EBAMRGodunov

P. Colella
D. T. Graves
T. J. Ligocki
B. Van Straalen

Applied Numerical Algorithms Group
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA

March 21, 2014

Contents

1 Introduction

This document describes our numerical method for integrating systems of conservation laws (e.g., the Euler equations of gas dynamics) on an AMR grid hierarchy with embedded boundaries. We use an unsplit, second-order Godunov method, extending the algorithms developed by Colella [3] and Saltzman [5].

2 Notation

All these operations take place in a very similar context to that presented in [2]. For non-embedded boundary notation, refer to that document.

The standard (i, j, k) is not sufficient here to denote a computational cell as there can be multiple VoFs per cell. We define $\boldsymbol{v}$ to be the notation for a VoF and $\boldsymbol{f}$ to be a face. The function $ind(\boldsymbol{v})$ produces the cell which the VoF lives in. We define $\boldsymbol{v}^+(\boldsymbol{f})$ to be the VoF on the high side of face $\boldsymbol{f}$; $\boldsymbol{v}^-(\boldsymbol{f})$ is the VoF on the low side of face $\boldsymbol{f}$; $\boldsymbol{f}_d^+(\boldsymbol{v})$ is the set of faces on the high side of VoF $\boldsymbol{v}$; $\boldsymbol{f}_d^-(\boldsymbol{v})$ is the set of faces on the low side of VoF $\boldsymbol{v}$, where $d \in \{x, y, z\}$ is a coordinate direction (the number of directions is D). Also, we compose these operators to represent the set of VoFs directly connected to a given VoF: $\boldsymbol{v}_d^+(\boldsymbol{v}) = \boldsymbol{v}^+(\boldsymbol{f}_d^+(\boldsymbol{v}))$ and $\boldsymbol{v}_d^-(\boldsymbol{v}) = \boldsymbol{v}^-(\boldsymbol{f}_d^-(\boldsymbol{v}))$. The $\ll$ operator shifts data in the direction of the right hand argument. The shift operator can yield multiple VoFs. In this case, the shift operator includes averaging the values at the shifted-to VoFs.

We follow the same approach in the EB case in defining multilevel data and operators as we did for ordinary AMR. Given an AMR mesh hierarchy $\{\Omega^l\}_{l=0}^{lmax}$, we define the valid VoFs on level l to be

$$\mathcal{V}_{valid}^l = ind^{-1}(\Omega_{valid}^l) \tag{1}$$

and composite cell-centered data

$$\varphi^{comp} = \{\varphi^{l,valid}\}_{l=0}^{lmax}, \varphi^{l,valid} : \mathcal{V}_{valid}^l \rightarrow \mathbb{R}^m \tag{2}$$

For face-centered data,

$$\begin{aligned} \mathcal{F}^{l,d}_{valid} &= ind^{-1}(\Omega^{l,\boldsymbol{e}^d}_{valid}) \\ \vec{F}^{l,valid} &= (F^{l,valid}_0, \ldots, F^{l,valid}_{D-1}) \\ F^{l,valid}_d &: \mathcal{F}^{l,d}_{valid} \rightarrow \mathbb{R}^m \end{aligned} \tag{3}$$

For computations at cell centers the notation

$$CC = A \mid B \mid C$$

means that the 3-point formula A is used for CC if all cell centered values it uses are available, the 2-point formula B is used if current cell borders the high side of the physical domain (i.e., no high side value), and the 2-point formula C is used if current cell borders the low side of the physical domain (i.e., no low side value). A value is "available" if its VoF is not covered and is within the domain of computation. For computations at face centers the analogous notation

$$FC = A \mid B \mid C$$

means that the 2-point formula A is used for FC if all cell centered values it uses are available, the 1-point formula B is used if current face coincides with the high side of the physical domain (i.e., no high side value), and the 1-point formula C is used if current face coincided with the low side of the physical domain (i.e., no low side value).

3 Equations of Motion

We are solving a hyperbolic system of equations of the form

$$\frac{\partial U}{\partial t} + \sum_{d=0}^{\mathbf{D}-1} \frac{\partial F^d}{\partial x^d} = S \tag{4}$$

For 3D polytropic gas dynamics,

$$\begin{aligned} U &= (\rho, \rho u_x, \rho u_y, \rho u_z, \rho E)^T \\ F^x &= \left(\rho u_x, \rho u_x^2, \rho u_x u_y, \rho u_x u_z, \rho u_x E + u_x p\right)^T \\ F^y &= \left(\rho u_y, \rho u_x u_y, \rho u_y^2, \rho u_y u_z, \rho u_y E + u_y p\right)^T \\ F^z &= \left(\rho u_z, \rho u_x u_z, \rho u_z u_y, \rho u_z^2, \rho u_z E + u_z p\right)^T \\ E &= \frac{\gamma p}{(\gamma - 1)\rho} + \frac{|\vec{u}|^2}{2} \end{aligned} \tag{5}$$

We are given boundary conditions on faces at the boundary of the domain and on the embedded boundary. We also assume there may be a change of variables $W = W(U)$ ($W \equiv$

"primitive variables") that can be applied to simplify the calculation of the characteristic structure of the equations. This leads to a similar system of equations in W.

$$\begin{aligned} \frac{\partial W}{\partial t} + \sum_{d=0}^{\mathbf{D}-1} A^d(W) \frac{\partial W^d}{\partial x^d} &= S' \\ A^d &= \nabla_U W \cdot \nabla_U F^d \cdot \nabla_W U \\ S' &= \nabla_U W \cdot S \end{aligned} \tag{6}$$

For 3D polytropic gas dynamics,

$$W = (\rho, u_x, u_y, u_z, p)^T$$

$$A^x = \begin{pmatrix} u_x & \rho & 0 & 0 & 0 \\ 0 & u_x & 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & u_x & 0 & 0 \\ 0 & 0 & 0 & u_x & 0 \\ 0 & \rho c^2 & 0 & 0 & u_x \end{pmatrix}$$

$$A^y = \begin{pmatrix} u_y & 0 & \rho & 0 & 0 \\ 0 & u_y & 0 & 0 & 0 \\ 0 & 0 & u_y & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 & u_y & 0 \\ 0 & 0 & \rho c^2 & 0 & u_y \end{pmatrix}$$

$$A^z = \begin{pmatrix} u_z & 0 & 0 & \rho & 0 \\ 0 & u_z & 0 & 0 & 0 \\ 0 & 0 & u_z & 0 & 0 \\ 0 & 0 & 0 & u_z & \frac{1}{\rho} \\ 0 & 0 & 0 & \rho c^2 & u_z \end{pmatrix}$$

4 Approximations to $\nabla \cdot F$.

To obtain a second-order approximation of the flux divergence in conservative form, first we must interpolate the flux to the face centroid. In two dimensions, this interpolation takes the form

$$\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}} = F_{\boldsymbol{f}}^{n+\frac{1}{2}} + |\bar{x}|(F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^d}^{n+\frac{1}{2}} - F_{\boldsymbol{f}}^{n+\frac{1}{2}}) \tag{7}$$

where $\bar{x}$ is the centroid in the direction d perpendicular to the face normal. In three dimensions, define $(\bar{x}, \bar{y})$ to be the coordinates of the centroid in the plane (d^1, d^2) perpendicular

to the face normal.

$$\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}} = F_{\boldsymbol{f}}^{n+\frac{1}{2}}(1-\bar{x}\bar{y}+|\bar{x}\bar{y}|)+ \tag{8}$$

$$F_{\boldsymbol{f}\ll sign(\bar{x})\boldsymbol{e}^{d1}}^{n+\frac{1}{2}}(|\bar{x}|-|\bar{x}\bar{y}|)+ \tag{9}$$

$$F_{\boldsymbol{f}\ll sign(\bar{x})\boldsymbol{e}^{d2}}^{n+\frac{1}{2}}(|\bar{y}|-|\bar{x}\bar{y}|)+ \tag{10}$$

$$F_{\boldsymbol{f}<<sign(\bar{x})\boldsymbol{e}^{d1}<<sign(\bar{x})\boldsymbol{e}^{d2}}^{n+\frac{1}{2}}(|\bar{x}\bar{y}|) \tag{11}$$

Centroids in any dimension are normalized by Δx and centered at the cell center. This interpolation is only done if the shifts that are used in the interpolation are uniquely-defined and single-valued.

We then define the conservative divergence approximation.

$$\nabla\cdot\vec{F} \equiv (D\cdot\vec{F})^c = \frac{1}{k_{\boldsymbol{v}}h}((\sum_{d=0}^{\mathbf{D}-1}\sum_{\pm=+,-}\sum_{\boldsymbol{f}\in\mathcal{F}_{\boldsymbol{v}}^{d,\pm}}\pm\alpha_{\boldsymbol{f}}\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}}) + \alpha_{\boldsymbol{v}}^{B}F_{\boldsymbol{v}}^{B,n+\frac{1}{2}}) \tag{12}$$

The non-conservative divergence approximation is defined below.

$$\nabla\cdot\vec{F} = (D\cdot\vec{F})^{NC} = \frac{1}{h}\sum_{\pm=+,-}\sum_{d=0}^{\mathbf{D}-1}\pm\bar{F}_{\boldsymbol{v},\pm,d}^{n+\frac{1}{2}} \tag{13}$$

$$\bar{F}_{\boldsymbol{v},\pm,d}^{n+\frac{1}{2}} = \begin{cases} \frac{1}{N(\mathcal{F}_{\boldsymbol{v}}^{d,\pm})}\sum_{\boldsymbol{f}\in\mathcal{F}_{\boldsymbol{v}}^{d,\pm}}F_{\boldsymbol{f}}^{n+\frac{1}{2}} & \text{if } N(\mathcal{F}_{\boldsymbol{v}}^{d,\pm})>0 \\ F_{\boldsymbol{v},\pm,d}^{\text{covered},n+\frac{1}{2}} & \text{otherwise} \end{cases} \tag{14}$$

The preliminary update update of the solution of the solution takes the form:

$$U_{\boldsymbol{v}}^{n+1} = U_{\boldsymbol{v}}^{n} - \Delta t((1-k_{\boldsymbol{v}})(D\cdot\vec{F})_{\boldsymbol{v}}^{NC} + k_{\boldsymbol{v}}(D\cdot\vec{F})_{\boldsymbol{v}}^{c}) \tag{15}$$

$$\delta M = -\Delta t k_{\boldsymbol{v}}(1-k_{\boldsymbol{v}})((D\cdot\vec{F})_{\boldsymbol{v}}^{c} - (D\cdot\vec{F})_{\boldsymbol{v}}^{NC}) \tag{16}$$

δM is the total mass increment that has been unaccounted for in the preliminary update. See the EBAMRTools document for how this mass gets redistributed in an AMR context. On a single level, the redistribution takes the following form:

$$U_{\boldsymbol{v}'}^{n+1} := U_{\boldsymbol{v}'}^{n+1} + w_{\boldsymbol{v},\boldsymbol{v}'},\delta M_{\boldsymbol{v}} \tag{17}$$

$$\boldsymbol{v}' \in \mathcal{N}(\boldsymbol{v}), \tag{18}$$

where $\mathcal{N}(\boldsymbol{v})$ is the set of VoFs that can be connected to $\boldsymbol{v}$ with a monotone path of length ≤ 1. The weights are nonnegative, and satisfy $\sum_{\boldsymbol{v}'\in\mathcal{N}(\boldsymbol{v})}\kappa_{\boldsymbol{v}'}w_{\boldsymbol{v},\boldsymbol{v}'} = 1$.

