SIAM J. SCI. COMPUT.
Vol. 24, No. 5, pp. 1465–1487

SPECTRAL DIFFERENCING WITH A TWIST*

RICHARD BALTENSPERGER† AND MANFRED R. TRUMMER‡

Abstract. Spectral collocation methods have become very useful in providing highly accurate solutions to differential equations. A straightforward implementation of these methods involves the use of spectral differentiation matrices. To obtain optimal accuracy these matrices must be computed carefully. We demonstrate that naive algorithms for computing these matrices suffer from severe loss of accuracy due to roundoff errors. Several improvements are analyzed and compared. A number of numerical examples are provided, demonstrating significant differences between the sensitivity of the forward problem and the inverse problem.

Key words. spectral collocation, spectral differentiation, roundoff errors, Chebyshev, Fourier and Legendre points

AMS subject classifications. 65D25, 65G50, 65M70, 65N35, 65F35

PII. S1064827501388182

1. Introduction.

Spectral collocation methods have become increasingly popular for solving differential equations. The unknown solution of the differential equation is approximated by a global interpolant, such as a polynomial or trigonometric polynomial of high degree. This global interpolant is then differentiated exactly, and the expansion coefficients are determined by requiring the equations to be satisfied at an appropriate number of collocation points. By contrast, finite difference and finite element methods use lower order local interpolants. Since interpolation, differentiation, and evaluation are all linear operations, the process of obtaining approximations to the values of the derivative of a function at the collocation points can be expressed as a matrix-vector multiplication; the matrices involved are called *spectral differentiation matrices*. The aim of this paper is to study the roundoff properties of these matrices.

The main advantage of spectral methods is their superior accuracy for problems whose solutions are sufficiently smooth functions. They converge exponentially fast compared to algebraic convergence rates for finite difference and finite element methods. In practice this means that good accuracy can be achieved with fairly coarse discretizations. Disadvantages are the appearance of full rather than sparse matrices, tighter stability restrictions, and less flexibility when dealing with irregular domains. For a thorough and enlightening treatment of the subject see the books [Boy, Chqz, Fun, Tre].

Several authors have studied roundoff error analysis of spectral differentiation matrices, mainly for Chebyshev differentiation matrices. In [Bre-Eve, Rot], the authors try to combat roundoff by preconditioning the problem. Don and Solomonoff in [Don-Sol1] and Tang and Trummer in [Tan-Tru] use trigonometric identities and a flipping trick to alleviate rounding errors. In [Don-Sol2], the authors propose a coordinate transform resulting in better stability property. A different approach is

*Received by the editors April 19, 2001; accepted for publication (in revised form) October 28, 2002; published electronically April 1, 2003. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant OGP0036901 and the Swiss National Science Foundation (SNSF) grant 81FR-57601.

http://www.siam.org/journals/sisc/24-5/38818.html

†Département de Mathématiques, Université de Fribourg, Pérolles, CH–1700 Fribourg, Suisse (richard.baltensperger@unifr.ch).

‡Department of Mathematics, Simon Fraser University, Burnaby, British Columbia V5A 1S6, Canada (trummer@sfu.ca).

suggested by [Bay-Cla-Mat] and [Bal-Ber, Bal-Ber2]. The diagonal of the matrix is modified to satisfy a relation among the entries of the differentiation matrix. (In many cases the row sums of a spectral differentiation matrix are all equal to zero.) This process is also applied in [Bal] to compute differentiation matrices based on arbitrary points.

There are a number of software packages implementing spectral methods. Among those are a Fortran package by Funaro, one by Don and Solomonoff (Pseudopack), a MATLAB package by Weideman and Reddy, and, recently, a (Fortran 90) package by Don and Costa (Pseudopack2000).[1] All of our computations are performed with MATLAB. Results may differ significantly when using other software. We have also observed variations when running our codes on different platforms (Intel/Pentium, SUN Ultra, SGI Indy, DEC Alpha).

2. Chebyshev differentiation.

2.1. Introduction. We approximate the first derivative of a function by interpolating the function with a polynomial at the Chebyshev–Gauss–Lobatto nodes

$$x_j := \cos\frac{\pi j}{N}, \qquad j = 0, 1, \dots, N, \tag{1}$$

differentiating the polynomial, and then evaluating the polynomial at the same collocation points. With $f_k := f(x_k)$ we have

$$p_N(x) := \sum_{k=0}^{N} f_k L_k(x) \tag{2}$$

for the interpolating polynomial, where L_k are the Lagrange interpolation polynomials,

$$L_k(x_j) = \begin{cases} 0 & \text{if } j \neq k, \\ 1 & \text{if } j = k. \end{cases}$$

Setting $\mathbf{f} := [f(x_0), \dots, f(x_N)]^T$ and $\mathbf{f'} := [f'(x_0), \dots, f'(x_N)]^T$, we approximate the derivative of f at the points x_j by differentiating and evaluating (2), i.e.,

$$\mathbf{f'} \approx D\mathbf{f}.$$

The entries of D are

$$D_{jk} = L_k'(x_j), \qquad j, k = 0, 1, \dots, N.$$

The first order Chebyshev differentiation matrix $D_C^{(1)} = D$ is then given by (see, e.g., [Chqz, Boy])

$$D_{kj} = \frac{c_j}{c_k}\frac{1}{x_k - x_j}, \qquad k \neq j, \tag{3}$$

$$D_{kk} = -\frac{x_k}{2(1 - x_k^2)}, \qquad k \neq 0, N, \tag{4}$$

$$D_{00} = -D_{NN} = \frac{2N^2 + 1}{6}, \tag{5}$$

where

$$c_k = (-1)^k, \quad k = 1, 2, \dots, N-1, \qquad c_0 = \frac{1}{2}, \quad c_N = \frac{(-1)^N}{2}. \tag{6}$$

Figure 1 depicts which formulas to use for which matrix entries.

[1] www.labma.ufrj.br/~bcosta/PseudoPack2000/Main.html

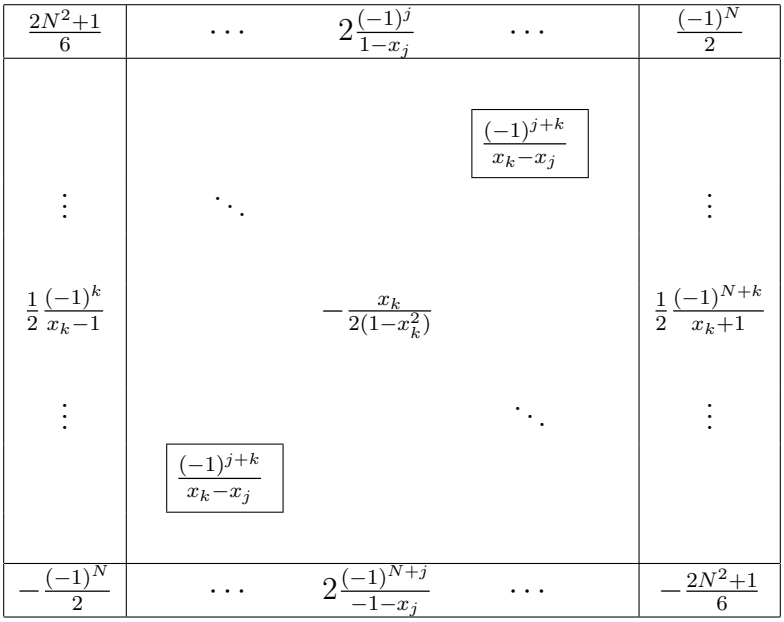

FIG. 1. *Chebyshev differentiation matrix.*

It is quite easy to see that for large N the direct implementation of (3)–(4) suffers from cancellation, causing large errors in the elements of the matrix D. This has been observed by various authors [Chqz, Bre-Eve, Rot, Bay-Cla-Mat, Don-Sol1, Don-Sol2, Tan-Tru, Bal-Ber, Bal-Ber2].

The spacing of Chebyshev nodes (1) near the boundary is $O(1/N^2)$, so like all sets of points giving good polynomial interpolation properties, they are more closely spaced near the endpoints of the interval. This spacing, while ensuring a small truncation error $||\mathbf{f}' - D\mathbf{f}||$, leads to larger roundoff errors. The largest errors occur in the upper left and lower right corner of the differentiation matrix D (see [Bre-Eve] and Figure 3). Figure 2 shows the magnitude of the elements of the spectral differentiation matrix (logarithmic scale). Next to it, Figure 3 shows a logarithmic plot of the (absolute) errors for the matrix elements. (The "exact differentiation matrix" was computed with 25 digit precision.) The peaks have become higher indicating a loss of relative precision.

