IMA Journal of Numerical Analysis (2020) **00**, 1–13
doi:10.1093/imanum/draa006

An algorithm for calculating Hermite-based finite difference weights

Bengt Fornberg
Department of Applied Mathematics, University of Colorado, Boulder, CO 80309, USA
fornberg@colorado.edu

[Received on 2 October 2019; revised on 25 January 2020]

Finite difference (FD) formulas approximate derivatives by weighted sums of function values. Given arbitrarily distributed node locations in one-dimension, a previous algorithm by the present author (1988, Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput*., **51**, 699–706) provides FD weights of optimal order of accuracy for approximating any order derivative at a specified location. This algorithm is extended here to the case of finding weights to apply not only to function values but also to first derivative values in the case that these also are available. The MATLAB code for the algorithm is provided, and two examples are given to illustrate how this type of FD stencil can be applied to solving partial differential equations.

Keywords: FD; Hermite-FD; Hermite interpolation; FD weights.

1. Introduction

Finite differences (FD) have a long history, being used for solving ordinary differential equations (ODEs) in the 19th century, and introduced for partial differential equations (PDEs) in a pioneering study by Richardson (1911). In the common case that node layouts in more than one-dimension are grid based, explicit one-dimensional approximations can usually be applied separately in each spatial direction.[1] Table 1 shows sketches of three different types of one-dimensional FD stencils. In the algorithms listed here, k denotes the order of derivative to be approximated. This can be any non-negative integer, with the algorithms in the $k = 0$ case providing weights for interpolation formulas. Algorithm 1 is convenient both for equispaced and nonequispaced nodes. It is computationally very fast, and does not involve any (potentially ill-conditioned) linear system solution step. Algorithm 2 requires (in a symbolic language, such as Mathematica) only two lines of code, and gives weights in exact (rather than floating point) format. As noted in its references, one of its applications is to provide weights for most standard classes of ODE linear multistep methods.

The reason for here generalizing Algorithm 1 to Algorithm 3 is to make these Hermite-type FD approximations equally easily available.[2] In situations where both f and f' values are available at the node points (which can be arbitrarily spaced), these types of approximations provide a higher order of accuracy for the same stencil width and, as we will also see in Section 3, lower leading error coefficient

[1] In the case of nodes irregularly scattered in multiple dimensions (for purposes such as aligning nodes with irregular boundaries or interfaces, or providing local refinement), FD generalizes to radial basis function-generated finite differences (RBF-FD) as surveyed, for example, in Fornberg & Flyer (2015a,b).

[2] Other options for finding Hermite-type weights include solving linear systems which enforce exact result for polynomials, and a contour-integration-based approach (Butcher *et al.*, 2011). Some greater flexibility can be gained at the expense of computational cost and numerical stability.

TABLE 1 *Summary of Algorithms* 1–3. *In the stencil sketches, the vertical separation between the markers has no significance other than graphical clarity. All the indicated locations are along a single axis (which we call the x-axis)*

Algorithm number	Stencil sketch	Weights applied to	Brief description
1	○ □ □□□ □□	$f^{(k)}$ f	Nodes arbitrarily spaced. Original Fornberg (1988) with FORTRAN code; MATLAB code in Fornberg & Flyer (2015a).
2	○○○○ □□□□□□□□	$f^{(k)}$ f	Nodes equally spaced in f and $f^{(k)}$. The two sets can be arbitrarily shifted. Reference Fornberg (1998) with Mathematica code; see also Fornberg & Flyer (2015a).[a]
3	○ ▽ ▽ ▽▽ ▽ ▽ □ □□□ □□	$f^{(k)}$ f' f	f and f' locations the same, arbitrarily spaced. Present study; MATLAB code provided.

[a]Using more recent Mathematica syntax.

for matching orders of accuracy (unsurprisingly since, in typical use, they pick up more information local to the point of interest than do stencils that only use function values).

Following the description in Section 2 of the present HFD (Hermite-FD) algorithm (Algorithm 3), two simple PDE test problems are used in Section 3 to illustrate the high accuracies that can be obtained even with small stencils. Some conclusions (in Section 4) are followed by references and two appendices, one with a MATLAB code for the present algorithm, and one with a couple of FD / HFD weight tables.

2. Hermite algorithm

We first summarize briefly the key concept behind Algorithm 1. This is followed by a description of how one builds on this to obtain the Hermite algorithm that the present study focuses on (referred to above as Algorithm 3).

2.1 *Concept behind Algorithm* 1

Since Algorithm 1 has been described repeatedly (first in Fornberg, 1988 and later, with graphical illustrations of its recursions, in Fornberg, 1990, 1992), we will here limit ourselves to a brief summary of its key concept.