5 Flux Estimation

Given $U_{\boldsymbol{i}}^n$ and $S_{\boldsymbol{i}}^n$, we want to compute a second-order accurate estimate of the fluxes: $F_{\boldsymbol{f}}^{n+\frac{1}{2}} \approx F^d(\boldsymbol{x}_0+(\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d)h, t^n+\frac{1}{2}\Delta t)$. Specifically, we want to compute the fluxes at the center of the Cartesian grid faces corresponding to the faces of the embedded boundary geometry. In addition, we want to compute fluxes at the centers of Cartesian grid faces corresponding to faces adjacent to VoFs, but that are completely covered. Pointwise operations are conceptually the same for both regular and irregular VoFs. In other operations we specify both the regular and irregular VoF calculation. The transformations $\nabla_U W$ and $\nabla_W U$ are functions of both space and time. We shall leave the precise centering of these transformations vague as this will be application-dependent. In outline, the method is given as follows.

5.1 Flux Estimation in Two Dimensions

1. Transform to primitive variables.

$$W_{\boldsymbol{v}}^n = W(U_{\boldsymbol{v}}^n) \tag{19}$$

2. Compute slopes $\Delta^d W_{\boldsymbol{v}}$. This is described separately in section 6.

3. Compute the effect of the normal derivative terms and the source term on the extrapolation in space and time from cell centers to faces. For $0 \leq d < \mathbf{D}$,

$$\begin{aligned} W_{\boldsymbol{v},\pm,d} &= W_{\boldsymbol{v}}^n + \frac{1}{2}(\pm I - \frac{\Delta t}{h} A_{\boldsymbol{v}}^d) P_\pm(\Delta^d W_{\boldsymbol{v}}) \\ A_{\boldsymbol{v}}^d &= A^d(W_{\boldsymbol{v}}) \\ P_\pm(W) &= \sum_{\pm\lambda_k > 0} (l_k \cdot W) r_k \\ W_{\boldsymbol{v},\pm,d} &= W_{\boldsymbol{v},\pm,d} + \frac{\Delta t}{2} \nabla_U W \cdot S_{\boldsymbol{v}}^n \end{aligned} \tag{20}$$

where λ_k are eigenvalues of $A_{\boldsymbol{i}}^d$, and l_k and r_k are the corresponding left and right eigenvectors. We then extrapolate to the covered faces. First we define the VoFs involved.

$$\begin{aligned} d' &= 1 - d \\ s^d &= sign(n^d) \\ \boldsymbol{v}^{up} &= ind^{-1}(ind(\boldsymbol{v}) + s^{d'}\boldsymbol{e}^{d'} - s^d \boldsymbol{e}^d) \\ \boldsymbol{v}^{side} &= ind^{-1}(ind(\boldsymbol{v}) + s^d \boldsymbol{e}^d) \\ \boldsymbol{v}^{corner} &= ind^{-1}(ind(\boldsymbol{v}) + s^{d'}\boldsymbol{e}^{d'}) \end{aligned} \tag{21}$$

Define $W^{\text{up,side,corner}}$, extrapolations to the edges near the VoFs near $\boldsymbol{v}$.

$$\begin{aligned}
W^{\text{up}} =& W_{\boldsymbol{v}^{\text{up}},\mp,d} \\
W^{\text{side}} =& W_{\boldsymbol{v}^{\text{side}},\mp,d} - s^d \Delta^d W \\
W^{\text{corner}} =& W_{\boldsymbol{v}^{\text{corner}},\mp,d} \\
\Delta^d W =& \begin{cases} \Delta^d W^n_{\boldsymbol{v}^{\text{side}}} & \text{if } n^d > n^{d'} \\ \Delta^d W^n_{\boldsymbol{v}^{\text{corner}}} & \text{otherwise} \end{cases} \\
\Delta^{d'} W =& \begin{cases} \Delta^{d'} W^n_{\boldsymbol{v}^{\text{corner}}} & \text{if } n^d > n^{d'} \\ \Delta^{d'} W^n_{\boldsymbol{v}^{\text{up}}} & \text{otherwise} \end{cases}
\end{aligned} \tag{22}$$

where the slopes are defined in section 6 If any of these vofs does not have a monotone path to the original VoF $\boldsymbol{v}$, we drop order the order of interpolation.

If $|n_d| < |n_{d'}|$:

$$W^{\text{full}} = \frac{|n_d|}{|n_{d'}|} W^{\text{corner}} + (1 - \frac{|n_d|}{|n_{d'}|}) W^{\text{up}} - (\frac{|n^d|}{|n^{d'}|} s^d \Delta^d W + s^{d'} \Delta^{d'} W) \tag{23}$$

$$W^{\text{covered}}_{\boldsymbol{v},\pm,d} = \begin{cases} W^{\text{full}} & \text{if both exist} \\ W^{\text{up}} & \text{if only } \boldsymbol{v}^{\text{up}} \text{ exists} \\ W^{\text{corner}} & \text{if only } \boldsymbol{v}^{\text{corner}} \text{ exists} \\ W^n_{\boldsymbol{v}} & \text{if neither exists} \end{cases} \tag{24}$$

If $|n_d| \geq |n_{d'}|$:

$$W^{\text{full}} = \frac{|n_{d'}|}{|n_d|} W^{\text{corner}} + (1 - \frac{|n_{d'}|}{|n_d|}) W^{\text{side}} - (\frac{|n^{d'}|}{|n^d|} s^{d'} \Delta^{d'} W + s^d \Delta^d W) \tag{25}$$

$$W^{\text{covered}}_{\boldsymbol{v},\pm,d} = \begin{cases} W^{\text{full}} & \text{if both exist} \\ W^{\text{side}} & \text{if only } \boldsymbol{v}^{\text{side}} \text{ exists} \\ W^{\text{corner}} & \text{if only } \boldsymbol{v}^{\text{corner}} \text{ exists} \\ W^n_{\boldsymbol{v}} & \text{if neither exists} \end{cases} \tag{26}$$

4. Compute estimates of F^d suitable for computing 1D flux derivatives $\frac{\partial F^d}{\partial x^d}$ using a Riemann solver for the interior, R, and for the boundary, R_B.

$$\begin{aligned} F^{\mathsf{1D}}_{\boldsymbol{f}} &= R(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d}, W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d}, d) \\ &\quad | \;\; R_B(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ &\quad | \;\; R_B(W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ d &= dir(\boldsymbol{f}) \end{aligned} \tag{27}$$

5. Compute the covered fluxes $F^{\mathsf{1D,covered}}$

$$\begin{aligned} F^{\mathsf{1D,\ covered}}_{\boldsymbol{v},+,d} &= R(W_{\boldsymbol{v},+,d}, W^{\mathsf{covered}}_{\boldsymbol{v},+,d}, d) \\ F^{\mathsf{1D,\ covered}}_{\boldsymbol{v},-,d} &= R(W^{\mathsf{covered}}_{\boldsymbol{v},-,d}, W_{\boldsymbol{v},-,d}, d) \end{aligned} \tag{28}$$

6. Compute corrections to $W_{\boldsymbol{i},\pm,d}$ due to the transverse derivatives. For regular cells, this takes the following form.

$$W^{n+\frac{1}{2}}_{\boldsymbol{i},\pm,d} = nW_{\boldsymbol{i},\pm,d} \; - \; \frac{\Delta t}{2h}\nabla_U W \cdot (F^{\mathsf{1D}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_1}} - F^{\mathsf{1D}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^{d_1}}) \tag{29}$$

$$d \neq d_1, \; 0 \leq d, d_1 < \mathbf{D} \tag{30}$$

For irregular cells, we compute the transverse derivatives and use them to correct the extrapolated values of U and obtain time-centered fluxes at centers of Cartesian faces. In two dimensions, this takes the form

$$\begin{aligned} D^{d,\perp}F_{\boldsymbol{v}} &= \frac{1}{h}(\bar{F}_{\boldsymbol{v},+,d_1} - \bar{F}_{\boldsymbol{v},-,d_1}) \\ \bar{F}_{\boldsymbol{v},\pm,d'} &= \begin{cases} \frac{1}{N_{\boldsymbol{v},\pm,d'}} \sum_{\boldsymbol{f}\in\mathcal{F}_{\boldsymbol{v},\pm,d'}} F^{1D}_{\boldsymbol{f},\pm,d'} & \text{if } N_{\boldsymbol{v},\pm,d'} > 0 \\ F^{\mathsf{1D,\ covered}}_{\boldsymbol{v},\pm,d'} & \text{otherwise} \end{cases} \\ d \neq d_1, \quad & 0 \leq d, d_1 < \mathbf{D} \\ W^{n+\frac{1}{2}}_{\boldsymbol{v},\pm,d} =& W_{\boldsymbol{v},\pm,d} - \frac{\Delta t}{2}\nabla_U W(D^{d,\perp}F_{\boldsymbol{v}}) \end{aligned} \tag{31}$$

Extrapolate to covered faces with the procedure described above using $W^{n+\frac{1}{2}}_{\cdot,\mp,d}$ to form $W^{n+\frac{1}{2},\mathsf{covered}}_{\cdot,\pm,d}$.

