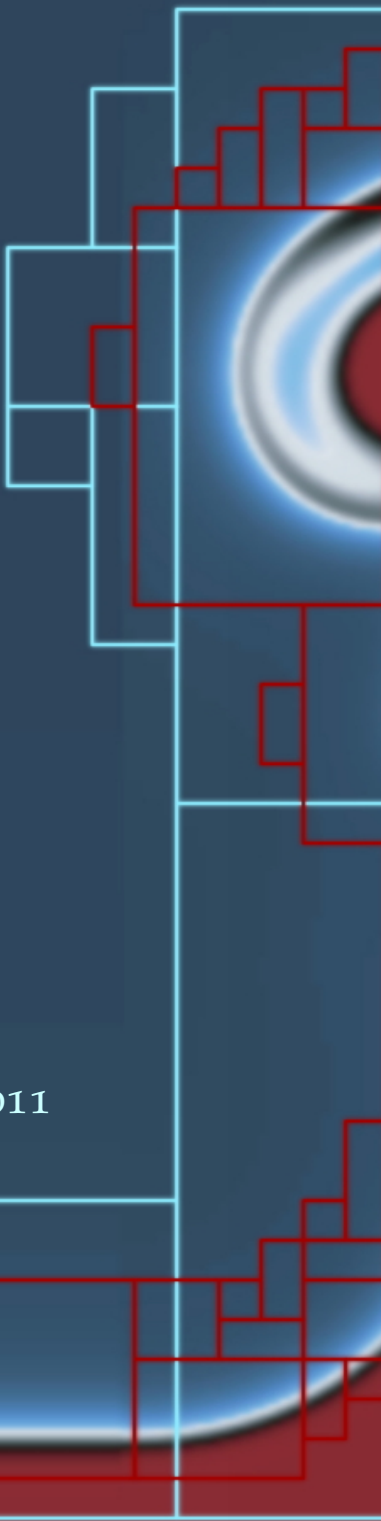
Communications in Applied Mathematics and Computational Science



vol. 6 no. 1 2011

mathematical sciences publishers

Communications in Applied Mathematics and Computational Science

pjm.math.berkeley.edu/camcos

See inside back cover or pjm.math.berkeley.edu/camcos for submission instructions.

The subscription price for 2011 is US $70/year for the electronic version, and $100/year for print and electronic. Subscriptions, requests for back issues from the last three years and changes of subscribers address should be sent to Mathematical Sciences Publishers, Department of Mathematics, University of California, Berkeley, CA 94720-3840, USA.

Communications in Applied Mathematics and Computational Science, at Mathematical Sciences Publishers, Department of Mathematics, University of California, Berkeley, CA 94720-3840 is published continuously online. Periodical rate postage paid at Berkeley, CA 94704, and additional mailing offices.

CAMCoS peer review and production are managed by EditFLOW™ from Mathematical Sciences Publishers.

PUBLISHED BY
mathematical sciences publishers
http://msp.org/
A NON-PROFIT CORPORATION
Typeset in LaTeX

COMM. APP. MATH. AND COMP. SCI.
Vol. 6, No. 1, 2011

msp

A HIGH-ORDER FINITE-VOLUME METHOD FOR CONSERVATION LAWS ON LOCALLY REFINED GRIDS

PETER MCCORQUODALE AND PHILLIP COLELLA

We present a fourth-order accurate finite-volume method for solving time-dependent hyperbolic systems of conservation laws on Cartesian grids with multiple levels of refinement. The underlying method is a generalization of that developed by Colella, Dorr, Hittinger and Martin (2009) to nonlinear systems, and is based on using fourth-order accurate quadratures for computing fluxes on faces, combined with fourth-order accurate Runge–Kutta discretization in time. To interpolate boundary conditions at refinement boundaries, we interpolate in time in a manner consistent with the individual stages of the Runge–Kutta method, and interpolate in space by solving a least-squares problem over a neighborhood of each target cell for the coefficients of a cubic polynomial. The method also uses a variation on the extremum-preserving limiter of Colella and Sekora (2008), as well as slope flattening and a fourth-order accurate artificial viscosity for strong shocks. We show that the resulting method is fourth-order accurate for smooth solutions, and is robust in the presence of complex combinations of shocks and smooth flows.

1. High-order finite-volume methods

In the finite-volume approach, the spatial domain in $\mathbb{R}^D$ is discretized as a union of rectangular control volumes that covers the spatial domain. For Cartesian-grid finite-volume methods, a control volume V_i takes the form

$$V_i = [ih, (i + u)h] \quad \text{for } i \in \mathbb{Z}^D,\ u = (1, 1, \ldots, 1),$$

where h is the grid spacing.

A finite-volume discretization of a partial differential equation is based on averaging that equation over control volumes, applying the divergence theorem to replace volume integrals by integrals over the boundary of the control volume,

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under contract no. DE-AC02-05CH11231.
MSC2000: 65M55.
Keywords: high-order methods, finite-volume methods, adaptive mesh refinement, hyperbolic partial differential equations.

and approximating the boundary integrals by quadratures. In this paper, we solve time-dependent problems that take the form of a conservation equation:

$$\frac{\partial U}{\partial t} + \nabla \cdot \vec{F}(U) = 0. \tag{1}$$

The discretized solution in space is the average of U over a control volume,

$$\langle U \rangle_{\boldsymbol{i}}(t) = \frac{1}{h^{\boldsymbol{D}}} \int_{V_i} U(\boldsymbol{x}, t)\, d\boldsymbol{x}. \tag{2}$$

We can compute the evolution of the spatially discretized system by a method-of-lines approach,

$$\frac{d\langle U \rangle_{\boldsymbol{i}}}{dt} = -\frac{1}{h^{\boldsymbol{D}}} \int_{V_i} \nabla \cdot \vec{F}\, d\boldsymbol{x} = -\frac{1}{h} \sum_d \langle F^d \rangle_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \langle F^d \rangle_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}, \tag{3}$$

$$\langle F^d \rangle_{\boldsymbol{i}\pm\frac{1}{2}\boldsymbol{e}^d} = \frac{1}{h^{\boldsymbol{D}-1}} \int_{A_d^{\pm}} F^d\, dA, \tag{4}$$

where $A_d^{\pm}$ are the high and low faces bounding $V_{\boldsymbol{i}}$ with normals pointing in the $\boldsymbol{e}^d$ direction. In this case, the finite-volume approach computes the average of the divergence of the fluxes on the left side of (4) with the sum of the integrals over faces on the right side, with the latter approximated using some quadrature rule. Such approximations are desirable because they lead to conserved quantities in the original PDE satisfying an analogous conservation law in the discretized system.

The approach we take in this paper is a generalization of the method in [5] to general nonlinear systems of hyperbolic conservation laws on locally refined grids, using fourth-order quadratures in space to evaluate the flux integrals (4) on the faces [1], and a Runge–Kutta method for evolving the ODE (3). We use this approach as the starting point for a block-structured adaptive mesh refinement method along the lines of that in [3].

2. Single-level algorithm

2.1. ***Temporal discretization.*** Given the solution $\langle U \rangle^n \approx \langle U \rangle(t^n)$, we compute a fourth-order temporal update to $\langle U \rangle^{n+1} \approx \langle U \rangle(t^n + \Delta t)$ using the classical fourth-order Runge–Kutta (RK4) scheme on (1). We are solving the autonomous system of ODEs

$$\frac{d\langle U \rangle}{dt} = -D \cdot \vec{F}, \tag{5}$$

$$D \cdot \vec{F} = D \cdot \vec{F}(\langle U \rangle) = \frac{1}{h} \sum_d \langle F^d \rangle_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \langle F^d \rangle_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}.$$

Then, starting with $\langle U\rangle^{(0)} = \langle U\rangle(t^n)$, set

$$k_1 = -D\cdot\vec{F}(\langle U\rangle^{(0)})\Delta t, \tag{6}$$

$$\langle U\rangle^{(1)} = \langle U\rangle^{(0)} + \tfrac{1}{2}k_1, \qquad k_2 = -D\cdot\vec{F}(\langle U\rangle^{(1)})\Delta t, \tag{7}$$

$$\langle U\rangle^{(2)} = \langle U\rangle^{(0)} + \tfrac{1}{2}k_2, \qquad k_3 = -D\cdot\vec{F}(\langle U\rangle^{(2)})\Delta t, \tag{8}$$

$$\langle U\rangle^{(3)} = \langle U\rangle^{(0)} + k_3, \qquad k_4 = -D\cdot\vec{F}(\langle U\rangle^{(3)})\Delta t. \tag{9}$$

Then to integrate one time step:

$$\langle U\rangle(t^n+\Delta t) = \langle U\rangle(t^n) + \tfrac{1}{6}(k_1+2k_2+2k_3+k_4) + O((\Delta t)^5). \tag{10}$$

The method given above is in conservation form. That is,

$$\langle U\rangle^{n+1} = \langle U\rangle^n - \frac{\Delta t}{h}\sum_d \langle F^d\rangle^{\mathrm{tot}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \langle F^d\rangle^{\mathrm{tot}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}, \tag{11}$$

$$\langle F^d\rangle^{\mathrm{tot}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{1}{6}\big(\langle F^d\rangle^{(0)}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + 2\langle F^d\rangle^{(1)}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + 2\langle F^d\rangle^{(2)}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + \langle F^d\rangle^{(3)}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}\big),$$

$$\langle F^d\rangle^{(s)}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \langle F^d(\langle U^{(s)}\rangle)\rangle_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}.$$

2.2. ***Spatial discretization.*** To complete the definition of the single-level algorithm, we need to specify how to compute $\langle F^d\rangle_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$ as a function of $\langle U\rangle$. Our approach generalizes that in [5] to the case of nonlinear systems of conservation laws. Following what often is done for second-order methods, we introduce a nonlinear change of variables $W = W(U)$. In the case of gas dynamics, this is the conversion from the conserved quantities mass, momentum, and energy, $U = (\rho, \rho\vec{u}, \rho E)$, to primitive variables $W = (\rho, \vec{u}, p)$, where ρ is the gas density, $\vec{u}$ is the velocity vector, E is the total energy per unit mass, and p is the pressure. Typically, this transformation is done to simplify the limiting process, for example, to permit the use of component-wise limiting. Some care is required in transforming from conservative to primitive variables in order to preserve fourth-order accuracy.

1. Convert from cell-averaged conserved variables to cell-averaged primitive variables, through cell-centered values, as follows.

   Calculate a fourth-order approximation to U at cell centers:

$$U_{\boldsymbol{i}} = \langle U\rangle_{\boldsymbol{i}} - \frac{h^2}{24}\Delta^{(2)}\langle U\rangle_{\boldsymbol{i}}, \tag{12}$$

   where $\Delta^{(2)}$ is the second-order accurate Laplacian

$$\Delta^{(2)}q_{\boldsymbol{i}} = \sum_d \frac{1}{h^2}(q_{\boldsymbol{i}-\boldsymbol{e}^d} - 2q_{\boldsymbol{i}} + q_{\boldsymbol{i}+\boldsymbol{e}^d}). \tag{13}$$

Then convert to primitive variables:

$$W_{\boldsymbol{i}} = W(U_{\boldsymbol{i}}), \tag{14}$$

$$\overline{W}_{\boldsymbol{i}} = W(\langle U \rangle_{\boldsymbol{i}}). \tag{15}$$

Calculate a fourth-order approximation to cell-averaged W:

$$\langle W \rangle_{\boldsymbol{i}} = W_{\boldsymbol{i}} + \tfrac{1}{24} h^2 \Delta^{(2)} \overline{W}_{\boldsymbol{i}}. \tag{16}$$

2. Interpolate from cell-averaged W to fourth-order face-averaged W over faces in dimension d, by:

$$\langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{7}{12}(\langle W \rangle_{\boldsymbol{i}} + \langle W \rangle_{\boldsymbol{i}+\boldsymbol{e}^d}) - \tfrac{1}{12}(\langle W \rangle_{\boldsymbol{i}-\boldsymbol{e}^d} + \langle W \rangle_{\boldsymbol{i}+2\boldsymbol{e}^d}), \tag{17}$$

for every d-face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$.

3. Calculate face-centered W:

$$W^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \tfrac{1}{24} h^2 \Delta^{d,2} \langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}, \tag{18}$$

where the transverse Laplacian is

$$\Delta^{d,2} q^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \sum_{d' \neq d} \frac{1}{h^2} \big(q^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d - \boldsymbol{e}^{d'}} - 2 q^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + q^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d + \boldsymbol{e}^{d'}} \big). \tag{19}$$

Then compute the face-averaged fluxes in each dimension d:

$$\langle F^d \rangle_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = F^d\big(W^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}\big) + \tfrac{1}{24} h^2 \Delta^{d,2} F^d\big(\langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}\big), \tag{20}$$

for every d-face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$.

Finally, the divergence is computed as in (3).

In Step 1 above, the Laplacian is applied in (16) to $\overline{W}_{\boldsymbol{i}}$ instead of $W_{\boldsymbol{i}}$ in order to minimize the size of stencil required; this substitution makes a difference of $O(h^4)$ in (16) because the discrete Laplacian of (13) is multiplied by h^2. Similarly, in Step 3, $\Delta^{d,2}$ is applied in (20) to $F^d(\langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d})$ instead of to $F^d(W^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d})$, in order to minimize the size of the required stencil without loss of fourth-order accuracy.

2.3. ***Modified stencils near physical boundaries.*** Near physical boundaries, the stencils in the algorithm of Section 2.2 are modified as follows.

In Step 1, in (13), when cell $\boldsymbol{i}$ is adjacent to the physical boundary in dimension d, we substitute for $\boldsymbol{i}$ the appropriate formula at $\boldsymbol{i} \pm \boldsymbol{e}^d$ so that all cells in the stencil are within the domain. Likewise, in Step 3, in (19), when face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ is adjacent to the physical boundary in dimension d', we substitute for $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ the appropriate formula at $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d \pm \boldsymbol{e}^{d'}$ so that all faces in the stencil are within the domain.

In Step 2, the stencil (17) is applied only when face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ is separated by at least two cells from physical boundaries along dimension d. In other cases:

- If face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ lies on, respectively, the low or high physical boundary in dimension d, then

$$\langle W\rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{1}{12}(25\langle W\rangle_{\boldsymbol{i}+\boldsymbol{e}^d} - 23\langle W\rangle_{\boldsymbol{i}+2\boldsymbol{e}^d} + 13\langle W\rangle_{\boldsymbol{i}+3\boldsymbol{e}^d} - 3\langle W\rangle_{\boldsymbol{i}+4\boldsymbol{e}^d})$$

or

$$\langle W\rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{1}{12}(25\langle W\rangle_{\boldsymbol{i}} - 23\langle W\rangle_{\boldsymbol{i}-\boldsymbol{e}^d} + 13\langle W\rangle_{\boldsymbol{i}-2\boldsymbol{e}^d} - 3\langle W\rangle_{\boldsymbol{i}-3\boldsymbol{e}^d}). \tag{21}$$

- If face $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ is separated by a single cell from, respectively, the low or high physical boundary in dimension d, then

$$\langle W\rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{1}{12}(3\langle W\rangle_{\boldsymbol{i}} + 13\langle W\rangle_{\boldsymbol{i}+\boldsymbol{e}^d} - 5\langle W\rangle_{\boldsymbol{i}+2\boldsymbol{e}^d} + \langle W\rangle_{\boldsymbol{i}+3\boldsymbol{e}^d})$$

or

$$\langle W\rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \tfrac{1}{12}(3\langle W\rangle_{\boldsymbol{i}+\boldsymbol{e}^d} + 13\langle W\rangle_{\boldsymbol{i}} - 5\langle W\rangle_{\boldsymbol{i}-\boldsymbol{e}^d} + \langle W\rangle_{\boldsymbol{i}-2\boldsymbol{e}^d}). \tag{22}$$

2.4. ***Limiters.*** For a method of lines such as the one employed here, limiters are used to suppress oscillations in the presence of shocks and underresolved gradients. In one approach, the limiter takes the form of replacing the single-valued solution value at cell faces by two values, each extrapolated from each adjacent cell. This pair of values is used to compute an upwind flux of some sort, such as one obtained by solving a Riemann problem. This is the type of limiter we employ here. We use a variant of the limiter proposed in [8], which is in turn a modification that preserves extrema of the limiter for the piecewise parabolic method (PPM) in [9]. We have modified this limiter in several ways. First, we have made a small change to the method in [8] for detecting extrema that to reduce sensitivity to roundoff error. Second we have modified the limiter to eliminate difficulty that arises in multidimensional problems. To illustrate this problem, consider a solution of the form $f(x, y) = x^3 - xy^2$. This function, for fixed y, has two extrema as a function of x located at $x = \pm y/\sqrt{3}$. It is not difficult to see that, for any fixed h, and all y sufficiently small, but nonzero, the limiter in [8] will be activated at those extrema, thus reducing the accuracy of the method in a region where the function is manifestly smooth enough to be discretized accurately by our underlying fourth-order method. This leads to a failure to converge at fourth-order accuracy in max norm for smooth problems. In order to eliminate this difficulty, we change the criterion by which we decide to apply the limiter in [8] at extrema, so that it is not applied to solutions that are small perturbations of a cubic profile. Finally, we have found that, in introducing the above changes, the fundamental structure of the PPM limiter, at least for the fourth-order Runge–Kutta time discretization used here, introduces too much dissipation. The PPM limiter limits the solution in two parts of the algorithm. The first is in the construction of the single value at the face, which is limited to be within a range defined by the adjacent cell values. The second step in the limiter is based on limiting parabolic profiles in the two cells adjacent to the face, leading to

a potentially double-valued solution at the face. We have found that, in the present setting, the initial limiting of the face values is redundant, and in fact introduces excessive dissipation for linear advection in one dimension, and that the limiting introduced in the second step is sufficient.

We make the following additions to Step 2 in the algorithm of Section 2.2, to apply limiting to $\langle W \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$. For each component w of the primitive variables W:

1. As described in 2.4.1 below, extrapolate $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$ to the left and right of each d-face to obtain $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ and $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$.
2. As described in 2.5.1, apply slope flattening to the extrapolants $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ and $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$.
3. Solve the Riemann problem on faces: From $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ and $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$, get the new $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$.

2.4.1. *Limiter on extrapolants.* We initialize both left and right extrapolated values $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\{\mathrm{L},\mathrm{R}\}}$ to $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$. At each cell $\boldsymbol{i}$, the limiter may change $\langle w \rangle^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$ or $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ or both.

The limiter for extrapolants $\langle w \rangle^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$ and $\langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ depends on $\langle w \rangle$ at cells $\boldsymbol{i}-3\boldsymbol{e}^d$ through $\boldsymbol{i}+3\boldsymbol{e}^d$, as well as the face averages $\langle w \rangle^d_{\boldsymbol{i}\pm\frac{1}{2}\boldsymbol{e}^d}$.

For each cell $\boldsymbol{i}$, set the differences

$$(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}} = \langle w \rangle_{\boldsymbol{i}} - \langle w \rangle^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}, \quad (\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}} = \langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \langle w \rangle_{\boldsymbol{i}}.$$

Also set the differences

$$\begin{aligned}(\delta^2 w)^{d,\mathrm{f}}_{\boldsymbol{i}} &= 6\big(\langle w \rangle^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d} - 2\langle w \rangle_{\boldsymbol{i}} + \langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}\big),\\ (\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}} &= \langle w \rangle_{\boldsymbol{i}-\boldsymbol{e}^d} - 2\langle w \rangle_{\boldsymbol{i}} + \langle w \rangle_{\boldsymbol{i}+\boldsymbol{e}^d},\end{aligned}$$

which approximate the second derivative, multiplied by h^2, at the center of cell $\boldsymbol{i}$.

At each cell face, $\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d$, set the difference

$$(\delta^3 w)^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = (\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}+\boldsymbol{e}^d} - (\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}}, \tag{23}$$

which approximates the third derivative, multiplied by h^3, at the center of face $\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d$.

1. If, at cell $\boldsymbol{i}$, either
$$(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}} \cdot (\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}} \le 0 \tag{24}$$
or
$$(\langle w \rangle^d_{\boldsymbol{i}} - \langle w \rangle^d_{\boldsymbol{i}-2\boldsymbol{e}^d}) \cdot (\langle w \rangle^d_{\boldsymbol{i}+2\boldsymbol{e}^d} - \langle w \rangle^d_{\boldsymbol{i}}) \le 0, \tag{25}$$
then w has an extremum on cell $\boldsymbol{i}$ along dimension d, and we modify
$$\langle w \rangle^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}} \quad \text{and} \quad \langle w \rangle^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$$
as follows.

• If $(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}-\boldsymbol{e}^d}$, $(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}}$, $(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}+\boldsymbol{e}^d}$, and $(\delta^2 w)^{d,\mathrm{f}}_{\boldsymbol{i}}$, all have the same sign, $s = \pm 1$, then set

$$(\delta^2 w)^{d,\lim}_{\boldsymbol{i}} = s \cdot \min\left\{\left|(\delta^2 w)^{d,\mathrm{f}}_{\boldsymbol{i}}\right|, C_2\left|(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}-\boldsymbol{e}^d}\right|, C_2\left|(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}}\right|, C_2\left|(\delta^2 w)^{d,\mathrm{c}}_{\boldsymbol{i}+\boldsymbol{e}^d}\right|\right\}, \tag{26}$$

where $C_2 = 1.25$. Otherwise, set $(\delta^2 w)^{d,\lim}_{\boldsymbol{i}} = 0$.

• If $|(\delta^2 w)^{d,\mathrm{f}}_{\boldsymbol{i}}| \le 10^{-12} \cdot \max\{|w_{\boldsymbol{i}-2\boldsymbol{e}^d}|, |w_{\boldsymbol{i}-\boldsymbol{e}^d}|, |w_{\boldsymbol{i}}|, |w_{\boldsymbol{i}+\boldsymbol{e}^d}|, |w_{\boldsymbol{i}+2\boldsymbol{e}^d}|\}$, then set $\rho_{\boldsymbol{i}} = 0$. Otherwise, set

$$\rho_{\boldsymbol{i}} = \frac{(\delta^2 w)^{d,\lim}_{\boldsymbol{i}}}{(\delta^2 w)^{d,\mathrm{f}}_{\boldsymbol{i}}}. \tag{27}$$

• If $\rho_{\boldsymbol{i}} \ge 1 - 10^{-12}$, a limiter is not applied. Otherwise, to check whether to apply a limiter, set

$$(\delta^3 w)^{d,\min}_{\boldsymbol{i}} = \min\left\{(\delta^3 w)^{d}_{\boldsymbol{i}-(3/2)\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}+(3/2)\boldsymbol{e}^d}\right\},$$
$$(\delta^3 w)^{d,\max}_{\boldsymbol{i}} = \max\left\{(\delta^3 w)^{d}_{\boldsymbol{i}-(3/2)\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}, (\delta^3 w)^{d}_{\boldsymbol{i}+(3/2)\boldsymbol{e}^d}\right\}.$$

A necessary condition for applying a limiter in this case is

$$C_3 \cdot \max\left\{\left|(\delta^3 w)^{d,\min}_{\boldsymbol{i}}\right|, \left|(\delta^3 w)^{d,\max}_{\boldsymbol{i}}\right|\right\} \le (\delta^3 w)^{d,\max}_{\boldsymbol{i}} - (\delta^3 w)^{d,\min}_{\boldsymbol{i}}, \tag{28}$$

where $C_3 = 0.1$. If (28) holds, then:

(a) if $(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}} \cdot (\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}} < 0$, set

$$\langle w\rangle^{d}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}} = \langle w\rangle^{d}_{\boldsymbol{i}} - \rho_{\boldsymbol{i}}(\delta^2 w)^{d,\mathrm{f},-}_{\boldsymbol{i}}, \tag{29}$$
$$\langle w\rangle^{d}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}} = \langle w\rangle^{d}_{\boldsymbol{i}} + \rho_{\boldsymbol{i}}(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}; \tag{30}$$

(b) otherwise, if $|(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}| \ge 2|(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}|$, set

$$\langle w\rangle^{d}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}} = \langle w\rangle^{d}_{\boldsymbol{i}} - 2(1-\rho_{\boldsymbol{i}})(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}} - \rho_{\boldsymbol{i}}(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}; \tag{31}$$

(c) otherwise, if $|(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}| \ge 2|(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}|$, set

$$\langle w\rangle^{d}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}} = \langle w\rangle^{d}_{\boldsymbol{i}} + 2(1-\rho_{\boldsymbol{i}})(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}} + \rho_{\boldsymbol{i}}(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}. \tag{32}$$

2. For cell indices $\boldsymbol{i}$ on which neither (24) nor (25) holds, we modify the extrapolants under the following conditions:

(a) if $|(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}| \ge 2|(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}|$, set

$$\langle w\rangle^{d}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}} = \langle w\rangle^{d}_{\boldsymbol{i}} - 2(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}; \tag{33}$$

(b) if $|(\delta w)^{d,\mathrm{f},+}_{\boldsymbol{i}}| \ge 2|(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}|$, set

$$\langle w\rangle^{d}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}} = \langle w\rangle^{d}_{\boldsymbol{i}} + 2(\delta w)^{d,\mathrm{f},-}_{\boldsymbol{i}}. \tag{34}$$

The differences between this extrapolant limiter and the one in [8, §2.4] are:

- Condition (25) tests for differences two cells away, rather than only one cell away as in [8]. This change reduces the sensitivity of the limiter to roundoff error.
- The third-derivative condition (28) is new. The purpose of this condition is to avoid applying the limiter to small perturbations of a cubic.
- There are new, smoother formulae (31)–(32) to be used instead of (29)–(30) in case (25) holds but (24) does not.
- The second term in the right side of Equations (33) and (34) above replaces a more complicated formula with square roots, in [8, Equation (26)].

2.5. ***Dissipation mechanisms for strong shocks.*** For the case of gas dynamics, it necessary include additional dissipation mechanisms to suppress oscillations at strong shocks. We use the approach in [9; 4] of flattening the interpolated profiles at discontinuities that are too steep, as well as the introduction of a modest artificial viscosity term in the total flux.

2.5.1. *Flattening.* In the algorithm of Section 2.2, at the end of Step 2 we apply slope flattening to the extrapolants. The flattening coefficients are those from [4], where the flattening coefficient for cell $\boldsymbol{i}$ is $\eta_{\boldsymbol{i}}$ (calculated from $\overline{W}$). Then the extrapolants are modified as follows:

- replace $\langle w \rangle^{d,\mathrm{PPM}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}}$ by $\eta_{\boldsymbol{i}} \langle w \rangle^{d,\mathrm{PPM}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d,\mathrm{L}} + (1-\eta_{\boldsymbol{i}})\langle w \rangle_{\boldsymbol{i}}$,
- replace $\langle w \rangle^{d,\mathrm{PPM}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}}$ by $\eta_{\boldsymbol{i}} \langle w \rangle^{d,\mathrm{PPM}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d,\mathrm{R}} + (1-\eta_{\boldsymbol{i}})\langle w \rangle_{\boldsymbol{i}}$.

2.5.2. *Artificial viscosity.* At the end of a full iteration in the algorithm of Section 2.2, we apply an artificial viscosity to $\langle F^d \rangle^{\mathrm{tot}}$ and $\langle U \rangle$. The artificial viscosity has constant parameters α and β.

Take velocity $\vec{u}^n_{\boldsymbol{i}}$, pressure $p^n_{\boldsymbol{i}}$, and density $\rho^n_{\boldsymbol{i}}$, components of $\overline{W}^n_{\boldsymbol{i}}$, from (15).

Calculate the face-centered divergence of the velocity:

$$\lambda^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \frac{1}{h}((u_d)^n_{\boldsymbol{i}+\boldsymbol{e}^d} - (u_d)^n_{\boldsymbol{i}}) + \frac{1}{4h}\sum_{d' \neq d}\left((u_{d'})^n_{\boldsymbol{i}+\boldsymbol{e}^d+\boldsymbol{e}^{d'}} - (u_{d'})^n_{\boldsymbol{i}+\boldsymbol{e}^d-\boldsymbol{e}^{d'}} + (u_{d'})^n_{\boldsymbol{i}+\boldsymbol{e}^{d'}} - (u_{d'})^n_{\boldsymbol{i}-\boldsymbol{e}^{d'}}\right). \quad (35)$$

We then compute the artificial viscosity coefficient $\nu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$ at each face by

$$\nu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = h\lambda^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} \min\left\{\frac{(h\lambda^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d})^2}{(c^{\min})^2_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}\cdot\beta}, 1\right\} \quad (36)$$

at faces where $\lambda^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} < 0$; otherwise, $\nu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}$ is set to zero. Here $c^{\min}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \min\{c_{\boldsymbol{i}}, c_{\boldsymbol{i}+\boldsymbol{e}^d}\}$ and $c_{\boldsymbol{i}} = c(\rho_{\boldsymbol{i}}, p_{\boldsymbol{i}})$ is the speed of sound. The artificial viscosity is then applied as follows:

$$\langle U\rangle^{n+1}_{\boldsymbol{i}} := \langle U\rangle^{n+1}_{\boldsymbol{i}} - \frac{\Delta t}{h}\sum_d(\mu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - \mu^d_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}), \tag{37}$$

$$\mu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \alpha\nu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}(\langle U\rangle^n_{\boldsymbol{i}+\boldsymbol{e}^d} - \langle U\rangle^n_{\boldsymbol{i}}). \tag{38}$$

This is equivalent to incrementing the total flux:

$$\langle F^d\rangle^{\text{tot}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} := \langle F^d\rangle^{\text{tot}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + \mu^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d}.$$

In cases where we use the total flux separately as part of the refluxing algorithm to maintain conservation on locally refined grids, we must make sure that the total fluxes are incremented in such a fashion. In regions of smooth flow, $\lambda = O(1)$, and the artificial viscosity makes an $O(h^4)$ contribution to the total flux, thus preserving fourth-order accuracy. At strong shocks, where the minimum in (36) takes on the value 1, the artificial viscosity reduces to the one used in [9; 4]. In all of the calculations shown here, we have set $\alpha = \beta = 0.3$.

3. Adaptive mesh refinement

We extend the uniform grid discretization to a locally refined, nested grid hierarchy. Our notation follows that in [13]; we review this notation only to the extent that it is needed to describe the algorithm presented here. We start with a family of nested discretizations of a rectangular domain $\{\Gamma^l\}_{l=0}^{l_{\max}}$, $\Gamma^l \subset \mathbb{Z}^{\boldsymbol{D}}$. Each point $\boldsymbol{i} \in \Gamma^l$ represents a control volume of the form $V_{\boldsymbol{i}} = [\boldsymbol{i}h^l, (\boldsymbol{i}+\boldsymbol{u})h^l]$ each with mesh spacing h^l, with $h^l = n^l_{\text{ref}}h^{l-1}$. To relate geometric regions and variables on different levels of the hierarchy to one another, we define a coarsening operator

$$\mathscr{C}_r(\boldsymbol{i}) = \left(\left\lfloor\frac{i_1}{r}\right\rfloor, \ldots, \left\lfloor\frac{i_{\boldsymbol{D}}}{r}\right\rfloor\right),$$

where the notation $\lfloor x\rfloor$ means the largest integer less than or equal to x. We assume that $\mathscr{C}^{-1}_{n^l_{\text{ref}}}(\Gamma^{l-1}) = \Gamma^l$.

At any given time, our computed solution will be defined using

$$\{\Omega^l\}_{l=0}^{l_{\max}}, \quad \Omega^l = \Omega^l(t) \subset \Gamma^l, \quad \mathscr{C}_{n^l_{\text{ref}}}(\Omega^l) \subset \Omega^{l-1}, \quad \Omega^0 = \Gamma^0.$$

We also allow refinement in time, as well as in space, with the assumption that the time steps at successive levels satisfy the condition that $\Delta t^l/\Delta t^{l+1}$ is a positive integer. The sets Ω^l are assumed to satisfy the condition of *proper nesting*, meaning that

$$\mathscr{C}^{-1}_{n^l_{\text{ref}}}(\mathscr{C}_{n^l_{\text{ref}}}(\Omega^l)) = \Omega^l,$$

and that there are at least $s^l > 0$ cells in any direction in Ω^l separating

$$\mathscr{C}_{n^l_{\text{ref}}}(\Omega^{l+1}) \quad \text{and} \quad \mathscr{C}^{-1}_{n^{l-1}_{\text{ref}}}(\Omega^{l-1}) - \Omega^l.$$

In the case of periodic domains, the proper-nesting condition is assumed to hold with respect to the periodic extensions of the grids. For boundaries in nonperiodic directions, we also impose the requirement that cells in $\mathscr{C}_{n^l_{\text{ref}}}(\Omega^{l+1})$ must either be adjacent to the boundary, or at least s^l level-l cells away from the boundary. Our choice of s is based on the requirement that, in order to interpolate ghost-cell values for evaluating the spatial operators described in the previous section, only cells at the next coarser level are required. In the present work,

$$s^l = \left\lceil \frac{5}{n^{l+1}_{\text{ref}}} \right\rceil + 2,$$

where the notation $\lceil x \rceil$ means the smallest integer greater than or equal to x.

The primary dependent variables on each level are defined on the grids at each level,

$$\langle U \rangle^l : \Omega^l \to \mathbb{R}^M.$$

In addition to Ω^l, we will also need values for $\langle U \rangle^l$ on all cells in the stencils required to compute the right side of (3). We will denote the extended solution also by $\langle U \rangle^l$. To advance the solution in time on such a grid hierarchy, we use the explicit time-stepping procedure in [3] (see also [7]) as outlined in Sidebar 1 for the function HyperbolicAdvance.

The only difference between this method and the one in [3], other than our choice of single-level integration method, is the choice of interpolation schemes that are used to compute the values that lie outside Ω^l (the "ghost-cell values" required for Step 1 of HyperbolicAdvance(l)) and are required to evaluate the right side of (3), and to compute the values on newly refined grids upon regridding in Step 4. In the previous work, we use a conservative piecewise-linear interpolation in space for both tasks, along with linear interpolation in time for computing the ghost-cell values. In the present work, we use fourth-order accurate interpolation in space derived using the method of least squares, for both ghost cells and regridding. For computing ghost-cell values, this is combined with a specialized interpolation in time that is closely related to the fourth-order Runge–Kutta method we are using for our single-level time discretization.

We first discuss the computation of the ghost cell values. We assume that, from Step 1 of HyperbolicAdvance($l-1$), we have sufficiently accurate estimates of $\langle U^{l-1} \rangle (t^{l-1})$ and $\langle U^{l-1} \rangle (t^{l-1} + \Delta t^{l-1})$. In order to evaluate the operator $D \cdot \vec{F}$ on Ω^l for the s-th stage of a Runge–Kutta method beginning at time t^l, we first interpolate the solution in time on all cells in Ω^{l-1} that are in the spatial interpolation stencil

HyperbolicAdvance(l)

1. Advance $\langle U\rangle^l$ on Ω^l from time t^l to time $t^l + \Delta t^l$, using the algorithm described in Section 2. For each stage of the RK4 scheme, it is necessary to interpolate a collection of values at cells in $\Gamma^l - \Omega^l$, in order to evaluate the fluxes. In the process of computing the fluxes, we accumulate values in flux registers on faces corresponding to the boundaries of Ω^l and Ω^{l+1}, using the total fluxes $\vec{F}^{\text{tot}}$.

2. Call **HyperbolicAdvance** for the next finer level:

while $t^{l+1} < t^l$
  call HyperbolicAdvance$(l+1)$
end while

3. Synchronize the solution on level l with the solution on the finer levels:

- Fill values of $\langle U\rangle^l$ on $\mathscr{C}_{n^l_{\text{ref}}}(\Omega^{l+1})$ with averages of the solution on the next finer level:
$$\langle U\rangle^l_{\boldsymbol{i}} = \frac{1}{(n^l_{\text{ref}})^{\boldsymbol{D}}} \sum_{\boldsymbol{j}\in\mathscr{C}^{-1}_{n^l_{\text{ref}}}(\{\boldsymbol{i}\})} \langle U\rangle^{l+1}_{\boldsymbol{j}}.$$
- Increment $\langle U\rangle^l$ using flux registers defined on boundary between Ω^{l+1} and Ω^l_{valid}.
- Update time: $t^l := t^l + \Delta t^l$.

4. If necessary, regrid on this level and all finer levels.

end HyperbolicAdvance

Sidebar 1. Pseudocode for adaptive mesh refinement in time algorithm.

for the ghost cells. Then we use those values on level $l-1$ to interpolate values on the level-l cells in $\Gamma^l - \Omega^l$ required to evaluate the fluxes. Only the values on the coarse grid at times t^{l-1} and $t^{l-1} + \Delta t^{l-1}$ are used to interpolate the ghost-cell values.

3.1. ***Coarse-fine interpolation in time.*** For any solution of our autonomous ODE integrated using fourth-order Runge–Kutta, from t^{l-1} to $t^{l-1} + \Delta t^{l-1}$, we can compute all of the derivatives through third order in terms of the stage values $k_1, \ldots, k_4$, using the formula derived by Fok and Rosales [10]. For $0 \le \chi \le 1$:

$$\langle U\rangle(t^{l-1}+\chi\Delta t^{l-1}) = \langle U\rangle^{(0)} + \chi k_1 + \tfrac{1}{2}\chi^2(-3k_1+2k_2+2k_3-k_4) + \tfrac{2}{3}\chi^3(k_1-k_2-k_3+k_4) + O((\Delta t^{l-1})^4), \quad (39)$$

where $\langle U\rangle^{(0)} = \langle U\rangle(t^{l-1})$ is the solution at the beginning of the coarse timestep, and k_1, k_2, k_3, k_4 are as defined in (6)–(9).

Hence the derivatives of $\langle U\rangle$ are

$$\frac{d\langle U\rangle}{dt}(t^{l-1}+\chi\Delta t^{l-1}) = \frac{1}{\Delta t^{l-1}}\big(k_1 + \chi(-3k_1+2k_2+2k_3-k_4) + 2\chi^2(k_1-k_2-k_3+k_4)\big) + O((\Delta t^{l-1})^3), \quad (40)$$

$$\frac{d^2\langle U\rangle}{dt^2}(t^{l-1}+\chi\Delta t^{l-1}) = \frac{1}{(\Delta t^{l-1})^2}\big((-3k_1+2k_2+2k_3-k_4) + 4\chi(k_1-k_2-k_3+k_4)\big) + O((\Delta t^{l-1})^2), \quad (41)$$

$$\frac{d^3\langle U\rangle}{dt^3}(t^{l-1}+\chi\Delta t^{l-1}) = \frac{4}{(\Delta t^{l-1})^3}(k_1-k_2-k_3+k_4) + O(\Delta t^{l-1}). \quad (42)$$

To advance the solution on the level l grid from time t^l to time $t^l+\Delta t^l$, we need to interpolate in time to find fourth-order approximations to $\langle U\rangle^{(0)}$, $\langle U\rangle^{(1)}$, $\langle U\rangle^{(2)}$, $\langle U\rangle^{(3)}$. To compute $\langle U\rangle^{(0)}$, we evaluate (39) at

$$\chi = \frac{t^l - t^{l-1}}{\Delta t^{l-1}}.$$

To find $\langle U\rangle^{(1)}$, $\langle U\rangle^{(2)}$, and $\langle U\rangle^{(3)}$ at fine timestep s, the simplest approach would be to substitute

$$\chi = \frac{t^l + \Delta t^l/2 - t^{l-1}}{\Delta t^{l-1}}, \quad \chi = \frac{t^l + \Delta t^l/2 - t^{l-1}}{\Delta t^{l-1}}, \quad \chi = \frac{t^l + \Delta t^l - t^{l-1}}{\Delta t^{l-1}},$$

respectively, in (39). In the absence of limiters, we found that such a procedure gave fourth-order accurate solution errors. However, when used in conjunction with the limiters, we found that the mismatch between the interpolated values and the intermediate steps in the Runge–Kutta time discretization on the fine grid can trigger the limiters even when the solution is smooth. For that reason, we interpolate ghost values that agree with the intermediate stages of the Runge–Kutta method to $O(\Delta t)^4$.

The fourth-order Taylor expansion of $\langle U\rangle^{(1)}$ is

$$\langle U\rangle^{(1)} = \langle U\rangle^{(0)} + \tfrac{1}{2}\Delta t^l f(\langle U\rangle^{(0)}), \quad (43)$$

while those of $\langle U\rangle^{(2)}$ and $\langle U\rangle^{(3)}$ are

$$\begin{aligned}\langle U\rangle^{(2)} &= \langle U\rangle^{(0)} + \tfrac{1}{2}\Delta t^l f(\langle U\rangle^{(1)}) \\ &= \langle U\rangle^{(0)} + \tfrac{1}{2}\Delta t^l f(\langle U\rangle^{(0)}) + \tfrac{1}{4}(\Delta t^l)^2 \frac{df}{d\langle U\rangle} f(\langle U\rangle^{(0)}) \\ &\qquad + \tfrac{1}{16}(\Delta t^l)^3 \frac{d^2 f}{d\langle U\rangle^2}(f(\langle U\rangle^{(0)}))^2 + O((\Delta t^l)^4), \quad (44)\end{aligned}$$

$$\begin{aligned}\langle U\rangle^{(3)} &= \langle U\rangle^{(0)} + \Delta t^l f(\langle U\rangle^{(2)}) \\ &= \langle U\rangle^{(0)} + \Delta t^l f(\langle U\rangle^{(0)}) + \tfrac{1}{2}(\Delta t^l)^2 \frac{df}{d\langle U\rangle} f(\langle U\rangle^{(1)}) \\ &\qquad + \tfrac{1}{8}(\Delta t^l)^3 \frac{d^2 f}{d\langle U\rangle^2}(f(\langle U\rangle^{(0)}))^2 + O((\Delta t^l)^4) \\ &= \langle U\rangle^{(0)} + \Delta t^l f(\langle U\rangle^{(0)}) + \tfrac{1}{2}(\Delta t^l)^2 \frac{df}{d\langle U\rangle} f(\langle U\rangle^{(0)}) \\ &\quad + \tfrac{1}{8}(\Delta t^l)^3 \left(\frac{d^2 f}{d\langle U\rangle^2}(f(\langle U\rangle^{(0)}))^2 + 2\left(\frac{df}{d\langle U\rangle}\right)^2 f(\langle U\rangle^{(0)}) \right) + O((\Delta t^l)^4). \quad (45)\end{aligned}$$

Here we use the notation $f(\langle U\rangle) = -D\cdot\vec{F}(\langle U\rangle)$, and the derivatives of the vector-valued f with respect to $\langle U\rangle$ are the appropriate Jacobians and Hessians of f. Note that, by the chain rule,

$$\frac{d^2\langle U\rangle}{dt^2} = \frac{df}{dt} = \frac{df}{d\langle U\rangle}\frac{d\langle U\rangle}{dt} = \frac{df}{d\langle U\rangle} f, \quad (46)$$

$$\frac{d^3\langle U\rangle}{dt^3} = \frac{d}{dt}\left(\frac{df}{d\langle U\rangle} f\right) = \frac{d^2 f}{d\langle U\rangle^2} f^2 + \left(\frac{df}{d\langle U\rangle}\right)^2 f. \quad (47)$$

We can approximate these derivatives using the coarse-grid values in (40)–(42). It follows from (44) and (45) that

$$\left(\frac{df}{d\langle U\rangle}\right)^2 f(\langle U\rangle^{(0)}) = \frac{4(f(\langle U\rangle^{(2)}) - f(\langle U\rangle^{(1)}))}{(\Delta t^l)^2} + O(\Delta t^l), \quad (48)$$

which we can also approximate from the coarser-level data as

$$\left(\frac{df}{d\langle U\rangle}\right)^2 f = \frac{4(k_3 - k_2)}{(\Delta t^{l-1})^2} + O(\Delta t^{l-1}). \quad (49)$$

In (43)–(45), the coefficients of the powers of Δt^l, such as $f(\langle U\rangle^{(0)})$ and the derivatives, can all be expressed in terms of derivatives of $\langle U\rangle$ evaluated at $t = t^l$. These in turn are approximated with the formulas (40)–(42), while

$$\left(\frac{df}{d\langle U\rangle}\right)^2 f(\langle U\rangle^{(0)})$$

is approximated using (49). These substitutions result in fourth-order accurate formulas for $\langle U \rangle^{(1)}$, $\langle U \rangle^{(2)}$, and $\langle U \rangle^{(3)}$ in terms of k_1, k_2, k_3, k_4, and $\langle U \rangle^{(0)}$.

3.2. ***Coarse-fine interpolation in space.*** We interpolate $\langle u \rangle^{\mathrm{c}}$, averages over coarse-level cells, to find $\langle u \rangle^{\mathrm{f}}$, averages over fine-level cells.

