FLASH UHD Documentation

Christopher C. Lindner, Miloš Milosavljević,

Department of Astronomy, University of Texas, 1 University Station C1400, Austin, TX 78712

ABSTRACT

The purpose of this documentation is to document the FLASH unsplit hydrodynamics solver and the changes needed to add radiation hydrodynamics to this solver.

1. Introduction

The latest version of the FLASH code is available in my directory
`/data1/r900-1/lindner/FLASH4.2.1/source/`

Most of the files relevant to the unsplit solver are in
`/data1/r900-1/lindner/FLASH4.2.1/source/physics/Hydro/HydroMain/unsplit/Hydro_Unspl`

Any file or directory names I mention will be relative to this directory.

Items highlighted in red indicate that they are going to be replaced by our new changes, indicated in blue.

Section 2 describes the basic equations of radiation hydrodynamics that FLASH is trying to solve. Section 3 contains a basic overview flowchart of how the solver works. There are some brief descriptions of select files as well.

2. FLASH Multispecies Hydrodynamics

The basic equations of hydrodynamics solved by the FLASH multispecies solver are mass conservation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{1}$$

momentum conservation

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla P_{\rm tot} = 0, \tag{2}$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla P_{\rm gas} + \frac{\lambda}{3} \nabla P_{\rm r} = 0, \tag{3}$$

total energy conservation,

$$\frac{\partial E_{\rm tot}}{\partial t} + \nabla \cdot \left[\left(\rho E_{\rm tot} + P_{\rm tot} \right) \mathbf{v} \right] = 0, \tag{4}$$

$$\frac{\partial E_{\rm gas+kin}}{\partial t} + \nabla \cdot \left[\left(\rho E_{\rm gas+kin} + P_{\rm gas} \right) \mathbf{v} \right] + \lambda v \cdot E_{\rm r} = 0, \tag{5}$$

$$\frac{\partial E_{\rm r}}{\partial t} + \nabla \cdot \left(\frac{3-f}{2} E_{\rm r} v \right) - \lambda v \cdot E_{\rm r} = 0, \tag{6}$$

and the individual equations for internal energy advection. In the three temperature, radiation hydrodynamics unit of FLASH, energy advection is split into three equations, one for each species

$$\frac{\partial \rho \epsilon_{\rm ele}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm ele} \mathbf{v} \right] + P_{\rm ele} \nabla \cdot \mathbf{v} = 0, \tag{7}$$

$$\frac{\partial \rho \epsilon_{\rm ion}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm ion} \mathbf{v} \right] + P_{\rm ion} \nabla \cdot \mathbf{v} = 0, \tag{8}$$

$$\frac{\partial \rho \epsilon_{\rm rad}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm rad} \mathbf{v} \right] + P_{\rm rad} \nabla \cdot \mathbf{v} = 0. \tag{9}$$

Note that we do not include terms involving radiation diffusion or radiation-matter coupling via opacity here. These terms are present in FLASH, but carried out in an operator-split fashion from the hydrodynamics, and solved in a different unit.

For the "RAGE-like" approach, FLASH first considers the three split energy advection equations without the work terms

$$\frac{\partial \rho \epsilon_{\rm ele}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm ele} \mathbf{v} \right] = 0, \tag{10}$$

$$\frac{\partial \rho \epsilon_{\rm ion}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm ion} \mathbf{v} \right] = 0, \tag{11}$$

$$\frac{\partial \rho \epsilon_{\rm rad}}{\partial t} + \nabla \cdot \left[\rho \epsilon_{\rm rad} \mathbf{v} \right] = 0. \tag{12}$$

FLASH then calculates the changes in total energy due to hydrodynamic work and shock heating and distributes this energy among the species according to the ratios of pressures in a "RAGE-like" approach (**?**). This is carried out in four steps:

1. Equations 1, 2, 5, 10, 11, and 12 are evolved simultaneously and internal energies are updated to reflect changes due to advection.