7. Compute the flux estimate.

$$\begin{aligned} F_{\boldsymbol{f}}^{n+\frac{1}{2}} &= R(W_{\boldsymbol{v}^-(\boldsymbol{f}),+,d}^{n+\frac{1}{2}}, W_{\boldsymbol{v}^+(\boldsymbol{f}),-,d}^{n+\frac{1}{2}}, d) \\ &\mid R_B(W_{\boldsymbol{v}^-(\boldsymbol{f}),+,d}^{n+\frac{1}{2}}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ &\mid R_B(W_{\boldsymbol{v}^+(\boldsymbol{f}),-,d}^{n+\frac{1}{2}}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ F_{\boldsymbol{v},-,d}^{n+\frac{1}{2},\text{covered}} &= R(W_{\boldsymbol{v},+,d}^{n+\frac{1}{2},\text{covered}}, W_{\boldsymbol{v},-,d}^{n+\frac{1}{2}}, d) \\ F_{\boldsymbol{v},+,d}^{n+\frac{1}{2},\text{covered}} &= R(W_{\boldsymbol{v},+,d}^{n+\frac{1}{2}}, W_{\boldsymbol{v},+,d}^{n+\frac{1}{2},\text{covered}}, d) \end{aligned} \tag{32}$$

8. Modify the flux with artificial viscosity where the flow is compressive.

5.2 Flux Estimation in Three Dimensions

1. Transform to primitive variables.

$$W_{\boldsymbol{v}}^n = W(U_{\boldsymbol{v}}^n) \tag{33}$$

2. Compute slopes $\Delta^d W_{\boldsymbol{v}}$. This is described separately in section 6.

3. Compute the effect of the normal derivative terms and the source term on the extrapolation in space and time from cell centers to faces. For $0 \le d < \mathbf{D}$,

$$\begin{aligned} W_{\boldsymbol{v},\pm,d} &= W_{\boldsymbol{v}}^n + \frac{1}{2}(\pm I - \frac{\Delta t}{h} A_{\boldsymbol{v}}^d) P_\pm(\Delta^d W_{\boldsymbol{v}}) \\ A_{\boldsymbol{v}}^d &= A^d(W_{\boldsymbol{v}}) \\ P_\pm(W) &= \sum_{\pm\lambda_k > 0} (l_k \cdot W) r_k \\ W_{\boldsymbol{v},\pm,d} &= W_{\boldsymbol{v},\pm,d} + \frac{\Delta t}{2} \nabla_U W \cdot S_{\boldsymbol{v}}^n \end{aligned} \tag{34}$$

where λ_k are eigenvalues of $A_{\boldsymbol{i}}^d$, and l_k and r_k are the corresponding left and right eigenvectors.

We then extrapolate to the covered faces. Define the direction of the face normal to be d_f and d_1, d_2 to be the directions tangential to the face. The procedure develops as follows

- We define the associated vofs.
- We form a 2x2 grid of values along a plane h away from the covered face and bilinearly interpolate to the point where the normal intersects the plane.
- We use the slopes of the solution to extrapolate along the normal to get a second-order approximation of the solution at the covered face.

Which plane is selected is determined by the direction of the normal. If any of these VoFs does not have a monotone path to the original VoF $\boldsymbol{v}$, we drop order the order of interpolation.

If $|n_f| \geq |n_{d_1}|$ and $|n_{d_f}| \geq |n_{d_2}|$:

$$\begin{aligned}
\boldsymbol{v}^{00} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_f}\boldsymbol{e}^{d_f}) \\
\boldsymbol{v}^{10} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1}) \\
\boldsymbol{v}^{01} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_2}\boldsymbol{e}^{d_2}) \\
\boldsymbol{v}^{11} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1} + s^{d_2}\boldsymbol{e}^{d_2}) \\
W^{00} =& W_{\boldsymbol{v}^{00},\mp,d_f} - s^{d_f}\Delta^{d_f} W_{\boldsymbol{v}^{00}} \\
W^{10} =& W_{\boldsymbol{v}^{10},\mp,d_f} \\
W^{01} =& W_{\boldsymbol{v}^{01},\mp,d_f} \\
W^{11} =& W_{\boldsymbol{v}^{11},\mp,d_f}
\end{aligned} \tag{35}$$

We form a bilinear function $W(x_{d_1}, x_{d_2})$ in the plane formed by the four faces at which the values live:

$$\begin{aligned}
W(x_{d_1}, x_{d_2}) =& Ax_{d_1} + Bx_{d_2} + Cx_{d_1}x_{d_2} + D \\
A =& s^{d_1}(W^{10} - W^{00}) \\
B =& s^{d_2}(W^{01} - W^{00}) \\
C =& s^{d_1}s^{d_2}(W^{11} - W^{00}) - (W^{10} - W^{00}) - (W^{01} - W^{00}) \\
D =& W^{00}
\end{aligned} \tag{36}$$

We then extrapolate to the covered face from the point on the plane where the normal intersects

$$W^{\text{full}} = W(s^{d_1}\frac{|n_{d_1}|}{|n_{d_f}|}, s^{d_2}\frac{|n_{d_2}|}{|n_{d_f}|}) - \Delta^{d_f} W_{\boldsymbol{v}^{00}} - s^{d_1}\frac{|n_{d_1}|}{|n_{d_f}|}\Delta^{d_1} W_{\boldsymbol{v}^{10}} - s^{d_2}\frac{|n_{d_2}|}{|n_{d_f}|}\Delta^{d_2} W_{\boldsymbol{v}^{01}} \tag{37}$$

Otherwise (assume $|n_{d_1}| \geq |n_{d_f}|$ and $|n_{d_1}| \geq |n_{d_2}|$):

$$\begin{aligned}
\boldsymbol{v}^{00} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1}) \\
\boldsymbol{v}^{10} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1}) - s^{d_f}\boldsymbol{e}^{d_f} \\
\boldsymbol{v}^{01} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1}) + s^{d_2}\boldsymbol{e}^{d_2} \\
\boldsymbol{v}^{11} =& ind^{-1}(ind(\boldsymbol{v}) + s^{d_1}\boldsymbol{e}^{d_1} - s^{d_f}\boldsymbol{e}^{d_f} + s^{d_2}\boldsymbol{e}^{d_2} \\
W^{00} =& W_{\boldsymbol{v}^{00},\mp,d_f} \\
W^{10} =& W_{\boldsymbol{v}^{10},\mp,d_f} \\
W^{01} =& W_{\boldsymbol{v}^{01},\mp,d_f} \\
W^{11} =& W_{\boldsymbol{v}^{11},\mp,d_f}
\end{aligned} \tag{38}$$

We form a bilinear function $W(x_{d_1}, x_{d_2})$ in the plane formed by the four faces at which the values live. This is shown in equation 36. We then extrapolate to the covered face from the point on the plane where the normal intersects

$$W^{\text{full}} = W(s^{d_f}\frac{|n_{d_f}|}{|n_{d_1}|}, s^{d_2}\frac{|n_{d_2}|}{|n_{d_1}|}) - \Delta^{d_1} W_{\boldsymbol{v}^{00}} - s^{d_f}\frac{|n_{d_f}|}{|n_{d_1}|}\Delta^{d_f} W_{\boldsymbol{v}^{10}} - s^{d_2}\frac{|n_{d_2}|}{|n_{d_1}|}\Delta^{d_2} W_{\boldsymbol{v}^{01}} \tag{39}$$

In either case,

$$W^{\text{covered}}_{\boldsymbol{v},\pm,d} = \begin{cases} W^{\text{full}} & \text{if all four VoFs exist} \\ W^n_{\boldsymbol{v}} & \text{otherwise} \end{cases} \tag{40}$$

4. Compute estimates of F^d suitable for computing 1D flux derivatives $\frac{\partial F^d}{\partial x^d}$ using a Riemann solver for the interior, R, and for the boundary, R_B.

$$\begin{aligned} F^{\text{1D}}_{\boldsymbol{f}} &= R(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d}, W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d}, d) \\ &\quad | \;\; R_B(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ &\quad | \;\; R_B(W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d}, (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, d) \\ d &= dir(\boldsymbol{f}) \end{aligned} \tag{41}$$

5. Compute the covered fluxes $F^{\text{1D,covered}}$

$$\begin{aligned} F^{\text{1D, covered}}_{\boldsymbol{v},+,d} &= R(W_{\boldsymbol{v},+,d}, W^{\text{covered}}_{\boldsymbol{v},+,d}, d) \\ F^{\text{1D, covered}}_{\boldsymbol{v},-,d} &= R(W^{\text{covered}}_{\boldsymbol{v},-,d}, W_{\boldsymbol{v},-,d}, d) \end{aligned} \tag{42}$$

6. Compute corrections to $U_{\boldsymbol{i},\pm,d}$ corresponding to one set of transverse derivatives appropriate to obtain $(1,1,1)$ diagonal coupling. This step is only meaningful in three dimensions. We compute 1D flux differences, and use them to compute $U_{\boldsymbol{v},\pm,d_1,d_2}$, the d_1-edge-centered state partially updated by the effect of derivatives in the d_1, d_2 directions.

$$\begin{aligned} D^{\text{1D}}_d F^{\text{1D}}_{\boldsymbol{v}} &= \frac{1}{h}(\bar{F}^{\text{1D}}_{\boldsymbol{v},+,d} - \bar{F}^{\text{1D}}_{\boldsymbol{v},-,d}) \\ \bar{F}_{\boldsymbol{v},\pm,d} &= \begin{cases} \frac{1}{N_{\pm,d}}(\sum\limits_{\boldsymbol{f}\in\mathcal{F}_{\boldsymbol{v},\pm,d}} F^{\text{1D}}_{\boldsymbol{f}}) & \text{if } N_{\boldsymbol{v},\pm,d} > 0 \\ F^{\text{1D, covered}}_{\boldsymbol{v},\pm,d} & \text{otherwise} \end{cases} \end{aligned} \tag{43}$$

$$W_{\boldsymbol{v},\pm,d_1,d_2} = W_{\boldsymbol{v},\pm,d_1} - \frac{\Delta t}{3}\nabla_U W(D^{\text{1D}}_{d_2} F^{\text{1D}})_{\boldsymbol{v}} \tag{44}$$