3.2.1. *Notations.* For each coarse cell indexed by $\boldsymbol{i} \in \mathbb{Z}^{\boldsymbol{D}}$, we use these notations:

- $\mathcal{F}(\boldsymbol{i})$ is the set of fine cells contained within $\boldsymbol{i}$.
- $a_{\boldsymbol{i},\boldsymbol{p}}$ (for $\boldsymbol{p} \in \mathbb{N}^{\boldsymbol{D}}$ such that $\|\boldsymbol{p}\|_1 = \sum_d |p_d| \leq 3$) are the coefficients that will be used for interpolation to $\langle u \rangle^{\mathrm{f}}_{\boldsymbol{k}}$ for all $\boldsymbol{k} \in \mathcal{F}(\boldsymbol{i})$. These will be the coefficients of the Taylor polynomial of degree 3 for u around the center of cell $\boldsymbol{i}$. The number of coefficients for each coarse cell in 2D is 10, and in 3D is 20. The coefficients will be computed from values of $\langle u \rangle^{\mathrm{c}}$.
- $\mathcal{N}(\boldsymbol{i})$ is the set of coarse cells used as a stencil from which to take $\langle u \rangle^{\mathrm{c}}$ in order to find the coefficients $a_{\boldsymbol{i},\boldsymbol{p}}$.

For $\boldsymbol{z} \in \mathbb{R}^{\boldsymbol{D}}$ and $\boldsymbol{p} \in \mathbb{N}^{\boldsymbol{D}}$, we write $\langle \boldsymbol{z}^{\boldsymbol{p}} \rangle^{\mathrm{c}}_{\boldsymbol{j}}$ or $\langle \boldsymbol{z}^{\boldsymbol{p}} \rangle^{\mathrm{f}}_{\boldsymbol{k}}$ to denote the average, respectively, over coarse cell $\boldsymbol{j}$ or fine cell $\boldsymbol{k}$, of

$$\boldsymbol{z}^{\boldsymbol{p}} = \prod_d (z_d^{p_d} - K(p_d)), \tag{50}$$

where

$$K(q) = \begin{cases} \dfrac{2^{-q}}{q+1} & \text{if } q > 0 \text{ and } q \text{ is even,} \\ 0 & \text{otherwise.} \end{cases} \tag{51}$$

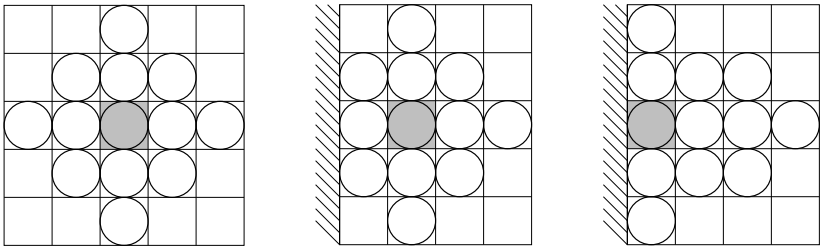
This constant is included to simplify numerical calculations; the average of $\boldsymbol{z}^{\boldsymbol{p}}$ on the cube $\left[-\frac{1}{2}, \frac{1}{2}\right]^{\boldsymbol{D}}$ is 1 if $\boldsymbol{p} = \boldsymbol{0}$, and 0 otherwise.

3.2.2. *Cells in the stencil.* The stencil $\mathcal{N}(\boldsymbol{i})$ for coarse cell $\boldsymbol{i}$ depends on the number of cells between $\boldsymbol{i}$ and the boundary of the domain.

$\mathcal{N}(\boldsymbol{i})$ consists of two sets of cells: an *inner set* and an *outer set*.

- The inner set is centered on a cell $c(\boldsymbol{i})$ that is identical to $\boldsymbol{i}$ if $\boldsymbol{i}$ is separated from the boundary by at least one other cell in every dimension; or if $\boldsymbol{i}$ is adjacent to the boundary, then $c(\boldsymbol{i})$ is one cell away from the boundary in each dimension in which $\boldsymbol{i}$ is adjacent to the boundary. The inner set consists of a square or cube of $3^{\boldsymbol{D}}$ cells with $c(\boldsymbol{i})$ at its center.
- The outer set consists of one cell beyond the inner set in each coordinate direction from $\boldsymbol{i}$ that is in the domain. Hence in every dimension, $\mathcal{N}(\boldsymbol{i})$ contains four or five cells in a row including $\boldsymbol{i}$.

The number of cells in the outer set is at most $2\boldsymbol{D}$, and by the proper-nesting condition, must also be at least $\boldsymbol{D} + 1$. Hence the total number of cells in $\mathcal{N}(\boldsymbol{i})$ in



13 cells, $\kappa = 16.6$ 12 cells, $\kappa = 17.3$ 12 cells, $\kappa = 92.0$

Figure 1. Three examples of 2D stencils, indicated by circles, of coarse cells that are used to interpolate to the fine cells (unmarked) within the shaded coarse cell. Hatching along an edge indicates a physical boundary on that edge. Modulo reflection and permutation of axes, these are all of the stencil possibilities that can arise in 2D. Because of the proper-nesting condition, the coarse cell containing fine ghost cells must be separated by the physical boundary by at least two other coarse cells in at least one of the dimensions. The three possible separations in the other dimension are two or more cells (left diagram), a single cell (middle), and no separation (right). In all cases, the stencil consists of a 3×3 block of cells together with the next cell beyond this block in each coordinate direction from the target cell, as long as this next cell is within the domain. Also shown are the number of cells in each stencil and the condition number of the matrix that converts stencil cell values to the 10 coefficients. Figure 3 shows an instance of each of these stencils being used in a sample set of patches.

2D is either 12 or 13, and in 3D is in the range 31 to 33. Examples of possible stencils $\mathcal{N}(\boldsymbol{i})$ are illustrated in Figure 1 (2D case) and Figure 2 (3D case).

3.2.3. *Calculating fine-cell averages from coarse-cell averages.* To obtain the coefficients $a_{\boldsymbol{i},\boldsymbol{p}}$ for coarse cell $\boldsymbol{i}$, we solve a constrained linear least-squares problem [11, pages 585–586] for the overdetermined system

$$\sum_{\substack{\boldsymbol{p}\in\mathbb{N}^D \\ \|\boldsymbol{p}\|_1\le 3}} a_{\boldsymbol{i},\boldsymbol{p}} \langle (\boldsymbol{x}-\boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle_{\boldsymbol{j}}^{\mathrm{c}} = \langle u \rangle_{\boldsymbol{j}}^{\mathrm{c}}, \quad \text{for all } \boldsymbol{j} \in \mathcal{N}(\boldsymbol{i}) - \{\boldsymbol{i}\}, \tag{52}$$

with the conservation constraint

$$\sum_{\substack{\boldsymbol{p}\in\mathbb{N}^D \\ \|\boldsymbol{p}\|_1\le 3}} a_{\boldsymbol{i},\boldsymbol{p}} \langle (\boldsymbol{x}-\boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle_{\boldsymbol{i}}^{\mathrm{c}} = \langle u \rangle_{\boldsymbol{i}}^{\mathrm{c}}, \tag{53}$$

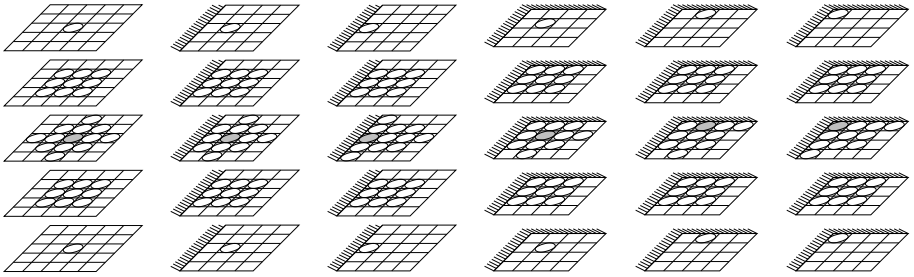


Figure 2. Six examples of 3D stencils, indicated by circles, of coarse cells that are used to interpolate to the fine cells (unmarked) within the shaded coarse cell. Hatching along an edge indicates a physical boundary on that edge. Modulo reflection and permutation of axes, these are all of the stencil possibilities that can arise in 3D. Because of the proper-nesting condition, the coarse cell containing fine ghost cells must be separated by the physical boundary by at least two other coarse cells in at least one of the dimensions. The six stencils shown here represent the possibilities in the other two dimensions for the target cell to be adjacent to the physical boundary or separated by a single cell or by two or more cells. In all cases, the stencil consists of a $3 \times 3 \times 3$ block of cells together with the next cell beyond this block in each coordinate direction from the target cell, as long as this next cell is within the domain. Also shown are the number of cells in each stencil and the condition number of the matrix that converts stencil cell values to the 20 coefficients.

where $\boldsymbol{x}_{\boldsymbol{i}}$ is the center of cell $\boldsymbol{i}$. We then use the coefficients $a_{\boldsymbol{i},\boldsymbol{p}}$ to interpolate for each fine cell $\boldsymbol{k} \in \mathcal{F}(\boldsymbol{i})$:

$$\langle u \rangle^{\mathrm{f}}_{\boldsymbol{k}} = \sum_{\substack{\boldsymbol{p} \in \mathbb{N}^{\boldsymbol{D}} \\ \|\boldsymbol{p}\|_1 \leq 3}} a_{\boldsymbol{i},\boldsymbol{p}} \langle (\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle^{\mathrm{f}}_{\boldsymbol{k}}. \tag{54}$$

The conservation constraint (53) is derived as follows. The average of all interpolated $\langle u \rangle^{\mathrm{f}}$ on fine cells within coarse cell $\boldsymbol{i}$ must equal $\langle u \rangle^{\mathrm{c}}_{\boldsymbol{i}}$. Hence, using (54):

$$\frac{1}{n^{\boldsymbol{D}}_{\mathrm{ref}}} \sum_{\boldsymbol{k} \in \mathcal{F}(\boldsymbol{i})} \sum_{\substack{\boldsymbol{p} \in \mathbb{N}^{\boldsymbol{D}} \\ \|\boldsymbol{p}\|_1 \leq 3}} a_{\boldsymbol{i},\boldsymbol{p}} \langle (\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle^{\mathrm{f}}_{\boldsymbol{k}} = \langle u \rangle^{\mathrm{c}}_{\boldsymbol{i}}. \tag{55}$$

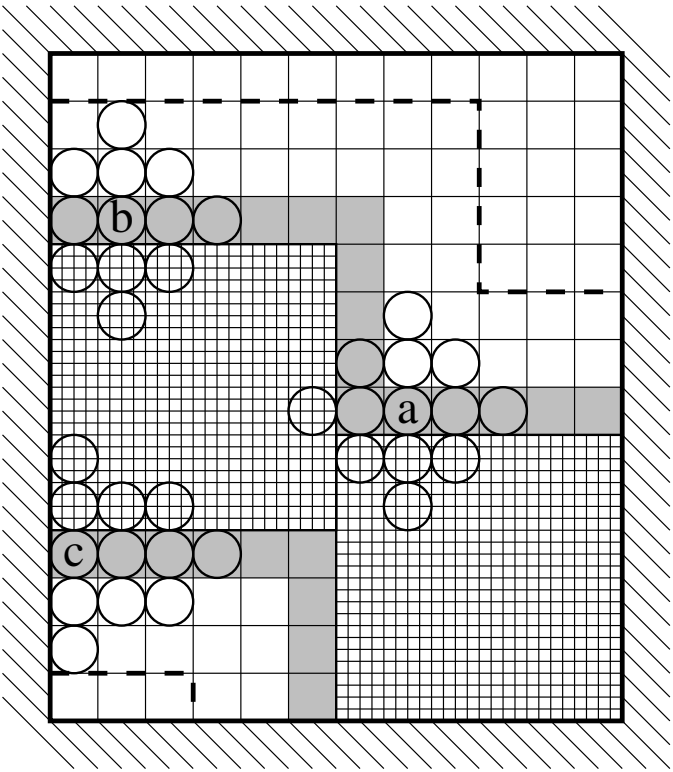


Figure 3. A 2D example of two levels with a refinement ratio of 4 and the coarser level covering the whole rectangular domain, whose boundary is indicated by hatching. Dashed lines mark the limit of coarse cells that are used in stencils to interpolate to fine ghost cells. The shaded coarse cells contain the fine ghost cells that need to be filled in. The letters indicate three such coarse cells where the stencils used are those of Figure 1; the coarse cells of each stencil are marked with circles. Note that the stencil may include coarse cells that are covered by the finer level.

But splitting up coarse cell $\boldsymbol{i}$ into its fine subcells, it is also true that for each $\boldsymbol{p}$,

$$\frac{1}{n_{\text{ref}}^{\boldsymbol{D}}} \sum_{\boldsymbol{k} \in \mathcal{F}(\boldsymbol{i})} \langle (\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle_{\boldsymbol{k}}^{\text{f}} = \langle (\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{i}})^{\boldsymbol{p}} \rangle_{\boldsymbol{i}}^{\text{c}}. \tag{56}$$

Reordering the summation in (55) and making the substitution (56) yields (53).

In 2D, (52) has 10 variables and 11 or 12 equations. In 3D, (52) has 20 variables and 30 to 32 equations. The variables are the coefficients $a_{\boldsymbol{i},\boldsymbol{p}}$ for $\boldsymbol{p} \in \mathbb{N}^{\boldsymbol{D}}$ such that $\|\boldsymbol{p}\|_1 \leq 3$, and in (52) there is one equation for each $\boldsymbol{j} \in \mathcal{N}(\boldsymbol{i}) - \{\boldsymbol{i}\}$.

4. Results

We use this method to solve the 1D advection equation, in order to show results with the new limiter, and then to solve the equations of gas dynamics in 2D and 3D. Unless otherwise stated, the calculations are performed with the full algorithm, that is, with limiters and dissipation mechanisms turned on. For gas-dynamics problems with smooth solutions, we compare our method with that obtained without limiters, indicated here as *limiter off*. We also perform a calculation of a standard shock reflection test problem.

Applying the analysis in [5] to the equations of gas dynamics gives a stability condition for time step Δt and mesh spacing h, of

$$\frac{\Delta t}{h}\sum_d (|\boldsymbol{v}\cdot\boldsymbol{e}^d| + c) \lessapprox 1.3925, \tag{57}$$

where $\boldsymbol{v}$ is velocity and c is the speed of sound. This condition comes from the combination of constraints for the fourth-order Runge–Kutta method in time, and first-order upwinding in space, which is the low-order scheme corresponding to the present method. Note that condition (57) is more restrictive than the one typically used in the method of [4], because there is no analogue of corner coupling that permits use of a larger time step.

4.1. ***1D advection with new limiter.*** We test the algorithm with limiter given in Section 2.4.1 on the 1D advection problem

$$\frac{\partial a}{\partial t} + u\frac{\partial a}{\partial x} = 0, \qquad \text{where } u \text{ is a constant.} \tag{58}$$

We can compare with the exact solution,

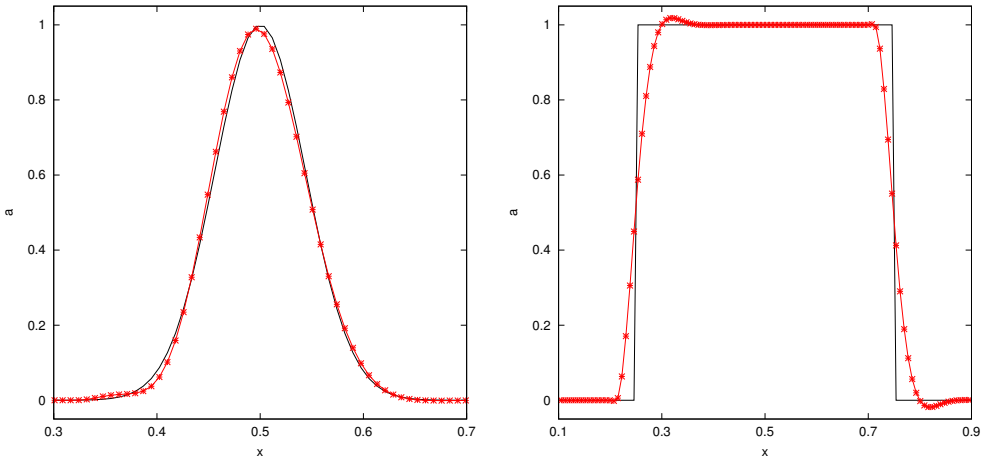
$$a(x,t) = a(x-ut,0). \tag{59}$$

We use the standard 1D test problems:

- Gaussian: $a(x,0) = e^{-256(x-\frac{1}{2})^2}$;
- square wave: $a(x,0) = 1$ if $\left|x-\frac{1}{2}\right| \le \frac{1}{4}$, otherwise 0.

Problem	Norm	1/128	Rate	1/256	Rate	1/512	Rate	1/1024
Gaussian	L_∞	4.03e-02	3.91	2.67e-03	4.01	1.66e-04	4.00	1.04e-05
Gaussian	L_1	4.75e-03	3.99	3.00e-04	3.99	1.88e-05	4.00	1.18e-06
Square wave	L_1	3.26e-02	0.79	1.89e-02	0.79	1.09e-02	0.80	6.29e-03

Table 1. Errors and convergence rates for 1D advection tests with the limiter of Section 2.4.1, at time 10, run with CFL number 0.2. The top row shows the mesh spacing.



Gaussian Square wave

Figure 4. Results using the limiter (red stars) and the exact solution (black curve) tested on 1D advection of a Gaussian (left) or a square wave (right). Both test problems were run on 128 cells and with CFL number 0.2; the results shown are for a at time 10.

All calculations are performed on the unit interval with periodic boundary conditions, advection velocity $u = 1$, and CFL number 0.2. The dissipation mechanisms of Section 2.5 do not apply. Table 1 shows errors and rates of convergence for these test problems. We find that the Gaussian problem exhibits fourth-order convergence. The square-wave problem has a convergence rate of $\frac{4}{5}$ in L_1-norm, as in [8].

Figure 4 shows some results for the two test problems when run with 128 cells.

4.2. ***Gaussian acoustic pulse.*** Our first gas-dynamics example is of a Gaussian acoustic pulse in a polytropic gas, in a periodic domain, $[0, 1]^D$. The initial conditions at a point in this domain are determined by the distance r from the center. Initially the velocity is zero, and the density is

$$\rho(r) = \begin{cases} \rho_0 + (\delta\rho_0)e^{-16r^2}\cos^6(\pi r) & \text{if } r \le \frac{1}{2}, \\ \rho_0 & \text{otherwise;} \end{cases} \tag{60}$$

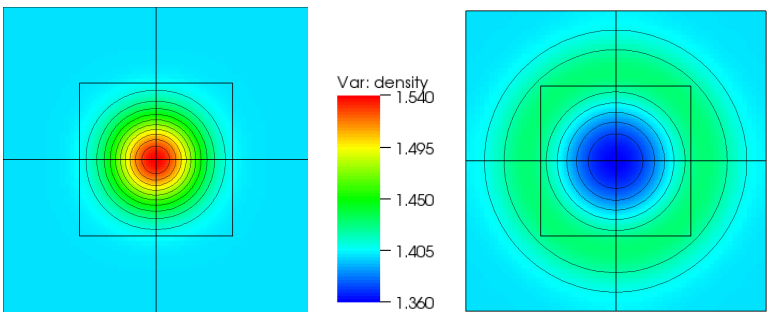
with $\rho_0 = 1.4$ and $\delta\rho_0 = 0.14$. The smoothing factor $\cos^6(\pi r)$ is present to ensure that $\rho = \rho_0$ on the domain boundaries. For isentropicity, the initial pressure is

$$p = \left(\frac{\rho}{\rho_0}\right)^\gamma, \quad \text{where } \gamma = 1.4. \tag{61}$$

We run this example in 2D on a single level, with flattening and artificial viscosity, and both with and without the limiter. Throughout each run, the time step is fixed,

limiter	1/128: 1/256	rate	1/256: 1/512	rate	1/512: 1/1024	rate	1/1024: 1/2048
on	1.32e-06	4.18	7.28e-08	4.01	4.53e-09	3.99	2.85e-10
off	1.15e-06	3.99	7.20e-08	4.00	4.51e-09	4.00	2.82e-10

Table 2. Convergence of differences in calculated density at time 0.24 for 2D Gaussian acoustic pulse, run on a uniform grid, and with the limiter of Section 2.4 either on or off. Columns alternate between showing the max-norm of the difference in densities between results with the indicated mesh spacings, and the convergence rate.



density at time 0 density at time 0.24

Figure 5. Gaussian acoustic pulse in 2D, on two levels.

set to $\Delta t = 0.192h$, where h is the mesh spacing. The results in Table 2 show fourth-order convergence.

We also run this same problem, with and without the limiter, in 2D and 3D on two levels, with a refinement factor of 2 between the levels. Grids at the coarser level cover a cube, and grids at the finer level cover half the length of the cube in each dimension. Figure 5 shows a color plot of density at initial and final times in 2D. Table 3 shows convergence results in 2D and 3D with the limiter either on or off, and indicates fourth-order convergence in all cases.

Finally, we run the 2D problem, with the limiter on, on two levels such that the refinement ratio is 2 and the grids on the finer level are determined adaptively, every two coarse time steps, by refining where $|\nabla\langle\rho\rangle|/\langle\rho\rangle > 0.2h$, with h the coarse-level mesh spacing. Table 4 shows the convergence of differences between results on such two-level adaptive grids and on corresponding uniform one-level grids, where the mesh spacing on the one-level grid is uniformly that on the finer of the two levels in the adaptive case. The truncation error for this method is $O(h^4)$ away from refinement boundaries, and $O(h^3)$ at refinement boundaries. Modified

limiter	1/64: 1/128	rate	1/128: 1/256	rate	1/256: 1/512	rate	1/512: 1/1024
on	7.28e-06	3.97	4.66e-07	3.95	3.01e-08	3.99	1.90e-09
off	7.29e-06	3.97	4.66e-07	3.95	3.01e-08	3.99	1.90e-09

limiter	1/16: 1/32	rate	1/32: 1/64	rate	1/64: 1/128	rate	1/128: 1/256
on	6.84e-04	3.39	6.54e-05	3.69	5.06e-06	3.78	3.70e-07
off	7.35e-04	3.22	7.88e-05	3.80	5.66e-06	3.94	3.69e-07

Table 3. Convergence of differences in calculated density at time 0.24 for 2D (top) and 3D (bottom) Gaussian acoustic pulse, run with fixed grids on two levels, and with the limiter of Section 2.4 either on or off. Columns alternate between showing the max-norm of the difference in densities between results with the indicated mesh spacings at the *coarser* of the two levels, and the convergence rate.

1/128	rate	1/256	rate	1/512	rate	1/1024	rate	1/2048
8.37e-06	3.44	7.69e-07	3.54	6.59e-08	3.73	4.96e-09	3.75	3.69e-10

Table 4. Convergence of differences in density at time 0.24 for 2D Gaussian acoustic pulse, between results calculated on a single-level grid with the indicated uniform mesh spacing, and results calculated on *adaptive* grids on two levels with finer-level mesh spacing as indicated here and with the coarser-level mesh spacing being double that. Columns alternate between showing the max-norm of the difference in densities, and the convergence rate.

equation arguments would indicate that, for adaptive calculations, in which the refinement boundaries are approximately characteristic, we would see a solution error somewhere between third and fourth order in the mesh spacing, in max norm. By combining these results with those in Table 3, top, we obtain a convergence rate that is approximately $O(h^{15/4})$ in max norm, which is consistent with such an analysis.

4.3. ***Shear problem.*** In this 2D polytropic gas problem, we start with constant density $\rho = 1.4$ and pressure $p = 7.$, with initial velocity on the unit square $[0, 1]^2$ set to

$$v_x(x, y) = \cos(2\pi y), \quad v_y(x, y) = \cos(2\pi x).$$

limiter	time interp.	1/64: 1/128	rate	1/128: 1/256	rate	1/256: 1/512	rate	1/512: 1/1024
on	(43)–(45)	1.32e-04	4.05	7.99e-06	3.95	5.17e-07	3.98	3.27e-08
off	(39)	1.13e-04	3.83	7.96e-06	3.92	5.24e-07	3.95	3.39e-08
on	(39)	1.32e-04	3.75	9.78e-06	1.39	3.74e-06	1.59	1.24e-06

Table 5. Convergence of max-norm of calculated differences in x-momentum for 2D shear problem at time 0.15, with limiter on or off, and time interpolation taking $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ either as in Equations (43)–(45) or by substitution of $\chi = (s + \frac{1}{2})/n_{\text{ref}}$, $(s + \frac{1}{2})/n_{\text{ref}}$, $(s + 1)/n_{\text{ref}}$, respectively, in (39).

We run on the same fixed two-level hierarchy as in Section 4.2. Throughout each run, the time step is fixed, with a CFL number of 0.508.

Table 5 shows convergence results with the limiters of Section 2.4 turned either *off* or *on*, and with the time interpolation either as described in Section 3.1 with $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ from Equations (43)–(45), or from substitution of $\chi = (s+\frac{1}{2})/n_{\text{ref}}$, $(s+\frac{1}{2})/n_{\text{ref}}$, and $(s+1)/n_{\text{ref}}$, respectively, in (39). Note that with sufficiently high refinement, the limiter interferes with the time interpolation using substitution in (39), so that convergence is not even second order. But when using that same time interpolation with the limiter turned off, or when using the time interpolation from (43)–(45) with the limiter turned on, convergence is fourth order.

4.4. ***Shock-ramp problem.*** We implement the shock-ramp problem of Woodward and Colella [14], on two levels (refinement ratio of 4 between them), with effective resolution 1024×256. The CFL number is initially 0.3 and is kept to at most 0.8. See Figure 6 for a color plot of the whole domain and Figure 7 for a close-up. The results we obtain here show that the present method has a treatment of

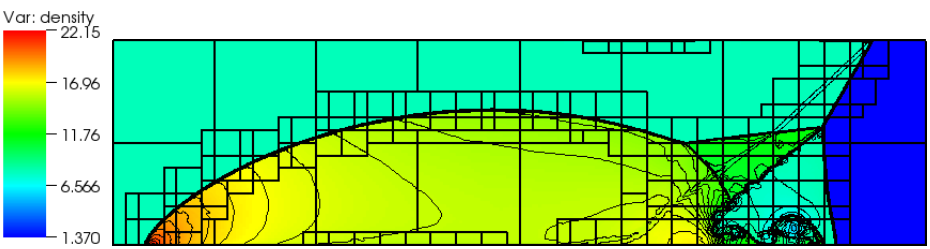


Figure 6. 2D Woodward–Colella shock-ramp problem, with a color plot and contour lines of density, and outlines of the blocks used at the two levels. Figure 7 shows a close-up of this plot.

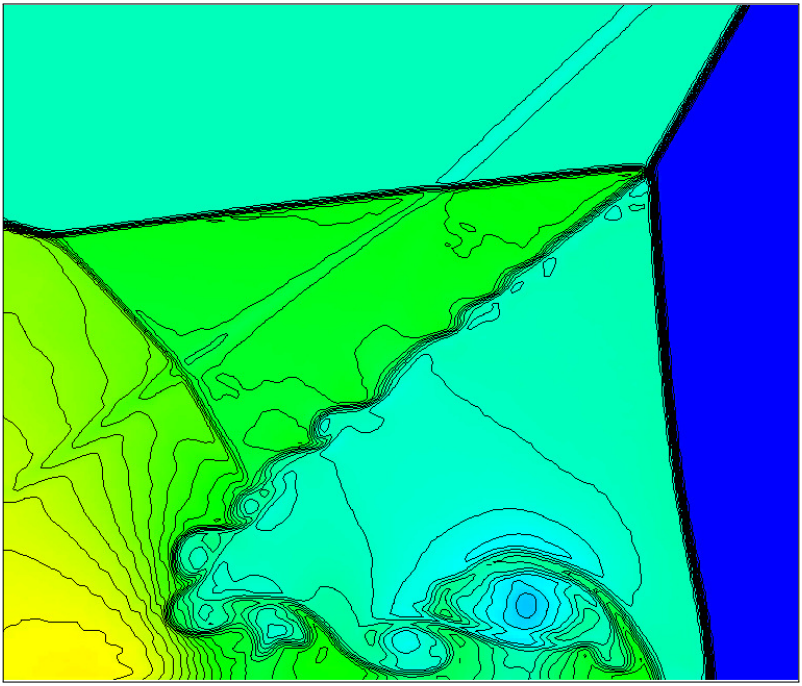


Figure 7. Close-up of Figure 6, showing a color plot and contour lines of density.

multidimensional time-dependent discontinuous flows that is comparable to that of the best state-of-the-art shock-capturing methods.

5. Conclusions

In this paper, we have described an extension of the finite-volume block-structured adaptive mesh refinement algorithm for hyperbolic conservation laws in [3] that is fourth-order accurate in space and time. The underlying single-grid algorithm is an extension of the algorithm in [5] that is comparably accurate and robust to the higher-order Godunov methods for problems involving strong shocks. To achieve this combination of accuracy and robustness, we needed to modify the limiter in [8] to eliminate sensitivity to roundoff error, and to better distinguish smooth extrema that arise in multidimensional problems.

There are a number of directions in which it is natural to extend this algorithm. One is to combine it with the ideas in [5] to compute AMR (adaptive mesh refinement) solutions on mapped grids. This is a key step to the application of this approach to problems such as climate modeling that require mapped-multiblock grids [6]. One essential issue is the extension of the approach in [2] to higher order

using the ideas in [5] so that free-stream preservation is satisfied. Another, less trivial extension is to develop a method analogous to the present one for hyperbolic-parabolic problems that is semiimplicit, treating the hyperbolic terms explicitly, and the parabolic terms implicitly. This has been done for advection-diffusion problems [15] using the fourth-order additive Runge–Kutta method in [12], but only for refinement in space: the same time step is used on all levels. The extension to refinement in time will require the use of an appropriate version of the "dense output" representation for intermediate values described in that paper, analogous to (39) for the explicit Runge–Kutta method used here.

Acknowledgement

We thank Jeff Hittinger, Dan Martin, and Mike Minion for helpful discussions.

References

[1] M. Barad and P. Colella, *A fourth-order accurate local refinement method for Poisson's equation*, J. Comput. Phys. **209** (2005), no. 1, 1–18. MR 2005m:65295 Zbl 1073.65126

[2] J. B. Bell, P. Colella, J. A. Trangenstein, and M. Welcome, *Adaptive mesh refinement on moving quadrilateral grids*, Proceedings of the 9th AIAA Computational Fluid Dynamics Conference, AIAA, June 1989, pp. 471–479.

[3] M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys. **82** (1989), no. 1, 64–84.

[4] P. Colella, *Multidimensional upwind methods for hyperbolic conservation laws*, J. Comput. Phys. **87** (1990), no. 1, 171–200. MR 91c:76087 Zbl 0694.65041

[5] P. Colella, M. Dorr, J. Hittinger, and D. F. Martin, *High-order, finite-volume methods in mapped coordinates*, J. Comput. Phys. **230** (2011), no. 8, 2952–2976.

[6] P. Colella, M. Dorr, J. Hittinger, P. McCorquodale, and D. F. Martin, *High-order finite-volume methods on locally-structured grids*, Numerical modeling of space plasma flows: ASTRONUM 2008, Astronomical Society of the Pacific Conference Series, no. 406, 2008, pp. 207–216.

[7] P. Colella, D. T. Graves, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. McCorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. V. Straalen, *Chombo software package for amr applications - design document*, 2009.

[8] P. Colella and M. D. Sekora, *A limiter for PPM that preserves accuracy at smooth extrema*, J. Comput. Phys. **227** (2008), no. 15, 7069–7076. MR 2009d:76079 Zbl 1152.65090

[9] P. Colella and P. R. Woodward, *The piecewise parabolic method (PPM) for gas dynamical simulations*, J. Comput. Phys. **54** (1984), 174–201.

[10] P.-W. Fok and R. R. Rosales, *Multirate integration of axisymmetric step-flow equations*, (2008), submitted to *J. Comp. Phys.* arXiv 0810.2517v1

[11] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3rd ed., Johns Hopkins Studies in Math. Sciences, Johns Hopkins University Press, Baltimore, 1996. MR 97g:65006 Zbl 0865.65009

[12] C. A. Kennedy and M. H. Carpenter, *Additive Runge–Kutta schemes for convection-diffusion-reaction equations*, Appl. Numer. Math. **44** (2003), no. 1-2, 139–181. MR 2003m:65111 Zbl 1013.65103

[13] D. F. Martin, P. Colella, and D. Graves, *A cell-centered adaptive projection method for the incompressible Navier–Stokes equations in three dimensions*, J. Comput. Phys. **227** (2008), no. 3, 1863–1886. MR 2009g:76085 Zbl 1137.76040

[14] P. Woodward and P. Colella, *The numerical simulation of two-dimensional fluid flow with strong shocks*, J. Comput. Phys. **54** (1984), no. 1, 115–173. MR 85e:76004 Zbl 0573.76057

[15] Q. Zhang, H. Johansen, and P. Colella, *A fourth-order accurate finite-volume method with structured adaptive mesh refinement for solving the advection-diffusion equation*, preprint, 2010, submitted to *SIAM J. Sci. Comp.*

Received June 4, 2010. Revised November 12, 2010.

PETER MCCORQUODALE: PWMcCorquodale@lbl.gov
Lawrence Berkeley National Laboratory, 1 Cyclotron Road, MS 50A-1148, Berkeley CA 94720, United States

PHILLIP COLELLA: PColella@lbl.gov
Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory, 1 Cyclotron Road MS 50A-1148, Berkeley CA 94720, United States

COMM. APP. MATH. AND COMP. SCI.
Vol. 6, No. 1, 2011

msp

AN UNSPLIT, HIGHER-ORDER GODUNOV METHOD USING QUADRATIC RECONSTRUCTION FOR ADVECTION IN TWO DIMENSIONS

SANDRA MAY, ANDREW NONAKA, ANN ALMGREN AND JOHN BELL

Linear advection of a scalar quantity by a specified velocity field arises in a number of different applications. Important examples include the transport of species and energy in low Mach number models for combustion, atmospheric flows and astrophysics, and contaminant transport in Darcy models of saturated subsurface flow. In this paper, we present a customized finite volume advection scheme for this class of problems that provides accurate resolution for smooth problems while avoiding undershoot and overshoot for nonsmooth profiles. The method is an extension of an algorithm by Bell, Dawson and Shubin (BDS), which was developed for a class of scalar conservation laws arising in porous media flows in two dimensions. The original BDS algorithm is a variant of unsplit, higher-order Godunov methods based on construction of a limited bilinear profile within each computational cell. The new method incorporates quadratic terms in the polynomial reconstruction, thereby reducing the L^1 error and better preserving the shape of advected profiles while continuing to satisfy a maximum principle for constant coefficient linear advection. We compare this new method to several other approaches, including the bilinear BDS method and unsplit piecewise parabolic (PPM) methods.

1. Introduction

The focus of much of the literature on numerical methods for hyperbolic partial differential equations is on general systems of conservation laws, particularly the compressible Euler equations (see [14] for an overview of the literature). However, there are a number of important problems in science and engineering where we need to solve linear advection problems of the form

$$s_t + (us)_x + (vs)_y = 0, \tag{1}$$

where $s = s(x, y, t)$ is a scalar field and (u, v) represents a known velocity field. One important example of this type of problem arises in projection algorithms for incompressible and other low Mach number flows, where the velocity field used

MSC2000: 35-04, 35L65.
Keywords: Godunov method, scalar conservation law, two-dimensional quadratic reconstruction.

for advection is constructed during the time step using a projection that enforces the divergence constraint. Applications of these low Mach number projection algorithms include low Mach number terrestrial combustion [11], nuclear flame simulation [4], low Mach number stratified atmospheric [23] and astrophysical flows [19], as well as general variable-density incompressible flow [2]. In these cases the density, species, and other scalar quantities are advected by a velocity field that is calculated before the advection step is performed. Advection problems also arise in contaminant transport in saturated groundwater flow [20]. We note that in several of the above problems, the full evolution equation for s often includes a right hand side representing reactions, diffusion or other processes. However, discretization approaches typically separate the computation of the advective flux from the treatment of the other terms. In particular, diffusion is typically treated in a form in which explicit hyperbolic fluxes appear as source terms in an implicit discretization of diffusion; reactions are typically included via operator splitting. Consequently, here we will focus on the homogeneous system; the reader is referred to the literature cited above for discussion of how to incorporate other processes.

There are several aspects of the class of problems we are considering that are worth noting. First, in most of these applications the velocity field is determined by solving a constraint equation that explicitly encapsulates a specific discrete form of the divergence of the velocity field with which we want the hyperbolic discretization to be consistent. Furthermore, we only have a limited characterization of the velocity field, typically integral averages of the normal component on edges of grid cells. Another aspect of the class of problems being considered is that they can be highly sensitive to overshoot and undershoot. For example, many chemical reaction systems are ill-defined when a species has a negative concentration. Similarly, although harder to detect, errors associated with overshoot in a species concentration can be significantly enhanced by the kinetics mechanism. Thus, we would like a method that provides an accurate discretization and preserves the shape of advected profiles while avoiding overshoot and undershoot. One final consequence of the type of problems we consider is that the computational cost is dominated by elliptic solvers, reaction networks, and/or calls to the equation of state, so the overall cost of advection is minor in comparison; thus accuracy is of more importance than cost in choosing the advection algorithm.

There is a vast literature on numerical methods for first-order hyperbolic partial differential equations, all of which can potentially be adapted to advection by a known velocity field. It is beyond the scope of this paper to survey all of that work; however, we will briefly describe some of the main themes underlying some of these approaches. We first note that dimensional operator splitting does not work well for advection by a nonconstant divergence-free velocity field. In a dimensionally split

approach, the fluid can experience an artificial compression in one sweep combined with an artificial expansion in another sweep, which can lead to significant artifacts. An example showing these types of artifacts is presented in [1]. Thus, we restrict ourselves here to unsplit discretizations.

The first unsplit second-order Godunov method, based on linear reconstruction, was presented by Colella [7], and was later extended to three dimensions by Saltzman [21]. Colella [7] motivated the development of the unsplit Godunov algorithm by introduction of the corner transport upwind (CTU) method. The CTU method is a first-order upwind advection scheme that incorporates diagonal coupling based on a piecewise-constant approximation and the geometry of characteristics for constant coefficient advection. However, the geometric interpretation was abandoned in the extension to general systems of conservation laws. Miller and Colella [17] developed a version of the unsplit scheme based on the piecewise parabolic method (PPM) of Colella and Woodward [9]. This approach uses the same formalism as the piecewise linear algorithms but constructs a parabolic rather than a linear profile in each coordinate direction. There has been some recent work aimed at improving the limiters for PPM. Colella and Sekora [8] developed a new PPM limiter that preserves accuracy at smooth extrema but suffers from sensitivity to roundoff error; more recently McCorquodale and Colella [16] introduced an improvement to that limiter which is less sensitive to roundoff error (P. Colella, private communication, 2010).

LeVeque [13] introduced higher-order advection schemes based on geometric ideas derived from a wave propagation perspective. The WAF approach of Billett and Toro [5] and the Mot-ICE-P1 scheme of Noelle [18] use similar geometric ideas and, in fact, share a number of features of the scheme that will be our starting point. Smolarkiewicz and collaborators developed multidimensional advection schemes for geophysical flows based on flux-corrected transport ideas; see [23] and the references cited therein. Another class of schemes is the ADER-type schemes developed by Toro and collaborators; see, for example, [24]. These schemes are somewhat more algebraic in their construction, using a Cauchy–Kowalewski procedure and Taylor series expansion to evaluate approximations at quadrature nodes on space-time edges of cells. Another class of schemes that has become popular for a wide range of problems is WENO-type schemes. The reader is referred to [22] for a general discussion of these types of methods. Of particular interest for multidimensional advection are unsplit, multidimensional versions of WENO such as those by Levy, Puppo, and Russo [15] and Kurganov and Petrova [12]. A final category of schemes is discontinuous Galerkin finite element methods. There have been recent special issues of journals focused on discontinuous Galerkin; see [10; 6]. Unlike the finite volume schemes discussed above, discontinuous Galerkin methods advance an entire polynomial representation in time.

Although all of the above literature is applicable to linear advection, most of these methods are designed for more general conservation laws. One approach to solving (1) is simply to adapt a method for general systems to the special case considered here. However, we wish to exploit the special structure of the linear advection problem to design a finite volume method that best meets the targets of accuracy and shape preservation without overshoot or undershoot and fits the existing constraints in terms of specification of the velocity field.

The method presented here is an extension of the two-dimensional, higher order scheme for linear advection and scalar conservation laws developed by Bell, Dawson and Shubin [3]. It exploits the observation that the equation is (trivially) diagonalizable and bases the construction of the fluxes on the detailed geometry of the characteristics. For constant coefficient advection, the Bell, Dawson and Shubin (BDS) scheme is numerically equivalent to fitting a profile within each cell, analytically advecting the reconstructed solution and averaging the solution onto the grid. We note also that both the method presented in this paper and the original BDS algorithm are fully explicit in time, and do not require a Runge–Kutta procedure.

This original BDS algorithm constructs a limited bilinear profile within each cell. The method presented here extends the BDS approach by constructing a limited, two-dimensional biquadratic representation of the solution within each cell. The two key elements of the algorithm are the construction of the limited quadratic profile and the modification of the quadrature rules used to compute the fluxes. In the next section, we review the original BDS algorithm. We then discuss the modifications needed to include the quadratic terms in the reconstruction and how to modify the flux computation to account for those terms. Next we present computational results comparing the quadratic BDS algorithm with the original BDS algorithm and two variations of the unsplit PPM algorithm. The initial tests are for advection by a prescribed velocity field. We then illustrate the performance of the algorithm for advection of density and a tracer in a variable density projection algorithm. Finally we present comparisons with some alternative schemes that have been discussed in the literature along with some discussions of their characteristics.

2. Bilinear BDS method

2.1. ***Overview.*** The BDS method was originally developed for scalar conservation laws that arise in porous media flow, of the form

$$s_t + [uf(s)]_x + [vg(s)]_y = qh(s), \tag{2}$$

where (u, v) represents a spatially dependent velocity field and the right side represents point sources and sinks of fluid of strength $q(x, y)$ and composition h.

The fluid was assumed to be incompressible; that is,

$$u_x + v_y = 0 \tag{3}$$

away from the support of q. A detailed description of the method can be found in [3]. Since our focus here is on linear advection in low Mach number models, we will restrict our consideration to the case where $f(s) = g(s) = s$ and not consider sources or sinks of fluid, so that $q = 0$. Although the test cases will all consider a divergence-free velocity field so that the equation satisfies a maximum principle, we will not make any assumptions about the divergence of the velocity in the specification of the algorithm. We summarize the original BDS method in three steps:

Step I. Construct an appropriately limited bilinear polynomial representation of s at time t^n in each cell (i, j).

Step II. Define edge values $s_{i+1/2,j}$, $s_{i,j+1/2}$, etc., by integrating over time and space assuming the bilinear profile.

Step III. Update the solution at time t^{n+1} using a conservative update,

$$\begin{aligned} s_{ij}^{n+1} = s_{ij}^n &- \frac{\Delta t}{\Delta x}(u_{i+1/2,j}s_{i+1/2,j} - u_{i-1/2,j}s_{i-1/2,j}) \\ &- \frac{\Delta t}{\Delta y}(v_{i,j+1/2}s_{i,j+1/2} - v_{i,j-1/2}s_{i,j-1/2}). \end{aligned} \tag{4}$$

Here Δt is the time step and Δx and Δy are the mesh spacings in the x- and y-directions, respectively.

In the next section we describe the construction of the bilinear polynomial; following that we discuss how to construct the face values by integrating over time and space.

2.2. ***Construction of the bilinear polynomial.*** Here, we describe an algorithm for Step I, the construction of a bilinear polynomial representation of s at time t^n, written in the form

$$p_{ij}^{\mathrm{bl}}(x, y) = s_{xy,ij}\,(x - x_i)(y - y_j) + s_{x,ij}\,(x - x_i) + s_{y,ij}\,(y - y_j) + \hat{s}, \tag{5}$$

where (x_i, y_j) denotes the cell center of cell (i, j).

To obtain estimates for the corner values on each cell, a multidimensional analog of the procedure used by Colella and Woodward [9] was chosen. For equally spaced grids this leads to

$$\begin{aligned} s_{i+1/2,j+1/2} = \big[&s_{i-1,j-1} - 7(s_{i,j-1} + s_{i+1,j-1}) + s_{i+2,j-1} \\ &-7s_{i-1,j} + 49(s_{ij} + s_{i+1,j}) - 7s_{i+2,j} \\ &-7s_{i-1,j+1} + 49(s_{i,j+1} + s_{i+1,j+1}) - 7s_{i+2,j+1} \\ &+s_{i-1,j+2} - 7(s_{i,j+2} + s_{i+1,j+2}) + s_{i+2,j+2}\big]/144. \end{aligned} \tag{6}$$

The estimates for the four corner values LL, LH, RL, and RH (left low, left high, right low, and right high) are then used to calculate slopes on the cell (i, j):

$$s_{x,ij} = \frac{(\mathrm{RH} + \mathrm{RL}) - (\mathrm{LH} + \mathrm{LL})}{2\Delta x}, \tag{7a}$$

$$s_{y,ij} = \frac{(\mathrm{LH} + \mathrm{RH}) - (\mathrm{LL} + \mathrm{RL})}{2\Delta y}, \tag{7b}$$

$$s_{xy,ij} = \frac{(\mathrm{RH} - \mathrm{RL}) - (\mathrm{LH} - \mathrm{LL})}{\Delta x \Delta y}. \tag{7c}$$

The constant term $\hat{s}$ is given by s_{ij}. Note that the integral over the linear and bilinear terms vanishes because the polynomial is centered at the cell center. So by construction, the average value of the polynomial over the cell (i, j) equals the cell value s_{ij}.