2.2. Rounding error analysis. We investigate the effect of roundoff error on the largest element in the matrix D, namely D_{01}. We have

$$x_0 - x_1 = 1 - \cos\frac{\pi}{N} = \frac{\pi^2}{2N^2} + O\left(\frac{1}{N^4}\right); \tag{7}$$

hence,

$$D_{01} = 2\frac{-1}{1-x_1} = \frac{-2}{\frac{\pi^2}{2N^2}\left(1+O\left(\frac{1}{N^2}\right)\right)} = -\frac{4}{\pi^2}N^2\left(1-O\left(\frac{1}{N^2}\right)\right). \tag{8}$$

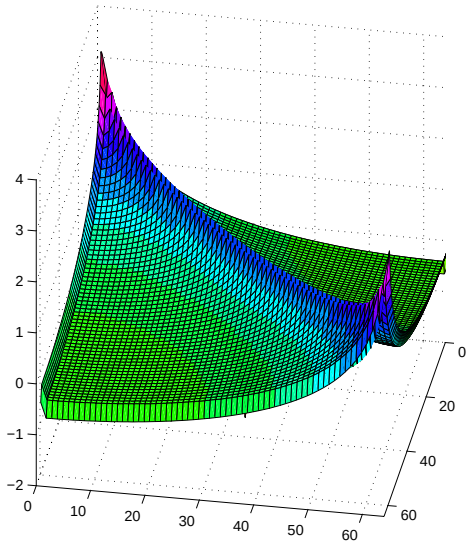

FIG. 2. *Chebyshev differentiation matrix* $N = 64$. *Logarithmic plot.*

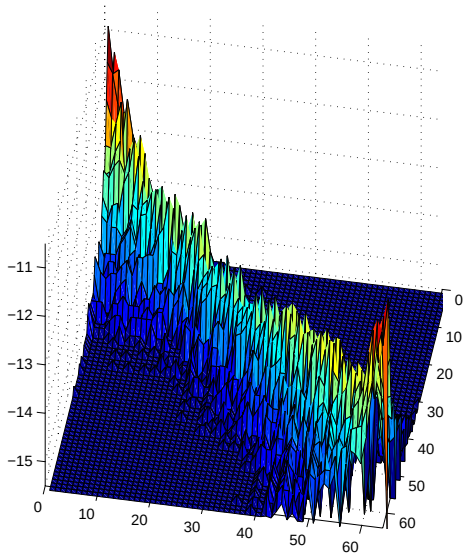

FIG. 3. *Errors in the Chebyshev differentiation matrix* $N = 64$. *Logarithmic plot.*

In finite precision arithmetic, however, we have

$$\tilde{x}_1 = x_1 + \delta, \tag{9}$$

where δ denotes a small (generic) error, with $|\delta|$ approximately equal to machine epsilon ϵ; the tilde denotes "computed" rather than exact quantities. Therefore,

$$x_0 \tilde{-} x_1 = x_0 - x_1 + \delta, \tag{10}$$

with absolute errors still being on the order of machine precision. But since $x_0 - x_1$ is small, this quantity has now a *large relative* error

$$\frac{2}{\pi^2} N^2 \delta,$$

caused by cancellation. Finally, the computed matrix element D_{01}, i.e.,

$$\frac{-2}{1 - x_1},$$

will have a relative error comparable to the one of $x_0 - x_1$, i.e., a relative error of size $\frac{2}{\pi^2} N^2 \delta$, hence combining with (8) we find

$$\tilde{D}_{01} - D_{01} = \frac{8}{\pi^4} N^4 \delta. \tag{11}$$

Table 1 lists the computed errors $\tilde{D}_{01} - D_{01}$.

Assume that all D_{kj} are computed to machine precision, i.e., with a (*relative*) error δ. In each row of D we have at most $O(1)$ entries of size $O(N^2)$,[2] all other

[2]These large elements occur only in the first few and last few rows of D.

TABLE 1
Computed errors in $\tilde{D}_{01} - D_{01}$.

N	$\tilde{D}_{01} - D_{01}$	$\frac{(\tilde{D}_{01} - D_{01})\pi^4}{\epsilon N^4}$
8	8.53e-014	9.13
16	-1.56e-013	-1.05
32	-2.44e-012	-1.02
64	-6.82e-012	-0.18
128	-1.10e-009	-1.80
256	-1.05e-009	-0.11
512	-5.13e-008	-0.33
1024	7.96e-007	0.32
2048	3.00e-005	0.75
4096	-6.16e-004	-0.96
8192	2.91e-003	0.28
machine epsilon $= \epsilon \approx$ 2.22e-016		

elements are of size $O(N)$ or $O(1)$. Thus, the absolute roundoff errors accumulate to no more than $O(N^2\delta)$ for all components of $D\mathbf{f}$. (A more careful analysis shows smaller roundoff error for all but the first few and last few components of $D\mathbf{f}$.) If, however, the differentiation matrix is computed via (3) and (4), we have relation (11), i.e., an absolute error of $O(N^4\delta)$ instead of the optimal $O(N^2\delta)$, leading to $O(N^4\delta)$ errors in $D\mathbf{f}$ near the boundary.

2.3. Improving accuracy. It is interesting to note that for smooth functions f which vanish at the boundary, roundoff error is much smaller. In this case, the badly computed matrix entry $\tilde{D}_{01}$ is multiplied by a small number,

$$f_1 \approx -\frac{1}{2} f'(x_0)\pi^2/N^2,$$

and therefore the error contributed to $(D\mathbf{f})_0$ is only

$$(\tilde{D}_{01} - D_{01})\tilde{f}_1 = O(N^2\delta), \tag{12}$$

alleviating the roundoff error effects on $D\mathbf{f}$ considerably.

This has been observed computationally and can be exploited by "preconditioning" (see [Rot], as well as [Bre-Eve, Don-Sol1, Bal-Ber, Bal-Ber2]) the problem: For example, with an arbitrary function f associate the function

$$\hat{f}(x) = f(x) - \left(\frac{1}{2}(f(1) - f(-1))x + \frac{1}{2}(f(1) + f(-1))\right);$$

then, approximate $\mathbf{f}'$ by

$$\mathbf{f}' \approx D\hat{\mathbf{f}} + \frac{1}{2}(f(1) - f(-1)).$$

This preconditioning, although effective, is perhaps not a very desirable way of dealing with the roundoff error problem. It may be easy to do in some applications of spectral differentiation matrices, but would prove very awkward at best in many others. What follows is a list of other fixes to the problem which have been proposed in the literature:

1. *Trigonometric identities.* We can replace (3) and (4) using trigonometric identities with the formulas

$$D_{kj} = \frac{c_j}{c_k} \frac{1}{2\sin\left((k+j)\pi/(2N)\right)\sin\left((k-j)\pi/(2N)\right)}, \qquad k \neq j, \tag{13}$$

$$D_{kk} = -\frac{x_k}{2\sin^2(k\pi/N)}, \qquad k \neq 0, N. \tag{14}$$

These formulas, proposed in [Chqz, Don-Sol1, Tan-Tru], improve the accuracy in the upper left corner of the matrix but not the lower right corner. The reason for this phenomenon is, that for small values of x, $\sin(\pi - x)$ cannot be computed nearly as accurately (i.e., with the same *relative* precision) as $\sin(x)$ (see [Don-Sol1] and section 3). Indeed, the relative error of $\sin(\pi - x)$ is $O(\delta/x)$, so, for example, when $x = x_{N-1}$, the relative error is $O(N^2\delta)$, and there is no significant gain in accuracy compared to the standard formulas.

2. *Flipping trick.* To avoid computing the sine function of arguments close to π one can take advantage of the symmetry property

$$D_{N-k,N-j} = -D_{kj}. \tag{15}$$

Compute the *upper half* of the matrix D, and then "flip" the matrix (see [Don-Sol1]). Alternatively, as suggested in [Tan-Tru], one can use formulas (13) and (14) to find the *upper left triangle* of D (i.e., compute D_{kj} with $k + j \leq N$) and then use relation (15) for the other elements. The latter method requires fewer evaluations of the sine function.

3. *Negative sum trick (NST).* Our main interest is not the accurate computation of the spectral differentiation matrix D per se, but having $D\mathbf{f}$ approximate $\mathbf{f}'$ as well as possible for a certain set of functions f. Differentiating the constant function $f(x) \equiv 1$ gives zero; therefore

$$\sum_{k=0}^{N} L_k'(x_j) = 0, \qquad 0 \leq j \leq N,$$

or, denoting by $\mathbf{1}$ the vector with each component equal to 1,

$$D\mathbf{1} = \mathbf{0} \qquad \Leftrightarrow \qquad \sum_{j=0}^{N} D_{kj} = 0, \qquad 0 \leq k \leq N. \tag{16}$$

After computing all the off-diagonal elements we can then use the formula

$$D_{kk} = -\sum_{\substack{j=0 \\ j\neq k}}^{N} D_{kj} \tag{17}$$

to compute the diagonal elements. It is worthwhile to note that this formula will actually produce less accurate entries on the diagonal, but, as we shall see, it gives good approximations $D\mathbf{f}$. This trick was proposed by [Bay-Cla-Mat] and [Bal-Ber, Bal-Ber2]. Note that for best results the sums in (17) must be computed carefully; i.e., we must sum the smaller elements first, to avoid smearing (see [Hen]).

TABLE 2
Errors $||D\mathbf{f} - \mathbf{f}'||$ *for various implementations of Chebyshev differentiation,* $f(x) = x^8$.