Let the nodes be located at $x = x_0, x_1, \ldots, x_n$ (distinct, but otherwise arbitrary) and consider stencils that approximate $\frac{\mathrm{d}^k f}{\mathrm{d}x^k}$, for simplicity at $x = 0$. The Lagrange interpolation polynomial takes the form

$$p_n(x) = \sum_{i=0}^{n} L_{i,n}(x)\, f(x_i), \tag{2.1}$$

where

$$L_{i,n}(x) = \begin{cases} 1, & x = x_i, \\ 0, & x = x_l, l \neq i. \end{cases} \tag{2.2}$$

These *Lagrange kernels* $L_{i,n}(x)$ can (uniquely) be written as

$$L_{i,n}(x) = \prod_{\substack{l=0 \\ l \neq i}}^{n} \frac{x - x_l}{x_i - x_l} = \sum_{j=0}^{n} c_{i,j,n} x^j. \tag{2.3}$$

Differentiating (2.1) and (2.3) k times gives

$$\frac{\mathrm{d}^k}{\mathrm{d}x^k} p_n(x)\bigg|_{x=0} = \sum_{i=0}^{n} \frac{\mathrm{d}^k}{\mathrm{d}x^k} L_{i,n}(x)\bigg|_{x=0} f(x_i) = \sum_{i=0}^{n} k!\, c_{i,k,n} f(x_i).$$

The desired FD weights can thus be read off from the coefficients $c_{i,j,n}$ in the right-hand side (RHS) of (2.3). The task of finding the FD weights is thus equivalent to that of rearranging the *Lagrange kernel* (2.3) into standard polynomial (finite Taylor expansion) form. The product form of $L_{i,n}(x)$ in (2.3) can be shown to provide a couple of recursion relations for the $c_{i,j,n}$ coefficients such that these all can be deduced explicitly starting from the trivial $c_{0,0,0} = 1$. In contrast to finding weights by solving linear systems, this algorithm is computationally very fast and also entirely stable numerically.

2.2 *Hermite algorithm (Algorithm 3)*

Considering interpolation formulas based on $f(x_i)$ and $f'(x_i)$ values (but no higher still derivatives), the Hermite polynomial counterparts to (2.1) and (2.2) take the form

$$q_{2n+1}(x) = \sum_{i=0}^{n} D_{i,n}(x) f(x_i) + \sum_{i=0}^{n} E_{i,n}(x) f'(x_i), \tag{2.4}$$

where

$$D_{i,n}(x) = \begin{cases} 1, & x = x_i, \\ 0, & x = x_l, l \neq i \end{cases} \quad \text{and} \quad \frac{\mathrm{d}D_{i,n}(x)}{\mathrm{d}x} = 0, x = x_i,$$

$$E_{i,n}(x) = 0, x = x_i \quad \text{and} \quad \frac{\mathrm{d}E_{i,n}(x)}{\mathrm{d}x} = \begin{cases} 1, & x = x_i, \\ 0, & x = x_l, l \neq i. \end{cases}$$

The function $(L_{i,n}(x))^2$ satisfies all requirements for $D_{i,n}(x)$ with the exception that, instead of being zero, $\frac{\mathrm{d}(L_{i,n}(x))^2}{\mathrm{d}x}|_{x=x_i} = 2\frac{\mathrm{d}L_{i,n}(x)}{\mathrm{d}x}|_{x=x_i} = 2\sum_{l \neq i} \frac{1}{x_i - x_l} = 2s_{i,j}$. The function $(x - x_i)(L_{i,n}(x))^2$ satisfies all the requirements for $E_{i,n}(x)$. Thus, we can write $D_{i,n}(x)$ and $E_{i,n}(x)$ in terms of $L_{i,n}(x)$ as

$$\begin{aligned} D_{i,n}(x) &= (1 - 2s_{i,j}(x - x_i)) \left(L_{i,n}(x)\right)^2, \\ E_{i,n}(x) &= (x - x_i) \left(L_{i,n}(x)\right)^2. \end{aligned} \tag{2.5}$$

By knowing (from Algorithm 1) the Taylor expansions for $L_{i,n}(x)$, we obtain from (2.5) the Taylor coefficients for $D_{i,n}(x)$ and $E_{i,n}(x)$. Since (2.4) implies

$$\left.\frac{\mathrm{d}^k q_{2n+1}(x)}{\mathrm{d}x^k}\right|_{x=0} = \sum_{i=0}^{n} \left.\frac{\mathrm{d}^k D_{i,n}(x)}{\mathrm{d}x^k}\right|_{x=0} f(x_i) + \sum_{i=0}^{n} \left.\frac{\mathrm{d}^k E_{i,n}(x)}{\mathrm{d}x^k}\right|_{x=0} f'(x_i),$$

these Taylor coefficients provide the desired weights in this Hermite case.[3]

Appendix A contains a MATLAB function *weights* that implements both Algorithms 1 and 3. The leading comment lines explain how the routine is used.

EXAMPLE Use the MATLAB function *weights* to obtain the weights at $x_k = -3,\ -2,\ -1,\ 0,\ 1,\ 2$ for approximating $f(0)$, $f'(0)$, $f''(0)$, $f'''(0)$ when using

1. $f(x_k)$ values only (output array c) and
2. $f(x_k)$ and $f'(x_k)$values (output arrays d, e).