2. The change in total specific internal energy is calculated via $\Delta \epsilon_{\rm tot} = \Delta E_{\rm tot} - \mathbf{v} \cdot \mathbf{v}/2$.

3. The change in specific internal energy due to hydrodynamic work and shock heating is calculated via $\Delta \epsilon_{\rm tot}^{\rm work} + \Delta \epsilon_{\rm tot}^{\rm shock} = \Delta \epsilon_{\rm tot} - \Delta \epsilon_{\rm tot}^{\rm adv}$, where $\Delta \epsilon_{\rm tot}^{\rm adv}$ is the change in specific internal energy solely due to advection.

4. The energy $\Delta \epsilon_{\rm tot}^{\rm work} + \Delta \epsilon_{\rm tot}^{\rm shock}$ is distributed evenly among all three species according to the ratio of their pressures.

This method ensures that energy is distributed among the species correctly in smooth flows, but may fail to do so near shocks[1]. By advecting each species separately, we are able to maintain separate temperatures for the gas and radiation. CASTRO seems to avoid this issue entirely by computing radiation and gas fluxes instead of the total fluxes. The formulation of their equations do not include the problematic $\nabla \cdot \mathbf{v}$ terms.

The equations above do not account for energy gains or losses from radiative diffusion, radiation-matter coupling, gravity, or artificial heating source terms. These components are carried out operator-split from the equations above.

If we include opacity effects in the low optical depth limits, the complete momentum equation should be

$$\partial_t(\rho\mathbf{v}) + \nabla\cdot(\rho\mathbf{v}\mathbf{v}) + \nabla P_{\rm tot} - \frac{\lambda}{3}\nabla P_{\rm r} = 0, \tag{13}$$

where $P_{\rm m} = P_{\rm e} + P_{\rm i}$ is the matter (electron + ion) pressure, λ is the flux limiter, and $P_{\rm r}$ is the radiation pressure. Note that we are working in the non-relativistic limit and ignoring terms of $\mathcal{O}(\frac{v}{C})$. We also do not implement a variable Eddington factor, a key difference from Z11. The flux-limited diffusion coefficient used in flash is

$$D = \frac{c}{3\sigma_{tot} + \frac{|\nabla u_r|}{\alpha_r u_r}}, \tag{14}$$

such that $D \approx \frac{c}{3\sigma_{tot}}$ for optically thick regions and $D \approx \frac{\alpha_{\rm r} c u_{\rm r}}{|\nabla u_{\rm r}|}$ such that $\nabla \cdot D\nabla u_r = \nabla \alpha_{\rm r} c u_{\rm r}$ for optically thin regions ($\alpha_{\rm r}$ is a dimensionless coefficient, set to unity in our simulations), σ_{tot} is the total opacity, and c is the speed of light.

3. General Outline

Hydro Main FLASH hydro driver that calls the unsplit solver.

hy_uhd_unsplit This basically runs through the entire unsplit solver. This function seems pretty general and I think few, if any changes are needed here. Instead, changes need to be made in the functions it calls. It calls the following functions (`hy_uhd_` prefixes have been omitted):

putGravityUnsplit Calls `Gravity_accelOneRow`. Fills `grav` arrays. No changes needed here.

getRiemannState Calculates and stores Riemann state values at cell interfaces so we can use these to compute fluxes, see MC 4.2.3.

- Computes the divergence of the velocity field.
- Prepares slope flattening variables that later get used later in PPM: MC 4.2.2 . These currently involve $P_{\rm tot}$. Need to replace with $P_{\rm mod} = P_{\rm g} + \lambda E_{\rm r}$ or do some sort of average like CASTRO?. See Section 5 for more info.

[1]For further justification and explanation of radiation hydrodynamics unit in FLASH, consult the FLASH Users Guide version 4 available at http://flash.uchicago.edu/.

