MAESTRO

An adaptive low-Mach number hydrodynamics code for stratified astrophysical flows

User's Guide

developed at

Center for Computational Sciences and Engineering /
Lawrence Berkeley National Laboratory

&

Stony Brook University

August 11, 2017

Chapter Listing

Contents

List of Figures

List of Tables

Preface

This documentation is a work in progress. MAESTRO is rapidly evolving, so portions of the text are likely to be incomplete or out-of-date.

MAESTRO is developed at the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory and at Stony Brook University. Development is/was supported by the SciDAC Program of the DOE Office of Mathematics, Information, and Computational Sciences and the SciDAC Program of the DOE Office of High Energy Physics under the U.S. Department of Energy under contract No. DE-AC02-05CH11231 to LBNL and by a DOE/Office of Nuclear Physics grant Nos. DE-FG02-06ER41448 and DE-FG02-87ER40317 to Stony Brook.

MAESTRO developers (past and present) include:

Ann Almgren
Vince Beckner
John Bell
Max Duarte
Candace Gilet
Adam Jacobs
Daniel Lecoanet
Mike Lijewski
Chris Malone
Andy Nonaka
Ryan Orvedahl
Charles Rendleman
Weiqun Zhang
Michael Zingale

MAESTRO uses the AMReX library (and formerly, the BoxLib library), developed at the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory. AMReX is distributed

separately from MAESTRO (see `https://github.com/AMReX-Codes/`)

The current version of the MAESTRO User's Guide can be found in the MAESTRO source tree under `MAESTRO/Docs/`.

Support is available via the mailing list: `maestro-help@googlegroups.com`. To subscribe to this list or read the archives, visit `https://groups.google.com/forum/#!forum/maestro-help`.

Getting Help

Support for MAESTRO is available through the `maestro-help@googlegroups.com` mailing list. To subscribe to the list or read through the archives, visit

`https://groups.google.com/forum/#!forum/maestro-help`

Questions can be sent to

maestro-help@googlegroups.com

Part I

Overview of the MAESTRO Algorithm

CHAPTER 1

Introduction

History of MAESTRO

MAESTRO describes the evolution of low Mach number flows that are in hydrostatic equilibrium. The idea for MAESTRO grew out of our success in applying low Mach number combustion methods developed for terrestrial flames [19] to small-scale astrophysical flames (including Landau-Darrieus instability [11], Rayleigh-Taylor unstable flames [39], and flame-turbulence interactions [8]). Our original small-scale astrophysical combustion algorithm is detailed in

- *Adaptive Low Mach Number Simulations of Nuclear Flames,* J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, & M. Zingale 2004, JCP, 195, 2, 677 (henceforth BDRWZ)

MAESTRO was developed initially for modeling the convecting phase in a white dwarf preceding the ignition of a Type Ia supernovae. As such, we needed to incorporate the compressibility effects due to large-scale stratification in the star. The method closest in spirit to MAESTRO is the pseudo-incompressible method of Durran [20], developed for terrestrial atmospheric flows (assuming an ideal gas). Part of the complexity of the equations in MAESTRO stems from the need to descibe a general equation of state. Additionally, since reactions can significantly alter the hydrostatic structure of a star, we incorporated extensions that capture the expansion of the background state [1]. The low Mach number equations for stellar flows were developed in a series of papers leading up to the first application to this problem:

- *Low Mach Number Modeling of Type Ia Supernovae. I. Hydrodynamics,* A. S. Almgren, J. B. Bell, C. A. Rendleman, & M. Zingale 2006, ApJ, 637, 922 (henceforth paper I)
- *Low Mach Number Modeling of Type Ia Supernovae. II. Energy Evolution,* A. S. Almgren, J. B. Bell, C. A. Rendleman, & M. Zingale 2006, ApJ, 649, 927 (henceforth paper II)
- *Low Mach Number Modeling of Type Ia Supernovae. III. Reactions,* A. S. Almgren, J. B. Bell, A. Nonaka, & M. Zingale 2008, ApJ, 684, 449 (henceforth paper III)

- *Low Mach Number Modeling of Type Ia Supernovae. IV. White Dwarf Convection,* M. Zingale, A. S. Almgren, J. B. Bell, A. Nonaka, & S. E. Woosley 2009, ApJ, 704, 196 (henceforth paper IV)

The current version of the algorithm is described in our multilevel paper:

- *MAESTRO: An Adaptive Low Mach Number Hydrodynamics Algorithm for Stellar Flows,* A. Nonaka, A. S. Almgren, J. B. Bell, M. J. Lijewski, C. M. Malone, & M. Zingale 2010, ApJS, 188, 358 (henceforth "the multilevel paper")

We have several papers that describe applications of the method to Type Ia supernovae, X-ray bursts, and stellar evolution. Some of these papers have appendices that describe enhancements to the code—these are noted below.

- *Multidimensional Modeling of Type I X-ray Bursts,* C. M. Malone, A. Nonaka, A. S. Almgren, J. B. Bell, & M. Zingale 2011, ApJ, 728, 118 (henceforth "the XRB paper")

  This introduces the thermal diffusion portion of the MAESTRO algorithm.

- *The Convective Phase Preceding Type Ia Supernovae,* M. Zingale, A. S. Almgren, J. B. Bell, C. M. Malone, A. Nonaka, & S. E. Woosley 2011, ApJ, 740, 8 (henceforth paper V)

- *High-Resolution Simulations of Convection Preceding Ignition in Type Ia Supernovae Using Adaptive Mesh Refinement,* A. Nonaka, M. Zingale, A. J. Aspden, A. S. Almgren, J. B. Bell, & S. E. Woosley 2012, ApJ, 745, 73

- *Low Mach Number Modeling of Convection in Helium Shells on Sub-Chandrasehkar White Dwarfs. I. Methodology,* M. Zingale, A. Nonaka, A. S. Almgren, J. B. Bell, C. M. Malone, & R. J. Orvedahl 2013, ApJ, 764, 97

- *Low-Mach Number Modeling of Core Convection in Massive Stars,* C. Gilet, A. S. Almgren, J. B. Bell, A. Nonaka, S. E. Woosley, & M. Zingale 2013, ApJ, 773, 137

- *Multidimensional Modeling of Type I X-ray Bursts. II. Two-Dimensional Convection in a Mixed H/He Accretor,* C. M. Malone, M. Zingale, A. Nonaka, A. S. Almgren, & J. B. Bell 2014, ApJ, 788, 115

- *Comparisons of Two- and Three-Dimensional Convection in Type I X-ray Bursts* M. Zingale, C. M. Malone, A. Nonaka, A. S. Almgren, & J. B. Bell 2015, ApJ, 807, 60.

  This has an appendix that describes the Godunov state construction in more detail than previous papers.

- *Low Mach Number Modeling of Convection in Helium Shells on Sub-Chandrasekhar White Dwarfs II: Bulk Properties of Simple Models,* A. M. Jacobs, M. Zingale, A. Nonaka, A. S. Almgren, & J. B. Bell 2015, submitted to ApJ

  This has an appendix that shows some test problems for the new energy formulation in MAESTRO.

Brief Overview of Low Speed Approximations

There are many low speed formulations of the equations of hydrodynamics in use, each with their own applications. All of these methods share in common a constraint equation on the velocity field that augments the equations of motion.

Incompressible Hydrodynamics

The simplest low Mach number approximation is incompressible hydrodynamics. This approximation is formally the $M \rightarrow 0$ limit of the Navier-Stokes equations. In incompressible hydrodynamics, the velocity satisfies a constraint equation:

$$\nabla \cdot \mathbf{U} = 0 \tag{1.1}$$

which acts to instantaneously equilibrate the flow, thereby filtering out soundwaves. The constraint equation implies that

$$D\rho/Dt = 0 \tag{1.2}$$

(through the continuity equation) which says that the density is constant along particle paths. This means that there are no compressibility effects modeled in this approximation.

Anelastic Hydrodynamics

In the anelastic approximation small amplitude thermodynamic perturbations are carried with respect to a static hydrostatic background (described by density ρ_0). The density perturbation is ignored in the continuity equation, resulting in a constraint equation:

$$\nabla \cdot (\rho_0 \mathbf{U}) = 0 \tag{1.3}$$

This properly captures the compressibility effects due to the stratification of the background. Because there is no evolution equation for the perturbational density, approximations are made to the buoyancy term in the momentum equation.

Low-Mach Number Combustion

In the low Mach number combustion model, the pressure is decomposed into a dynamic, π, and thermodynamic component, p_0, the ratio of which is $O(M^2)$. The total pressure is replaced everywhere by the thermodynamic pressure, except in the momentum equation. This decouples the pressure and density and filters out the sound waves. Large amplitude density and temperature fluctuations are allowed. The only requirement is that the total pressure stay close to the background pressure, which is assumed constant. This requirement can be expressed as:

$$p = p_0 \tag{1.4}$$

and differentiating this along particle paths leads to a constraint on the velocity field:

$$\nabla \cdot \mathbf{U} = S \tag{1.5}$$

This looks like the constraint for incompressible hydrodynamics, but now we have a source term, S, representing the local compressibility effects due to the energy generation and thermal diffusion. Since the background pressure is taken to be constant, we cannot model flows that cover a large fraction of a pressure scale height. However, this method is ideal for exploring the physics of flames.

Pseudo-Incompressible Methods

The pseudo-incompressible method incorporates both the local changes to compressibility due to reaction/heat release, and the large-scale changes due to the background stratification. This was originally derived for an ideal gas equation of state for atmospherical flows. Allowing the background pressure, p_0 to vary (e.g. in hydrostatic equilibrium), differentiating pressure along particle paths gives:

$$\nabla \cdot (p_0^{1/\gamma} \mathbf{U}) = H \tag{1.6}$$

where γ is the ratio of specific heats and H is the source.

MAESTRO is based on this method, generalizing this constraint to an arbitrary equation of state and allowing for the time-variation of the base state.

Alternate Energy Formulation

Several authors [23, 34] showed that with a slightly different momentum equation, the low Mach number system can conserve an energy (that is, a quantity that looks like the compressible energy, but formed using the low Mach number quantities). This change manifests itself as either a change to the buoyancy term or by changing $\nabla \pi$ to $\beta_0 \nabla (\pi / \beta_0)$. Furthermore, [34] showed that the new formulation better captures the vertical propagation of gravity waves. As of Dec. 2013, this new formulation is the default in MAESTRO.

Projection Methods 101

Most astrophysical hydrodynamics codes (e.g. CASTRO [2] or FLASH [21]) solve the compressible Euler equations, which can be written in the form:

$$\mathbf{U}_t + \nabla \cdot F(\mathbf{U}) = 0 \tag{1.7}$$

where $\mathbf{U}$ is the vector of conserved quantities, $\mathbf{U} = (\rho, \rho u, \rho E)$, with ρ the density, u the velocity, and E the total energy per unit mass. This system of equations can be expressed as a system of advection equations:

$$\mathbf{q}_t + A(\mathbf{q}) \mathbf{q}_x = 0 \tag{1.8}$$

where $\mathbf{q}$ are called the primitive variables, and A is the Jacobian, $A \equiv \partial F / \partial U$. The eigenvalues of the matrix A are the characteristic speeds—the speeds at which information propagates. For the Euler equations, these are u and $u \pm c$, where c is the sound speed. Solution methods for the compressible equations make use of this wave-nature to compute fluxes at the interfaces of grid cells to update the state in time. An excellent introduction to these methods is provided by LeVeque's book [24]. The timestep for these methods is limited by the time it takes for the maximum characteristic speed to traverse one grid cell. For very subsonic flows, this means that the timestep is dominated by the propagation of soundwaves, which may not be important to the overall dynamics of the flow.

In contrast, solving low Mach number systems (including the equations of incompressible hydrodynamics) typically involves solving one or more advection-like equations (representing, e.g. conservation of mass and momentum) coupled with a divergence constraint on the velocity field. For example,

the equations of constant-density incompressible flow are:

$$\mathbf{U}_t = -\mathbf{U}\cdot\nabla\mathbf{U} + \nabla p \tag{1.9}$$

$$\nabla\cdot\mathbf{U} = 0 \tag{1.10}$$

Here, $\mathbf{U}$ represents the velocity vector[1] and p is the dynamical pressure. The time-evolution equation for the velocity (Eq. 1.9) can be solved using techniques similar to those developed for compressible hydrodynamics, updating the old velocity, $\mathbf{U}^n$, to the new time-level, $\mathbf{U}^\star$. Here the '$\star$' indicates that the updated velocity does not, in general, satisfy the divergence constraint. A projection method will take this updated velocity and force it to obey the constraint equation. The basic idea follows from the fact that any vector field can be expressed as the sum of a divergence-free quantity and the gradient of a scalar. For the velocity, we can write:

$$\mathbf{U}^\star = \mathbf{U}^d + \nabla\phi \tag{1.11}$$

where $\mathbf{U}^d$ is the divergence free portion of the velocity vector, $\mathbf{U}^\star$, and ϕ is a scalar. Taking the divergence of Eq. (1.11), we have

$$\nabla^2\phi = \nabla\cdot\mathbf{U}^\star \tag{1.12}$$

(where we used $\nabla\cdot\mathbf{U}^d = 0$). With appropriate boundary conditions, this Poisson equation can be solved for ϕ, and the final, divergence-free velocity can be computed as

$$\mathbf{U}^{n+1} = \mathbf{U}^\star - \nabla\phi \tag{1.13}$$

Because soundwaves are filtered, the timestep constraint now depends only on $|\mathbf{U}|$.

Extensions to variable-density incompressible flows [14] involve a slightly different decomposition of the velocity field and, as a result, a slightly different Poisson equation. There is also a variety of different ways to express what is being projected [3], and different discretizations of the divergence and gradient operators lead to slightly different mathematical properties of the methods (leading to "approximate projections" [7]). Finally, for second-order methods, two projections are typically done per timestep. The first (the 'MAC' projection [10]) operates on the half-time, edge-centered advective velocities, making sure that they satisfy the divergence constraint. These advective velocities are used to construct the fluxes through the interfaces to advance the solution to the new time. The second/final projection operates on the cell-centered velocities at the new time, again enforcing the divergence constraint. The MAESTRO algorithm performs both of these projections.

The MAESTRO algorithm builds upon these ideas, using a different velocity constraint equation that captures the compressibility due to local sources and large-scale stratification.

Notation

Throughout the papers describing MAESTRO, we've largely kept our notation consistent. Table 1.1 defines the frequently-used quantities and provides their units. Additionally, for any quantity ϕ, we denote the average of ϕ over a layer at constaint radius (or height for plane-parallel simulations) as $\overline{\phi}$.

[1] Here we see an unfortunate conflict of notation between the compressible hydro community and the incompressible community. In papers on compressible hydrodynamics, $\mathbf{U}$ will usually mean the vector of conserved quantities. In incompressible / low speed papers, $\mathbf{U}$ will mean the velocity vector.