We then extrapolate to covered faces with the procedure described above using $W_{\cdot,\pm,d_1,d_2}$ to form $W^{\text{covered},d}_{\cdot,\pm,d_1,d_2}$ and compute an estimate to the fluxes:

$$\begin{aligned} F_{\boldsymbol{f},d_1,d_2} &= R(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d_1,d_2}, W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d_1,d_2}, d_1) \\ &\quad \mid\ R_B(W_{\boldsymbol{v}_-(\boldsymbol{f}),+,d_1,d_2}, (\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d)h, d_1) \\ &\quad \mid\ R_B(W_{\boldsymbol{v}_+(\boldsymbol{f}),-,d_1,d_2}, (\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d)h, d_1) \\ d &= dir(\boldsymbol{f}) \\ F^{\text{covered}}_{\boldsymbol{v},-,d_1,d_2} &= R(W^{\text{covered}}_{\boldsymbol{v},-,d1,d2}, W_{\boldsymbol{v},-,d_1,d_2}, d_1) \\ F^{\text{covered}}_{\boldsymbol{v},+,d_1,d_2} &= R(W_{\boldsymbol{v},+,d1,d2}, W^{\text{covered}}_{\boldsymbol{v},+,d_1,d_2}, d_1) \end{aligned} \tag{45}$$

7. Compute final corrections to $W_{\boldsymbol{i},\pm,d}$ due to the final transverse derivatives. We compute the $2\mathbf{D}$ transverse derivatives and use them to correct the extrapolated values of U and obtain time-centered fluxes at centers of Cartesian faces. In three dimensions, this takes the form:

$$\begin{aligned} D^{d,\perp}F_{\boldsymbol{v}} &= \frac{1}{h}(\bar{F}_{\boldsymbol{v},+,d_1,d_2} - \bar{F}_{\boldsymbol{v},-,d_1,d_2} + \bar{F}_{\boldsymbol{v},+,d_2,d_1} - \bar{F}_{\boldsymbol{v},-,d_2,d_1}) \\ \bar{F}_{\boldsymbol{v},\pm,d',d''} &= \begin{cases} \frac{1}{N_{\boldsymbol{v},\pm,d'}} \sum_{\boldsymbol{f}\in\mathcal{F}_{\boldsymbol{v},\pm,d'}} F_{\boldsymbol{f},\pm,d',d''} & \text{if } N_{\boldsymbol{v},\pm,d'} > 0 \\ F^{\text{covered}}_{\boldsymbol{v},\pm,d',d''} & \text{otherwise} \end{cases} \\ d \neq d_1 \neq d_2 &\quad 0 \leq d, d_1, d_2 < \mathbf{D} \\ W^{n+\frac{1}{2}}_{\boldsymbol{v},\pm,d} &= W_{\boldsymbol{v},\pm,d} - \frac{\Delta t}{2}\nabla_U W(D^{d,\perp}F_{\boldsymbol{v}}) \end{aligned} \tag{46}$$

We then extrapolate to covered faces with the procedure described above using $W^{n+\frac{1}{2}}_{\cdot,\pm,d}$ to form $W^{n+\frac{1}{2},\text{covered},d}_{\cdot,\pm,d}$.

8. Compute the flux estimate.

$$\begin{aligned} F^{n+\frac{1}{2}}_{\boldsymbol{f}} &= R(W^{n+\frac{1}{2}}_{\boldsymbol{v}^-(\boldsymbol{f}),+,d}, W^{n+\frac{1}{2}}_{\boldsymbol{v}^+(\boldsymbol{f}),-,d}, d) \\ &\quad \mid\ R_B(W^{n+\frac{1}{2}}_{\boldsymbol{v}^-(\boldsymbol{f}),+,d}, (\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d)h, d) \\ &\quad \mid\ R_B(W^{n+\frac{1}{2}}_{\boldsymbol{v}^+(\boldsymbol{f}),-,d}, (\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d)h, d) \\ F^{n+\frac{1}{2},\text{covered}}_{\boldsymbol{v},-,d} &= R(W^{n+\frac{1}{2},\text{covered}}_{\boldsymbol{v},+,d}, W^{n+\frac{1}{2}}_{\boldsymbol{v},-,d}, d) \\ F^{n+\frac{1}{2},\text{covered}}_{\boldsymbol{v},+,d} &= R(W^{n+\frac{1}{2}}_{\boldsymbol{v},+,d}, W^{n+\frac{1}{2},\text{covered}}_{\boldsymbol{v},+,d}, d) \end{aligned} \tag{47}$$

9. Modify the flux with artificial viscosity where the flow is compressive.

5.3 Modifications for R-Z Computations

For R-Z calculations, we make some adjustments to the algorithm. Specifically, we separate the radial pressure force as a separate flux. This makes free-stream preservation in the radial direction easier to achieve. For this section, we will confine ourselves to the compressible Euler equations.

5.3.1 Equations of Motion

The compressible Euler equations in R-Z coordinates are given by

$$\frac{\partial U}{\partial t} + \frac{1}{r}\frac{\partial (rF^r)}{\partial r} + \frac{1}{r}\frac{\partial (rF^z)}{\partial z} + \frac{\partial H}{\partial r} + \frac{\partial H}{\partial z} = 0 \tag{48}$$

where

$$\begin{aligned} U =& (\rho, \rho u_r, \rho u_z, \rho E)^T \\ F^r =& (\rho u_r, \rho u_r^2, \rho u_r u_z, \rho u_r (E+p))^T \\ F^z =& (\rho u_z, \rho u_r u_z, \rho u_z^2, \rho u_z (E+p))^T \\ H =& (0, p, p, 0)^T \end{aligned} \tag{49}$$

5.3.2 Flux Divergence Approximations

In section 4, we describe our solution update strategy and this remains largely unchanged. Our update still takes the form of equation 16 and redistribution still takes the form of equation 18. The definitions of the divergence approximations do change, however. The volume of a full cell ΔV_j is given by

$$\Delta V_j = (j + \frac{1}{2})h^3 \tag{50}$$

where $(i, j) = ind^{-1}(\boldsymbol{v})$. Define $\kappa_{\boldsymbol{v}}^{vol}$ to be the real volume of the cell that the VoF occupies.

$$\kappa_{\boldsymbol{v}}^{vol} = \frac{1}{\Delta V}\int_{\Delta_{\boldsymbol{v}}} r dr dz = \frac{1}{\Delta V}\int_{\partial\Delta_{\boldsymbol{v}}} \frac{r^2}{2} n_r dl \tag{51}$$

$$\kappa_{\boldsymbol{v}}^{vol} = \frac{h}{2\Delta V}((\alpha r^2)_{\boldsymbol{f}(\boldsymbol{v},+,r)} - (\alpha r^2)_{\boldsymbol{f}(\boldsymbol{v},-,r)} - \alpha_B \bar{r}_{\delta\boldsymbol{v}}^2 n^r) \tag{52}$$

The conservative divergence of the flux in RZ is given by

$$\begin{aligned} (D \cdot \vec{F})_{\boldsymbol{v}}^c = \frac{h}{\Delta V \kappa_{\boldsymbol{v}}^{vol}}&((r\bar{F}^r\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,r)} - (r\bar{F}^r\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,r)} \\ &+ (\bar{r}\bar{F}^z\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,z)} - (\bar{r}\bar{F}^z\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,z)}) \end{aligned}$$

$$\left(\frac{\partial H}{\partial r}\right)^c = \frac{1}{\kappa_{\boldsymbol{v}} h^2}\int \frac{\partial H}{\partial r} dr dz = \frac{1}{\kappa_{\boldsymbol{v}} h^2}\int H n_r dl$$
$$\left(\frac{\partial H}{\partial z}\right)^c = \frac{1}{\kappa_{\boldsymbol{v}} h^2}\int \frac{\partial H}{\partial z} dr dz = \frac{1}{\kappa_{\boldsymbol{v}} h^2}\int H n_z dl$$

We always deal with these divergences in a form multiplied by the volume fraction κ.

$$\kappa_{\boldsymbol{v}}(D\cdot\vec{F})^c_{\boldsymbol{v}} = \frac{h\kappa_{\boldsymbol{v}}}{\Delta V \kappa^{vol}_{\boldsymbol{v}}}((r\bar{F}^r\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,r)} - (r\bar{F}^r\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,r)} + (\bar{r}\bar{F}^z\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,z)} - (\bar{r}\bar{F}^z\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,z)})$$

$$\kappa_{\boldsymbol{v}}\left(\frac{\partial H}{\partial r}\right)^c = \frac{1}{h^2}\int H n_r dl = \frac{1}{h}((H\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,r)} - (H\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,r)})$$
$$\kappa_{\boldsymbol{v}}\left(\frac{\partial H}{\partial z}\right)^c = \frac{1}{h^2}\int H n_z dl = \frac{1}{h}((H\alpha)_{\boldsymbol{f}(\boldsymbol{v},+,z)} - (H\alpha)_{\boldsymbol{f}(\boldsymbol{v},-,z)})$$

where $\bar{F}$ has been interpolated to face centroids where α denotes the ordinary area fraction. The nonconservative divergence of the flux in RZ is given by

$$(D\cdot\vec{F})^{nc}_{\boldsymbol{v}} = \frac{1}{h r_{\boldsymbol{v}}}((rF^r)_{\boldsymbol{f}(\boldsymbol{v},+,r)} - (rF^r)_{\boldsymbol{f}(\boldsymbol{v},-,r)}) + \frac{1}{h}(F^z_{\boldsymbol{f}(\boldsymbol{v},+,z)} - F^z_{\boldsymbol{f}(\boldsymbol{v},-,z)})$$

$$\left(\frac{\partial H}{\partial r}\right)^{nc} = \frac{1}{h}(H_{\boldsymbol{f}(\boldsymbol{v},+,r)} - H_{\boldsymbol{f}(\boldsymbol{v},-,r)})$$
$$\left(\frac{\partial H}{\partial z}\right)^{nc} = \frac{1}{h}(H_{\boldsymbol{f}(\boldsymbol{v},+,z)} - H_{\boldsymbol{f}(\boldsymbol{v},-,z)})$$