The statements

$$\begin{aligned} z &= 0; \\ x &= -3:3; \\ m &= 3; \\ [c,d,e] &= \text{weights}(z,x,m) \end{aligned}$$

produce as output[4]

$$\mathrm{c} = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{60} & \frac{3}{20} & -\frac{3}{4} & 0 & \frac{3}{4} & -\frac{3}{20} & \frac{1}{60} \\ \frac{1}{90} & -\frac{3}{20} & \frac{3}{2} & -\frac{49}{18} & \frac{3}{2} & -\frac{3}{20} & \frac{1}{90} \\ -\frac{1}{8} & -1 & \frac{13}{8} & 0 & -\frac{13}{8} & 1 & -\frac{1}{8} \end{matrix}$$

$$\mathrm{d} = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{157}{18000} & \frac{69}{250} & \frac{39}{16} & -\frac{49}{9} & \frac{39}{16} & \frac{69}{250} & \frac{157}{18000} \\ -\frac{167}{18000} & -\frac{963}{2000} & -\frac{171}{16} & 0 & \frac{171}{16} & \frac{963}{2000} & \frac{167}{18000} \end{matrix}$$

[3] Formulas (but not FD weights) for non-equispaced Hermite interpolation on periodic data were considered in Salzer (1960).

[4] Using MATLAB symbolic toolbox, and declaring (noninteger) variables as 'sym', makes the algorithm return weights in exact rational form.

$$\mathrm{e} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{600} & \frac{9}{100} & \frac{9}{8} & 0 & -\frac{9}{8} & -\frac{9}{100} & -\frac{1}{600} \\ -\frac{1}{600} & -\frac{27}{200} & -\frac{27}{8} & -\frac{49}{3} & -\frac{27}{8} & -\frac{27}{200} & -\frac{1}{600} \end{matrix}$$

All the nontrivial entries in this output can be seen also in Tables B1 and B2 (which are often convenient as a reference when using centered FD schemes in equispaced node cases).[5]

3. Numerical illustrations for solving two simple PDEs

Before applying different types of FD formulas to a PDE demonstration problem, let us first describe the formulas we are considering in some more detail.

3.1 *The FD formulas to be used*

The different FD formulas that are employed in the one-dimensional PDE test are listed in Table 2. These are all provided by the present MATLAB function *weights*. In the CFD (compact FD) cases, we need a simple trick. For example, to generate the CFD 4 formula, we start by forming the HFD approximation

$$f^{(5)}(x) = [\ 90 \quad 0 \quad -90\ ]/h^5 f + [\ 30 \quad 120 \quad 30\ ]/h^4 f' + \mathcal{O}(h^2) f^{(7)}(x), \tag{3.1}$$

which is exact for $f(x) = 1, x, \ldots, x^6$. The left-hand side and the error term both vanish for $f(x) = 1, x, \ldots, x^4$. Then multiplying (3.1) by $\frac{h^4}{180}$ normalizes the relation to give the CFD 4 formula in Table 2. For the CFD 8 case, we similarly start by considering $f^{(9)}(x)$.

3.2 *Accuracy comparisons by dispersion analysis*

We will, as our first PDE demonstration problem, consider the one-dimensional wave equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0. \tag{3.2}$$

Before turning to numerical results, we present some dispersion analysis of the methods listed in Table 2. The single Fourier mode solution to (3.2) is $u(x,t) = e^{i\omega x - i\alpha(\omega)t}$, with $\alpha(\omega) = \omega$. When the space direction x is discretized with a step length h, the available frequency range becomes limited to $-\frac{\pi}{h} < \omega \le \frac{\pi}{h}$.[6] If, for example, we approximate $\frac{\partial u}{\partial x}$ in (3.2) using FD 2, we obtain

$$\alpha(\omega) = \frac{1}{ih}\left(-\frac{1}{2}e^{-i\omega h} + \frac{1}{2}e^{i\omega h}\right) = \frac{\sin \omega h}{h} = \omega - \frac{h^2}{6}\omega^3 + \cdots$$

This function $\alpha(\omega)h$ is shown as the bottom curve in Fig. 1, and its approximation for small ω as the top-right entry in Table 2.

---

[5] As noted above, the present algorithm *weights* works just as well in noncentered cases, and when the nodes are not equally spaced.

[6] Due to *aliasing*; any higher frequency will, on the grid, appear identical to one within this range.

TABLE 2 *Summary of the regular, compact and HFD formulas that are compared in Sections 3.2 and 3.3*

Scheme (acc. order)	Stencil width	Stencil coefficients (nodes spaced distance h apart)	$\alpha(\omega)$, expanded around origin
FD 2	3	$f' \approx \left[-\frac{1}{2}\ 0\ \frac{1}{2}\right]/h\, f$	$\omega - \frac{h^2}{6}\omega^3 + \cdots$
4	5	$f' \approx \left[\frac{1}{12} - \frac{2}{3}\ 0\ \frac{2}{3} - \frac{1}{12}\right]/h\, f$	$\omega - \frac{h^4}{30}\omega^5 + \cdots$
6	7	$f' \approx \left[-\frac{1}{60}\ \frac{3}{20}\ -\frac{3}{4}\ 0\ \frac{3}{4}\ -\frac{3}{20}\ \frac{1}{60}\right]/h\, f$	$\omega - \frac{h^6}{140}\omega^7 + \cdots$
8	9	$f' \approx \left[\frac{1}{280}\ -\frac{4}{105}\ \frac{1}{5}\ -\frac{4}{5}\ 0\ \frac{4}{5}\ -\frac{1}{5}\ \frac{4}{105}\ -\frac{1}{280}\right]/h\, f$	$\omega - \frac{h^8}{630}\omega^9 + \cdots$
CFD 4	3	$\left[\frac{1}{6}\ \frac{2}{3}\ \frac{1}{6}\right] f' \approx \left[-\frac{1}{2}\ 0\ \frac{1}{2}\right]/h\, f$	$\omega - \frac{h^4}{180}\omega^5 + \cdots$
8	5	$\left[\frac{1}{70}\ \frac{8}{35}\ \frac{18}{35}\ \frac{8}{35}\ \frac{1}{70}\right] f' \approx \left[-\frac{5}{84}\ -\frac{8}{21}\ 0\ \frac{8}{21}\ \frac{5}{84}\right]/h\, f$	$\omega - \frac{h^8}{44100}\omega^9 + \cdots$
HFD 4	3	$f'' \approx [2\ -4\ 2]/h^2 f + \left[\frac{1}{2}\ 0\ -\frac{1}{2}\right]/h\, f'$	$\omega - \frac{h^4}{1080}\omega^5 + \cdots$
8	5	$f'' \approx \left[\frac{7}{54}\ \frac{64}{27}\ -5\ \frac{64}{27}\ \frac{7}{54}\right]/h^2 f + \left[\frac{1}{36}\ \frac{8}{9}\ 0\ -\frac{8}{9}\ -\frac{1}{36}\right]/h\, f'$	$\omega - \frac{h^8}{441000}\omega^9 + \cdots$