Table 1.1: Definition of symbols.

symbol	meaning	units
c_p	specific heat at constant pressure ($c_p \equiv \partial h/\partial T\|_{p,X_k}$)	erg g^{-1} K^{-1}
f	volume discrepancy factor ($0 \leq f \leq 1$)	–
g	gravitational acceleration	cm s^{-2}
h	specific enthalpy	erg g^{-1}
$H_{\rm ext}$	external heating energy generation rate	erg g^{-1} s^{-1}
$H_{\rm nuc}$	nuclear energy generation rate	erg g^{-1} s^{-1}
h_p	$h_p \equiv \partial h/\partial p\|_{T,X_k}$	cm^3 g^{-1}
$k_{\rm th}$	thermal conductivity	erg cm^{-1} s^{-1} K^{-1}
p_0	base state pressure	erg cm^{-3}
p_T	$p_T \equiv \partial p/\partial T\|_{\rho,X_k}$	erg cm^{-3} K^{-1}
p_{X_k}	$p_{X_k} \equiv \partial p/\partial X_k\|_{p,T,X_{j,j\neq k}}$	erg cm^{-3}
p_ρ	$p_\rho \equiv \partial p/\partial \rho\|_{T,X_k}$	erg g^{-1}
q_k	specific nuclear binding energy	erg g^{-1}
r	radial coordinate (direction of gravity)	cm
s	specific entropy	erg g^{-1} K^{-1}
S	source term to the divergence constraint	s^{-1}
t	time	s
T	temperature	K
$\mathbf{U}$	total velocity ($\mathbf{U} = \widetilde{\mathbf{U}} + w_0\mathbf{e}_r$)	cm s^{-1}
$\widetilde{\mathbf{U}}$	local velocity	cm s^{-1}
$\widetilde{\mathbf{U}}^{\rm ADV}$	advective velocity (edge-centered)	cm s^{-1}
w_0	base state expansion velocity	cm s^{-1}
X_k	mass fraction of the species ($\sum_k X_k = 1$)	–
β_0	coefficient to velocity in velocity constraint equation	g cm^{-3}
Γ_1	first adiabatic exponent ($\Gamma_1 \equiv d\log p/d\log\rho\|_s$)	–
η_ρ	$\eta_\rho \equiv \overline{(\rho'\mathbf{U}\cdot\mathbf{e}_r)}$	g cm^{-2} s^{-1}
ξ_k	$\xi_k \equiv \partial h/\partial X_k\|_{p,T,X_{j,j\neq k}}$	erg g^{-1}
π	dynamic pressure	erg cm^{-3}

continued on next page

Table 1.1—continued

symbol	**meaning**	**units**
π_0	base state dynamic pressure	erg cm^{-3}
ρ	mass density	g cm^{-3}
ρ_0	base state mass density	g cm^{-3}
ρ'	perturbational density ($\rho' = \rho - \rho_0$)	g cm^{-3}
$(\rho h)_0$	base state enthalpy density	erg cm^{-3}
$(\rho h)'$	perturbational enthalpy density $[(\rho h)' = \rho h - (\rho h)_0]$	erg cm^{-3}
σ	$\sigma \equiv p_T/(\rho c_p p_\rho)$	erg^{-1} g
ψ	$\psi \equiv D_0 p_0/Dt = \partial p_0/\partial t + w_0 \partial p_0/\partial r$	erg cm^{-3} s^{-1}
$\dot{\omega}_k$	creation rate for species k ($\dot{\omega}_k \equiv DX_k/Dt$)	s^{-1}

CHAPTER 2

Equation Set and Algorithm Flowchart

The equation set and solution procedure used by MAESTRO has evolved over time. In this chapter, we outline the algorithm currently implemented in the code. The latest published reference for MAESTROis the multilevel paper [27]. In this description, we make frequent reference to paper I [5], paper II [6], paper III [4], and paper IV [37].

Summary of the MAESTRO Equation Set

Here we summarize the equations solved by MAESTRO. We refer the reader to papers I through IV and the multilevel paper for the derivation and motivation of the equation set. (Note: this 'traditional' algorithm uses Strang-splitting for the reactions. An alternate implementation, using spectral deferred corrections is outlined in Chapter 20.)

Base State

The stratified atmosphere is characterized by a one-dimensional time-dependent base state, defined by a base state density, ρ_0, and a base state pressure, p_0, in hydrostatic equilibrum:

$$\nabla p_0 = -\rho_0 |g| \mathbf{e}_r \tag{2.1}$$

The gravitational acceleration, g is either constant or a point-mass with a $1/r^2$ dependence (see §26.1) for plane-parallel geometries, or a monopole constructed by integrating the base state density for spherical geometries.

For the time-dependence, we will define a base state velocity, w_0, which will adjust the base state from one hydrostatic equilibrum to another in response to heating.

For convenience, we define a base state enthalphy, h_0, as needed by laterally averaging the full enthalpy, h.

Continuity

Conservation of mass gives the same continuity equation we have with compressible flow:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{2.2}$$

Additionally, we carry species around which can react. The creation and destruction of the species is described by their create rate, $\dot{\omega}_k$, and the species are defined by their mass fractions, $X_k \equiv \rho_k / \rho$, giving

$$\frac{\partial \rho X_k}{\partial t} + \nabla \cdot (\rho \mathbf{U} X_k) = \rho \dot{\omega}_k \tag{2.3}$$

and

$$\sum_k X_k = 1 \tag{2.4}$$

The base state density evolution equation can be defined by laterally averaging the full continuity equation, giving:

$$\frac{\partial \rho_0}{\partial t} = -\nabla \cdot (\rho_0 w_0 \mathbf{e}_r), \tag{2.5}$$

Subtracting these two yields the evolution equation for the perturbational density, $\rho' \equiv \rho - \rho_0$:

$$\frac{\partial \rho'}{\partial t} = -\mathbf{U} \cdot \nabla \rho' - \rho' \nabla \cdot \mathbf{U} - \nabla \cdot \left(\rho_0 \widetilde{\mathbf{U}} \right) \tag{2.6}$$

As first discussed in paper III and then refined in the multilevel paper, we capture the changes that can occur due to significant convective overturning by imposing the constraint that $\overline{\rho'} = 0$ for all time. This gives

$$\frac{\partial \overline{\rho'}}{\partial t} = -\nabla \cdot (\eta_\rho \mathbf{e}_r). \tag{2.7}$$

where

$$\eta_\rho = \overline{(\rho' \mathbf{U} \cdot \mathbf{e}_r)} \tag{2.8}$$

In practice, we correct the drift by simply setting $\rho_0 = \overline{\rho}$ after the advective update of ρ. However we still need to explicitly compute η_ρ since it appears in other equations.

Constraint

The equation of state is cast into an elliptic constraint on the velocity field by differentiating $p_0(\rho, s, X_k)$ along particle paths, giving:

$$\nabla \cdot (\beta_0 \mathbf{U}) = \beta_0 \left(S - \frac{1}{\overline{\Gamma}_1 p_0} \frac{\partial p_0}{\partial t} \right) \tag{2.9}$$

where β_0 is a density-like variable that carries background stratification, defined as

$$\beta_0(r,t) = \rho_0(0,t) \exp\left(\int_0^r \frac{1}{\overline{\Gamma}_1 p_0} \frac{\partial p_0}{\partial r'} dr'\right), \tag{2.10}$$

and

$$S = -\sigma \sum_k \xi_k \dot{\omega}_k + \frac{1}{\rho p_\rho} \sum_k p_{X_k} \dot{\omega}_k + \sigma H_{\rm nuc} + \sigma H_{\rm ext} + \frac{\sigma}{\rho} \nabla \cdot k_{\rm th} \nabla T \tag{2.11}$$

where $p_{X_k} \equiv \partial p/\partial X_k|_{\rho,T,X_{j,j\neq k}}$, $\xi_k \equiv \partial h/\partial X_k|_{p,T,X_{j,j\neq k}}$, $p_\rho \equiv \partial p/\partial \rho|_{T,X_k}$, and $\sigma \equiv p_T/(\rho c_p p_\rho)$, with $p_T \equiv \partial p/\partial T|_{\rho,X_k}$ and $c_p \equiv \partial h/\partial T|_{p,X_k}$ is the specific heat at constant pressure. The last term is only present if we are using thermal diffusion (`use_thermal_diffusion = T`). In this term, $k_{\rm th}$ is the thermal conductivity.

In this constraint, $\overline{\Gamma}_1$ is the lateral average of $\Gamma_1 \equiv d\log p/d\log\rho|_s$. Using the lateral average here makes it possible to cast the constraint as a divergence. [23] discuss the general case where we want to keep the local variations of Γ_1 (and we explored this in paper III). We also look at this in § 2.2.4.

Momentum

The compressible momentum equation (written in terms of velocity is):

$$\rho \frac{\partial \mathbf{U}}{\partial t} + \rho \mathbf{U} \cdot \nabla \mathbf{U} + \nabla p = -\rho |g| \mathbf{e}_r \tag{2.12}$$

Subtracting off the base state, and defining the perturbational pressure (sometimes called the dynamic pressure) as $\pi \equiv p - p_0$, and perturbational density as $\rho' \equiv \rho - \rho_0$, we have:

$$\rho \frac{\partial \mathbf{U}}{\partial t} + \rho \mathbf{U} \cdot \nabla \mathbf{U} + \nabla \pi = -\rho' |g| \mathbf{e}_r \tag{2.13}$$

or

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \frac{1}{\rho} \nabla \pi = -\frac{\rho'}{\rho} |g| \mathbf{e}_r \tag{2.14}$$

This is the form of the momentum equation that we solved in papers I–IV and in the multilevel paper.

Several authors [23, 34] explored the idea of energy conservation in a low Mach number system and found that an additional term (which can look like a buoyancy) is needed in the low Mach number formulation, yielding:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \frac{\beta_0}{\rho} \nabla \left(\frac{p'}{\beta_0}\right) = -\frac{\rho'}{\rho} |g| \mathbf{e}_r \tag{2.15}$$

This is the form that we enforce in MAESTRO, and the choice is controlled by `use_alt_energy_fix`.

We decompose the full velocity field into a base state velocity, w_0, that governs the base state dynamics, and a local velocity, $\widetilde{\mathbf{U}}$, that governs the local dynamics, i.e.,

$$\mathbf{U} = w_0(r,t)\mathbf{e}_r + \widetilde{\mathbf{U}}(\mathbf{x},t). \tag{2.16}$$

with $\overline{(\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)} = 0$ and $w_0 = \overline{(\mathbf{U} \cdot \mathbf{e}_r)}$—the motivation for this splitting was given in paper II. The velocity evolution equations are then

$$\frac{\partial w_0}{\partial t} = -w_0 \frac{\partial w_0}{\partial r} - \frac{\beta_0}{\rho_0} \frac{\partial(\pi_0/\beta_0)}{\partial r}, \tag{2.17}$$

$$\frac{\partial \widetilde{\mathbf{U}}}{\partial t} = -(\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r) \cdot \nabla \widetilde{\mathbf{U}} - \left(\widetilde{\mathbf{U}} \cdot \mathbf{e}_r\right) \frac{\partial w_0}{\partial r} \mathbf{e}_r - \frac{\beta_0}{\rho} \nabla \left(\frac{\pi}{\beta_0}\right) + \frac{\beta_0}{\rho_0} \frac{\partial(\pi_0/\beta_0)}{\partial r} \mathbf{e}_r - \frac{\rho - \rho_0}{\rho} g \mathbf{e}_r. \tag{2.18}$$

where π_0 is the base state component of the perturbational pressure. By laterally averaging to equation (2.9), we obtain a divergence constraint for w_0:

$$\nabla \cdot (\beta_0 w_0 \mathbf{e}_r) = \beta_0 \left(\overline{S} - \frac{1}{\overline{\Gamma}_1 p_0} \frac{\partial p_0}{\partial t} \right). \tag{2.19}$$

The divergence constraint for $\widetilde{\mathbf{U}}$ can be found by subtracting (2.19) into (2.9), resulting in

$$\nabla \cdot \left(\beta_0 \widetilde{\mathbf{U}} \right) = \beta_0 \left(S - \overline{S} \right). \tag{2.20}$$

Base State Expansion

In practice, we calculate w_0 by integrating the one-dimensional divergence constraint. For a plane-parallel atmosphere, the evolution is:

$$\frac{\partial w_0}{\partial r} = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0} \eta_\rho g \tag{2.21}$$

his term needs the $_0$'s as shown. Also, s this really the orm, or should it be $\beta_0/\rho_0)\partial(\pi_0/\beta_0)/\partial r$?

Then we define

$$-\frac{\beta_0}{\rho_0} \frac{\partial (\pi_0/\beta_0)}{\partial r} = \frac{\partial w_0}{\partial t} + w_0 \frac{\partial w_0}{\partial r} \ , \tag{2.22}$$

once w_0 at the old and new times is known, and the advective term is computed explicitly. Then we can include this for completeness in the update for $\tilde{u}$.

Energy

Finally, we add an equation for specific enthalpy evolution to our system. Strictly speaking this is not necessary to close the system, but it becomes convenient at times to define the temperature.

$$\frac{\partial (\rho h)}{\partial t} = -\nabla \cdot (\rho h \mathbf{U}) + \frac{D p_0}{D t} + \rho H_{\text{nuc}} + \rho H_{\text{ext}}, \tag{2.23}$$

We will often expand Dp_0/Dt as

$$\frac{D p_0}{D t} = \psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r) \frac{\partial p_0}{\partial r} \tag{2.24}$$

where we defined

$$\psi \equiv \frac{\partial p_0}{\partial t} + w_0 \frac{\partial p_0}{\partial r} \tag{2.25}$$

When we are using thermal diffusion, there will be an additional term in the enthalpy equation (see § 2.2.5).

In paper III, we showed that for a plane-parallel atmosphere with constant gravity, $\psi = \eta_\rho g$

At times, we will define a temperature equation by writing $h = h(T, p, X_k)$ and differentiating:

$$\frac{DT}{Dt} = \frac{1}{\rho c_p} \left\{ \left(1 - \rho h_p\right) \left[\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r) \frac{\partial p_0}{\partial r} \right] - \sum_k \rho \xi_k \dot{\omega}_k + \rho H_{\text{nuc}} + \rho H_{\text{ext}} \right\} \ . \tag{2.26}$$

The base state evolution equations for density and enthalpy can be found by averaging Eq 2.23 over a layer of constant radius, resulting in

$$\frac{\partial(\rho h)_0}{\partial t} = -\nabla\cdot[(\rho h)_0 w_0 \mathbf{e}_r] + \psi + \overline{\rho H_{\rm nuc}} + \overline{\rho H_{\rm ext}}. \tag{2.27}$$

Subtracting it from the full enthalpy equation gives:

$$\frac{\partial(\rho h)'}{\partial t} = -\mathbf{U}\cdot\nabla(\rho h)' - (\rho h)'\nabla\cdot\mathbf{U} - \nabla\cdot\left[(\rho h)_0\widetilde{\mathbf{U}}\right] + \widetilde{\mathbf{U}}\cdot\nabla p_0 + (\rho H_{\rm nuc} - \overline{\rho H_{\rm nuc}}) + (\rho H_{\rm ext} - \overline{\rho H_{\rm ext}}) \tag{2.28}$$

Time Advancement Algorithm

Here is the current description of the algorithm, based on the description in the multilevel paper. The initialization has also been included with more detail than was given in paper III. The main driver for a single step of the algorithm is `advance.f90`—refer to that code to see the sequence of functions called to implement each step.

Definitions

Below we define operations that will be referenced in § 2.2.2.

- **React State**$[\rho^{\rm in}, (\rho h)^{\rm in}, X_k^{\rm in}, T^{\rm in}, (\rho H_{\rm ext})^{\rm in}, p_0^{\rm in}] \rightarrow [\rho^{\rm out}, (\rho h)^{\rm out}, X_k^{\rm out}, T^{\rm out}, (\rho\dot\omega_k)^{\rm out}, (\rho H_{\rm nuc})^{\rm out}]$ evolves the species and enthalpy due to reactions through $\Delta t/2$ according to:

$$\frac{dX_k}{dt} = \dot\omega_k(\rho, X_k, T); \qquad \frac{dT}{dt} = \frac{1}{c_p}\left(-\sum_k \xi_k\dot\omega_k + H_{\rm nuc}\right). \tag{2.29}$$

  Here the temperature equation comes from Eq. 2.26 with $Dp_0/Dt = 0$ for the burning part of the evolution.

  Full details of the solution procedure can be found in Paper III. We then define:

$$(\rho\dot\omega_k)^{\rm out} = \frac{\rho^{\rm out}(X_k^{\rm out} - X_k^{\rm in})}{\Delta t/2}, \tag{2.30}$$

$$(\rho h)^{\rm out} = (\rho h)^{\rm in} + \frac{\Delta t}{2}(\rho H_{\rm nuc})^{\rm out} + \frac{\Delta t}{2}(\rho H_{\rm ext})^{\rm in}. \tag{2.31}$$

  where the enthalpy update includes external heat sources $(\rho H_{\rm ext})^{\rm in}$. As introduced in Paper IV, we update the temperature using $T^{\rm out} = T(\rho^{\rm out}, h^{\rm out}, X_k^{\rm out})$ for planar geometry or $T^{\rm out} = T(\rho^{\rm out}, p_0^{\rm in}, X_k^{\rm out})$ for spherical geometry, with this behavior controlled by `use_tfromp`. Note that the density remains unchanged within **React State**, i.e., $\rho^{\rm out} = \rho^{\rm in}$.

  The source code for this operation can be found in `react_state.f90`.