5.3.3 Primitive Variable Form of the Equations

In the predictor step, we use the nonconservative form of the equations of motion. See Courant and Friedrichs [4] for derivations.

$$\frac{\partial W}{\partial t} + A^r\frac{\partial W}{\partial r} + A^z\frac{\partial W}{\partial z} = S \tag{53}$$

where

$$W = (\rho, u_r, u_z, p)^T$$
$$S = \left(-\rho\frac{u_r}{r}, 0, 0, -\rho c^2\frac{u_r}{r}\right)^T$$

$$A^r = \begin{pmatrix} u_r & \rho & 0 & 0 \\ 0 & u_r & 0 & \frac{1}{\rho} \\ 0 & 0 & u_r & 0 \\ 0 & \rho c^2 & 0 & u_r \end{pmatrix}$$

$$A^r = \begin{pmatrix} u_z & \rho & 0 & 0 \\ 0 & u_z & 0 & 0 \\ 0 & 0 & u_z & \frac{1}{\rho} \\ 0 & 0 & \rho c^2 & u_z \end{pmatrix}$$

5.3.4 Flux Registers

Refluxing is the balancing the fluxes at coarse-fine interfaces so the coarse side of the interface is using the same flux as the integral of the fine fluxes over the same area. In this way, we maintain strong mass conservation at coarse-fine interfaces. As shown in equation, 5.3.2, the conservative divergence in cylindrical coordinates is has a difference form than in Cartesian coordinates. It is therefore necessary to describe the refluxing operation specifically for cylindrical coordinates.

Let $\vec{F}^{comp} = \{\vec{F}^f, \vec{F}^{c,valid}\}$ be a two-level composite vector field. We want to define a composite divergence $D^{comp}(\vec{F}^f, \vec{F}^{c,valid})_{\boldsymbol{v}}$, for $\boldsymbol{v} \in V^c_{valid}$. We do this by extending $F^{c,valid}$ to the faces adjacent to $\boldsymbol{v} \in V^c_{valid}$, but are covered by $\mathcal{F}^f_{valid}$.

$$< F^f_z >_{\boldsymbol{f}_c} = \left(\frac{\kappa_{\boldsymbol{v}_c}}{\kappa^{vol}_{\boldsymbol{v}_c}\Delta V_{\boldsymbol{v}_c}}\right)\left(\frac{h^2}{(n_{ref})^{(\mathbf{D}-1)}}\right) \sum_{\boldsymbol{f}\in\mathcal{C}^{-1}_{n_{ref}}(\boldsymbol{f}_c)} (\bar{r}\alpha)_{\boldsymbol{f}}(\bar{F}^z + \bar{H})_{\boldsymbol{f}}$$

$$< F^f_r >_{\boldsymbol{f}_c} = \left(\frac{\kappa_{\boldsymbol{v}_c}}{\kappa^{vol}_{\boldsymbol{v}_c}\Delta V_{\boldsymbol{v}_c}}\right)\left(\frac{h^2}{(n_{ref})^{(\mathbf{D}-1)}}\right) \sum_{\boldsymbol{f}\in\mathcal{C}^{-1}_{n_{ref}}(\boldsymbol{f}_c)} (r\alpha)_{\boldsymbol{f}}(F^r + H)_{\boldsymbol{f}}$$

$$F^c_{r,\boldsymbol{f}_c} = \left(\frac{\kappa_{\boldsymbol{v}_c}}{\kappa^{vol}_{\boldsymbol{v}_c}\Delta V_{\boldsymbol{v}_c}}\right)(h^2(r\alpha)_{\boldsymbol{f}_c})(F^r + H)_{\boldsymbol{f}_c}$$

$$F^c_{z,\boldsymbol{f}_c} = \left(\frac{\kappa_{\boldsymbol{v}_c}}{\kappa^{vol}_{\boldsymbol{v}_c}\Delta V_{\boldsymbol{v}_c}}\right)(h^2(\bar{r}\alpha)_{\boldsymbol{f}_c})(\bar{F}^z + \bar{H})_{\boldsymbol{f}_c}$$

$$\boldsymbol{f}_c \in ind^{-1}(\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d), \boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d \in \zeta^f_{d,+} \cup \zeta^f_{d,-}$$

$$\zeta^f_{d,\pm} = \{\boldsymbol{i} \pm \frac{1}{2}\boldsymbol{e}^d : \boldsymbol{i} \pm \boldsymbol{e}^d \in \Omega^c_{valid}, \boldsymbol{i} \in \mathcal{C}_{n_{ref}}(\Omega^f)\}$$

The VoF $\boldsymbol{v}_c$ is the coarse volume that is adjacent to the coarse-fine interface and $r_{\boldsymbol{v}_c}$ is the radius of its cell center. Then we can define $(D\cdot\vec{F})_{\boldsymbol{v}}, \boldsymbol{v} \in \mathcal{V}^c_{valid}$, using the expression above, with $\tilde{F}_{\boldsymbol{f}} =< F^f_d >$ on faces covered by $\mathcal{F}^f$. We can express the composite divergence in terms of a level divergence, plus a correction. We define a flux register $\delta\vec{F}^f$,

associated with the fine level

$$\delta\vec{F}^f = (\delta F^f_{0,\ldots}\delta F^f_{D-1})$$
$$\delta F^f_d : ind^{-1}(\zeta^f_{d,+} \cup \zeta^f_{d,-}) \rightarrow \mathbb{R}^m$$

If $\vec{F}^c$ is any coarse level vector field that extends $\vec{F}^{c,valid}$, i.e. $F^c_d = F^{c,valid}_d$ on $\mathcal{F}^{c,d}_{valid}$ then for $\boldsymbol{v} \in \mathcal{V}^c_{valid}$

$$D^{comp}(\vec{F}^f, \vec{F}^{c,valid})_{\boldsymbol{v}} = (D\vec{F}^c)_{\boldsymbol{v}} + D_R(\delta\vec{F}^c)_{\boldsymbol{v}} \tag{54}$$

Here $\delta\vec{F}^f$ is a flux register, set to be

$$\delta F^f_d =< F^f_d > -F^c_d \text{ on } ind^{-1}(\zeta^c_{d,+} \cup \zeta^c_{d,-}) \tag{55}$$

D_R is the reflux divergence operator. For valid coarse vofs adjacent to Ω^f it is given by

$$\kappa_{\boldsymbol{v}}(D_R\delta\vec{F}^f)_{\boldsymbol{v}} = \sum_{d=0}^{D-1}\Big(\sum_{\boldsymbol{f}:\boldsymbol{v}=\boldsymbol{v}^+(\boldsymbol{f})} \delta F^f_{d,\boldsymbol{f}} - \sum_{\boldsymbol{f}:\boldsymbol{v}=\boldsymbol{v}^-(\boldsymbol{f})} \delta F^f_{d,\boldsymbol{f}}\Big) \tag{56}$$

For the remaining vofs in $\mathcal{V}^f_{valid}$,

$$(D_R\delta\vec{F}^f) \equiv 0 \tag{57}$$

We then add the reflux divergence to adjust the coarse solution U^c to preserve conservation.

$$U^c_{\boldsymbol{v}} \mathrel{+}= \kappa_{\boldsymbol{v}}(D_R(\delta F))_{\boldsymbol{v}} \tag{58}$$

5.4 Artificial Viscosity

The artificial viscosity coefficient is K_0, the velocity is $\vec{u}$ and $d = dir(\boldsymbol{f})$.

$$(D\vec{u})_{\boldsymbol{f}} = (u^d_{\boldsymbol{v}^+(\boldsymbol{f})} - u^d_{\boldsymbol{v}^-(\boldsymbol{f})}) + \sum_{d' \neq d}\frac{1}{2}(\Delta^{d'} u^{d'}_{\boldsymbol{v}^+(\boldsymbol{f})} + \Delta^{d'} u^{d'}_{\boldsymbol{v}^-(\boldsymbol{f})})$$
$$K_{\boldsymbol{f}} = K_0 \max(-(D\vec{u})_{\boldsymbol{f}}, 0)$$
$$F^{n+\frac{1}{2}}_{\boldsymbol{f}} = F^{n+\frac{1}{2}}_{\boldsymbol{f}} - K_{\boldsymbol{f}}(U^n_{\boldsymbol{v}^+(\boldsymbol{f})} - U^n_{\boldsymbol{v}^-(\boldsymbol{f})})$$
$$F^{\text{covered}}_{\boldsymbol{v},\pm,d} = F^{\text{covered}}_{\boldsymbol{v},\pm,d} - K_{\boldsymbol{f}}(U^n_{\boldsymbol{v}^+(\boldsymbol{f})} - U^n_{\boldsymbol{v}^-(\boldsymbol{f})})$$

We modify the covered face with the same divergence used in the adjacent uncovered face.

$$F^{\text{covered}}_{\boldsymbol{v},\pm,d} = F^{\text{covered}}_{\boldsymbol{v},\pm,d} - K_{\boldsymbol{f}}(U^n_{\boldsymbol{v}^+(\boldsymbol{f})} - U^n_{\boldsymbol{v}^-(\boldsymbol{f})})$$
$$\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{v}, \mp, d)$$

This has the effect of negating the effect of artificial viscosity on the non-conservative divergence of the flux at irregular cells. We describe later that the solid wall boundary condition at the embedded boundary is also modified with artificial viscosity.

