcal{G}_1(x; 10^5) + a_2\, \mathcal{G}_2(x; 10^5) + a_3\, \mathcal{G}_3(x; 10^5),$$

where

$$a_0 = -\left(1 + \frac{3333333333\,\pi}{10000000000}\right), \qquad a_1 = \frac{29999999993\,\sqrt{3333333333}}{5000000000000000},$$
$$a_2 = -\frac{\pi\,\sqrt{138888888881944444445}}{12500000000}, \qquad a_3 = \frac{57\,\sqrt{10992393248472057341709537}}{1250000000000000}.$$

In Figure 9 we show the comparison for the coefficients when the samples coincide with the values of the latter polynomial, for a fixed value of N, namely, $N = 10^5$, and increasing values of n. The algorithm NEW is faster and has an accuracy of essentially double precision in both bases. The error of STANDARD in these examples increases and reaches a rank deficiency for $n \geq 31$.

In Figure 10 we compare the speed of the algorithms when N and n increase simultaneously in such a way that $n = \lfloor \sqrt{N}/10 \rfloor$. NEW is faster in both bases. STANDARD has a rank deficiency problem for $N \geq 10^5$.

In Figure 11 we compare the algorithms when $n = 7$ and N increases. NEW computes the coefficients with $\mathcal{O}(1)$ operations and is essentially accurate in double precision.

In Figure 12 we display the residual sum of squares, defined by

$$RSS = \|f - p_n(\cdot; f)\|^2 = \sum_{j=1}^{N} (f(x_j) - p_n(x_j; f))^2,$$

to compare the precision of the algorithms for the least squares approximation of samples that are the values at the mesh points of the smooth function $f(x) = \sin(15x)$ with $N = 5 \cdot 10^4$. We use Horner's algorithm [21, p. 28] to evaluate the polynomial provided both by STANDARD and by our method in monomial basis. NEW uses Clenshaw's algorithm. The methods have approximately the same precision for all values of n up to $n = 31$ when STANDARD begins to show rank deficiency. Despite the procedure to combat the roundoff error in the calculation of the Fourier coefficients in NEW, for some reason it appears for $35 \leq n \leq 40$ but for $n \geq 41$ the roundoff

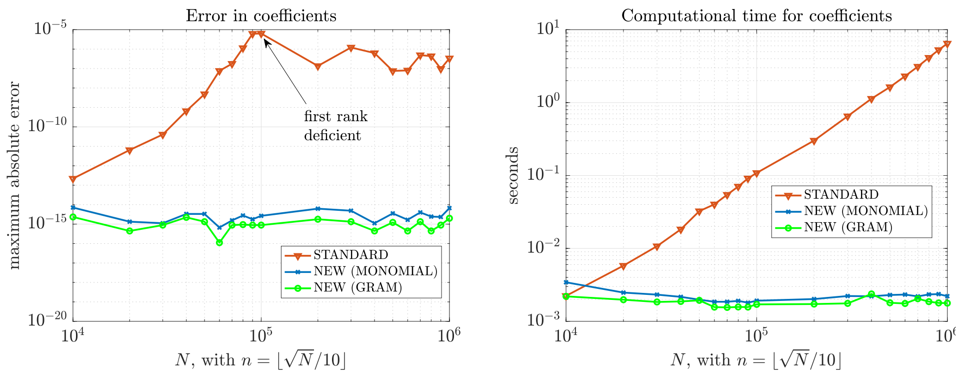

FIG. 10. *Maximum absolute error in the coefficients for the polynomial* $f(x_j) = x_j^3 - \pi x_j^2 - 1$ *with* $n = \lfloor\sqrt{N}/10\rfloor$ (*left*) *and the computational time* (*right*). *The same polynomials* f *in the Gram basis are easily computed and we omit them here.*

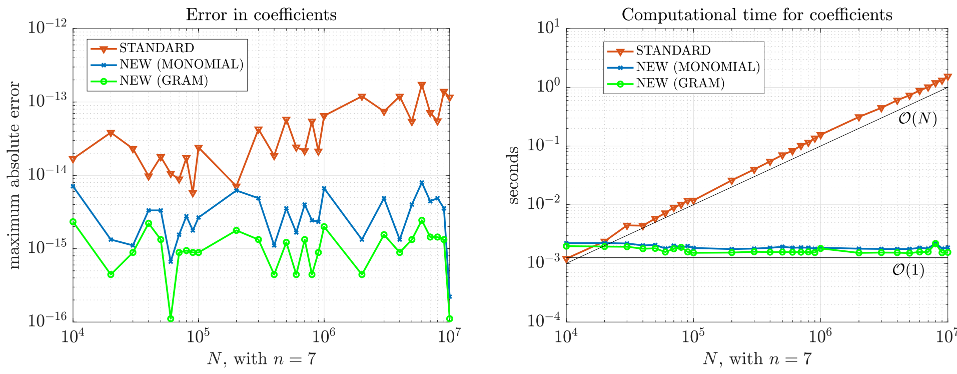

FIG. 11. *Maximum absolute error in the coefficients for* $f(x_j) = x_j^3 - \pi x_j^2 - 1$ *with* $n = 7$ (*left*) *and the computational time* (*right*).