- **Advect Base Density**$[\rho_0^{\rm in}, w_0^{\rm in}] \rightarrow [\rho_0^{\rm out}, \rho_0^{{\rm out},n+1/2}]$ is the process by which we update the base state density through Δt in time. We keep the time-centered edge states, $\rho_0^{{\rm out},n+1/2}$, since they are used later in discretization of η_ρ for planar problems.

planar: We discretize equation (2.5) to compute the new base state density,

$$\rho_{0,j}^{\text{out}} = \rho_{0,j}^{\text{in}} - \frac{\Delta t}{\Delta r}\left[\left(\rho_0^{\text{out},n+1/2} w_0^{\text{in}}\right)_{j+1/2} - \left(\rho_0^{\text{out},n+1/2} w_0^{\text{in}}\right)_{j-1/2}\right]. \tag{2.32}$$

We compute the time-centered edge states, $\rho_{0\,j\pm1/2}^{\text{out},n+1/2}$, by discretizing an expanded form of equation (2.5):

$$\frac{\partial \rho_0}{\partial t} + w_0 \frac{\partial \rho_0}{\partial r} = -\rho_0 \frac{\partial w_0}{\partial r}, \tag{2.33}$$

where the right hand side is used as the force term.

spherical: The base state density update now includes the area factors in the divergences:

$$\rho_{0,j}^{\text{out}} = \rho_{0,j}^{\text{in}} - \frac{1}{r_j^2}\frac{\Delta t}{\Delta r}\left[\left(r^2 \rho_0^{\text{out},n+1/2} w_0^{\text{in}}\right)_{j+1/2} - \left(r^2 \rho_0^{\text{out},n+1/2} w_0^{\text{in}}\right)_{j-1/2}\right]. \tag{2.34}$$

In order to compute the time-centered edge states, an additional geometric term is added to the forcing, due to the spherical discretization of (2.5):

$$\frac{\partial \rho_0}{\partial t} + w_0 \frac{\partial \rho_0}{\partial r} = -\rho_0 \frac{\partial w_0}{\partial r} - \frac{2\rho_0 w_0}{r}. \tag{2.35}$$

The source code for this operation can be found in `advect_base.f90`.

- **Enforce HSE**$[p_0^{\text{in}}, \rho_0^{\text{in}}] \rightarrow [p_0^{\text{out}}]$ has replaced **Advect Base Pressure** from Paper III as the process by which we update the base state pressure. Rather than discretizing the evolution equation for p_0, we enforce hydrostatic equilibrium directly, which is numerically simpler and analytically equivalent. We first set $p_{0,j=0}^{\text{out}} = p_{0,j=0}^{\text{in}}$ and then update p_0^{out} using:

$$p_{0,j+1}^{\text{out}} = p_{0,j}^{\text{out}} + \Delta r g_{j+1/2} \frac{\left(\rho_{0,j+1}^{\text{in}} + \rho_{0,j}^{\text{in}}\right)}{2}, \tag{2.36}$$

where $g = g(\rho_0^{\text{in}})$. As soon as $\rho_{0,j}^{\text{in}} < \rho_{\text{cutoff}}$, we set $p_{0,j+1}^{\text{out}} = p_{0,j}^{\text{out}}$ for all remaining values of j. Then we compare $p_{0,j_{\max}}^{\text{out}}$ with $p_{0,j_{\max}}^{\text{in}}$ and offset every element in p_0^{out} so that $p_{0,j_{\max}}^{\text{out}} = p_{0,j_{\max}}^{\text{in}}$. We are effectively using the location where the ρ_0^{in} drops below ρ_{cutoff} as the starting point for integration.

The source code for this operation can be found in `enforce_HSE.f90`.

- **Advect Base Enthalpy**$[(\rho h)_0^{\text{in}}, w_0^{\text{in}}, \psi^{\text{in}}] \rightarrow [(\rho h)_0^{\text{out}}]$ is the process by which we update the base state enthalpy through Δt in time.

  planar: We discretize equation (2.27), neglecting reaction source terms, to compute the new base state enthalpy,

$$(\rho h)_{0,j}^{\text{out}} = (\rho h)_{0,j}^{\text{in}} - \frac{\Delta t}{\Delta r}\left\{\left[(\rho h)_0^{n+1/2} w_0^{\text{in}}\right]_{j+1/2} - \left[(\rho h)_0^{n+1/2} w_0^{\text{in}}\right]_{j-1/2}\right\} + \Delta t \psi_j^{\text{in}}. \tag{2.37}$$

  We compute the time-centered edge states, $(\rho h)_0^{n+1/2}$, by discretizing an expanded form of equation (2.27):

$$\frac{\partial (\rho h)_0}{\partial t} + w_0 \frac{\partial (\rho h)_0}{\partial r} = -(\rho h)_0 \frac{\partial w_0}{\partial r} + \psi. \tag{2.38}$$

spherical: The base state enthalpy update now includes the area factors in the divergences:

$$\begin{aligned}(\rho h)_{0,j}^{\text{out}} &= (\rho h)_{0,j}^{\text{in}} \\ &\quad -\frac{1}{r_j^2}\frac{\Delta t}{\Delta r}\left\{\left[r^2(\rho h)_0^{n+1/2}w_0^{\text{in}}\right]_{j+1/2} - \left[r^2(\rho h)_0^{n+1/2}w_0^{\text{in}}\right]_{j-1/2}\right\} + \Delta t\psi^{\text{in},n+1/2}.\end{aligned} \tag{2.39}$$

In order to compute the time-centered edge states, an additional geometric term is added to the forcing, due to the spherical discretization of (2.27):

$$\frac{\partial(\rho h)_0}{\partial t} + w_0\frac{\partial(\rho h)_0}{\partial r} = -(\rho h)_0\frac{\partial w_0}{\partial r} - \frac{2(\rho h)_0 w_0}{r} + \psi. \tag{2.40}$$

The source code for this operation can be found in `advect_base.f90`.

- **Computing w_0** Here we describe the process by which we compute w_0. The arguments are different for planar and spherical geometries.

  Compute w_0 Planar $[\overline{S}^{\text{in}}, \overline{\Gamma}_1^{\text{in}}, p_0^{\text{in}}, \psi^{\text{in}}] \rightarrow [w_0^{\text{out}}]$:

  In Paper III, we showed that $\psi = \eta_\rho g$ for planar geometries, and derived derived Eq. 2.21 as an alternate expression for Eq. 2.19. We discretize this as:

$$\frac{w_{0,j+1/2}^{\text{out}} - w_{0,j-1/2}^{\text{out}}}{\Delta r} = \left(\overline{S}^{\text{in}} - \frac{1}{\overline{\Gamma}_1^{\text{in}} p_0^{\text{in}}}\psi^{\text{in}}\right)_j, \tag{2.41}$$

  with $w_{0,-1/2} = 0$.

  Compute w_0 Spherical $[\overline{S}^{\text{in}}, \overline{\Gamma}_1^{\text{in}}, \rho_0^{\text{in}}, p_0^{\text{in}}, \eta_\rho^{\text{in}}] \rightarrow [w_0^{\text{out}}]$:

  We begin with equation (2.19) written in spherical coordinates:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\beta_0 w_0\right) = \beta_0\left(\overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t}\right). \tag{2.42}$$

  We expand the spatial derivative and recall from Paper I that

$$\frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial r} = \frac{1}{\beta_0}\frac{\partial \beta_0}{\partial r}, \tag{2.43}$$

  giving:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 w_0\right) = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\underbrace{\left(\frac{\partial p_0}{\partial t} + w_0\frac{\partial p_0}{\partial r}\right)}_{\psi}. \tag{2.44}$$

  We solve this equation for w_0 as described in Appendix B of the multilevel paper.

The source code for this operation can be found in `make_w0.f90`.

Single Step

Step 0. *Initialization*

This step remains unchanged from Paper III. See §2.3 for details. The initialization step only occurs at the beginning of the simulation. The initial values for $\mathbf{U}^0, \rho^0, (\rho h)^0, X_k^0, T^0, \rho_0^0, p_0^0$, and $\overline{\Gamma_1^0}$ are specified from the problem-dependent initial conditions. The initial time step, Δt^0, is computed as in Paper III. Finally, initial values for $w_0^{-1/2}, \eta_\rho^{-1/2}, \psi^{-1/2}, \pi^{-1/2}, S^0$, and S^1 come from a preliminary pass through the algorithm.

Step 1. *React the full state through the first time interval of* $\Delta t/2$.

Call **React State**$[\rho^n, (\rho h)^n, X_k^n, T^n, (\rho H_{\rm ext})^n, p_0^n] \rightarrow [\rho^{(1)}, (\rho h)^{(1)}, X_k^{(1)}, T^{(1)}, (\rho \dot{\omega}_k)^{(1)}, (\rho H_{\rm nuc})^{(1)}]$.

Step 2. *Compute the provisional time-centered expansion,* $S^{n+1/2,\star\star}$*, provisional base state velocity,* $w_0^{n+1/2,\star}$*, and provisional base state velocity forcing.*

A. Compute $S^{n+1/2,\star\star}$. We compute an estimate for the time-centered expansion term in the velocity divergence constraint, as given in Eq. 2.11. For the first time step ($n = 0$), we set

$$S^{n+1/2,\star\star} = \frac{S^0 + S^1}{2}, \tag{2.45}$$

where S^1 is found during initialization. For other time steps ($n \neq 0$), following [13], we extrapolate to the half-time using S at the previous and current time levels

$$S^{n+1/2,\star\star} = S^n + \frac{\Delta t^n}{2} \frac{S^n - S^{n-1}}{\Delta t^{n-1}}. \tag{2.46}$$

Next, compute

$$\overline{S^{n+1/2,\star\star}} = \mathbf{Avg}\left(S^{n+1/2,\star\star}\right). \tag{2.47}$$

B. Compute $w_0^{n+1/2,\star}$.

For planar geometry, call
Compute w_0 Planar$[\overline{S^{n+1/2,\star\star}}, \overline{\Gamma_1^n}, p_0^n, \psi^{n-1/2}] \rightarrow [w_0^{n+1/2,\star}]$.

For spherical geometry, call
Compute w_0 Spherical$[\overline{S^{n+1/2,\star\star}}, \overline{\Gamma_1^n}, \rho_0^n, p_0^n, \eta_\rho^{n-1/2}] \rightarrow [w_0^{n+1/2,\star}]$.

C. Compute the provisional base state velocity forcing, using equation (38) from paper III,

$$-\frac{\beta_0}{\rho_0}\frac{\partial(\pi_0/\beta_0)}{\partial r} = \frac{\partial w_0}{\partial t} + w_0 \frac{\partial w_0}{\partial r}, \tag{2.48}$$

with the following discretization:

$$\left(\frac{\beta_0}{\rho_0}\frac{\partial(\pi_0/\beta_0)}{\partial r}\right)^{n,\star} = -\frac{w_0^{n+1/2,\star} - w_0^{n-1/2}}{(\Delta t^n + \Delta t^{n-1})/2} - w_0^{n,\star}\left(\frac{\partial w_0}{\partial r}\right)^{n,\star}, \tag{2.49}$$

where $w_0^{n,\star}$ and $(\partial w_0/\partial r)^{n,\star}$ are defined as

$$w_0^{n,\star} = \frac{\Delta t^n w_0^{n-1/2} + \Delta t^{n-1} w_0^{n+1/2,\star}}{\Delta t^n + \Delta t^{n-1}}, \tag{2.50}$$

$$\left(\frac{\partial w_0}{\partial r}\right)^{n,\star} = \frac{1}{\Delta t^n + \Delta t^{n-1}} \left[\Delta t^n \left(\frac{\partial w_0}{\partial r}\right)^{n-1/2} + \Delta t^{n-1} \left(\frac{\partial w_0}{\partial r}\right)^{n+1/2,\star}\right]. \tag{2.51}$$

If $n = 0$, we use $\Delta t^{-1} = \Delta t^0$.

Step 3. *Construct the provisional time-centered advective velocity on edges, $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$.*

The local velocity field is described by Eq. 2.18. From this, we compute time-centered edge velocities, $\widetilde{\mathbf{U}}^{\mathrm{ADV},\dagger,\star}$, using $\mathbf{U} = \widetilde{\mathbf{U}}^n + w_0^{n+1/2,\star}$. The † superscript refers to the fact that the predicted velocity field does not satisfy the divergence constraint. We then construct $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$ from $\widetilde{\mathbf{U}}^{\mathrm{ADV},\dagger,\star}$ using a MAC projection, as described in detail in Appendix B of Paper III. We note that $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$ satisfies the discrete version of $\overline{(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} \cdot \mathbf{e}_r)} = 0$ as well as

$$\nabla \cdot \left(\beta_0^n \widetilde{\mathbf{U}}^{\mathrm{ADV},\star}\right) = \beta_0^n \left(S^{n+1/2,\star\star} - \overline{S^{n+1/2,\star\star}}\right), \tag{2.52}$$

$$\beta_0^n = \beta_0 \left(\rho_0^n, p_0^n, \overline{\Gamma_1^n}\right), \tag{2.53}$$

where β_0 is computed as described in Appendix C of Paper III (although note that an alternate procedure that uses linear reconstruction of g in constructing β_0 is enabled with `use_linear_grav_in_beta`).

Step 4. *Advect the base state and full state through a time interval of Δt.*

A. Update ρ_0, saving the time-centered density at radial edges by calling

Advect Base Density$[\rho_0^n, w_0^{n+1/2,\star}] \rightarrow [\rho_0^{(2a),\star}, \rho_0^{n+1/2,\star,\mathrm{pred}}]$.

B. Update (ρX_k) using a discretized version of the species continuity equation, Eq. 2.3, where we omit the reaction terms, which were already accounted for in **React State**:

$$\frac{\partial(\rho X_k)}{\partial t} + \nabla \cdot (\mathbf{U} \rho X_k) = 0 \ . \tag{2.54}$$

The update consists of two steps:

i. Compute the time-centered species edge states, $(\rho X_k)^{n+1/2,\star,\mathrm{pred}}$, for the conservative update of $(\rho X_k)^{(1)}$.

There are a variety of choices of quantities to predict to the edges (controlled by `species_pred_type`—see Chapter 19). By default, we use the equations

$$\frac{\partial \rho'}{\partial t} = -\mathbf{U} \cdot \nabla \rho' - \rho' \nabla \cdot \mathbf{U} - \nabla \cdot \left(\rho_0 \widetilde{\mathbf{U}}\right), \tag{2.55}$$

$$\frac{\partial X_k}{\partial t} = -\mathbf{U} \cdot \nabla X_k + \dot{\omega}_k. \tag{2.56}$$

to predict $\rho'^{(1)} = \rho^{(1)} - \rho_0^n$ and $X_k^{(1)} = (\rho X_k)^{(1)}/\rho^{(1)}$ to time-centered edges using $\mathbf{U} = \widetilde{\mathbf{U}}^{\mathrm{ADV},\star} + w_0^{n+1/2,\star}\mathbf{e}_r$, yielding $\rho'^{n+1/2,\star,\mathrm{pred}}$ and $X_k^{n+1/2,\star,\mathrm{pred}}$. We convert the perturbational density to full state density using

$$\rho^{n+1/2,\star,\mathrm{pred}} = \rho'^{n+1/2,\star,\mathrm{pred}} + \frac{\rho_0^n + \rho_0^{(2a),\star}}{2}, \tag{2.57}$$

where the base state density terms are mapped to Cartesian edges. Then, $(\rho X_k)^{n+1/2,\star,\mathrm{pred}} = \rho^{n+1/2,\star,\mathrm{pred}}\, X_k^{n+1/2,\star,\mathrm{pred}}$.

ii. Evolve $(\rho X_k)^{(1)} \rightarrow (\rho X_k)^{(2),\star}$ using

$$\begin{aligned}(\rho X_k)^{(2),\star} &= (\rho X_k)^{(1)} \\ &\quad -\Delta t\left\{\nabla\cdot\left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} + w_0^{n+1/2,\star}\mathbf{e}_r\right)(\rho X_k)^{n+1/2,\star,\mathrm{pred}}\right]\right\},\end{aligned} \tag{2.58}$$

$$\rho^{(2),\star} = \sum_k (\rho X_k)^{(2),\star}, \qquad X_k^{(2),\star} = (\rho X_k)^{(2),\star}/\rho^{(2),\star}. \tag{2.59}$$

C. Define a radial edge-centered $\eta_\rho^{n+1/2,\star}$ (Eq. 2.8).

For planar geometry, since $\eta_\rho = \overline{\rho'(\mathbf{U}\cdot\mathbf{e}_r)} = \overline{\rho(\mathbf{U}\cdot\mathbf{e}_r)} - \overline{\rho_0(\mathbf{U}\cdot\mathbf{e}_r)} = \overline{\rho(\mathbf{U}\cdot\mathbf{e}_r)} - \rho_0 w_0$,

$$\begin{aligned}\eta_\rho^{n+1/2,\star} &= \mathbf{Avg}\sum_k\left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}\cdot\mathbf{e}_r + w_0^{n+1/2,\star}\right)(\rho X_k)^{n+1/2,\star,\mathrm{pred}}\right] \\ &\quad - w_0^{n+1/2,\star}\rho_0^{n+1/2,\star,\mathrm{pred}},\end{aligned} \tag{2.60}$$

For spherical geometry, first construct $\eta_\rho^{\mathrm{cart},n+1/2,\star} = [\rho'(\mathbf{U}\cdot\mathbf{e}_r)]^{n+1/2,\star}$ on Cartesian cell centers using:

$$\begin{aligned}\eta_\rho^{\mathrm{cart},n+1/2,\star} &= \left[\left(\frac{\rho^{(1)} + \rho^{(2),\star}}{2}\right) - \left(\frac{\rho_0^n + \rho_0^{(2a),\star}}{2}\right)\right] \\ &\quad \cdot\left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}\cdot\mathbf{e}_r + w_0^{n+1/2,\star}\right).\end{aligned} \tag{2.61}$$

Then,

$$\eta_\rho^{n+1/2,\star} = \mathbf{Avg}\left(\eta_\rho^{\mathrm{cart},n+1/2,\star}\right). \tag{2.62}$$

This gives a radial cell-centered $\eta_\rho^{n+1/2,\star}$. To get $\eta_\rho^{n+1/2,\star}$ at radial edges, average the two neighboring radial cell-centered values.