N	Original D	FFT	Trig/flip (WR)	NST only
16	8.88e-014	2.13e-014	7.11e-015	3.55e-015
32	5.85e-012	3.14e-013	2.27e-013	1.33e-014
50	2.74e-012	7.28e-013	2.41e-013	2.40e-014
64	2.36e-011	2.61e-013	1.02e-012	1.08e-013
100	3.54e-010	3.47e-011	2.73e-012	2.27e-013
128	6.43e-010	3.10e-013	1.82e-012	9.09e-013
250	2.00e-009	8.55e-011	4.79e-011	3.64e-012
256	1.39e-008	1.77e-011	1.68e-011	2.86e-012
500	7.32e-008	1.65e-010	4.00e-011	1.46e-011
512	1.78e-007	3.50e-011	2.91e-011	1.66e-011
1000	3.96e-006	2.97e-008	1.04e-009	1.16e-010
1024	2.02e-006	6.85e-011	1.46e-010	4.27e-011
2000	1.73e-005	9.44e-008	2.44e-009	3.26e-010
2048	4.53e-005	6.31e-010	2.89e-010	3.18e-010

The above tricks can be combined. For example, the routine `chebdif.m` of Weideman and Reddy [Wei-Red] uses all the tricks mentioned above when computing the Chebyshev differentiation matrix. Moreover, the nodes themselves are computed by $x_j = \sin\left(\pi(N-2j)/(2N)\right),\ j = 0, 1, \ldots, N$, preserving the symmetry about the origin. It should be noted that formula (1) gives less accurate values for j close to N, i.e., for x_j near -1. Perfect symmetry and more accurate values for the Chebyshev nodes can also be achieved by computing the nodes $x_j > 0$ and reflecting these to obtain the nodes $x_j < 0$.

A somewhat different approach is presented in [Don-Sol2]. A coordinate transform first proposed in [Kos-Tal] is employed which spaces the interpolation points farther apart at the boundary, resulting in better stability properties of the ensuing spectral method, while at the same time exhibiting less severe roundoff error.

We should also point out that $D\mathbf{f}$ can be computed without computing the matrix D. The FFT (fast Fourier transform) can be used to compute the coefficients in the Chebyshev expansion from the function values f_k and vice versa. Differentiation of the Chebyshev expansion can be accomplished via a simple recursion formula for the coefficients. The cost of obtaining $D\mathbf{f}$ from f is $O(N \log N)$, resulting in an asymptotically faster algorithm.

2.4. Negative sum trick.

2.4.1. Understanding the NST. Intuitively, one would suspect that computing the spectral differentiation matrix D in the most accurate way (for example, as programmed in [Wei-Red]) should lead to the best numerical results. It therefore comes as a surprise that simply using the original formula (3) *and* the NST (17) gives consistently the best results (see Table 2 and Figures 4, 5, and 6). When looking at the function $f(x) = x^8$ all the errors observed are due to roundoff (hence we have increasing errors with N). The second function $f(x) = \sin(8x)/(x+1.1)^{3/2}$ is more typical, as we can at first observe the exponential rate of convergence for the spectral approximation, whereas for larger N roundoff error is again dominant. Note that when N is not a power of 2, then the FFT approach produces much less accurate results. This is not as surprising as the fact that the same holds for the case where trigonometric identities, the flipping trick, and the NST (Weideman–Reddy code) are used. Using the NST on the original matrix gives a much smoother error curve. Most

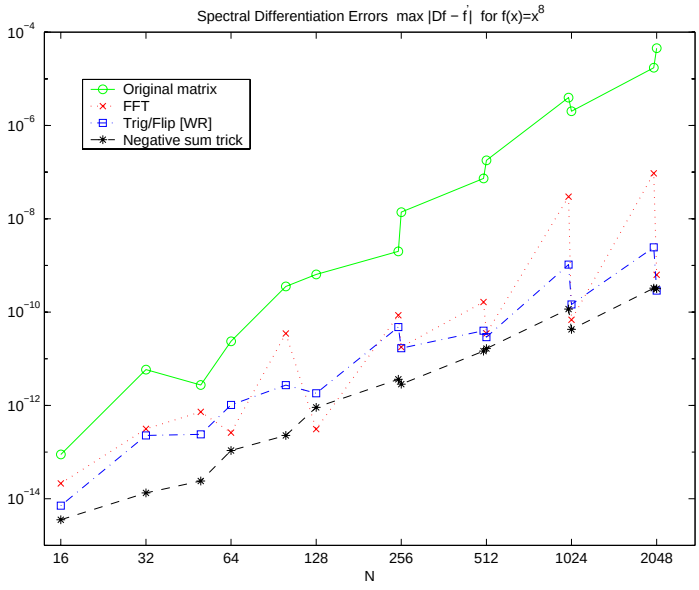

FIG. 4. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = x^8$.

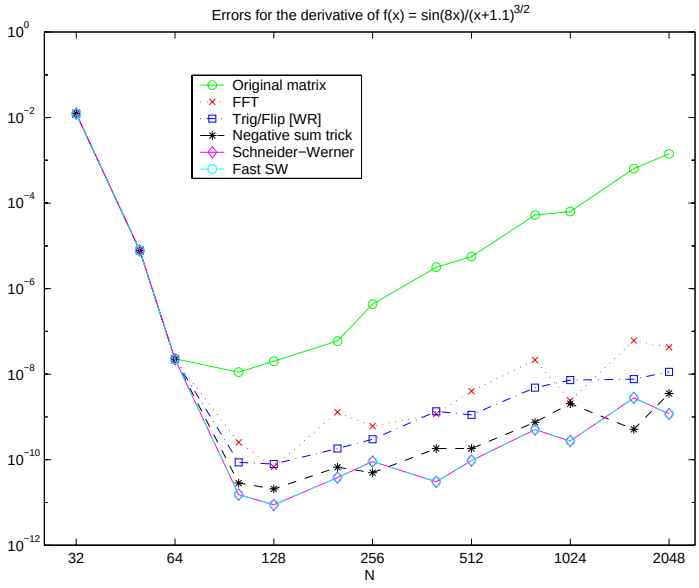

FIG. 5. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \frac{\sin 8x}{(x+1.1)^{3/2}}$.

of our computations are performed in MATLAB on a Pentium III, where the FFT in MATLAB produces much more accurate results than on other platforms we have tested (SUN Ultra, SGI Indy, DEC Alpha). This may be due to the 80-bit registers on the Intel/Pentium machine. Our version of MATLAB makes use of LAPACK and BLAS routines (older versions of MATLAB are based on LINPACK).

We now explain why applying the NST to the original formulas gives superior

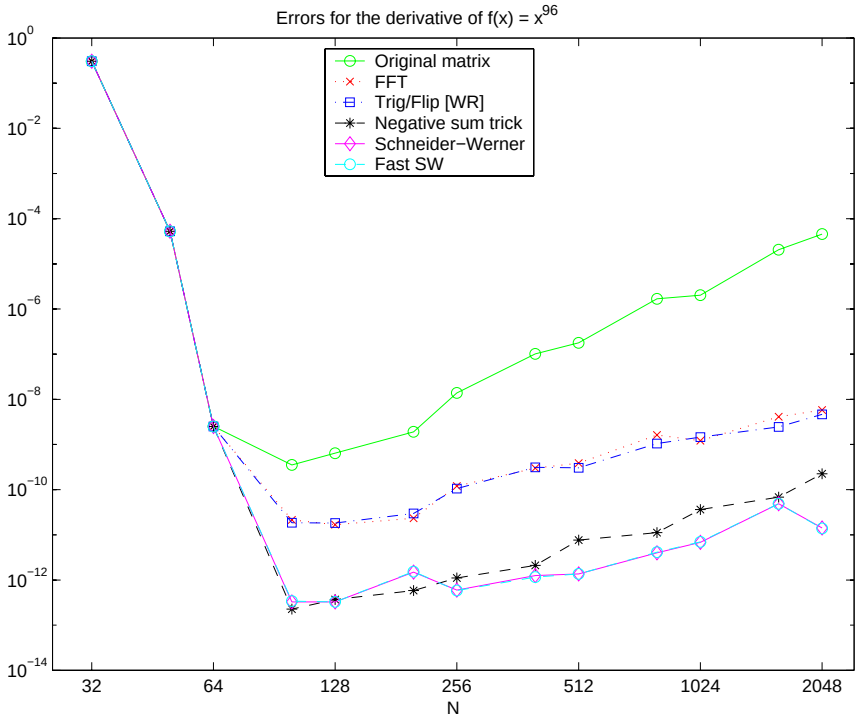

FIG. 6. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \frac{\sin 8x}{(x+1.1)^{3/2}}$.

results.

Denote by $D = D_C^{(1)}$ the exact Chebyshev differentiation matrix and by $\tilde{D}$ the computed matrix to which the NST has been applied. Similarly, $\mathbf{f}$ denotes the exact function values, whereas $\tilde{\mathbf{f}}$ refers to the computed values. For now we ignore the error of the well conditioned matrix-vector multiplication; i.e., we assume $\widetilde{\tilde{D}\tilde{\mathbf{f}}} = \tilde{D}\tilde{\mathbf{f}}$. The error

$$\tilde{D}\tilde{\mathbf{f}} - D\mathbf{f} = (\tilde{D} - D)\tilde{\mathbf{f}} + D(\tilde{\mathbf{f}} - \mathbf{f})$$

consists of two terms. It is clear that even with the exact differentiation matrix, i.e., $\tilde{D} = D$, we would still have an error due to errors in $\mathbf{f}$. We investigate the first of the two terms, $(\tilde{D} - D)\tilde{\mathbf{f}}$, in more detail. Note that

$$\tilde{D}_{kk} - D_{kk} = -\sum_{\substack{j=0 \\ j\neq k}}^{N} \left(\tilde{D}_{kj} - D_{kj}\right). \tag{18}$$

We therefore find

$$\begin{aligned}((\tilde{D}-D)\mathbf{f})_k &= \sum_{j=0}^{N}(\tilde{D}_{kj}-D_{kj})f_j = \sum_{\substack{j=0\\ j\neq k}}^{N}(\tilde{D}_{kj}-D_{kj})f_j + (\tilde{D}_{kk}-D_{kk})f_k \\ &= \sum_{\substack{j=0\\ j\neq k}}^{N}(\tilde{D}_{kj}-D_{kj})f_j - \sum_{\substack{j=0\\ j\neq k}}^{N}(\tilde{D}_{kj}-D_{kj})f_k \qquad (19)\\ &= \sum_{\substack{j=0\\ j\neq k}}^{N}(\tilde{D}_{kj}-D_{kj})(f_j-f_k).\end{aligned}$$

Equation (19) shows that whenever the matrix element D_{kj} has a large absolute error, it is multiplied by a small number. For example, just as in (12) the $O(N^4\delta)$ error in D_{01} is multiplied by $f_1 - f_0$ which is $O(1/N^2)$, contributing only an $O(N^2\delta)$ error term to the overall error compared to an $O(N^4\delta)$ term for the original matrix (without NST).