Remark. The presence of the bilinear term $s_{xy,ij}$ leads to an improved preservation of shapes for off-axis movement compared to methods which only include linear terms (and possibly pure quadratic terms) in their profile reconstruction.

2.3. ***Limiting the bilinear polynomial.*** As noted in [3], the limiting of (5) can be cast as an optimization problem: minimize, in each cell, the L^2 norm of the difference between the limited polynomial and the original interpolation function given by (7a)–(7c) subject to two constraints:

(1) The average of the polynomial evaluated at the four corners of cell (i, j) must equal the cell average s_{ij}.

(2) The polynomial evaluated at a corner must lie between the minimum and the maximum of the cell averages of the four cells surrounding the corner.

However, to reduce the computational cost a simple heuristic algorithm was developed that produced results within 10% of the results obtained from the minimization procedure in terms of overall L^1 error in a variety of test cases. This heuristic algorithm goes as follows:

Step i. Compute the values of the bilinear polynomial at the cell corners:

$$\mathrm{LL}_{\mathrm{temp}} = s_{ij} - \frac{\Delta x}{2} s_{x,ij} - \frac{\Delta y}{2} s_{y,ij} + \frac{\Delta x}{2}\frac{\Delta y}{2} s_{xy,ij}, \tag{8a}$$

$$\mathrm{LH}_{\mathrm{temp}} = s_{ij} - \frac{\Delta x}{2} s_{x,ij} + \frac{\Delta y}{2} s_{y,ij} - \frac{\Delta x}{2}\frac{\Delta y}{2} s_{xy,ij}, \tag{8b}$$

$$\mathrm{RL}_{\mathrm{temp}} = s_{ij} + \frac{\Delta x}{2} s_{x,ij} - \frac{\Delta y}{2} s_{y,ij} - \frac{\Delta x}{2}\frac{\Delta y}{2} s_{xy,ij}, \tag{8c}$$

$$\mathrm{RH}_{\mathrm{temp}} = s_{ij} + \frac{\Delta x}{2} s_{x,ij} + \frac{\Delta y}{2} s_{y,ij} + \frac{\Delta x}{2}\frac{\Delta y}{2} s_{xy,ij}. \tag{8d}$$

Step ii. Check to see if each temporary value is in the range defined by the four neighboring cell values, for example, check whether $\mathrm{LL}_{\mathrm{temp}}$ lies between $\min_{\mathrm{LL}}$ and $\max_{\mathrm{LL}}$, where these limits are defined by

$$\min_{\mathrm{LL}} = \min(s_{i-1,j-1}, s_{i,j-1}, s_{i-1,j}, s_{ij}), \tag{9a}$$

$$\max_{\mathrm{LL}} = \max(s_{i-1,j-1}, s_{i,j-1}, s_{i-1,j}, s_{ij}). \tag{9b}$$

If all of the temporary values $\mathrm{LL}_{\mathrm{temp}}, \ldots, \mathrm{RH}_{\mathrm{temp}}$ happen to lie between their respective bounds, the polynomial does not need to be limited. In that case, skip Steps iii and iv, and keep the original values for s_x, s_y, and s_{xy} as computed in equations (7a)–(7c). Otherwise, constrain these temporary values so they do not introduce any new extrema, for example, set

$$\mathrm{LL}_{\mathrm{temp}} = \max[\min(\mathrm{LL}_{\mathrm{temp}}, \max_{\mathrm{LL}}), \min_{\mathrm{LL}}]. \tag{10}$$

Step iii. Iterative Loop:

(a) Compute the difference between the sum of the temporary values and the cell average, s_{ij}, multiplied by four:

$$\mathrm{sumdif} = (\mathrm{LL}_{\mathrm{temp}} + \mathrm{LH}_{\mathrm{temp}} + \mathrm{RL}_{\mathrm{temp}} + \mathrm{RH}_{\mathrm{temp}}) - 4s_{ij}. \tag{11}$$

Assume for now that sumdif ≥ 0; the case where sumdif ≤ 0 is analogous.

(b) Find out which temporary corner values are larger (smaller) than s_{ij} by more than $\epsilon = 10^{-10}$. Let kdp be the number of corners with this property.

(c) *Loop over corners:* If the temporary corner value is larger than s_{ij} by more than $\epsilon = 10^{-10}$, make the following assignments (using corner $\mathrm{LL}_{\mathrm{temp}}$ as an example assuming that $\mathrm{LL}_{\mathrm{temp}}$ fulfills the criterion in (b)):

- redfac $\leftarrow \min[\mathrm{sumdif}/\mathrm{kdp}, \mathrm{LL}_{\mathrm{temp}} - \min(s_{i-1,j-1}, s_{i,j-1}, s_{i-1,j}, s_{ij})]$
- kdp $\leftarrow$ kdp $- 1$
- sumdif $\leftarrow$ sumdif $-$ redfac
- $\mathrm{LL}_{\mathrm{temp}} \leftarrow \mathrm{LL}_{\mathrm{temp}} -$redfac

Step iv. Compute the final slopes following equations (7a)–(7c), using $\mathrm{LL}_{\mathrm{temp}}$, etc., rather than LL, etc.

Remark. Numerical tests have shown that three iterations are sufficient to complete the limiting process in Step iii.

2.4. ***Construction of edge states.*** The other key part of the BDS algorithm is the calculation of the edge states $s_{i+1/2,j}$, $s_{i,j+1/2}$, etc., in Step II that are used to construct the update terms in (4) in Step III. For clarity of exposition, we focus on the problem

$$s_t + us_x + vs_y = 0, \quad u, v > 0 \text{ constant}; \tag{12}$$

the extension to spatially varying (u, v) will be described later.

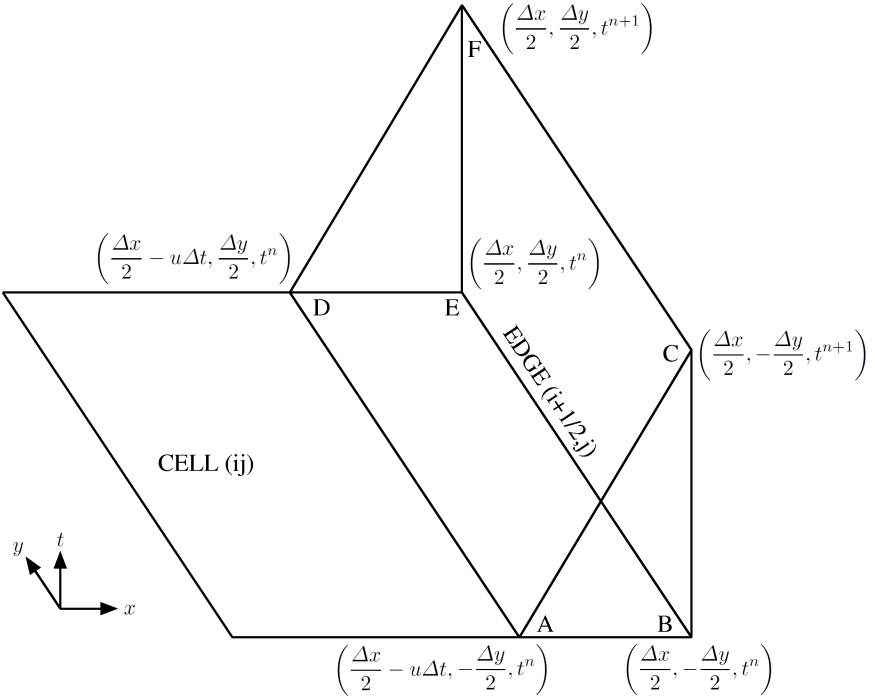


Figure 1. Characteristic domain of dependence of edge $(i+1/2,\, j)$. x- and y-coordinates specified relative to block center. Figure taken from [3].

2.4.1. *Constant velocity.* Since $u > 0$, the characteristic domain of dependence of edge $(i+1/2,\, j)$ is the space-time region $ABCDEF$ as depicted in Figure 1. Thus, to compute $s_{i+1/2,\,j}$ we compute the average of s over the face $BCEF$, which we denote $s^L_{i+1/2,\,j}$. To obtain $s^L_{i+1/2,\,j}$, we integrate (12) over $ABCDEF$ and use the divergence theorem, taking advantage of the fact that the resulting integral over face $ACDF$ vanishes to obtain

$$u\, s^L_{i+1/2,\,j}\, \Delta t \Delta y = u \iint_{BCEF} s\, dy\, dt$$
$$= \iint_{ABDE} s\, dx\, dy + v \iint_{ABC} s\, dx\, dt - v \iint_{DEF} s\, dx\, dt. \quad (13)$$

By construction, s is piecewise bilinear and the edges of $ABDE$ are aligned with the coordinate axes. Therefore, one can use the midpoint formula to evaluate the integral over $ABDE$ exactly:

$$\iint_{ABDE} s\, dx\, dy = u\Delta t \Delta y\, s_{M,F}, \quad \text{where } s_{M,F} = \frac{\Delta x - u\Delta t}{2} s_{x,ij} + s_{ij}. \quad (14)$$

Consequently,

$$u\, s^{L}_{i+1/2,j} = \frac{u}{\Delta t \Delta y} \iint_{BCEF} s\, dy\, dt$$
$$= u\, s_{M,F} - \frac{v}{\Delta t \Delta y}\Big(\iint_{DEF} s\, dx\, dt - \iint_{ABC} s\, dx\, dt \Big). \qquad (15)$$

To evaluate the integrals over the surface triangles we use the same idea. Integrating $s_t + u s_x + v s_y = 0$ over the volume $DEFG$ shown in Figure 2, one can express the integral over DEF in terms of the integral over DEG, using the observation that contributions over the faces GEF and GFD vanish, to obtain:

$$v \iint_{DEF} s\, dx\, dt = \iint_{DEG} s\, dx\, dy. \qquad (16)$$

For the evaluation of the integral on the right side, the midpoint quadrature rule can be applied for the constant and linear parts of the bilinear polynomial on cell (i, j). For the bilinear term this rule is not exact. Therefore, the bilinear term is evaluated at the midpoints of the three edges and their sum is divided by three. The evaluation of the integral over the face ABC in (15) is analogous, noting that the characteristic domain of dependence extends into cell $(i, j-1)$, and therefore we evaluate the bilinear polynomial on cell $(i, j-1)$.

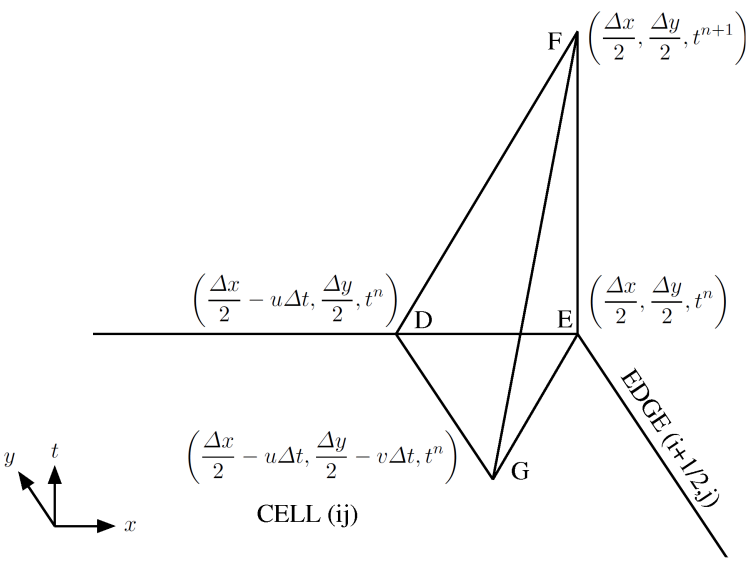


Figure 2. Characteristic domain of dependence of triangle DEF. The x- and y-coordinates are specified relative to block center. Figure taken from [3].

2.4.2. *Nonconstant velocity.* Here we generalize the construction above to the more general problem where (u, v) is spatially varying. At each edge we need to calculate the upwind edge state. If we assume $u_{i+1/2,j} > 0$, then we need to calculate $s^L_{i+1/2,j}$. For nonconstant velocity, we write the equation in the form

$$s_t + us_x + (vs)_y + su_x = 0,$$

and integrate over region $ABCDEF$ to obtain

$$\begin{aligned} &s^L_{i+1/2,j} \\ &\quad = s_{M,F} - \frac{\Delta t}{2}\frac{(u_{i+1/2,j} - u_{i-1/2,j})}{\Delta x} s_{M,F} - \frac{\Delta t}{2\Delta y}(v_{i,j+1/2}\Gamma^+ - v_{i,j-1/2}\Gamma^-), \end{aligned} \tag{17}$$

where Γ^+ and Γ^- represent the average values of the flux vs over the triangles DEF and ABC, respectively, and

$$s_{M,F} = \frac{\Delta x - u_{i+1/2,j}\Delta t}{2} s_{x,ij} + s_{ij}. \tag{18}$$

Here the term

$$-\frac{\Delta t}{2}\frac{(u_{i+1/2,j} - u_{i-1/2,j})}{\Delta x} s_{M,F}$$

approximates the volume integral of su_x over $ABCDEF$ using explicit Euler quadrature in time and treating u_x as a constant given by the difference of the edge velocities.

To compute the transverse correction term Γ^+, we write the equation in the form

$$s_t + us_x + vs_y + s(u_x + v_y) = 0.$$

If $v_{i,j+1/2} > 0$, we define

$$\begin{aligned} s^+_m &= \frac{1}{m(DEG)} \iint_{DEG} s\, dx\, dy \\ &= \frac{s_{xy,ij}}{12}\left[3\Delta x\Delta y - 4u_{i+1/2,j}\Delta t\Delta y - 2v_{i,j+1/2}\Delta x\Delta t + 3u_{i+1/2,j}v_{i,j+1/2}(\Delta t)^2\right] \\ &\qquad + \frac{s_{x,ij}}{6}(3\Delta x - 4\Delta t u_{i+1/2,j}) + \frac{s_{y,ij}}{6}(3\Delta y - 2\Delta t v_{i,j+1/2}) + s_{ij}. \end{aligned} \tag{19}$$

Then

$$\begin{aligned} \Gamma^+ &= \frac{1}{m(DEG)}\left(\iint_{DEG} s\, dx\, dy - \iiint_{DEFG} s(u_x + v_y)\, dx\, dy\, dt \right) \\ &= s^+_m - \frac{1}{m(DEG)} \iiint_{DEFG} s(u_x + v_y)\, dx\, dy\, dt \\ &= s^+_m \cdot \left[1 - \frac{\Delta t}{3}\left(\frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{\Delta y}\right)\right], \end{aligned} \tag{20}$$

where we have again used explicit Euler quadrature in time for the volume integral.

Otherwise if $v_{i,j+1/2} < 0$, we define

$$u_{\text{proj}} = \begin{cases} u_{i+1/2,j+1} & \text{if } u_{i+1/2,j} \cdot u_{i+1/2,j+1} > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

that is, in the second case we project corner G in Figure 2 from cell $(i+1,\ j+1)$ onto the edge $(i+1/2,\ j+1)$. This ensures that the characteristic domain of dependence is contained within one cell. Then s_m^+ is given by

$$\begin{aligned} s_m^+ = \frac{s_{xy,i,j+1}}{12}\big[&-3\Delta x\Delta y + 2(u_{i+1/2,j} + u_{\text{proj}})\Delta t\Delta y - 2v_{i,j+1/2}\Delta x\Delta t \\ &+ (u_{i+1/2,j} + 2u_{\text{proj}})v_{i,j+1/2}(\Delta t)^2\big] \\ &+ \frac{s_{x,i,j+1}}{6}\left[3\Delta x - 2\Delta t(u_{i+1/2,j} + u_{\text{proj}})\right] \\ &+ \frac{s_{y,i,j+1}}{6}\left(-3\Delta y - 2\Delta t v_{i,j+1/2}\right) + s_{i,j+1} \end{aligned} \tag{22}$$

and

$$\Gamma^+ = s_m^+ \cdot \left[1 - \frac{\Delta t}{3}\left(\frac{u_{i+1/2,j+1} - u_{i-1/2,j+1}}{\Delta x} + \frac{v_{i,j+3/2} - v_{i,j+1/2}}{\Delta y}\right)\right]. \tag{23}$$

The formula for Γ^- is analogous. We note that Γ^+ computed for edge $(i+1/2,\ j)$ is, in general, not the same as Γ^- computed for edge $(i+1/2,\ j+1)$.

If $u_{i,j+1/2} < 0$, we need to calculate the edge value from cell $(i+1,\ j)$, which we denote by $s^R_{i+1/2,j}$, using formulae analogous to the above. The calculation of $s_{i,j+1/2}$ is done similarly.

3. New quadratic BDS method

3.1. ***Overview.*** The main drawback of the bilinear BDS method compared to PPM-style methods is that it uses only a bilinear polynomial for the profile reconstruction. As a result, the method is only second-order accurate. To address that limitation we now include quadratic terms in the polynomial reconstruction. Each step of the quadratic BDS method is similar to that of the bilinear method; Step III is unchanged, while Step I and Step II now differ because we work with a quadratic rather than bilinear polynomial.

3.2. ***Construction of the quadratic polynomial.*** Here, we describe an algorithm for the construction of a quadratic polynomial representation of s at time t^n, written in the form

$$\begin{aligned} p^{\text{q}}_{ij}(x,\ y) = s_{xx,ij}(x - x_i)^2 + s_{yy,ij}(y - y_j)^2 + s_{xy,ij}(x - x_i)(y - y_j) \\ + s_{x,ij}(x - x_i) + s_{y,ij}(y - y_j) + \bar{s}. \end{aligned} \tag{24}$$

We denote the constant term by $\bar{s}$ instead of $\hat{s}$ (as we did for the bilinear polynomial p^{bl}) since the constant term will no longer be equal to s_{ij}. Note that this is not a full biquadratic polynomial; no mixed quadratic terms are included. The construction and limiting of a full biquadratic polynomial would be considerably more complicated and would not lead to a higher order of convergence.

We begin the construction of the quadratic polynomial as in the bilinear method, using (6) to define corner values for each cell. We will determine the quadratic terms independently.

To obtain an estimate for s_{xx} at the center of cell (i, j) we construct the quartic polynomial whose cell average matches the cell averages $s_{i-2,j}$, $s_{i-1,j}$, s_{ij}, $s_{i+1,j}$, and $s_{i+2,j}$. We then approximate the second derivative of the function s by the second derivative of the quartic polynomial at the center of cell (i, j). This leads to the following formulae:

second derivative at cell center of cell (i, j) in x-direction =

$$\frac{1}{8(\Delta x)^2}\left(-s_{i-2,j} + 12s_{i-1,j} - 22s_{ij} + 12s_{i+1,j} - s_{i+2,j}\right), \quad (25a)$$

second derivative at cell center of cell (i, j) in y-direction =

$$\frac{1}{8(\Delta y)^2}\left(-s_{i,j-2} + 12s_{i,j-1} - 22s_{ij} + 12s_{i,j+1} - s_{i,j+2}\right). \quad (25b)$$

By construction, these formulae are exact for one-dimensional polynomials up to order four. (In fact, they are even exact for a quintic polynomial due to their symmetry.) The coefficients for the quadratic terms $s_{xx,ij}$ and $s_{yy,ij}$ are then given by dividing the estimates for the second derivatives by two.

We then want to construct a polynomial out of the above information. We calculate $s_{xy,ij}$, $s_{x,ij}$, and $s_{y,ij}$ from the estimates for the corner values RH, RL, LH, and LL using equations (7a)–(7c), and $s_{xx,ij}$ and $s_{yy,ij}$ out of the estimates for the second derivatives:

$$s_{xx,ij} = \tfrac{1}{2}(\text{second derivative at cell center of cell } (i, j) \text{ in } x\text{-direction}), \quad (26a)$$

$$s_{yy,ij} = \tfrac{1}{2}(\text{second derivative at cell center of cell } (i, j) \text{ in } y\text{-direction}). \quad (26b)$$

To make sure that the average of the polynomial over cell (i, j) equals the cell average s_{ij}, we redefine the constant term

$$\bar{s} = s_{ij} - \frac{1}{\Delta x \Delta y}\iint_{\mathrm{cell}(i,j)} s_{xx,ij}(x - x_i)^2 + s_{yy,ij}(y - y_j)^2\, dx\, dy \quad (27a)$$

$$= s_{ij} - \tfrac{1}{12}\left[s_{xx,ij}(\Delta x)^2 + s_{yy,ij}(\Delta y)^2\right]. \quad (27b)$$

It is straightforward to show that this algorithm reconstructs a polynomial of the form (24) exactly.

3.3. ***Limiting the quadratic profile.*** The limiting of the quadratic polynomial (24) is split up into three parts:

Step 1. Test whether all estimated corner values are smaller (or larger) than the cell average s_{ij}. This can happen, for example, if the peak of a Gaussian lies in the middle of a cell. If that is the case, we set the polynomial to be constant on that cell with the value s_{ij} and the limiting process is complete.

Step 2. Otherwise, we attempt to accept the polynomial with unlimited slopes $s_{xy,ij}$, $s_{x,ij}$, and $s_{y,ij}$ and only limit the quadratic coefficients $s_{xx,ij}$ and $s_{yy,ij}$ appropriately. The resulting polynomial is tested to see if it is suitable. If so, the limiting is complete.

Step 3. In the third step, we first construct the *limited* bilinear profile using the original BDS approach. We then adjust $s_{xx,ij}$ and $s_{yy,ij}$ to construct a suitable quadratic polynomial.

The tests in Step 2 usually fail close to a discontinuity, that is, we need to use the more restrictive limiting in Step 3 for that case. Furthermore, these tests do ensure that the minimum and maximum values of the polynomial (if accepted in Step 2) are bounded by the cell values of neighboring cells (in order to satisfy a maximum principle for constant coefficient linear advection). We first discuss our general strategy for limiting the quadratic terms before giving detailed algorithms.

For the limiting of $s_{xx,ij}$ and $s_{yy,ij}$ we mainly consider the partial derivatives of the quadratic polynomial p^{q} in the x- and in the y-direction, or to be more precise: we determine whether p^{q}_x and/or p^{q}_y vanish in the interior of the cell. In Step 2 of our limiting procedure we limit the quadratic coefficients if both partial derivatives happen to vanish in the same cell, that is, if we happen to have an interior extremum or saddle point. In the more restrictive Step 3 of our limiting we limit $s_{xx,ij}$ if p^{q}_x vanishes in the interior of the cell independently of p^{q}_y. The limiting is set up in such a way that the position of the root of p^{q}_x is projected onto the edge closer to that position (w.r.t. the x-coordinate). The same holds true for $s_{yy,ij}$.

In describing the limiting procedure for $s_{xx,ij}$ in more detail, we will suppress the ij index in (24). The partial derivative of $p^{\mathrm{q}}(x, y)$ with respect to x is given by

$$p^{\mathrm{q}}_x(x, y) = 2s_{xx}(x - x_i) + s_{xy}(y - y_j) + s_x. \tag{28}$$

Setting $p^{\mathrm{q}}_x(x, y) = 0$ then corresponds to

$$-s_x - s_{xy}(y - y_j) = 2s_{xx}(x - x_i). \tag{29}$$

The right side of (29) varies within $\left[-2|s_{xx}|\frac{\Delta x}{2}, 2|s_{xx}|\frac{\Delta x}{2}\right]$ in cell (i, j). Therefore, in order for (29) to never hold within cell (i, j) (which is equivalent to saying that

$p_x^{\mathrm{q}}(x, y) \neq 0$ within cell (i, j)), we need

$$|s_x + s_{xy}(y - y_j)| \geq 2|s_{xx}|\frac{\Delta x}{2} \qquad \text{for all } y \in \left[y_j - \frac{\Delta y}{2}, y_j + \frac{\Delta y}{2}\right]. \tag{30}$$

Since $s_x + s_{xy}(y - y_j)$ is a linear function of y, this can't be true if the function values for $y_j - \Delta y/2$ and $y_j + \Delta y/2$ have opposite signs. So if

$$\operatorname{sign}\left(s_x + s_{xy}\frac{\Delta y}{2}\right) \cdot \operatorname{sign}\left(s_x - s_{xy}\frac{\Delta y}{2}\right) < 0, \tag{31}$$

then there exists a $\hat{y} \in [y_j - \Delta y/2, y_j + \Delta y/2]$ such that $p_x^{\mathrm{q}}(x_i, \hat{y}) = 0$. Otherwise the two terms have the same sign and we can take the one with the smaller absolute value as a limiting value. We define

$$\mathrm{cmp} = \min\left(\left|s_x + s_{xy}\frac{\Delta y}{2}\right|, \left|s_x - s_{xy}\frac{\Delta y}{2}\right|\right) \tag{32}$$

and ask for $\mathrm{cmp} \geq \Delta x|s_{xx}|$ to be true. In Step 2, if (31) is satisfied *and* if additionally one of the two conditions analogous to (31) and (32) for s_{yy} is true then we set $s_{xx} = 0$. Otherwise (i.e., (31) not true) if $\mathrm{cmp} < \Delta x|s_{xx}|$ *and* if additionally one of the two analogous conditions for s_{yy} is true we redefine s_{xx} as

$$s_{xx} = \operatorname{sign}(s_{xx})\frac{\mathrm{cmp}}{\Delta x}, \tag{33}$$

that is, we project the position of the root of p_x^{q} from the interior of the cell onto the edge. The analogous limiting applies for s_{yy}. In our second, more restrictive algorithm used in Step 3 we limit s_{xx} and s_{yy} independently of each other. That means if (31) is true, we set $s_{xx} = 0$. Otherwise if $\mathrm{cmp} < \Delta x|s_{xx}|$, we define

$$s_{xx} = \operatorname{sign}(s_{xx})\frac{\mathrm{cmp}}{\Delta x}. \tag{34}$$

The limiting for s_{yy} is analogous.

With this basic approach to limiting the quadratic term, we now provide the details of Step 2 and Step 3 of the limiting procedure. We start with the algorithm used in Step 2 which is designed for smooth areas of the solution.

3.3.1. *Limiting in smooth parts of the solution.* The idea for this algorithm is to try to keep the unlimited polynomial if possible, but still satisfy a maximum principle. To achieve that goal we take the unlimited slopes $s_{x,ij}$, $s_{y,ij}$, and $s_{xy,ij}$ as given by (7a)–(7c). The estimates for the quadratic terms $s_{xx,ij}$ and $s_{yy,ij}$ are only limited if both p_x^{q} and p_y^{q} vanish in the same cell as described above. Since the overall algorithm is designed to satisfy a maximum principle for constant coefficient advection, we need to check whether the minimum and maximum values of the polynomial over the entire cell lie inside the corresponding bounds. The limiting of $s_{xx,ij}$ and $s_{yy,ij}$ ensures that the minimum and maximum lie on the boundary of the cell. Hence, we check whether all corner values and the minimum/maximum

on all four edges lie between the cell values of neighboring cells. Since the edges are aligned with the coordinate axes, the two-dimensional biquadratic polynomial simplifies to a one-dimensional quadratic polynomial there. Consider the upper y-edge, that is, fix $y = \Delta y/2$. If there is an interior extremum at all, that is, if

$$|s_{x,ij} + s_{xy,ij}\Delta y/2| < |s_{xx,ij}\Delta x|, \tag{35}$$

then the extremum has to have the x-coordinate

$$x_{extr,+} = -\frac{s_{x,ij} + s_{xy,ij}\Delta y/2}{2s_{xx,ij}} \tag{36}$$

relative to x_i. The formulae for the other three edges are deduced analogously. This leads to the following algorithm:

Step 2: Limiting of quadratic profile in smooth parts of solution.

Assume given the estimated corner values $\mathrm{LL}_{ij}, \ldots, \mathrm{RH}_{ij}$ and estimated coefficients $s_{xx,ij}$ and $s_{yy,ij}$ using equations (6) as well as (25a), (26a) and (25b), (26b).

(1) Calculate the unlimited slopes $s_{x,ij}$, $s_{y,ij}$, $s_{xy,ij}$ out of $\mathrm{LL}_{ij}, \ldots, \mathrm{RH}_{ij}$ using (7a)–(7c).

(2) Check whether both p_x^{q} and p_y^{q} vanish in the interior of the same cell:
set $\mathrm{test1}_{xx} \leftarrow$ false, $\mathrm{test2}_{xx} \leftarrow$ false, $\mathrm{test1}_{yy} \leftarrow$ false, $\mathrm{test2}_{yy} \leftarrow$ false.

- *if* $\mathrm{sign}(s_{x,ij} + s_{xy,ij}\Delta y/2) \cdot \mathrm{sign}(s_{x,ij} - s_{xy,ij}\Delta y/2) < 0$, set $\mathrm{test1}_{xx} \leftarrow$ true.
- *else if* $\mathrm{cmp} < \Delta x|s_{xx,ij}|$, where cmp is defined by (32), set $\mathrm{test2}_{xx} \leftarrow$ true.

Proceed analogously with $\mathrm{test1}_{yy}$ and $\mathrm{test2}_{yy}$. If $\mathrm{test1}_{xx}$ evaluates to true *and* either one of the tests for yy is true, set $s_{xx,ij} \leftarrow 0$. If $\mathrm{test2}_{xx}$ evaluates to true *and* either one of the tests for yy is true, set $s_{xx,ij} \leftarrow \mathrm{sign}(s_{xx,ij})\,\mathrm{cmp}/\Delta x$. Proceed analogous for $s_{yy,ij}$.

(3) Adjust the constant term $\bar{s}$ using (27b).

(4) Check whether the reconstructed polynomial lies in bounds:

- Evaluate the quadratic polynomial p^{q} given in (24) at the four corners. For each corner, check whether the value of the polynomial lies between the cell averages of the four cells surrounding that corner.
- Calculate the extremal position on each of the four edges (if one exists) using equations (35) and (36) and evaluate p^{q} there. Check whether the value of that point lies between the cell averages of the four cells closest to the position of the extremum.

If all tests are satisfied, keep that polynomial and skip (5) immediately below.

(5) Limit $s_{xx,ij}$ independently of $s_{yy,ij}$ if one of the following conditions holds true:

- *if* $\operatorname{sign}(s_{x,ij} + s_{xy,ij}\Delta y/2) \cdot \operatorname{sign}(s_{x,ij} - s_{xy,ij}\Delta y/2) < 0$, set $s_{xx,ij} \leftarrow 0$.
- *else if* $\text{cmp} < \Delta x|s_{xx,ij}|$ (see (32)), set $s_{xx,ij} \leftarrow \operatorname{sign}(s_{xx,ij})\,\text{cmp}/\Delta x$.

Calculate $s_{yy,ij}$ following the same rules. Check again whether the corners are in bounds. If yes, keep that polynomial. If not, this algorithm was *not* successful (and we'll continue with *Step 3* described below).

Remark. In (5), we only need to check the corners, because by limiting $s_{xx,ij}$ and $s_{yy,ij}$ the way we do there, the minimum and maximum values of p^q now occur at the corners.

Numerical tests suggest that this limiting leads to very good performance in the sense of the overall L^1 error for smooth initial data.

3.3.2. *Limiting ensuring monotonicity.* The algorithm in this subsection is designed for discontinuities. Whereas we took the unlimited slopes $s_{x,ij}$, $s_{y,ij}$, and $s_{xy,ij}$ in the algorithm above, we now apply the limiting procedure from the original bilinear BDS method to the slopes $s_{x,ij}$, $s_{y,ij}$, and $s_{xy,ij}$. In this way we make sure that we preserve the behavior of the bilinear BDS method close to discontinuities. Additionally, we limit the coefficients $s_{xx,ij}$ and $s_{yy,ij}$ using the more restrictive way described above such that the polynomial on cell (i, j) is monotone in x- and y-direction over the whole cell. This leads to the following algorithm:

Step 3: Limiting of quadratic profile close to discontinuities.

Assume the estimated corner values $\text{LL}_{ij}, \ldots, \text{RH}_{ij}$ and estimated coefficients $s_{xx,ij}$ and $s_{yy,ij}$ using equations (6) as well as (25a), (26a) and (25b), (26b) are given.

(1) Use the limiting procedure in the original bilinear BDS to limit the (bi-)linear coefficients, i.e., limit $s_{x,ij}$, $s_{y,ij}$, and $s_{xy,ij}$ following the algorithm given in Section 2.3.

(2) Limit $s_{xx,ij}$ if one of the following conditions holds true:
- *if* $\operatorname{sign}(s_{x,ij} + s_{xy,ij}\Delta y/2) \cdot \operatorname{sign}(s_{x,ij} - s_{xy,ij}\Delta y/2) < 0$, set $s_{xx,ij} \leftarrow 0$.
- *else if* $\text{cmp} < \Delta x|s_{xx,ij}|$ (see (32)), set $s_{xx,ij} \leftarrow \operatorname{sign}(s_{xx,ij})\,\text{cmp}/\Delta x$.

Calculate $s_{yy,ij}$ analogously.

(3) Adjust the constant term $\bar{s}$ using (27b) .

(4) Test whether the corner values of the fully reconstructed quadratic polynomial exceed the cell averages of the neighboring cells. If this is the case, set $s_{xx,ij} \leftarrow 0$ and $s_{yy,ij} \leftarrow 0$, and set the constant term equal to s_{ij}.

The last step ensures that the corner values of the quadratic polynomial lie between the minimum/maximum values of the neighboring cells, that is, it ensures that the algorithm satisfies a maximum principle for constant coefficient linear

advection. Without the quadratic terms the four corners of the cell (i, j) have the values

$$\pm\frac{\Delta x\Delta y}{4}s_{xy,ij} \pm \frac{\Delta x}{2}s_{x,ij} \pm \frac{\Delta y}{2}s_{y,ij} + s_{ij}, \tag{37}$$

which are guaranteed to lie between the minimum/maximum bounds due to the bilinear limiting strategy. Then we add the quadratic terms and change the constant term from s_{ij} to $\bar{s} = s_{ij} - \frac{1}{12}s_{xx,ij}(\Delta x)^2 - \frac{1}{12}s_{yy,ij}(\Delta y)^2$. Consequently, the corners have the values

$$\begin{aligned} &s_{xx,ij}\frac{(\Delta x)^2}{4} + s_{yy,ij}\frac{(\Delta y)^2}{4} \pm \frac{\Delta x\Delta y}{4}s_{xy,ij} \pm \frac{\Delta x}{2}s_{x,ij} \pm \frac{\Delta y}{2}s_{y,ij} + \bar{s} \\ &\quad = s_{xx,ij}\frac{(\Delta x)^2}{6} + s_{yy,ij}\frac{(\Delta y)^2}{6} \pm \frac{\Delta x\Delta y}{4}s_{xy,ij} \pm \frac{\Delta x}{2}s_{x,ij} \pm \frac{\Delta y}{2}s_{y,ij} + s_{ij}. \end{aligned} \tag{38}$$

So compared to the bilinear polynomial p^{bl} the values of the quadratic polynomial p^{q} differ at every corner by $\frac{1}{6}s_{xx,ij}(\Delta x)^2 + \frac{1}{6}s_{yy,ij}(\Delta y)^2$ (which is a constant for every cell).

For smooth initial data, this constant often even helps to keep the corner values in bounds. In our numerical tests, violations were usually seen only for discontinuous initial data. Therefore, we chose the straightforward way to fix this problem just described: at the end of the limiting routine, we check whether the values of the quadratic polynomial at the corners lie inside bounds. If that's the case, we are done. Otherwise, we set $s_{xx,ij} = 0$ and $s_{yy,ij} = 0$, that is, we go back to the bilinear polynomial.

This leads to a method that obeys the maximum principle for constant coefficient linear advection (up to numerical roundoff error):

- The minimum and maximum value of the quadratic polynomial p^{q}_{ij} on cell (i, j) are limited by the values on the boundary in Step 2 of the limiting process and by the values at the corners in Step 3.
- The values on the boundary and the corner values of the quadratic polynomial don't exceed the minimum/maximum of the neighboring cell averages.

3.4. ***Construction of edge states.*** The formalism based on integrating over the characteristic domain of dependence remains the same as in the bilinear scheme. The only changes that we need to make are to substitute higher order quadrature formulae to evaluate the integrals over the quadratic terms exactly. Let us first consider the integral

$$\iint_{ABDE} s(x, y)\,dx\,dy, \tag{39}$$

for the case $u_{i+1/2,j} > 0$ appearing in the calculation of the flux $s^L_{i+1/2,j}$ in (13). For the bilinear, linear, and constant terms of the polynomial, we can use the midpoint

formula as before. The integrals over the quadratic terms can be calculated explicitly:

$$\begin{aligned}
&\iint_{ABDE} s_{xx,ij}(x-x_i)^2\,dx\,dy \\
&\quad = s_{xx,ij}\Delta y \int_{x_{i+1/2}-u_{i+1/2,j}\Delta t}^{x_{i+1/2}} (x-x_i)^2\,dx \\
&\quad = s_{xx,ij}\Delta y\left[\tfrac{1}{4}(\Delta x)^2 u_{i+1/2,j}\Delta t - \tfrac{1}{2}\Delta x(u_{i+1/2,j}\Delta t)^2 + \tfrac{1}{3}(u_{i+1/2,j}\Delta t)^3\right], \qquad (40)
\end{aligned}$$

and

$$\iint_{ABDE} s_{yy,ij}(y-y_j)^2\,dx\,dy = \tfrac{1}{12}s_{yy,ij}u_{i+1/2,j}\Delta t(\Delta y)^3. \qquad (41)$$

That means that $s_{M,F}$ given by (18) is replaced by the following $s^Q_{M,F}$ in formula (17):

$$\begin{aligned}
s^Q_{M,F} &= \bar{s} + \frac{\Delta x - u_{i+1/2,j}\Delta t}{2} s_{x,ij} \\
&\quad + s_{xx,ij}\left[\tfrac{1}{4}(\Delta x)^2 - \tfrac{1}{2}\Delta x u_{i+1/2,j}\Delta t + \tfrac{1}{3}(u_{i+1/2,j}\Delta t)^2\right] + \tfrac{1}{12}s_{yy,ij}(\Delta y)^2. \qquad (42)
\end{aligned}$$

Additionally, we need to adjust the calculations of Γ^+ and Γ^- appropriately. This corresponds to changing the evaluations of the contributions coming from the triangles DEF and ABC. We evaluate the linear part of the polynomial with the midpoint rule. For the quadratic and bilinear terms we use the same rule as used for the bilinear term in the original BDS: we evaluate the terms at the midpoints of all three edges and divide the corresponding sum by three. In this way, we are evaluating all two-dimensional integrals exactly. The same changes hold true for all the other cases considered in Section 2.4.2 (i.e., the calculation of $s^R_{i+1/2,j}$, $s^L_{i,j+1/2}$, and $s^R_{i,j+1/2}$). We note that for constant coefficient advection, analogous to the original BDS algorithm, the quadratic BDS algorithm is equivalent to fitting a limited quadratic profile to the solution at time t^n, advecting that profile exactly and averaging it back onto the grid, which guarantees that the solution satisfies a maximum principle.

4. Numerical results

In this section we present a series of numerical tests using our new quadratic method (BDS_Q). In Section 4.1, we advect smooth and discontinuous initial data using a constant velocity field. We explore the effects of angle dependence of the velocity field on the accuracy and overshoot of our method. In Section 4.2, we explore the effects of advecting smooth and discontinuous data in a velocity field that varies in space. In Section 4.3, we integrate the algorithm into a variable density projection method [2], and thus the velocity varies in both space and time. In this example, we examine the behavior of the density in addition to a passively advected scalar.

In each of these sections, we compare our results to the original bilinear method (BDS_BL, [3]) and two PPM methods. The first, which we call PPM1, is the PPM algorithm [9] that has been in use for over 25 years. The second, which we call PPM2, is based on a recent effort [8] to preserve accuracy at smooth extrema. The algorithm as described in [8] suffers from sensitivity to roundoff error; we incorporate the correction to that as described in [16]. Finally, in Section 4.4 we discuss the performance of BDS_Q on some additional test problems discussed in the literature and compare results to other schemes.

In Sections 4.1 and 4.2 we solve the equation

$$s_t + (us)_x + (vs)_y = 0 \tag{43}$$

for both smooth and discontinuous initial conditions, where (u, v) is a specified velocity field. For the smooth case we define

$$s(x, y, t = 0) = e^{-60r^2}, \tag{44}$$

where $r^2 = (x-1)^2 + (y-1)^2$ on the domain $(0, 2)^2$. The discontinuous initial data is given by a round tophat of the form

$$s(x, y, t = 0) = \begin{cases} 1 & \text{if } r < 0.2, \\ 0 & \text{otherwise,} \end{cases} \tag{45}$$

where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$ on the domain $(0, 1)^2$. Given the analytic profile of s at $t = 0$, we discretize the smooth initial data using Gaussian quadrature rules, which give us a fourth-order estimate of the cell average. As a result, while the maximum of the analytical function in (44) is 1, the numerical maximum will be slightly lower for each resolution. To discretize the discontinuous round tophat we approximate the integral of the function over the cell by dividing the cell into 16 subcells, evaluating the analytic function at the center of each subcell, then averaging the 16 values to define the average over the original cell.

In Section 4.3, we integrate the algorithm into a variable density projection method [2]. First we consider the advection of a passive tracer in a constant density incompressible flow. Here although the velocity varies in space and time the tracer does not couple back to the fluid. In the second example we consider the advection of density in a variable density flow. For this case, the density couples back into the evolution of the velocity field.

For all of the tests we set the time step based on the CFL condition

$$\Delta t = \sigma_{\mathrm{CFL}} \min_{ij} \left(\frac{\Delta x}{|u_{ij}|}, \frac{\Delta y}{|v_{ij}|} \right),$$

with a CFL number, $\sigma_{\mathrm{CFL}} = 0.9$, and impose periodic boundary conditions on all faces.

Before presenting our results we would like to comment on the computational efficiency of the BDS_Q algorithm as compared to BDS_BL and PPM2. In our testing, BDS_BL is a factor of 1.79 times more expensive than PPM2, and BDS_Q is a factor of 2.47 times more expensive than PPM2. However, in our intended applications the computational cost is dominated by elliptic solvers, reaction networks, and/or calls to the equation of state, so the overall cost of advection is minor, even with the more expensive BDS_Q algorithm.

4.1. *Constant velocity advection.*

4.1.1. *Smooth initial data.* In this section we consider the evolution of s with initial data given by (44). The motivation for extending BDS_BL to BDS_Q by adding the quadratic terms was to increase the accuracy for problems with smooth initial data, while maintaining the lack of under- and overshoot that we see with BDS_BL for discontinuous problems. We report the L^1 norm of error relative to the exact solution for each method at three different resolutions for both the limited and unlimited forms of each algorithm in Table 1, where $(u, v) = (1, 0)$, and Table 2, where $(u, v) = (1, 0.2)$. As expected we see third-order convergence for unlimited BDS_Q as opposed to second-order convergence for BDS_BL, i.e., the error decreases by a factor of 8 rather than 4 for each factor 2 decrease in mesh spacing. With limiting, the ratio of errors with BDS_Q decreases slightly in the off-axis test, but we observe that BDS_Q is the only method to maintain such high convergence rates in this test. All other methods demonstrate second-order convergence for the off-axis test, even though PPM2 shows a high rate of convergence for the axis-aligned case.

Looking at the magnitudes of errors as opposed to the ratios, we see that for flow aligned with the x-axis, the errors are lowest using PPM2, which for this particular problem is equivalent to PPM without limiters. However, this relative advantage disappears when the flow does not align with a coordinate axis. At the highest

Method	100^2 Error	Ratio	200^2 Error	Ratio	400^2 Error
BDS_Q	1.89e-04	8.0	2.36e-05	8.3	2.83e-06
BDS_BL	6.18e-04	4.1	1.49e-04	4.1	3.62e-05
PPM1	5.86e-04	5.0	1.18e-04	5.1	2.30e-05
PPM2	4.48e-05	14.	3.16e-06	12.	2.62e-07
BDS_Q, no limiting	5.80e-05	8.7	6.69e-06	8.2	8.18e-07
BDS_BL, no limiting	5.45e-04	4.0	1.37e-04	4.0	3.45e-05
PPM, no limiting	4.48e-05	14.	3.16e-06	12.	2.62e-07

Table 1. Error in the L^1 norm at $t = 2$ for smooth initial data with $(u, v) = (1, 0)$.

Method	100^2 Error	Ratio	200^2 Error	Ratio	400^2 Error
BDS_Q	1.33e-03	7.2	1.85e-04	7.4	2.51e-05
BDS_BL	4.71e-03	4.1	1.15e-03	4.0	2.89e-04
PPM1	7.88e-03	3.8	2.07e-03	3.9	5.29e-04
PPM2	7.86e-03	4.0	1.98e-03	4.0	4.97e-04
BDS_Q, no limiting	7.49e-04	8.4	8.95e-05	8.1	1.10e-05
BDS_BL, no limiting	4.53e-03	4.0	1.13e-03	4.0	2.82e-04
PPM, no limiting	7.88e-03	4.0	1.98e-03	4.0	4.97e-04

Table 2. Error in the L^1 norm at $t = 10$ for smooth initial data with $(u, v) = (1, 0.2)$.

resolution of the off-axis test, the error in the solution is more than an order of magnitude smaller with BDS_Q than with any of the other methods.