D. Correct ρ_0 by setting $\rho_0^{n+1,\star} = \mathbf{Avg}(\rho^{(2),\star})$.

E. Update p_0 by calling **Enforce HSE**$[p_0^n, \rho_0^{n+1,\star}] \rightarrow [p_0^{n+1,\star}]$.

F. Compute $\psi^{n+1/2,\star}$.

For planar geometry,

$$\psi_j^{n+1/2,\star} = \frac{1}{2}\left(\eta_{\rho,j-1/2}^{n+1/2,\star} + \eta_{\rho,j+1/2}^{n+1/2,\star}\right) g. \tag{2.63}$$

For spherical geometry, first compute:

$$\overline{\Gamma_1^{(1)}} = \mathbf{Avg}\left[\Gamma_1\left(\rho^{(1)}, p_0^n, X_k^{(1)}\right)\right], \tag{2.64}$$

$$\overline{\Gamma_1^{(2),\star}} = \mathbf{Avg}\left[\Gamma_1\left(\rho^{(2),\star}, p_0^{n+1,\star}, X_k^{(2),\star}\right)\right]. \tag{2.65}$$

Then, define $\psi^{n+1/2,\star}$ using equation (2.44)

$$\psi_j^{n+1/2,\star} = \left(\frac{\overline{\Gamma_1^{(1)}} + \overline{\Gamma_1^{(2),\star}}}{2}\right)_j \left(\frac{p_0^n + p_0^{n+1,\star}}{2}\right)_j \left\{\overline{S_j^{n+1/2,\star}} - \frac{1}{r_j^2}\left[\left(r^2 w_0^{n+1/2,\star}\right)_{j+1/2} - \left(r^2 w_0^{n+1/2,\star}\right)_{j-1/2}\right]\right\}. \tag{2.66}$$

G. Update $(\rho h)_0$. First, compute $(\rho h)_0^n = \mathbf{Avg}[(\rho h)^{(1)}]$. Then, call **Advect Base Enthalpy**$[(\rho h)_0^n, w_0^{n+1/2,\star}, \psi^{n+1/2,\star}] \rightarrow [(\rho h)_0^{n+1,\star}]$.

H. Update the enthalpy using a discretized version of the enthalpy evolution equation (Eq. 2.23), again omitting the reaction and heating terms since we already accounted for them in **React State**. This equation takes the form:

$$\frac{\partial(\rho h)}{\partial t} = -\nabla\cdot(\mathbf{U}\rho h) + \psi + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r}. \tag{2.67}$$

For spherical geometry, we solve the analytically equivalent form,

$$\frac{\partial(\rho h)}{\partial t} = -\nabla\cdot(\mathbf{U}\rho h) + \psi + \nabla\cdot(\widetilde{\mathbf{U}}p_0) - p_0\nabla\cdot\widetilde{\mathbf{U}}, \tag{2.68}$$

which experience has shown to minimize the drift from thermodynamic equilibrium. The update consists of two steps:

i. Compute the time-centered enthalpy edge state, $(\rho h)^{n+1/2,\star,\mathrm{pred}}$, for the conservative update of $(\rho h)^{(1)}$. There are a variety of quantities that we can predict to the interfaces here (controlled by `enthalpy_pred_type`—see Chapter 19). For the default case, we use the perturbational enthalpy equation, Eq. 2.28, neglecting reactions,

$$\frac{\partial(\rho h)'}{\partial t} = -\mathbf{U}\cdot\nabla(\rho h)' - (\rho h)'\nabla\cdot\mathbf{U} - \nabla\cdot\left[(\rho h)_0\widetilde{\mathbf{U}}\right] + \widetilde{\mathbf{U}}\cdot\nabla p_0. \tag{2.69}$$

to predict $(\rho h)' = (\rho h)^{(1)} - (\rho h)_0^n$ to time-centered edges, using $\mathbf{U} = \widetilde{\mathbf{U}}^{\mathrm{ADV},\star} + w_0^{n+1/2,\star}\mathbf{e}_r$, yielding $(\rho h)'^{n+1/2,\star,\mathrm{pred}}$. We convert the perturbational enthalpy to a full state enthalpy using

$$(\rho h)^{n+1/2,\star,\mathrm{pred}} = (\rho h)'^{n+1/2,\star,\mathrm{pred}} + \frac{(\rho h)_0^n + (\rho h)_0^{n+1,\star}}{2}. \tag{2.70}$$

For planar geometry, we map $(\rho h)_0$ directly to Cartesian edges. In spherical geometry, our experience has shown that a slightly different approach leads to reduced discretization errors. We first map $h_0 \equiv (\rho h)_0/\rho_0$ and ρ_0 to Cartesian edges separately, and then multiply these terms to get $(\rho h)_0$.

ii. Evolve $(\rho h)^{(1)} \rightarrow (\rho h)^{(2),\star}$.

For planar geometry,

$$\begin{aligned}(\rho h)^{(2),\star} &= (\rho h)^{(1)} \\ &\quad -\Delta t \left\{ \nabla \cdot \left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} + w_0^{n+1/2,\star} \mathbf{e}_r \right) (\rho h)^{n+1/2,\star,\mathrm{pred}} \right] \right\} \\ &\quad +\Delta t \left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} \cdot \mathbf{e}_r \right) \left(\frac{\partial p_0}{\partial r} \right)^n + \Delta t \psi^{n+1/2,\star}, \end{aligned} \tag{2.71}$$

For spherical geometry,

$$\begin{aligned}(\rho h)^{(2),\star} &= (\rho h)^{(1)} \\ &\quad -\Delta t \left\{ \nabla \cdot \left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} + w_0^{n+1/2,\star} \mathbf{e}_r \right) (\rho h)^{n+1/2,\star,\mathrm{pred}} \right] \right\} \\ &\quad +\Delta t \left\{ \nabla \cdot \left(\widetilde{\mathbf{U}}^{\mathrm{ADV},\star} p_0^n \right) - p_0^n \nabla \cdot \widetilde{\mathbf{U}}^{\mathrm{ADV},\star} \right\} \\ &\quad +\Delta t \psi^{n+1/2,\star}, \end{aligned} \tag{2.72}$$

Then, for each Cartesian cell where $\rho^{(2),\star} < \rho_{\mathrm{cutoff}}$, we recompute enthalpy using

$$(\rho h)^{(2),\star} = \rho^{(2),\star} h \left(\rho^{(2),\star}, p_0^{n+1,\star}, X_k^{(2),\star} \right). \tag{2.73}$$

This behavior is controlled by `do_eos_h_above_cutoff`.

I. Update the temperature using the equation of state: $T^{(2),\star} = T(\rho^{(2),\star}, h^{(2),\star}, X_k^{(2),\star})$ (planar geometry) or $T^{(2),\star} = T(\rho^{(2),\star}, p_0^{n+1,\star}, X_k^{(2),\star})$ (spherical geometry).

As before, this behavior is controlled by `use_tfromp`.

Step 5. *React the full state through a second time interval of* $\Delta t/2$.

Call **React State**$[\rho^{(2),\star}, (\rho h)^{(2),\star}, X_k^{(2),\star}, T^{(2),\star}, (\rho H_{\mathrm{ext}})^{(2),\star}, p_0^{n+1,\star}]$
$\rightarrow [\rho^{n+1,\star}, (\rho h)^{n+1,\star}, X_k^{n+1,\star}, T^{n+1,\star}, (\rho \dot{\omega}_k)^{(2),\star}, (\rho H_{\mathrm{nuc}})^{(2),\star}]$.

Step 6. *Compute the time-centered expansion,* $S^{n+1/2,\star}$*, base state velocity,* $w_0^{n+1/2}$*, and base state velocity forcing.*

A. Compute $S^{n+1/2,\star}$. First, compute $S^{n+1,\star}$ with

$$S^{n+1,\star} = -\sigma \sum_k \xi_k (\dot{\omega}_k)^{(2),\star} + \frac{1}{\rho^{n+1,\star} p_\rho} \sum_k p_{X_k} (\dot{\omega}_k)^{(2),\star} + \sigma {H_{\mathrm{nuc}}}^{(2),\star} + \sigma {H_{\mathrm{ext}}}^{(2),\star}, \tag{2.74}$$

where $(\dot{\omega}_k)^{(2),\star} = (\rho \dot{\omega}_k)^{(2),\star} / \rho^{(2),\star}$ and the thermodynamic quantities are defined using $\rho^{n+1,\star}$, $X_k^{n+1,\star}$, and $T^{n+1,\star}$ as inputs to the equation of state. If we are using diffusion then we would include the diffusion term in S. Then, define

$$\overline{S^{n+1/2,\star}} = \mathbf{Avg}(S^{n+1/2,\star}), \qquad S^{n+1/2,\star} = \frac{S^n + S^{n+1,\star}}{2}, \tag{2.75}$$

B. Compute $w_0^{n+1/2}$. First, define

$$\overline{\Gamma_1^{n+1/2,\star}} = \frac{\overline{\Gamma_1^n} + \overline{\Gamma_1^{n+1,\star}}}{2}, \quad \rho_0^{n+1/2,\star} = \frac{\rho_0^n + \rho_0^{n+1,\star}}{2}, \quad p_0^{n+1/2,\star} = \frac{p_0^n + p_0^{n+1,\star}}{2}, \tag{2.76}$$

with

$$\overline{\Gamma_1^{n+1,\star}} = \mathbf{Avg}\left[\Gamma_1\left(\rho^{n+1,\star}, p_0^{n+1,\star}, X_k^{n+1,\star}\right)\right]. \tag{2.77}$$

For planar geometry, call

Compute w_0 Planar$[\overline{S^{n+1/2,\star}}, \overline{\Gamma_1^{n+1/2,\star}}, p_0^{n+1/2,\star}, \psi^{n+1/2,\star}] \rightarrow [w_0^{n+1/2}]$.

For spherical geometry, call

Compute w_0 Spherical$[\overline{S^{n+1/2,\star}}, \overline{\Gamma_1^{n+1/2,\star}}, \rho_0^{n+1/2,\star}, p_0^{n+1/2,\star}, \eta_\rho^{n+1/2,\star}] \rightarrow [w_0^{n+1/2}]$.

C. Compute the base state velocity forcing. Rearrange equation (2.48),

$$\left(\frac{\beta_0}{\rho_0}\frac{\partial(\pi_0/\beta_0)}{\partial r}\right)^n = -\frac{w_0^{n+1/2} - w_0^{n-1/2}}{1/2(\Delta t^n + \Delta t^{n-1})} - w_0^n\left(\frac{\partial w_0}{\partial r}\right)^n, \tag{2.78}$$

where w_0^n and $(\partial w_0/\partial r)^n$ are defined as

$$w_0^n = \frac{\Delta t^n w_0^{n-1/2} + \Delta t^{n-1} w_0^{n+1/2}}{\Delta t^n + \Delta t^{n-1}}, \tag{2.79}$$

$$\left(\frac{\partial w_0}{\partial r}\right)^n = \frac{1}{\Delta t^n + \Delta t^{n-1}}\left[\Delta t^n\left(\frac{\partial w_0}{\partial r}\right)^{n-1/2} + \Delta t^{n-1}\left(\frac{\partial w_0}{\partial r}\right)^{n+1/2}\right]. \tag{2.80}$$

If $n = 0$, we use $\Delta t^{-1} = \Delta t^0$.

Step 7. *Construct the time-centered advective velocity on edges,* $\widetilde{\mathbf{U}}^{\mathrm{ADV}}$.

The procedure to construct $\widetilde{\mathbf{U}}^{\mathrm{ADV},\dagger}$ is identical to the procedure for computing $\widetilde{\mathbf{U}}^{\mathrm{ADV},\dagger,\star}$ in **Step 3**, but uses the updated values $w_0^{n+1/2}$ and π_0^n rather than $w_0^{n+1/2,\star}$ and $\pi_0^{n,\star}$. We note that $\widetilde{\mathbf{U}}^{\mathrm{ADV}}$ satisfies the discrete version of $\overline{(\widetilde{\mathbf{U}}^{\mathrm{ADV}} \cdot \mathbf{e}_r)} = 0$ as well as

$$\nabla \cdot \left(\beta_0^{n+1/2,\star}\widetilde{\mathbf{U}}^{\mathrm{ADV}}\right) = \beta_0^{n+1/2,\star}\left(S^{n+1/2,\star} - \overline{S^{n+1/2,\star}}\right), \tag{2.81}$$

$$\beta_0^{n+1/2,\star} = \frac{\beta_0^n + \beta_0^{n+1,\star}}{2}; \qquad \beta_0^{n+1,\star} = \beta_0\left(\rho_0^{n+1,\star}, p_0^{n+1,\star}, \overline{\Gamma_1^{n+1,\star}}\right). \tag{2.82}$$

Step 8. *Advect the base state and full state through a time interval of* Δt.

A. Update ρ_0, saving the time-centered density at radial edges by calling

Advect Base Density$[\rho_0^n, w_0^{n+1/2}] \rightarrow [\rho_0^{(2a)}, \rho_0^{n+1/2,\mathrm{pred}}]$.