error becomes negligible again. In any case, NEW is more precise than STANDARD for every $n \geq 32$, both in the monomial and the Gram bases. Our algorithm is also faster in each basis for every $n > 1$. We have observed that Horner's algorithm is faster than Clenshaw's when $N \geq 5 \cdot 10^4$.

In Figure 13 we display the residual sum of squares and the time to calculate the least squares approximations for slightly noisy samples with $N = 3 \cdot 10^4$. All the methods have around the same precision. Our algorithm is faster in both bases for $n \geq 5$.

In Tables 4 and 5 we compare STANDARD (ST) and NEW, with the use of the Clenshaw algorithm for the Gram basis, via the RSS and the time for evaluation for another set of noisy data, given by $f(x_j) = \cos(20x_j) + 10^{-8}\delta_N$, for various values of n e N. The tables show the abbreviation of the algorithm which performs better for the specific values of n and N while the dot symbol means that they perform equally. Further, NEW* means that the NEW algorithm solves the problem where STANDARD does not work at all. In these cases attempts to run the latter result in MATLAB messages like the following: "Requested 100000000x31 (23.1GB) array exceeds maximum array size preference. Creation of arrays greater than this limit may take a long time and cause MATLAB to become unresponsive."

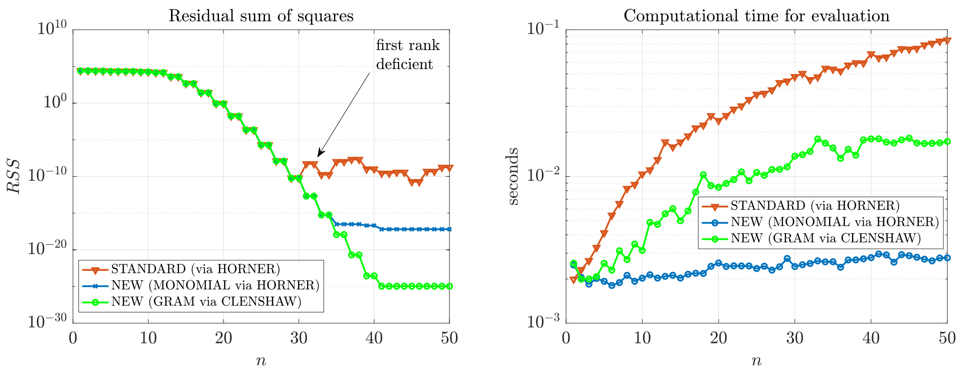

FIG. 12. *The residual sum of squares for the function $f(x_j) = \sin(15x_j)$ with $N = 5 \cdot 10^4$ (left) and the computation time for evaluation (right).*

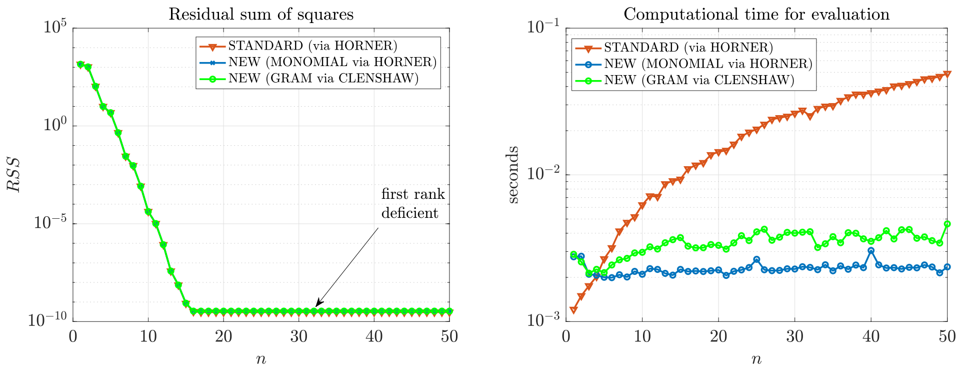

FIG. 13. *The residual sum of squares (left) and the computation time (right) for the best approximation of the samples $f(x_j) = e^{x_j^3} + 10^{-7}\delta_N(j)$ with $N = 3 \cdot 10^4$.*

TABLE 4
Comparison of RSS.