Another metric for the performance of a method for scalar advection is the degree to which it preserves the maximum of a smooth peak, and the degree to which the final solution under- or overshoots the minimum and maximum, respectively, of the original solution. Analytically, scalar advection with a divergence-free velocity field should preserve the maximum and minimum of the original solution.

In Table 3 we show the peak value of the solution for the axis-aligned flow at final time; in this case, none of the methods exhibit undershoot. In Table 4 we show the peak value at the final times and the largest value of undershoot for the off-axis case. Disappointingly, the peak for BDS_Q is slightly lower than the peaks for BDS_BL and for eitherPPM method; this is an issue we hope to address in future work. It is clear that the reduction results from the limiting in BDS_Q; without limiting the maximum for BDS_Q is comparable to that of unlimited PPM. We also verify

Method	100^2 max	200^2 max	400^2 max
Analytic	0.98417	0.99601	0.99900
BDS_Q	0.95344	0.98454	0.99493
BDS_BL	0.95432	0.98493	0.99506
PPM1	0.96476	0.98873	0.99642
PPM2	0.98405	0.99598	0.99900
BDS_Q, no limiting	0.98281	0.99585	0.99897
BDS_BL, no limiting	0.98279	0.99588	0.99899
PPM, no limiting	0.98405	0.99598	0.99900

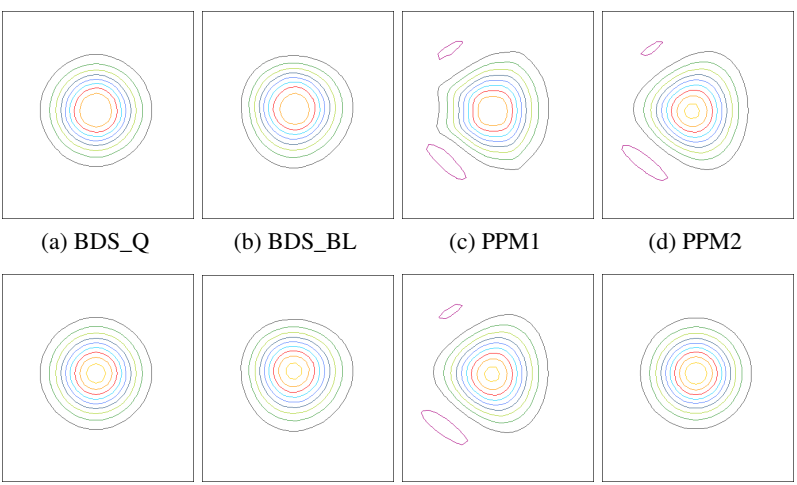
Table 3. Peak at $t = 2$ for smooth initial data with $(u, v) = (1, 0)$. None of the methods exhibit undershoot for this problem.

Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	0.98417	0.00000	0.99601	0.00000	0.99900	0.00000
BDS_Q	0.87065	0.00000	0.95442	0.00000	0.98383	0.00000
BDS_BL	0.86967	0.00000	0.95790	0.00000	0.98598	0.00000
PPM1	0.88597	–0.02133	0.96678	–0.00003	0.98907	0.00000
PPM2	0.94632	–0.01701	0.99261	0.00000	0.99877	0.00000
BDS_Q, no limiting	0.96600	0.00000	0.99377	0.00000	0.99873	0.00000
BDS_BL, no limiting	0.99512	–0.00316	0.99321	0.00000	0.99875	0.00000
PPM, no limiting	0.94632	–0.01783	0.99261	0.00000	0.99877	0.00000

Table 4. Maxima and minima at $t = 10$ for smooth initial data with $(u, v) = (1, 0.2)$.

in this test that both BDS_BL and BDS_Q show no undershoot for the off-axis problem, unlike both PPM approaches, which undershoot on the coarser grids.

A final metric we consider is the distortion of the original shape at the final time. Analytically, the initial data should stay undistorted throughout the entire evolution. Figure 3 shows contours of the solution at $t = 10$ for evolution with $(u, v) = (1, 0.2)$. BDS_Q clearly preserves the round shape most accurately, with some distortion evident for BDS_BL. The solutions for PPM1 and PPM2 show



(a) BDS_Q (b) BDS_BL (c) PPM1 (d) PPM2

(e) unlim. BDS_Q (f) unlim. BDS_BL (g) unlim. PPM (h) analytic

Figure 3. Final solution for smooth initial data, $(u, v) = (1, 0.2)$, and 100^2 resolution. There are nine contour lines evenly spaced from 0.1 to 0.9; in addition, the violet contour encloses the region where $s < -0.01$.

significant distortion. We can also see in this figure the flattening of the peak for the limited versions of BDS_BL and BDS_Q, as well as the undershoot for the PPM methods.

4.1.2. *Discontinuous initial data.* In this section we consider the evolution of discontinuous initial data given by (45), again considering two different velocity fields, $(u, v) = (1, 0)$ and $(u, v) = (1, 0.2)$. In Tables 5 and 6 we show the L^1 norm of error for each method at three different resolutions for both the limited and unlimited forms of each algorithm. In this table we no longer show the ratios of error because we do not expect to approach the asymptotic convergence rates with discontinuous initial data. We make three observations from these tables: first, the errors for the different methods are much closer to each other for discontinuous initial data than for smooth initial data; second, unlike for smooth initial data, for both velocity fields and all the methods, limiting reduces the L^1 error of the solution; third, for the axis-aligned flow field the errors for the PPM methods are slightly

Method	100^2 Error	200^2 Error	400^2 Error
BDS_Q	5.40e-03	3.30e-03	1.99e-03
BDS_BL	5.69e-03	3.56e-03	2.23e-03
PPM1	3.67e-03	2.18e-03	1.29e-03
PPM2	3.83e-03	2.32e-03	1.40e-03
BDS_Q, no limiting	6.97e-03	4.22e-03	2.52e-03
BDS_BL, no limiting	7.65e-03	5.01e-03	3.33e-03
PPM, no limiting	7.71e-03	4.68e-03	2.85e-03

Table 5. Error in the L^1 norm at $t = 1$ for discontinuous initial data with $(u, v) = (1, 0)$.

Method	100^2 Error	200^2 Error	400^2 Error
BDS_Q	1.23e-02	7.30e-03	4.34e-03
BDS_BL	1.45e-02	9.13e-03	5.82e-03
PPM1	2.31e-02	1.53e-02	1.02e-02
PPM2	2.33e-02	1.57e-02	1.06e-02
BDS_Q, no limiting	1.49e-02	8.79e-03	5.17e-03
BDS_BL, no limiting	2.22e-02	1.49e-02	9.96e-03
PPM, no limiting	2.86e-02	1.91e-02	1.27e-02

Table 6. Error in the L^1 norm at $t = 5$ for discontinuous initial data with $(u, v) = (1, 0.2)$.

Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_BL	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM1	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM2	1.00000	0.00000	1.00443	0.00000	1.00080	0.00000
BDS_Q, no limiting	1.10593	–0.10032	1.09505	–0.09264	1.09159	–0.08632
BDS_BL, no limiting	1.10527	–0.10060	1.13222	–0.13264	1.15713	–0.15713
PPM, no limiting	1.17344	–0.17344	1.14547	–0.14547	1.16618	–0.16618

Table 7. Maxima and minima at $t = 1$ for discontinuous initial data with $(u, v) = (1, 0)$.

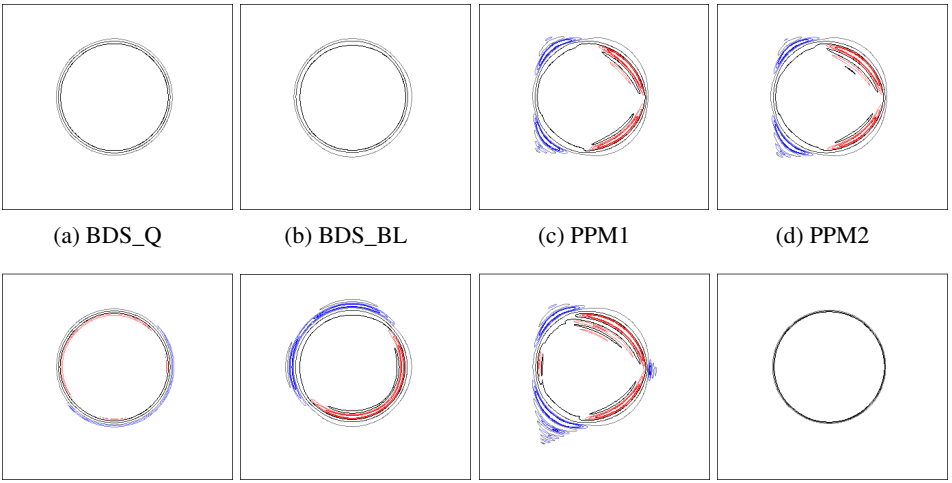
Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_BL	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM1	1.18611	–0.28803	1.21003	–0.32074	1.22745	–0.30010
PPM2	1.19335	–0.29089	1.21697	–0.29623	1.23178	–0.28074
BDS_Q, no limiting	1.10668	-0.06558	1.09510	-0.07061	1.09311	-0.07344
BDS_BL, no limiting	1.17508	-0.16271	1.19262	-0.18948	1.21179	-0.20986
PPM, no limiting	1.21288	-0.33820	1.23017	-0.32474	1.24244	-0.32613

Table 8. Maxima and minima at $t = 5$ for discontinuous initial data with $(u, v) = (1, 0.2)$.

lower; for the diagonal flow the BDS methods have slightly lower error.

In Tables 7 and 8 we show the maxima and minima of the solution at the final time. For the axis-aligned flow we see that, of the methods with limiters, only PPM2 introduces new maxima to the solution. However, for the off-axis flow the solutions for both PPM1 and PPM2 overshoot by more than 20% at the two higher resolutions, while BDS_BL and BDS_Q retain the initial maximum value of 1. We see similar results with the minima. We also note that while the L^1 error decreases with mesh spacing, the magnitude of the over- and undershoot does not.

Again we look at the distortion of the final solution for the case with the off-axis velocity field. In Figure 4 we see contours of the solution with contours in the regions of undershoot and overshoot given in color. We notice a slight spreading



(a) BDS_Q (b) BDS_BL (c) PPM1 (d) PPM2

(e) unlim. BDS_Q (f) unlim. BDS_BL (g) unlim. PPM (h) analytic

Figure 4. Final solution for discontinuous initial data, $(u, v) = (1, 0.2)$, and 400^2 resolution. There are three black contours from 0.05 to 0.95, three red contours from 1.05 to 1.15 marking the overshoot and three blue contours going from -0.15 to -0.05 marking the undershoot.

of the contours for all of the methods but no over- or undershoot for the limited BDS_Q and BDS_BL methods. All the PPM solutions show severe distortion from the over- and undershoot.

Finally, we examine the question of how the over- and undershoot vary with the angle of the velocity relative to the x-axis. Table 9 shows the maximum and minimum of each solution after 500 time steps for each of the limited methods using the discontinuous initial data at 100^2 resolution. We see that the $(u, v) = (1, 0.2)$ case is representative of off-axis velocities, and in fact the over- and undershoot seem to peak for both PPM1 and PPM2 when the velocity field is approximately 30° off the x-axis.

4.2. ***Variable velocity advection.*** Here we consider the velocity field given by $(u, v) = [1.0, \sin(\pi x)]$ in the domain $(0, 2)^2$. We advect s until $t = 10$ for both smooth and discontinuous initial data, with the discontinuous data now centered at $(1, 1)$ instead of $(0.5, 0.5)$ as was done earlier. The goal of this test is to examine if the conclusions of the previous section hold when the velocity field is no longer spatially constant.

4.2.1. *Smooth initial data.* Results for smooth initial data are shown in Tables 10 and 11. In Table 10 we report the L^1 error at $t = 10$, in Table 11 we show the

Method	(u, v) $(1,0)$	$(1,0.2)$	$(1,0.4)$	$(1,0.5)$	$(1,0.6)$	$(1,0.8)$	$(1,1)$
				Maximum			
Analytic	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
BDS_Q	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
BDS_BL	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
PPM1	1.00000	1.18762	1.19635	1.20484	1.20307	1.17445	1.16009
PPM2	1.00324	1.18856	1.21372	1.22929	1.22770	1.17534	1.15975
				Minimum			
Analytic	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
BDS_Q	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
BDS_BL	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
PPM1	0.00000	–0.28937	–0.32127	–0.32719	–0.32405	–0.32244	–0.28363
PPM2	0.00000	–0.27999	–0.32064	–0.32297	–0.32934	–0.31379	–0.26969

Table 9. Maxima and minima after 500 time steps for each limited method using discontinuous initial data at 100^2 resolution.

minimum and maximum value at $t = 10$ for all methods considered. Analogously to the conclusions in Section 4.1.1 for off-axis flow we observe, for smooth initial data:

- third-order accuracy for BDS_Q and second-order accuracy for other methods,
- flattening of the peak for BDS_Q and BDS_BL relative to PPM,
- BDS_Q outperforms BDS_BL by every metric except for a slightly lower peak at 400^2,
- no under- or overshoot for the BDS algorithms, some undershoot for the PPM algorithms.

Method	100^2 Error	Ratio	200^2 Error	Ratio	400^2 Error
BDS_Q	2.61e-03	8.0	3.25e-04	7.3	4.45e-05
BDS_BL	4.96e-03	4.4	1.13e-03	4.2	2.70e-04
PPM1	5.29e-03	4.1	1.28e-03	4.2	3.07e-04
PPM2	4.46e-03	4.0	1.11e-03	4.0	2.79e-04
BDS_Q, no limiting	1.88e-03	8.1	2.32e-04	8.2	2.84e-05
BDS_BL, no limiting	4.68e-03	4.4	1.07e-03	4.1	2.58e-04
PPM, no limiting	4.43e-03	4.0	1.12e-03	4.0	2.79e-04

Table 10. Error in the L^1 norm at $t = 10$ for smooth initial data with $(u, v) = [1.0, \sin(\pi x)]$.

Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	0.98417	0.00000	0.99601	0.00000	0.99900	0.00000
BDS_Q	0.83400	0.00000	0.94420	0.00000	0.98117	0.00000
BDS_BL	0.82682	0.00000	0.94381	0.00000	0.98189	0.00000
PPM1	0.88298	–0.01126	0.97013	–0.00014	0.99172	0.00000
PPM2	0.93607	–0.01069	0.98961	–0.00013	0.99824	0.00000
BDS_Q, no limiting	0.94487	–0.00163	0.99076	0.00000	0.99835	0.00000
BDS_BL, no limiting	0.92712	–0.01581	0.98848	–0.00027	0.99810	0.00000
PPM, no limiting	0.93715	–0.01224	0.98961	-0.00012	0.99824	0.00000

Table 11. Maxima and minima at $t = 10$ for smooth initial data with $(u, v) = [1.0, \sin(\pi x)]$.

Note that both BDS implementations, with limiting, do not undershoot at any time with respect to the scale used in our tables. In fact, neither BDS_Q nor BDS_BL show any under/overshoot larger than 10^{-9} at any of our reported resolutions. (This error depends directly on the choice of ϵ in the limiting algorithm described in Section 2.3.) By contrast, the PPM1/PPM2 algorithms first exhibit undershoot with magnitude greater than 10^{-5} at $t = 0.882$ (PPM1, 100^2), $t = 0.918$ (PPM2, 100^2), $t = 5.193$ (PPM1, 200^2), and $t = 5.148$ (PPM2, 200^2). We observe no undershoot on our finest grid for the PPM methods with respect to the scale in Table 11.

4.2.2. *Discontinuous initial data.* Repeating the same tests with discontinuous initial data leads to the results shown in Tables 12, 13, and 14. In these tables, we report the L^1 error at $t = 10$ as well as the minimum and maximum value both after only one time step and at $t = 10$. Analogous to the conclusions in Section 4.1.2 for off-axis flow, we observe for discontinuous initial data:

Method	100^2 Error	200^2 Error	400^2 Error
BDS_Q	2.98e-02	1.77e-02	1.05e-02
BDS_BL	3.34e-02	2.04e-02	1.25e-02
PPM1	3.59e-02	2.30e-02	1.49e-02
PPM2	3.58e-02	2.27e-02	1.47e-02
BDS_Q, no limiting	3.54e-02	2.13e-02	1.26e-02
BDS_BL, no limiting	4.41e-02	2.87e-02	1.86e-02
PPM, no limiting	4.00e-02	2.50e-02	1.58e-02

Table 12. Error in the L^1 norm at $t = 10$ for discontinuous initial data with $(u, v) = [1.0, \sin(\pi x)]$.

Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_BL	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM1	1.04271	–0.04141	1.04393	–0.04301	1.04444	–0.04062
PPM2	1.03456	–0.03403	1.03931	–0.03666	1.04231	–0.04037
BDS_Q, no limiting	1.05006	–0.05824	1.05962	–0.05489	1.04745	–0.05812
BDS_BL, no limiting	1.04201	–0.04645	1.04771	–0.04659	1.04002	–0.04863
PPM, no limiting	1.05559	–0.05606	1.06203	–0.06010	1.05846	–0.06410

Table 13. Maxima and minima after one time step for discontinuous initial data with $(u, v) = [1.0, \sin(\pi x)]$.

Method	100^2 max	100^2 min	200^2 max	200^2 min	400^2 max	400^2 min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	0.99544	0.00000	0.99998	0.00000	1.00000	0.00000
BDS_BL	0.99665	0.00000	0.99999	0.00000	1.00000	0.00000
PPM1	1.10007	–0.09099	1.10253	–0.11853	1.12603	–0.14435
PPM2	1.09626	–0.09067	1.09608	–0.12249	1.12464	–0.15188
BDS_Q, no limiting	1.13956	–0.06600	1.12081	–0.06652	1.11089	–0.06679
BDS_BL, no limiting	1.22656	–0.11437	1.22889	–0.14710	1.22822	–0.17816
PPM, no limiting	1.15451	–0.12074	1.12697	–0.14695	1.13860	–0.17400

Table 14. Maxima and minima at $t = 10$ for discontinuous initial data with $(u, v) = [1.0, \sin(\pi x)]$.

- a lower error for BDS_Q than for any of the other algorithms,
- no undershoot or overshoot for the BDS algorithms at any time,
- an overshoot for the PPM algorithms of $> 3\%$ after one time step, and $> 9\%$ at $t = 10$.

We conclude that the behavior of the BDS_Q algorithm with a spatially varying velocity field is consistent with that observed with a constant velocity field.

4.3. ***Shear layer example.*** Here we consider a temporally and spatially evolving velocity field that is determined by solving the incompressible Euler equations using the approximate projection algorithm described in [2]. We note that although the algorithm uses an approximate projection to define the cell-centered velocity

at the end of each time step, the edge-based velocities that are used for advection are discretely divergence-free (to the tolerance of the iterative solver that is used to enforce the constraint). We initialize the problem with parallel shear layers perturbed slightly; the initial velocity field is given by

$$(u, v)|_{t=0} = \left(\tanh\left[60(0.07 - |y - 0.5|)\right], 0.5\sin(2\pi x)\right). \tag{46}$$

In the first test we initialize the density to be constant everywhere; because the flow is divergence-free the density will remain constant. We solve (43) for a passively advected tracer, s, that is initialized with discontinuous data described in (45). In Tables 15 and 16 we see the over- and undershoot associated with each method after one time step and at $t = 0.23$, respectively. Both PPM1 and PPM2 over/undershoot by 3–4% after one time step, and by 6–25% at the final time. The BDS methods do not over/undershoot at any time during the simulation. Figure 5 shows the solution for both BDS_Q and PPM2; the locations of the overshoot and undershoot are evident in the blue and red coloring of the figure.

In the second test we consider a variable density case in which the initial density, ρ, is given by

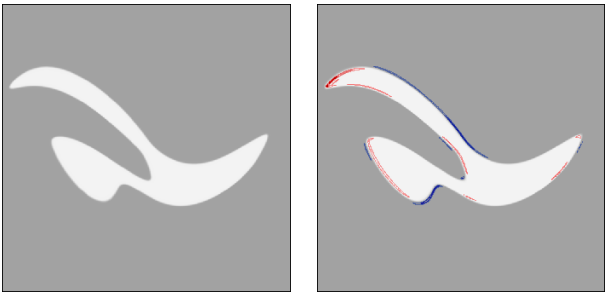
$$\rho(x, y, t = 0) = 1.5 + 0.5\tanh\left[600(0.02 - |y - 0.5|)\right] \tag{47}$$

	128^2		256^2		512^2	
Method	max	min	max	min	max	min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_BL	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM1	1.04177	–0.03766	1.04200	–0.03845	1.04166	–0.04293
PPM2	1.03531	–0.03570	1.03942	–0.03798	1.03976	–0.04026

Table 15. Maxima and minima of s after one time step for the constant density shear layer problem with discontinuous initial data.

	128^2		256^2		512^2	
Method	max	min	max	min	max	min
Analytic	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_Q	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
BDS_BL	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
PPM1	1.13413	–0.05983	1.19031	–0.07449	1.20885	–0.09356
PPM2	1.17185	–0.06706	1.23665	–0.08806	1.25714	–0.10572

Table 16. Maxima and minima of s at $t = 0.23$ for the constant density shear layer problem with discontinuous initial data.



(a) BDS_Q (b) PPM2

Figure 5. Solution at time $t = 0.23$ for the constant density shear layer problem with discontinuous initial data and resolution 512^2. Undershoot between -0.1 and -0.01 is marked in blue. Overshoot between 1.01 and 1.15 is marked in red. More intense colors are used for areas of higher over- and undershoot.

and initial velocity field given by

$$(u, v)|_{t=0} = \{\tanh\left[600(0.07 - |y - 0.5|)\right], 3.5\sin(2\pi x)\}. \tag{48}$$

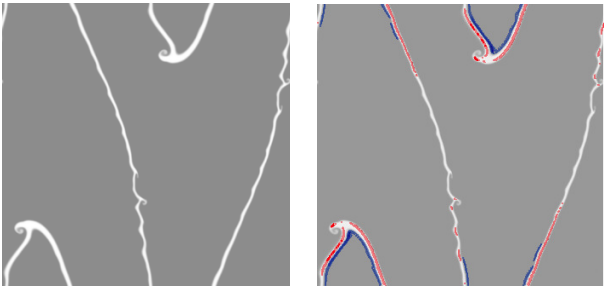
For this case, the mass conservation equation,

$$\rho_t + (u\rho)_x + (v\rho)_y = 0, \tag{49}$$

is part of the evolution. We note that this test differs from all the others in that the density couples back into the calculation of the velocities at each time step. In Table 17 we see the over- and undershoot associated with each method at $t = 0.19$. As in the previous tests both PPM1 and PPM2 create noticeable under- and overshoot while neither of the BDS methods do. In fact, the BDS methods do not over- or undershoot at any time during the simulation, whereas the PPM methods suffer at early times. For example, for the 512^2 simulations, we first observe an overshoot $> 10^{-5}$ at $t = 0.017$ (PPM1) and $t = 0.003$ (PPM2). Figure 6 shows the solution for

	128^2		256^2		512^2	
Method	max	min	max	min	max	min
Analytic	2.00000	1.00000	2.00000	1.00000	2.00000	1.00000
BDS_Q	1.92762	1.00000	1.99013	1.00000	1.99967	1.00000
BDS_BL	1.92277	1.00000	1.99376	1.00000	1.99981	1.00000
PPM1	2.02653	0.89027	2.10101	0.86719	2.13961	0.87522
PPM2	2.09583	0.80600	2.09924	0.87192	2.16143	0.84435

Table 17. Maxima and minima of ρ at $t = 0.19$ for the variable density shear layer problem.



(a) BDS_Q (b) PPM2

Figure 6. Solution at time $t = 0.23$ for the variable density shear layer problem at resolution 512^2. Undershoot between 0.85 and 0.99 is marked in blue. Overshoot between 2.01 and 2.15 is marked in red. More intense colors are used for areas of higher over- and undershoot.

both BDS_Q and PPM2; the locations of the overshoot and undershoot are again evident in the blue and red coloring of the figure.

4.4. ***Additional comparisons.*** In this section, we examine the results of BDS_Q on some test problems for linear advection that have been discussed in the literature. It is impossible to provide a comprehensive comparison of BDS to the full gamut of possible advection schemes; here we will consider the third-order central WENO scheme of Levy, Puppo, and Russo (LPR) [15] and the ADER schemes discussed by Toro and Titarov [24]. LPR is a multidimensional staggered grid scheme that uses a limited quadratic reconstruction algorithm. The ADER schemes use a similar reconstuction but use a Cauchy–Kowaleski procedure and Taylor series expansion to approximate the flux at Gaussian quadrature nodes on the space-time edges of the control volume.

First we consider the linear advection test problem, $s_t + s_x + s_y = 0$, examined in [15]. The initial conditions are given by

$$s(x, y, t = 0) = \sin^2(\pi x) \sin^2(\pi y)$$

on the domain $(0, 1)^2$ and errors for LPR are computed at $t = 1$ based on a CFL of 0.425. The BDS_Q scheme has a less restrictive stability limit so we use a CFL of 0.9, which represents approximately the same fraction of the maximum stable time step. We first compare the unlimited version of BDS_Q with the unlimited version of LPR. Tests over a range of resolutions from 10^2 to 160^2 show that both methods converge at third order accuracy. Furthermore, at each resolution, the accuracy of BDS_Q is a factor of at least 20 better than LPR in both the L^1 and L^∞ norms. When we repeat the experiment with the limited schemes, BDS_Q is approximately

a factor of 10 more accurate on the 10^2 grid but the accuracy of LPR improves relative to BDS_Q to the point that on the 160^2 grid, BDS_Q is less than a factor of two more accurate in the L^1 norm and a factor of 2.03 worse in the L^∞ norm.

We conjecture that this relative loss of accuracy reflects the more restrictive limiting of BDS_Q needed to enforce a maximum principle. To test this hypothesis, we have implemented the WENO reconstruction in [15] and tested it in conjunction with the BDS_Q flux computation. The resulting hybrid scheme for the smooth problems has errors that are an order of magnitude less than LPR in the L^1 and L^∞ norms on a 160^2 grid. However, on a discontinuous advection example, this less restrictive reconstruction leads to approximately 2% overshoot and undershoot. These tests confirm the conjecture and indicate that if we do not need to enforce a maximum principle, a less restrictive reconstruction can be used that will result in reduced errors for smooth problems.

We next provide a comparison to the higher-order ADER schemes of Toro and Titarev [24]. They consider a variable coefficient linear advection example of the form (1), referred to as the frontogenesis problem. The initial conditions are given by

$$s(x, y, t = 0) = \tanh \frac{y}{\delta}, \quad \delta = 1$$

on the domain $(-5, 5)^2$, and the velocity field is constant in time, given by

$$(u, v) = \omega(r)(-y, x),$$

with $\omega(r) = (1/r)U_T(r)$, $U_T(r) = 2.5980762\ \mathrm{sech}^2 r \tanh r$, and $r^2 = x^2 + y^2$. The errors are computed at $t = 4$ as compared to the exact solution,

$$s(x, y, t) = \tanh[y \cos(\omega t) - x \sin(\omega t)].$$

As with the LPR scheme, the stability limit of the ADER schemes appears to be a CFL of 0.5. Here we compare results of BDS_Q at a CFL of 0.9 for comparison to the ADER schemes at CFL of 0.45, which again represents the same fraction of the maximum allowable time step. At a resolution of 100^2, errors in the ADER3 scheme are intermediate between the BDS_Q scheme with and without limiting in both L^1 and L^∞. (The errors are quite close and lie within a narrow band.) We conjecture that this again reflects the more restrictive limiting in BDS_Q needed to enforce a maximum principle. As the grid is refined, the errors in ADER3 improve more rapidly than BDS_Q. In particular ADER3 shows convergence rates higher than third order whereas BDS_Q is between second and third order accurate. The issue here is related to the representation of the velocity field that is used by the different schemes. For the intended applications of BDS_Q, we typically obtain the velocity field from the solution of an elliptic PDE, thus the only characterization of the advection velocity is the integral average of the normal component over an edge.

This restriction leads to a reduction of convergence to less than third order for the frontogenesis problem with BDS_Q. The ADER schemes use derivative information about the velocity field that we assume is not available, allowing them to construct a more accurate solution on finer grids. Unfortunately, this also implies that the ADER schemes cannot be directly applied in the context considered here; it is not clear whether it would be possible to modify the algorithm to work with the restricted velocity information without a loss of accuracy. We have also considered the case when $\delta \to 0$ so that the discrete initial scalar field changes from -1 to 1 across the $y = 0$ interface. We note that BDS_Q was able to treat the discontinuous initial data without undershoot or overshoot over a range of mesh spacings and CFL conditions.

5. Summary and conclusions

We have presented a new finite volume scheme for linear advection in two dimensions that is based on reconstructing an appropriately limited multidimensional biquadratic profile and deriving a flux based on the multidimensional geometry of the characteristics. This scheme, which we refer to as BDS_Q, is third-order accurate for smooth problems and satisfies a maximum principle when the advective velocity field is spatially constant. Numerical evidence shows that the method continues to satisfy the maximum principle in more general circumstances. The new method is compared to two variations of unsplit PPM, which represent state-of-the-art algorithms for general systems of conservation laws. For advection that is not aligned with one of the coordinate axes, the new algorithm has better accuracy than either PPM scheme. Furthermore, the PPM algorithms do not satisfy a maximum principle and are subject to significant overshoot and undershoot. We have also shown that our method is competitive with modern WENO and ADER schemes.

Two aspects of the BDS_Q algorithm differ from the PPM algorithms and are important for guaranteeing a maximum principle. First, the BDS_Q algorithm uses a fully multidimensional limiting, whereas the PPM schemes limit in one direction only. Thus, the reconstructed profile over the entire cell in PPM can introduce new maxima and minima. Also, from a geometric point of view, the transverse flux corrections in the PPM schemes corresponding to $\Gamma^{+,-}$ are not evaluated at the correct location to mimic exact advection of the reconstructed profile. Note, however, that the transverse corrections to the edge states represent a higher-order effect in time so that the differences in the schemes are reduced as the CFL becomes smaller.

The extension of the method presented here to a more general scalar conservation law as done in [3] is straightforward. There are also a number of potential improvements that could be considered for the reconstruction phase of the algorithm, such as establishing a more formal approach to limiting, not only in the present context but also for the construction of higher-order approximations and to avoid limiting

smooth extrema. Additionally of interest would be the extension of this approach to three dimensions; we plan to pursue this in future work.

Acknowledgments

The work at LBNL was supported by the Applied Mathematics Program of the DOE Office of Advanced Scientific Computing Research under the U.S. Department of Energy under contract No. DE-AC02-05CH11231. S. May also received support from the Courant Institute of Mathematical Sciences under the U.S. Department of Energy under contract No. DE-FG02-88ER25053. Visualization was performed using the VisIt visualization software [25].

References

[1] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, and M. Zingale, *CASTRO: A new compressible astrophysical solver, I: Hydrodynamics and self-gravity*, Astrophys. J. **715** (2010), no. 2, 1221–1238.

[2] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome, *A conservative adaptive projection method for the variable density incompressible Navier–Stokes equations*, J. Comput. Phys. **142** (1998), no. 1, 1–46. MR 99k:76096 Zbl 0933.76055

[3] J. B. Bell, C. N. Dawson, and G. R. Shubin, *An unsplit, higher order Godunov method for scalar conservation laws in multiple dimensions*, J. Comput. Phys. **74** (1988), 1–24.

[4] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. A. Zingale, *Adaptive low Mach number simulations of nuclear flame microphysics*, J. Comput. Phys. **195** (2004), no. 2, 677–694.

[5] S. J. Billett and E. F. Toro, *On WAF-type schemes for multidimensional hyperbolic conservation laws*, J. Comput. Phys. **130** (1997), no. 1, 1–24. MR 97i:65148 Zbl 0873.65088

[6] B. Cockburn and C.-W. Shu, *Foreword [to the Proceedings of the First International Symposium on DG Methods]*, J. Sci. Comput. **22/23** (2005), 1–3. MR 2142187

[7] P. Colella, *Multidimensional upwind methods for hyperbolic conservation laws*, J. Comput. Phys. **87** (1990), no. 1, 171–200. MR 91c:76087 Zbl 0694.65041

[8] P. Colella and M. D. Sekora, *A limiter for PPM that preserves accuracy at smooth extrema*, J. Comput. Phys. **227** (2008), no. 15, 7069–7076. MR 2009d:76079 Zbl 1152.65090

[9] P. Colella and P. R. Woodward, *The piecewise parabolic method (ppm) for gas-dynamical simulations*, J. Comput. Phys. **54** (1984), 174–201.

[10] C. Dawson, *Foreword [special issue on discontinuous galerkin methods for incompressible elastic materials]*, Comput. Methods Appl. Mech. Engrg. **195** (2006), 3183.

[11] M. S. Day and J. B. Bell, *Numerical simulation of laminar reacting flows with complex chemistry*, Combust. Theory Modelling **4** (2000), no. 4, 535–556.

[12] A. Kurganov and G. Petrova, *A third-order semi-discrete genuinely multidimensional central scheme for hyperbolic conservation laws and related problems*, Numer. Math. **88** (2001), no. 4, 683–729. MR 2002e:65118 Zbl 0987.65090

[13] R. J. Leveque, *High-resolution conservative algorithms for advection in incompressible flow*, SIAM J. Numer. Anal. **33** (1996), no. 2, 627–665. MR 98b:76049 Zbl 0852.76057

[14] _____, *Finite volume methods for hyperbolic problems*, Cambridge University Press, 2002. Zbl 1010.65040

[15] D. Levy, G. Puppo, and G. Russo, *Compact central WENO schemes for multidimensional conservation laws*, SIAM J. Sci. Comput. **22** (2000), no. 2, 656–672. MR 2001d:65110 Zbl 0967.65089

[16] P. McCorquodale and P. Colella, *A high-order finite-volume method for hyperbolic conservation laws on locally-refined grids*, Commun. Appl. Math. Comput. Sci. **6** (2011), no. 1, 1–25.

[17] G. H. Miller and P. Colella, *A conservative three-dimensional Eulerian method for coupled solid-fluid shock capturing*, J. Comput. Phys. **183** (2002), no. 1, 26–82. MR 2003j:76080 Zbl 1057.76558

[18] S. Noelle, *The MoT-ICE: a new high-resolution wave-propagation algorithm for multidimensional systems of conservation laws based on Fey's method of transport*, J. Comput. Phys. **164** (2000), no. 2, 283–334. MR 2001f:76061

[19] A. Nonaka, A. S. Almgren, J. B. Bell, M. J. Lijewski, C. M. Malone, and M. Zingale, *MAESTRO: An adaptive low Mach number hydrodynamics algorithm for stellar flows*, Astrophys. J. Suppl. **188** (2010), no. 2, 358–383.

[20] G. S. H. Pau, A. S. Almgren, J. B. Bell, and M. J. Lijewski, *A parallel second-order adaptive mesh algorithm for incompressible flow in porous media*, Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. **367** (2009), no. 1907, 4633–4654. MR 2010j:76127 Zbl 1192.76056

[21] J. Saltzman, *An unsplit 3D upwind method for hyperbolic conservation laws*, J. Comput. Phys. **115** (1994), no. 1, 153–168. MR 1300337 Zbl 0813.65111

[22] C.-W. Shu, *High order weighted essentially nonoscillatory schemes for convection dominated problems*, SIAM Rev. **51** (2009), no. 1, 82–126. MR 2010a:65162 Zbl 1160.65330

[23] P. K. Smolarkiewicz and L. G. Margolin, *MPDATA: a finite-difference solver for geophysical flows*, J. Comput. Phys. **140** (1998), no. 2, 459–480. MR 98m:86002 Zbl 0935.76064

[24] E. F. Toro and V. A. Titarev, *ADER schemes for scalar non-linear hyperbolic conservation laws with source terms in three-space dimensions*, J. Comput. Phys. **202** (2005), no. 1, 196–215. MR 2005h:65140 Zbl 1061.65103

[25] *VisIt User's Manual*, version 1.5, Lawrence Livermore National Laboratory, Livermore, CA, 2005.

Received August 23, 2010. Revised February 7, 2011.

SANDRA MAY: may@cims.nyu.edu
Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, Mail Code: 0711, New York, NY 10012, United States

ANDREW NONAKA: AJNonaka@lbl.gov
Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, MS 50A-1148, Berkeley, CA 94720, United States

ANN ALMGREN: ASAlmgren@lbl.gov
Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, MS 50A-1148, Berkeley, CA 94720, United States

JOHN BELL: jbbell@lbl.gov
Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, MS 50A-1148, 1 Cyclotron Road, Berkeley, CA 94720, United States

COMM. APP. MATH. AND COMP. SCI.
Vol. 6, No. 1, 2011

msp

CONDITIONAL PATH SAMPLING FOR STOCHASTIC DIFFERENTIAL EQUATIONS THROUGH DRIFT RELAXATION

PANOS STINIS

We present an algorithm for the efficient sampling of conditional paths of stochastic differential equations (SDEs). While unconditional path sampling of SDEs is straightforward, albeit expensive for high dimensional systems of SDEs, conditional path sampling can be difficult even for low dimensional systems. This is because we need to produce sample paths of the SDE that respect both the dynamics of the SDE and the initial and endpoint conditions. The dynamics of a SDE are governed by the deterministic term (drift) and the stochastic term (noise). Instead of producing conditional paths directly from the original SDE, one can consider a sequence of SDEs with modified drifts. The modified drifts should be chosen so that it is easier to produce sample paths that satisfy the initial and endpoint conditions. Also, the sequence of modified drifts should converge to the drift of the original SDE. We construct a simple Markov chain Monte Carlo algorithm that samples, in sequence, conditional paths from the modified SDEs, by taking the last sampled path at each level of the sequence as an initial condition for the sampling at the next level in the sequence. The algorithm can be thought of as a stochastic analog of deterministic homotopy methods for solving nonlinear algebraic equations or as a SDE generalization of simulated annealing. The algorithm is particularly suited for filtering/smoothing applications. We show how it can be used to improve the performance of particle filters. Numerical results for filtering of a stochastic differential equation are included.

Introduction

The study of systems arising in different areas, from signal processing and chemical kinetics to econometrics and finance [1; 19] often requires the sampling of paths of stochastic differential equations (SDEs) subject to initial and endpoint conditions. While unconditional path sampling of SDEs is straightforward, albeit expensive for high dimensional systems of SDEs, conditional path sampling can be difficult even for low dimensional systems. This is because we need to produce sample paths of the SDE that respect both the dynamics of the SDE and the initial and endpoint

MSC2000: 60G35, 62M20, 65C05, 65C30, 93E10.

Keywords: conditional path sampling, stochastic differential equations, particle filters, homotopy methods, Monte Carlo, simulated annealing.

conditions. An analogous situation arises in ordinary differential equations, where it can be considerably more difficult to create solutions to boundary value problems than it is to construct solutions to initial value problems (see, for example, Chapter 8 in [5]). The problem of conditional path sampling of SDEs has been a subject of active research in recent years and some very interesting approaches have already been developed [2; 19; 23].

The dynamics of a SDE are governed by the deterministic term (drift) and the stochastic term (noise). Instead of producing conditional paths directly from the original SDE, one can consider a sequence of SDEs with modified drifts. The modified drifts should be chosen so that it is easier to produce sample paths that satisfy the initial and endpoint conditions. Also, the sequence of modified drifts should converge to the drift of the original SDE. We construct a simple Markov chain Monte Carlo (MCMC) algorithm that samples, in sequence, conditional paths from the modified SDEs, by taking the last sampled path at each level of the sequence as an initial condition for the sampling at the next level in the sequence. We have called the algorithm the drift relaxation algorithm.

We have used the drift relaxation algorithm to modify a popular filtering method called particle filter [6]. A particle filter is a sequential importance sampling algorithm based on the recursive (online) Bayesian updating of the values of samples (called particles) to incorporate information from noisy observations of the state of a dynamic model. While the particle filter is a very versatile method it may require a very large number of samples to approximate accurately the conditional density of the state of the model. This has led to considerable research [8; 20; 24] into how one can modify a particle filter to make it more efficient (see also [4; 3] for a different approach to particle filtering). As an application of the drift relaxation algorithm we show in Section 2 how it can be used to construct a more efficient particle filter.

The paper is organized as follows. Section 1 presents the drift relaxation algorithm for an SDE conditional path sampling problem. Section 2 shows how to use the algorithm to modify a particle filter. Section 3 contains numerical results for the application of the modified particle filter to the standard example of filtering a diffusion in a double-well potential (more elaborate examples will be presented in [15]). Finally, Section 4 discusses the results as well as current and future work.

1. Conditional path sampling and drift relaxation

Suppose that we are given a system of stochastic differential equations (SDEs)

$$dX_t = a(X_t)\,dt + \sigma(X_t)\,dB_t, \tag{1}$$

Suppose also that we want to construct, in the time interval $[0, T]$, sample paths from (1) such that the endpoints are distributed according to the densities $h(X_0)$

and $g(X_T)$ respectively. Equation (1) can be discretized in the interval $[0, T]$ by some numerical approximation scheme [11]. Suppose that we have discretized the interval $[0, T]$ using a stepsize $\Delta t = T/I$. Let $0 = T_0 < T_1 < \ldots < T_I = T$. To construct conditional paths of (1) we have to sample the density

$$h(X_{T_0}) \prod_{i=1}^{I} p(X_{T_i}|X_{T_{i-1}}) g(X_{T_I}), \tag{2}$$

where $p(X_{T_i}|X_{T_{i-1}})$ is the transition probability from $X_{T_{i-1}}$ at time T_{i-1} to X_{T_i} at time T_i. The density (2) can be sampled using MCMC assuming that the transition densities $p(X_{T_i}|X_{T_{i-1}})$ can be evaluated. However, the major issue with the MCMC sampling is whether it can be performed efficiently [23; 4]. Instead of MCMC sampling directly from the density (2) i.e., starting from an arbitrary initial path and modifying it to become a path corresponding to (2), we can aid the MCMC sampling process by providing the MCMC sampler of the density (2) with a better initial condition.

To this end, consider an SDE system with modified drift

$$dY_t = b(Y_t)\,dt + \sigma(Y_t)\,dB_t, \tag{3}$$

where $b(Y_t)$ can be suitably chosen to facilitate the conditional path sampling problem (see also comments at the end of this section).

Also, consider the collection of $L+1$ modified SDE systems

$$dY_t^l = (1-\epsilon_l) b(Y_t^l)\,dt + \epsilon_l a(Y_t^l)dt + \sigma(Y_t^l)\,dB_t,$$

where $\epsilon_l \in [0, 1]$, $l = 0, \ldots, L$, with $\epsilon_l < \epsilon_{l+1}$, $\epsilon_0 = 0$ and $\epsilon_L = 1$. Note that the zeroth level SDE corresponds to (3) while the L-th level SDE corresponds to the original SDE (1). Also, for the l-th SDE in the sequence we denote as $p_l(Y^l_{T_i}|Y^l_{T_{i-1}})$ the corresponding transition probability. With this notation, $p_L(Y^L_{T_i}|Y^L_{T_{i-1}}) = p(X_{T_i}|X_{T_{i-1}})$.

The main idea behind drift relaxation is that instead of sampling directly a conditional path for the SDE (1), one can sample a conditional path for the modified SDE (3) and gradually morph the path into a path of (1).

Drift relaxation algorithm:

- Sample through MCMC the density $h(Y^0_{T_0}) \prod_{i=1}^{I} p_0(Y^0_{T_i}|Y^0_{T_{i-1}}) g(Y^0_{T_I})$.
- For $l = 1, \ldots, L$ take the last sample path from the $(l-1)$-st level and use it as an initial condition for MCMC sampling of the density

  $$h(Y^l_{T_0}) \prod_{i=1}^{I} p_l(Y^l_{T_i}|Y^l_{T_{i-1}}) g(Y^l_{T_I})$$

  at the l-th level.
- Keep the last sample path at the L-th level.

We repeat here that the levels from 0 to $L-1$ are auxiliary and only serve the purpose of providing the sampler at level L with a better initial condition. The final sampling is performed at the L-th level that corresponds to the original SDE (1).

The drift relaxation algorithm is similar to simulated annealing (SA), used in equilibrium statistical mechanics [12]. However, instead of modifying a temperature as in SA, here we modify the drift of the system. Also, the idea behind drift relaxation resembles the main idea behind homotopy methods used in deterministic optimization problems [7; 10].

Note that there are two ways to utilize the drift relaxation idea. The first one is by sampling the densities for the different levels sequentially as in the algorithm presented above. The second is to consider the $L+1$ systems *in parallel*, sample simultaneously the conditional densities

$$h(Y^l_{T_0}) \prod_{i=1}^{I} p_l(Y^l_{T_i}|Y^l_{T_{i-1}}) g(Y^l_{T_I}), \quad \text{for } l = 0, \ldots, L,$$

and occasionally swap paths between levels (the swapping of paths between levels should be performed in a manner that preserves detailed balance [12]). This approach is in the spirit of parallel tempering used in Monte Carlo sampling [12]. In the current work we have applied the drift relaxation idea only in the form presented in the algorithm above.