6 Slope Calculation

We will use the 4th order slope calculation in Colella and Glaz [1] combined with characteristic limiting.

$$
\begin{aligned}
\Delta^d W_{\boldsymbol{v}} &= \zeta_{\boldsymbol{v}}\ \widetilde{\Delta}^d W_{\boldsymbol{v}} \\
\widetilde{\Delta}^d W_{\boldsymbol{v}} &= \Delta^{vL}(\Delta^B W_{\boldsymbol{v}}, \Delta^L W_{\boldsymbol{v}}, \Delta^R W_{\boldsymbol{v}}) \mid \Delta_2^d W_{\boldsymbol{v}} \mid \Delta_2^d W_{\boldsymbol{v}} \\
\Delta_2^d W_{\boldsymbol{v}} &= \Delta^{vL}(\Delta^C W_{\boldsymbol{v}}, \Delta^L W_{\boldsymbol{v}}, \Delta^R W_{\boldsymbol{v}}) \mid \Delta^{VLL} W_{\boldsymbol{v}} \mid \Delta^{VLR} W_{\boldsymbol{v}} \\
\Delta^B W_{\boldsymbol{v}} &= \frac{2}{3}((W - \frac{1}{4}\Delta_2^d W) \ll \boldsymbol{e}^d)_{\boldsymbol{v}} - ((W + \frac{1}{4}\Delta_2^d W) \ll -\boldsymbol{e}^d)_{\boldsymbol{v}}) \\
\Delta^C W_{\boldsymbol{v}} &= \frac{1}{2}((W^n \ll \boldsymbol{e}^d)_{\boldsymbol{v}} - (W^n \ll -\boldsymbol{e}^d)_{\boldsymbol{v}}) \\
\Delta^L W_{\boldsymbol{v}} &= W^n_{\boldsymbol{v}} - (W^n \ll -\boldsymbol{e}^d)_{\boldsymbol{v}} \\
\Delta^R W_{\boldsymbol{v}} &= (W^n \ll \boldsymbol{e}^d)_{\boldsymbol{v}} - W^n_{\boldsymbol{v}} \\
\Delta^{3L} W_{\boldsymbol{v}} &= \frac{1}{2}(3W^n_{\boldsymbol{v}} - 4(W^n \ll -\boldsymbol{e}^d)_{\boldsymbol{v}} + (W^n \ll -2\boldsymbol{e}^d)_{\boldsymbol{v}}) \\
\Delta^{3R} W_{\boldsymbol{v}} &= \frac{1}{2}(-3W^n_{\boldsymbol{v}} + 4(W^n \ll \boldsymbol{e}^d)_{\boldsymbol{v}} - (W^n \ll 2\boldsymbol{e}^d)_{\boldsymbol{v}}) \\
\Delta^{VLL} W_{\boldsymbol{v}} &= \begin{cases} \mathsf{min}(\Delta^{3L} W_{\boldsymbol{v}}, \Delta^L_{\boldsymbol{v}}) & \text{if } \Delta^{3L} W_{\boldsymbol{v}} \cdot \Delta^L W_{\boldsymbol{v}} > 0 \\ 0 & \text{otherwise} \end{cases} \\
\Delta^{VLR} W_{\boldsymbol{v}} &= \begin{cases} \mathsf{min}(\Delta^{3R} W_{\boldsymbol{v}}, \Delta^R_{\boldsymbol{v}}) & \text{if } \Delta^{3R} W_{\boldsymbol{v}} \cdot \Delta^R W_{\boldsymbol{v}} > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

At domain boundaries, $\Delta^L W_{\boldsymbol{v}}$ and $\Delta^R W_{\boldsymbol{v}}$ may be overwritten by the application. There are two versions of the van Leer limiter $\Delta^{vL}(\delta W_C, \delta W_L, \delta W_R)$ that are commonly used. One is to apply a limiter to the differences in characteristic variables.

1. Compute expansion of one-sided and centered differences in characteristic variables.

$$\alpha^k_L = l^k \cdot \delta W_L \tag{59}$$
$$\alpha^k_R = l^k \cdot \delta W_R \tag{60}$$
$$\alpha^k_C = l^k \cdot \delta W \tag{61}$$

2. Apply van Leer limiter

$$\alpha^k = \begin{cases} \mathsf{min}(2 \mid \alpha^k_L \mid,\ 2 \mid \alpha^k_R \mid,\ \mid \alpha^k_C \mid) & \text{if } \alpha^k_L \cdot \alpha^k_R > 0 \\ 0 & \text{otherwise} \end{cases} \tag{62}$$

3. $\Delta^{vL} = \sum_k \alpha^k r^k$

Here, $l^k = l^k(W_{\boldsymbol{i}}^n)$ and $r^k = r^k(W_{\boldsymbol{i}}^n)$.

For a variety of problems, it suffices to apply the van Leer limiter componentwise to the differences. Formally, this can be obtain from the more general case above by taking the matrices of left and right eigenvectors to be the identity.

6.1 Flattening

Finally, we give the algorithm for computing the flattening coefficient $\zeta_{\boldsymbol{i}}$. We assume that there is a quantity corresponding to the pressure in gas dynamics (denoted here as p) which can act as a steepness indicator, and a quantity corresponding to the bulk modulus (denoted here as K, given as γp in a gas), that can be used to non-dimensionalize differences in p.

$$\begin{aligned}
\zeta_{\boldsymbol{v}} &= \begin{cases} \min\limits_{0\le d<\mathbf{D}} \zeta_{\boldsymbol{v}}^d & \text{if } \sum_{d=0}^{\mathbf{D}-1} \Delta_1^d u_{\boldsymbol{v}}^d < 0 \\ 1 & \text{otherwise} \end{cases} \\
\zeta_{\boldsymbol{v}}^d &= min_3(\widetilde{\zeta}^d, d)_{\boldsymbol{v}} \\
\widetilde{\zeta}_{\boldsymbol{v}}^d &= \eta(\Delta_1^d p_{\boldsymbol{v}},\, \Delta_2^d p_{\boldsymbol{v}},\, min_3(K,d)_{\boldsymbol{v}}) \\
\Delta_1^d p_{\boldsymbol{v}} &= \Delta^C p_{\boldsymbol{v}} \mid \Delta^L p_{\boldsymbol{v}} \mid \Delta^R p_{\boldsymbol{v}} \\
\Delta_2^d p_{\boldsymbol{v}} &= (\Delta_1^d p \ll \boldsymbol{e}^d)_{\boldsymbol{v}} + (\Delta_1^d p \ll -\boldsymbol{e}^d)_{\boldsymbol{v}} \mid 2\Delta_1^d p_{\boldsymbol{v}} \mid 2\Delta_1^d p_{\boldsymbol{v}}
\end{aligned} \tag{63}$$

The functions min_3 and ζ are given below.

$$\begin{aligned}
min_3(q,d)_{\boldsymbol{v}} &= \mathrm{min}((q\ll \boldsymbol{e}^d)_{\boldsymbol{v}}, q_{\boldsymbol{v}}, (q\ll -\boldsymbol{e}^d)_{\boldsymbol{v}}) \mid \mathrm{min} q_{\boldsymbol{v}}, (q\ll -\boldsymbol{e}^d)_{\boldsymbol{v}}) \mid \mathrm{min}(q\ll \boldsymbol{e}^d)_{\boldsymbol{v}}, q_{\boldsymbol{v}}) \\
\zeta(\delta p_1, \delta p_2, p_0) &= \begin{cases} 0 & \text{if } \frac{|\delta p_1|}{p_0} > d \text{ and } \frac{|\delta p_1|}{|\delta p_2|} > r_1 \\ 1 - \frac{\frac{|\delta p_1|}{|\delta p_2|} - r_0}{r_1 - r_0} & \text{if } \frac{|\delta p_1|}{p_0} > d \text{ and } r_1 \ge \frac{|\delta p_1|}{|\delta p_2|} > r_0 \\ 1 & \text{otherwise} \end{cases} \\
& r_0 = 0.75,\ r_1 = 0.85,\ d = 0.33
\end{aligned} \tag{64}$$

Note that min_3 is not the minimum over all available VoFs but involves the minimum of shifted VoFs which includes an averaging operation.

7 Computing fluxes at the irregular boundary

The flux at the embedded boundary is centered at the centroid of the boundary $\bar{\mathbf{x}}$. We extrapolate the primitive solution in space from the cell center. We then transform to the conservative solution and extrapolate in time using the stable, non-conservative estimate

of the flux divergence described in equation 14.

$$W_{\boldsymbol{v},B} = W^n_{\boldsymbol{v}} + \sum_{d=0}^{D-1}(\bar{x}_d \Delta^d W^n_{\boldsymbol{v}} - \frac{\Delta t}{2\Delta x} A^d \Delta^d W^n_{\boldsymbol{v}}) \tag{65}$$

$$F^{n+\frac{1}{2}}_{\boldsymbol{v},B} = R_B(U^{n+\frac{1}{2}}_{\boldsymbol{v},B}, \boldsymbol{n}^B_{\boldsymbol{v}}) \tag{66}$$

For polytropic gas dynamics, this becomes

$$\begin{aligned}
\rho_{\boldsymbol{v},B} =& \rho^n_{\boldsymbol{v}} + \sum_{d=0}^{D-1}(\bar{x}_d \Delta^d \rho^n_{\boldsymbol{v}} - \frac{\Delta t}{2\Delta x}(u^d \Delta^d \rho + \rho \Delta^d u^d)^n_{\boldsymbol{v}}) \\
p_{\boldsymbol{v},B} =& p^n_{\boldsymbol{v}} + \sum_{d=0}^{D-1}(\bar{x}_d \Delta^d p^n_{\boldsymbol{v}} - \frac{\Delta t}{2\Delta x}(u^d \Delta^d p + \rho c^2 \Delta^d u^d)^n_{\boldsymbol{v}}) \\
u^{d1}_{\boldsymbol{v},B} =& (u^{d1})^n_{\boldsymbol{v}} + \sum_{d=0}^{D-1}(\bar{x}_d \Delta^d (u^{d1})^n_{\boldsymbol{v}}) \\
& - \frac{\Delta t}{2\Delta x}(u^{d1} \Delta^{d1} u^{d1} + \frac{1}{\rho}\Delta^{d1} p)^n_{\boldsymbol{v}} \\
& - \frac{\Delta t}{2\Delta x}(\sum_{d2 \neq d1}(u^{d2} \Delta^{d2} u^{d1}))
\end{aligned} \tag{67}$$

If we are using solid-wall boundary conditions at the irregular boundary, we calculate an approximation of the divergence of the velocity at the irregular cell $D(\vec{u})_{\boldsymbol{v}}$ and use it to modify the flux to be consistent with artificial viscosity. The d-direction momentum flux at the irregular boundary is given by $-p^r n^d$ where p^r is the pressure to emerge from the Riemann solution in equation 67. For artificial viscosity, we modify this flux as follows.

$$\begin{aligned}
(D\vec{u})_{\boldsymbol{v}} &= \sum_{d'=0}^{D-1} \Delta^{d'} u^{d'}_{\boldsymbol{v}} \\
p^r &= p^r - 2K_0 \ \mathsf{max}(-(D\vec{u})_{\boldsymbol{v}}, 0)\vec{u} \cdot \hat{n}
\end{aligned}$$

8 Results

We run the Modiano problem for one time step to compute the truncation error of the operator. The error at a given level of refinement E^h is approximated by

$$E^{\mathsf{trunc}} = \frac{U^h(t) - U^e(t)}{t} \tag{68}$$

where $U^h(t)$ is the discrete solution and $U^e(t)$ is the exact solution at time $t = \Delta t$. We run the Modiano problem for a fixed time to compute the solution error of the operator.