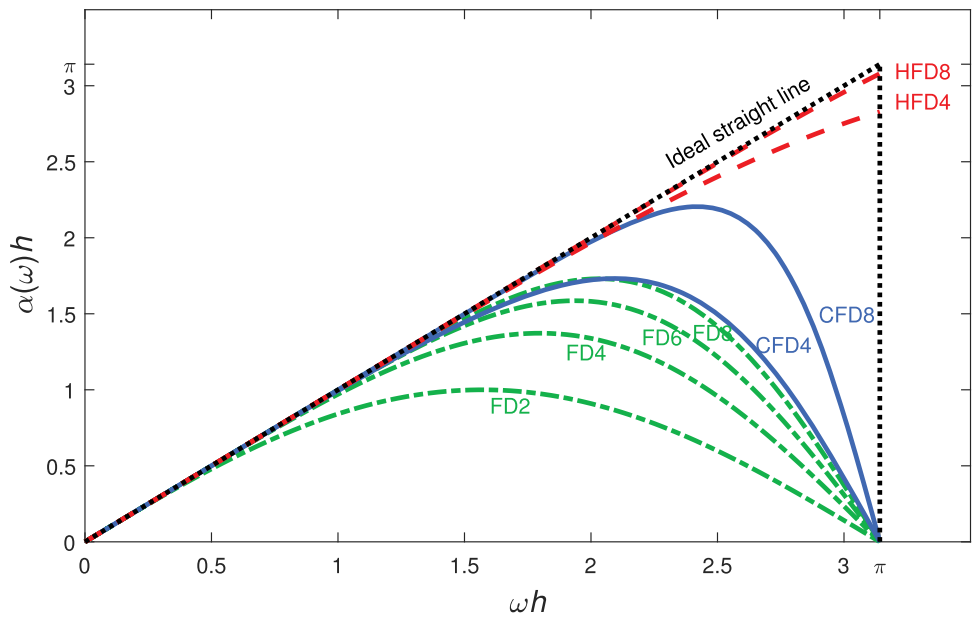

FIG. 1. Display of the dispersion curves $\alpha(\omega)h$ (as functions of ωh) for the different FD, CFD and HFD methods compared in Section 3. Since all these functions are odd, they are displayed only over $[0, \pi]$ (rather than $[-\pi, \pi]$).

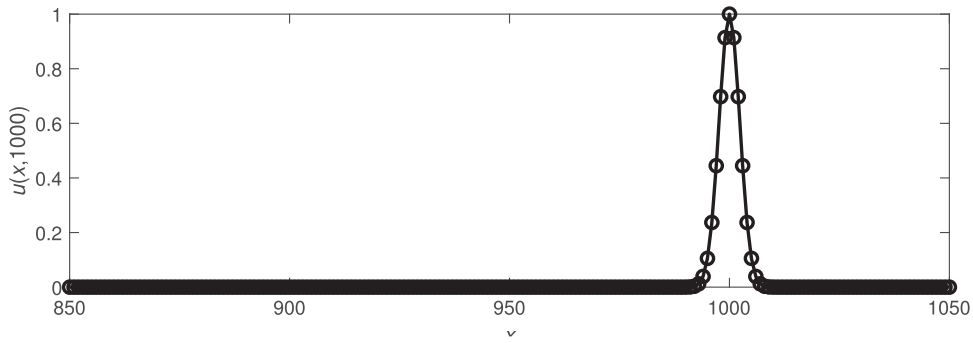

FIG. 2. Analytic solution to PDE test problem at time $t = 1000$, displayed over the spatial interval $850 \leq x \leq 1050$ with markers shown at integer x-coordinates.