We end this section with a brief discussion on the choice of the modified drift. For stochastic gradient flows with transitions between multiple metastable states, one can choose the modified drift as a mollified version of the original drift (see also the discussion in Section 3). This amounts to making the potential wells shallower and thus facilitates the transitions between metastable states. For general problems, one can choose to use for the modified drift a mean-field drift. This has been used successfully by the author to improve the performance of particle filters for multiple target tracking [15].

2. Application to particle filtering

We show in this section how the drift relaxation algorithm can be applied to particle filtering with the aim of bringing the samples closer to the observations.

2.1. ***Generic particle filter.*** Suppose that we are given an SDE system and that we also have access to noisy observations $Z_{T_1}, \ldots, Z_{T_K}$ of the state of the system at specified instants $T_1, \ldots, T_K$. The observations are functions of the state of the system, say given by $Z_{T_k} = G(X_{T_k}, \xi_k)$, where $\xi_k, k = 1, \ldots, K$ are mutually independent random variables. For simplicity, let us assume that the distribution of the observations admits a density $g(X_{T_k}, Z_{T_k})$, i.e., $p(Z_{T_k}|X_{T_k}) \propto g(X_{T_k}, Z_{T_k})$.

The filtering problem consists of computing estimates of the conditional expectation $E[f(X_{T_k})|\{Z_{T_j}\}_{j=1}^k]$, i.e., the conditional expectation of the state of the system given the (noisy) observations. Equivalently, we are looking to compute the conditional density of the state of the system given the observations $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)$. There are several ways to compute this conditional density and the associated conditional expectation but for practical applications they are rather expensive.

Particle filters fall in the category of importance sampling methods. Because computing averages with respect to the conditional density involves the sampling of the conditional density, which can be difficult, importance sampling methods proceed by sampling a reference density $q(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)$, which can be easily sampled and then compute the weighted sample mean

$$E[f(X_{T_k})|\{Z_{T_j}\}_{j=1}^k] \approx \frac{1}{N}\sum_{n=1}^{N} f(X_{T_k}^n)\frac{p(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}{q(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)},$$

or the related estimate

$$E[f(X_{T_k})|\{Z_{T_j}\}_{j=1}^k] \approx \frac{\sum_{n=1}^{N} f(X_{T_k}^n)\frac{p(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}{q(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}}{\sum_{n=1}^{N}\frac{p(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}{q(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}}, \tag{4}$$

where N has been replaced by the approximation

$$N \approx \sum_{n=1}^{N}\frac{p(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}{q(X_{T_k}^n|\{Z_{T_j}\}_{j=1}^k)}.$$

Particle filtering is a recursive implementation of the importance sampling approach. It is based on the recursion

$$p(X_{T_k}|\{Z_{T_j}\}_{j=1}^k) \propto g(X_{T_k}, Z_{T_k})p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k-1}), \tag{5}$$

$$\text{where } p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k-1}) = \int p(X_{T_k}|X_{T_{k-1}})p(X_{T_{k-1}}|\{Z_{T_j}\}_{j=1}^{k-1})dX_{T_{k-1}}. \tag{6}$$

If we set

$$q(X_{T_k}|\{Z_{T_j}\}_{j=1}^k) = p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k-1}),$$

then from (5) we get

$$\frac{p(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)}{q(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)} \propto g(X_{T_k}, Z_{T_k}).$$

The approximation in expression (4) becomes

$$E[f(X_{T_i})|\{Z_{T_j}\}_{j=1}^k] \approx \frac{\sum_{n=1}^{N} f(X_{T_k}^n)g(X_{T_k}^n, Z_{T_k})}{\sum_{n=1}^{N} g(X_{T_k}^n, Z_{T_k})}. \tag{7}$$

From (7) we see that if we can construct samples from the predictive distribution $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k-1})$ then we can define the (normalized) weights

$$W_{T_k}^n = \frac{g(X_{T_k}^n, Z_{T_k})}{\sum_{n=1}^N g(X_{T_k}^n, Z_{T_k})}$$

and use them to weigh the samples, and the weighted samples will be distributed according to the posterior distribution $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k})$.

In many applications, most samples will have a negligible weight with respect to the observation, so carrying them along does not contribute significantly to the conditional expectation estimate (this is the problem of degeneracy [12]). To create larger diversity one can resample the weights to create more copies of the samples with significant weights. The particle filter with resampling is summarized in the following algorithm, due to Gordon et al. [9].

Particle filter:

(1) Begin with N unweighted samples $X_{T_{k-1}}^n$ from $p(X_{T_{k-1}}|\{Z_{T_j}\}_{j=1}^{k-1})$.

(2) *Prediction*: Generate N samples $X_{T_k}'^n$ from $p(X_{T_k}|X_{T_{k-1}})$.

(3) *Update*: Evaluate the weights

$$W_{T_k}^n = \frac{g(X_{T_k}'^n, Z_{T_k})}{\sum_{n=1}^N g(X_{T_k}'^n, Z_{T_k})}.$$

(4) *Resampling*: Generate N independent uniform random variables $\{\theta^n\}_{n=1}^N$ in $(0, 1)$. For $n = 1, \ldots, N$ let $X_{T_k}^n = X_{T_k}'^j$ where

$$\sum_{l=1}^{j-1} W_{T_k}^l \leq \theta^j < \sum_{l=1}^{j} W_{T_k}^l$$

where j can range from 1 to N.

(5) Set $k = k + 1$ and proceed to Step 1.

The particle filter algorithm is easy to implement and adapt for different problems since the only part of the algorithm that depends on the specific dynamics of the problem is the prediction step. This has led to the particle filter algorithm's increased popularity [6]. However, even with the resampling step, the particle filter can still need a lot of samples in order to describe accurately the conditional density $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^{k})$. Snyder et al. [18] have shown how the particle filter can fail in simple high dimensional problems because one sample dominates the weight distribution. The rest of the samples are not in statistically significant regions. Even worse, as we will show in the numerical results section, there are simple examples where not even one sample is in a statistically significant region. In the

next subsection we present how drift relaxation can be used to push samples closer to statistically significant regions.

2.2. ***Particle filter with MCMC step.*** Several authors (see, e.g., [8; 24]) have suggested the use of a MCMC step after the resampling step (Step 4) in order to move samples away from statistically insignificant regions. There are many possible ways to append an MCMC step after the resampling step in order to achieve that objective. The important point is that the MCMC step must preserve the conditional density $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)$.

We begin by noting that one can use the resampling step (Step 4) in the particle filter algorithm to create more copies not only of the good samples according to the observation, but also of the values (initial conditions) of the samples at the previous observation. These values are the ones who have evolved into good samples for the current observation (see more details in [24]). The motivation behind producing more copies of the pairs of initial and final conditions is to use the good initial conditions as starting points to produce statistically more significant samples according to the current observation. This process can be accomplished in two steps. First, Step 4 of the particle filter algorithm is replaced by:

Resampling. Generate N independent uniform random variables $\{\theta^n\}_{n=1}^N$ in $(0, 1)$. For $n = 1, \ldots, N$ let $(X_{T_{k-1}}^n, X_{T_k}^n) = (X_{T_{k-1}}^{\prime j}, X_{T_k}^{\prime j})$ where

$$\sum_{l=1}^{j-1} W_{T_k}^l \le \theta^j < \sum_{l=1}^{j} W_{T_k}^l.$$

Also, with Bayes' rule [24] one can show that the posterior density $p(X_{T_k}|\{Z_{T_j}\}_{j=1}^k)$ is preserved if one samples from the density

$$g(X_{T_k}, Z_{T_k}) p(X_{T_k}|X_{T_{k-1}}),$$

where $X_{T_{k-1}}$ are given by the modified resampling step. This is a problem of conditional sampling for (continuous-time or discrete) stochastic systems. The important issue is to perform the necessary sampling efficiently [4; 24]. We propose to do that here using drift relaxation (see Section 1). The particle filter with MCMC step algorithm is given by:

Particle filter with MCMC step.

(1) Begin with N unweighted samples $X_{T_{k-1}}^n$ from $p(X_{T_{k-1}}|\{Z_{T_j}\}_{j=1}^{k-1})$.

(2) *Prediction*: Generate N samples $X_{T_k}^{\prime n}$ from $p(X_{T_k}|X_{T_{k-1}})$.

(3) *Update*: Evaluate the weights

$$W_{T_k}^n = \frac{g(X_{T_k}^{\prime n}, Z_{T_k})}{\sum_{n=1}^N g(X_{T_k}^{\prime n}, Z_{T_k})}.$$

(4) *Resampling*: Generate N independent uniform random variables $\{\theta^n\}_{n=1}^N$ in $(0, 1)$. For $n = 1, \ldots, N$ let $(X^n_{T_{k-1}}, X^n_{T_k}) = (X'^j_{T_{k-1}}, X'^j_{T_k})$ where

$$\sum_{l=1}^{j-1} W^l_{T_k} \leq \theta^j < \sum_{l=1}^{j} W^l_{T_k}, \quad j = 1, \ldots, N.$$

(5) *MCMC step*: For $n = 1, \ldots, N$ choose a modified drift (possibly different for each n). Construct through drift relaxation a Markov chain for $Y^n_{T_k}$ with initial value $X^n_{T_k}$ and stationary distribution

$$g(Y^n_{T_k}, Z_{T_k}) p(Y^n_{T_k} | X^n_{T_{k-1}}).$$

(6) Set $X^n_{T_k} = Y^n_{T_k}$.

(7) Set $k = k + 1$ and proceed to Step 1.

3. Numerical results

We present numerical results of the particle filter algorithm with MCMC step for the standard problem of diffusion in a double-well potential (more elaborate applications of the method will be presented elsewhere [15]). Our objective here is to show how the generic particle filter's performance can be significantly improved by incorporating the MCMC step via drift relaxation.

The problem of diffusion in a double well potential is described by the scalar SDE

$$dX_t = -4X_t(X_t^2 - 1) + \tfrac{1}{2} dB_t. \tag{8}$$

The deterministic part (drift) describes a gradient flow for the potential $U(x) = x^4 - 2x^2$, which has two minima, at $x = \pm 1$. In the notation of Section 1 we have $a(X_t) = -4X_t(X_t^2 - 1)$ and $\sigma(X_t) = \frac{1}{2}$. If the stochastic term is zero the solution wanders around one of the minima depending on the value of the initial condition. A weak stochastic term leads to rare transitions between the minima of the potential. We have chosen the coefficient $\frac{1}{2}$ to make the stochastic term rather weak. This is done because we plan to enforce the observations to alternate among the minima, and thus check if the particle filter can track these transitions.

The SDE (8) is discretized by the Euler–Maruyama [11] scheme with step size $\Delta t = 10^{-2}$. The initial condition is set to -1 and there is a total of 10 observations at $T_k = k, k = 1, \ldots, 10$. The observations are given by $Z_{T_k} = X_{T_k} + \xi_k$, where $\xi_k \sim N(0, 0.01)$ for $k = 1, \ldots, 10$. Note that we have chosen a rather small variance for the observation noise, which in turn makes the filtering problem more difficult. For this choice of observation noise, the observation density (also called likelihood) is given by

$$g(X_{T_k}, Z_{T_k}) \propto \exp\left[-\frac{(Z_{T_k} - X_{T_k})^2}{2 \times 0.01} \right]. \tag{9}$$

The observations alternate between 1 and -1. In particular, for $k = 1, \ldots, 10$ we have $Z_{T_k} = -1$ if k is odd and $Z_{T_k} = 1$ if k is even. Given that the stochastic term is rather weak, such frequent transitions between the two potential minima are rare.

In order to apply the MCMC step with drift relaxation we need to define the modified drift $b(Y_t)$ for the process Y_t given by

$$dY_t = b(Y_t) + \tfrac{1}{2} dB_t. \tag{10}$$

The modified drift can be the same for all the samples or different for each sample. Since the difficulty in tracking the observations comes from the inability of the original SDE (8) to make frequent transitions between the two minima of the double well, an intuitively appealing choice for $b(Y_t)$ is $b(Y_t) = -\alpha 4 Y_t (Y_t^2 - 1)$, where $0 < \alpha < 1$. This drift corresponds to the potential $W(y) = \alpha(y^4 - 2y^2)$. The potential $W(y)$ has its minima also located at $y = \pm 1$. However, the value of the potential at the minima is $-\alpha$ instead of -1 for the potential $U(x)$. This means that the wells corresponding to the minima of $W(y)$ are shallower than the wells corresponding to the minima of $U(x)$. This makes the transitions between the two wells for the process Y_t more frequent than for the original process X_t. For the numerical experiments we have chosen $\alpha = 0.1$.

The sequence of modified SDEs for the drift relaxation algorithm with L levels is given by

$$dY_t^l = (1 - \epsilon_l) b(Y_t^l) dt + \epsilon_l a(Y_t^l) dt + \tfrac{1}{2} dB_t, \tag{11}$$

where $\epsilon_l \in [0, 1]$, $l = 0, \ldots, L$, with $\epsilon_l < \epsilon_{l+1}$, $\epsilon_0 = 0$ and $\epsilon_L = 1$. For our numerical experiments we chose $L = 10$ and $\epsilon_l = l/10$.

Recall that the density we want to sample during the MCMC step is given by

$$g(X_{T_k}, Z_{T_k}) p(X_{T_k} | X_{T_{k-1}}),$$

where $p(X_{T_k} | X_{T_{k-1}})$ is the transition probability between $X_{T_{k-1}}$ and X_{T_k}. For many applications, sampling directly from $p(X_{T_k} | X_{T_{k-1}})$ may be impossible. Thus, one needs to resort to some numerical approximation scheme to approximate the path between $X_{T_{k-1}}$ and X_{T_k} by a discretized path. However (see [24] for details), even the evaluation of the discretized path's density may not be efficient. Instead, by using the fact that each Brownian path in (8) gives rise to a unique path for X_t [17], we can replace the sampling of $g(X_{T_k}, Z_{T_k}) p(X_{T_k} | X_{T_{k-1}})$ by sampling from the density

$$\exp\left[-\frac{(Z_T - X_T^n(\{\Delta B_i^n\}_{i=0}^{I-1}))^2}{2 \times 0.01} \right] \prod_{i=0}^{I-1} \exp\left[-\frac{(\Delta B_i^n)^2}{2\,\Delta t} \right] =$$

$$\exp\left[-\left(\frac{(Z_T - X_T^n(\{\Delta B_i^n\}_{i=0}^{I-1}))^2}{2 \times 0.01} + \sum_{i=0}^{I-1} \frac{(\Delta B_i^n)^2}{2\,\Delta t} \right) \right], \tag{12}$$

where $\{\Delta B_i^n\}_{i=0}^{I-1}$ are the Brownian increments of the discretized path connecting $X_{T_{k-1}}$ and X_{T_k}. Also, note that the final point X_{T_k} has now become a function of the entire Brownian path $\{\Delta B_i^n\}_{i=0}^{I-1}$. For the numerical experiments we have chosen

$$\Delta t = \frac{T_k - T_{k-1}}{I} = 10^{-2},$$

which, since $T_k - T_{k-1} = 1$, gives $I = 100$.

We use drift relaxation to produce samples from the density (12). The Markov chain at each level of the drift relaxation algorithm is constructed using hybrid Monte Carlo (HMC) [12]. At the l-th level, we can discretize (11), say with the Euler–Maruyama scheme, and the points on the path will be given by

$$Y_{i\Delta t}^{l,n} = Y_{(i-1)\Delta t}^{l,n} + (1-\epsilon_l) b(Y_{(i-1)\Delta t}^{l,n})\Delta t + \epsilon_l a(Y_{(i-1)\Delta t}^{l,n})\Delta t + \tfrac{1}{2}\Delta B_{i-1}^{l,n},$$

for $i = 1, \ldots, I$. We can use more sophisticated schemes than the Euler–Maruyama scheme for the discretization of the simplified SDE (10) at the cost of making the expression for the density more complicated.

We can define a potential $V_{\epsilon_l}(\{\Delta B_i^{l,n}\}_{i=0}^{I-1})$ for the variables $\{\Delta B_i^{l,n}\}_{i=0}^{I-1}$. The potential is given by

$$V_{\epsilon_l}\big(\{\{\Delta B_i^{l,n}\}_{i=0}^{I-1}\big) = \frac{(Z_T - Y_{I\Delta t}^{l,n}(\{\Delta B_i^{l,n}\}_{i=0}^{I-1}))^2}{2\times 0.01} + \sum_{i=0}^{I-1}\frac{(\Delta B_i^{l,n})^2}{2\,\Delta t},$$

and the density to be sampled can be written as

$$\exp\big[-V_{\epsilon_l}\big(\{\Delta B_i^{l,n}\}_{i=0}^{I-1}\big)\big].$$

The subscript ϵ_l is to denote the dependence of the potential on the drift relaxation parameter ϵ_l. In HMC one considers the variables on which the potential depends as the position variables of a Hamiltonian system. In our case we have I position variables so we can define a I-dimensional position vector $\{q_i\}_{i=1}^I$. The next step is to augment the position variables vector by a vector of associated momenta $\{p_i\}_{i=1}^I$. Together they form a Hamiltonian system with Hamiltonian given by

$$H_{\epsilon_l}(\{q_i\}_{i=1}^I, \{p_i\}_{i=1}^I) = V_{\epsilon_l}(\{q_i\}_{i=1}^I) + \frac{p^T p}{2},$$

where $p = (p_1, \ldots, p_I)$ is the vector of momenta. Thus, the momenta variables are Gaussian distributed random variables with mean zero and variance 1. The equations of motion for this Hamiltonian system are given by Hamilton's equations

$$\frac{dq_i}{d\tau} = \frac{\partial H_{\epsilon_l}}{\partial p_i} \quad \text{and} \quad \frac{dp_i}{d\tau} = -\frac{\partial H_{\epsilon_l}}{\partial q_i}, \quad \text{for } i = 1, \ldots, I. \tag{13}$$

HMC proceeds by assigning initial conditions to the momentum variables (through sampling from $\exp(-p^T p/2)$), evolving the Hamiltonian system in fictitious time

τ for a given number of steps of size $\delta\tau$ and then using the solution of the system to perform a Metropolis accept/reject step (more details in [12]). After the Metropolis step, the values of the momenta are discarded. The most popular method for solving the Hamiltonian system, which is the one we also used, is the Verlet leapfrog scheme. In our numerical implementation, we did not attempt to optimize the performance of the HMC algorithm. For the sampling at each level of the drift relaxation process we used 10 Metropolis accept/reject steps and 1 HMC step of size $\delta\tau = 10^{-2}$ to construct a trial path. A detailed study of the drift relaxation/HMC algorithm for conditional path sampling problems outside of the context of particle filtering will be presented in a future publication.

For the chosen values of the parameters for the drift relaxation and HMC steps, the particle filter with MCMC step is about 500 times more expensive per sample (particle) than the generic particle filter. However, we show that this increase in cost per sample is worthwhile. Figure 1 compares the performance of the particle filter with MCMC step with 10 samples and the generic particle filter with 5000 samples. It is obvious that the particle filter with MCMC step follows accurately all the transitions between the two minima of the double-well. On the other hand, the generic particle filter captures accurately only every other observation. It fails to perform the transitions between the two minima of the double-well.

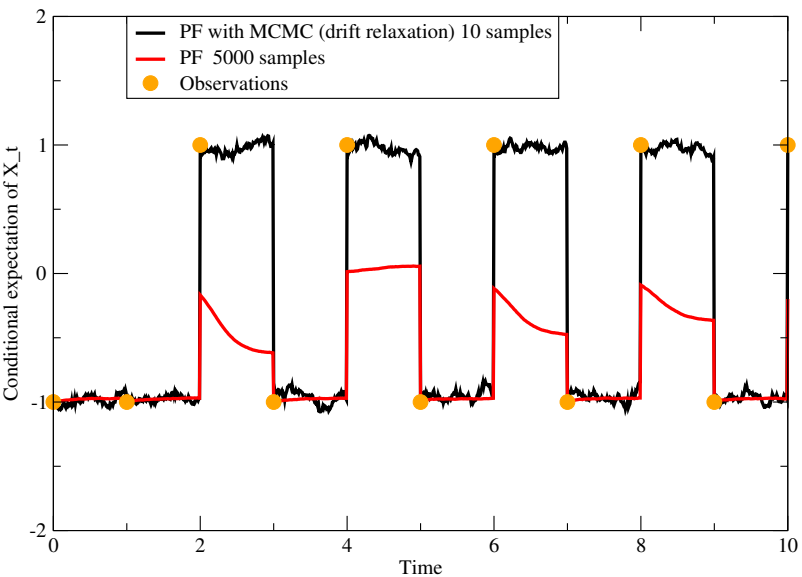


Figure 1. Comparison of the conditional expectation of X_t as computed by the generic particle filter and the particle filter with MCMC step.

Since the particle filter with MCMC step uses only 10 samples the conditional expectation estimate of the hidden signal is not as smooth as the estimate of the generic particle filter, which uses 5000 samples. The generic particle filter needs about 10^5 samples to capture accurately the transitions between the two minima. However, from the 10^5 samples, only 2 or 3 dominate the observation weight distribution at each transition, thus making the use of the generic particle filter very inefficient.

Finally, we compare the performance of the particle filter with MCMC step and drift relaxation to a particle filter with MCMC step without drift relaxation. This comparison is made to examine whether the drift relaxation algorithm offers any advantage over direct sampling of the conditional density (12). The particle filter with MCMC step without drift relaxation involved 110 Metropolis accept/reject steps and 1 HMC step of size $\delta\tau = 10^{-2}$ to construct a trial path for each observation. This makes the computational complexity the same as for the particle filter with drift relaxation.

From Figure 2 one can see that there is an advantage in the use of drift relaxation as far as the conditional expectation estimate is concerned. For the majority of the observations, the MCMC step with drift relaxation gives superior results to the particle filter with MCMC step without drift relaxation. In particular, the particle filter without drift relaxation is not as effective in bringing the samples close to the

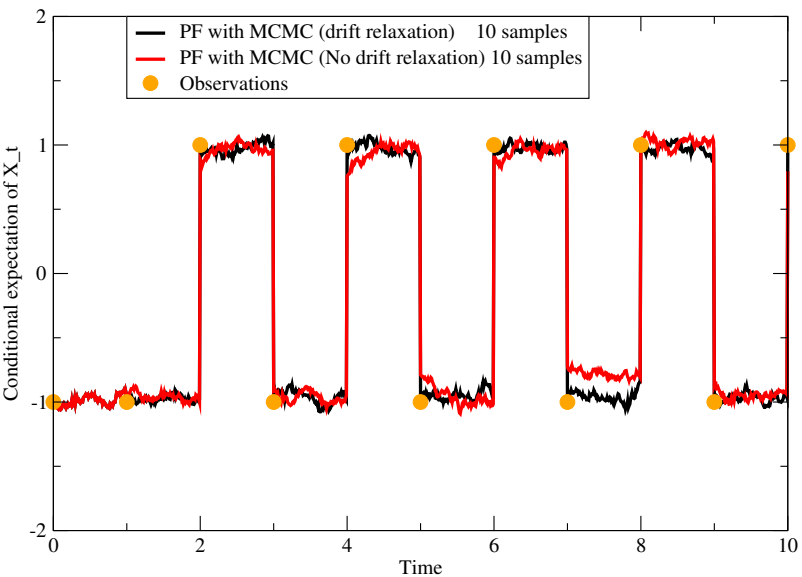


Figure 2. Comparison of the conditional expectation of X_t as computed by the particle filter with MCMC step and drift relaxation and the particle filter with MCMC step without drift relaxation.

observations as the particle filter with drift relations. The mathematical reason for the better performance of the MCMC step with drift relaxation is that the shallower modified potential allows the density of the observation $g(X_{T_k}, Z_{T_k})$ to alter faster the Brownian increments that give rise to the path between the two wells (this is straightforward to see by examination of Hamilton's equations (13) for the HMC sampler). Indeed, for the particle filter with drift relaxation, about 70% of the samples have already crossed from one well to the other after the zeroth level MCMC sampling. Numerical experiments with different choices in the number of drift relaxation levels and/or number of Metropolis accept/reject steps in HMC support the trend shown in Figure 2.

In Figure 3 we plot the error estimate of the conditional expectation estimate of X_t as computed by the particle filter with MCMC step with and without drift relaxation. The error is a measure of the tightness of the distribution of the sample values around the mean. It is obvious that the use of drift relaxation leads to a tighter distribution of the samples around the mean.

In order for the particle filter with MCMC step without drift relaxation to obtain an estimate comparable to the one of the filter with drift relaxation shown in Figure 2, one needs to use about 1000 Metropolis accept/reject steps. This is about 10 times more expensive than the particle filter with drift relaxation. This corroborates the

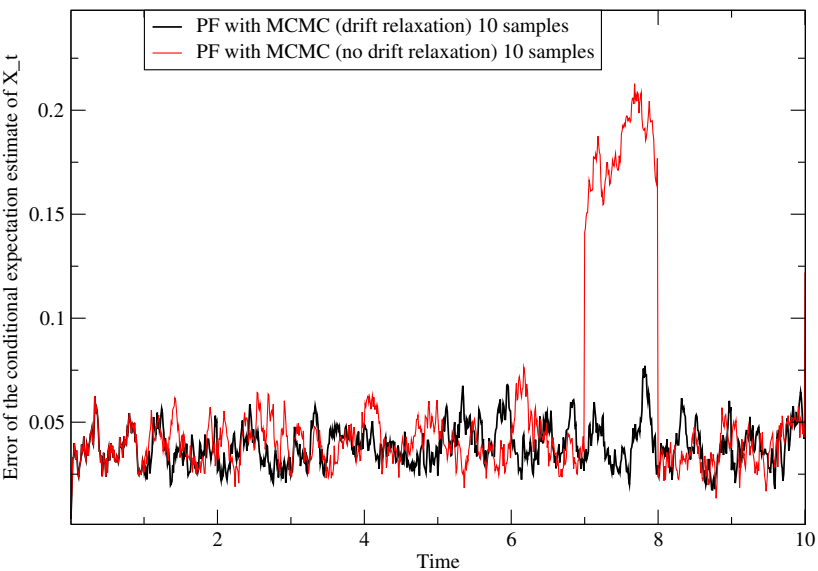


Figure 3. Comparison of the error estimate of the conditional expectation estimate of X_t as computed by the particle filter with MCMC step and drift relaxation and the particle filter with MCMC step without drift relaxation.

conclusion reached in [24], that the use of an MCMC step on its own is not enough to make for a more efficient particle filter. In particular, one has to use an *efficient* algorithm for implementing the MCMC step. A more thorough comparison between the MCMC step with and without drift relaxation will be published elsewhere.

4. Discussion

We have presented an algorithm for conditional path sampling of SDEs. The proposed algorithm is based on drift relaxation, which allows to sample conditional paths from a modified drift equation. The conditional paths of the modified drift equation are then morphed into conditional paths of the original equation. We have called this process of gradually enforcing the drift of the original equation drift relaxation. The algorithm has been used to create a modified particle filter for SDEs. We have shown that the modified particle filter's performance is significantly better than the performance of a generic particle filter.

In the current work, we have examined the application of drift relaxation to the filtering problem of diffusion in a double-well potential, a standard example in the filtering literature. The same algorithm can be applied to the problem of tracking a single target. A problem of great practical interest is that of tracking not only one but multiple moving targets [13; 16; 21; 22]. The multitarget tracking problem is much more difficult than the single-target problem due to the combinatorial explosion of the number of possible target-observation association arrangements. In this context, the accurate tracking of each target becomes crucial. Suppose that only one of the targets is of interest and the rest act as decoys [14]. The inability to track each potential target accurately can lead to ambiguity about the targets' movement if the observations for different targets are close. We have already applied the drift relaxation modified particle filter to multitarget tracking problems with very encouraging results that will appear elsewhere [15].

Acknowledgements

I am grateful to Profs. V. Maroulas and J. Weare for many helpful discussions and comments. I would also like to thank for its hospitality the Institute for Mathematics and its Applications (IMA) at the University of Minnesota where the current work was completed.

References

[1] P. G. Bolhuis, D. Chandler, C. Dellago, and P. L. Geissler, *Transition path sampling: throwing ropes over rough mountain passes in the dark*, Ann. Rev. Phys. Chem. **53** (2002), 291–318.

[2] A. J. Chorin, *Monte Carlo without chains*, Commun. Appl. Math. Comput. Sci. **3** (2008), 77–93. MR 2425547 Zbl 1165.65302

[3] A. J. Chorin, M. Morzfeld, and X. Tu, *Implicit filters for data assimilation*, Comm. Appl. Math. Comp. Sc. **5** (2010), no. 2, 221–240. Zbl 05833566

[4] A. J. Chorin and X. Tu, *Implicit sampling for particle filters*, Proc. Nat. Acad. Sc. USA **106** (2009), 17249–17254.

[5] P. Deuflhard and F. Bornemann, *Scientific computing with ordinary differential equations*, Texts in Applied Mathematics, no. 42, Springer, New York, 2002. MR 2003e:65001 Zbl 1001.65071

[6] A. Doucet, N. de Freitas, and N. Gordon (eds.), *Sequential Monte Carlo methods in practice*, Springer, New York, 2001. MR 2003h:65007

[7] D. Dunlavy and D. O'Leary, *Homotopy optimization methods for global optimization*, technical report SAND2005-7495, Sandia National Labs, 2005.

[8] W. R. Gilks and C. Berzuini, *Following a moving target: Monte Carlo inference for dynamic Bayesian models*, J. R. Stat. Soc. Ser. B Stat. Methodol. **63** (2001), no. 1, 127–146. MR 2001m:62038 Zbl 0976.62021

[9] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, *Novel approach to nonlinear/non-Gaussian bayesian state estimation*, Proc. Inst. Elect. Eng. F **140** (1993), 107–113.

[10] C. Hillermeier, *Nonlinear multiobjective optimization: A generalized homotopy approach*, International Series of Numerical Math., no. 135, Birkhäuser, Basel, 2001. MR 2002b:90002 Zbl 0966.90069

[11] P. E. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, Applications of Mathematics, no. 23, Springer, Berlin, 1992. MR 94b:60069 Zbl 0752.60043

[12] J. S. Liu, *Monte Carlo strategies in scientific computing*, Springer Series in Statistics, no. 16, Springer, New York, 2001. MR 2002i:65006 Zbl 0991.65001

[13] R. P. S. Mahler, *Statistical multisource-multitarget information fusion*, Artech House, London, 2007. Zbl 1126.68080

[14] R. P. S. Mahler and V. Maroulas, *Tracking spawning objects*, preprint, 2010.

[15] V. Maroulas and P. Stinis, *A drift relaxation Monte Carlo approach to particle filtering for multi-target tracking*, preprint, 2010. arXiv 1006.3100v3

[16] W. Ng, J. F. Li, S. J. Godsill, and J. Vermaak, *A hybrid approach for online joint detection and tracking for multiple targets*, 2005, pp. 2126–2141.

[17] B. Oksendal, *Stochastic differential equations: An introduction with applications*, 6th ed., Springer, 2005.

[18] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, *Obstacles to high-dimensional particle filtering*, Mon. Wea. Rev. **136** (2008), 4629–4640.

[19] A. M. Stuart, J. Voss, and P. Wiberg, *Conditional path sampling of SDEs and the Langevin MCMC method*, Commun. Math. Sci. **2** (2004), no. 4, 685–697. MR 2119934

[20] P. J. Van Leeuwen, *Nonlinear data assimilation in geosciences: an extremely efficient particle filter*, Q. J. R. Meteorol. Soc. **136** (2010), 1991–1996.

[21] J. Vermaak, S. Godsill, and P. Perez, *Monte Carlo filtering for multi-target tracking and data association*, IEEE Trans. Aero. Elect. Sys. **41** (2005), no. 1, 309–332.

[22] B.-N. Vo, S. Singh, and A. Doucet, *Sequential Monte Carlo methods for multi-target filtering with random finite sets*, IEEE Trans. Aero. Elect. Sys. **41** (2005), no. 4, 1224–1245.

[23] J. Weare, *Efficient Monte Carlo sampling by parallel marginalization*, Proc. Natl. Acad. Sci. USA **104** (2007), no. 31, 12657–12662.

[24] J. Weare, *Particle filtering with path sampling and an application to a bimodal ocean current model*, J. Comput. Phys. **228** (2009), no. 12, 4312–4331. MR 2010g:86010 Zbl 1165.76045

Received August 30, 2010. Revised February 23, 2011.

PANOS STINIS: stinis@math.umn.edu
School of Mathematics, University of Minnesota, 206 Church St SE, Minneapolis, MN 55455, United States
http://www.math.umn.edu/~stinis/

COMM. APP. MATH. AND COMP. SCI.
Vol. 6, No. 1, 2011

msp

A FREE-SPACE ADAPTIVE FMM-BASED PDE SOLVER IN THREE DIMENSIONS

M. HARPER LANGSTON, LESLIE GREENGARD AND DENIS ZORIN

We present a kernel-independent, adaptive fast multipole method (FMM) of arbitrary order accuracy for solving elliptic PDEs in three dimensions with radiation and periodic boundary conditions. The algorithm requires only the ability to evaluate the Green's function for the governing equation and a representation of the source distribution (the right-hand side) that can be evaluated at arbitrary points. The performance is accelerated in three ways. First, we construct a piecewise polynomial approximation of the right-hand side and compute far-field expansions in the FMM from the coefficients of this approximation. Second, we precompute tables of quadratures to handle the near-field interactions on adaptive octree data structures, keeping the total storage requirements in check through the exploitation of symmetries. Third, we employ shared-memory parallelization methods and load-balancing techniques to accelerate the major algorithmic loops of the FMM. We present numerical examples for the Laplace, modified Helmholtz and Stokes equations.

1. Introduction

Many problems in scientific computing call for the efficient solution to linear partial differential equations with constant coefficients. On regular grids with separable Dirichlet, Neumann or periodic boundary conditions, such equations can be solved using fast, direct methods. For free-space boundary conditions and highly nonuniform source distributions defined on adaptive and/or unstructured grids, alternative approaches are necessary. We describe a *direct* high-order adaptive solver for inhomogeneous linear constant-coefficient PDEs in three dimensions with decay conditions at infinity. A typical case is the Poisson equation

$$-\Delta u = g, \quad \operatorname{supp}(g) \subset \Omega, \tag{1}$$

Langston's work was supported by the U.S. Department of Energy CPES contract; Greengard's was supported in part by the U.S. Department of Energy under contract DEFG0288ER25053; Zorin's was supported by the U.S. Department of Energy (CPES contract) and the National Science Foundation (contract DMS-0612624).

MSC2010: primary 31B10, 65N99, 65R10, 65Y20; secondary 65N15, 76D07.

Keywords: volume integrals, Poisson solver, fast multipole method, adaptive methods, kernel-independent fast multipole method.

where Ω is a bounded domain in $\mathbb{R}^3$, and $u(\boldsymbol{x}) = O(1/|\boldsymbol{x}|)$ as $|\boldsymbol{x}|$ goes to infinity. Our solver uses a kernel-independent fast multipole method (FMM) [60; 61] which can be applied to any PDE for which a free-space Green's function evaluation routine is provided. It handles highly nonuniform sources in an efficient manner, using an adaptive approximation of the right-hand side in (1). The structure of the solver allows for natural integration with FMM-based boundary integral equation techniques, leading to the construction of an adaptive kernel-independent solver for inhomogeneous PDEs in complex geometries, to be described elsewhere.

Related work. For regular grids in separable coordinate systems (rectangles, disks, spheres, etc.), fast methods for constant-coefficient second order PDEs are well established [15; 16]. These methods generally rely on cyclic reduction and/or fast Fourier transforms (FFTs) to achieve nearly linear scaling. For many problems, however, adaptive meshes resulting from adaptive mesh refinement (AMR) strategies are essential [2; 7; 49], and existing solvers typically rely on domain decomposition strategies [23] or multigrid acceleration [18; 37; 40; 44]. For complex geometries, unstructured grid generation techniques are often used [45]. In such cases, both the grid generation process and the solution of the resulting linear systems can be computationally expensive. The lack of regularity in the data structures adds complexities in parallelization as well [1; 18].

A more recent class of methods combines ideas from potential theory with finite difference methods. In [39], fast direct solvers were used on a sequence of refined grids with boundary conditions inherited from the coarser levels. This results in discontinuities at coarse-fine interfaces, which are corrected using a second pass through the grid hierarchy. In [4], the method of local corrections (MLC) [3] was combined with multigrid methods to solve the Poisson equation on a hierarchy of nested grids. The fastest free-space Poisson solver for three-dimensional problems of which we are aware is described in [47]. It first solves local Poisson problems on fine grids using FFT-based techniques and then couples together the solutions on coarser grids using MLC. This approach was shown to be very effective in parallel, with good scaling up to 1024 processors. (A similar two-dimensional scheme is described in [29].) For unstructured meshes, the preceding methods do not apply without significant modification and most fast solvers are based on iterative methods using multigrid or domain decomposition acceleration [13; 14; 17].

Here we concentrate on the integral equation (or, more precisely, the integral transform) viewpoint. Rather than solving (1), for example, we simply compute

$$u(\boldsymbol{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{1}{|\boldsymbol{x} - \boldsymbol{y}|} g(\boldsymbol{y})\, d\boldsymbol{y}. \tag{2}$$

Among the advantages of this approach is the increase in precision in computing derivatives. In PDE-based methods, if first or second derivatives of the solution are

needed, accuracy tends to degrade due to the need for numerical differentiation. Instead, we can differentiate the kernel in (2) and compute derivatives from their integral representation as well. Other advantages are that free-space radiation conditions are automatically satisfied, we can obtain simple *a priori* error estimates, and high order accuracy is straightforward to achieve. However, the computational complexity of a naïve implementation is high: computing the solution u at N points $\boldsymbol{x}$ given N discretization points $\boldsymbol{y}$ requires $O(N^2)$ work. There have been a number of methods proposed to overcome this barrier. These include panel-clustering techniques [11; 36], hierarchical matrices ($\mathcal{H}$,$\mathcal{H}^2$-matrices) [10; 34; 35], the Barnes–Hut method [5], and the fast multipole method (FMM) [20; 32; 25; 51] originally designed for gravitational/Coulomb interactions. These schemes all achieve linear $O(N)$ or nearly linear $O(N \log N)$ scaling. Most of these methods fall into the class of what are often called *tree codes* because they separate near- and far-field interactions on a hierarchy of spatial scales using quadtree (2D) or octree (3D) data structures. Because it can achieve arbitrary precision at modest cost with straightforward error estimates, we concentrate on the FMM in the present setting. The classical FMM is kernel-specific and relies on detailed separation of variables solutions of the governing PDE. While the FMM references above considered the Laplace equation, the Helmholtz equation was subsequently treated in [52]. A three-dimensional version effective for all frequencies (and additional references) can be found in [21]. The modified Helmholtz equation was discussed in [12; 27], and the biharmonic equation in [28; 33; 58]. The Stokes equations are somewhat exceptional, since they can be handled by a sequence of calls to the original (Coulomb) FMM [56; 50]. An attractive alternative that avoids much of the detailed analytic work of these methods is the kernel-independent approach of [60; 61]. In this approach, expansions in special functions are replaced with equivalent source densities. The result is that the same numerical apparatus can be used for a variety of PDEs, and the user need only supply a subroutine for the evaluation of the relevant Green's function.

While the bulk of the work on FMMs over the last two decades has concentrated on particle interactions or the acceleration of boundary integral equation methods, there has been some work on solving inhomogeneous PDEs. One option is to couple the FMM with finite difference methods to allow for fast solvers in complex geometries [46; 48; 9]. While this is a significant improvement in terms of range of applicability over classical fast solvers, these methods require a regular volume mesh on which is superimposed an irregular boundary. Adaptive FMMs for volume source distributions in two dimensions were described in [22; 24; 29]. The present paper extends these two-dimensional schemes to three dimensions, incorporates them into kernel-independent FMMs, and introduces several new performance optimizations. The result is an efficient, adaptive method that is capable of computing volume

integrals in three dimensions for a broad variety of PDE kernels.

Before turning to the method itself, we should also note that there has been a significant body of work in the quantum chemistry community on accelerating volume integral calculations using the FMM, where collections of Gaussians are typically used to describe the charge distribution [53; 59]. These are Poisson problems in free-space but with a different approach to defining the right-hand side.

2. Equations and kernels

Given a linear, constant-coefficient PDE

$$\mathcal{L}(u)(\boldsymbol{x}) = g(\boldsymbol{x}), \tag{3}$$

classical mathematical methods can be used to compute the corresponding Green's function $K(\boldsymbol{x}, \boldsymbol{y})$ in free space, such that

$$u(\boldsymbol{x}) = \int_\Omega K(\boldsymbol{x}, \boldsymbol{y}) g(\boldsymbol{y})\, d\boldsymbol{y}, \tag{4}$$

where Ω is the support of g. $K(\boldsymbol{x}, \boldsymbol{y})$ is, in general, weakly singular; assuming $g(\boldsymbol{x})$ is given at N points and $u(\boldsymbol{x})$ is desired at N points, the nonlocal character of the integral representation, as indicated above, would lead to an $O(N^2)$ solution procedure. Thus, we need both a suitable quadrature approach and a fast algorithm for (4) to yield a useful numerical technique. Assuming this is achieved, a number of advantages follow. First, no linear system needs to be solved. Second, adaptivity is achieved through the approximation of the right-hand side. Third, as mentioned in the previous section, derivatives can be computed without loss of precision. (There is some loss in accuracy for derivatives of order greater than two, since at that point the integral operator becomes hypersingular and some catastrophic cancellation cannot be avoided.) Finally, we have simple a priori error estimates. To see this, let $\hat{g}(\boldsymbol{x})$ be the approximation to $g(\boldsymbol{x})$, and let $\hat{\boldsymbol{Q}}[f](\boldsymbol{x})$ denote the quadrature approximation of $\int_\Omega K(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y})\, d\boldsymbol{y}$. Assuming the near field is computed exactly, the quadrature error satisfies an estimate of the form

$$\left| \hat{\boldsymbol{Q}}[f](\boldsymbol{x}) - \int_\Omega K(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y})\, d\boldsymbol{y} \right| \leq \epsilon \| f \|_1,$$

where ϵ is the approximation error in the FMM. In turn, ϵ is controlled by the parameter p that determines the number of discretization points used for equivalent densities, as described in Section 4.1 and [60].

To estimate the total error, let us assume $\hat{g}(\boldsymbol{x})$ is a k-th order polynomial approximation of the right-hand side,

$$\hat{g}(\boldsymbol{x}) - g(\boldsymbol{x}) \leq \delta = O(h^k),$$

and that

$$\hat{u}(\boldsymbol{x}) = \hat{\boldsymbol{Q}}[\hat{g}](\boldsymbol{x})\,. \tag{5}$$

Then

$$\begin{aligned} e(\boldsymbol{x}) = u(\boldsymbol{x}) - \hat{u}(\boldsymbol{x}) &= \int_\Omega K(\boldsymbol{x},\boldsymbol{y})g(\boldsymbol{y})d\boldsymbol{y} - \int_\Omega \hat{\boldsymbol{Q}}[\hat{g}](\boldsymbol{x}) \\ &\le \int_\Omega K(\boldsymbol{x},\boldsymbol{y})[g(\boldsymbol{y}) - \hat{g}(\boldsymbol{y})]d\boldsymbol{y} + \left| \int_\Omega K(\boldsymbol{x},\boldsymbol{y})\hat{g}(\boldsymbol{y})d\boldsymbol{y} - \hat{\boldsymbol{Q}}[\hat{g}](\boldsymbol{x}) \right| \\ &\le C_1 \|g(\boldsymbol{y}) - \hat{g}(\boldsymbol{y})\|_\infty + \|\hat{g}(\boldsymbol{y})\|_1\, \epsilon, \end{aligned} \tag{6}$$

where

$$C_1 = \max_x \int_\Omega |K(\boldsymbol{x},\boldsymbol{y})|\, d\boldsymbol{y} \le C_1\delta + \|\hat{g}(\boldsymbol{y})\|_1\, \epsilon.$$

This estimate is much sharper than one typically obtained when discretizing the PDE itself, where the order of accuracy is determined by high derivatives of the solution. Here, it depends only on the quality of the approximation of the right-hand side ($\delta = O(h^k)$) and the FMM tolerance (ϵ). Note that the constant C_1 is a bounded quantity determined by the volume of Ω with no dependence on the data. If ϵ is chosen to be of the same order as δ, the scheme is formally k-th order accurate. In practice, it is convenient to decouple the right-hand side approximation error from the FMM tolerance, as above, permitting the user to control them independently.

The principal drawback with the integral formulation is that, when implemented naïvely, the complexity of the approach is quadratic in the number of sample points. FMM algorithms overcome this computational barrier by making systematic use of the smoothness of distant interactions on a hierarchy of spatial scales [6; 24; 30]. The kernel-independent versions of the FMM [60; 61] are particularly useful because of their generality; they make it possible to compute solutions of the form (4) for any (nonoscillatory) elliptic PDE, provided only a module which evaluates the kernel.