The error at a given level of refinement E^h is approximated by

$$E^{\text{soln}} = U^h(t) - U^e(t) \tag{69}$$

where $U^h(t)$ is the discrete solution and $U^e(t)$ is the exact solution at time t. The order of convergence p is given by

$$p = \frac{\log(\frac{|E^{2h}|}{|E^h|})}{\log(2)} \tag{70}$$

9 Class Hierarchy

The principal EBAMRGodunov classes follow.

- `EBAMRGodunov`, the `AMRLevel`-derived class which is driven by the `AMR` class.
- `EBLevelGodunov`, a class owned by `AMRGodunov`. `EBLevelGodunov` advances the solution on a level and can exist outside the context of an AMR hierarchy. This class makes possible Richardson extrapolation for error estimation.
- EBPatchGodunov, is a base class which encapsulates the operations required to advance a solution on a single patch.
- `EBPhysIBC` is a base class which encapsulates initial conditions and flux-based boundary conditions.

9.1 Class `EBAMRGodunov`

`EBAMRGodunov` is the `AMRLevel`-derived class with which the `AMR` class will directly interact. Its user interface is therefore constrained by the `AMRLevel` interface. The important data members of the `EBAMRGodunov` class are as follows.

- `LevelData<EBCellFAB> m_state_old, m_state_new;`

  The state data at old and new times. Both need to be kept because subcycling in time requires temporal interpolation.

- `Real m_cfl, m_dx;`

  CFL number and grid spacing for this level.

- `EBPWLFineInterp m_fine_interp;`

  Interpolation operator for refining data during regridding that were previously only covered by coarser data.

Variable	Coarse Error	Fine Error	Order
mass-density	3.127796e-05	1.669137e-05	9.060445e-01
x-momentum	3.292329e-05	1.675957e-05	9.741235e-01
y-momentum	6.766401e-05	3.141857e-05	1.106771e+00
energy-density	1.094807e-04	5.842373e-05	9.060502e-01

Table 1: Truncation error convergence rates using L-0 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$

Variable	Coarse Error	Fine Error	Order
mass-density	7.358933e-08	1.616991e-08	2.186185e+00
x-momentum	7.569344e-08	2.010648e-08	1.912508e+00
y-momentum	1.764416e-07	4.648945e-08	1.924216e+00
energy-density	2.575709e-07	5.659651e-08	2.186185e+00

Table 2: Truncation error convergence rates using L-1 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$

Variable	Coarse Error	Fine Error	Order
mass-density	4.010155e-07	1.164273e-07	1.784228e+00
x-momentum	6.057493e-07	2.402295e-07	1.334308e+00
y-momentum	1.717569e-06	5.992271e-07	1.519193e+00
energy-density	1.403616e-06	4.075112e-07	1.784237e+00

Table 3: Truncation error convergence rates using L-2 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$

Variable	Coarse Error	Fine Error	Order
mass-density	3.769203e-07	7.212809e-08	2.385626e+00
x-momentum	3.427140e-07	7.681266e-08	2.157589e+00
y-momentum	7.501614e-07	1.692840e-07	2.147755e+00
energy-density	1.319233e-06	2.524508e-07	2.385625e+00

Table 4: Solution error convergence rates using L-0 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$

- `EBCoarseAverage m_coarse_average;`

  This is the averaging operator which replaces data on coarser levels with the average of the data on this level where they coincide in space.

- `RefCountedPtr<EBPhysIBC> m_phys_ibc_ptr;`

  This boundary condition operator provides flux-based boundary data at domain boundaries and also provides initial conditions.

The `EBAMRGodunov` implementation of the `AMRLevel` currently does the following for each of the important interface functions.

- `Real EBAMRGodunov::advance()`

  This function advances the conservative state by one time step. It calls the `EBLevelGodunov::step` function. The timestep returned by that function is stored in member data.

- `void EBAMRGodunov::postTimeStep()`

  This function calls refluxing from the next finer level and averages its solution to the next finer level.

- `void regrid(const Vector<Box>& a_new_grids)`

  This function changes the union of rectangles over which the data is defined. At places where the two sets of rectangles intersect, the data is copied from the previous set of rectangles. At places where there was only data from the next coarser level, piecewise linear interpolation is used to fill the data.

- `void initialData()`

  In this function the initial state is filled by calling `m_phys_ibc_ptr->initialize`.

- `void computeDt()`

  This function returns the timestep stored during the `advance()` call.

- `void computeInitialDt()`

  This function calculates the time step using the maximum wavespeed returned by a `EBLevelGodunov::getMaxWaveSpeed` call. Define the maximum wavespeed to be w and the initial timestep multiplier to be K and the grid spacing at this level to be h,

$$\Delta t = \frac{Kh}{w}. \tag{71}$$

- `DisjointBoxLayout loadBalance(const Vector<Box>& a_grids)`

  Calls the Chombo load balancer to create the returned layout.

9.2 Class EBLevelGodunov

EBLevelGodunov is a class owned by AMRGodunov. EBLevelGodunov advances the solution on a level and can exist outside the context of an AMR hierarchy. This class makes possible Richardson extrapolation for error estimation. The important functions of the public interface of EBLevelGodunov follow.

- ```
  void define(const DisjointBoxLayout& a_thisDBL,
              const DisjointBoxLayout& a_coarDBL,
              const EBISLayout&        a_thisEBISL,
              const EBISLayout&        a_coarEBISL,
              const RedistSTencil&     a_redStencil,
              const Box&               a_DProblem,
              const int&               a_numGhost,
              const int&               a_nRefine,
              const Real&              a_dx,
              const EBPatchGodunov* const a_integrator,
              const bool& a_hasCoarser,
              const bool& a_hasFiner);
  ```

  Define the internal data structures. For the coarsest level, an empty DisjointBoxLayout is passed in for coarserDisjointBoxLayout.

  - a_thisDBL, a_coarDBL The layouts at this level and the next coarser level. For the coarsest level, an empty DisjointBoxLayout is passed in for coarDBL.
  - a_DProblem, a_dx The problem domain and grid spacing at this level.
  - a_nRefine The refinement ratio between this level and the next coarser level.
  - a_numGhost The number of ghost cells (assumed to be isotropic) required to advance the solution.
  - a_bc Boundary conditions and initial conditions are encapsulated in this object.

- ```
  Real step(LevelData<EBCellFAB>&        a_U,
            LevelData<BaseIVFAB<Real> >& a_massDiff,
            EBFluxRegister&              a_coarFluxRegister,
            EBFluxRegister&              a_fineFluxRegister
            const LevelData<EBCellFAB>&  a_UCoarseOld,
            const LevelData<EBCellFAB>&  a_UCoarseNew,
            const Real&                  a_time,
            const Real&                  a_TCold,
            const Real&                  a_TCNew,
            const Real&                  a_dt);
  ```

  Advance the solution at this timeStep for one time step.

  - a_UCoarseOld, a_UCoarseNew The solution at the next coarser level at the old and new coarse times.

- a_time, a_TCold, a_TCNew The time of this solution (before the advance) and the old and new coarse solution times.
- a_dt The time step at this level.
- a_U The solution at this level.
- a_massDiff Redistribution mass.
- a_coarFluxRegister, a_fineFluxRegisters The flux registers between this level and the adjacent levels.

- `Real getMaxWaveSpeed(const LevelData<EBCellFAB>& a_state);`

  Return the maximum wave speed of input a_state for purposes of limiting the time step.

9.3 Class EBPatchGodunov

The base class EBPatchGodunov provides a skeleton for the application-dependent pieces of a second-order unsplit Godunov method. The virtual functions are called by EBLevelGodunov, which manages the overall assembly of the second-order unsplit fluxes. As part of EBPatchGodunov, we provide some member functions (slope, flattening), that we expect to be useful across applications, but require either virtual functions or parameter information by the user.

There are three types of grid variables that appear in the unsplit Godunov method in section (**??**): conserved quantities, primitive variables, and fluxes, denoted below by U, q, F, respectively. It is often convenient to have the number of components for primitive variables and for fluxes exceed that for conserved quantities. In the case of primitive variables, redundant quantities are carried that parameterize the equation of state in order to avoid multiple calls to that function. In the case of fluxes, it is often convenient to split the flux for some variables into multiple components, e.g., dividing the momentum flux into advective and pressure terms. The API given here provides the flexibility to support these various options.
Construction Methods:

- `void setPhysIBC(RefCountedPtr<EBPhysIBC> a_bc)`

  Set the boundary condition pointer of the integrator.

- ```
  virtual void define(
              const Box& a_domain,
              const Real& a_dx);
  ```

  Set the domain variables for this level.

- `virtual EBPatchGodunov* new_patchGodunov = 0;`

  Factory method. Return pointer to new PatchGodunov object with its boundary conditions defined.
  ```

EBLevelGodunov API: (Translation: these are the only things that actually get called by EBLevelGodunov.