2.4.2. NST and Schneider–Werner formula. In 1986 Schneider and Werner presented a number of results [Sch-Wer] about rational interpolation in barycentric form. Specifically, they gave algorithms to evaluate the rational interpolant and its derivatives. The barycentric form of a rational (or, as a special case, *polynomial*) interpolant is given by

$$r(x) = \frac{\displaystyle\sum_{k=0}^{N}\frac{w_k f_k}{x-x_k}}{\displaystyle\sum_{k=0}^{N}\frac{w_k}{x-x_k}}. \qquad (20)$$

The x_k are the nodes of the interpolation (collocation points), the w_k are the weights, and the f_k are the values at the nodes which are to be interpolated between the nodes by the function $r(x)$. The function $r(x)$ has the interpolating property $r(x_k) = f_k$ for any choice of the w_k (which are only determined up to a constant factor). A particular choice of w_k gives rational functions with specific properties. For example, one may wish to have a rational function without poles on the interval $[x_0, x_N]$, which can be guaranteed as long as the w_k have alternating signs. For special choices of the weights, the function $r(x)$ becomes a polynomial, and (20) is the barycentric form of the interpolating polynomial; see, e.g., [Hen]. For the Chebyshev–Gauss–Lobatto nodes defined in (1) and weights $w_k = c_k$ defined in (6), formula (20) represents precisely the interpolating polynomial. Applying the Schneider–Werner formula (i.e., evaluating $r'(x_k)$) gives

$$-\sum_{\substack{j=0\\ j\neq k}}^{N}\frac{c_j}{c_k}\frac{f_j-f_k}{x_j-x_k} = \sum_{\substack{j=0\\ j\neq k}}^{N}D_{kj}(f_j-f_k). \qquad (21)$$

Due to the very nice numerical properties of the barycentric formula, formula (21) gives almost always the most precise answers. (The differentiation is, however, not accomplished by a matrix-vector multiplication, and the Schneider–Werner formula

does not directly produce a differentiation matrix.) It shows very nicely that the derivative of f is approximated by a weighted sum of divided differences. When $x_j - x_k$ is small, then so is $f_j - f_k$, giving formula (21) its good stability properties. Figures 5 and 6 show typical results.

It is now easy to see that the NST is a rearrangement of the Schneider–Werner formula—mathematically equivalent, albeit not numerically:

$$\underbrace{\sum_{\substack{j=0\\ j\neq k}}^{N} D_{kj}(f_j - f_k)}_{\text{Schneider–Werner}} = \underbrace{\left(-\sum_{\substack{j=0\\ j\neq k}}^{N} D_{kj}\right) f_k + \sum_{\substack{j=0\\ j\neq k}}^{N} D_{kj} f_j}_{\text{NST}}. \tag{22}$$

This equivalence provides an explanation for the superior accuracy of the NST. One would, however, suspect that applying the NST to a more accurately computed matrix should produce even better results than applying NST simply to the original formula (3). The next section tries to explain why this is not true.

2.4.3. Cancellation of rounding errors. In exact arithmetic, $D\mathbf{f}$ gives the exact values of the derivative of f, whenever f is a polynomial of degree not more than N,

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N. \tag{23}$$

NST makes sure that the constant term is differentiated exactly. What happens to the other terms in (23), i.e., to monomials $f(x) = x^\ell (\ell > 0)$?

For $f(x) = x^\ell$, we have $f_j = \tilde{x}_j^\ell$, and

$$f_j - f_k = \tilde{x_j}^\ell - \tilde{x}_k^\ell = (\tilde{x}_j - \tilde{x}_k)(\tilde{x}_j^{\ell-1} + \tilde{x}_j^{\ell-2}\tilde{x}_k + \cdots + \tilde{x}_j \tilde{x}_k^{\ell-2} + \tilde{x}_k^{\ell-1}).$$

If we apply the matrix $\tilde{D}$ to such a monomial, we obtain, just like in (19),

$$(\tilde{D}\mathbf{f})_k = \sum_{\substack{j=0\\ j\neq k}}^{N} \tilde{D}_{kj}(f_j - f_k) = \sum_{\substack{j=0\\ j\neq k}}^{N} \tilde{D}_{kj}(\tilde{x_j}^\ell - \tilde{x}_k^\ell) \tag{24}$$

$$= \sum_{\substack{j=0\\ j\neq k}}^{N} \frac{\frac{c_j}{c_k}}{\tilde{x}_k - \tilde{x}_j}(\tilde{x}_j - \tilde{x}_k)(\tilde{x}_j^{\ell-1} + \tilde{x}_j^{\ell-2}\tilde{x}_k + \cdots + \tilde{x}_j \tilde{x}_k^{\ell-2} + \tilde{x}_k^{\ell-1}) \tag{25}$$

$$= \sum_{\substack{j=0\\ j\neq k}}^{N} -\frac{c_k}{c_j}(\tilde{x}_j^{\ell-1} + \tilde{x}_j^{\ell-2}\tilde{x}_k + \cdots + \tilde{x}_j \tilde{x}_k^{\ell-2} + \tilde{x}_k^{\ell-1}). \tag{26}$$

Assuming that $\tilde{D}$ is computed by the original formula (3), we will observe a "cancellation of the cancellation" when $\tilde{D}$ is applied to monomials. The denominator in each matrix element of $\tilde{D}$ is multiplied by a term containing precisely this denominator (with a large relative error) as one of its factors. Although we cannot expect a perfect cancellation of the roundoff errors in (25), we can observe more accurate results for each of the terms in (25) when $\tilde{D}$ is computed by formula (3) and with the NST (17). Although a more accurately computed matrix (like the one from the Weideman–Reddy codes) is closer to the exact matrix, it will not exhibit this alleviation of cancellation errors. We believe this is the reason why using the original

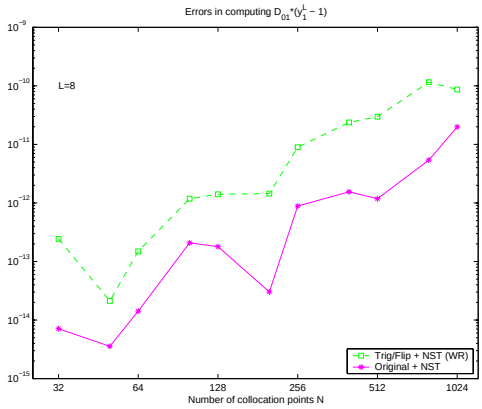

FIG. 7. *Errors in* $D_{01}(x_1^8 - 1)$.

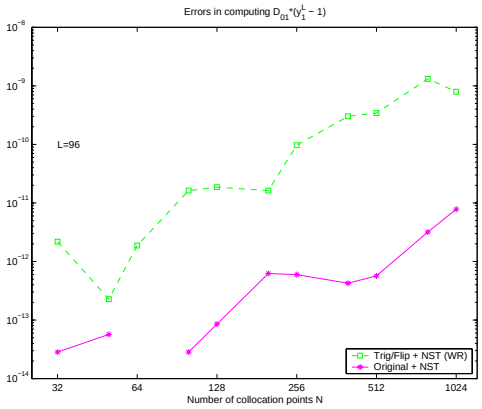

FIG. 8. *Errors in* $D_{01}(x_1^{96} - 1)$.

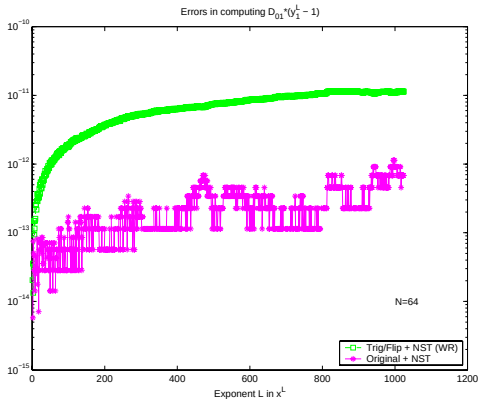

FIG. 9. *Errors in computing* $D_{01}(x_1^\ell - 1)$ *for* $N = 64$.

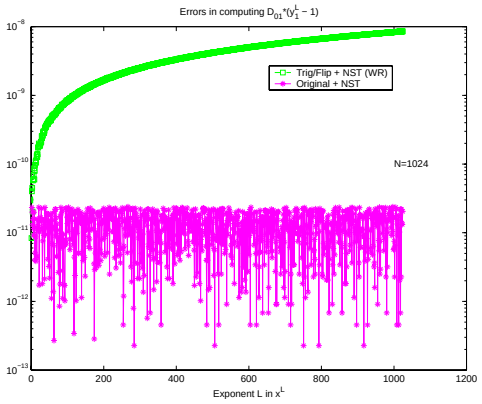

FIG. 10. *Errors in computing* $D_{01}(x_1^\ell - 1)$ *for* $N = 1024$.