B. Update (ρX_k). This step is identical to **Step 4B** except we use the updated values $w_0^{n+1/2}$, $\widetilde{\mathbf{U}}^{\mathrm{ADV}}$, and $\rho_0^{(2a)}$ rather than $w_0^{n+1/2,\star}$, $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$, and $\rho_0^{(2a),\star}$. In particular:

i. Compute the time-centered species edge states, $(\rho X_k)^{n+1/2,\text{pred}}$, for the conservative update of $(\rho X_k)^{(1)}$. We use equations (2.55) and (2.56) to predict $\rho'^{(1)} = \rho^{(1)} - \rho_0^n$ and $X_k^{(1)} = (\rho X_k)^{(1)}/\rho^{(1)}$ to time-centered edges with $\mathbf{U} = \widetilde{\mathbf{U}}^{\text{ADV}} + w_0^{n+1/2}\mathbf{e}_r$, yielding $\rho'^{n+1/2,\text{pred}}$ and $X_k^{n+1/2,\text{pred}}$. We convert the perturbational density to a full state density using

$$\rho^{n+1/2,\text{pred}} = \rho'^{n+1/2,\text{pred}} + \frac{\rho_0^n + \rho_0^{(2a)}}{2}. \tag{2.83}$$

Then, $(\rho X_k)^{n+1/2,\text{pred}} = \rho^{n+1/2,\text{pred}}\, X_k^{n+1/2,\text{pred}}$.

ii. Evolve $(\rho X_k)^{(1)} \rightarrow (\rho X_k)^{(2)}$ using

$$(\rho X_k)^{(2)} = (\rho X_k)^{(1)} - \Delta t \left\{ \nabla \cdot \left[\left(\widetilde{\mathbf{U}}^{\text{ADV}} + w_0^{n+1/2}\mathbf{e}_r \right) (\rho X_k)^{n+1/2,\text{pred}} \right] \right\}, \tag{2.84}$$

$$\rho^{(2)} = \sum_k (\rho X_k)^{(2)}, \qquad X_k^{(2)} = (\rho X_k)^{(2)}/\rho^{(2)}. \tag{2.85}$$

C. Define a radial edge-centered $\eta_\rho^{n+1/2}$.

For planar geometry,

$$\begin{aligned} \eta_\rho^{n+1/2} &= \mathbf{Avg} \sum_k \left[\left(\widetilde{\mathbf{U}}^{\text{ADV}} \cdot \mathbf{e}_r + w_0^{n+1/2} \right) (\rho X_k)^{n+1/2,\text{pred}} \right] \\ &\quad - w_0^{n+1/2} \rho_0^{n+1/2,\text{pred}}, \end{aligned} \tag{2.86}$$

For spherical geometry, first construct $\eta_\rho^{\text{cart},n+1/2} = [\rho'(\mathbf{U}\cdot\mathbf{e}_r)]^{n+1/2}$ on Cartesian cell centers using:

$$\eta_\rho^{\text{cart},n+1/2} = \left[\left(\frac{\rho^{(1)} + \rho^{(2)}}{2} \right) - \left(\frac{\rho_0^n + \rho_0^{(2a)}}{2} \right) \right] \left(\widetilde{\mathbf{U}}^{\text{ADV}} \cdot \mathbf{e}_r + w_0^{n+1/2} \right). \tag{2.87}$$

Then,

$$\eta_\rho^{n+1/2} = \mathbf{Avg} \left(\eta_\rho^{\text{cart},n+1/2} \right). \tag{2.88}$$

This gives a radial cell-centered $\eta_\rho^{n+1/2}$. To get $\eta_\rho^{n+1/2}$ at radial edges, average the two neighboring cell-centered values.

D. Correct ρ_0 by setting $\rho_0^{n+1} = \mathbf{Avg}(\rho^{(2)})$.

E. Update p_0 by calling **Enforce HSE**$[p_0^n, \rho_0^{n+1}] \rightarrow [p_0^{n+1}]$.

F. Compute $\psi^{n+1/2}$.

For planar geometry,

$$\psi_j^{n+1/2} = \frac{1}{2} \left(\eta_{\rho,j-1/2}^{n+1/2} + \eta_{\rho,j+1/2}^{n+1/2} \right) g. \tag{2.89}$$

For spherical geometry, first compute:

$$\overline{\Gamma_1^{(2)}} = \mathbf{Avg} \left[\Gamma_1 \left(\rho^{(2)}, p_0^{n+1}, X_k^{(2)} \right) \right]. \tag{2.90}$$

Then, define $\psi^{n+1/2}$ using equation (2.44):

$$\psi_j^{n+1/2} = \left(\frac{\overline{\Gamma_1^{(1)}} + \overline{\Gamma_1^{(2)}}}{2}\right)_j \left(\frac{p_0^n + p_0^{n+1}}{2}\right)_j \left\{\overline{S_j^{n+1/2}} - \frac{1}{r_j^2}\left[\left(r^2 w_0^{n+1/2}\right)_{j+1/2} - \left(r^2 w_0^{n+1/2}\right)_{j-1/2}\right]\right\}. \quad (2.91)$$

G. Update $(\rho h)_0$ by calling **Advect Base Enthalpy**$[(\rho h)_0^n, w_0^{n+1/2}, \psi^{n+1/2}] \rightarrow [(\rho h)_0^{n+1}]$.

H. Update the enthalpy. This step is identical to **Step 4H** except we use the updated values $w_0^{n+1/2}, \widetilde{\mathbf{U}}^{\mathrm{ADV}}, \rho_0^{n+1}, (\rho h)_0^{n+1}, p_0^{n+1/2}$, and $\psi^{n+1/2}$ rather than $w_0^{n+1/2,\star}, \widetilde{\mathbf{U}}^{\mathrm{ADV},\star}, \rho_0^{n+1,\star}, (\rho h)_0^{n+1,\star}, p_0^n$, and $\psi^{n+1/2,\star}$. In particular:

i. Compute the time-centered enthalpy edge state, $(\rho h)^{n+1/2,\mathrm{pred}}$, for the conservative update of $(\rho h)^{(1)}$. We use equation (2.69) to predict $(\rho h)' = (\rho h)^{(1)} - (\rho h)_0^n$ to time-centered edges with $\mathbf{U} = \widetilde{\mathbf{U}}^{\mathrm{ADV}} + w_0^{n+1/2}\mathbf{e}_r$, yielding $(\rho h)'^{n+1/2,\mathrm{pred}}$. We convert the perturbational enthalpy to a full state enthalpy using

$$(\rho h)^{n+1/2,\mathrm{pred}} = (\rho h)'^{n+1/2,\mathrm{pred}} + \frac{(\rho h)_0^n + (\rho h)_0^{n+1}}{2}. \quad (2.92)$$

ii. Evolve $(\rho h)^{(1)} \rightarrow (\rho h)^{(2)}$.

For planar geometry,

$$(\rho h)^{(2)} = (\rho h)^{(1)} - \Delta t\left\{\nabla\cdot\left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV}} + w_0^{n+1/2}\mathbf{e}_r\right)(\rho h)^{n+1/2,\mathrm{pred}}\right]\right\} + \Delta t\left(\widetilde{\mathbf{U}}^{\mathrm{ADV}}\cdot\mathbf{e}_r\right)\left(\frac{\partial p_0}{\partial r}\right)^{n+1/2} + \Delta t\psi^{n+1/2}, \quad (2.93)$$

For spherical geometry,

$$(\rho h)^{(2)} = (\rho h)^{(1)} - \Delta t\left\{\nabla\cdot\left[\left(\widetilde{\mathbf{U}}^{\mathrm{ADV}} + w_0^{n+1/2}\mathbf{e}_r\right)(\rho h)^{n+1/2,\mathrm{pred}}\right]\right\} + \Delta t\left[\nabla\cdot\left(\widetilde{\mathbf{U}}^{\mathrm{ADV}} p_0^{n+1/2}\right) - p_0^{n+1/2}\nabla\cdot\widetilde{\mathbf{U}}^{\mathrm{ADV}}\right] + \Delta t\psi^{n+1/2}, \quad (2.94)$$

where $p_0^{n+1/2}$ is defined as $p_0^{n+1/2} = (p_0^n + p_0^{n+1})/2$.

Then, for each Cartesian cell where $\rho^{(2)} < \rho_{\mathrm{cutoff}}$, we recompute enthalpy using

$$(\rho h)^{(2)} = \rho^{(2)} h\left(\rho^{(2)}, p_0^{n+1}, X_k^{(2)}\right). \quad (2.95)$$

I. Update the temperature using the equation of state: $T^{(2)} = T(\rho^{(2)}, h^{(2)}, X_k^{(2)})$ (planar geometry) or $T^{(2)} = T(\rho^{(2)}, p_0^{n+1}, X_k^{(2)})$ (spherical geometry).

Again, the actual inputs depend of `use_tfromp`.

Step 9. *React the full state through a second time interval of* $\Delta t/2$.

Call **React State**$[\rho^{(2)}, (\rho h)^{(2)}, X_k^{(2)}, T^{(2)}, (\rho H_{\rm ext})^{(2)}, p_0^{n+1}]$
$\rightarrow [\rho^{n+1}, (\rho h)^{n+1}, X_k^{n+1}, T^{n+1}, (\rho\dot{\omega}_k)^{(2)}, (\rho H_{\rm nuc})^{(2)}]$.

Step 10. *Define the new time expansion,* S^{n+1}, *and* $\overline{\Gamma_1^{n+1}}$.

A. Define

$$S^{n+1} = -\sigma \sum_k \xi_k (\dot{\omega}_k)^{(2)} + \sigma {H_{\rm nuc}}^{(2)} + \frac{1}{\rho^{n+1} p_\rho} \sum_k p_{X_k} (\dot{\omega}_k)^{(2)} + \sigma {H_{\rm ext}}^{(2)}, \tag{2.96}$$

where $(\dot{\omega}_k)^{(2)} = (\rho\dot{\omega}_k)^{(2)}/\rho^{(2)}$ and the thermodynamic quantities are defined using ρ^{n+1}, X_k^{n+1}, and T^{n+1} as inputs to the equation of state. If we are doing thermal diffusion (`use_thermal_diffusion= T`) then we also include the diffusive term in S. Then, compute

$$\overline{S^{n+1}} = \mathbf{Avg}(S^{n+1}). \tag{2.97}$$

B. Define

$$\overline{\Gamma_1^{n+1}} = \mathbf{Avg}\left[\Gamma_1\left(\rho^{n+1}, p_0^{n+1}, X_k^{n+1}\right)\right]. \tag{2.98}$$

Step 11. *Update the velocity.*

First, we compute the time-centered edge velocities, $\widetilde{\mathbf{U}}^{n+1/2,\rm pred}$. Then, we define

$$\rho^{n+1/2} = \frac{\rho^n + \rho^{n+1}}{2}, \qquad \rho_0^{n+1/2} = \frac{\rho_0^n + \rho_0^{n+1}}{2}. \tag{2.99}$$

We update the velocity field $\widetilde{\mathbf{U}}^n$ to $\widetilde{\mathbf{U}}^{n+1,\dagger}$ by discretizing equation (2.18) as

$$\begin{aligned}
\widetilde{\mathbf{U}}^{n+1,\dagger} \;=\; & \widetilde{\mathbf{U}}^n - \Delta t \left[\left(\widetilde{\mathbf{U}}^{\rm ADV} + w_0^{n+1/2}\mathbf{e}_r\right)\cdot\nabla\widetilde{\mathbf{U}}^{n+1/2,\rm pred}\right] \\
& -\Delta t \left(\widetilde{\mathbf{U}}^{\rm ADV}\cdot\mathbf{e}_r\right)\left(\frac{\partial w_0}{\partial r}\right)^{n+1/2}\mathbf{e}_r \\
& +\Delta t\left[-\frac{\beta_0^{n+1/2}}{\rho^{n+1/2}}\mathbf{G}\left(\frac{\pi}{\beta_0}\right)^{n-1/2} + \left(\frac{\beta_0}{\rho_0}\frac{\partial(\pi_0/\beta_0)}{\partial r}\right)^n \mathbf{e}_r - \frac{\left(\rho^{n+1/2}-\rho_0^{n+1/2}\right)}{\rho^{n+1/2}} g^{n+1/2}\mathbf{e}_r\right],
\end{aligned} \tag{2.100}$$

where $\mathbf{G}$ approximates a cell-centered gradient from nodal data. Again, the † superscript refers to the fact that the updated velocity does not satisfy the divergence constraint.

Finally, we use an approximate nodal projection to define $\widetilde{\mathbf{U}}^{n+1}$ from $\widetilde{\mathbf{U}}^{n+1,\dagger}$, such that $\widetilde{\mathbf{U}}^{n+1}$ approximately satisfies

$$\nabla\cdot\left(\beta_0^{n+1/2}\widetilde{\mathbf{U}}^{n+1}\right) = \beta_0^{n+1/2}\left(S^{n+1} - \overline{S^{n+1}}\right), \tag{2.101}$$

where $\beta_0^{n+1/2}$ is defined as

$$\beta_0^{n+1/2} = \frac{\beta_0^n + \beta_0^{n+1}}{2}; \qquad \beta_0^{n+1} = \beta\left(\rho_0^{n+1}, p_0^{n+1}, \overline{\Gamma_1^{n+1}}, g^{n+1}\right). \tag{2.102}$$

As part of the projection we also define the new-time perturbational pressure, $\pi^{n+1/2}$. This projection necessarily differs from the MAC projection used in **Step 3** and **Step 7** because the velocities in those steps are defined on edges and $\widetilde{\mathbf{U}}^{n+1}$ is defined at cell centers, requiring different divergence and gradient operators. Details of the approximate projection are given in Paper III.

Step 12. *Compute a new* Δt.

Compute Δt for the next time step with the procedure described in §3.4 of Paper III using w_0 as computed in **Step 6** and $\widetilde{\mathbf{U}}^{n+1}$ as computed in **Step 11**.

This completes one step of the algorithm.

Volume Discrepancy Changes

Chapter 14 describes the reasoning behind the volume discrepancy term—a forcing term added to the constraint equation to bring us back to the equation of state. This addition of this term (enabled with `dpdt_factor= T`) modifies our equation set in the following way:

- In **Step 2B**, to compute w_0, we need to account for the volume discrepancy term by first defining $p_{\text{EOS}}^n = \overline{p(\rho, h, X_k)^n}$, and then using:

$$\frac{\partial w_0^{n+1/2,\star}}{\partial r} = \overline{S^{n+1/2,\star\star}} - \frac{1}{\overline{\Gamma_1^n} p_0^n}\psi^{n-1/2} - \underbrace{\frac{f}{\overline{\Gamma_1^n} p_0^n}\left(\frac{p_0^n - \overline{p_{\text{EOS}}^n}}{\Delta t}\right)}_{\delta\chi_{w_0}}. \tag{2.103}$$

- In **Step 3**, the MAC projection should account for the volume discrepancy term:

$$\nabla\cdot\left(\beta_0^n \widetilde{\mathbf{U}}^{\text{ADV},\star}\right) = \beta_0^n\left[\left(S^{n+1/2,\star\star} - \overline{S^{n+1/2,\star\star}}\right) + \underbrace{\frac{f}{\overline{\Gamma_1^n} p_0^n}\left(\frac{p_{\text{EOS}}^n - \overline{p_{\text{EOS}}^n}}{\Delta t^n}\right)}_{\delta\chi}\right]. \tag{2.104}$$

- In **Step 6B**, to compute w_0, we need to account for the volume discrepancy term by first defining $p_{\text{EOS}}^{n+1,\star} = p(\rho, h, X_k)^{n+1,\star}$, $\overline{\Gamma_1^{n+1/2,\star}} = (\overline{\Gamma_1^n} + \overline{\Gamma_1^{n+1,\star}})/2$, and $p^{n+1/2,\star} = (p^n + p^{n+1,\star})/2$, and then using:

$$\frac{\partial w_0^{n+1/2}}{\partial r} = \overline{S^{n+1/2,\star}} - \frac{1}{\overline{\Gamma_1^{n+1/2,\star}} p_0^{n+1/2,\star}}\psi^{n+1/2,\star} - \frac{f}{\overline{\Gamma_1^{n+1,\star}} p_0^{n+1,\star}}\left(\frac{p_0^{n+1,\star} - \overline{p_{\text{EOS}}^{n+1,\star}}}{\Delta t}\right) - \delta\chi_{w_0} \tag{2.105}$$

- In **Step 7**, the MAC projection should account for the volume discrepancy term:

$$\nabla \cdot \left(\beta_0^{n+1/2,\star}\widetilde{\mathbf{U}}^{\mathrm{ADV}}\right) = \beta_0^{n+1/2,\star}\left[\left(S^{n+1/2,\star} - \overline{S^{n+1/2,\star}}\right) + \frac{f}{\overline{\Gamma_1^{n+1,\star}}p_0^{n+1,\star}}\left(\frac{p_{\mathrm{EOS}}^{n+1,\star} - \overline{p_{\mathrm{EOS}}^{n+1,\star}}}{\Delta t^n}\right) + \delta\chi\right], \tag{2.106}$$

  where $p(\rho,h,X_k)^{n+1/2,\star} = \left[p(\rho,h,X_k)^n + p(\rho,h,X_k)^{n+1,\star}\right]/2$.