After describing the details of the approach, we demonstrate its performance for the Poisson equation

$$-\Delta u(\boldsymbol{x}) = g(\boldsymbol{x}), \tag{7}$$

the modified Helmholtz equation

$$\alpha u(\boldsymbol{x}) - \Delta u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \alpha > 0, \tag{8}$$

and the Stokes equations

$$\begin{aligned} \nabla p(\boldsymbol{x}) - \mu\Delta \boldsymbol{u}(\boldsymbol{x}) &= \boldsymbol{g}(\boldsymbol{x}), \quad \mu > 0, \\ \nabla \cdot \boldsymbol{u}(\boldsymbol{x}) &= 0. \end{aligned} \tag{9}$$

Defining $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{y}$ and $r = \|\boldsymbol{r}\|$, the corresponding kernels in three dimensions are given, respectively, by

$$K(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{4\pi r}, \tag{10}$$

$$K(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{4\pi r} e^{-\sqrt{\alpha} r}, \tag{11}$$

$$K(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{8\pi \mu}\left(\frac{1}{r} I + \frac{\boldsymbol{r} \otimes \boldsymbol{r}}{r^3}\right). \tag{12}$$

The classical FMM is reviewed briefly in Section 3, the kernel-independent method is described in Section 4, and symmetries for optimizing storage are discussed in Section 5. Numerical experiments are presented in Section 6 as well as a brief discussion on extending our method to periodic boundary conditions or including a singular source component along with a smooth background force. We provide additional error analysis in the Appendix as well as a brief summary of how the method is optimized using OpenMP and load-balancing techniques to achieve near-linear strong scaling.

3. Analytic fast multipole method

We briefly review the structure of the original two-dimensional FMM for the case of particle interactions [30]. Given a set of N_{src} charges of strength $g(\boldsymbol{y}_i)$ at locations $(\boldsymbol{y})_i$, the FMM was designed to compute the induced potentials u_j at N_{trg} target locations, $\boldsymbol{x}_j$,

$$u_j = u(\boldsymbol{x}_j) = \sum_{i=1}^{N_{\text{src}}} K(\boldsymbol{x}_j, \boldsymbol{y}_i) g(\boldsymbol{y}_i), \quad j = 1, \ldots, N_{\text{trg}}, \tag{13}$$

where $K(\boldsymbol{x}, \boldsymbol{y}) = -\log|\{\boldsymbol{x} - \boldsymbol{y}\}|/2\pi$. For $N_{\text{src}} \approx N_{\text{trg}} = N$, the FMM decreases the computational cost from $O(N^2)$ to $O(N)$ for fixed user-prescribed accuracy by introducing a hierarchical partition (represented by a tree data structure T) of a regular bounding domain D and two series expansions for each box at each level of the hierarchy. More precisely, the root of T is associated with the entire box D and defined to be at *level* $\ell = 0$. Level $\ell + 1$ is obtained from level ℓ recursively, dividing each subdomain at level ℓ into four equal-sized children. For a regular box B of width H, B's *near field*, $\mathcal{N}^B$, is defined as the set of all boxes in D that lie within a box centered at B of width $3H$. The *neighbor list* L_N^B is defined as the set of boxes in $\mathcal{N}^B$ which share a vertex with B. In the nonadaptive case, $L_N^B = \mathcal{N}^B$. The *far field*, $\mathcal{F}^B$, is the complement of the near field: $\mathcal{F}^B = D \setminus \mathcal{N}^B$. Finally, the *interaction list* L_I^B is the set of children of B's parent's neighbors that are not neighbors themselves. Thus, $L_I^B \subseteq \mathcal{F}^B$. The depth of T is chosen so that the smallest boxes (leaves in T) contain no more than some fixed number of points —

say, s. We first consider uniformly refined trees, where all leaves T are at the same level. Note that the total number of boxes in a 2D quadtree is bounded by $4N/3s$ (and $8N/3s$ in a 3D octree). Thus, if the workload per box is constant, the net algorithm has $O(N)$ complexity.

A *local expansion* is used to represent within each box B the influence of all sources in the far field of B. A *multipole expansion* about the center of B is used to represent the influence of sources *inside* B on boxes in the far field $\mathcal{F}^B$ [30].

The FMM computes the total field at a target point in leaf box B as the sum of (a) the field due to the source points contained in the boxes of the neighbor list L_N^B and (b) the contribution from sources in the far field $\mathcal{F}^B$. The contributions from source points inside the boxes of L_N^B are computed directly using (13), while the contributions from $\mathcal{F}^B$ are obtained by evaluating the local expansion of box B at the target. The essential task of the FMM is the *construction* of the local expansions in a hierarchical manner. This takes place in two steps.

The upward pass. This pass begins at the finest level of the tree data structure, converting charge strengths at source points into multipole expansions for each leaf box; this computation is carried out by the *source-to-multipole* (S2M) operator. Multipole expansions for each nonleaf box B at each coarser level are obtained recursively. More precisely, the multipole expansions for the four children of B are merged into a single expansion about B's center using the *multipole-to-multipole* (M2M) operator.

The downward pass. For each box B, starting at the coarsest level, the local expansion of $\mathcal{F}^B$ is obtained by shifting the local expansion of B's parent to the center of B using the *local-to-local* (L2L) operator and by mapping multipole expansions centered at each box in L_I^B to B's local expansion using the *multipole-to-local* (M2L) operator. For leaf box B, local expansions are then evaluated at each target point using the *local to target* (L2T) operator.

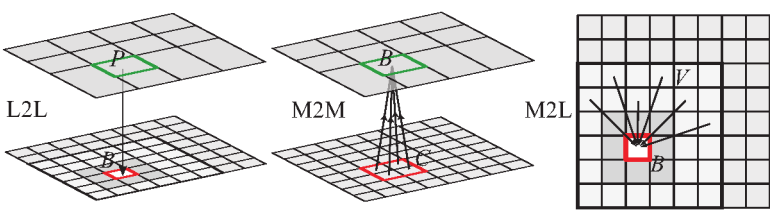


Figure 1. Boxes used by M2M, L2L and M2L operators. For box B at level ℓ, P in the L2L operator represents the parent of B at level $\ell - 1$, and in the M2M operator, C represents the children of B at level $\ell + 1$. Boxes labeled V in the M2L operator reside in L_I^B.

Summary. FMM uses S2M, M2M, M2L, L2L, and L2T linear operators: M2M and L2L operators are determined uniquely by the relative position of a box and its parent; each M2L operator is determined by the relative position of a box in the interaction list; S2M and L2T operators depend on source and target point locations and can be different for each (finest level) box. Figure 1 on the previous page illustrates the data flow involved in the M2M, M2L and L2L operators.

For the Laplace kernel in three dimensions, far-field expansions are represented using a mixture of spherical harmonics [31] and plane-wave representations [32].

We turn now to the kernel-independent approach [60; 61] in order to design a volume integral FMM in three dimensions that can handle a broad class of PDEs.

4. 3D kernel-independent FMM volume integral solver

Given an octree T for our 3D bounding domain D, let $D = \sum\{B_i\}$, $i = 1 \dots M$ be the set of leaf boxes resulting from hierarchical subdivision. For a single-layer kernel K, we compute the integral (4) at some point $\boldsymbol{x}$ as

$$u(\boldsymbol{x}) = \sum_{i=1}^{M} K[B_i, g^{B_i}](\boldsymbol{x}), \tag{14}$$

where $K[B, g^B](\boldsymbol{x}) = \int_B K(\boldsymbol{x}, \boldsymbol{y}) g(\boldsymbol{y})\, d\boldsymbol{y}$, and g^B represents the restriction of the source distribution to the box B.

The principal difference between the approach of this paper and the analytic FMM for point sources is that we use *sampled equivalent densities* instead of classical special functions and series expansions to account for far-field interactions, as in [60; 61]. This requires only a *black-box* kernel evaluation routine and allows for a kernel-independent implementation. A second difference between the approach of this paper and prior kernel-independent FMM schemes is that we are dealing with a continuous source distribution rather than a collection of point-like particles. To extend the method of [60; 61] to this setting, we use polynomial basis functions to approximate the source distribution g on each leaf box, following the two-dimensional approach of [22; 29; 24]. More precisely, we assume that the input source is given on each leaf box B by a polynomial g^B of degree $k+1$ with coefficients γ^B:

$$g^B = \sum_{j=1}^{N_k} \gamma_j^B \beta_j(2^\ell(\boldsymbol{x} - \boldsymbol{c}_B)), \tag{15}$$

where β_j are polynomial basis functions, ℓ is the depth of the box B ($\ell = 0$ at the root of T), and $\boldsymbol{c}_B$ is its center. We use monomials for low-order accuracy and tensor-product Chebyshev polynomials for higher-order accuracy. The number of coefficients is $N_k = k(k+1)(k+2)/6$ for each scalar source function g. We describe an interpolation scheme to convert a set of source values defined on a

grid of sample points to a polynomial representation in Section 4.6. As output, our algorithm can return either point values of the potential at each target point or a polynomial approximation of the potential on each leaf box (which can then be evaluated at arbitrary locations).

To simplify the exposition, we present our algorithm first for a uniformly refined octree of depth ℓ and then discuss the changes necessary for the adaptive octree case separately. The final algorithmic steps are outlined in Section 4.8, and we briefly discuss how the major loops are optimized for shared-memory parallelization in the Appendix (Section A.4).

4.1. ***Equivalent densities.*** The kernel-independent approach to translation operators is based on the following idea. For kernel K, suppose we have an arbitrary (smooth or nonsmooth) source distribution g_s in a volume Ω_s with surface Γ_s. Let Γ_t denote an auxiliary surface in the exterior of Γ_s, and let Γ_{check} denote yet another auxiliary surface in the exterior of Γ_t. Finally, let E denote the exterior of Γ_{check}. We will compute a charge density ϕ_t on Γ_t such that the potentials $K[\Omega_s, g_s]$ and $K[\Gamma_t, \phi_t]$ coincide in E. This is always possible if the exterior Dirichlet problem on Γ_t has a unique solution and the exterior field can be represented in terms of a single layer potential.[1]

Remark. For some problems, such as the Helmholtz equation, a combination of single and double layer sources may be required because of nonphysical resonances in the single layer representation, but it is generally sufficient for nonoscillatory kernels ([41] for the Poisson equation, [42] for the Stokes equations).

Our goal is to use $K[\Gamma_t, \phi_t]$ to represent the far-field instead of a multipole expansion. For this, we let Γ_{check} approximate the outer boundary of the neighbor list L_N^B and solve a Fredholm integral equation of the first kind for ϕ_t,

$$K[\Gamma_t, \phi_t](\boldsymbol{x}) = K[\Omega_s, g_s](\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \Gamma_{\text{check}}. \tag{16}$$

Having matched the field on Γ_{check}, the fields will match in the exterior E (with precise estimates depending on the specific kernel). We refer to Γ_t as an *equivalent surface* with *equivalent density* ϕ_t, and Γ_{check} as a *check surface*. In the case when the original density is concentrated on the surface Γ_s, then (16) can be written as

$$K[\Gamma_t, \phi_t](\boldsymbol{x}) = K[\Gamma_s, \phi_s](\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \boldsymbol{x}_t. \tag{17}$$

We match the field created by charges *outside* the near neighbors of a box by a discretized layer potential defined on a surface enclosing the box, and a different equivalent density will be used to replace the local expansion. The number of samples used to represent the equivalent density is the analog of the number of

[1] For some kernels (Stokes), a low-dimensional nullspace may need to be eliminated.

expansion terms in a classical FMM. Γ_t and Γ_{check} are cubic surfaces, uniformly sampled at p locations. In discretized form, (17) can be written as

$$\boldsymbol{K}^{\Gamma_t,\boldsymbol{x}_t}\phi_t = \boldsymbol{K}^{\Gamma_s,\boldsymbol{x}_t}\phi_s, \tag{18}$$

where ϕ_s and ϕ_t are vectors of point-sampled densities, and $\boldsymbol{K}^{a,b}$ are matrices with entries given by $\boldsymbol{K}^{a,b}_{ij} = K(a_i, b_j)$ for sample points a_i and a_j on surfaces a and b. For known ϕ_s and solving for ϕ_t, (18) is a discretization of a Fredholm equation of the first kind. For large p, linear systems may be poorly conditioned; in such cases, we choose to utilize Tikhonov regularization methods [41] to invert $\boldsymbol{K}^{\Gamma_t,\boldsymbol{x}_t}$. We discuss this approach and its accuracy in the Appendix (sections A.1 and A.3).

Kernel invariance and matrix precomputation. For all equations we consider, the kernels are invariant with respect to a rigid transformation $\mathcal{T}$: for scalar kernels, $K(\mathcal{T}\boldsymbol{x}, \mathcal{T}\boldsymbol{y}) = \mathcal{T}K(\boldsymbol{x}, \boldsymbol{y})$, and for matrix kernels, $K(\mathcal{T}\boldsymbol{x}, \mathcal{T}\boldsymbol{y}) = \mathcal{T}K(\boldsymbol{x}, \boldsymbol{y})\mathcal{T}^T$. Hence, all matrices $\boldsymbol{K}$ need to be computed only once for each class of pairs of equivalent surfaces, closed with respect to a specific $\mathcal{T}$. Furthermore, many kernels are *homogeneous*: for all $c > 0$, there exists a *scaling exponent* $r \neq 0$ such that $K(c\boldsymbol{x}, c\boldsymbol{y}) = c^r K(\boldsymbol{x}, \boldsymbol{y})$, further reducing the number of classes of surface pairs requiring separate matrices. We consider optimizations due to invariance for each translation operator in the next sections, assuming scalar kernels for simplicity, although our implementation can handle matrix kernels.

4.2. ***Upward pass.*** For the upward pass, recall now that for each leaf box, the *sources* are polynomials approximating the source distribution. For consistency with the FMM summary above, we use S, M, etc. in describing translation operator names.

Source to multipole S2M *translations.* For each leaf box B, we choose $\boldsymbol{y}^{B,u}$, the *upward equivalent surface*, and $\boldsymbol{x}^{B,u}$, the *upward check surface*, as in [60]. Equation (16) for upward equivalent density $\phi^{B,u}$ in this case becomes

$$K[\boldsymbol{y}^{B,u}, \phi^{B,u}](\boldsymbol{x}) = K[B, g^B](\boldsymbol{x}), \tag{19}$$

$$K[B, g^B](\boldsymbol{x}) \approx \sum_{j=1}^{N_k} \gamma_j^B F_j^B(\boldsymbol{x}), \tag{20}$$

$$\text{where } F_j^B(\boldsymbol{x}) = \int_B \beta_j(2^\ell(\boldsymbol{y} - \boldsymbol{c}_B))K(\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{y} \quad \text{for } \boldsymbol{x} \in \boldsymbol{x}^{B,u}. \tag{21}$$

By translation invariance, $F_j^B(\boldsymbol{x})$ depends only on the choice of β_j and level ℓ of B. To evaluate the integrals in (21), we use adaptive Gaussian quadrature [8]. In matrix form, the Nyström discretization of (19)–(21) at p sample points $\boldsymbol{\phi}^{B,u}$ on $\boldsymbol{y}^{B,u}$ yields

$$\boldsymbol{K}^B_{\text{S2M}}\boldsymbol{\phi}^{B,u} = \boldsymbol{F}^B_{\text{S2M}}\gamma^B, \tag{22}$$

where $\boldsymbol{F}^B_{\mathrm{S2M}}$ is the matrix of precomputed weights (21) and $\boldsymbol{K}^B_{\mathrm{S2M}}$ is the matrix with entries $K(\boldsymbol{x}_i, \boldsymbol{y}_j)$, $i = 1 \ldots p$, $j = 1 \ldots p$. Solving for $\boldsymbol{\phi}^{B,u}$,

$$\boldsymbol{\phi}^{B,u} = (\boldsymbol{K}^B_{\mathrm{S2M}})^{-1} \boldsymbol{F}^B_{\mathrm{S2M}} \gamma^B = \boldsymbol{T}^B_{\mathrm{S2M}} \gamma^B. \tag{23}$$

For a uniformly refined tree, $\boldsymbol{T}^B_{\mathrm{S2M}}$ depends only on ℓ, so one matrix is computed. Figure 2, left, illustrates the computation of $\phi^{B,u}$ from γ^B.

*Multipole to multipole (*M2M*) translations.* M2M translation operators translate $\phi^{C,u}$ at a child box C to $\phi^{B,u}$ for the parent box B, shown in Figure 2, right: for all $\boldsymbol{x} \in \boldsymbol{x}^{B,u}$,

$$K[\boldsymbol{y}^{B,u}, \phi^{B,u}](\boldsymbol{x}) = \sum_C K[\boldsymbol{y}^{C,u}, \phi^{C,u}](\boldsymbol{x}),$$

or, in matrix form,

$$\boldsymbol{K}^{B,B}_{\mathrm{M2M}} \boldsymbol{\phi}^{B,u} = \sum_C \boldsymbol{K}^{C,B}_{\mathrm{M2M}} \boldsymbol{\phi}^{C,u}. \tag{24}$$

Similar to the S2M computations, these systems are solved as

$$\boldsymbol{\phi}^{B,u} = \sum_C (\boldsymbol{K}^{B,B}_{\mathrm{M2M}})^{-1} \boldsymbol{K}^{C,B}_{\mathrm{M2M}} \boldsymbol{\phi}^{C,u} = \sum_C \boldsymbol{T}^{C,B}_{\mathrm{M2M}} \boldsymbol{\phi}^{C,u}. \tag{25}$$

For any two children C_1 and C_2, rotation R maps C_1 to C_2; therefore, only *one* $\boldsymbol{T}^{C,B}_{\mathrm{M2M}}$ is computed per level, with the contribution to $\boldsymbol{\phi}^{B,u}$ from any other child obtained by composing this matrix with an appropriate permutation of $\phi^{C,u}$. Further, for *homogeneous* kernels, only one matrix is stored at a single level ℓ and scaled as necessary.

4.3. ***Downward pass.*** In the downward pass *downward equivalent densities* are computed through the M2L, L2L, and L2T operators.

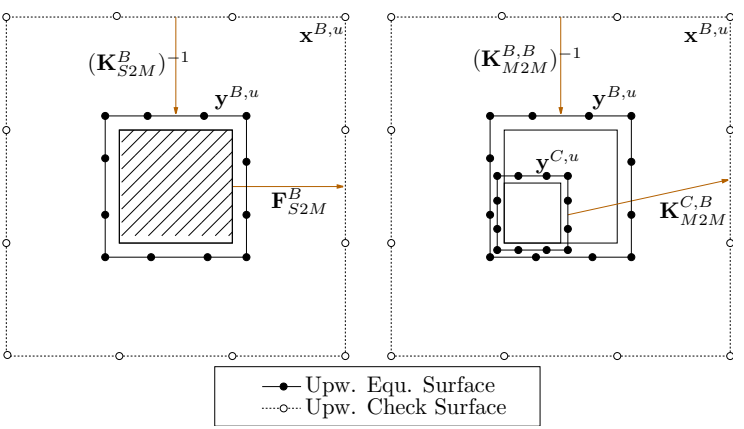


Figure 2. Kernel-independent FMM translation operators for S2M (left) and M2M (right).

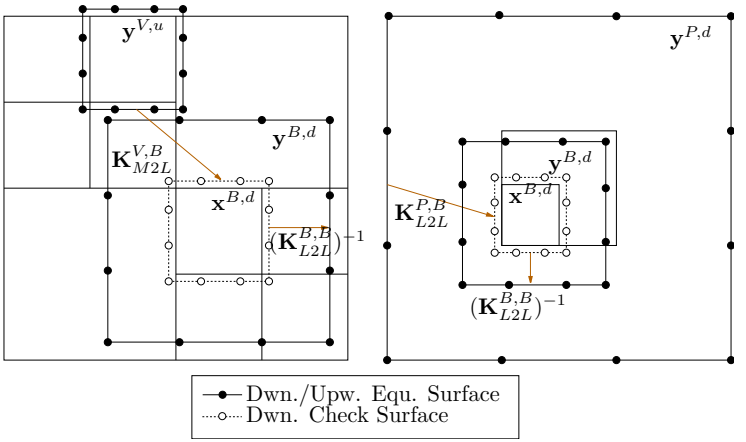


Figure 3. Kernel-independent FMM translation operators for M2L (left) and L2L (right).

*Multipole to local (*M2L*) translations.* For any box B, M2L operators (Figure 3, left) translate $\phi^{V,u}$, approximating the field of sources inside of $V \in L_I^B$, to a *downward equivalent density* $\phi^{B,d}$. In this case, we seek to induce identical potentials *inside* of B, effectively swapping upward equivalent and check surfaces to obtain downward equivalent and check surfaces: $\boldsymbol{y}^{B,d} = \boldsymbol{x}^{B,u}$ and $\boldsymbol{x}^{B,d} = \boldsymbol{y}^{B,u}$. Equation (17) takes the form

$$K[\boldsymbol{y}^{B,d}, \phi^{B,d}](\boldsymbol{x}) = \sum_V K[\boldsymbol{y}^{V,u}, \phi^{V,u}](\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \boldsymbol{x}^{B,d}, \tag{26}$$

where $\phi^{B,d}$ is discretized at p uniformly spaced samples on $\boldsymbol{y}^{B,d}$. The right-hand side of (26) is computed and stored as a *downward check potential*, $u^{B,d}$ at $\boldsymbol{x}^{B,d}$, and $\boldsymbol{\phi}^{B,d}$ is recovered after the L2L contribution is added.

$$u^{B,d}_{\text{M2L}} = \sum_V \boldsymbol{K}^{V,B}_{\text{M2L}} \boldsymbol{\phi}^{V,u}. \tag{27}$$

We efficiently evaluate $u^{B,d}$ with FFTs by treating densities as being defined on extensions of $\boldsymbol{y}^{V,u}$ and $\boldsymbol{x}^{B,d}$ to 3D Cartesian grids with zero values in the interior. This results in $O(p^3/2)$ sample locations, and the computational cost of $O(p^3/2 \log p)$ for evaluation.

There are at most 189 possible locations for $V \in L_I^B$ relative to any particular B; however, using translation and rotation invariance of the kernel as discussed in Section 5, we store at most 16 total $\boldsymbol{K}^{V,B}_{\text{M2L}}$ matrices for a homogeneous kernel.

*Local to local (*L2L*) translations.* Contributions from $\mathcal{F}^B \setminus L_I^B$ are captured through the local field computed for B's parent box, P, using L2L operators (Figure 3,

right). We translate $\phi^{P,d}$ at $\boldsymbol{y}^{P,d}$ to $\phi^{B,d}$ at $\boldsymbol{y}^{B,d}$ using the equation

$$K[\boldsymbol{y}^{B,d}, \phi^{B,d}](\boldsymbol{x}) = K[\boldsymbol{y}^{P,d}, \phi^{P,d}](\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \boldsymbol{x}^{B,d}. \tag{28}$$

The right-hand side is computed as a contribution to $u^{B,d}$, so (28) for B at depth ℓ becomes

$$u^{B,d}_{\mathrm{L2L}} = \boldsymbol{K}^{P,B}_{\mathrm{L2L}} \boldsymbol{\phi}^{P,d}, \tag{29}$$

such that

$$\boldsymbol{\phi}^{B,d} = (\boldsymbol{K}^{B,B}_{\mathrm{L2L}})^{-1}\big(u^{B,d}_{\mathrm{M2L}} + u^{B,d}_{\mathrm{L2L}}\big). \tag{30}$$

The precomputation of matrix $\boldsymbol{K}^{P,B}_{\mathrm{L2L}}$ is completely analogous to $\boldsymbol{K}^{C,B}_{\mathrm{M2M}}$, with parent and child swapped.

*Local to grid target (*L2T*) translations.* For each leaf box B, we evaluate $u^{B,g}$ at grid locations, $\boldsymbol{x}^{B,g}$. At depth ℓ, $\phi^{B,d}$ accounts for all contributions from $\mathcal{F}^B$ while direct near-field calculations (discussed in detail in Section 4.4) account for the contributions from $\mathcal{N}^B$. The far-field potential is computed using L2T operators (Figure 4).

$$u(\boldsymbol{x}) = K[\boldsymbol{y}^{B,d}, \boldsymbol{\phi}^{B,d}](\boldsymbol{x}), \quad \boldsymbol{x} \in \boldsymbol{x}^{B,g},$$

or, in matrix form

$$u^{B,g} = \boldsymbol{K}^{B,B}_{\mathrm{L2T}} \boldsymbol{\phi}^{B,d}. \tag{31}$$

For a uniformly refined tree, all leaves are at the same level, so we precompute and store one $\boldsymbol{K}^{B,B}_{\mathrm{L2T}}$ matrix.

4.4. ***Near-field interactions.*** After the far-field contributions are computed, the final step is to compute near-field interactions for leaf boxes. This is the most expensive step in the computation, if carried out naïvely, and it is essential to

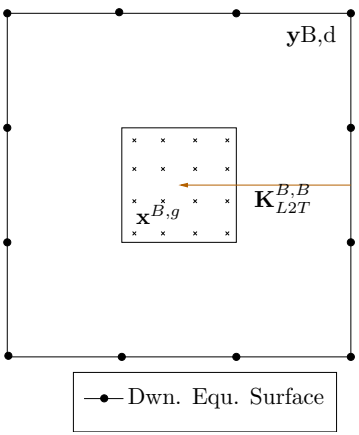


Figure 4. Kernel-independent FMM translation operators for L2T.

optimize this part of the algorithm. For each leaf box B, we need to compute the influence of the volume density g^U for every box $U \in L_B^N$ (the near field boxes). Given a polynomial approximation γ^U to g^U, we evaluate the potential on an $n \times n \times n$ grid of samples $\boldsymbol{x}^{B,g}$ on B, which we then add to the far field contribution computed in (31).

The principal mechanism to accelerate this step is based on the observation that we may use a regular grid pattern of points in B, permitting the use of precomputation. More precisely,

$$u^{B,g}(\boldsymbol{x}) = \sum_U K[U, g](\boldsymbol{x}) = \sum_U \sum_{j=1}^{N_k} \gamma_j^U F_j^{U,B}(\boldsymbol{x}), \tag{32}$$

$$F_j^{U,B}(\boldsymbol{x}) = \int_U \beta_j \left(2^\ell(\boldsymbol{y} - \boldsymbol{c}_U)\right) K(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y} \quad \text{for } \boldsymbol{x} \in \boldsymbol{x}^{B,g}, \tag{33}$$

where $\boldsymbol{c}_U$ is the center of box U. We evaluate $u^{B,g}$ on a uniform grid $\boldsymbol{x}_i^{B,g}$, $i = 1 \ldots n^3$ for $n < 6$ and on a tensor product Chebyshev grid for $n > 6$ to avoid condition problems, as discussed in Section A.2 of the Appendix. In matrix form (32) becomes

$$u^{B,g} = \sum_U \boldsymbol{F}^{U,B} \gamma^U. \tag{34}$$

For a uniform octree, there are at most 27 possible locations for $U \in L_N^B$ with respect to B itself; using symmetries, however, only 4 are unique up to translation and rotation (Section 5). As in the S2M computations, adaptive Gaussian quadrature [8] is used to precompute and store the weights for these matrices. This can be done to machine precision for the function u and its first or second derivatives.

As each leaf box, B is not dependent on the near-field computations of any leaf box in T, it can quickly be seen how even the simplest approaches can take advantage of parallel architectures in the near-field computations. In Section A.4 of the Appendix, we discuss how we use OpenMP [19] and load-balancing approaches to parallelize the near-field and other computational steps of the FMM for shared-memory, multiprocessor architectures.

4.5. ***Polynomial approximation of the solution.*** In order to compute the value of u^B at an arbitrary point in the box, it is convenient to approximate it as a polynomial υ^B using a least-squares fit: minimizing

$$\sum_{i=1}^{n^3} \left\| u^B(\boldsymbol{x}_i) - \sum_{j=1}^{N_n} \upsilon_j^B \beta_j(\boldsymbol{x}_i - \boldsymbol{c}_B) \right\|^2 \quad \text{for } \boldsymbol{x}_i \in \boldsymbol{x}^B, \tag{35}$$

where $\beta_j \in \{P_a(x) P_b(y) P_c(z), 0 \le a + b + c \le n - 1\}$ using either a monomial or Chebyshev polynomial basis, depending on the desired order n. For $n \le 6$ it is more convenient to use regular grids, while for $n > 6$ Chebyshev grid points provide

greater stability. In Section A.2 we demonstrate the accuracy of equispaced points and Chebyshev points for $n = 4, 6, 8$. For box B at depth ℓ, if Γ is the matrix with entries $\Gamma_{ij} = \beta_j(2^\ell(\boldsymbol{x} - \boldsymbol{c}_B))$, (35) leads to the equation $\upsilon^B = \Gamma^{(+)} u^{B,g}$, where the pseudoinverse $\Gamma^{(+)}$ needs to be precomputed only once as it does not depend on the kernel and is scale-invariant in *all* cases; that is, $\Gamma_{ij} = \beta_j(\boldsymbol{x}_i^*)$ where $\boldsymbol{x}_i^*$ are grid points in $B^* = [-1, 1]^3$. Once the v_j^B are known, we can evaluate the solution at an arbitrary point $\boldsymbol{x}_t \in B$ as

$$u(\boldsymbol{x}_t) = \sum_{j=1}^{N_n} \upsilon_j^B \beta_j(\boldsymbol{x}_t - \boldsymbol{c}_B). \tag{36}$$

In general, we assume that k, the order of the approximation γ^B of the force g^B, is equal to n, the order of approximation of υ^B; however, as source and target locations need not be the same, k and n can be different.

4.6. ***Polynomial force approximation from grid samples.*** We have assumed the right-hand side is already given as a polynomial. However, if the force is available in another form (e.g., as samples on an AMR grid or polynomials on an unstructured finite element grid), we need simply to build a k-th order approximation of the right-hand side to the desired tolerance at regular grid points on each leaf node, followed by conversion to a polynomial representation, as in the preceding section. We view this step as outside the scope of the present paper.

4.7. ***Nonuniform source distributions and adaptive FMM.*** For nonuniform source distributions, leaf boxes may appear at different levels, leading to several additional types of interactions between boxes that need to be taken into account. For adaptive octrees, the number of relative positions of boxes one needs to consider can become very large. To avoid storing large number of precomputed matrices, we consider *level-restricted refinement*: we require adjacent leaf boxes be within one level of each other, a common restriction in tree codes and structured grids. Many fast approaches exist to convert arbitrary octrees to ones satisfying this constraint [55]; we currently use a straightforward sequential algorithm similar to [24].

Lists for adaptive FMM. Our definitions and notation follow [26; 30; 31]. For leaf box B, we define U and W lists:

- The *U-list*, L_U^B, consists of leaves adjacent to B, including itself; $L_U^B = L_N^B$ for uniform trees.
- The *W-list*, L_W^B, is the set of descendants of B's neighbors, not adjacent to B, but whose parents are adjacent to B. For any $W \in L_W^B$, W is at a finer level than B and $W \in \mathcal{N}^B$ (conversely, $B \in \mathcal{F}^W$).

For leaf and nonleaf boxes B, we define V and X lists.

- The *V-list*, L_V^B, is the set of B's parent's neighbor's children, not adjacent to B. $L_V^B = L_I^B$ for uniform trees.
- The *X-list*, L_X^B, is the set of boxes A such that $B \in L_W^A$.

We note that

$$B \in L_U^A \iff A \in L_U^B, \quad B \in L_V^A \iff A \in L_V^B, \quad B \in L_W^A \iff A \in L_X^B.$$

An example domain with labeled lists is shown in [60]; possible positions of boxes in the L_U^B, L_V^B, L_W^B and L_X^B are shown in Figure 5.

By the following lemma, for level-restricted trees, boxes in W and X lists have finite possible positions.

Lemma 4.1. *For a level-restricted tree T in which all neighboring leaf boxes are within one level of each other in the octree, for a box, B, all boxes in L_W^B and L_X^B must also be within one level of B.*

Proof. For box B assume there exists $W \in L_W^B$ such that $\ell_W - \ell_B \geq 2$. Then W's parent, P_W, satisfies $\ell_{P_W} - \ell_B \geq 1$, so there exist descendants D of P_W with $D \in L_U^B$ and $\ell_D - \ell_B \geq 2$, violating our tree-level restriction. Thus, $\ell_W - \ell_B \leq 1$. Since $W \in L_W^B$ implies $B \in L_X^W$, we have $\ell_B - \ell_X \leq 1$ for all $X \in L_X^B$. □

Boxes in L_U^B and L_V^B are handled as boxes in L_N^B and L_I^B, respectively, are in the uniform case. For leaf box B, if $W \in L_W^B$, then $W \notin \mathcal{F}^B$; therefore, W's contribution to B is not accounted for through its parent, P_B, but since $B \in \mathcal{F}^W$, we can evaluate $\phi^{W,u}$ at $\boldsymbol{x}^{B,g}$. Hence, using notation analogous to other operators, M2T operators need to be defined. Further, for $X \in L_X^B$, $B \in \mathcal{N}^X$, but $X \in \mathcal{F}^B$. Thus, we need to

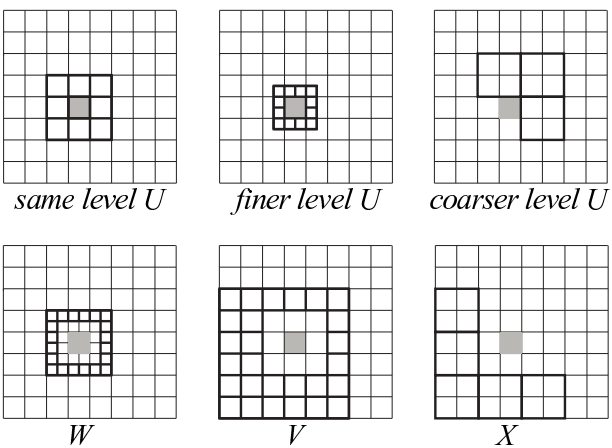


Figure 5. Possible box positions for different lists in a level-restricted tree in 2D. The configurations in 3D are analogous.

evaluate contributions from X directly but can apply them to $\phi^{B,d}$; that is, we need to define an S2L operator.

To summarize, for adaptive FMM, in addition to M2M, M2L, L2L, and L2T, two additional operators, M2T and S2L need to be defined, and S2M and near-field (S2T) operators need to handle leaf boxes at arbitrary levels. We begin by describing changes to S2M and S2T operators and follow with a discussion of M2T and S2L operators.

S2M *operators for the adaptive case.* For *homogeneous* kernels, we store a *single* matrix $\boldsymbol{T}^{B^*}_{\mathrm{S2M}}$, scaling for level ℓ as was done for the M2M and L2L operators. Let $B^* = [-1, 1]^3$ at $\ell = 0$. Then, for $\boldsymbol{x} \in B$ at level ℓ, let $\boldsymbol{x}^* = 2^\ell(\boldsymbol{x} - \boldsymbol{c}^B)$ for $\boldsymbol{x}^* \in B^*$. For scaling exponent r, we have $K(\boldsymbol{x}_i, \boldsymbol{y}_j) = 2^{-r\ell} K(\boldsymbol{x}_i^*, \boldsymbol{y}_j^*)$, and (21) becomes

$$F_j^B(\boldsymbol{x}) = 2^{-(r+2)\ell} K[B^*, \beta_j](\boldsymbol{x}^*) = 2^{-(r+2)\ell} F_j^{B^*}(\boldsymbol{x}^*) \quad \text{for all } \boldsymbol{x} \in \boldsymbol{x}^{B,g},\ \boldsymbol{x}^* \in \boldsymbol{x}^{B^*,g}.$$

In matrix form, $\boldsymbol{F}^B_{\mathrm{S2M}} = 2^{-(r+2)\ell} \boldsymbol{F}^{B^*}_{\mathrm{S2M}}$ and $\boldsymbol{K}^B_{\mathrm{S2M}} = 2^{-r\ell} \boldsymbol{K}^{B^*}_{\mathrm{S2M}}$. Solving for $\boldsymbol{\phi}^{B,u}$, (23) becomes

$$\boldsymbol{\phi}^{B,u} = \boldsymbol{T}^{B^*}_{\mathrm{S2M}} \gamma^B, \tag{37}$$

where $\boldsymbol{T}^{B^*}_{\mathrm{S2M}}$ is precomputed and stored. (For inhomogeneous kernels, we store one matrix per leaf level).

Neighbor list interactions for adaptive trees. For adaptive level-restricted trees, leaves may exist at any level and $U \in L^B_U$ may exist at one level finer or coarser than B. As above for the S2M operators, for homogeneous kernels with scaling exponent r, we only compute matrices for pairs (B, U) with B scaled to B^* (U is appropriately scaled as well to U^*) such that (33) and (34) become

$$F_j^{U,B}(\boldsymbol{x}) = 2^{-(r+2)\ell} K[U^*, \beta_j](\boldsymbol{x}^*) = 2^{-(r+2)\ell} F_j^{(U,B)^*}(\boldsymbol{x}^*), \quad \text{for all } \boldsymbol{x} \in \boldsymbol{x}^{B,g}, \boldsymbol{x}^* \in \boldsymbol{x}^{B^*,g},$$

$$u^{B,g} = \sum_U \boldsymbol{F}^{U,B}_{\mathrm{S2T}} \gamma^U = 2^{-(r+2)\ell} \sum_U \boldsymbol{F}^{(U,B)^*}_{\mathrm{S2T}} \gamma^U.$$

Along with 27 possible same-level neighbors, there are 56 fine-level neighbors (one level deeper) and 7 coarse-level neighbors (one level higher), all constituting the 90 possible locations for boxes in L^B_U in a level-restricted octree. Using symmetries (Section 5), we only precompute and store 10 matrices. For inhomogeneous kernels, this set of matrices is precomputed for each level for which leaf boxes exist.

M2T *and* S2L *operators.* For leaf box B and $W \in L^B_W$, we need an operator that evaluates $\phi^{W,u}$ at $\boldsymbol{x}^{B,g}$:

$$u^{B,g}(\boldsymbol{x}) = \sum_W K[\boldsymbol{y}^{W,u}, \boldsymbol{\phi}^{W,u}](\boldsymbol{x}) \text{for } \boldsymbol{x} \in \boldsymbol{x}^{B,g},$$

or, in matrix form,

$$u^{B,g} = \sum_W \boldsymbol{K}_{\mathrm{M2T}}^{W,B} \phi^{W,u}, \tag{38}$$

for precomputed $\boldsymbol{K}_{\mathrm{M2T}}^{W,B}$. For all boxes B, L_X^B contains leaves X, for which contributions to B are computed at $\boldsymbol{x}^{B,d}$:

$$u^{B,d}(\boldsymbol{x}) = \sum_X K[X, g^X](\boldsymbol{x}) \approx \sum_X \sum_{j=1}^{N_k} \gamma_j^X F_j(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \boldsymbol{x}^{B,d},$$

or, in matrix form,

$$u^{B,d} = \sum_X \boldsymbol{F}_{\mathrm{S2L}}^{X,B} \gamma^X. \tag{39}$$

There are 152 possible locations for $W \in L_W^B$; however, only six locations are distinct up to translation and rotation; also, due to the inverse relationship between L_X^B and L_W^B, the number of symmetry classes is the same (Section 5). For homogeneous kernels, only six $\boldsymbol{K}_{\mathrm{M2T}}^{W,B}$ and six $\boldsymbol{F}_{\mathrm{S2L}}^{X,B}$ matrices are precomputed for level $\ell = 0$ and scaled as necessary. For inhomogeneous kernels we compute and store these sets for each leaf level.

Remark. In cases where the order of γ is low compared to the order of $\phi^{B,u}$ and $\phi^{B,d}$, the size of M2T and S2L operators may actually be *larger* than those needed for direct computation of contributions from $W \in L_W^B$ or $X \in L_X^B$ to $\boldsymbol{x}^{B,g}$. Assuming we have a homogeneous kernel, if $W \in L_W^B$ is a leaf box, we can replace $\boldsymbol{K}_{\mathrm{M2T}}^{W,B}$ by $\boldsymbol{F}_{\mathrm{S2T}}^{W,B}$, constructed exactly in the same way as for boxes in the neighbor list L_U^B. Similarly, for leaf box B and a box $X \in L_X^B$, we can replace $\boldsymbol{F}_{\mathrm{S2L}}^{X,B}$ with $\boldsymbol{F}_{\mathrm{S2T}}^{X,B}$. For homogeneous kernels, these operators are computed for B^* only and scaled as necessary as in Section 4.4.

4.8. *Pseudocode and complexity for kernel-independent FMM volume solver.*

Pseudocode. The algorithm is summarized on the next page. We assume that a tree-level restricted octree T already exists [24] and that for each box, B, we are given the approximation, γ^B, to the force g^B (we discuss how to construct γ from g in Section 4.6). For clarity, we do not include the optimization of replacing M2T and S2L with S2T operators when more efficient as discussed above.

Computational complexity and storage requirements. We analyze the complexity for a uniformly refined octree. The analysis for the adaptive FMM is similar but slightly more complicated. We assume a homogeneous scalar kernel such as the Laplace kernel in (10) for analyzing the storage and computational complexities. Further, we assume that there are ℓ levels in the octree T. For a uniform tree, this implies we have $M_\ell = 8^\ell$ leaves and $M_t = (8^{\ell+1} - 1)/7$ total boxes in T. If we are using a k-th order polynomial approximation to the force at each leaf, we further assume there are approximately $N = M_\ell n^3$ total target points and $C = M_\ell N_k$ total

STEP 1 - BUILD LISTS
for each box B in *preorder* traversal of T **do**
  build L_U^B, L_W^B, L_X^B, and L_V^B (Section 4.7)
end for
STEP 2 - UPWARD PASS (Section 4.2)
for each box B in *postorder* traversal of T **do**
  if B is a leaf box **then**
    Convert local force approximations to upward densities:
    $\phi^{B,u} := \boldsymbol{T}_{\mathrm{S2M}}^{B} \gamma^{B}$ (23)
  else
    Translate children's upward densities to parent's upward density:
    $\phi^{B,u} := \sum_{C} \boldsymbol{T}_{\mathrm{M2M}}^{C,B} \phi^{C_i,u}$ (25)
  end if
end for
STEP 3 - DOWNWARD PASS (Section 4.3)
for each nonroot box B in *preorder* traversal of T **do**
  Add potentials due to parent downward density, U and X boxes to get the downward check potential:
  $u^{B,d} := \boldsymbol{K}_{\mathrm{L2L}}^{P,B} \phi^{P,d} + \sum_{V \in L_V^B} \boldsymbol{K}_{\mathrm{M2L}}^{V,B} \phi^{V,u} + \sum_{X \in L_X^B} \boldsymbol{F}_{\mathrm{S2L}}^{X,B} \boldsymbol{\gamma}^{X}$ (29), (27), (39)
  Translate the check potential to the downward density:
  $\phi^{B,d} := (\boldsymbol{K}_{\mathrm{L2L}}^{B,B})^{-1} u^{B,d}$ (30)
  if B is a leaf box **then**
    Compute potentials from adjacent and W boxes to the potential at grid locations:
    $u^{B,g} := \sum_{U \in L_U^B} \boldsymbol{F}_{\mathrm{S2T}}^{U,B} \gamma^{U} + \sum_{W \in L_W^B} \boldsymbol{K}_{\mathrm{M2T}}^{W,B} \phi^{W,u}$ (34), (38)
    Add the potential from the far field:
    $u^{B,g} := u^{B,g} + \boldsymbol{F}_{\mathrm{L2T}} \phi^{B,d}$ (31)
  end if
end for

Algorithm 1. Kernel-independent volume FMM.

coefficients. Let p be the number of coefficients sought in the multipole expansion, affecting the size of the equivalent densities and surfaces. For a desired level of precision in the expansion, say $\epsilon_{\mathrm{fmm}} = 10^{-n_p}$, we have $p = n_p^3 - (n_p - 2)^3$. In Table 1, we indicate the computational complexity of each step of the nonadaptive FMM algorithm as well as the amount of precomputation and storage used for operators at each step. For nonuniform source distributions, we store additional

Operator	Complexity	Storage
S2M: $\boldsymbol{T}^{B}_{\mathrm{S2M}}$	$O(Cp)$	pN_k
M2M: $\boldsymbol{T}^{C,B}_{\mathrm{M2M}}$	$O((M_t - M_\ell)p^2)$	p^2
M2L: $\boldsymbol{K}^{V,B}_{\mathrm{M2L}}$	$O(M_t p^{3/2}\log p + 189 M_t p^{3/2})$	$16p^{3/2}$
L2L: $\boldsymbol{K}^{P,B}_{\mathrm{L2L}}$, $(\boldsymbol{K}^{B,B}_{\mathrm{L2L}})^{-1}$	$O(M_t p^2)$	$2p^2$
L2T: $\boldsymbol{K}^{B,B}_{\mathrm{L2T}}$	$O(Np)$	pn^3
Near Interaction: $\boldsymbol{F}^{U,B}_{\mathrm{S2T}}$	$O(27NN_k)$	$4N_k n^3$
U-list (adaptive): $\boldsymbol{F}^{U,B}_{\mathrm{S2T}}$		$10N_k n^3$
W-list: $\boldsymbol{F}^{W,B}_{\mathrm{S2T}}$, $\boldsymbol{K}^{W,B}_{\mathrm{M2T}}$		$6n^3(N_k + p)$
X-list: $\boldsymbol{F}^{X,B}_{\mathrm{S2T}}$, $\boldsymbol{F}^{X,B}_{\mathrm{S2L}}$		$6N_k(n^3 + p)$

Table 1. Computational complexity and storage requirements for a scalar homogeneous kernel. These values scale linearly for matrix and inhomogeneous kernels.

operators for the near-field interactions in the U, W, and X operators; the complexity of these operators are based on the degree of adaptivity.