"bad" formula for D combined with the NST gives usually superior results to using a more accurately computed matrix (with trigonometric identities and flipping trick) with NST. To confirm this we conduct experiments comparing the accuracy of the first term in the sum (24) for $(\tilde{D}\mathbf{f})_0$ with the matrix computed by (3) and NST to the matrix computed by the Weideman–Reddy code [Wei-Red]. The results are plotted in Figures 7–10. In summary, the NST improves accuracy not only by enforcing $D\mathbf{1} = \mathbf{0}$, but also by improving the accuracy of $D\mathbf{x}^\ell$.

When using the Schneider–Werner or the NST formula (22), the accuracy of our computation depends to some extent on how accurately we compute the divided differences (or finite difference quotients)

$$\frac{f_j - f_k}{x_j - x_k}.$$

If $x_j - x_k$ is computed to much higher accuracy than $f_j - f_k$ (which is the case for the Weideman–Reddy code), then the accuracy of the divided difference will be of the same order as the one for the difference $f_j - f_k$. Evaluating $x_j - x_k$ in a straightforward manner will result in significantly less accurate values for this difference, but

it provides for the opportunity of a *cancellation* of the errors in the numerator and denominator of the divided difference.

2.5. Fast Schneider–Werner. Provided the matrix D is precomputed, the very accurate Schneider–Werner formula (21) can be implemented in $3N^2$ flops, a 50% increase in the workload over the straightforward matrix-vector multiplication. Since most of the errors in the matrix D are in the upper left and lower right corner, a faster implementation of this algorithm is possible without losing accuracy (see Figures 5 and 6). We compute $D\mathbf{f}$ by applying the Schneider–Werner formula (21) only to the $\sqrt{N}$ by $\sqrt{N}$ upper left and lower right corner of the matrix computed via (3), and a (modified) NST. The cost becomes $2N^2 + O(N)$, comparable to the matrix-vector multiplication.

3. Fourier differentiation.

3.1. Introduction. In the case of a periodic domain, say with period 2π, the trigonometric polynomial of degree $[N/2]$ interpolating a 2π-periodic function f between the equidistant points $x_j := 2j\pi/N,\ j = 0, 1, \ldots, N-1$, is given by

$$p_N(x) := \sum_{k=0}^{N-1} f_k L_k(x), \tag{27}$$

where $f_k := f(x_k)$ and L_k are the trigonometric Lagrange interpolation polynomials defined by (see [Ber])

$$L_k(x) := (-1)^k a_0 L(x) \operatorname{cst}\left(\frac{x - x_k}{2}\right)$$

with

$$a_0 := \prod_{i=1}^{N-1} \sin\left(\frac{x_0 - x_i}{2}\right), \qquad L(x) := \prod_{k=0}^{N-1} \sin\left(\frac{x - x_k}{2}\right)$$

and

$$\operatorname{cst}\varphi := \begin{cases} \csc\varphi & \text{if } N \text{ is } \textit{odd}, \\ \cot\varphi & \text{if } N \text{ is } \textit{even}. \end{cases}$$

As in section 2, we approximate the derivative of a 2π-periodic function f by differentiating and evaluating (27). The first order differentiation matrix $D_F^{(1)} = D$ is given by (see [Boy] or [Wel1])

$$D_{kj} = \frac{1}{2}(-1)^{k+j} \operatorname{cst}\left(\frac{x_k - x_j}{2}\right), \quad k \neq j, \tag{28}$$

$$D_{kk} = 0, \quad k = 0, 1, \ldots, N-1. \tag{29}$$

The matrix D is circulant (for the definition and properties of circulant matrices, see [Dav]) and the matrix-vector multiplication can be performed in only $O(N \log N)$ operations (if $N = 2^\ell,\ \ell = 1, 2, 3, \ldots$).

3.2. Rounding error analysis. We investigate the effect of roundoff error on the matrix D for N *even* (for N *odd*, the investigation can be done in the same way).

As explained in [Don-Sol1], the *relative* error in computing $\sin(x)$ for small x is roughly δ. On the other hand, $\sin(\pi - x)$ for small x (which equals $\sin(x)$) can only be calculated with *absolute* error comparable to δ, i.e., with a *relative* error of δ/x.

This results in a roundoff error of $O(N^2\delta)$ for the matrix elements in the upper right and lower left corner of D.

In exact arithmetic we have

$$\begin{aligned} D_{0,N-1} = -D_{N-1,0} &= \frac{(-1)^N}{2}\cot\left(\pi - \frac{\pi}{N}\right) = \frac{(-1)^N}{2}\cot\left(\frac{\pi}{N}\right) \\ &= (-1)^N \frac{\pi}{2N}\left(1 + O\left(\frac{\pi}{N}\right)\right). \end{aligned}$$

In finite precision arithmetic, however, for small x we have $\sin(\pi - x) = \sin(x) + O(\delta)$, so that

$$\tilde{D}_{0,N-1} = -\tilde{D}_{N-1,0} = \frac{(-1)^N}{2}\frac{\cos(\pi/N)}{\sin(\pi/N) + O(\delta)} = D_{0,N-1} + O\left(\frac{N^2}{\pi^2}\delta\right).$$

We can expect an $O(N^2\delta)$ roundoff error growth in the calculation of $D\mathbf{f}$ through (28) and (29).

3.3. Improving accuracy. We describe a number of different strategies to diminish roundoff errors in the computation of $D\mathbf{f}$.

1. *Flipping trick.* To avoid computing the sine function of arguments close to π one can again take advantage of the symmetry property and compute only half of D. To avoid cancellation in the term $x_j - x_k$, one can compute $x_j - x_k$ as $(j-k)\pi/n$. This method is implemented by Weideman and Reddy [Wei-Red] and gives nearly optimal results.

2. *NST.* In the Fourier case, we still have $\sum_{k=0}^{N-1} L_k(x) = 1$, so that we can use relation (17) to compute the diagonal elements of the differentiation matrix. This formula produces less accurate entries on the diagonal but still gives good approximate $D\mathbf{f}$. For best results, the sum in (17) must be computed carefully to avoid smearing.

3. *Schneider and Werner.* Analogous to (21) the derivative of p_N at the point x_k can be computed by

$$p'_N(x_k) = \sum_{\substack{j=0 \\ j\neq k}}^{N-1} D_{kj}(f_j - f_k).$$

In this case, however, the differentiation is not accomplished by a matrix-vector multiplication.

4. *FFT.* The FFT can be used to compute the coefficients in the Fourier expansion from the function values f_k. The cost of obtaining $D\mathbf{f}$ (without computing the matrix D) from $\mathbf{f}$ is $O(N \log N)$, resulting in an asymptotically faster algorithm.

In Figures 11 and 12 we see that the flipping trick and the NST give nearly the same results. The Schneider and Werner formula appears to give the best results. The improvement of the NST is, however, more visible in Figure 11 than in Figure 12. In the next section, we try to give an explanation of this phenomenon.

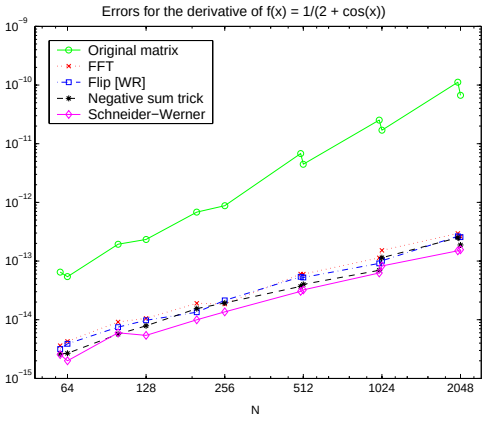

FIG. 11. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \frac{1}{2+\cos(x)}$.

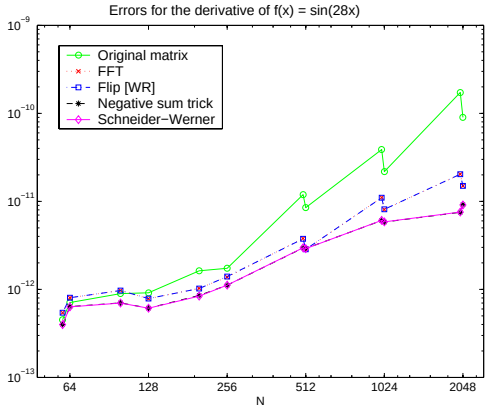

FIG. 12. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \sin(28x)$.

3.4. Error analysis of the NST. In exact arithmetic $D\mathbf{f}$ gives the exact derivative of f, whenever f is a trigonometric polynomial of degree not more than $[N/2]$,

$$f(x) = \frac{a_0}{2} + \sum_{i=1}^{[N/2]} (a_i \cos(ix) + b_i \sin(ix)). \tag{30}$$

NST makes sure that the constant term is differentiated exactly. Again, we can see what happens to the next few terms in (30), that is, to monomials $\sin(ix)$ and $\cos(ix)$, $i > 0$.

We apply the de Moivre equality and rewrite $\sin(i\alpha)$ as

$$\sin(i\alpha) = \sin(\alpha) \underbrace{\sum_{l=0}^{i} (-1)^l \binom{i}{2l+1} \cos^{i-2l-1}(\alpha) \sin^{2l}(\alpha)}_{=: M_i(\alpha)}, \tag{31}$$

where $\binom{n}{m} = 0$ when $n < m$.