- Prepares the vector of variables and calls **dataReconstOneStep**, which evolves cell-centered values by $\Delta t/2$ at cell interfaces using PPM characteristic tracing. Most of this is carried out in the function below.
  - **DataReconstructNormalDir_PPM** The driver of PPM and characteristic tracing. The steps described are as follows:
    1. Calculate $\boldsymbol{\lambda}_0$, which is the vector of characteristic speeds. See Section 4 for the changes to this.
    2. Estimates the slope of the primitives and some additional variables at the interfaces by calling `hy_uhd_TVDslope` (see below). Projects $\bar{\Delta}$ (AKA `delbar`, δq), the estimated slope at each interface, to primitives (?).
    3. Performs PPM Polynomial interpolation. Carries out $q_{i+\frac{1}{2}} = \frac{1}{2}(q_{i+1} + q_i) - \frac{1}{6}(\delta q_{i+1} - \delta q_i)$, which is equation 66 in MC. Note that they mention CW equation 1.9, which is equivalent to MC equation 67. I don't see this done anywhere, however. I think this might be for some sort of special case or higher order estimate or for use when you have a simpler version of δq, but I'm really not sure.
    4. Interpolates species and mass scalars (which we don't use).
    5. Carries out steepening (which is turned off in the setup we use).
    6. Flattening. The flattening coefficent is actually set up in `hy_uhd_getRiemannState` and carried into this function. Flattening is described in Equations 74-80 in MC. This is applied to the species first, then to `vecL` and `vecR`, which are the left and right interface values, $q_{i-\frac{1}{2}}$ and $q_{i+\frac{1}{2}}$ in MC notation. MC equations 70a and 70b are carried out here. See Section 5 for more info.
    7. Monotonicity Check. This is equivalent to equations 69a, 69b, and 69c of MC.
    8. "Take initial guesses for the left and right states." This calculates Δq_i (`delW`) and q_{6i} (`W6`, note the typo in MC; there should not be a Δ here) as defined in MC Equations 72b and 72c. I describe these matrices a bit more in Section 4 below. There are some additional computations relating to the `Wp` and `Wm` arrays in the code. However, it turns out that all of these changes are overwritten later on, so these are apparently vestigial and not used. Everything in the sections labeled 7a, 7b, 7c, and 7d in FLASH can be ignored. The Δq_i and q_{6i} arrays need to be updated with our new variables. See 4 for details.
    9. "Advance the above interpolated interface values by $\frac{1}{2}$ time step using characteristic Tracing." I have more detailed notes on what happens during this, but it's similar to MC 4.2.3 . See Section 4.
    10. Finalizes the calculation of Riemann states. This is described in Section 4, but it is unclear to me how these equations relate to what is done in MC. Regardless, we end with what should be the value of all of the primitives at each of the faces a half timestep advanced.

    These functions are called during this:

  - ∗ **eigenParameters** Calculates several parameters needed for the wave speed eigenvalue calculations. These need to be replaced with the appropriate adjusted parameters for the eigenvalues. See Section 4.
  - ∗ **eigenValue** Calculates the eigenvalues (wave speeds) that fill the eigenvectors. Fills λ_0 (see Section 4). These need to be replaced with the appropriate adjusted values. Some of these fixeswill take place in `eigenValue`. See Section 4.
  - ∗ **eigenVector** Fills the **L** and **R** eigenvectors in primative or conservative form (see Section 4). These need to be replaced with the appropriate for the eigenvectors. See Section 4. Note that `cons` is hard-wired to be `false` for all relevant calls.
  - ∗ **upwindTransverseFlux** "This routine advances species by locally an Eulerian algorithm in an unsplit way." (sic) ??? See Section 3 of MC, especially Section 3.2. This computes `sig`, the "transverse flux vector."
  - ∗ **TVDslope** "This routine calculates limited slopes depending on the user's choice of the slope limiter." We use a `minmod` slope limiter. `delbar` (the vector $\bar{\Delta}$) is returned, which contains the limited slopes for primitive variables + `gamc`, `game`, and gravity. It is similar to, but not the same as some of the slope limiters described on pages 46 and 47 of MC.