- In **Step 11**, the approximate projection should account for the volume discrepancy term:

$$\nabla \cdot \left(\beta_0^{n+1/2}\widetilde{\mathbf{U}}^{n+1}\right) = \beta_0^{n+1/2}\left\{\left(S^{n+1} - \overline{S^{n+1}}\right) + \frac{f}{\overline{\Gamma_1^{n+1}}p_0^{n+1}}\left[\frac{p(\rho,h,X_k)^{n+1} - \overline{p(\rho,h,X_k)^{n+1}}}{\Delta t^n}\right]\right\}. \tag{2.107}$$

Γ_1 Variation Changes

The constraint we derive from requiring the pressure to be close to the background hydrostatic pressure takes the form:

$$\nabla\cdot\mathbf{U} + \frac{1}{\Gamma_1 p_0}\frac{Dp_0}{Dt} = S \ . \tag{2.108}$$

The default MAESTRO algorithm replaces Γ_1 with $\overline{\Gamma}_1$, allowing us to write this as a divergence constraint. In paper III, we explored the effects of localized variations in Γ_1 by writing $\Gamma_1 = \overline{\Gamma}_1 + \delta\Gamma_1$. This gives us:

$$\nabla\cdot\mathbf{U} + \frac{1}{\left(\overline{\Gamma}_1 + \delta\Gamma_1\right)\,p_0}\mathbf{U}\cdot\nabla p_0 = S - \frac{1}{\left(\overline{\Gamma}_1 + \delta\Gamma_1\right)\,p_0}\frac{\partial p_0}{\partial t} \ . \tag{2.109}$$

Assuming that $\delta\Gamma_1 \ll \overline{\Gamma}_1$, we then have

$$\nabla\cdot\mathbf{U} + \frac{1}{\overline{\Gamma}_1 p_0}\mathbf{U}\cdot\nabla p_0\left[1 - \frac{\delta\Gamma_1}{\overline{\Gamma}_1} + \frac{(\delta\Gamma_1)^2}{\overline{\Gamma}_1^2}\right] = S - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t}\left[1 - \frac{\delta\Gamma_1}{\overline{\Gamma}_1} + \frac{(\delta\Gamma_1)^2}{\overline{\Gamma}_1^2}\right] \tag{2.110}$$

Grouping by order of the correction, we have

$$\nabla\cdot\mathbf{U} + \frac{1}{\overline{\Gamma}_1 p_0}\mathbf{U}\cdot\nabla p_0 = S - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} + \underbrace{\frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\left[\frac{\partial p_0}{\partial t} + \mathbf{U}\cdot\nabla p_0\right]}_{\text{first order corrections}} - \underbrace{\frac{(\delta\Gamma_1)^2}{\overline{\Gamma}_1^3 p_0}\left[\frac{\partial p_0}{\partial t} + \mathbf{U}\cdot\nabla p_0\right]}_{\text{second order corrections}} \ , \tag{2.111}$$

Keeping to First Order in $\delta\Gamma_1$

The base state evolution equation is the average of Eq. 2.111 over a layer

$$\nabla\cdot w_0\mathbf{e}_r + \frac{1}{\overline{\Gamma}_1 p_0}w_0\mathbf{e}_r\cdot\nabla p_0 = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} + \overline{\left(\frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0\right)} \ . \tag{2.112}$$

where we see that the $[\delta\Gamma_1/(\Gamma_1^2 p_0)]\partial p_0/\partial t$ terms averages to zero, since the average of $\delta\Gamma_1$ term is zero. Subtracting this from equation (2.111), we have

$$\nabla\cdot\widetilde{\mathbf{U}} + \frac{1}{\overline{\Gamma}_1 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0 = S - \overline{S} + \frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\left(\psi + \widetilde{\mathbf{U}}\cdot\nabla p_0\right) - \overline{\left(\frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0\right)} \ . \tag{2.113}$$

These can be written more compactly as:

$$\frac{\partial w_0}{\partial r} = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\psi + \overline{\left(\frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0\right)} \quad , \tag{2.114}$$

for plane-parallel geometries (analogous to Eq. 2.21), and

$$\nabla\cdot(\beta_0\widetilde{\mathbf{U}}) = \beta_0\left[S - \overline{S} + \frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\psi + \frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0 - \overline{\left(\frac{\delta\Gamma_1}{\overline{\Gamma}_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0\right)}\right] \quad , \tag{2.115}$$

This constraint is not in a form that can be projected. To solve this form, we need to use a lagged $\widetilde{\mathbf{U}}$ in the righthand side.

This change comes into MAESTRO in a variety of steps, summarized here. To enable this portion of the algorithm, set `use_delta_gamma1_term = T`.

- In **Step 3**, we are doing the "predictor" portion of the MAESTRO algorithm, getting the MAC velocity that satisfies the constraint, so we do not try to incorporate the $\delta\Gamma_1$ effect. We set all the $\delta\Gamma_1$ terms in Eq. 2.115 to zero.

- In **Step 6**, we are computing the new time-centered source, $S^{n+1/2,\star}$ and the base state velocity, $w_0^{n+1/2}$. Now we can incorporate the $\delta\Gamma_1$ effect. First we construct:

$$\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0 \approx \frac{\Gamma_1^{n+1,\star} - \overline{\Gamma_1^{n+1,\star}}}{\overline{\Gamma_1^{n+1,\star}}^2}\frac{1}{p_0^n}\widetilde{\mathbf{U}}^n\cdot\nabla p_0^n \tag{2.116}$$

  Then we call **average** to construct the lateral average of this

$$\overline{\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0} = \mathbf{Avg}\left(\frac{\Gamma_1^{n+1,\star} - \overline{\Gamma_1^{n+1,\star}}}{\overline{\Gamma_1^{n+1,\star}}^2}\frac{1}{p_0^n}\widetilde{\mathbf{U}}^n\cdot\nabla p_0^n\right) \tag{2.117}$$

  Since the average of this is needed in advancing w_0, we modify $\overline{S}$ to include this average:

$$\overline{S^{n+1/2,\star}} \leftarrow \overline{S^{n+1/2,\star}} + \overline{\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0} \tag{2.118}$$

- In **Step 7**, we now include the $\delta\Gamma_1$ term in the righthand side for the constraint by solving:

$$\nabla\cdot\left(\beta_0^{n+1/2,\star}\widetilde{\mathbf{U}}^{\mathrm{ADV}}\right) = \beta_0^{n+1/2,\star}\left(S^{n+1/2,\star} - \overline{S^{n+1/2,\star}} + \frac{\Gamma_1^{n+1,\star} - \overline{\Gamma_1^{n+1,\star}}}{\overline{\Gamma_1^{n+1,\star}}^2}\frac{1}{p_0^n}(\psi^{n+1/2,\star} + \widetilde{\mathbf{U}}^n\cdot\nabla p_0^n)\right) \tag{2.119}$$

  We note that this includes the average of the correction term as shown in Eq. 2.115 because we modified $\bar{S}$ to include this already.

- In **Step 10**, we do a construction much like that done in **Step 6**, but with the time-centerings of some of the quantities changed. First we construct:

$$\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0 \approx \frac{\Gamma_1^{n+1}-\overline{\Gamma_1^{n+1}}}{\overline{\Gamma_1^{n+1}}^2}\frac{1}{p_0^{n+1}}\widetilde{\mathbf{U}}^n\cdot\nabla p_0^{n+1} \tag{2.120}$$

  Then we call **average** to construct the lateral average of this

$$\overline{\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0} = \mathbf{Avg}\left(\frac{\Gamma_1^{n+1}-\overline{\Gamma_1^{n+1}}}{\overline{\Gamma_1^{n+1}}^2}\frac{1}{p_0^{n+1}}\widetilde{\mathbf{U}}^n\cdot\nabla p_0^{n+1}\right) \tag{2.121}$$

  Again we modify $\overline{S}$ to include this average:

$$\overline{S^{n+1}} \leftarrow \overline{S^{n+1}} + \overline{\frac{\delta\Gamma_1}{\Gamma_1^2 p_0}\widetilde{\mathbf{U}}\cdot\nabla p_0} \tag{2.122}$$

- In **Step 11**, we modify the source of the constraint to include the $\delta\Gamma_1$ information. In particular, we solve:

$$\nabla\cdot\left(\beta_0^{n+1/2}\widetilde{\mathbf{U}}^{n+1}\right) = \beta_0^{n+1/2}\left(S^{n+1}-\overline{S^{n+1}}+\frac{\Gamma_1^{n+1}-\overline{\Gamma_1^{n+1}}}{\overline{\Gamma_1^{n+1}}^2}\frac{1}{p_0^{n+1}}(\psi^{n+1/2}+\widetilde{\mathbf{U}}^n\cdot\nabla p_0^{n+1})\right) \tag{2.123}$$

Thermal Diffusion Changes

Thermal diffusion was introduced in the XRB paper [25]. This introduces a new term to S as well as the enthalpy equation. Treating the enthalpy equation now requires a parabolic solve. We describe that process here.

Immediately after **Step 4H**, diffuse the enthalpy through a time interval of Δt. First, define $(\rho h)^{(1a),\star} = (\rho h)^{(2),\star}$. We recompute $(\rho h)^{(2),\star}$ to account for thermal diffusion. Here we begin with the enthalpy equation, but consider only the diffusion terms,

$$\frac{\partial(\rho h)}{\partial t} = \nabla\cdot k_{\rm th}\nabla T. \tag{2.124}$$

We can recast this as an enthalpy-diffusion equation by applying the chain-rule to $h(p_0, T, X_k)$,

$$\nabla h = h_p\nabla p_0 + c_p\nabla T + \sum_k \xi_k\nabla X_k \ , \tag{2.125}$$

giving

$$\frac{\partial(\rho h)}{\partial t} = \nabla\cdot\frac{k_{\rm th}}{c_p}\nabla h - \sum_k \nabla\cdot\frac{\xi_k k_{\rm th}}{c_p}\nabla X_k - \nabla\cdot\frac{h_p k_{\rm th}}{c_p}\nabla p_0. \tag{2.126}$$

Compute $k_{\rm th}^{(1)}, c_p^{(1)}$, and $\xi_k^{(1)}$ from $\rho^{(1)}, T^{(1)}$, and $X_k^{(1)}$ as inputs to the equation of state. The update is given by

$$\begin{aligned}(\rho h)^{(2),\star} &= (\rho h)^{(1a),\star} + \frac{\Delta t}{2}\nabla\cdot\left(\frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla h^{(2),\star} + \frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla h^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\sum_k \nabla\cdot\left(\frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(2),\star} + \frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\nabla\cdot\left(\frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n+1,\star} + \frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n}\right),\end{aligned} \tag{2.127}$$

which is numerically implemented as a diffusion equation for $h^{(2),\star}$,

$$\begin{aligned}\left(\rho^{(2),\star} - \frac{\Delta t}{2}\nabla\cdot\frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla\right)h^{(2),\star} &= (\rho h)^{(1a),\star} + \frac{\Delta t}{2}\nabla\cdot\frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla h^{(1)}\\ &\quad -\frac{\Delta t}{2}\sum_k \nabla\cdot\left(\frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(2),\star} + \frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\nabla\cdot\left(\frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n+1,\star} + \frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n}\right),\end{aligned} \tag{2.128}$$

Immediately after **Step 8H**, diffuse the enthalpy through a time interval of Δt. First, define $(\rho h)^{(1a)} = (\rho h)^{(2)}$. We recompute $(\rho h)^{(2)}$ to account for thermal diffusion. Compute $k_{\rm th}^{(2),\star}, c_p^{(2),\star}$, and $\xi_k^{(2),\star}$, from $\rho^{(2),\star}, T^{(2),\star}$, and $X_k^{(2),\star}$ as inputs to the equation of state. The update is given by

$$\begin{aligned}(\rho h)^{(2)} &= (\rho h)^{(1a)} + \frac{\Delta t}{2}\nabla\cdot\left(\frac{k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla h^{(2)} + \frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla h^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\sum_k \nabla\cdot\left(\frac{\xi_k^{(2),\star}k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla X_k^{(2)} + \frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\nabla\cdot\left(\frac{h_p^{(2),\star}k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla p_0^{n+1} + \frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n}\right),\end{aligned} \tag{2.129}$$

which is numerically implemented as a diffusion equation for $h^{(2)}$.

$$\begin{aligned}\left(\rho^{(2)} - \frac{\Delta t}{2}\nabla\cdot\frac{k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla\right)h^{(2)} &= (\rho h)^{(1a)} + \frac{\Delta t}{2}\nabla\cdot\frac{k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla h^{(1)}\\ &\quad -\frac{\Delta t}{2}\sum_k \nabla\cdot\left(\frac{\xi_k^{(2),\star}k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla X_k^{(2)} + \frac{\xi_k^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla X_k^{(1)}\right)\\ &\quad -\frac{\Delta t}{2}\nabla\cdot\left(\frac{h_p^{(2),\star}k_{\rm th}^{(2),\star}}{c_p^{(2),\star}}\nabla p_0^{n+1} + \frac{h_p^{(1)}k_{\rm th}^{(1)}}{c_p^{(1)}}\nabla p_0^{n}\right),\end{aligned} \tag{2.130}$$

Initialization

We start each calculation with user-specified initial values for ρ, X_k and T, as well as an initial background state. In order for the low Mach number assumption to hold, the initial data must be thermodynamically consistent with the initial background state. In addition, the initial velocity field must satisfy an initial approximation to the divergence constraint. We use an iterative procedure to compute both an initial right-hand-side for the constraint equation and an initial velocity field that satisfies the constraint. The user specifies the number of iterations, $N_{\rm iters}^S$, in this first step of the initialization procedure.

The initial perturbational pressure also needs to be determined for use in **Steps 3**, **7**, and **11**. This is done through a second iterative procedure which follows the time advancement algorithm as described in **Steps 1-11** in §2.2. The user specifies the number of iterations, $N_{\rm iters}^\pi$, in this second step of the initialization procedure. The details for both iterations are given below.

Step 0. *Initialization*

First, we need to construct approximations to $S^0, w_0^{-1/2}, \Delta t^0$, and $\mathbf{U}^0$. Start with initial data $X_k^{\rm init}, \rho^{\rm init}$, $T^{\rm init}$, an initial base state, and an initial guess for the velocity, $\mathbf{U}^{\rm init}$. Set $w_0^0 = 0$ as an initial approximation. Use the equation of state to determine $(\rho h)^{\rm init}$. Compute β_0^0 as a function of the initial data. The next part of the initialization process proceeds as follows.

a. *Initial Projection*: if `do_initial_projection = T`, then we first project the velocity field with $\rho = 1$ and β_0^0. The initial projection does not see reactions or external heating, and thus we set $\dot\omega = H_{\rm nuc} = H_{\rm ext} = 0$ in S. The reason for ignoring reactions and heating is that we need some kind of time scale over which to compute the effect of reactions, but we first need an estimate of the velocity field in order to derive the time step that will be used as a time scale. The elliptic equation we solve is

$$\nabla \cdot \beta_0^0 \nabla \phi = \underbrace{\beta_0^0 (S - \overline{S})}_{\text{0 except for diffusion}} - \nabla \cdot (\beta_0^0 \mathbf{U}^{\rm init}) \tag{2.131}$$

This ϕ is then used to correct the velocity field to obtain $\mathbf{U}^{0,0}$. If `do_initial_projection = F`, set $\mathbf{U}^{0,0} = \mathbf{U}^{\rm init}$.

b. *"Divu" iterations*: Next we do `init_divu_iter` iterations to project the velocity field using a constraint that sees reactions and external heating. The initial timestep estimate is provided by `firstdt` and $\mathbf{U}^{0,0}$, to allow us to compute the effect of reactions over $\Delta t/2$.

Do $\nu = 1, ..., N_{\rm iters}^S$.

i. Estimate Δt^ν using $\mathbf{U}^{0,\nu-1}$ and $w_0^{\nu-1}$.

ii. **React State**$[\rho^{\rm init}, (\rho h)^{\rm init}, X_k^{\rm init}, T^{\rm init}, (\rho^{\rm init} H_{\rm ext}), p_0^{\rm init}] \rightarrow [\rho^{\rm out}, (\rho h)^{\rm out}, X_k^{\rm out}, T^{\rm out}, (\rho \dot\omega_k)^{0,\nu}]$.

iii. Compute $S^{0,\nu}$ from equation **(??)** using $(\rho\dot\omega_k)^{0,\nu}$ and the initial data.

iv. Compute $\overline{S^{0,\nu}} = \mathbf{Avg}(S^{0,\nu})$.

v. Compute w_0^ν as in **Step 2B** using $\overline{S^{0,\nu}}$ and $\psi = 0$.

vi. Project $\mathbf{U}^{0,\nu-1}$ using β_0^0 and $(S^{0,\nu} - \overline{S^{0,\nu}})$ as the source term. This yields $\mathbf{U}^{0,\nu}$. In this projection, again the density is set to 1, and the elliptic equation we solve is:

$$\nabla \cdot \beta_0^0 \nabla \phi = \beta_0 (S - \overline{S}) - \nabla \cdot (\beta_0^0 \mathbf{U}^{0,\nu-1}) \tag{2.132}$$

End do.