- ```
  virtual void
  regularUpdate(EBCellFAB&       a_consState,
                EBFluxFAB&       a_flux,
                BaseIVFAB<Real>& a_nonConservativeDivergence,
                const EBCellFAB& a_source,
                const Box&       a_box);
  ```

  Update the state using flux difference that ignores EB. Store fluxes used in this update Store non-conservative divergence. Flux coming out of this this should exist at cell face centers.

- ```
  interpolateFluxToCentroids(BaseIFFAB<Real>              a_centroidFlux[SpaceDim],
                             const BaseIFFAB<Real>* const a_fluxInterpolant[SpaceDim],
                             const IntVectSet&            a_irregIVS);
  ```

  Interpolates cell-face centered fluxes to centroids over irregular cells. Flux going into this should exist at cell face centers.

- ```
  virtual void
  irregularUpdate(EBCellFAB&             a_consState,
                  Real&                  a_maxWaveSpeed,
                  BaseIVFAB<Real>&       a_massDiff,
                  const BaseIFFAB<Real>  a_centroidFlux[SpaceDim],
                  const BaseIVFAB<Real>& a_nonConservativeDivergence,
                  const Box&             a_box,
                  const IntVectSet&      a_ivs);
  ```

  Update the state at irregular VoFs and compute mass difference and the maximum wave speed over the entire box. Flux going into this should exist at VoF centroids.

- ```
  virtual Real getMaxWaveSpeed(
                const EBCellFAB& a_U,
                const Box& a_box)= 0;
  ```

  Return the maximum wave speed on over this patch.

- ```
  void setValidBox(const Box& a_validBox,
                   const EBISBox& a_ebisbox,
                   const Real& a_time,
                   const Real& a_dt);
  ```

  Set the valid box of the patch.

Virtual interface:
  ```

- ```
  virtual void consToPrim(EBCellFAB&       a_primState,
                          const EBCellFAB& a_conState) = 0;
  ```

  Compute the primitive state given the conserved state. $W_i = W(U_i)$.

- ```
  virtual void incrementWithSource(
                              EBCellFAB&       a_primState,
                              const EBCellFAB& a_source,
                              const Real& a_scale,
                              const Box&  a_box) = 0;
  ```

  Increment the primitive variables by the source term, as in (34). `a_scale = 0.5*dt`.

- ```
  virtual void normalPred(EBCellFAB&       a_qlo,
                          EBCellFAB&       a_qhi,
                          const EBCellFAB& a_q,
                          const EBCellFAB& a_dq,
                          const Real&      a_scale,
                          const int&       a_dir,
                          const Box&       a_box) = 0;
  ```

  Extrapolate in the low and high direction from q, as in (34). A default implementation is provided which assumes the existence of the virtual functions `limit`.

- ```
  virtual void riemann(EBFaceFAB&       a_flux,
                       const EBCellFAB& a_qleft,
                       const EBCellFAB& a_qright,
                       const int&        a_dir,
                       const Box&       a_box) = 0;
  virtual void riemann(BaseIVFAB<Real>&       a_coveredFlux,
                       const BaseIVFAB<Real>& a_extendedState,
                       const EBCellFAB&       a_primState,
                       const IntVecSet&       a_coveredFace,
                       const int&             a_dir,
                       const Side::LoHiSide&  a_sd) = 0;
  ```

  Given input left and right states, compute a suitably-upwinded flux (e.g. by solving a Riemann problem), as in equation 41.

- ```
  virtual void updateCons(EBCellFAB&             a_conState,
                          const EBFaceFAB&       a_flux,
                          const BaseIVFAB<Real>& a_coveredFluxMinu,
                          const BaseIVFAB<Real>& a_coveredFluxPlus,
                          const IntVecSet&       a_coveredFaceMinu,
                          const IntVecSet&       a_coveredFacePlus,
                          const int&             a_dir,
                          const Box&             a_box,
                          const Real&            a_scale) = 0;
  ```
  ```

Given the value of the flux, update the conserved quantities and modify in place the flux for the purpose of passing it to a EBFluxRegister.

```
consstate_i +=a_scale*(flux_i-1/2 - flux_i+1/2)
```

.

- ```
  virtual void updatePrim(EBCellFAB&             a_qminus,
                          EBCellFAB&             a_qplus,
                          const EBFaceFAB&       a_flux,
                          const BaseIVFAB<Real>& a_coveredFluxMinu,
                          const BaseIVFAB<Real>& a_coveredFluxPlus,
                          const IntVecSet&       a_coveredFaceMinu,
                          const IntVecSet&       a_coveredFacePlus,
                          const int&             a_dir,
                          const Box&             a_box,
                          const Real&            a_scale) = 0;
  ```

  Given a_flux, the value of the flux in the direction a_dir, update q_plus, q_minus, the extrapolated primitive quantities, as in (**??**,29,30).

  ```
  primstate_i += a_scale*Grad_W U(flux_i-1/2 - flux_i+1/2)
  ```

- ```
  virtual void applyLimiter(EBCellFAB&       a_dq,
                            const EBCellFAB& a_dql,
                            const EBCellFAB& a_dqr,
                            const int&       a_dir,
                            const Box&       a_box) = 0;
  ```

  Given left and right one-sided undivided differences a_dql,a_dqr, apply van Leer limiter vL defined in section (6) to a_dq. Called by the default implementation of EBPatchGodunov::slope.

- ```
  virtual int numPrimitives() const = 0;
  ```

  Returns number of components for primitive variables.

- ```
  virtual int numFluxes() const = 0;
  ```

  Returns number of components for flux variables.

- ```
  virtual int numConserved() const = 0;
  ```

  Returns number of components for conserved variables.

- ```
  virtual Interval velocityInterval() const = 0;
  ```

  Returns the interval of component indices in the primitive variable EBCellFAB for the velocities.

- `virtual int pressureIndex() const = 0;`

  Returns the component index for the pressure. Called only if flattening is used.

- `virtual int bulkModulusIndex() const = 0;`

  Returns the component index for the bulk modulus, used as a normalization to measure shock strength in flattening. Called only if flattening is used.

- `virtual Real artificialViscosityCoefficient() const = 0;`

  Returns value of artificial viscosity. Called only if artificial viscosity is being used.

Useful member functions:

- ```
  void slope(EBCellFAB&       a_dq,
             const EBCellFAB& a_q,
             const EBCellFAB& a_flattening,
             int              a_dir,
             const Box&       a_box) const;
  ```

  Compute the limited slope `a_dq` of the primitive variables `a_q` for the components in the interval `a_interval`, using the algorithm described in (6). Calls user-supplied `EBPatchGodunov::applyLimiter`.

- `void getFlattening(const EBCellFAB& a_q);`

  Computes the flattening coefficient (63) and stores it in the member data `m_flatcoef`. Called from `EBPatchGodunov::slope`, if required.

9.4 Class `EBPhysIBC`

`EBPhysIBC` is an interface class owned and used by PatchGodunov through which a user specifies the initial and boundary of conditions of her particular problem. These boundary conditions are flux-based. `EBPhysIBC` contains as member data the mesh spacing (`Real a_dx`) and the domain of computation (`ProblemDomain a_domain`). The important user functions of `EBPhysIBC` are as follows.

- ```
  virtual void define(const Box&  a_domain
                      const Real& a_dx) = 0;
  ```

  Define the internals of the class.

- `virtual EBPhysIBC* new_ebphysIBC() = 0;`

  Factory method. Return a new `EBPhysIBC` object.

- ```
  virtual void fluxBC(EBFaceFAB& a_flux,
                      const EBCellFAB& a_Wextrap,
                      const EBCellFAB& a_Wcenter,
  ```
  ```

```
                    const int& a_dir,
                    const Side::LoHiSide& a_side,
                    const Real& a_time) = 0;
```

Enforce the flux boundary condition on the boundary of the domain and place the result in `a_flux`. The arguments to this function are as follows

- `a_flux` is the array of the fluxes over the box. This values in the array that correspond to the boundary faces of the domain are to be replaced with boundary values.
- `a_Wextrap` is the extrapolated value of the state's primitive variables. This data is cell-centered.
- `a_Wcenter` is the cell-centered value of the primitive variables at the start of the time step. This data is cell-centered.
- `a_dir, a_side` is the direction normal and the side of the domain where the function will be enforcing boundary conditions.
- `a_time` is the time at which boundary conditions will be imposed.

- `virtual void initialize(LevelData<FArrayBox>& a_conState);`

  Fill the input with the initial conserved variable data of the problem.

- ```
  void
  setBndrySlopes(EBCellFAB&       a_deltaPrim,
                 const EBCellFAB& a_primState,
                 const int&       a_dir)
  ```

  Set the slopes at domain boundaries as described in section 6.

References

[1] P. Colella and H. M. Glaz. Efficient solution algorithms for the Riemann problem for real gases. *J. Comput. Phys.*, 59:264, 1985.

[2] 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. Van Straalen. Chombo Software Package for AMR Applications - Design Document. unpublished, 2000.

[3] Phillip Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comput. Phys.*, 87:171–200, 1990.

[4] R. Courant and K. O. Friedrichs. *Supersonic Flow and Shock Waves*. NYU, New York, NY, 1948.

[5] Jeff Saltzman. An unsplit 3d upwind method for hyperbolic conservation laws. *J. Comput. Phys.*, 115:153–168, 1994.
  ```

Variable	Coarse Error	Fine Error	Order
mass-density	1.103779e-09	1.855826e-10	2.572317e+00
x-momentum	1.125935e-09	2.356203e-10	2.256588e+00
y-momentum	1.617258e-09	2.371548e-10	2.769649e+00
energy-density	3.863314e-09	6.495531e-10	2.572320e+00

Table 5: Solution error convergence rates using L-1 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$

Variable	Coarse Error	Fine Error	Order
mass-density	5.553216e-09	1.114919e-09	2.316385e+00
x-momentum	6.038922e-09	1.251264e-09	2.270905e+00
y-momentum	9.515687e-09	2.244841e-09	2.083695e+00
energy-density	1.943688e-08	3.902358e-09	2.316379e+00

Table 6: Solution error convergence rates using L-2 norm. $h_f = \frac{1}{512}$ and $h_c = 2h_f$, $D = 2$