The implementation of the regular FD approximations is entirely straightforward—just apply the stencils weights listed in Table 2. For the CFD approximations, the u_x-values at the nodes are obtained from the u-values by solving a tridiagonal and a five-diagonal linear system, respectively. For the HFD formulas we start by noting that $u_t = -u_x$ implies $(u_x)_t = -u_{xx}$. Writing $u_x = v$, we can thus solve (3.2) by advancing in time,

$$\begin{bmatrix} u \\ v \end{bmatrix}_t = - \begin{bmatrix} v \\ u_{xx} \end{bmatrix}. \tag{3.3}$$

The only spatial approximation required is to approximate u_{xx} in terms of u and $v = u_x$, i.e. exactly what the HFD formulas in Table 2 provide.

The remaining curves in Fig. 1 and approximations in Table 2 are obtained similarly to the FD 2 case, described in some detail above. We can now make several observations:

- The orders of accuracy for the spatial approximations match the orders by which the curves approximate the ideal straight line in Fig. 1 at $\omega h = 0$.
- For matching orders of accuracy, the CFD and HFD curves fit this straight line over a wider part of the frequency range $-\pi < \omega h \leq \pi$ than do the corresponding FD curves.
- The FD and CFD curves all go to zero at $\pm\pi$ (since the fastest oscillating grid function $\ldots, +1, -1, +1, -1, +1, \ldots$, arising for $\omega h = \pm\pi$, is symmetric around each node point, causing the first derivative to be approximated by zero at all of these). One key to HFD's superior accuracy is that these methods avoid this issue.

3.3 *Accuracy comparisons when solving the one-dimensional PDE test problem*

In this test we solve (3.2) with $u(x, 0) = e^{-(0.3x)^2}$ as initial condition. With uniform space discretization $h = 1$, the analytic solution at time $t = 1000$ will look as shown in Fig. 2. Figure 3 shows (over the same display interval $850 \leq x \leq 1050$) the numerical solutions (having used accurate time stepping, such that all errors that are seen come from the spatial approximations). The display is organized by FD stencil width (increasing from front to back) and by FD type (left to right).

It is visually obvious from Fig. 3 that the regular FD falls well short of the other methods in terms of accuracy—with highly conspicuous trailing wave trains unless the FD stencil widths are large. The

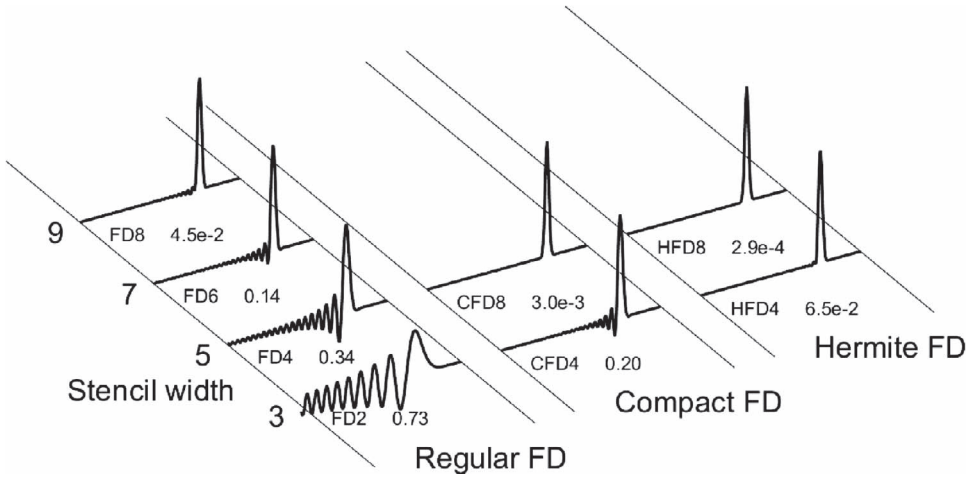

FIG. 3. Numerical solutions to the PDE test problem, using various FD schemes and different stencil widths/orders of accuracy, displayed final time of $t = 1000$, as in Fig. 2. For each case, the number shown following the acronym of the method is the max norm error of the numerical solution.

CFD methods are more accurate, but incur the cost of having to solve a banded linear system for each time step. The present Hermite approach offers by far the best accuracy. It steps forward in time twice the number of variables (u and v, rather than just u), but it involves no solutions of linear systems.

When using an explicit method in time, one would for $u_t = -u_x$ expect a time-step condition of the form $k/h \leq$ constant. The reformulation (3.3) does not alter this. With $\underline{u}(t)$ and $\underline{v}(t)$ denoting column vectors with u- and v-values after the Hermite-type spatial discretization, (3.3) requires time stepping of a system with block structure

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\begin{array}{c}\underline{u}(t)\\ \hline \underline{v}(t)\end{array}\right] = -\left[\begin{array}{c|c} 0 & I \\ \hline \mathcal{O}(\frac{1}{h^2}) & \mathcal{O}(\frac{1}{h})\end{array}\right]\left[\begin{array}{c}\underline{u}(t)\\ \hline \underline{v}(t)\end{array}\right]. \tag{3.4}$$

A similarity transformation of this matrix with $\left[\begin{array}{c|c} I & 0 \\ \hline 0 & hI\end{array}\right]$ makes all elements $\mathcal{O}(\frac{1}{h})$ or less, therefore making $\mathcal{O}(\frac{1}{h})$ also an upper bound for the magnitudes of matrix eigenvalues.