Define $S^0 = S^{0,N_{\rm iters}^S}$, $w_0^{-1/2} = w_0^{N_{\rm iters}^S}$, $\Delta t^0 = \Delta t^{N_{\rm iters}^S}$, and $\mathbf{U}^0 = \mathbf{U}^{0,N_{\rm iters}^S}$.

Next, we need to construct approximations to $\eta_\rho^{-1/2}$, $\psi^{-1/2}$, S^1, and $\pi^{-1/2}$. As initial approximations, set $\eta_\rho^{-1/2} = 0$, $\psi^{-1/2} = 0$, $S^{1,0} = S^0$, and $\pi^{-1/2} = 0$.

c. *Pressure iterations*: Here we do `init_iter` iterations to get an approximation for the lagged pressure:

Do $\nu = 1, ..., N_{\rm iters}^\pi$.

i. Perform **Steps 1-11** as described above, using $S^{1/2,\star\star} = (S^0 + S^{1,\nu-1})/2$ in **Step 2** as described. The only other difference in the time advancement is that in **Step 11** we define $\mathbf{V} = (\widetilde{\mathbf{U}}^{1,\star} - \widetilde{\mathbf{U}}^0)$ and solve

$$L_\beta^\rho \phi = D\left(\beta_0^{1/2}\mathbf{V}\right) - \beta_0^{1/2}\left[\left(S^1 - \overline{S^1}\right) - \left(S^0 - \overline{S^0}\right)\right] \quad . \tag{2.133}$$

(The motivation for this form of the projection in the initial pressure iterations is discussed in [3].) We discard the new velocity resulting from this, but keep the new value for $\pi^{1/2} = \pi^{-1/2} + (1/\Delta t)\ \phi$. These steps also yield new scalar data at time Δt, which we discard, and new values for $\eta_\rho^{1/2}$ (**Step 8C**), $\psi^{1/2}$ (**Step 8F**), $S^{1,\nu}$ (**Step 10A**), and $\pi^{1/2}$ (**Step 11**), which we keep.

ii. Set $\pi^{-1/2} = \pi^{1/2}$, $\eta_\rho^{-1/2} = \eta_\rho^{1/2}$, and $\psi^{-1/2} = \psi^{1/2}$.

End do.

Finally, we define $S^1 = S^{1,N_{\rm iters}^\pi}$.

The tolerances for these elliptic solves are described in § 22.4.1.

Changes from Earlier Implementations

Changes Between Paper 3 and Paper 4

1. We defined the mapping of data between a 1D radial array and the 3D Cartesian grid for spherical problems (which we improve upon in the multilevel paper).
2. We update T after the call to **React State**.
3. We have created a `burning_cutoff_density`, where the burning does not happen below this density. It is presently set to `base_cutoff_density`.
4. Use corner coupling in advection.
5. We have an option, controlled by `use_tfromp`, to update temperature using $T = T(\rho, X_k, p_0)$ rather than $T = T(\rho, h, X_k)$. The former completely decouples enthalpy from our system. For spherical problems, we use `use_tfromp = TRUE`, for planar problems, we use `use_tfromp = FALSE`.

6. For spherical problems, we have changed the discretization of $\widetilde{\mathbf{U}} \cdot \nabla p_0$ in the enthalpy update to $\nabla \cdot (\widetilde{\mathbf{U}} p_0) - p_0 \nabla \cdot \widetilde{\mathbf{U}}$.
7. In paper III we discretized the enthalpy evolution equation in terms of T. Since then we have discovered that discretizing the enthalpy evolution in perturbational form, $(\rho h)'$, leads to better numerical properties. We use `enthalpy_pred_type= 1`. This is more like paper II.
8. We have turned off the evolution of h above the atmosphere and instead compute h with the EOS using `do_eos_h_above_cutoff = T`.

Changes Between Paper 4 and the Multilevel Paper

See the multilevel paper for the latest.

Changes Between the Multilevel Paper and Paper 5 [38]

1. Added rotation.

Changes Between Paper 5 and the XRB Paper

1. We have added thermal diffusion, controlled by `use_thermal_diffusion`, `temp_diffusion_formulation`, and `thermal_diffusion_type`.
2. We added the volume discrepancy term to the velocity constraint equation, controlled by the input parameter, `dpdt_factor`.
3. For certain problems, we need to set `do_eos_h_above_cutoff = F` to prevent large, unphysical velocities from appearing near the edge of the star.

Changes Since the XRB Paper

1. We switched to the new form of the momentum equation to Eq. 2.15 to conserve the low-Mach number form of energy.
2. We changed the form of the volume discrepancy term to get better agreement between the two temperatures.

Future Considerations

- Should we use a predictor-corrector for updating the full-state density? Specifically, after calling **Correct Base**, should we do a full-state density advance and **Correct Base** using the more accurate estimate of ρ_0^{n+1}?
- We are still exploring the effects of `use_tfromp = F` for spherical problems. We would eventually like to run in this mode, but $T = T(\rho, X_k, p_0)$ and $T = T(\rho, h, X_k)$ drift away from each other more than we would like. Our attempts at incorporating a `dpdt_factor` for spherical problems have not been successful.

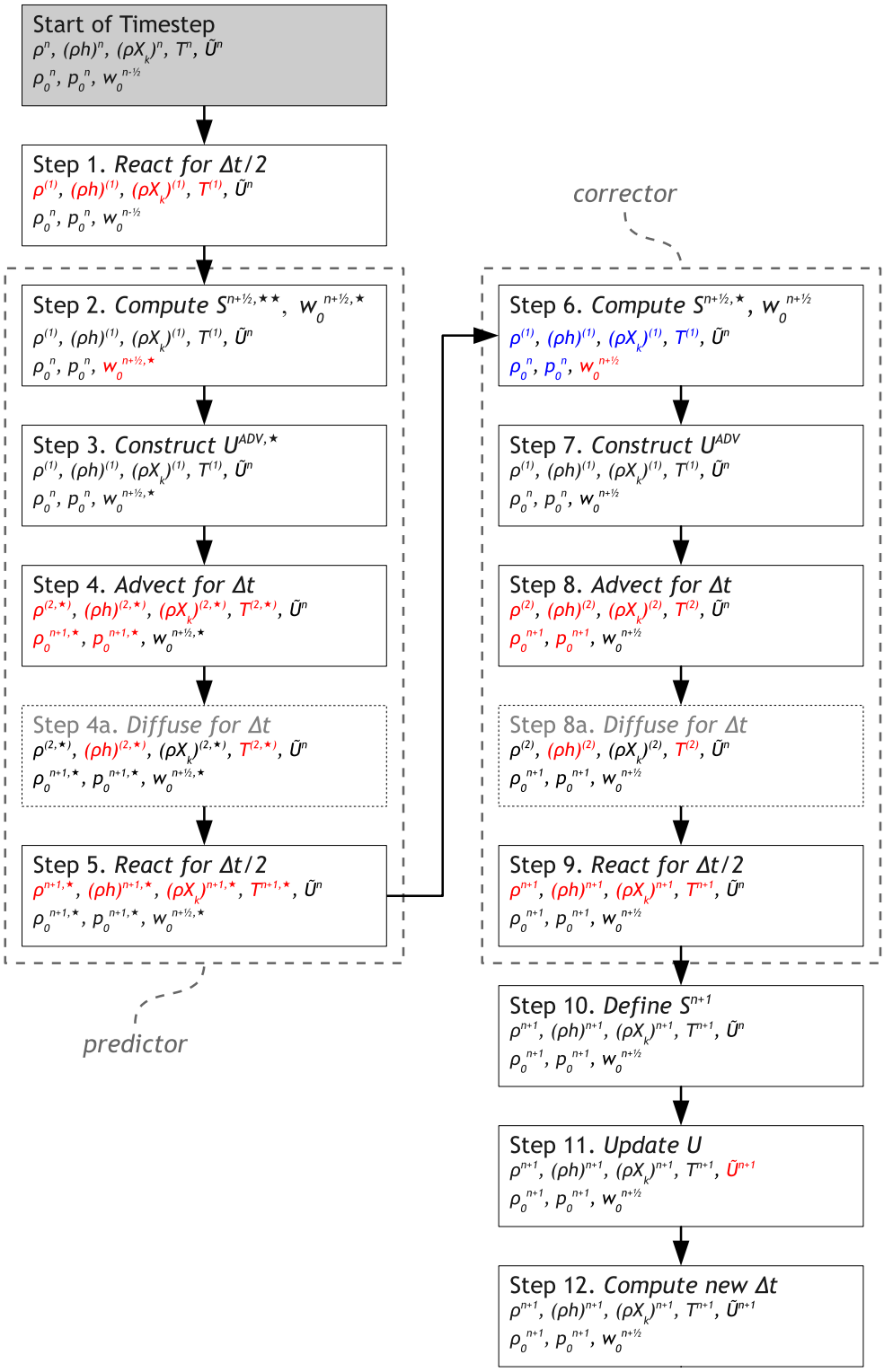


Figure 2.1: A flowchart of the algorithm. The thermodynamic state variables, base state variables, and local velocity are indicated in each step. Red text indicates that quantity was updated during that step. The predictor-corrector steps are outlined by the dotted box. The blue text indicates state variables that are the same in **Step 6** as they are in **Step 2**, i.e., they are unchanged by the predictor steps. The diffusion steps (4a and 8a) are optional, depending on `use_thermal_diffusion`.

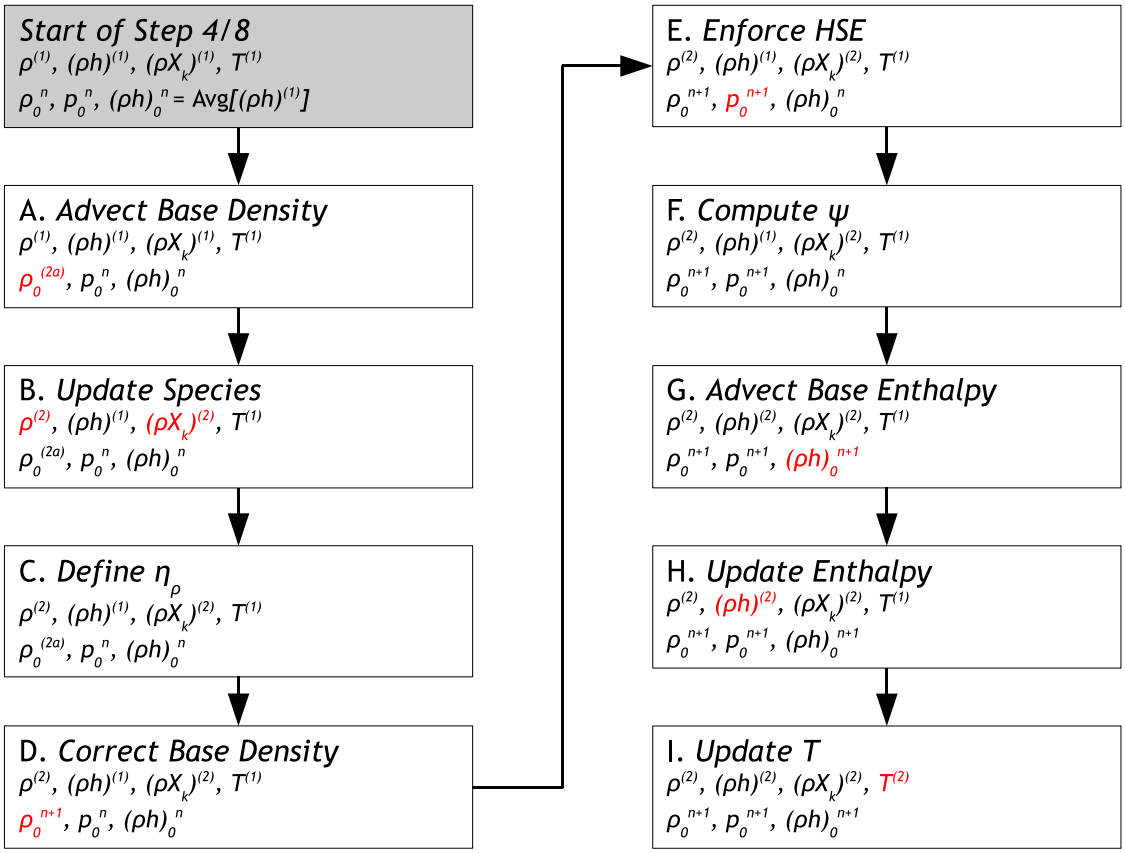


Figure 2.2: A flowchart for **Steps 4** and **8**. The thermodynamic state variables and base state variables are indicated in each step. Red text indicates that quantity was updated during that step. Note, for **Step 4**, the updated quantities should also have a $\star$ superscript, e.g., **Step 8I** defines $T^{(2)}$ while **Step 4I** defines $T^{(2),\star}$.

Part II

Using MAESTRO

CHAPTER 3

Getting Started

In this chapter we give an overview of MAESTRO, including some of the standard problems, how to run the code, some basic runtime parameters, and how to look at the output.

Quick Start

Here we will run the standard `reacting_bubble` problem (three reacting bubbles in a plane-parallel stratified atmosphere) on a single processor[1]. This test problem was shown in paper 3.

1. *Get a copy of* MAESTRO.

   If you don't already have a copy of MAESTRO, you can obtain one from the project's github page: `https://github.com/AMReX-Astro/MAESTRO` . There are several options: you can fork it directly on github (recommended if you intend to develop the code) or clone it using `git` from the project page.

   MAESTRO is under active development, so you will want to keep in sync with the changes by periodically pulling from the repository. Simply type

   ```
   git pull
   ```

   in the `MAESTRO/` directory.

2. *Get a copy of* Microphysics.

   MAESTRO and its compressible counterpart CASTRO share a common-set of microphysics solvers (nuclear reaction networks and equations of state). These are kept in a separate repo. Microphysics is also available on github and can be obtained via:

[1] In earlier versions of MAESTRO this problem was called `test2`

```
git clone https://github.com/starkiller-astro/Microphysics.git
```

You will periodic want to update Microphysics by doing

```
git pull
```

in the `Microphysics/` directory .

3. *Get a copy of* AMReX.

   MAESTRO requires the AMReX library to manage the grids and parallelization. We also rely on the build system in AMReX to build a MAESTRO executable. AMReX is also available on github and can be obtained via:

   ```
   git clone https://github.com/AMReX-Codes/amrex.git
   ```

   You will periodic want to update AMReX by doing

   ```
   git pull
   ```

   in the `amrex/` directory.

4. *Setup your shell environment*.

   MAESTRO needs to know where to find AMReX, by specifying the `AMREX_HOME` environment variable, and where to find Microphysics, bt specifying the `MICROPHYSICS_HOME` environment variable.

   If your shell is `Bash`, add

   ```
   export AMREX_HOME="/path/to/amrex/"
   export MICROPHYSICS_HOME="/path/to/Microphysics/"
   ```

   to your `.bashrc`.

   If your shell is `Csh/Tcsh`, add

   ```
   setenv AMREX_HOME /path/to/amrex/
   setenv MICROPHYSICS_HOME /path/to/Microphysics/
   ```

   to your `.cshrc`.

   Note: you must specify the full path to the `AMReX/` and `Microphysics/` directory. Do not use "$\sim$" to refer to your home directory—the scripts used by the build system will not be able to process this.

5. *Setup the problem's* `GNUmakefile`.

   In MAESTRO, each problem lives under one of three sub-directories of `MAESTRO/Exec`: `SCIENCE/`, `TEST_PROBLEMS/`, or `UNIT_TESTS/`. This problem sub-directory will contain any problem-specific files as well as the `GNUmakefile` that specifies how to build the executable. Note: we rely on features of GNU `make`. Full details of the `GNUmakefile` can be found in § 5.3. Here we will configure for a serial build.