- Applies geometric terms. Not needed for Cartesian.
- Updates Gravity. It looks like they're just adding in the last term of MC 53.
- Stores scratch terms for each direction.
- Applies transverse correction terms for 3D. `hy_use3dFullCTU` is turned on in our code. The bulk of this seems to be handled by the next function call below.
- **upwindTransverseFlux** "This routine advances species by locally an Eulerian algorithm in an unsplit way." (sic) ??? This is also called above, inside the PPM solver. I don't understand why this is called in two separate places here.

getFaceFlux Computes fluxes at cell faces. Also calls specific solvers (e.g. **HLL**). Calculates min timestep and alters fluxes for artificial viscosity. There are some calculations related to `updateInternalEnergy` that need to be adjusted here. Some fluxes are calculated here, not just in `prim2flx` below. Advects species interenal energies: $F_{\rm erad} = F_\rho \epsilon_{\rm rad}$ etc. for radiation, ion, and electron internal energies.

- **HLL** This is the most stable solver we've been using. It's in charge of computing the high-order Godunov fluxes based on the L and R Riemann states.
  1. Calculates sound speed: $c^2 = \frac{\gamma P}{\rho}$ Replace with $c_{\rm m}$???
  2. **prim2con** Calculates conversions from primitive to conserved variables. The conserved variables are $\rho, \rho\vec{v}$, $E_{\rm tot}$ $E_{\rm kin+gas}$, $E_{\rm rad}$
  3. **prim2flx** Converts from variables to fluxes. $F = \rho v$, $F = \rho v v + P$, and $F = v(E+P)$ We need to decide which fluxes to place here. Most likely: replace total energy flux with

radiation energy and gas energy fluxes. $F = \rho v$, $F = \rho v v + p_{\rm gas}$, $F = E_{\rm gas+kin} v + p_{\rm gas} v$, $F = \frac{(3-f)}{2} E_{\rm r} v$

4. Left and right state logic. Calculates $F^* = [S_R F_L - S_L F_R + S_R S_L (U_R - U_L)] / (S_R - S_L)$ in the case of rarefaction waves...? This appears to be the standard equation for fluxes calculated in HLL (see HLL reference).

unsplitUpdate Given the fluxes, updates the conserved, cell-centered quantities. This is responsible for getting all the variables and geometric terms in the right form so it can call two subfunctions that are present in this file.

- **updateConservedVariable** Computes $U = U - \frac{\Delta t}{\Delta x}(F_R - F_L)$. See equations 53 and 63a in MC?
- **updateInternalEnergy** When flag `hy_useAuxEintEqn` is set, computes

$$\rho\epsilon = \rho\epsilon + \frac{\Delta t}{\Delta x}\left(eF_L - eF_R + P(F_{P,L} - F_{P,R})\right). \tag{15}$$

  This is turned on by default in our simulations. It seems like this should be modified.

multiTemp/unsplitUpdateMultiTemp Updates energies for the special 3T conditions. See Section 2 and the FLASH manual for more information on the "Rage-like" approach used here. Some of the weird uhd crashes I've seen occur here. There are comments that explicitly state these crashes occur for "unknown reasons."

- **hy_uhd_ragelike** FLASH then calculates the changes in total energy due to hydrodynamic work and shock heating and distributes this energy among the species according to the ratios of pressures in a "RAGE-like" approach. See Section 2. Calls **hy_uhd_getPressure** and **Hydro_recalibrateEints**. We may not need this at all if we flux radiation and gas energy instead of total energy. See Section 2.

energyFix Corrects energy in cases where the total energy is advected and `eint` can become negative. In the three temperature case, this step is actually handled by **unsplitUpdateMultiTemp**.