For $f(x) = \sin(ix)$, we have

$$\begin{aligned} f_k - f_j = \sin(ix_k) - \sin(ix_j) &= 2\cos\left(i\frac{x_k + x_j}{2}\right) \sin\left(i\frac{x_k - x_j}{2}\right) \\ &= 2\cos\left(i\frac{x_k + x_j}{2}\right) \sin\left(\frac{x_k - x_j}{2}\right) M_i\left(\frac{x_k - x_j}{2}\right). \end{aligned}$$

For $f(x) = \cos(ix)$, we have

$$\begin{aligned} f_j - f_k = \cos(ix_k) - \cos(ix_j) &= -2\sin\left(i\frac{x_k + x_j}{2}\right) \sin\left(i\frac{x_k - x_j}{2}\right) \\ &= -2\sin\left(i\frac{x_k + x_j}{2}\right) \sin\left(\frac{x_k - x_j}{2}\right) M_i\left(\frac{x_k - x_j}{2}\right). \end{aligned}$$

If we apply the NST matrix to $\sin(ix)$ or $\cos(ix)$ we expect to observe again a "cancellation of the cancellation" as the term in the denominator of (28) contains $\sin(\frac{x_k - x_j}{2})$.

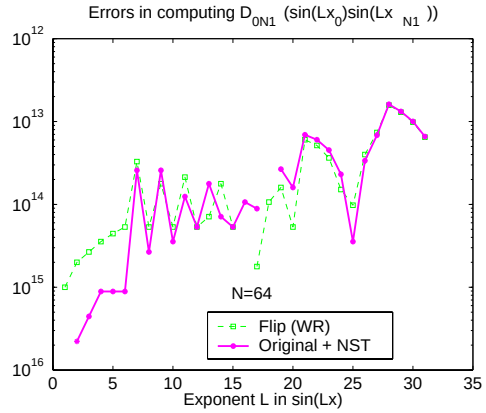

FIG. 13. *Errors in computing* $D_{0,N-1}\ (\sin(\ell x_0) - \sin(\ell x_{N-1}))$ *for* $N = 64$.

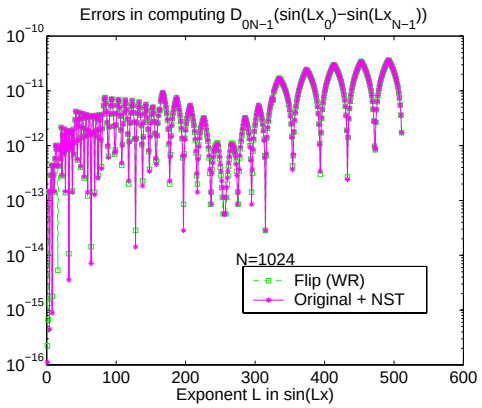

FIG. 14. *Errors in computing* $D_{0,N-1}\ (\sin(\ell x_0) - \sin(\ell x_{N-1}))$ *for* $N = 1024$.

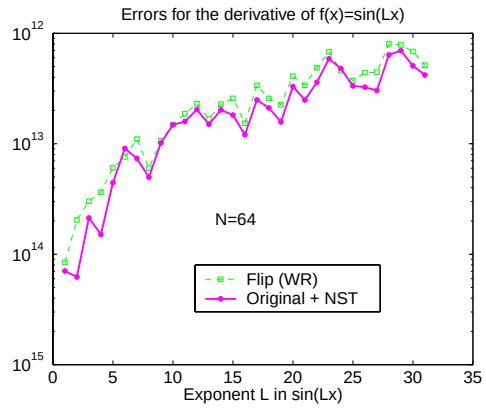

FIG. 15. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \sin(\ell x)$, $\ell = 0, 1, \ldots, 31$, $N = 64$.

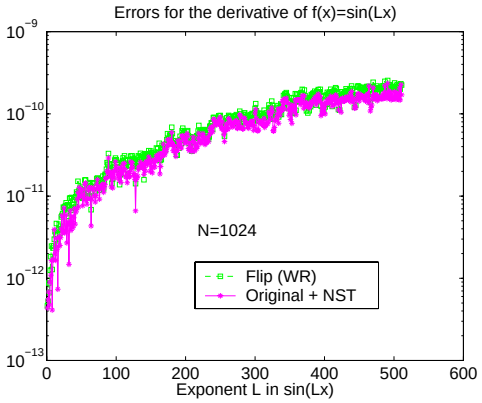

FIG. 16. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = \sin(\ell x)$, $\ell = 0, 1, \ldots, 511$, $N = 1024$.

We see in Figure 12 that the Weideman–Reddy, FFT, NST, and (even!) Schneider–Werner methods have almost all the same accuracy. We believe this is due to the fact that when evaluating the sine function, an argument reduction is performed, which destroys (in part) the nice "cancellation of cancellation" property encountered in section 2.

To confirm this, we repeat the numerical experiments conducted in section 2.4.3. We compare the accuracy of the quantity

$$D_{0,N-1}(\sin(\ell x_0) - \sin(\ell x_{N-1})), \qquad \ell = 1, 2, \ldots,$$

for $D_{0,N-1}$ computed by (28) (i.e., the original matrix *and* NST) to $D_{0,N-1}$ computed by the Weideman–Reddy code [Wei-Red]. The results are plotted in Figures 13 and 14. Figures 15 and 16 show the errors in approximating the derivatives for a pure harmonic with various frequencies. We observe the "cancellation of cancellation" in the NST method only for *small* values of ℓ. This confirms the results obtained in Figure 11 where the Schneider–Werner method was the best among all methods.

In summary, the NST is not nearly as effective in the Fourier case as in Chebyshev and other polynomial differentiation.

4. Polynomial differentiation. In this section, we present the error growth for the approximation of derivatives of a function at arbitrary interpolation points.

4.1. Differentiation. There are a number of formulas to compute the first order differentiation matrix $D = D_p^{(1)}$ of a polynomial (of degree $\leq N$) interpolating a function $f(x)$ on the interval $[a, b]$. The Lagrangian form of this polynomial is

$$p_N(x) := \sum_{k=0}^{N} f_k L_k(x), \tag{32}$$

where $f_k := f(x_k)$ and x_k, $k = 0, 1, \ldots, N$, is a set of distinct points.

The entries of the ℓth order differentiation matrix are given by the ℓth derivative of the Lagrange polynomial at the interpolating points:

$$D_{jk}^{(\ell)} := \frac{d^\ell}{dx^\ell} \left[L_k(x) \right]_{x=x_j}.$$

Welfert [Wel1] and Funaro [Fun] give formulas for the first order differentiation matrix. To alleviate the errors in the computation of the matrix $D^{(\ell)}$, we suggest (as in [Bal-Ber, Bal-Ber2] and [Bal]) using the barycentric representation of $p_N(x)$ [Hen]:

$$p_N(x) = \frac{\displaystyle\sum_{k=0}^{N} \frac{\lambda_k}{x - x_k} f_k}{\displaystyle\sum_{k=0}^{N} \frac{\lambda_k}{x - x_k}}, \tag{33}$$

where $\lambda_k^{-1} := \prod_{\substack{j=0 \\ j \neq k}}^{N} (x_k - x_j)$.

For large values of N roundoff error will occur in the computation of the λ_k's. To avoid this problem, the λ_k should be computed as (see [Cos-Don])

$$\eta_k := \sum_{\substack{j=0 \\ j \neq k}}^{N} \ln \mid x_k - x_j \mid, \qquad \lambda_k^{-1} := (-1)^k e^{-\eta_k}.$$

The barycentric form of the polynomial $p_N(x)$ is one of the most stable ways to evaluate the polynomial (see [Hen]), and proves also useful in the computation of the derivatives of the polynomial.

For the first order differentiation matrix, we employ the formula of Schneider and Werner (see [Sch-Wer]) to obtain

$$D_{kj} = \frac{\lambda_j}{\lambda_k} \frac{1}{x_k - x_j}, \quad k \neq j, \tag{34}$$

$$D_{kk} = -\sum_{\substack{j=0 \\ j \neq k}}^{N} D_{kj}, \quad k = 0, 1, \ldots, N. \tag{35}$$

Here we explicitly have the NST.

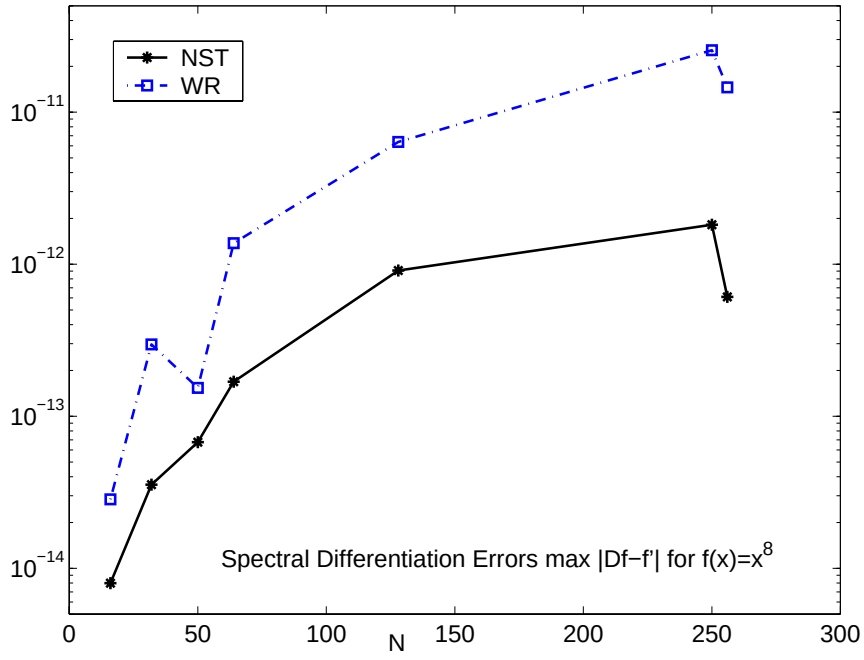

FIG. 17. *Errors* $||D\mathbf{f} - \mathbf{f}'||_\infty$ *for* $f(x) = x^8$.