3.4 *A two-dimensional PDE illustration*

A simple two-dimensional generalization of (3.2) is

$$\frac{\partial u}{\partial t} + \alpha\frac{\partial u}{\partial x} + \beta\frac{\partial u}{\partial y} = 0, \tag{3.5}$$

where we let α and β be constants. With $v = u_x$, $w = u_y$, this PDE can be written as

$$\left[\begin{array}{c} u \\ v \\ w \end{array}\right]_t = -\left[\begin{array}{c} \alpha v + \beta w \\ \alpha u_{xx} + \beta u_{xy} \\ \alpha u_{xy} + \beta u_{yy} \end{array}\right],$$

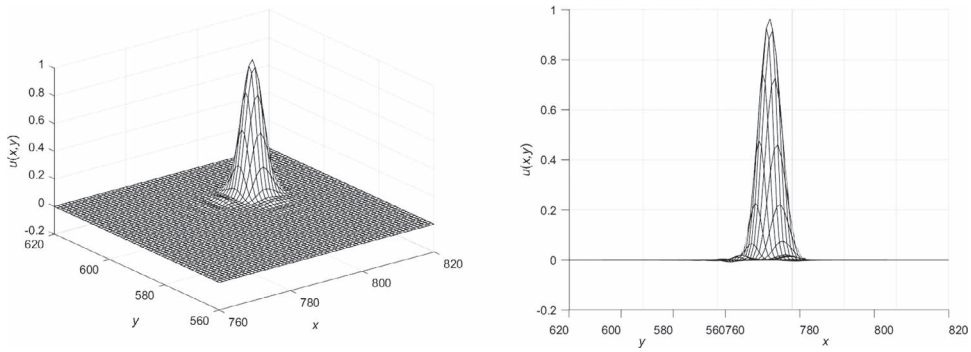

FIG. 4. The numerical solution in the two-dimensional PDE test case after the Gaussian cone has moved a distance of 1000 from its original position at the origin. The displayed grid is the same as (a section of) the computational grid.

where we can approximate the derivatives in the RHS by

$$u_{xx} \approx \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \frac{2}{h^2} u + \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \frac{1}{2h} v, \tag{3.6}$$

$$u_{yy} \approx \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \frac{2}{h^2} u + \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \frac{1}{2h} w, \tag{3.7}$$

$$\begin{bmatrix} 1 & 4 & 1 \end{bmatrix} u_{xy} \approx - \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \frac{3}{2h^2} u + \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} \frac{1}{2h} v + \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \frac{3}{h} w. \tag{3.8}$$

Relations (3.6) and (3.7) are exactly the same as the HFD 4 formula used in the one-dimensional PDE example. Equation (3.8) is of a slightly different form (and can be replaced or alternated with an equivalent one that requires a tridiagonal solve in the y-direction, rather than in the x-direction). By Taylor expansion, it is easily verified to be 6th-order accurate.

We choose as initial condition a rotated version of exactly the same initial condition as in the one-dimensional test: $u(x,y)|_{t=0} = e^{-(0.3^2(x^2+y^2))}$ and then step (3.5) forward (with $\alpha = \frac{4}{5}$, $\beta = \frac{3}{5}$ and space step $h = 1$) until $t = 1000$, at which time the cone will have moved a distance of 1000 in a direction not aligned with the grid, to the position $x = 800$, $y = 600$. Figure 4 shows from two perspectives the numerical solution at this end time, over an x, y-region surrounding this point. The max norm error is comparable to the one-dimensional case: $7.2 \cdot 10^{-2}$ vs. $6.5 \cdot 10^{-2}$ (both using 3-node-wide stencils).

3.5 *Some literature comments*

There already exists a rich literature on CFD approximations, starting with Fox (1947) and Collatz (1960, p. 538), with later works including Fornberg & Ghrist (1999) and Lele (1992). Hermite-type FD methods for approximating wave propagation have been described in several recent works; see for example, Appelö *et al.* (2018) for implementations and references. The approach taken in these works, however, differs fundamentally from the present implementation, with the Hermite polynomials defined by numerous derivatives at just two nodes for each. Another key ingredient in these implementations is node staggering in both space and time.

An application of the present FD and HFD approximations to Euler–Maclaurin expansions is described in Fornberg (2020).

4. Conclusions

Finite difference approximations are ubiquitous in computational mathematics. For many situations (especially when using equispaced grids), tables of weights can be generated once and for all. However, when using noncentered stencils (for example, near boundaries) or when nodes are not equispaced, it is essential to have available convenient, fast and numerically stable algorithms for computing weights. We provide here a generalization of a previous algorithm (Fornberg, 1988), applicable when not only function values but also first derivative values are available (and an arbitrary-order derivative is to be approximated). With this tool, various nonstandard FD methods can be conveniently explored and applied. We have here limited the discussion regarding applications to two simple PDE examples.

References

Appelö, D., Hagstrom, T. & Vargas, A. (2018) Hermite methods for the scalar wave equation. *SIAM J. Sci. Comput.*, **40**, A3902–A3927.

Butcher, J., Corless, R., Gonzales-Vega, L. & Shakoori, A. (2011) Polynomial algebra for Birkhoff interpolants. *Numer. Algorithms*, **56**, 319–347.

Collatz, L. (1960) *The Numerical Treatment of Differential Equations*. Berlin, Göttingen, Heidelberg: Springer.

Fornberg, B. (1988) Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput.*, **51**, 699–706.