Grid_putFluxData "Put the fluxes in a direction specified by axis for boundary cells for block blockID. This routine needs to be used with adaptive mesh since fluxes calculated by the two blocks that are at fine-coarse boundary have different accuracy. The fluxes calculated by individual blocks are reported to the Grid through this call. Once that is done, a call to `Grid_conserveFluxes` applies the flux conservation algorithm to make it consistent across the fine-coarse boundaries." Basically it takes an array of fluxes for all variables and stores them in a more general fashion. It also does a special adjustment for fluxes that "scale as flux densities" (?).

Grid_conserveFluxes Corrects fluxes across fine-course resolution boundaries. See above.

Eos_wrapped

putGravityUnsplit

addGravityUnsplit

energyFix

multiTempAfter This is only needed for when you are using MHD and three temperature.

4. Characteristic Tracing

These are equations I was able to decipher from `hy_uhdDataReconstructNormalDir_PPM`. I'm going to focus on step 8, characteristic tracing, which is similar to MC Section 4.2.3 .

The equations of our hyperbolic system are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0, \tag{16}$$

$$\frac{\partial (\rho v)}{\partial t} + \nabla \cdot (\rho v v) + \nabla P_{\rm gas} + \lambda \nabla E_{\rm r} = 0, \tag{17}$$

$$\frac{\partial E_{\rm gas+kin}}{\partial t} + \nabla \cdot \left[\left(\rho E_{\rm gas+kin} + P_{\rm gas} \right) \mathbf{v} \right] + \lambda v \cdot E_{\rm r} = 0, \tag{18}$$

$$\frac{\partial E_{\rm r}}{\partial t} + \nabla \cdot \left(\frac{3-f}{2} E_{\rm r} v \right) - \lambda v \cdot E_{\rm r} = 0. \tag{19}$$

Our vector of primitive variables is

$$\mathbf{Q} = \begin{pmatrix} \rho \\ v_x \\ v_y \\ v_z \\ P_{\rm tot} \end{pmatrix} \mathbf{Q} = \begin{pmatrix} \rho \\ v_x \\ v_y \\ v_z \\ P_{\rm gas} \\ E_{\rm r} \end{pmatrix}. \tag{20}$$

Z11 linearizes the the Euler equations via

$$\frac{\partial q}{\partial t} + \mathbf{A} \frac{\partial q}{\partial x} = s. \tag{21}$$

I believe our version of **A** is

$$\mathbf{A} = \begin{pmatrix} v & rho & 0 & 0 & 0 & 0 \\ 0 & v & 0 & 0 & \frac{1}{\rho} & \frac{\lambda}{rho} \\ 0 & 0 & v & 0 & 0 & 0 \\ 0 & 0 & 0 & v & 0 & 0 \\ 0 & \gamma P_{\rm gas} & 0 & 0 & v & 0 \\ 0 & \frac{3-f}{2} E_{\rm r} & 0 & 0 & 0 & -\lambda v \end{pmatrix} \tag{22}$$

Our vector of eigenvalues is

$$\boldsymbol{\lambda}_0 = \begin{pmatrix} v - c \\ v \\ v \\ v \\ v + c \end{pmatrix} \boldsymbol{\lambda}_0 = \begin{pmatrix} v - c_{\rm m} \\ v + c_{\rm m} \\ v \\ v \\ v \\ v \end{pmatrix} \tag{23}$$

Where c is sound speed and $c_{\rm m} = \sqrt{\gamma\frac{P_{\rm g}}{\rho} + (\lambda+1)\frac{\lambda E_{\rm r}}{\rho}}$ is the radiation modified sound speed. Note: I'm a little confused about the order of these values in the new version. Z11 presents these in a different ordering than Z13. This is computed in the `hy_uhd_eigenValue` function. This is the variable `lambda0` in the code and known as Λ in MC.