For higher order differentiation matrices, we propose using

$$D_{kj}^{(\ell)} = \frac{\ell}{x_k - x_j}\left(\frac{\lambda_j}{\lambda_k} D_{kk}^{(\ell-1)} - D_{kj}^{(\ell-1)}\right), \quad k \neq j, \tag{36}$$

$$D_{kk}^{(\ell)} = -\sum_{\substack{j=0 \\ j\neq k}}^{N} D_{kj}^{(\ell)}, \quad k = 0, 1, \ldots, N. \tag{37}$$

For $j \neq k$ this is the formula proposed in [Wei-Red] and [Wel2]. For $j = k$, we use the NST. The error analysis of sections 2 and 3 can be applied to the general polynomial case. With the NST, we have "compensation" of roundoff errors in the calculation of the approximation of the ℓth derivative of f.

4.2. Legendre–Gauss–Lobatto differentiation. The first order differentiation matrix for (Legendre–)Gauss–Lobatto points can be given explicitly (see [Chqz] or [Wel1]).

In Figure 17, we compare two methods for computing the derivative of the interpolating polynomial: the method proposed by Weideman and Reddy [Wei-Red] and the NST applied to the (Legendre–)Gauss–Lobatto differentiation matrix. The NST produces more accurate results than the Weideman–Reddy method.

We can repeat the error analysis given in section 2.4 to show that the error behaves like $O(N^2\delta)$. The NST compensates for the $O(N^4\delta)$ error growth in the $(0, 1)$ element of the matrix.

As explained in [Bal], we can use the flipping trick of Solomonoff [Sol]. This halves the number of operations needed for calculating the derivative of a function.

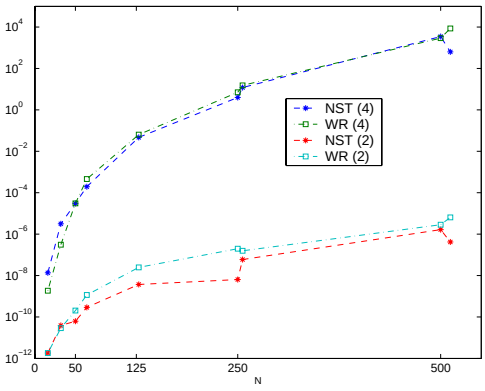

FIG. 18. *Errors* $||D^{(2)}\mathbf{f} - \mathbf{f}^{(2)}||_\infty$ *and* $||D^{(4)}\mathbf{f} - \mathbf{f}^{(4)}||_\infty$ *for* $f(x) = x^8$, *Chebyshev case.*

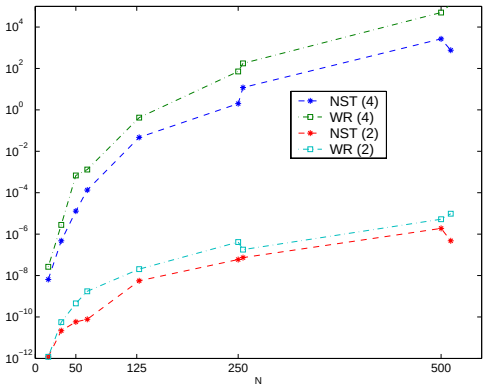

FIG. 19. *Errors* $||D^{(2)}\mathbf{f} - \mathbf{f}^{(2)}||_\infty$ *and* $||D^{(4)}\mathbf{f} - \mathbf{f}^{(4)}||_\infty$ *for* $f(x) = x^8$, *Legendre case.*

It does not, however, improve the quality of the approximation error (as is the case for Chebyshev points). For further computational results we refer to [Bal].

4.3. Higher order differentiation. There are different ways of computing higher order differentiation matrices.

For the Chebyshev case, Bayliss, Class, and Matkowsky [Bay-Cla-Mat] propose to first compute the differentiation matrix $D^{(\ell)}$ and then correct the diagonals of the resulting matrix product by the NST.

For the Chebyshev case, Weideman and Reddy [Wei-Red] use (36)–(37). For the general polynomial case, they use the recursive procedure proposed by Welfert in [Wel1].

We have conducted experiments comparing the different methods. For higher order differentiation matrix we recommend using the Weideman–Reddy code poldif.m *and* correcting the diagonals of the resulting matrix by the NST.

Figures 18 and 19 show the errors in computing the second and fourth order derivative of the function $f(x) = x^8$ for the Chebyshev–Gauss–Lobatto and Legendre–Gauss–Lobatto points.

For the Chebyshev case, we compare the results obtained through the routine chebdif.m with the results obtained by poldif.m *and* the NST applied to the final matrix. Again, it is very important to sum carefully to avoid smearing.

For the Legendre case, we compare the results obtained by poldif.m with the results obtained by poldif.m *and* the NST applied to the final matrix.

5. Applications. In most applications of spectral differentiation we are not simply faced with the problem of finding the derivative of a function. In fact, rather than finding $g = Df$, we often solve the inverse problem, i.e., given g we want to find a function (vector) u such that $Du = g$ (or more realistically, $Au = g$, where A is generated in one form or another by D).

In the previous sections we saw that the forward problem $f \to Df$ is quite sensitive to perturbations in D. It may not be a surprise that the inverse problem appears to be much less sensitive to how well the matrix D is computed.

Again as noted in the previous sections, the sensitivity of the forward problem is highest near the boundary, i.e., it is most important to compute the first few and

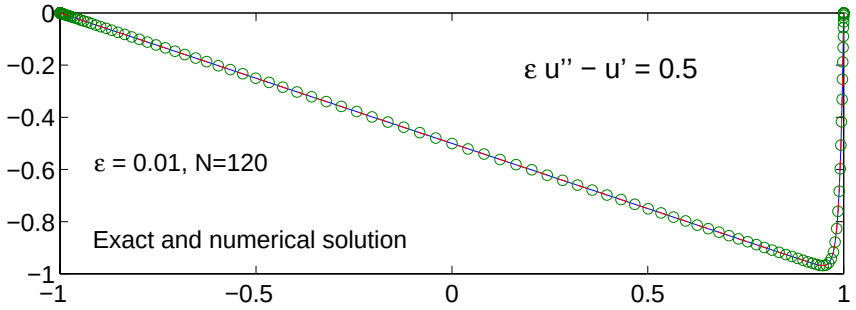

FIG. 20. *Solution to* $\epsilon u'' - u' = \frac{1}{2}$.

last few rows of D accurately. Because of boundary conditions, these are precisely the rows of D which often are removed from the matrix.

The examples below show that in general it is best to compute the spectral differentiation matrix accurately (in our opinion, that means applying the NST), but the penalty for ignoring this advice is not always severe.

5.1. Singularly perturbed boundary value problem (BVP). We solve a singularly perturbed second order BVP

$$\epsilon u''(x) - u'(x) = \frac{1}{2}, \qquad -1 < x < 1, \qquad u(-1) = u(+1) = 0. \tag{38}$$

The exact solution,

$$u(x) = -\frac{x+1}{2} + \frac{e^{(x-1)/\epsilon} - e^{-2/\epsilon}}{1 - e^{-2/\epsilon}},$$

has a layer at the right boundary.

We solve the problem numerically for $\epsilon = 0.01$ using a Chebyshev spectral collocation method ($N = 120$) and a coordinate transformation to resolve the layer (see [Tan-Tru] for details of the method). We compute the differentiation matrices using trigonometric identities and flipping trick (but no NST) as in [Tan-Tru], with the Weideman–Reddy code, and with NST only. Figures 20 and 21 show the solution and the errors with the three methods.

The method using NST only performs best, but the improvement is not large. The method of [Tan-Tru] is more accurate (by about a factor of 10) than the method using the original Chebyshev matrix formulas.

5.2. Fourth order BVP. We solve a fourth order BVP, namely,

$$\begin{aligned} u^{(4)}(x) &= f(x), \qquad -1 < x < 1, \\ -u(-1) = u(+1) &= 5, \qquad u''(-1) = u''(+1) = 0. \end{aligned} \tag{39}$$

The exact solution is given by $u(x) = 10\sin(x)(x^2-1)^3 + 5x$.

In Figure 22, we compare three different methods for solving problem (39) using a Legendre(–Gauss–Lobatto) spectral collocation method. The first method uses the "original matrix," the second method uses the routine poldif.m of Weideman and Reddy, and the third method uses the routine poldif.m *and* the NST.

The method using NST gives the best results, but the improvement is not large.