	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$	$N = 10^8$
$n = 5$	·	·	·	·	·	·
$n = 10$	·	·	·	·	·	NEW*
$n = 20$	·	·	·	·	·	NEW*
$n = 30$	ST	·	·	NEW	·	NEW*
$n = 40$	ST	ST	NEW	NEW	NEW	NEW*
$n = 50$	ST	ST	NEW	NEW	NEW	NEW*

TABLE 5
Comparison of the time for evaluation.

	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$	$N = 10^8$
$n = 5$	ST	ST	NEW	NEW	NEW	NEW
$n = 10$	ST	·	NEW	NEW	NEW	NEW*
$n = 20$	ST	NEW	NEW	NEW	NEW	NEW*
$n = 30$	ST	NEW	NEW	NEW	NEW	NEW*
$n = 40$	ST	NEW	NEW	NEW	NEW	NEW*
$n = 50$	ST	NEW	NEW	NEW	NEW	NEW*

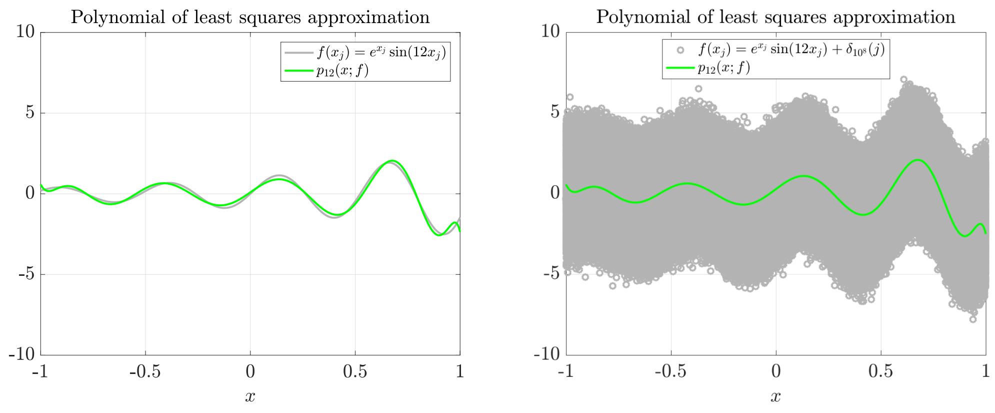

FIG. 14. *The approximate solutions of CLSAP of degree* 12*, with* $N = 10^8$*, for* $f(x_j) = e^{x_j}\sin 12\,x_j$ *(left) and the noisy data* $f(x_j) = e^{x_j}\sin 12x_j + \delta_{10^8}(j)$ *(right).*

Finally in Figure 14 we show our method of solving the CLSAP for a large number of samples, $N = 10^8$, both for an explicit function and its random perturbations. The STANDARD algorithm does not provide results because it exceeds the maximum array size preference in MATLAB. In the second case the best approximation was calculated via the Horner algorithm for the monomial basis because Clenshaw also exceeded maximum array size.

The latter examples of approximation of noisy samples confirm once again, though empirically, the observation of Demanet and Townsend [12] that "least squares polynomial fitting is robust to noisy samples."

We have performed a large number of experiments to compare the algorithms, both in terms of precision and time, in all possible situations, depending on the "smoothness" of the data, on the values of N and n, and on the basis we represent when calculated by NEW and the conclusion is as follows. When the data is taken from a process, described by a very smooth function, which is very common in practice, NEW is faster and more accurate, both in the Gram and the monomial bases. When the data shows some lack of smoothness or, even worse, the samples are noisy, that is, when random perturbations are introduced, STANDARD sometimes performs better, especially for small values of N, while NEW begins to produce more precise results and is much faster for large values of N (see Tables 4 and 5).

It is worth mentioning that in general NEW performs much better when the approximating polynomial $p_n(x;f)$ is represented in Gram's basis. This phenomenon is not surprising for various reasons. Trefethen [28, Chapter 3] expresses the general principal that "the monomial basis is familiar and comfortable, but you should never use it for numerical work with functions on an interval" and suggests the use of the basis of Chebyshev polynomials when one works with uniform approximation. Similarly, since the Gram polynomials are the natural orthonormal basis for the CLSAP, we do recommend their use here.

We emphasize also that, when n is fixed and N increases, our algorithm computes the coefficients in $\mathcal{O}(1)$ operations. Furthermore, the evaluation of the polynomial $p_n(x;f)$ is faster than Horner's algorithm used with STANDARD.

All numerical computations and the comparisons have been performed by MATLAB R2017b on a 2012 2.3 GHz Intel Core i7 MacBook Pro. The MATLAB codes are available at http://clsap.dcce.ibilce.unesp.br.

Acknowledgment. We thank the referees for their valuable and constructive remarks, which helped us to improve the initial version of the algorithm and the presentation significantly.