The left eigenvector (variable `leig0`) as defined in `hy_uhd_eigenVector` is

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{2}c & 0 & 0 & 0 & \frac{1}{2c} \\ 0 & -\rho & 0 & \rho & 0 \\ 0 & 0 & 0 & \rho & 0 \\ \frac{1}{2\rho c^2} & 0 & \frac{1}{c} & 0 & \frac{1}{2\rho c^2} \end{pmatrix} \quad \mathbf{L} = \begin{pmatrix} 0 & -\frac{\rho}{2c_{\rm m}} & 0 & 0 & \frac{1}{2c_{\rm m}^2} & \frac{\lambda}{2c_{\rm m}^2} \\ 0 & \frac{\rho}{2c_{\rm m}} & 0 & 0 & \frac{1}{2c_{\rm m}^2} & \frac{\lambda}{2c_{\rm m}^2} \\ 0 & 0 & 0 & 0 & \frac{c_{\rm gas}^2 - c_{\rm m}^2}{c_{\rm m}^2\lambda} & \frac{c_{\rm gas}^2}{c_{\rm m}^2} \\ 1 & 0 & 0 & 0 & -\frac{1}{c_{\rm m}^2} & -\frac{\lambda}{c_{\rm m}^2} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{24}$$

This new version the transpose of what we derived previously, and is the exact form of the matrix found in the code.

The right eigenvector (variable `reig0`) is

$$\mathbf{R} = \begin{pmatrix} \rho & 0 & 1 & 0 & \rho \\ -c & 0 & 0 & 0 & c \\ 0 & 0 & -\frac{1}{\rho} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\rho} & 0 \\ \rho c^2 & 0 & 0 & 0 & \rho c^2 \end{pmatrix} \quad \mathbf{R} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ -\frac{c_{\rm m}}{\rho} & \frac{c_{\rm m}}{\rho} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ c_{\rm gas}^2 & c_{\rm gas}^2 & -\lambda & 0 & 0 & 0 \\ \frac{c_{\rm rad}^2}{\lambda} & \frac{c_{\rm rad}^2}{\lambda} & 1 & 0 & 0 & 0 \end{pmatrix} \tag{25}$$

As above, this is the new version we derived and the exact version now in FLASH..

The array `delW` is

$$\Delta\mathbf{Q} = \mathbf{Q}_{\rm R} - \mathbf{Q}_{\rm L}, \tag{26}$$

where $Q_{\rm R}$ and $Q_{\rm L}$ are the right and left interface values of the primitive variables, which is equivalent to $\delta\mathbf{q}$ defined in equation 64 of MC.

The array `W6` is

$$\mathbf{Q}_6 = 6\mathbf{Q}_{\rm C} - \frac{1}{2}(\mathbf{Q}_{\rm R} + \mathbf{Q}_{\rm L}), \tag{27}$$

where $Q_{\rm C}$ is the cell-centered value of the primitive vector, $\mathbf{Q}$. This is equivalent to q_{6i}

Section 7a of `DataReconstructNormalDir_PPM` calculates

$$q_i = q_{i-\frac{1}{2}} - \lambda_{\max}\frac{\Delta t}{2\Delta x}\left[\Delta q_i - q_{6i}\left(1 - \lambda_{\max}\frac{4}{3}\frac{\Delta t}{\Delta x}\right)\right]. \tag{28}$$

This is similar to, but deviates from equation 71 in MC. It turns out that this calculation is overwritten later in the code, so this is unused.