Finally, we note that the computational and storage complexities will scale linearly for matrix or inhomogeneous kernels. For example, for the Stokes kernel in (12), the number of coefficients, p, scales as a results of the matrix kernel size to $p = 9(n_p^3 - (n_p - 2)^3)$. For the modified Helmholtz kernel in (11), the inhomogeneous nature of the kernel results in an increased storage complexity, which varies depending on the number of different levels in the tree.

5. Symmetries for precomputed interaction operators

For a box B and all boxes in L^B_U, L^B_V, L^B_W and L^B_X, the number of different relative positions can be large, so precomputing all possible interaction matrices may require significant time and storage. Performance can also be affected by the need for random access of large amounts of precomputed data. The number of precomputed matrices can be substantially reduced via symmetries; that is, many box positions are equivalent in the sense that there is a rigid transformation $\mathcal{T}$, mapping box Z_1 to Z_2 and box B to itself. We store a single matrix for a representative box for each symmetry class, obtaining matrices for all elements of the class by applying $\mathcal{T}$ to the matrix for the representative box.

For every list type $Z \in \{U, V, W, X\}$, we define a set of possible positions $\mathrm{Pos}(Z)$ and a set of symmetry classes which form a partition of $\mathrm{Pos}(Z)$. For each class, we

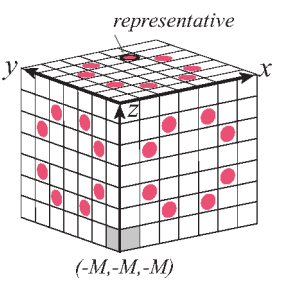
define a *reference box*, and for each box position in $\mathrm{Pos}(Z)$, we need an efficient way to determine its class and a transformation $\mathcal{T}(B) : \mathbb{R}^3 \to \mathbb{R}^3$ mapping it to the reference box.

For all lists, the symmetries are related to the transformations of space which map a grid of cubes to itself. We consider N^3 grids of sizes 1^3 to 7^3 (we discuss which lists correspond to which cubes in more detail below) and begin by classifying all symmetries of such grids.

Grid symmetries. The cubes on the N^3 grid are indexed by (i, j, k) with values $-M \ldots -1, 0, 1 \ldots M$ for odd $N = 2M + 1$ and $-M \ldots -1, 1 \ldots M$ for even $N = 2M$. We skip index 0 for even grids to ensure cube centers and indices are transformed by symmetries in the same way. If the cube size is 1, cube centers are exactly the indices (i, j, k) for odd N and differ by $\pm\frac{1}{2}$ for even N, depending on the index sign. Each N^3 grid can be partitioned into M (for even N) or $M + 1$ (for odd N) layers. Layer 0 consists of one cube and exists for odd N, and layer M consists of cubes on the surface of the N^3 grid. For odd N, layer l has size $(2l + 1)^3$ and for even N, layer l has size $(2l)^3$.

The group of symmetries G_{cube} of a cube has order 48. For a cube centered at zero, transformations in G_{cube} are compositions of rotations and reflections, mapping each axis direction to another, possibly with orientation reversed. Any permutation of directions is possible, so we identify the group with $S_3 \times J^3$, where S_3 is the group of permutations of length 3, and J is the two-element group of reflections. The rotational part of any element of G_{cube} can be specified as a permutation of length 3 on the set of axes $\{x, y, z\}$, with an orientation 1 or -1 specified for each axis. Transformations from G_{cube} encoded in this way can be applied to points very efficiently: for a point $\boldsymbol{x} \in \mathbb{R}^3$, the permutation is applied to its coordinates, which are then scaled by 1 or -1.

For the N^3 grid, the equivalence classes under the action of G_{cube} can be enumerated combinatorially. If two indices (i, j, k) and (i', j', k') differ only by signs of components, corresponding cubes are in the same class, mapped by reflections. To enumerate all classes, we consider cubes with nonnegative indices. Two cubes with nonnegative indices (i, j, k) and (i', j', k') are in the same class if and only if there is a permutation mapping (i, j, k) to (i', j', k'). For $i \neq j \neq k$ and $i, j, k \in [1, M]$, seven series of equivalence classes are easily enumerated, corresponding to signatures (i, j, k), (i, i, j), (i, i, i), $(0, i, i)$, $(0, 0, i)$ and $(0, 0, 0)$. A reference box in every class is uniquely defined by requiring that its three indices are all nonnegative and are in nondecreasing order. As an example, the figure shows the representative box for the $(1, 2, 3)$ class in the 7^3 grid for $M = 3$.



The properties of classes in each series are summarized in Table 2. For a box Z, with grid index (i, j, k) relative to B, the reference box is obtained by taking absolute values and sorting the indices; sign changes and a permutation mapping (i, j, k) to the reference box index also encode the transformation.

series signature	7^3 classes	reference cube	classes per grid	classes per layer	class size
(i, j, k)		$(\|i\|, \|j\|, \|k\|)$, $\|i\| < \|j\| < \|k\|$	$\binom{M}{3}$	$\binom{M-1}{2}$	48
(i, i, j)		$(\|i\|, \|i\|, \|j\|)$, $\|i\| < \|j\|$ or $(\|i\|, \|j\|, \|j\|)$	$M(M-1)$	$2(M-1)$	24
(i, i, i)		$(\|i\|, \|i\|, \|i\|)$	M	1	8
$(0, i, j)$		$(0, \|i\|, \|j\|)$	$\binom{M}{2}$	$M-1$	24
$(0, i, i)$		$(0, \|i\|, \|i\|)$	M	1	12
$(0, i, i)$		$(0, 0, \|i\|)$	M	1	6
$(0, 0, 0)$	—	$(0, 0, 0)$	1	—	1

Table 2. Series of equivalence classes of cubes in an N^3 grid, with $M = \lfloor N/2 \rfloor$. For even N only the first 3 series of classes may be nonempty. For odd N all classes are present. For $M \leq 2$, (i, j, k) classes are empty, and for $M = 1$, (i, i, j) and $(0, i, j)$ classes are also empty. Class (0, 0, 0), corresponding to the center of the grid, exists only in layer 0. Boxes in different classes in one series are marked with circles of different colors; representative boxes are marked with circles with black border. The view is from the top, with first index direction to the right, second direction up and third towards the viewer, as in the figure on page 99. The total number of classes for $(2M)^3$ layers is $(M+1)M/2$ (classes (i, j, M) with $i, j = 1 \ldots M$, $i \leq j$), and for $(2M+1)^3$ layers, it is $(M+2)(M+1)/2$ (classes (i, j, M) with $i, j = 0 \ldots M$, $i \leq j$).

Symmetries of L_U^B. Due to the tree-level restriction, boxes $U \in L_U^B$ are either neighbors of B, neighbors of B's parent and adjacent to B, or adjacent children of neighbors of B. We denote these three sublists of L_U^B by $L_{U,n}^B$, $L_{U,p}^B$ and $L_{U,c}^B$ respectively. Note that $A \in L_{U,p}^B$ is equivalent to $B \in L_{U,c}^A$; hence, it is sufficient to consider $L_{U,n}^B$ and $L_{U,c}^B$. The neighbors of B on the same level as B form a 3^3 grid centered at B, so from Table 2, the number of classes is 4: $(1, 1, 1)$, $(0, 1, 1)$, $(0, 0, 1)$, and $(0, 0, 0)$. Locations of $U \in L_{U,c}^B$ can be thought of as the outer layer of a 4^3 grid, with $M = 2$ and B as the 2^3 subgrid in the center, so we obtain 3 classes: $(1, 1, 2)$, $(1, 2, 2)$, $(2, 2, 2)$, giving 10 classes for L_U^B.

Symmetries of L_V^B. Boxes in L_V^B are children of neighbors of the parent of B, so they can all be represented by cubes of a 6^3 grid; however, the group of rigid transformations of the grid mapping to itself do not necessarily preserve B. Hence, instead regard L_V^B as a subset of a 7^3 grid centered at B with $M = 3$. All $V \in L_V^B$ are in layers 2 and 3, and there are 10 classes: for layer 3, classes $(i, j, 3)$, $i, j = 1 \ldots 3$, $i \leq j$ and for layer 2, classes $(i, j, 2)$, $i, j = 0, 1, 2$, $i \leq j$. Because we consider only a subset of the full 7^3 grid, the class sizes are smaller, but it can easily be seen that no class becomes empty, so the number is optimal.

Symmetries of L_W^B, L_X^B. For a level-restricted tree, boxes $W \in L_W^B$ are children of neighbors of B not adjacent to B, that is, they reside in the surface layer of a 6^3 grid with B as the central 2^3 grid. For $M = 3$, we have 6 classes from Table 2: $(i, j, 3)$, $i, j = 1 \ldots 3$, $i \leq j$. Due to duality, the number of classes for L_X^B is the same, but the class sizes may *not* be the same.

Summary. For a given pair (B, Z), if $Z \in \{L_{U,n}^B, L_{U,c}^B, L_V^B, L_W^B\}$, determine the translation and scaling which map B to the central box or 2^3 subgrid of a larger grid. Then, apply the same transformation to the center of Z; resulting coordinates yield the index (i, j, k), which is translated into the reference box and rotation as described above.

6. Numerical results

Our algorithm has been implemented in C++, and we have tested several kernels and source and target point distributions. Our tests were run on an Intel Xeon-based X7560 (2.27 GHz, 64 bit) system with 16 CPUs and 128 GB of RAM; the major computation loops are accelerated with OpenMP [19] as discussed in Section A.4.

We first test the free-space Poisson solver on three different types of problems designed to show how our algorithm handles increasing levels of complexity in the force distribution.

We use an adaptive-refinement strategy similar to [24]. For this, we compute a k-th order polynomial approximation, γ^B, to the force $g^B(\boldsymbol{x})$ sampled on a

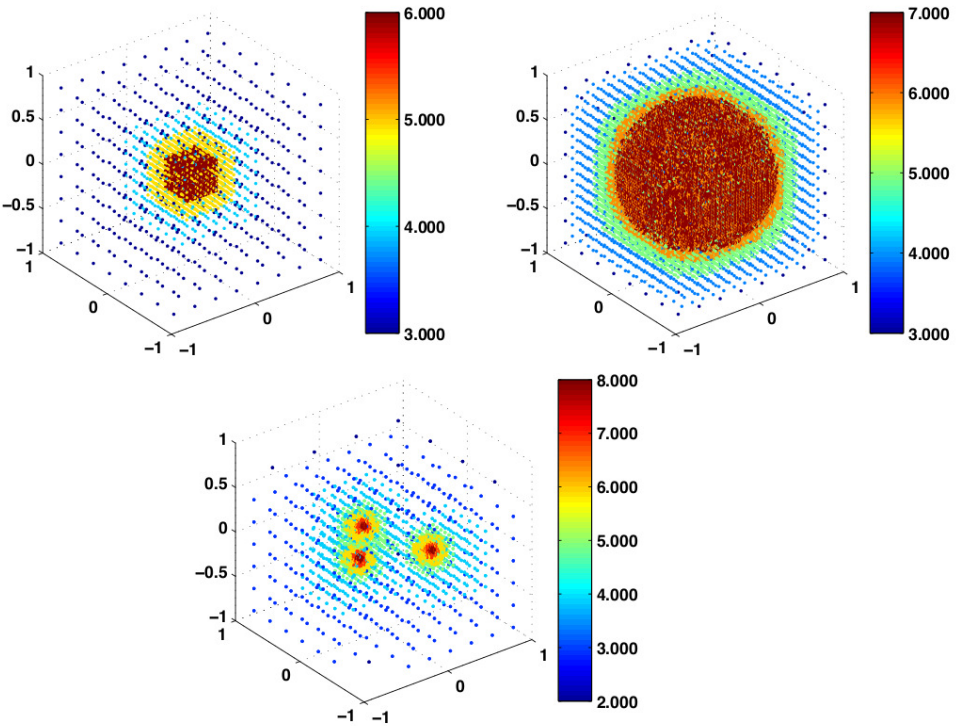


Figure 6. Sample force distributions based on adaptive refinement. Each point, colored by its tree level, indicates the center of a leaf box B. Top left: A single sharply peaked Gaussian function (Example 1). Top right: A discontinuous force distribution, equal to 1 inside a sphere and 0 outside (Example 2). Bottom: A discontinuous force distribution involving oscillatory functions restricted to the interiors of a set of three spheres (Example 3).

$k \times k \times k$ grid. We let $\tilde{g}^B$ be the force evaluated on a refined $2k \times 2k \times 2k$ grid. If $\|g^B(\boldsymbol{x}) - \tilde{g}^B(\boldsymbol{x})\|_2 > \epsilon_{\text{rhs}}$, B is subdivided, and the octree is balanced as needed. Three force distributions, used in Examples 1–3 below, are shown in Figure 6.

Example 1. The first experiment tests the accuracy of our method for solving the Poisson equation — Equation (7) with kernel (10) — with a fast-decaying smooth right-hand side:

$$-\Delta u(\boldsymbol{x}) = \sum_{i=0}^{8} -e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}(4L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2 - 6L), \quad L = 250,$$

with solution

$$u(\boldsymbol{x}) = \sum_{i=0}^{8} -e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2},$$

where $\boldsymbol{x}_i = \left(\pm\frac{3}{40}, \pm\frac{3}{40}, \pm\frac{3}{40}\right)$ inside the $[-1, 1]^3$ box. This test requires a high degree of adaptivity to achieve good accuracy with a limited number of points.

In Table 3, ϵ_{fmm} is the precision of the translation operators, ϵ_{rhs} is the refinement criterion for the adaptive refinement of the source distribution, and M_ℓ is the number of leaves in the tree T with L_T levels. The number of points N_{pts} is computed as $M_\ell k^3$ where k is the order of the polynomial. This number of points per leaf is chosen to be sufficiently large to build the polynomial approximation of order k. The computation time T_{FMM} is given in seconds, and the *rate* is in points per second. E_2 and E_∞ are the relative L^2 and L^∞ errors, respectively. Timings include FMM evaluation times only; when the precision ϵ_{fmm} remains constant, the rate of work

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	rate
fourth-order force approximation								
10^{-2}	10^{-2}	736	47104	6	$2.3{\cdot}10^{-2}$	$2.4{\cdot}10^{-2}$	$9.1453{\cdot}10^{-3}$	$5.15{\cdot}10^{+6}$
10^{-4}	10^{-2}	736	47104	6	$1.1{\cdot}10^{-3}$	$6.8{\cdot}10^{-4}$	$2.4328{\cdot}10^{-2}$	$1.94{\cdot}10^{+6}$
10^{-4}	10^{-4}	3088	197632	7	$1.1{\cdot}10^{-4}$	$2.7{\cdot}10^{-4}$	$9.6590{\cdot}10^{-2}$	$2.05{\cdot}10^{+6}$
10^{-6}	10^{-4}	3088	197632	7	$3.8{\cdot}10^{-5}$	$2.5{\cdot}10^{-5}$	$3.7906{\cdot}10^{-1}$	$5.21{\cdot}10^{+5}$
10^{-6}	10^{-6}	19328	1236992	8	$3.8{\cdot}10^{-6}$	$3.6{\cdot}10^{-6}$	$2.3889{\cdot}10^{+0}$	$5.18{\cdot}10^{+5}$
10^{-8}	10^{-6}	19328	1236992	8	$3.7{\cdot}10^{-6}$	$1.3{\cdot}10^{-6}$	$6.2057{\cdot}10^{+0}$	$1.99{\cdot}10^{+5}$
10^{-8}	10^{-8}	143088	9157632	9	$1.6{\cdot}10^{-7}$	$8.8{\cdot}10^{-8}$	$4.5280{\cdot}10^{+1}$	$2.02{\cdot}10^{+5}$
sixth-order force approximation								
10^{-4}	10^{-4}	1408	304128	6	$1.1{\cdot}10^{-4}$	$2.3{\cdot}10^{-4}$	$1.1016{\cdot}10^{-1}$	$2.76{\cdot}10^{+6}$
10^{-6}	10^{-4}	1408	304128	6	$1.4{\cdot}10^{-5}$	$3.5{\cdot}10^{-5}$	$1.8717{\cdot}10^{-1}$	$1.62{\cdot}10^{+6}$
10^{-6}	10^{-6}	4936	1066176	7	$9.0{\cdot}10^{-7}$	$2.2{\cdot}10^{-6}$	$6.6737{\cdot}10^{-1}$	$1.60{\cdot}10^{+6}$
10^{-8}	10^{-6}	4936	1066176	7	$3.3{\cdot}10^{-7}$	$1.6{\cdot}10^{-7}$	$1.6155{\cdot}10^{+0}$	$6.60{\cdot}10^{+5}$
10^{-8}	10^{-8}	20112	4344192	8	$2.4{\cdot}10^{-8}$	$6.2{\cdot}10^{-8}$	$6.8652{\cdot}10^{+0}$	$6.33{\cdot}10^{+5}$
10^{-10}	10^{-8}	20112	4344192	8	$1.8{\cdot}10^{-8}$	$1.0{\cdot}10^{-8}$	$1.5771{\cdot}10^{+1}$	$2.76{\cdot}10^{+5}$
10^{-10}	10^{-10}	92072	19887552	9	$6.3{\cdot}10^{-9}$	$9.7{\cdot}10^{-9}$	$7.5103{\cdot}10^{+1}$	$2.65{\cdot}10^{+5}$
eighth-order force approximation								
10^{-6}	10^{-6}	2024	1036288	7	$9.2{\cdot}10^{-7}$	$3.5{\cdot}10^{-6}$	$4.6688{\cdot}10^{-1}$	$2.22{\cdot}10^{+6}$
10^{-8}	10^{-6}	2024	1036288	7	$3.7{\cdot}10^{-7}$	$8.2{\cdot}10^{-7}$	$7.4189{\cdot}10^{-1}$	$1.40{\cdot}10^{+6}$
10^{-8}	10^{-8}	5440	2785280	7	$1.9{\cdot}10^{-8}$	$6.6{\cdot}10^{-8}$	$2.1128{\cdot}10^{+0}$	$1.32{\cdot}10^{+6}$
10^{-10}	10^{-8}	5440	2785280	7	$7.7{\cdot}10^{-9}$	$7.6{\cdot}10^{-9}$	$4.3588{\cdot}10^{+0}$	$6.40{\cdot}10^{+5}$
10^{-10}	10^{-10}	22800	11673600	8	$4.1{\cdot}10^{-9}$	$5.6{\cdot}10^{-9}$	$1.9449{\cdot}10^{+1}$	$6.00{\cdot}10^{+5}$
10^{-12}	10^{-10}	22800	11673600	8	$2.6{\cdot}10^{-9}$	$4.7{\cdot}10^{-9}$	$4.1602{\cdot}10^{+1}$	$2.81{\cdot}10^{+5}$
10^{-12}	10^{-12}	50352	25780224	9	$2.1{\cdot}10^{-9}$	$4.6{\cdot}10^{-9}$	$9.2139{\cdot}10^{+1}$	$2.80{\cdot}10^{+5}$

Table 3. Results for free-space Poisson equation (Example 1): Gaussian bump at the origin.

per source and target point remains close to constant, as we would expect since the FMM algorithm scales linearly.

Example 2. In this example, we consider a discontinuous right-hand side, with $g(\boldsymbol{x}) = 1$ inside a sphere of radius $R = 0.75$, and $g(\boldsymbol{x}) = 0$ outside the sphere. Letting $r = \|\boldsymbol{x}\|$, the problem becomes

$$-\Delta u(\boldsymbol{x}) = \begin{cases} 1 & \text{if } r \le R, \\ 0 & \text{else,} \end{cases}$$

with solution

$$-\Delta u(\boldsymbol{x}) = \begin{cases} (R^2 - r^2)/6 + R^2/3 & \text{if } r \le R, \\ R^3/3r^2 & \text{else.} \end{cases}$$

While this problem can be handled analytically, it serves as a useful test of performance on adaptive data structures that are refined in the neighborhood of a surface. The number of points indicates the total number of points both inside and outside the sphere. Since the coefficient representation of the force for a leaf node entirely outside of the sphere is zero, these boxes are ignored in all evaluation phases; this increases the computed rate somewhat. A greater speedup is achieved from the observation that leaf nodes entirely in the interior have a constant source distribution, so that only one polynomial coefficient is nonzero. This significantly accelerates both the near-field and S2M calculation stages. Results are shown in Table 4.

ϵ_{fmm}	ϵ_{rhs}	k	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	rate
fourth-order force approximation									
10^{-2}	10^{-2}	4	232	14848	4	$1.1{\cdot}10^{-2}$	$1.6{\cdot}10^{-2}$	$1.8346{\cdot}10^{-3}$	$8.09{\cdot}10^{+6}$
10^{-3}	10^{-3}	4	1184	75776	5	$4.3{\cdot}10^{-3}$	$4.6{\cdot}10^{-3}$	$1.1007{\cdot}10^{-2}$	$6.88{\cdot}10^{+6}$
10^{-4}	10^{-4}	4	5888	376832	6	$1.4{\cdot}10^{-4}$	$2.5{\cdot}10^{-4}$	$1.1327{\cdot}10^{-1}$	$3.33{\cdot}10^{+6}$
10^{-5}	10^{-5}	6	11432	2469312	7	$1.6{\cdot}10^{-5}$	$3.3{\cdot}10^{-5}$	$6.2147{\cdot}10^{-1}$	$3.97{\cdot}10^{+6}$
10^{-6}	10^{-6}	6	80088	17299008	8	$7.2{\cdot}10^{-6}$	$1.8{\cdot}10^{-5}$	$6.0647{\cdot}10^{+0}$	$2.85{\cdot}10^{+6}$
10^{-7}	10^{-7}	8	127856	65462272	8	$9.4{\cdot}10^{-7}$	$3.3{\cdot}10^{-6}$	$1.8852{\cdot}10^{+1}$	$3.47{\cdot}10^{+6}$
10^{-8}	10^{-8}	8	528984	270839808	10	$3.7{\cdot}10^{-7}$	$9.8{\cdot}10^{-7}$	$1.2335{\cdot}10^{+2}$	$2.20{\cdot}10^{+6}$
10^{-9}	10^{-9}	8	2074360	1062072320	10	$3.2{\cdot}10^{-8}$	$2.6{\cdot}10^{-7}$	$6.4900{\cdot}10^{+2}$	$1.64{\cdot}10^{+6}$

Table 4. Free-space Poisson equation (Example 2): discontinuous force.

Example 3. For our third example, we replicate an experiment from [47] for a highly oscillatory force with discontinuities along multiple surfaces, setting

$$f_m(r) = \begin{cases} ((r - r^2)\sin(2m\pi r))^2 & \text{if } r < 1, \\ 0 & \text{if } r \ge 1, \end{cases}$$

$$-\Delta u(\boldsymbol{x}) = \frac{1}{R^3} \sum_{i=0}^{2} f_m(|\boldsymbol{x} - \boldsymbol{c}_i|/R), \tag{40}$$

where $\boldsymbol{c}_0 = \left(\frac{3}{16}, \frac{7}{16}, \frac{13}{16}\right)$, $\boldsymbol{c}_1 = \left(\frac{7}{16}, \frac{13}{16}, \frac{3}{16}\right)$, $\boldsymbol{c}_2 = \left(\frac{13}{16}, \frac{3}{16}, \frac{7}{16}\right)$, $R = 0.05$, and the wavelength of f_m is $\lambda_m = R/(2m) = (1/40m)$. Defining

$$\begin{aligned}\phi_m(r) = {} & \frac{(10r^6 - 28r^5 + 21r^4 - 7)}{840} + \frac{(60r - 120)}{r\lambda_m^6} - \frac{9}{\lambda_m^4} \\ & + \left[\frac{(300r - 120)}{r\lambda_m^6} - + \frac{(30r^2 - 36r + 9)}{\lambda_m^4} + \frac{(r^4 - 2r^3 + r^2)}{2\lambda_m^2}\right]\cos(r\lambda_m) \\ & + \left[-\frac{360}{r\lambda_m^7} + \frac{(120r^2 - 96r + 12)}{r\lambda_m^5} + \frac{(5r^3 + 8r^2 - 3r)}{\lambda_m^3}\right]\sin(r\lambda_m),\end{aligned}$$

and

$$\theta_m(r) = \left(\frac{360}{\lambda_m^6} - \frac{12}{\lambda_m^4} - \frac{1}{120}\right)/r,$$

we write the solution to (40) as

$$u^{\text{exact}}(\boldsymbol{x}) = \begin{cases} \phi(\|\boldsymbol{x} - \boldsymbol{c}_0\|/R) + \sum_{i=1,2}\theta(\|\boldsymbol{x} - \boldsymbol{c}_i\|/R) & \text{if } \|\boldsymbol{x} - \boldsymbol{c}_0\| < R, \\ \phi(\|\boldsymbol{x} - \boldsymbol{c}_1\|/R) + \sum_{i=0,2}\theta(\|\boldsymbol{x} - \boldsymbol{c}_i\|/R) & \text{if } \|\boldsymbol{x} - \boldsymbol{c}_1\| < R, \\ \phi(\|\boldsymbol{x} - \boldsymbol{c}_2\|/R) + \sum_{i=0,1}\theta(\|\boldsymbol{x} - \boldsymbol{c}_i\|/R) & \text{if } \|\boldsymbol{x} - \boldsymbol{c}_2\| < R, \\ \sum_{i=0}^{2}\theta(\|\boldsymbol{x} - \boldsymbol{c}_i\|/R) & \text{else.} \end{cases}$$

In order to compare our results to [47], we use the error metric introduced there. Let ϵ^B be the vector of errors calculated as the difference between the calculated and exact solutions on B and calculate the following norm over all leaf boxes.

$$\|\epsilon_{\text{all}}^B\|_2 = \sum_B \left(\int \frac{\epsilon^B}{\|u^{\text{exact}}\|_\infty}\right)^{1/2}.$$

As indicated in [47],

$$\|u^{\text{exact}}\|_\infty = \left|\left(-\frac{6}{\lambda_m^4} - \frac{1}{120}\right)/R + \left(\frac{720}{\lambda_m^6} - \frac{24}{\lambda_m^4} - \frac{1}{120}\right)/\|\boldsymbol{c}_i - \boldsymbol{c}_j\|\right| \quad \text{for all } i, j = 0, 1, 2, i \neq j.$$

Our automatic refinement strategy refines within or near the sphere surfaces, with refinement taking place in the exterior of the spheres only for the purpose of tree-balancing. We build coefficients only on leaf boxes which contain nonzero source distributions: either interior to or intersecting one of the three spheres. Results are shown in Table 5.

While the performance of our code cannot be compared easily to the optimized and parallelized scheme presented in [47], we have implemented schemes accurate to much higher order. Thus, as expected, we are able to reach comparable accuracies with significantly fewer points. To compare the number of points required, we consider the number of points in the finest level solve of their three-level examples. For

m	ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	k	$\|\epsilon^B_{\text{all}}\|_2$	$\|\epsilon^B_{\text{all}}\|_\infty$
1	10^{-6}	10^{-6}	3984	254976	8	4	$2.3{\cdot}10^{-7}$	$2.2{\cdot}10^{-5}$
1	10^{-8}	10^{-8}	7296	1575936	9	6	$1.1{\cdot}10^{-8}$	$7.1{\cdot}10^{-7}$
1	10^{-10}	10^{-10}	24144	12361728	9	8	$1.6{\cdot}10^{-10}$	$1.8{\cdot}10^{-8}$
7	10^{-6}	10^{-6}	93816	6004224	10	4	$2.2{\cdot}10^{-6}$	$1.8{\cdot}10^{-4}$
7	10^{-8}	10^{-8}	195984	42332544	10	6	$9.3{\cdot}10^{-9}$	$1.1{\cdot}10^{-6}$
7	10^{-10}	10^{-10}	228312	116895744	10	8	$1.1{\cdot}10^{-10}$	$4.3{\cdot}10^{-8}$
15	10^{-6}	10^{-6}	140568	8996352	10	4	$6.4{\cdot}10^{-7}$	$8.6{\cdot}10^{-5}$
15	10^{-8}	10^{-8}	1092456	235970496	11	6	$7.2{\cdot}10^{-8}$	$4.2{\cdot}10^{-6}$
15	10^{-10}	10^{-10}	1596672	817496064	11	8	$4.8{\cdot}10^{-10}$	$6.1{\cdot}10^{-8}$
30	10^{-6}	10^{-6}	148272	9489408	10	4	$5.8{\cdot}10^{-7}$	$6.2{\cdot}10^{-5}$
30	10^{-8}	10^{-8}	1491216	322102656	11	6	$2.1{\cdot}10^{-8}$	$3.8{\cdot}10^{-6}$
30	10^{-10}	10^{-10}	1720152	880717824	12	8	$4.9{\cdot}10^{-9}$	$2.0{\cdot}10^{-6}$
60	10^{-6}	10^{-6}	150288	9618432	11	4	$6.0{\cdot}10^{-7}$	$9.6{\cdot}10^{-5}$
60	10^{-8}	10^{-8}	1502592	324559872	11	6	$9.2{\cdot}10^{-8}$	$1.4{\cdot}10^{-5}$
60	10^{-10}	10^{-9}	1659312	849567744	11	8	$7.7{\cdot}10^{-8}$	$1.7{\cdot}10^{-5}$

Table 5. Free-space Poisson equation (Example 3): Discontinuities along several spherical surfaces containing oscillating source distributions.

$m = 7$, we achieve accuracy on par with their most accurate tests with approximately $\frac{1}{100}$ as many points. For $m = 15$, we require approximately $\frac{1}{5}$ as many points, and with $\frac{1}{4}$ as many points, we achieve about two orders of magnitude greater accuracy. For $m = 30$, we achieve equivalent results with approximately $\frac{1}{4}$ as many points. Additionally, we extended the examples for an even higher wavenumber component ($m = 60$), decreasing the wavelength to $4.17 \cdot 10^{-4}$, and achieving good results with fewer than 10^9 points.

Example 4. For the modified Helmholtz equation — (8) with kernel (11) — we use a right-hand side similar to that of Example 1, setting the Helmholtz parameter (inverse Debye length) to $\alpha = \pi$:

$$\alpha\, u(\boldsymbol{x}) - \Delta u(\boldsymbol{x}) = \sum_{i=0}^{8} -e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}(4L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2 - 6L - \alpha), \qquad L = 250,$$

with solution

$$u(\boldsymbol{x}) = \sum_{i=0}^{8} -e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}, \qquad \text{for } \boldsymbol{x}_i = \left(\pm\tfrac{3}{40}, \pm\tfrac{3}{40}, \pm\tfrac{3}{40}\right)$$

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞
fourth-order force approximation						
10^{-2}	10^{-2}	736	47104	6	$2.3{\cdot}10^{-2}$	$2.4{\cdot}10^{-2}$
10^{-4}	10^{-2}	736	47104	6	$6.0{\cdot}10^{-4}$	$5.6{\cdot}10^{-4}$
10^{-4}	10^{-4}	3088	197632	7	$1.1{\cdot}10^{-4}$	$1.8{\cdot}10^{-4}$
10^{-6}	10^{-4}	3088	197632	7	$2.4{\cdot}10^{-5}$	$2.1{\cdot}10^{-5}$
10^{-6}	10^{-6}	19328	1236992	8	$2.3{\cdot}10^{-6}$	$3.4{\cdot}10^{-6}$
sixth-order force approximation						
10^{-4}	10^{-4}	1408	304128	6	$1.1{\cdot}10^{-4}$	$2.4{\cdot}10^{-4}$
10^{-6}	10^{-4}	1408	304128	6	$1.4{\cdot}10^{-5}$	$3.5{\cdot}10^{-5}$
10^{-6}	10^{-6}	4936	1066176	7	$8.5{\cdot}10^{-7}$	$2.5{\cdot}10^{-6}$
10^{-8}	10^{-6}	4936	1066176	7	$1.4{\cdot}10^{-7}$	$1.3{\cdot}10^{-7}$
10^{-8}	10^{-8}	20112	4344192	8	$1.5{\cdot}10^{-8}$	$7.5{\cdot}10^{-8}$
10^{-10}	10^{-8}	20112	4344192	8	$8.9{\cdot}10^{-9}$	$7.4{\cdot}10^{-9}$
10^{-10}	10^{-10}	92072	19887552	9	$2.2{\cdot}10^{-9}$	$5.7{\cdot}10^{-9}$
eighth-order force approximation						
10^{-6}	10^{-6}	2024	1036288	7	$1.9{\cdot}10^{-6}$	$4.7{\cdot}10^{-6}$
10^{-8}	10^{-6}	2024	1036288	7	$1.8{\cdot}10^{-7}$	$4.6{\cdot}10^{-7}$
10^{-8}	10^{-8}	5440	2785280	7	$1.2{\cdot}10^{-8}$	$1.4{\cdot}10^{-8}$
10^{-10}	10^{-8}	5440	2785280	7	$7.7{\cdot}10^{-9}$	$7.6{\cdot}10^{-9}$
10^{-10}	10^{-10}	22800	11673600	8	$3.4{\cdot}10^{-9}$	$5.6{\cdot}10^{-9}$
10^{-12}	10^{-10}	22800	11673600	8	$1.3{\cdot}10^{-9}$	$2.0{\cdot}10^{-9}$
10^{-12}	10^{-12}	50352	25780224	9	$2.0{\cdot}10^{-9}$	$2.6{\cdot}10^{-9}$

Table 6. Free-space modified Helmholtz equation (Example 4): Gaussian bump at the origin.

inside of the $[-1, 1]^3$ box. All translation matrices are computed to a precision of $\epsilon_{\text{fmm}}/10$. These matrices can be computed at run-time in a lazy manner; if α is known before run-time, these tables can be precomputed, stored, and loaded as necessary. Additionally, since the right-hand side is the same as in Example 1, we use the same point distribution; hence, the timings are essentially the same as in that example and are omitted here. Results are shown in Table 6.

Example 5. We test the ability of our code to handle matrix kernels by solving the Stokes equations — (9) with kernel (12) — with the following divergence-free fast-decaying force.

$$-\mu\Delta u(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \sum_{i=0}^{8} (8L^3\|\boldsymbol{x}-\boldsymbol{x}_i\|^2 - 20L^2)e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}\big(\nabla\times(\boldsymbol{x}-\boldsymbol{x}_i)\big),$$

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	rate
fourth-order force approximation								
10^{-2}	10^{-2}	2038	130432	6	$1.3{\cdot}10^{-1}$	$1.5{\cdot}10^{-1}$	$1.3485{\cdot}10^{-1}$	$9.67{\cdot}10^{+5}$
10^{-4}	10^{-2}	2038	130432	6	$1.0{\cdot}10^{-3}$	$8.4{\cdot}10^{-4}$	$9.4899{\cdot}10^{-1}$	$1.37{\cdot}10^{+5}$
10^{-4}	10^{-4}	10606	678784	7	$9.1{\cdot}10^{-4}$	$9.4{\cdot}10^{-4}$	$5.1145{\cdot}10^{+0}$	$1.33{\cdot}10^{+5}$
10^{-6}	10^{-4}	10606	678784	7	$8.4{\cdot}10^{-6}$	$1.1{\cdot}10^{-5}$	$2.2359{\cdot}10^{+1}$	$3.04{\cdot}10^{+4}$
10^{-6}	10^{-6}	69140	4424960	8	$7.5{\cdot}10^{-6}$	$1.4{\cdot}10^{-5}$	$1.4655{\cdot}10^{+2}$	$3.02{\cdot}10^{+4}$
10^{-8}	10^{-6}	69140	4424960	8	$2.2{\cdot}10^{-7}$	$4.4{\cdot}10^{-7}$	$4.8311{\cdot}10^{+2}$	$9.16{\cdot}10^{+3}$
10^{-8}	10^{-8}	484408	31002112	9	$1.4{\cdot}10^{-7}$	$4.4{\cdot}10^{-7}$	$3.2253{\cdot}10^{+3}$	$9.61{\cdot}10^{+3}$
sixth-order force approximation								
10^{-4}	10^{-4}	2696	582336	7	$4.9{\cdot}10^{-4}$	$8.6{\cdot}10^{-4}$	$1.5720{\cdot}10^{+0}$	$3.70{\cdot}10^{+5}$
10^{-6}	10^{-4}	2696	582336	7	$4.3{\cdot}10^{-6}$	$9.9{\cdot}10^{-6}$	$5.9948{\cdot}10^{+0}$	$9.71{\cdot}10^{+4}$
10^{-6}	10^{-6}	10396	2245536	7	$8.1{\cdot}10^{-6}$	$1.3{\cdot}10^{-6}$	$2.3602{\cdot}10^{+1}$	$9.51{\cdot}10^{+4}$
10^{-8}	10^{-6}	10396	2245536	7	$1.6{\cdot}10^{-7}$	$4.1{\cdot}10^{-7}$	$7.7862{\cdot}10^{+1}$	$2.88{\cdot}10^{+4}$
10^{-8}	10^{-8}	59830	12923280	8	$1.4{\cdot}10^{-7}$	$4.3{\cdot}10^{-7}$	$4.1067{\cdot}10^{+2}$	$3.15{\cdot}10^{+4}$
10^{-10}	10^{-8}	59830	12923280	8	$5.4{\cdot}10^{-9}$	$1.3{\cdot}10^{-8}$	$1.0326{\cdot}10^{+3}$	$1.25{\cdot}10^{+4}$
10^{-10}	10^{-10}	295100	63741600	9	$5.2{\cdot}10^{-9}$	$1.1{\cdot}10^{-8}$	$5.3102{\cdot}10^{+3}$	$1.20{\cdot}10^{+4}$
eighth-order force approximation								
10^{-6}	10^{-6}	4894	2505728	7	$4.8{\cdot}10^{-6}$	$1.5{\cdot}10^{-5}$	$1.4287{\cdot}10^{+1}$	$1.75{\cdot}10^{+5}$
10^{-8}	10^{-6}	4894	2505728	7	$8.0{\cdot}10^{-8}$	$4.3{\cdot}10^{-7}$	$4.1362{\cdot}10^{+1}$	$6.06{\cdot}10^{+4}$
10^{-8}	10^{-8}	12860	6584320	7	$1.5{\cdot}10^{-7}$	$4.5{\cdot}10^{-7}$	$1.0268{\cdot}10^{+2}$	$6.41{\cdot}10^{+4}$
10^{-10}	10^{-8}	12860	6584320	7	$6.0{\cdot}10^{-9}$	$1.5{\cdot}10^{-8}$	$2.3717{\cdot}10^{+2}$	$2.78{\cdot}10^{+4}$
10^{-10}	10^{-10}	55854	28597248	8	$4.7{\cdot}10^{-9}$	$1.6{\cdot}10^{-8}$	$1.0489{\cdot}10^{+3}$	$2.73{\cdot}10^{+4}$
10^{-12}	10^{-10}	55854	28597248	8	$6.3{\cdot}10^{-9}$	$8.8{\cdot}10^{-9}$	$1.9641{\cdot}10^{+3}$	$1.46{\cdot}10^{+4}$
10^{-12}	10^{-12}	132490	67834880	9	$5.6{\cdot}10^{-9}$	$7.0{\cdot}10^{-9}$	$4.4599{\cdot}10^{+3}$	$1.52{\cdot}10^{+4}$

Table 7. Free-space Stokes equation (Example 5): Gaussian bump at the origin.

with solution

$$u(\boldsymbol{x}) = \frac{2L}{\mu} \sum_{i=0}^{8} e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2} \big(\nabla \times (\boldsymbol{x} - \boldsymbol{x}_i)\big),$$

for $\boldsymbol{x}_i = \big(\pm\frac{3}{40}, \pm\frac{3}{40}, \pm\frac{3}{40}\big)$, $\mu = 1$, $L = 125$, inside of the $[-1, 1]^3$ box. Errors are again similar to those seen in the fast-decaying experiments from examples 1 and 4; timings are worse, as expected, since we are dealing with nine times as many degrees of freedom per point. Results are shown in Table 7.

Example 6. It is straightforward to extend the solver infrastructure described above to the case of periodic boundary conditions, using the classical method of images of

Lord Rayleigh [54], following the discussion of [24]. The influence of all separated image boxes can be incorporated using either a recursive approach [38] or a scheme based on lattice sums [30]. In either case, the additional work depends only on ϵ_{fmm} and not on the number of degrees of freedom. The main difference is that the unit cell B now has near neighbors, whose influence must be accounted for. This, too, has relatively little impact on performance. A small number of additional boxes are added to both the interaction and near neighbor lists, but no additional data structures are created; instead, everything is handled via careful book-keeping to minimize additional memory consumption.

As an example, we consider the periodic source function

$$f(\boldsymbol{x}) = CM^2\pi^2 \sin(\pi Mx)\sin(\pi My)\sin(\pi Mz),$$

for which the solution is

$$u(\boldsymbol{x}) = C\sin(\pi Mx)\sin(\pi My)\sin(\pi Mz).$$

We conduct our experiments for a nontrivial oscillatory force, choosing $C = 5$ *and* $M = 7$ on the domain $[-1, 1]^3$ with varying degrees of depth and precision. We check the relative L_2 and L_∞ with results shown in Table 8.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	rate
fourth-order force approximation								
10^{-2}	10^{-2}	32768	2097152	6	$2.6{\cdot}10^{-2}$	$3.3{\cdot}10^{-2}$	$3.3250{\cdot}10^{-1}$	$6.31{\cdot}10^{+6}$
10^{-4}	10^{-4}	262144	16777216	7	$2.9{\cdot}10^{-4}$	$7.6{\cdot}10^{-4}$	$1.6296{\cdot}10^{+1}$	$1.03{\cdot}10^{+6}$
10^{-6}	10^{-6}	2097152	134217728	8	$5.6{\cdot}10^{-6}$	$1.9{\cdot}10^{-5}$	$2.9403{\cdot}10^{+2}$	$4.56{\cdot}10^{+5}$
sixth-order force approximation								
10^{-2}	10^{-2}	32768	7077888	6	$2.7{\cdot}10^{-2}$	$3.4{\cdot}10^{-2}$	$6.0730{\cdot}10^{-1}$	$1.17{\cdot}10^{+7}$
10^{-4}	10^{-4}	37248	8045568	7	$2.7{\cdot}10^{-4}$	$8.2{\cdot}10^{-4}$	$1.7895{\cdot}10^{+0}$	$4.50{\cdot}10^{+6}$
10^{-6}	10^{-6}	262144	56623104	7	$5.5{\cdot}10^{-6}$	$1.8{\cdot}10^{-5}$	$3.6250{\cdot}10^{+1}$	$1.56{\cdot}10^{+6}$
10^{-8}	10^{-8}	2097152	452984832	8	$1.0{\cdot}10^{-7}$	$3.0{\cdot}10^{-7}$	$7.2101{\cdot}10^{+2}$	$6.28{\cdot}10^{+5}$
eighth-order force approximation								
10^{-2}	10^{-2}	4096	2097152	5	$1.8{\cdot}10^{-2}$	$2.8{\cdot}10^{-2}$	$2.5445{\cdot}10^{-1}$	$8.24{\cdot}10^{+6}$
10^{-4}	10^{-4}	32768	16777216	6	$3.7{\cdot}10^{-4}$	$9.5{\cdot}10^{-4}$	$2.9459{\cdot}10^{+0}$	$5.70{\cdot}10^{+6}$
10^{-6}	10^{-6}	242432	124125184	7	$6.5{\cdot}10^{-6}$	$2.0{\cdot}10^{-5}$	$5.4881{\cdot}10^{+1}$	$2.26{\cdot}10^{+6}$
10^{-8}	10^{-8}	262144	134217728	7	$1.4{\cdot}10^{-7}$	$5.3{\cdot}10^{-7}$	$1.3578{\cdot}10^{+2}$	$9.88{\cdot}10^{+5}$
10^{-10}	10^{-10}	2097152	1073741824	8	$9.0{\cdot}10^{-9}$	$4.3{\cdot}10^{-8}$	$1.8685{\cdot}10^{+3}$	$5.75{\cdot}10^{+5}$

Table 8. Periodic boundary conditions: Example 6.

It is straightforward to extend this approach to a variety of homogeneous Dirichlet, Neumann or mixed boundary conditions by the method of images as well with very little additional effort.

Example 7. A number of applications require the modeling of source distributions that contain both a smooth component and a singular component. In electrostatics, for example, positively charged ions are often approximated as point charges and the neutralizing electrons as an We consider such a case here. The relevant Poisson equation takes the form

$$\Delta u(\boldsymbol{x}) = f_{\text{smooth}}(\boldsymbol{x}) + \sum_{i=1}^{N} q_i \delta(\boldsymbol{x} - \boldsymbol{x}_i),$$

where the q_i are positive and the neutralizing background takes the form of a sum of Gaussian distributions

$$f^i_{\text{smooth}}(\boldsymbol{x}) = 1/(\sqrt{2\pi\sigma^2})e^{-(\boldsymbol{x}-\sigma)^2/2\sigma^2}$$

centered on each δ-function.