Fonberg, B. & Ghrist, M. (1999) Spatial finite difference approximations for wave-type equations. *SIAM J. Numer. Anal.*, **37**, 105–130.

Fornberg, B. (1990) Rapid generation of weights in finite difference formulas. *Numerical Analysis 1989* (D. F. Griffiths & G. A. Watson eds). Harlow, UK: Longman Scientific and Technical, pp. 105–121.

Fornberg, B. (1992) Fast generation of weights in finite difference formulas. *Recent Developments in Numerical Methods and Software for ODEs/DAEs/PDEs* (G. D. Byrne & W. E. Schiesser eds). Singapore: World Scientific, pp. 97–123.

Fornberg, B. (1996) *A Practical Guide to Pseudospectral Methods*. Cambridge, UK: Cambridge University Press.

Fornberg, B. (1998) Calculation of weights in finite difference formulas. *SIAM Rev.*, **40**, 685–691.

Fornberg, B. (2020) Euler–Maclaurin expansions without analytic derivatives. Submitted.

Fornberg, B. & Flyer, N. (2015a) *A Primer on Radial Basis Functions with Applications to the Geosciences*. Philadelphia: SIAM.

Fornberg, B. & Flyer, N. (2015b) Solving PDEs with radial basis functions. *Acta Numer.*, **24**, 215–258.

Fox, L. (1947) Some improvements in the use of relaxation methods for the solution of ordinary and partial differential equations. *Proc. Roy. Soc. A*, **190**, 31–59.

Lele, S. (1992) Compact finite-difference schemes with spectral-like resolution. *J. Comput. Phys.*, **103**, 16–42.

Richardson, L. (1911) The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. R. Soc. A*, **210**, 307–357.

Salzer, H. (1960) New formulas for trigonometric interpolation. *Stud. Appl. Math.*, **39**, 83–96.

Appendix A. MATLAB code

The MATLAB function *weights*:

```
function [c, d, e] = weights (z, x, m)   % Calculates FD weights.
% Input parameters:
%   z location where approximations are to be accurate
%   x row vector with x-coordinates for grid points
%   m highest derivative that we want to find weights for
% Output parameters
%  c, d, e  arrays size (m+1, length (x)), containing (as output) in
%           successive rows the weights for derivatives 0, 1, ..., m.
%           The weights in c are when using f-values only, in d and
%           e for when using both f- and f′values
% ____________________________________________________________________
n=length(x);  c=zeros(m+2,n);  c(2,1)=1;
A=x'-x;  s=sum(1./(A+eye(n)))-1;
b=cumprod([ones(n,1),A],2);  rm=repmat((0:m+1)',1,n-1);
d0=diag(b);  d1(1:n-1)=d0(1:n-1)./d0(2:n);
for i=2:n
    mn=min(i,m+1);
    c(2:mn+1,i)=d1(i-1)*(rm(1:mn, 1).*c(1:mn,i-1)-(x(i-1)-z)...
                *c(2:mn+1,i-1));
    c(2:mn+1,1:i-1)=((x(i)-z)*c(2:mn+1,1:i-1)- ...
                rm(1:mn,1:i-1).*c(1:mn,1:i-1))./(x(i)-x(1:i-1));
end
c2=zeros(m+2,n);  cp=cumprod([1,1:m])';
c(1,:)=[];  cc=c./cp;
for k=1:m+1
    c2(k+1,:)=sum(cc(1:k,:).*cc(k:-1:1,:),1);
end
e=c2(1:m+1,:)-(x-z).*c2(2:m+2,:);  d = c2(2:m+2,:)+2*s.*e;
d=d.*cp;  e=e.*cp;
```

Appendix B. Two weight tables for equispaced centered derivative approximations

Tables B1 and B2 display some weights to be used for centered FD and HFD approximations on unit-spaced grids, approximating derivatives up to orders 4 and 3, respectively. If the grid has spacing h (rather than $h = 1$), the weights need to be adjusted as follows:

- Table B1: All first derivative weights need to be divided by h, all second derivative weights by h^2, etc.
- Table B2: The $f(x_i)$ weights for the second derivative should be divided by h^2, for the third derivative by h^3, etc., with (in all cases) one power of h less for the $f'(x_i)$ weights.

TABLE B1 *Weights for some centered FD formulas on a unit-spaced grid, giving approximations to* $f^{(k)}(x)|_{x=0}$, $k = 0, 1, \ldots, 4$. *This table contains the same information as Fornberg (1996, Table 3.1-1)*