The final Riemann states are calculated via

$$\mathbf{Q}_{i-\frac{1}{2}}^{t+\frac{\Delta t}{2}} = \mathbf{Q}_i^t + \frac{\mathbf{Q}_{i6}}{12} + \sum_n L_n^*. \tag{29}$$

L_n^*, which is referred to as `vecL` in the code is

$$\begin{aligned} L_n^* &= \frac{1}{2}\left(-1-\frac{\Delta t}{\Delta x}\lambda_0^n\right)\mathbf{R}^n\left(\mathbf{L}\cdot\Delta\mathbf{Q}\right) \\ &+ \frac{1}{4}\left[1+2\frac{\Delta t}{\Delta x}\lambda_0^n+\frac{4}{3}\left(\frac{\Delta t}{\Delta x}\lambda_0^n\right)^2\right]\mathbf{R}^n\left(\mathbf{L}\cdot -\mathbf{Q}_{i6}\right) \end{aligned} \tag{30}$$

where $\mathbf{R}^n$ is the nth column of $\mathbf{R}$. There is a corresponding form of these equations for the right face states as well.

5. Flattening

In `hy_uhd_getRiemannState`, FLASH first calculates the following quantities that will help determine flattening coefficients

$$\Delta P_1 = P_{i+1} - P_{i-1} \tag{31}$$

$$\Delta P_2 = P_{i+2} - P_{i-2} \tag{32}$$

$$\Delta v_1 = v_{i+1} - v_{i-1} \tag{33}$$

$$P_{\mathrm{L}} = P_{i-1} \tag{34}$$

$$P_{\mathrm{R}} = P_{i+1}, \tag{35}$$

where P here is the total pressure that has NOT been modified by the flux limiter, and each value of P and v are taken at cell centers (e.g. P_{i+1} is the cell-centered total pressure at the cell to the right of the current cell we're looking at).

Then we calculate the flattening `tilde` coefficient $\tilde{\chi}$ via

$$\tilde{\chi}_i = \begin{cases} \max\left(0.0, \min\left(1, 10\left(\frac{\Delta P_1}{\Delta P_2} - 0.75\right)\right)\right), & \text{if } |\Delta P_2| > 10^{-15} \text{ and } \left(\frac{|\Delta P_1|}{\min(P_{\mathrm{L}}, P_{\mathrm{R}})} \geq \frac{1}{3} \text{ or } \Delta v_1 < 0\right) \\ 0, & \text{otherwise.} \end{cases}$$

Using this, we calculate the actual flattening coefficient χ

$$\chi_i = \begin{cases} \max\left(\tilde{\chi}_i, \tilde{\chi}_{i+1}\right), & \text{if } \Delta P_1 < 0 \\ \max\left(\tilde{\chi}_i, \tilde{\chi}_{i-1}\right), & \text{if } \Delta P_1 > 0 \\ \tilde{\chi}_i, & \text{otherwise.} \end{cases}$$

The values of χ are sent to `hy_uhd_dataReconstOnestep`, which calls `hy_uhd_DataReconstructNormal` where the flattening is actually carried out via

$$S_{f,i-\frac{1}{2}} = \chi S_i + (1-\chi) S_{i-\frac{1}{2}} \tag{36}$$

$$S_{f,i+\frac{1}{2}} = \chi S_i + (1-\chi) S_{i+\frac{1}{2}}. \tag{37}$$

$$\tag{38}$$

6. GetFaceFlux

This is a detailed summary of what happens in `hy_uhd_getFaceFlux`. The FLASH description is "This routine computes high-order Godunov fluxes at cell interface centers for each spatial direction using a choice of Riemann solvers."

First, we simply get the data in the appropriate formats and arrays and call the HLL solver. The HLL solver returns `Fstar`, the computed flux data. `Fstar` contains fluxes for ρ, p_x (x momentum), p_y, p_z, $E_{\rm tot}$ ρ, p_x (x momentum), p_y, p_z, $E_{\rm kin+gas}$, and $E_{\rm rad}$.

Then, artificial viscosity is applied. In one dimension it is

$$\texttt{cvisc} = \texttt{hy\_cvisc} \max\left(v_{x,i-1} - v_{x,i}, 0\right) F^{(\rho)}_{\rm visc} = F^{(\rho)} + \texttt{cvisc} \tag{39}$$