The smooth portion can be handled as above, while the particle sources can be handled with the corresponding particle-based kernel-independent FMM [61].

N_{pts}	ϵ_{fmm}	M_ℓ	L_T	S2M/M2M	Near	M2L	L2L/L2T	T_{FMM}
$1.024{\cdot}10^{+6}$	10^{-2}	4096	5	$2.52{\cdot}10^{-2}$	$6.42{\cdot}10^{-1}$	$5.09{\cdot}10^{-2}$	$4.84{\cdot}10^{-3}$	$7.23{\cdot}10^{-1}$
	10^{-4}	4096	5	$6.97{\cdot}10^{-2}$	$6.49{\cdot}10^{-1}$	$1.29{\cdot}10^{-1}$	$2.35{\cdot}10^{-2}$	$8.71{\cdot}10^{-1}$
	10^{-6}	4096	5	$1.41{\cdot}10^{-1}$	$6.50{\cdot}10^{-1}$	$3.77{\cdot}10^{-1}$	$6.66{\cdot}10^{-2}$	$1.24{\cdot}10^{+0}$
	10^{-8}	4096	5	$2.48{\cdot}10^{-1}$	$6.41{\cdot}10^{-1}$	$1.03{\cdot}10^{+0}$	$1.58{\cdot}10^{-1}$	$2.08{\cdot}10^{+0}$
$4.096{\cdot}10^{+6}$	10^{-2}	32768	6	$1.11{\cdot}10^{-1}$	$1.58{\cdot}10^{+0}$	$2.39{\cdot}10^{-1}$	$1.85{\cdot}10^{-2}$	$1.95{\cdot}10^{+0}$
	10^{-4}	32768	6	$3.04{\cdot}10^{-1}$	$1.61{\cdot}10^{+0}$	$1.32{\cdot}10^{+0}$	$1.01{\cdot}10^{-1}$	$3.34{\cdot}10^{+0}$
	10^{-6}	32768	6	$6.29{\cdot}10^{-1}$	$1.61{\cdot}10^{+0}$	$3.55{\cdot}10^{+0}$	$2.92{\cdot}10^{-1}$	$6.08{\cdot}10^{+0}$
	10^{-8}	32768	6	$1.17{\cdot}10^{+0}$	$1.60{\cdot}10^{+0}$	$9.20{\cdot}10^{+0}$	$6.48{\cdot}10^{-1}$	$1.26{\cdot}10^{+1}$
$1.638{\cdot}10^{+7}$	10^{-2}	262144	7	$1.58{\cdot}10^{+0}$	$4.31{\cdot}10^{+1}$	$2.14{\cdot}10^{+0}$	$2.58{\cdot}10^{-1}$	$4.71{\cdot}10^{+1}$
	10^{-4}	262144	7	$4.31{\cdot}10^{+0}$	$4.34{\cdot}10^{+1}$	$1.14{\cdot}10^{+1}$	$1.48{\cdot}10^{+0}$	$6.06{\cdot}10^{+1}$
	10^{-6}	262144	7	$8.79{\cdot}10^{+0}$	$4.38{\cdot}10^{+1}$	$3.05{\cdot}10^{+1}$	$4.20{\cdot}10^{+0}$	$8.72{\cdot}10^{+1}$
	10^{-8}	262144	7	$1.55{\cdot}10^{+1}$	$4.32{\cdot}10^{+1}$	$8.26{\cdot}10^{+1}$	$9.96{\cdot}10^{+0}$	$1.51{\cdot}10^{+2}$
$6.554{\cdot}10^{+7}$	10^{-2}	2097152	8	$7.00{\cdot}10^{+0}$	$1.04{\cdot}10^{+2}$	$1.80{\cdot}10^{+1}$	$1.40{\cdot}10^{+0}$	$1.31{\cdot}10^{+2}$
	10^{-4}	2097152	8	$1.93{\cdot}10^{+1}$	$1.05{\cdot}10^{+2}$	$9.14{\cdot}10^{+1}$	$6.76{\cdot}10^{+0}$	$2.23{\cdot}10^{+2}$
	10^{-6}	2097152	8	$4.01{\cdot}10^{+1}$	$1.07{\cdot}10^{+2}$	$2.61{\cdot}10^{+2}$	$1.93{\cdot}10^{+1}$	$4.27{\cdot}10^{+2}$
	10^{-8}	2097152	8	$7.34{\cdot}10^{+1}$	$1.06{\cdot}10^{+2}$	$7.00{\cdot}10^{+2}$	$4.19{\cdot}10^{+1}$	$9.21{\cdot}10^{+2}$

Table 9. Example 7: Poisson equation with a mixture of smooth and singular sources.

However, it is trivial to modify our solver to incorporate the particle sources into the S2M operator of Section 4.2 by modifying (22):

$$\boldsymbol{K}_{\mathrm{S2M}}^{B}\boldsymbol{\phi}^{B,u} = \boldsymbol{F}_{\mathrm{S2M}}^{B}\gamma^{B} + \sum_{i=1}^{N} q_i G(\boldsymbol{x}, \boldsymbol{x}_i), \tag{41}$$

where G is the kernel used for evaluating the singular component, consisting of the point charges. Once the point charges are incorporated into $\boldsymbol{\phi}^{B,u}$, the rest of the components of the algorithm (i.e., M2M, M2L, and L2L) take care of the far-field interactions. We need only calculate the influence of near-field particle interactions directly, and evaluate both the local expansions (L2T) and the smooth contributions at particle locations. The latter is done by interpolation, as discussed in Section 4.5.

The performance of our scheme is shown in Table 9 on the previous page.

The upward pass timings are minimally larger than for the particle-only case, and the downward pass timings are agnostic about the nature of the sources, dependent only on the tree-structure itself. The Near computation timings are simply the sum of the particle and volume-based cases, since there is no amortization of cost in this step.

7. Conclusions

We have presented a kernel-independent FMM for solving a variety of constant-coefficient elliptic PDEs in free space, allowing for arbitrary levels of adaptivity, highly nonhomogeneous forces and arbitrarily distributed target locations. Results for the Poisson, modified Helmholtz, and Stokes equations show that the performance is similar for each. Applying the method to other equations requires only a kernel evaluation routine.

Compared to the state-of-the-art technique [47], our method is accurate to higher order and therefore solves similar problems with fewer degrees of freedom, and the work per point is approximately the same. Our current implementation uses OpenMP (but not MPI), although we expect the extension to be straightforward, as our solver is built on top of the MPI-based code of [61]. We discuss how the major loops are optimized for OpenMP shared-memory parallelization in Section A.4 for this current implementation.

As in [24], we have extended our solver to handle periodic, Dirichlet and Neumann boundary conditions for problems on cubic domains using the method of images. We are also coupling the present volume integral code with boundary integral methods to allow for the solution of linear, constant-coefficient, inhomogeneous elliptic PDEs in complex geometries, as in [9]. Additional current work involves incorporating this solver into the state-of-the-art in [43]. These extensions will be reported at a later date.

Appendix

In this appendix, we first verify numerically that the equivalent density representation yields the expected accuracy, followed by a discussion of how the choice of grids affects the order of convergence and an overall numerical justification for the use of Tikhonov regularization. We close with a discussion of how we accelerate the major computation loops of our algorithm using OpenMP shared-memory parallelization and load-balancing techniques.

A.1. ***Equivalent density accuracy.*** As discussed in Section 4.1, we invert several matrices of discretized Fredholm equations of the first kind in order to build out far-field representations,

$$\boldsymbol{K}^{\boldsymbol{y}_d,\boldsymbol{x}_d}\phi_d = \boldsymbol{K}^{\boldsymbol{y}_s,\boldsymbol{x}_d}\phi_s.$$

As in [60], we choose to use Tikhonov regularization [41] when solving these ill-conditioned systems. This solves two problems: in this way, we eliminate the null space in the cases when it is present (Stokes kernel) and we significantly improve accuracy of the inversion for higher numbers of samples (Section A.3). We verify the potential we get from $\phi^{B,u}$, computed using our regularized method in the S2M operation, approximates well $u(\boldsymbol{x})$, computed directly from a force. We test using $g^B = \sum_{a,b,c}^{(a+b+c)\leq(k-1)} x^a y^b z^c$ for box B of width 2 and compute

$$u(\boldsymbol{x}) = \int_B K(\boldsymbol{x},\boldsymbol{y})g^B(\boldsymbol{y})d\boldsymbol{y}, \quad \boldsymbol{x}\in\boldsymbol{x}^{B,u},$$

to within 10^{-16} accuracy using adaptive Gaussian quadrature [8]. We then compute $\phi^{B,u}$ at $\boldsymbol{y}^{B,u}$ using (23) where $(\boldsymbol{K}_{\text{S2M}})^{-1}$ is replaced with

$$(\alpha I + (\boldsymbol{K}_{\text{S2M}})^*\boldsymbol{K}_{\text{S2M}})^{-1}\boldsymbol{K}^*_{\text{S2M}}.$$

For FMM precision, n_p, we choose $\alpha = 10^{-(n_p+1)}$. More details on the choice of α are available in [60]. Our algorithm relies on the fact that for surfaces outside the near field of B, $\phi^{B,u}$ is a sufficiently accurate representation of B's volume force. We compute

$$u(\boldsymbol{x})_{\text{equiv}} = \int_{\boldsymbol{y}^{B,u}} K(\boldsymbol{x},\boldsymbol{y})\phi^{B,u}(\boldsymbol{y})\,d\boldsymbol{y}$$

for $\boldsymbol{x}\in S$, some surface. To evaluate the accuracy of this approximation, we compute

$$u(\boldsymbol{x})_{\text{exact}} = \int_B K(\boldsymbol{x},\boldsymbol{y})g^B(\boldsymbol{y})\,d\boldsymbol{y}$$

up to an accuracy of 10^{-16} [8]. In Figure 7, we compare the infinity-norm of the resulting error for three different kernels (Laplace, modified Helmholtz, Stokes) and varying levels of the polynomial approximation and multiple degrees of FMM

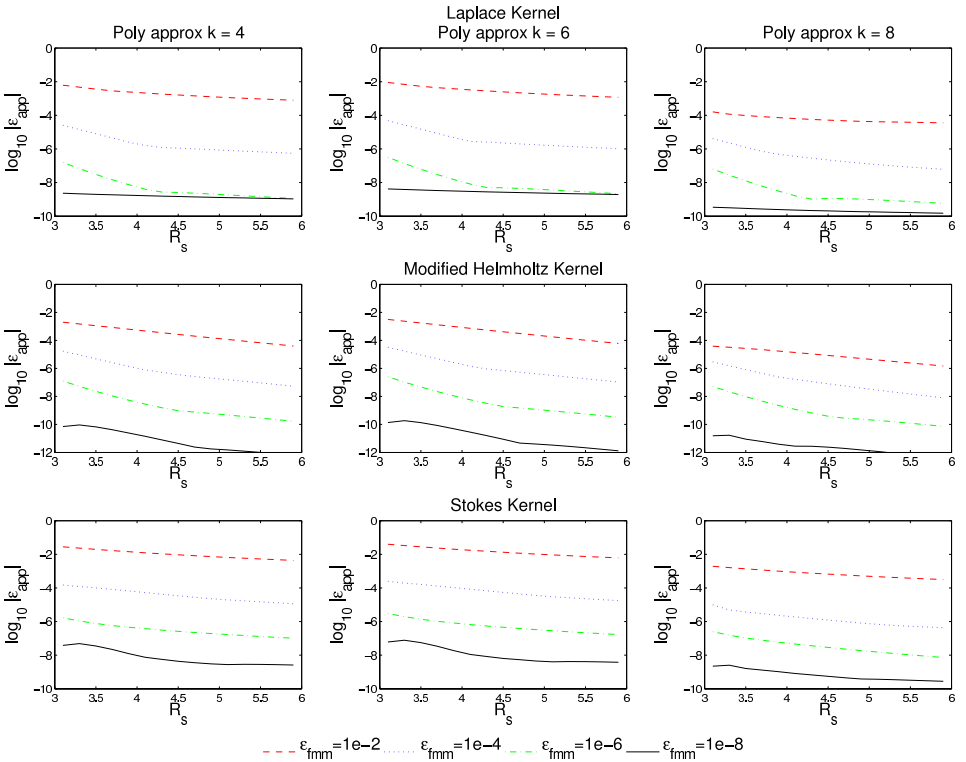


Figure 7. Error due to upward equivalent density approximation of the field. From left to right, three columns show the errors for the polynomial force approximations of degree 4, 6 and 8. Each plot shows four levels of FMM precision, $\epsilon_{\text{fmm}} = 10^{-n_p}$, $p = n_p^3 - (n_p - 2)^3$ points are used on the surfaces $\boldsymbol{y}^{B,u}$ and $\boldsymbol{x}^{B,u}$. For the evaluation surfaces S, we vary the radius R_S from 3.1 to 5.9, the region covering $L_I^B \in \mathcal{F}^B$. The y-axis of each plot is the infinity norm $\|u_{\text{equiv}} - u_{\text{exact}}\|_\infty$ computed over 488 samples on S.

evaluation precision. For each of the kernels of interest, $\phi^{B,u}$, computed by inverting our ill-conditioned kernels, is recovered on each surface S to within the requested degree of precision. For evaluating the accuracy of the kernel inversion and regularization in the computation of $\phi^{B,d}$, we note that this computation is equivalent to the particle-based FMM, of which numerical analysis for the M2L and L2L operators is available in [60].

A.2. ***Polynomial basis and grid spacing.*** As discussed in Section 4.4, we evaluate the solution at a leaf box B on a grid of points $\boldsymbol{x}^{B,g}$, and construct an approximating polynomial from these points. Additionally, we construct a k-th order polynomial

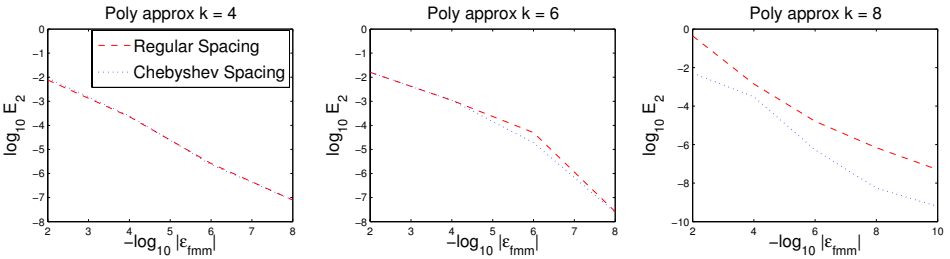


Figure 8. For each of the test examples, the x-axis indicates the negative log of the requested FMM accuracy, ϵ_{fmm}, and the y-axis indicates the log of E_2. The number of points chosen for each ϵ_{fmm} is equivalent to those in Example 1 (page 102) for $\epsilon_{\text{rhs}} = \epsilon_{\text{fmm}}$. Left: for polynomial approximation of degree 4 and $\boldsymbol{x}^{B,g}$ of size 4^3 on each leaf B, overall relative error is close for equispaced and Chebyshev points. Middle: For $n, k = 6$ differences are visible but insignificant. Right: For $n, k = 8$, solutions based on equispaced grid are less accurate.

approximation to B's distributed force if g^B is given on a grid (see Section 4.6). For consistency with AMR codes and efficiency of implementation, it would have been desirable to use uniform grid samples. This approach works well for $n \leq 6$, but it is well-known for large n that equispaced grids lead to instabilities [57]; as a result, for $n > 6$ we use Chebyshev grid points. To show that regularly spaced grid points perform poorly for $n, k > 6$, we consider the following test case:

$$-\Delta u(\boldsymbol{x}) = e^{-L(\|\boldsymbol{x}\|_2)^2}(4L(\|\boldsymbol{x}\|_2)^2 - 6L), \quad L = 250, \boldsymbol{x} \in [-1, 1]^3.$$

In Figure 8, we compare the overall relative L^2 error, E_2, for solutions using equispaced and Chebyshev grid points in the evaluation of the solution and construction of the polynomial approximations of degree 4, 6 and 8. Errors for discretizations using equispaced or Chebyshev grid points are similar for $k \leq$, but for $k = 8$, Chebyshev points are more accurate.

A.3. ***Tikhonov regularization.*** As discussed in Section A.1, we use Tikhonov regularization [41] to invert Fredholm equations of the first kind, specifically the S2M, M2M, and L2L operators in Section 4. Further, in Section A.1, we looked specifically at the accuracy resulting from this inversion process. To justify the overall use of Tikhonov regularization, we consider for the Poisson equation the test case

$$-\Delta u(\boldsymbol{x}) = e^{-L(\|\boldsymbol{x}\|_2)^2}(4L(\|\boldsymbol{x}\|_2)^2 - 6L), \quad L = 250, \boldsymbol{x} \in [-1, 1]^3,$$

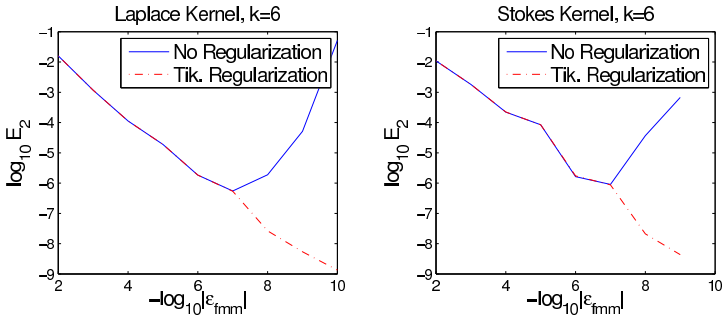


Figure 9. For each of the test examples, the x-axis indicates the negative log of the requested FMM accuracy, ϵ_{fmm}, and the y-axis indicates the log of E_2. The number of points chosen for each ϵ_{fmm} is equivalent to those in Examples 1 and 5 of Section 6 for $\epsilon_{\text{rhs}} = \epsilon_{\text{fmm}}$. Left: for polynomial approximation of degree 6 for the Laplace kernel with and without regularization. Right: for polynomial approximation of degree 6 for the Stokes kernel with and without regularization.

and for the Stokes equation

$$-\Delta u(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \left(8L^3\|\boldsymbol{x}-\boldsymbol{x}_i\|^2 - 20L^2\right)e^{-L\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}\left(\nabla \times (\boldsymbol{x}-\boldsymbol{x}_i)\right),$$
$$L = 125,\ \ \boldsymbol{x} \in [-1,1]^3.$$

In Figure 9, we compare the overall relative L^2 error, E_2, solutions, resulting from Tikhonov regularization versus no regularization and the construction of polynomial approximations of degree $k = 6$ for the right-hand sides (errors for $k = 4, 8$ are similar). For decreasing levels of ϵ_{fmm}, we choose $\epsilon_{\text{fmm}} = \epsilon_{\text{rhs}}$.

We notice that for $\epsilon_{\text{fmm}} > 10^{-7}$, the effect of not employing regularization is equivalent to using regularization for both the Laplace and Stokes operators. However, as ϵ_{fmm} decreases, the number of sample points on the equivalent and check surfaces increases, resulting in larger linear systems, which as mentioned earlier, may be poorly conditioned. Indeed, for such larger systems resulting from $\epsilon_{\text{fmm}} \leq 10^{-7}$, it is necessary to regularize the systems to achieve desirable results.

A.4. ***Shared-memory parallelization and load-balancing.*** We have designed the code to take advantage of shared-memory architectures through the use of OpenMP (see Section 7). In particular, we highlight the steps to accelerate the various major steps in Algorithm 1. For details on the nature of OpenMP and its usage, see [19].

S2M *and* M2M *computations.* In the upward pass (step 2 of Algorithm 1 and Section 4.2), we begin by building a list of all leaf boxes, B in the octree T, which have sources. We then do a simple OpenMP parallelization step over these boxes

for the S2M step. As all components of (23) are of the same size for each leaf box, there is no need to rebalance the load among threads.

In order to ensure proper order of computation, we proceed by sorting all nonleaves in reverse-order by depth. For each nonleaf level in T, beginning at the deepest level, we translate a box B's children's upward equivalent densities to its own through the M2M computation in (25). Again, as each of the components is of the same size, there is no need to rebalance among threads. As we parallelize only among boxes at the same depth in T, level ℓ is not processed until $\ell + 1$ has completed. Further, once we have reached coarse level $\ell = 1$ (which only occurs for periodic or Dirichlet boundary conditions), we discontinue the parallelization.

M2L, L2L, *and* L2T *computations.* In the downward pass of Algorithm 1 (see Section 4.3), we perform a similar operation as above for the M2M step. First, we sort all boxes B in T from the shallowest to deepest levels in the tree. For each level, ℓ, we parallelize among the boxes being processed at that level for the M2L and L2L computations. The L2L components in (29) are of equivalent size for each box B; however, for each box B, the size of L_V^B vary greatly from other boxes (for example, this list is much smaller for boxes on the edge or corners of our domain). To ensure proper balancing among threads, we further sort all boxes for each level, ℓ by the size of L_V^B and then reorder the boxes such that the sum of all L_V^B for each thread is of roughly the same size.

For the L2T computations in (31), we once again build a list of only leaf boxes, for which the target solution is desired, and we parallelize the computations in this list. The components of the discretized equation are all the same size, as with the S2M computation, so there is no need to rebalance among threads for this step.

Near-field computations. We focus our discussion here on the U-list computations. Parallelizing the near-field computations in (34) is the most straightforward in that no leaf box B is dependent on the completion of computations by any other box. That is, we can simply parallelize the computations among leaf boxes, for which the L_U^B exists. However, even more so than with the M2L computations, the sizes of L_U^B can be very different among leaf boxes (especially in the most adaptively refined octrees). Thus, we sort all leaf boxes B in T by the size of L_U^B and reorder the list of leaves such that the sum of the size of L_U^B among each thread is roughly equivalent, ensuring a relatively well-balanced load among threads. The size of the components and operators are the same for each box B, so balancing by list sizes is optimal. We note that this rebalancing is largely unnecessary for uniformly refined trees.

Additionally, for matrix kernels (e.g., Stokes) and larger orders of polynomial approximation, constantly loading large matrices into memory results in little speedup as we increase the number of processes. To correct this, for each equivalence class as described in Section 5, we perform all of the operations involving a single

class first before performing all computations for other classes of operators. Hence, we only load each matrix operator at most once per processor.

We note that for the M2L step, as we have to process level ℓ before moving to level $\ell - 1$, operators will constantly have to be reloaded. Performing all computations in order for each equivalence class at each level is done, but we have seen little time savings for this in practice as opposed to the near-field computations, where it is essential for good speedup.

Remark. As with the near-field computations, for adaptively refined trees, we rebalance the loads among threads for the X and W lists, which involve additional near-field S2T M2T, and S2L computations in Equations (39) and (38), based on the sizes of L_X^B and L_W^B, respectively. Additionally, we perform all computations in order of equivalence class, again loading each matrix operator at most once.

Timing results versus number of processors. To see the effect of our use of OpenMP and load-balancing strategies, we investigate the strong scaling of two fixed problems. First, in Example 1 (page 102), we set the polynomial order, ϵ_{rhs}, and ϵ_{fmm} to 8. The reasoning behind this is to ensure that for a single processor, neither the near-field nor far-field computations fully dominate the timings. In Table 10

N_{procs}	S2M/M2M	Near	M2L	L2L/L2T	T_{FMM}	scaling rate
Poisson equation (Example 1)	$\epsilon_{\text{rhs}} = \epsilon_{\text{fmm}} = 8$	$M_\ell = 5440$	$N_{\text{pts}} = 2785280$			
1	$1.125 \cdot 10^{+0}$	$8.534 \cdot 10^{+0}$	$1.464 \cdot 10^{+1}$	$9.340 \cdot 10^{-1}$	$2.523 \cdot 10^{+1}$	
2	$5.880 \cdot 10^{-1}$	$5.112 \cdot 10^{+0}$	$7.285 \cdot 10^{+0}$	$4.779 \cdot 10^{-1}$	$1.346 \cdot 10^{+1}$	$1.874 \cdot 10^{+0}$
4	$3.750 \cdot 10^{-1}$	$2.377 \cdot 10^{+0}$	$4.559 \cdot 10^{+0}$	$2.927 \cdot 10^{-1}$	$7.604 \cdot 10^{+0}$	$1.770 \cdot 10^{+0}$
8	$1.838 \cdot 10^{-1}$	$1.231 \cdot 10^{+0}$	$2.279 \cdot 10^{+0}$	$1.459 \cdot 10^{-1}$	$3.841 \cdot 10^{+0}$	$1.979 \cdot 10^{+0}$
16	$9.700 \cdot 10^{-2}$	$6.894 \cdot 10^{-1}$	$1.240 \cdot 10^{+0}$	$8.546 \cdot 10^{-2}$	$2.112 \cdot 10^{+0}$	$1.818 \cdot 10^{+0}$
Stokes equations (Example 5)	$\epsilon_{\text{rhs}} = \epsilon_{\text{fmm}} = 6$	$M_\ell = 4894$	$N_{\text{pts}} = 2505728$			
1	$8.794 \cdot 10^{+0}$	$5.416 \cdot 10^{+1}$	$9.402 \cdot 10^{+1}$	$6.832 \cdot 10^{+0}$	$1.638 \cdot 10^{+2}$	
2	$5.182 \cdot 10^{+0}$	$2.857 \cdot 10^{+1}$	$5.213 \cdot 10^{+1}$	$3.617 \cdot 10^{+0}$	$8.951 \cdot 10^{+1}$	$1.830 \cdot 10^{+0}$
4	$2.866 \cdot 10^{+0}$	$1.307 \cdot 10^{+1}$	$3.029 \cdot 10^{+1}$	$1.733 \cdot 10^{+0}$	$4.797 \cdot 10^{+1}$	$1.865 \cdot 10^{+0}$
8	$1.569 \cdot 10^{+0}$	$6.781 \cdot 10^{+0}$	$1.613 \cdot 10^{+1}$	$8.367 \cdot 10^{-1}$	$2.532 \cdot 10^{+1}$	$1.894 \cdot 10^{+0}$
16	$8.248 \cdot 10^{-1}$	$3.611 \cdot 10^{+0}$	$9.452 \cdot 10^{+0}$	$3.918 \cdot 10^{-1}$	$1.428 \cdot 10^{+1}$	$1.772 \cdot 10^{+0}$

Table 10. Timings (in wall-time seconds) for the various components of the FMM volume solver for two fixed problem sizes. The tree level, L_T, is 7 and the polynomial order is 8 and in each case. N_{procs}, M_ℓ, and N_{pts} are the number of processors, leaves, and points; we scale N_{procs} linearly. We separate the S2M/M2M, Near (U,W, X-list computations), M2L (V-list computations), and L2L/L2T timings, with the total shown as T_{FMM}.

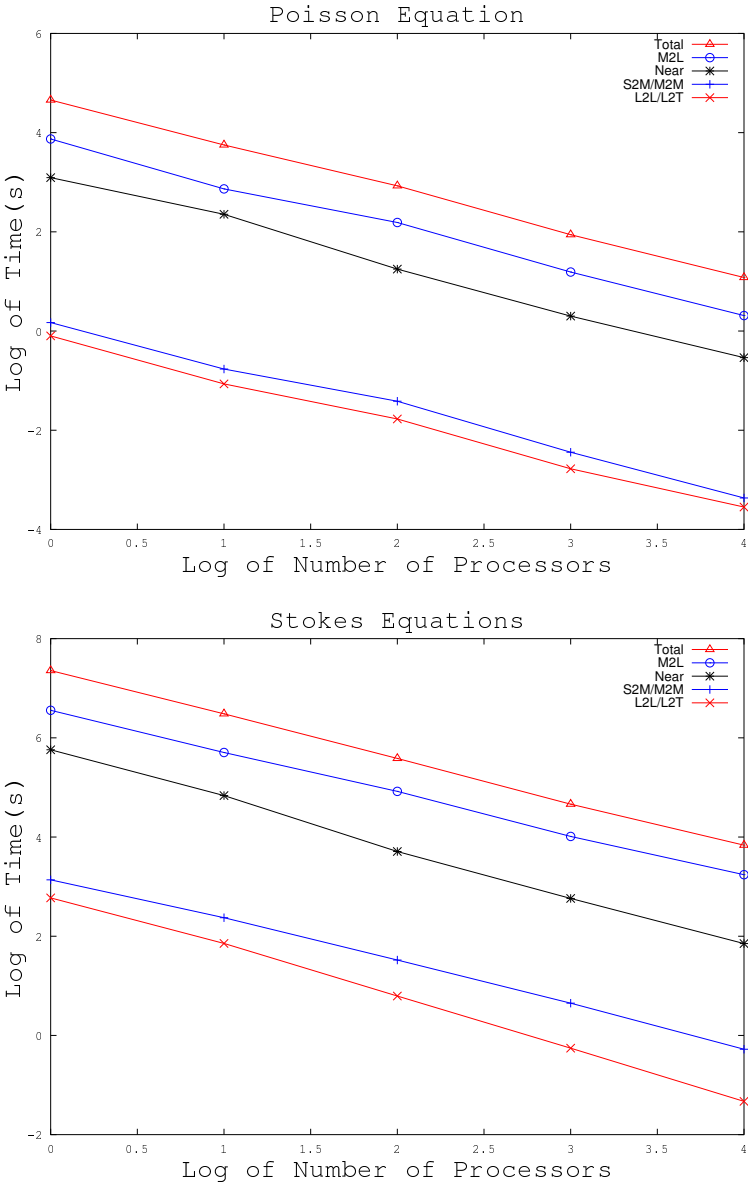


Figure 10. Log-log plots for timings from Table 10.

(top part) we look at the timings for the different algorithmic steps (note that the near-field computation times include U, W, and X list computation times) and we plot the decreasing times in Figure 10, top.

For our second study of the effect of shared-memory parallelization, we look at the Stokes kernel tests from Example 5 (page 107). We fix the polynomial order at

8 and look at $\epsilon_{\text{rhs}} = \epsilon_{\text{fmm}} = 6$, again in an effort to not have one step fully dominate the computational time, allowing us to look at the effect of scaling the number of processors. Timing results can be seen in the bottom parts of Table 10 and Figure 10.

As can be seen in Table 10, our scheme exhibits the desirable result of nearly linear speedup as we scale the number of processors. As indicated in the conclusion, current work is being done to incorporate this work with [43] in order to achieve parallelization on a significantly larger scale.

Acknowledgement

The authors acknowledge the New York University HPC resources,[2] which contributed to the research results reported within this paper. These resources have been largely funded by the NYU Information Technology Services group and an Office of Naval Research DURIP program grant from the Center for Atmosphere Ocean Science (CAOS) at the Courant Institute.

References

[1] M. F. Adams and J. Demmel, *Parallel multigrid solver algorithms and implementations for 3D unstructured finite element problem*, Internat. J. Numer. Methods Engrg. **48**, no. 8, 1241–1262.

[2] M. J. Aftosmis, M. J. Berger, and J. E. Melton, *Adaptive Cartesian mesh generation*, The handbook of grid generation (J. F. Thompson, ed.), CRC Press, Boca Raton, FL, 1998, pp. 22–1–22–26.

[3] C. R. Anderson, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, J. Comput. Phys. **62** (1986), no. 1, 111–123. MR 87d:76050 Zbl 0575.76031

[4] G. T. Balls and P. Colella, *A finite difference domain decomposition method using local corrections for the solution of Poisson's equation*, J. Comput. Phys. **180** (2002), no. 1, 25–53. MR 2003c:65102 Zbl 1003.65140

[5] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force calculation algorithm*, Nature **324** (1986), 446–449.

[6] R. Beatson and L. Greengard, *A short course on fast multipole methods*, Wavelets, multilevel methods and elliptic PDEs (M. Ainsworth et al., eds.), Clarendon Press, Oxford, 1997, pp. 1–37. MR 99a:65142 Zbl 0882.65106

[7] M. J. Berger, M. Aftosmis, and J. Melton, *Accuracy, adaptive methods and complex geometry*, Proc. 1st AFOSR Conference on Dynamic Motion CFD (L. Sakell and D. Knight, eds.), 1996.

[8] J. Berntsen, T. O. Espelid, and A. Genz, *Algorithm 698: DCUHRE: an adaptive multidimensional integration routine for a vector of integrals*, ACM Trans. Math. Software **17** (1991), no. 4, 452–456. MR 1140035 Zbl 0900.65053

[9] G. Biros, L. Ying, and D. Zorin, *A fast solver for the Stokes equations with distributed forces in complex geometries*, J. Comput. Phys. **193** (2004), no. 1, 317–348. MR 2022697 Zbl 1047.76065

[2] http://www.nyu.edu/its/research/hpc

[10] S. Börm, *H^2-matrix arithmetics in linear complexity*, Computing **77** (2006), no. 1, 1–28. MR 2006k:65111 Zbl 1086.65036

[11] S. Börm and W. Hackbusch, *Hierarchical quadrature for singular integrals*, Computing **74** (2005), no. 2, 75–100. MR 2006c:41037 Zbl 1003.65140

[12] A. H. Boschitsch, M. O. Fenley, and W. K. Olson, *A fast adaptive multipole algorithm for calculating screened Coulomb (Yukawa) interactions*, J. Comput. Phys. **151** (1999), no. 1, 212–241. MR 1701576 Zbl 1017.92500

[13] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp. **31** (1977), no. 138, 333–390. MR 55 #4714 Zbl 0373.65054

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. MR 2001h:65002 Zbl 0958.65128

[15] B. L. Buzbee, G. H. Golub, and C. W. Nielson, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal. **7** (1970), 627–656. MR 44 #4920 Zbl 0217.52902

[16] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral methods in fluid dynamics*, Springer, New York, 1988. MR 89m:76004 Zbl 0658.76001

[17] T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, *Domain decomposition methods*, SIAM, Philadelphia, 1989. MR 89j:65010

[18] T. F. Chan and B. F. Smith, *Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes*, Electron. Trans. Numer. Anal. **2** (1994), no. Dec., 171–182. MR 95i:65173 Zbl 0852.65108

[19] B. Chapman, G. Jost, and R. Pas, *Using openmp: Portable shared memory parallel programming (scientific and engineering computation)*, (2007).

[20] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, J. Comput. Phys. **155** (1999), no. 2, 468–498. MR 2000h:65178 Zbl 0937.65126

[21] H. Cheng, W. Y. Crutchfield, Z. Gimbutas, L. F. Greengard, J. F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao, *A wideband fast multipole method for the Helmholtz equation in three dimensions*, J. Comput. Phys. **216** (2006), no. 1, 300–325. MR 2007a:65193 Zbl 1093.65117

[22] H. Cheng, J. Huang, and T. J. Leiterman, *An adaptive fast solver for the modified Helmholtz equation in two dimensions*, J. Comput. Phys. **211** (2006), no. 2, 616–637. MR 2006e:65242 Zbl 1117.65161

[23] G. Chesshire and W. D. Hanshaw, *Composite overlapping meshes for the solution of partial differential equations*, J. Comput. Phys. **90** (1990), no. 1, 1–64. MR 91f:76043 Zbl 0709.65090

[24] F. Ethridge and L. Greengard, *A new fast-multipole accelerated Poisson solver in two dimensions*, SIAM J. Sci. Comput. **23** (2001), no. 3, 741–760. MR 2002i:65146 Zbl 1002.65131

[25] L. Greengard, *The rapid evaluation of potential fields in particle systems*, MIT Press, Cambridge, MA, 1988. MR 89k:31008 Zbl 1001.31500

[26] ———, *Fast algorithms for classical physics*, Science **265** (1994), no. 5174, 909–914. MR 95f:65236

[27] L. Greengard and J. Huang, *A new version of the fast multipole method for screened Coulomb interactions in three dimensions*, J. Comput. Phys. **180** (2002), no. 2, 642–658. MR 2003h:78014 Zbl 1143.78372

[28] L. Greengard, M. C. Kropinski, and A. Mayo, *Integral equation methods for Stokes flow and isotropic elasticity in the plane*, J. Comput. Phys. **125** (1996), no. 2, 403–414. MR 97a:73022 Zbl 0847.76066

[29] L. Greengard and J.-Y. Lee, *A direct adaptive Poisson solver of arbitrary order accuracy*, J. Comput. Phys. **125** (1996), no. 2, 415–424. MR 96m:65090 Zbl 0851.65090

[30] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys. **73** (1987), no. 2, 325–348. MR 88k:82007 Zbl 0629.65005

[31] ———, *The rapid evaluation of potential fields in three dimensions*, Vortex methods: Proceedings of the U.C.L.A. Workshop (C. Anderson and C. Greengard, eds.), Lecture Notes in Math., no. 1360, Springer, Berlin, 1988, pp. 121–141. MR 979565 Zbl 0661.70006

[32] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., no. 6, Cambridge Univ. Press, 1997, pp. 229–269. MR 99c:65012 Zbl 0889.65115

[33] N. A. Gumerov and R. Duraiswami, *Fast multipole method for the biharmonic equation in three dimensions*, J. Comput. Phys. **215** (2006), no. 1, 363–383. MR 2006j:65373 Zbl 1103.65122

[34] W. Hackbusch, *A sparse matrix arithmetic based on* H*-matrices. I. Introduction to* H*-matrices*, Computing **62** (1999), no. 2, 89–108. MR 2000c:65039 Zbl 0927.65063

[35] W. Hackbusch and S. Börm, H^2*-matrix approximation of integral operators by interpolation*, Appl. Numer. Math. **43** (2002), no. 1-2, 129–143. MR 1936106 Zbl 1019.65103

[36] W. Hackbusch and Z. P. Nowak, *On the fast matrix multiplication in the boundary element method by panel clustering*, Numer. Math. **54** (1989), no. 4, 463–491. MR 89k:65162 Zbl 0641.65038

[37] W. Hackbusch and U. Trottenberg (eds.), *Multigrid methods*, Lecture Notes in Mathematics, no. 960, Springer, Berlin, 1982. MR 84b:65007 Zbl 0497.00015

[38] J. Helsing, *Fast and accurate calculations of structural parameters for suspensions*, Proc. Roy. Soc. Lond. A **445** (1994), 127–140.

[39] J. Huang and L. Greengard, *A fast direct solver for elliptic partial differential equations on adaptively refined meshes*, SIAM J. Sci. Comput. **21** (1999/00), no. 4, 1551–1566. MR 2001c:65132 Zbl 0957.65091

[40] H. Johansen and P. Colella, *A Cartesian grid embedded boundary method for Poisson's equation on irregular domains*, J. Comput. Phys. **147** (1998), no. 1, 60–85. MR 99m:65231 Zbl 0923.65079

[41] R. Kress, *Linear integral equations*, 2nd ed., Applied Mathematical Sciences, no. 82, Springer, New York, 1999. MR 2000h:45001 Zbl 0920.45001

[42] O. A. Ladyzhenskaya, *The mathematical theory of viscous incompressible flow*, 2nd ed., Mathematics and its Applications, no. 2, Gordon and Breach, New York, 1969. MR 40 #7610 Zbl 0184.52603

[43] I. Lashuk, A. Chandramowlishwaran, M. Langston, T. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, *A massively parallel adaptive fast-multipole method on heterogeneous architectures*, SC'2009 Conference, IEEE/ACM SIGARCH, 2009.

[44] D. Martin and K. Cartwright, *Solving Poisson's equations using adaptive mesh refinement*, technical report M96/66, Electronic Research Laboratory, University of California, Berkeley, 1996.

[45] D. J. Mavriplis, *Unstructured grid techniques*, Annu. Rev. Fluid Mech. (1997), no. 29, 473–514. MR 97j:76044

[46] A. Mayo, *Fast high order accurate solution of Laplace's equation on irregular regions*, SIAM J. Sci. Statist. Comput. **6** (1985), no. 1, 144–157. MR 86i:65066 Zbl 0559.65082

[47] P. McCorquodale, P. Colella, G. T. Balls, and S. B. Baden, *A local corrections algorithm for solving Poisson's equation in three dimensions*, Commun. Appl. Math. Comput. Sci. **2** (2007), 57–81. MR 2008i:65291 Zbl 1133.65106

[48] A. McKenney, L. Greengard, and A. Mayo, *A fast Poisson solver for complex geometries*, J. Comput. Phys. **118** (1995), no. 2, 348–355. MR 96a:65179 Zbl 0823.65115

[49] M. L. Minion, *A projection method for locally refined grids*, J. Comput. Phys. **127** (1996), no. 1, 158–178. MR 97g:76072 Zbl 0859.76047

[50] G. J. Rodin and Y. Fu, *Fast solution method for three-dimensional Stokesian many-particle problems*, Comm. Numer. Methods Engrg. **16** (2000), no. 2, 145–149.

[51] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys. **60** (1985), no. 2, 187–207. MR 86k:65120 Zbl 0629.65122

[52] ———, *Rapid solution of integral equations of scattering theory in two dimensions*, J. Comput. Phys. **86** (1990), no. 2, 414–439. MR 90k:76081 Zbl 0686.65079

[53] M. Strain, G. Scuseria, and M. Frisch, *Achieving linear scaling for the electronic quantum Coulomb problem*, Science **271** (1996), 51–53.

[54] J. W. Strutt (Lord Rayleigh), *On the influence of obstacles arranged in rectangular order upon the properties of a medium*, Phil. Mag. **34** (1892), 481–502.

[55] H. Sundar, R. S. Sampath, and G. Biros, *Bottom-up construction and 2:1 balance refinement of linear octrees in parallel*, SIAM J. Sci. Comput. **30** (2008), no. 5, 2675–2708. MR 2010d:68192 Zbl 1186.68554

[56] A.-K. Tornberg and L. Greengard, *A fast multipole method for the three-dimensional Stokes equations*, J. Comput. Phys. **227** (2008), no. 3, 1613–1619. MR 2009g:76110 Zbl 05248608

[57] L. N. Trefethen and D. Bau, III, *Numerical linear algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. MR 98k:65002 Zbl 0874.65013

[58] H. Wang, T. Lei, J. Li, J. Huang, and Z. Yao, *A parallel fast multipole accelerated integral equation scheme for 3D Stokes equations*, Internat. J. Numer. Methods Engrg. **70** (2007), no. 7, 812–839. MR 2008a:76120 Zbl 1194.76221

[59] C. Whitea, B. Johnson, P. M. W. Gill, and M. Head-Gordon, *The continuous fast multipole method*, Chem. Phys. Lett. **230** (1994), 8–16.

[60] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys. **196** (2004), no. 2, 591–626. MR 2005d:65235 Zbl 1053.65095

[61] L. Ying, G. Biros, D. Zorin, and M. H. Langston, *A new parallel kernel-independent fast multipole method*, SC'2003 Conference CD, IEEE/ACM SIGARCH, 2003.

Received April 1, 2011. Revised July 20, 2011.

M. HARPER LANGSTON: harper@cims.nyu.edu
Courant Institute, New York University, 251 Mercer Street, New York 10012, United States
http://cs.nyu.edu/~harper/

LESLIE GREENGARD: greengard@cims.nyu.edu
Courant Institute, New York University, 251 Mercer Street, New York NY 10012, United States
http://math.nyu.edu/faculty/greengar/

DENIS ZORIN: dzorin@cims.nyu.edu
Courant Institute, New York University, 251 Mercer Street, New York 10012, United States
http://mrl.nyu.edu/~dzorin/

Guidelines for Authors

Authors may submit manuscripts in PDF format on-line at the Submission page at pjm.math.berkeley.edu/camcos.

Originality. Submission of a manuscript acknowledges that the manuscript is original and and is not, in whole or in part, published or under consideration for publication elsewhere. It is understood also that the manuscript will not be submitted elsewhere while under consideration for publication in this journal.

Language. Articles in CAMCoS are usually in English, but articles written in other languages are welcome.

Required items. A brief abstract of about 150 words or less must be included. It should be self-contained and not make any reference to the bibliography. If the article is not in English, two versions of the abstract must be included, one in the language of the article and one in English. Also required are keywords and subject classifications for the article, and, for each author, postal address, affiliation (if appropriate), and email address.

Format. Authors are encouraged to use LaTeX but submissions in other varieties of TeX, and exceptionally in other formats, are acceptable. Initial uploads should be in PDF format; after the refereeing process we will ask you to submit all source material.

References. Bibliographical references should be complete, including article titles and page ranges. All references in the bibliography should be cited in the text. The use of BibTeX is preferred but not required. Tags will be converted to the house format, however, for submission you may use the format of your choice. Links will be provided to all literature with known web locations and authors are encouraged to provide their own links in addition to those supplied in the editorial process.

Figures. Figures must be of publication quality. After acceptance, you will need to submit the original source files in vector graphics format for all diagrams in your manuscript: vector EPS or vector PDF files are the most useful.

Most drawing and graphing packages (Mathematica, Adobe Illustrator, Corel Draw, MATLAB, etc.) allow the user to save files in one of these formats. Make sure that what you are saving is vector graphics and not a bitmap. If you need help, please write to graphics@mathscipub.org with details about how your graphics were generated.

White space. Forced line breaks or page breaks should not be inserted in the document. There is no point in your trying to optimize line and page breaks in the original manuscript. The manuscript will be reformatted to use the journal's preferred fonts and layout.

Proofs. Page proofs will be made available to authors (or to the designated corresponding author) at a Web site in PDF format. Failure to acknowledge the receipt of proofs or to return corrections within the requested deadline may cause publication to be postponed.

Communications in Applied Mathematics and Computational Science

vol. 6 no. 1 2011