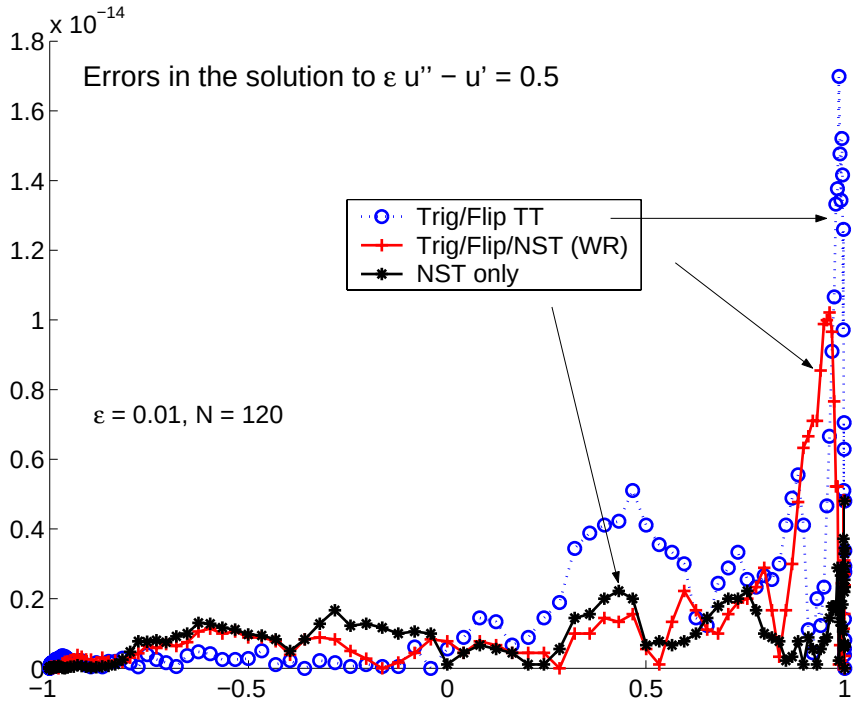

FIG. 21. *Errors in the numerical solution of* (38).

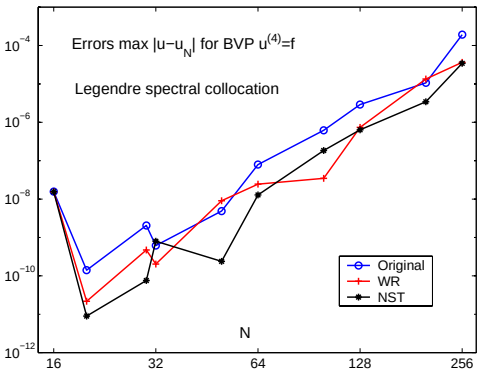

FIG. 22. *Errors* $||u - u_N||_\infty$ *for* (39).

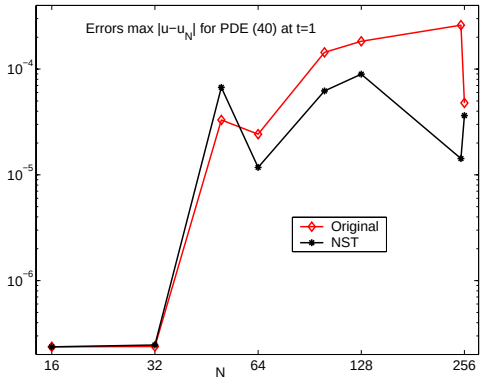

FIG. 23. *Errors* $||u - u_N||_\infty$ *for* (40).

5.3. Time dependent problem. We solve the time dependent problem

$$\begin{aligned} &u_t + xu_x = 0, \quad -1 \le x \le 1, \quad 0 < t \le T, \\ &u(x,0) = f(x), \quad -1 \le x \le 1. \end{aligned} \tag{40}$$

The function f is given by $f(x) := \cos^2(\frac{\pi x}{2})$ and the exact solution is $u(x,t) = f(x\exp(-t))$.

In this problem, no boundary condition is required. We solve (40) using the Chebyshev collocation method in space and a Runge–Kutta method (the built-in MATLAB solver ode45; see [Sha-Rei] for details) in time. We compare two different

methods. The first method uses the "original matrix" given by formulas (3) to (6) and the second method uses (3) *and* the NST.

In Figure 23 we plot the errors at time $t = 1$ versus the number of collocation points. The NST method produces more accurate results, but the improvement is nowhere nearly as dramatic as for the forward problem.

6. Conclusion. In this paper we have studied the roundoff properties of spectral differentiation matrices for Chebyshev, Legendre, equally spaced (Fourier) and arbitrary collocation points. We have demonstrated that the most accurate way to approximate the derivative of a function f via spectral differentiation matrices is to use the inaccurate standard formulas and to apply the "negative sum trick." We have shown the close relation between the negative sum trick and the formulas of Schneider and Werner. We have pointed out that there is a cancellation of roundoff errors when one uses the straightforward formulas (3) and the negative sum trick.

The forward problem $f \to Df$ is very sensitive to roundoff errors, especially near the boundary, whereas the inverse problem (that appears in most applications) is much less sensitive to roundoff.

In general, it is better to compute the differentiation matrix accurately by applying the negative sum trick.

Acknowledgments. The authors would like to thank Ricardo Carretero for valuable discussions and Cleve Moler for helping us understand some floating point aspects of MATLAB. The first author would like to thank all those who supported his stay at Simon Fraser University.

REFERENCES

[Bal] R. BALTENSPERGER, *Improving the accuracy of the matrix differentiation method for arbitrary collocation points,* Appl. Numer. Math., 33 (2000), pp. 143–149.

[Bal-Ber] R. BALTENSPERGER AND J.-P. BERRUT, *The errors in calculating the pseudospectral differentiation matrices for Čebyšev-Gauss-Lobatto points,* Comput. Math. Appl., 37 (1999), pp. 41–48.

[Bal-Ber2] R. BALTENSPERGER AND J.-P. BERRUT, *Errata to: "The errors in calculating the pseudospectral differentiation matrices for Čebyšev-Gauss-Lobatto points," [Comput. Math. Appl.,* 37 (1999), *no.* 1, 41–48*]*, Comput. Math. Appl., 38 (1999), p. 119.

[Bay-Cla-Mat] A. BAYLISS, A. CLASS, AND B. MATKOWSKY, *Roundoff error in computing derivatives using the Chebyshev differentiation matrix,* J. Comput. Phys., 116 (1995), pp. 380–383.

[Ber] J.-P. BERRUT, *Baryzentrische Formeln zur trigonometrischen Interpolation* (I), Z. Angew. Math. Phys., 35 (1984), pp. 91–105.

[Bre-Eve] K. S. BREUER AND R. M. EVERSON, *On the errors incurred calculating derivatives using Chebyshev polynomials,* J. Comput. Phys., 99 (1992), pp. 56–67.

[Boy] J. P. BOYD, *Chebyshev and Fourier Spectral Methods*, Lecture Notes in Engineering 49, Springer-Verlag, Berlin, 1989.

[Cos-Don] B. COSTA AND W. S. DON, *On the computation of high order pseudospectral derivatives,* Appl. Numer. Math., 33 (2000), pp. 151–159.

[Chqz] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods in Fluid Dynamics*, Springer Ser. Comput. Phys., Springer-Verlag, New York, 1988.

[Dav] P. J. DAVIS, *Circulant Matrices*, John Wiley, New York, Chichester, Brisbane, 1979.

[Don-Sol1] W. S. DON AND A. SOLOMONOFF, *Accuracy and speed in computing the Chebyshev collocation derivative,* SIAM J. Sci. Comput., 16 (1995), pp. 1253–1268.

[Don-Sol2] W. S. Don and A. Solomonoff, *Accuracy enhancement for higher derivatives using Chebyshev collocation and a mapping technique,* SIAM J. Sci. Comput., 18 (1997), pp. 1040–1055.

[Fun] D. Funaro, *Polynomial Approximation of Differential Equations*, Springer-Verlag, Berlin, 1992.

[Hen] P. Henrici, *Essentials of Numerical Analysis*, John Wiley, New York, 1982.

[Kos-Tal] D. Kosloff and H. Tal-Ezer, *A modified Chebyshev pseudospectral method with an $\mathcal{O}(N^{-1})$ time step restriction,* J. Comput. Phys., 104 (1993), pp. 457–469.

[Rot] E. E. Rothman, *Reducing round-off error in Chebyshev pseudospectral computations,* in High Performance Computing II, M. Durand and F. El Dabaghi, eds., Elsevier–North Holland, Amsterdam, 1991, pp. 423–439.

[Sch-Wer] C. Schneider and W. Werner, *Some new aspects of rational interpolation,* Math. Comp., 47 (1986), pp. 285–299.

[Sha-Rei] L. F. Shampine and M. W. Reichelt, *The* Matlab *ODE suite,* SIAM J. Sci. Comput., 18 (1997), pp. 1–22.

[Sol] A. Solomonoff, *A fast algorithm for spectral differentiation,* J. Comput. Phys., 98 (1992), pp. 174–177.

[Tan-Tru] T. Tang and M. R. Trummer, *Boundary layer resolving pseudospectral methods for singular perturbation problems,* SIAM J. Sci. Comput., 17 (1996), pp. 430–438.

[Tre] L. N. Trefethen, *Spectral Methods in* Matlab, SIAM, Philadelphia, 2000.

[Wei-Red] J. A. C. Weideman and S. C. Reddy, *A* Matlab *differentiation matrix suite*, ACM Trans. Math. Software, 26 (2000), pp. 465–519.

[Wel1] B. D. Welfert, *A Remark on Pseudospectral Differentiation Matrices*, Report, Department of Mathematics, Arizona State University, Tempe, AZ, 1992; SIAM J. Numer. Anal., submitted.

[Wel2] B. D. Welfert, *Generation of pseudospectral differentiation matrices* I, SIAM J. Numer. Anal., 34 (1997), pp. 1640–1657.