Order of		Approximations at $x = 0$; x coordinates at nodes								
derivative	accuracy	-4	-3	-2	-1	0	1	2	3	4
0	∞					1				
1	2				$-\frac{1}{2}$	0	$\frac{1}{2}$			
	4			$\frac{1}{12}$	$-\frac{2}{3}$	0	$\frac{2}{3}$	$-\frac{1}{12}$		
	6		$-\frac{1}{60}$	$\frac{3}{20}$	$-\frac{3}{4}$	0	$\frac{3}{4}$	$-\frac{3}{20}$	$\frac{1}{60}$	
	8	$\frac{1}{280}$	$-\frac{4}{105}$	$\frac{1}{5}$	$-\frac{4}{5}$	0	$\frac{4}{5}$	$-\frac{1}{5}$	$\frac{4}{105}$	$-\frac{1}{280}$
2	2				1	-2	1			
	4			$-\frac{1}{12}$	$\frac{4}{3}$	$-\frac{5}{2}$	$\frac{4}{3}$	$-\frac{1}{12}$		
	6		$\frac{1}{90}$	$-\frac{3}{20}$	$\frac{3}{2}$	$-\frac{49}{18}$	$\frac{3}{2}$	$-\frac{3}{20}$	$\frac{1}{90}$	
	8	$-\frac{1}{560}$	$\frac{8}{315}$	$-\frac{1}{5}$	$\frac{8}{5}$	$-\frac{205}{72}$	$\frac{8}{5}$	$-\frac{1}{5}$	$\frac{8}{315}$	$-\frac{1}{560}$
3	2			$-\frac{1}{2}$	1	0	-1	$\frac{1}{2}$		
	4		$\frac{1}{8}$	-1	$\frac{13}{8}$	0	$-\frac{13}{8}$	1	$-\frac{1}{8}$	
	6	$-\frac{7}{240}$	$\frac{3}{10}$	$-\frac{169}{120}$	$\frac{61}{30}$	0	$-\frac{61}{30}$	$\frac{169}{120}$	$-\frac{3}{10}$	$\frac{7}{240}$
4	2			1	-4	6	-4	1		
	4		$-\frac{1}{6}$	2	$-\frac{13}{2}$	$\frac{28}{3}$	$-\frac{13}{2}$	2	$-\frac{1}{6}$	
	6	$\frac{7}{240}$	$-\frac{2}{5}$	$\frac{169}{60}$	$-\frac{122}{15}$	$\frac{91}{8}$	$-\frac{122}{15}$	$\frac{169}{60}$	$-\frac{2}{5}$	$\frac{7}{240}$

TABLE B2 *Weights for some centered HFD formulas on a unit-spaced grid. Applied to $f(i)$ and $f'(i)$ values with i integers centered around the origin, they give approximations (at $x = 0$) to $f^{(k)}(x)|_{x=0}$, $k = 0, 1, \ldots, 3$*

Weights for	Order of		Approximations at $x = 0$; x coordinates at nodes								
	derivative	accuracy	-4	-3	-2	-1	0	1	2	3	4
$f(x_i)$	0	∞					1				
	1	∞					0				
	2	4				2	-4	2			
		8			$\frac{7}{54}$	$\frac{64}{27}$	-5	$\frac{64}{27}$	$\frac{7}{54}$		
		12		$\frac{157}{18000}$	$\frac{69}{250}$	$\frac{39}{16}$	$-\frac{49}{9}$	$\frac{39}{16}$	$\frac{69}{250}$	$\frac{157}{18000}$	
		16	$\frac{199}{343000}$	$\frac{11824}{385875}$	$\frac{48}{125}$	$\frac{304}{125}$	$-\frac{205}{36}$	$\frac{304}{125}$	$\frac{48}{125}$	$\frac{11824}{385875}$	$\frac{199}{343000}$
	3	4				$-\frac{15}{2}$	0	$\frac{15}{2}$			
		8			$-\frac{31}{144}$	$-\frac{88}{9}$	0	$\frac{88}{9}$	$\frac{31}{144}$		
		12		$-\frac{167}{18000}$	$-\frac{963}{2000}$	$-\frac{171}{16}$	0	$\frac{171}{16}$	$\frac{963}{2000}$	$\frac{167}{18000}$	
		16	$-\frac{2493}{5488000}$	$-\frac{12944}{385875}$	$-\frac{87}{125}$	$-\frac{1392}{125}$	0	$\frac{1392}{125}$	$\frac{87}{125}$	$\frac{12944}{385875}$	$\frac{2493}{5488000}$
$f'(x_i)$	0	∞					0				
	1	∞					1				
	2	4				$\frac{1}{2}$	0	$-\frac{1}{2}$			
		8			$\frac{1}{36}$	$\frac{8}{9}$	0	$-\frac{8}{9}$	$-\frac{1}{36}$		
		12		$\frac{1}{600}$	$\frac{9}{100}$	$\frac{9}{8}$	0	$-\frac{9}{8}$	$-\frac{9}{100}$	$-\frac{1}{600}$	
		16	$\frac{1}{9800}$	$\frac{32}{3675}$	$\frac{4}{25}$	$\frac{32}{25}$	0	$-\frac{32}{25}$	$-\frac{4}{25}$	$-\frac{32}{3675}$	$-\frac{1}{9800}$
	3	4				$-\frac{3}{2}$	-12	$-\frac{3}{2}$			
		8			$-\frac{1}{24}$	$-\frac{8}{3}$	-15	$-\frac{8}{3}$	$-\frac{1}{24}$		
		12		$-\frac{1}{600}$	$-\frac{27}{200}$	$-\frac{27}{8}$	$-\frac{49}{3}$	$-\frac{27}{8}$	$-\frac{27}{200}$	$-\frac{1}{600}$	
		16	$-\frac{3}{39200}$	$-\frac{32}{3675}$	$-\frac{6}{25}$	$-\frac{96}{25}$	$-\frac{205}{12}$	$-\frac{96}{25}$	$-\frac{6}{25}$	$-\frac{32}{3675}$	$-\frac{3}{39200}$