REFERENCES

[1] G. E. Andrews, R. Askey, and R. Roy, *Special Functions*, Cambridge University Press, Cambridge, 1999.

[2] I. Area, D. K. Dimitrov, E. Godoy, and V. G. Paschoa, *Approximate calculation of sums* I: *Bounds for the zeros of Gram polynomials*, SIAM J. Numer. Anal., 52 (2014), pp. 1867–1886.

[3] I. Area, D. K. Dimitrov, E. Godoy, and V. G. Paschoa, *Approximate calculation of sums* II: *Gaussian type quadrature*, SIAM J. Numer. Anal., 54 (2016), pp. 2210–2227.

[4] N. S. Bakhvalov, *On the optimality of linear methods for operator approximation in convex classes of functions*, Comput. Math. Math. Phys., 11 (1971), pp. 244–249.

[5] R. W. Barnard, G. Dahlquist, K. Pearce, L. Reichel, and K. C. Richards, *Gram polynomials and the Kummer function*, J. Approx. Theory, 94 (1998), pp. 128–143.

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

[7] B. D. Bojanov, H. Hakopian, and B. Sahakian, *Spline Functions and Multivariate Interpolations*, Springer, New York, 1993.

[8] J. P. Boyd, *Defeating the Runge phenomenon for equispaced polynomial interpolation via Tikhonov regularization*, Appl. Math. Lett., 5 (1992), pp. 57–59.

[9] J. P. Boyd and J. R. Ong, *Exponentially-convergent strategies for defeating the Runge phenomenon for the approximation of non-periodic functions, Part* II: *Multi-interval polynomial schemes and multidomain Chebyshev interpolation*, Appl. Numer. Math., 61 (2011), pp. 460–472.

[10] T. S. Chihara, *An Introduction to Orthogonal Polynomials*, Mathematics and its Applications, Gordon and Breach Science Publishers, New York, 1978.

[11] C. W. Clenshaw, *A note on the summation of Chebyshev series*, Math. Comp., 9 (1955), pp. 118–120.

[12] L. Demanet and A. Townsend, *Stable extrapolation of analytic functions*, Found. Comput. Math., 19 (2019), pp. 297–331.

[13] P. Deuflhard, *On algorithms for the summation of certain special functions*, Computing, 17 (1976), pp. 37–48.

[14] F. Deutsch, *Best Approximation in Inner Product Spaces*, Springer, New York, 2001.

[15] K. Dochev, *Modified Newton method for the simultaneous approximate calculation of all roots of a given algebraic equation*, Mat. Spis. Bulgar. Akad. Nauk, 5 (1962) pp. 136–139 (in Bulgarian).

[16] I. E. Durand, *Solutions Numérique des Équations Algébraiques. Tome* I: *Équations du Type $F(x) = 0$; Racines d'une Polynome,* Masson, Paris, 1960.

[17] K.-J. Förster and K. Petras, *Inequalities for the zeros of ultraspherical polynomials and Bessel functions*, ZAMM Z. Angew. Math. Mech., 73 (1993), pp. 232–236.

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

[19] E. Grosse, `gaussq.f` GO quadrature library, 1983, http://netlib.org/go/gaussq.f

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

[21] F. B. Hildebrand, *Introduction to Numerical Analysis*, 2nd ed., Dover, New York, 1987.

[22] P. Hoffman and K. C. Reddy, *Numerical differentiation by high order interpolation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 979–987.

[23] I. O. Kerner, *Ein Gesamtschrittverfahren zur Berechnung der Nullstellen von Polynomen*, Numer. Math. 8 (1966) pp. 290–294.

[24] F. G. Lether, *On the construction of Gauss-Legendre quadrature rules*, J. Comput. Appl. Math., 4 (1978), pp. 47–52.

[25] R. B. Platte, L. N. Trefethen, and A. B. J. Kuijlaars, *Impossibility of fast stable approximation of analytic functions from equispaced samples*, SIAM Rev., 53 (2011), pp. 308–318.

[26] S. A. Smolyak, *On the Optimal Recovery of Functions and Functionals of Them*, Candidate's dissertation, Moscow State University, Moscow, 1965 (in Russian).

[27] G. Szegő, *Orthogonal Polynomials*, 4th ed., American Mathematical Society, Colloq. Publ., Providence, RI, 1975.
[28] L. N. Trefethen, *Approximation Theory and Approximation Practice, Extended Edition*, SIAM, Philadelphia, 2019.
[29] K. Weierstrass, *Neuer Beweis des Satzes, dass jede ganze rationale Funktion einer Veränderlichen dargestellt werden kann als ein Product aus linearen Funktionen derstelben Veränderlichen*, Ges. Werke, 3 (1903) pp. 251–269.