Change directory to MAESTRO/Exec/TEST_PROBLEMS/reacting_bubble/. We only need to worry about the options at the very top of the GNUmakefile for now. These should be set as follows:

- NDEBUG := t

  This option determines whether we compile with support for debugging (usually also enabling some runtime checks). Setting NDEBUG := t turns off debugging.

- MPI :=

  This determines whether we are doing a parallel build, using the Message Passing Interface (MPI) library. For now, we leave this option empty, disabling MPI. This will build MAESTRO in serial mode, so no MPI library needs to be present on the host system.

- OMP :=

  This determines whether we are using OpenMP to do parallelism within a shared memory node. OpenMP is used together with MPI, with MPI distributing the grids across the processors and within a shared-memory node, OpenMP allows many cores to operate on the same grid. For now, we leave this option empty, disabling OpenMP.

- SDC :=

  This option determines whether we want to use the alternate SDC (Spectral Deferred Corrections) source for MAESTRO (see Chapter 20). This is experimental, and this option not present in all problems. We leave this blank so we compile the default MAESTRO source.

- COMP := gfortran

  This option specifies the Fortran compiler. We will use gfortran, which is the preferred compiler for MAESTRO. Specifying this compiler will automatically pull in the compiler settings as specified in AMREX_HOME/Tools/F_mk/. (Alternate compiler choices include Intel, PGI, PathScale, Cray.)

6. *Build the executable*.

   Type make. The build system will first find the dependencies amongst all the source files and then build the executable. When finished, the executable will have a name like main.Linux.gfortran.exe, where the specific parts of the name depend on the compilers and OS used.

   Note, at the end of the build process, a link will be made in the current directory to the data table needed for the equation of state (Microphysics/EOS/helmholtz/helm_table.dat).

7. *Run!*

   Each problem requires an input file. The inputs file (a Fortran namelist) consists of lines of the form *parameter* = *value*, where *parameter* is one of the many runtime parameters MAESTRO knows, and *value* overrides the default value for that parameter. For the reacting_bubble problem, we will use the inputs file inputs_2d. An overview of some of the more common runtime parameters is given in § 3.5, and a full list of all MAESTRO runtime parameters and their default values is given in Chapter 6.

   MAESTRO is run simply as:

```
./main.Linux.Intel.exe inputs_2d
```

We can also override the default value of any runtime parameter by specifying it on the commandline as

```
./main.Linux.Intel.exe inputs_2d --parameter value
```

As the code runs, a lot of information will pass through the screen. For each timestep, each of the steps 1 through 12 shown in the MAESTRO flowchart (Chapter 2) will be shown along with diagnostic information about the solution. Upon completion some memory usage information is printed.

8. *Examine the output*.

   As the code runs, it will output both plotfiles and checkpoints as well as one or more text diagnostic files (`maestro_diag.out` by default) with integral or extrema information (like maximum Mach number) from each timestep.

   By default, the plotfiles will be named `plt`*nnnnn*, where the number *nnnnn* is the timestep number when the file was outputted. Similarly, the checkpoints are named `chk`*nnnnn*. AMReX plotfiles and checkpoints are actually directories, with the data stored in sub-directories grouped by refinement level. Details of the simulation (build information, number of processors used, output date, output directory, runtime parameter values, ...) are stored in the plaintext `job_info` file in each plotfile directory.

   Note: unless otherwise specified all quantities in MAESTRO **are assumed to be in CGS units.**

   Visualization of results is described in the next section.

Working with the Output

Visualization and analysis are done on the plotfiles. A number of in-house and externally developed tools can work with AMReX-formatted plotfiles[2]. An example plot of the `reacting_bubble` problem run above is shown in Figure 3.1.

Amrvis

Amrvis is an easy-to-use visualization tool developed at LBL for 2- and 3D datasets which can plot slices through 3D datasets as well as volume-renderings. It can also very easily extract 1D lines through the dataset along any coordinate direction. It is distributed separately from the MAESTRO distribution.

Amrvis can be obtained via git as:

```
git clone https://ccse.lbl.gov/pub/Downloads/Amrvis.git
```

Amrvis is built in the C++ AMReX framework (instead of the Fortran AMReX framework that MAESTRO uses). The build systems are similar, but differ in a few ways.

Amrvis uses the `Motif` library for defining the GUI. On a Linux system, you may need to install the `lesstif` package and any related development packages (e.g. `lesstif-devel`). Depending on

[2]The plotfiles are in the same format as those made by the BoxLib library upon which MAESTRO was previously based.

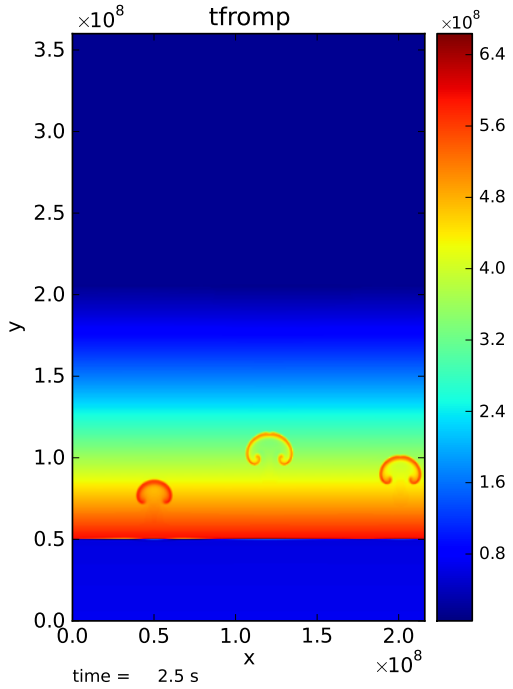


Figure 3.1: Visualization of the final output of the `reacting_bubble` problem showing the temperature field (as derived from the pressure). This plot was done with the `AmrPostprocessing` tools.

your Linux system, you may also need to install `libXpm` and related development packages (e.g. `libXpm-devel`).

Further details on the C++ AMReX build system used by Amrvis can be found in the AMReX documentation.

`AmrPostprocessing` scripts

Several useful analysis scripts (written in Fortran 90) can be found in `AmrPostprocessing/F_Src/` (distributed separately from MAESTRO). The `GNUmakefile` there needs to be edited to indicate which of the tools to build. For example, to extract the density along a line from the center of a plotfile, `plt00200`, in the y-direction:

```
fextract.Linux.gfortran.exe -d 2 -v "density" -p plt00200
```

These routines are described in § 9.1.

There is also a python visualization method in `AmrPostprocessing/python`. This is described in § 8.4.

VisIt

VisIt is a powerful, DOE-supported visualization tool for 2- and 3D datasets. It can do contouring, volume rendering, streamlines, ... , directly from AMReX plotfiles. Details on VisIt can be found at:

https://wci.llnl.gov/codes/visit/home.html.
The easiest way to get started with VisIt is to download a precompiled binary from the VisIt webpage.

Once VisIt is installed, you can open a AMReX plotfile by pointing VisIt to the Header file in the plotfile directory.

yt

yt (version 3.0 and later) can natively read the MAESTRO plotfiles. See the yt documentation or § 8.5.

Diagnostic Files

By default, MAESTRO outputs global diagnostics each timestep into a file called maestro_diag.out. This includes the maximum Mach number, peak temperature, and peak nuclear energy generation rate. Individual problems can provide their own diag.f90 file to produce custom diagnostic output. This information can be plotted directly with GNUplot, for example.

'Standard' Test Problems

Different problems in MAESTRO are contained in one of three sub-directories under MAESTRO/Exec: (SCIENCE/, TEST_PROBLEMS/, or UNIT_TESTS/). The GNUmakefile in each problem directory lists the components of MAESTRO that are used to build the executable. TEST_PROBLEMS/ contains simple problems that were used in the development of MAESTRO. Many of these were featured in the papers describing the MAESTRO algorithm.

Some of the test problems available are:

- double_bubble

  A rising bubble problem where the bubble(s) can have a different gamma than the surrounding atmosphere. This uses the multigamma EOS.

- incomp_shear_jet

  A simple pure-incompressible shear layer problem. This is the example problem used in [9]. This is useful to see how to use MAESTRO as an incompressible solver.

- reacting_bubble

  reacting_bubble places 3 hots spots in a plane-parallel atmosphere. Burning makes these bubbles buoyant, and then roll up. This problem was used in [4] to compare with compressible solvers.

  This problem can also be run adaptively. The tag_boxes.f90 file in the problem directory tags cells for refinement if the perturbational temperature, T', exceeds some threshold.

- rt

A Rayleigh-Taylor instability problem. There are two methods that the code is run here, in the standard (using `inputs_2d`), the base state has the stratified atmosphere and we introduce a velocity perturbation to start the instability. The alternate method, `inputs_2d_SNe`, uses the `do_smallscale` runtime parameter to eliminate the base state and instead use the incompressible constraint to evolve the system.

- `test_convect`

  `test_convect` drives convection through a plane-parallel atmosphere using an externally-specified heat source. This problem was used to compare with compressible solvers in [4] and to test the multilevel algorithm in [27].

- `test_spherical`

  This problem sets up an isentropically stratified star and stirs it up with a random velocity field. The low Mach number constraint is replaced with the anelastic constraint (through the `beta_type` runtime parameter). Analytically, under these conditions, the density of the star should not change. This test problem was discussed in Maestro paper IV [37].

Distributed Science Problems

The following problems were used for science studies. It is anticipated that more will be made available with time.

- `flame`

  A combustion-mode problem where we model a thermonuclear flame in a small domain. This enforces the low Mach combustion constraint divU = S. Hot ash and cool fuel are put into contact and a flame will ignite and propagate across the grid. Inflow boundary conditions are used to allow for an inflow velocity to be set to keep the laminar flame stationary.

  In this mode, MAESTRO behaves like the code described in [13], which was used for models of Rayleigh-Taylor unstable flames [11, 12, 39].

- `flame_1d`

  A 1-d version of the `flame` problem above. This uses a special elliptic solver in AMReX that only works for a single grid, so no parallel runs are allowed for this problem.

- `toy_convect`

  A nova-like problem for studying convection. This problem has seen extensive use in understanding which prediction types are the best when we have sharp species gradients. See Mike Z or Ryan for details.

- `wdconvect`

  Model convection leading up to ignition in the Chandraseskhar-mass SNe Ia progenitor model. This setup was the basis for the simulations presented in [37, 38, 28].

Common Runtime Parameters

Controlling Timestepping and Output

Parameters that set the maximum time for the simulation to run include:

- `stop_time` is the maximum simulation time, in seconds, to evolve the system for.
- `max_step` is the maximum number of steps to take.

Parameters affecting the size of the timestep include:

- `cflfac` is a multiplicative factor (≤ 1) applied to the advective CFL timestep
- `init_shrink` is the factor (≤ 1) by which to reduce the initial timestep from the estimated first timestep.

Parameters affecting output and restart include:

- `restart` tells MAESTRO to restart from a checkpoint. The value of this parameter should be the file number to restart from. For example, to restart from the checkpoint file `chk00010`, you would set `restart = 10`.
- `plot_int` is the number of steps to take between outputting a plotfile
- `plot_deltat` is the simulation time to evolve between outputting a plotfile. Note: to output only based on simulation time, set `plot_int = -1`.
- `check_int` is the number of steps to take between outputting a checkpoint.
- `plot_base_name` is the basename to use for the plotfile filename. The step number will be appended to this name.

Note that in addition to the normal plotfiles, there are *mini* plotfiles that store a small subset of the fields, intended to be output more frequently. These are described in § 8.1.1.

Defining the Grid and Boundary Conditions

Parameters that determine the spatial extent of the grid, the types of boundaries, and the number of computational cells include:

- `max_levs` is the maximum number of grid levels in the AMR hierarchy to use. `max_levs = 1` indicates running with only a single level spanning the whole domain.
- `n_cellx`, `n_celly`, `n_cellz` the size of base level in terms of number of cells, in the x, y, and z coordinate directions.
- `max_grid_size` the maximum extend of a grid, in any coordinate direction, as measured in terms of number of cells.

  For multilevel problems, the parameter `max_grid_size_1` controls the maximum extent on level 1 (the base grid), `max_grid_size_2` controls the maximum extent on level 2, and `max_grid_size_3` controls the maximum extent on levels 3 and higher.

- `prob_lo_x`, `prob_lo_y`, `prob_lo_z` is the physical coordinate of the lower extent of the domain boundary in the x, y, and z coordinate directions.

- `prob_hi_x`, `prob_hi_y`, `prob_hi_z` is the physical coordinate of the upper extent of the domain boundary in the x, y, and z coordinate directions.

- There are two ways to specify boundary conditions—via integer flags or descriptive string names. If the string names are present, then they will override the integer quantities in determining the boundary conditions.

  - `bcx_lo`, `bcy_lo`, `bcz_lo` , `bcx_hi`, `bcy_hi`, `bcz_hi` are the boundary condition types at the lower (‘`lo`’) and upper (‘`hi`’) domain boundaries in the x, y, and z coordinate directions. The different types are set via integer flags listed in table 3.1.

Table 3.1: Boundary condition flags

BC type	integer flag
periodic	-1
inlet (user-defined)	11
outlet	12
symmetry	13
slip wall	14
no-slip wall	15

  - `xlo_boundary_type`, `ylo_boundary_type`, `zlo_boundary_type`, `xhi_boundary_type`, `yhi_boundary_type`, `zhi_boundary_type` are the boundary condition types at the lower and upper domain boundaries in the x, y, and z coordinate directions. The boundary type is set by providing a string name—valid values are listed in table 3.2

Table 3.2: Boundary condition string names

BC type	integer flag
periodic	“periodic”
inlet (user-defined)	“inlet”
outlet	“outlet”
symmetry	“symmetry”
slip wall	“slip wall”
no-slip wall	“no slip wall”

  The string-based parameters are a newer option for specifying boundary conditions, and are preferred due to clarity. The conversion between the string names and integer flags is done by AMReX in the `bc_module` at the time of initializing the runtime parameters.

Note that grid cells must be square, i.e. $\Delta x = \Delta y = \Delta z$ where Δx on the base grid is computed as $(\texttt{prob\_hi\_x} - \texttt{prob\_lo\_x})/\texttt{n\_cellx}$. For multilevel problems, the effective number of zones on the finest grid in the x direction will be $\texttt{n\_cellx} \cdot 2^{(\texttt{max\_levels}-1)}$.

Development Model

When you clone MAESTRO from github, you will be on the `master` branch of the repo. New changes to MAESTRO are first introduced into the `development` branch in the MAESTRO git repository. Nightly regression tests are run on `development` to ensure that our answers don't change. Around the first work day of each month, we merge from `development` $\rightarrow$ `master` (assuming tests pass) and tag the state of the code with a date-based tag `YY-MM`. We do this on all the other repos in the AMReX-ecosystem, including `amrex/`, `Microphysics/`, and `Castro/`.

If you want to contribute to MAESTRO's development, issue a pull-request through github onto the `development` branch.

Parallel Jobs

To run in parallel with MPI, you would set `MPI := t` in your `GNUmakefile`. For a machine with working MPI compiler wrappers (`mpif90` and `mpicc`), the build system should find these and compile with MPI support automatically. This is the easiest way to do a parallel build, and should work on most Linux systems.

More generally, the build system needs to know about your MPI installation. For popular national computing facilities, this is already setup, and the build system looks at the machine hostname to set the proper libraries. For other machines, you may need to edit the `GMake.MPI` file in the AMReX build files. See § 10 for more details.

OpenMP can be used to parallelize on shared-memory machines (i.e. within a node). OpenMP support is accomplished through the compiler. Setting `OMP := t` in the `GNUmakefile` will enable the proper compiler flags to build with OpenMP. Note: not all MAESTRO modules have OpenMP support. Microphysics routines need to be written in a threadsafe manner. This can be tested via the `test_react` unit test (see § 4).

CHAPTER 4

Unit Tests

In addition to the MAESTRO science problems, which use the full capabilities of MAESTRO, there are a number of unit tests that exercise only specific components of the MAESTRO solvers. These tests have their own drivers (a custom `varden.f90`) that initialize only the data needed for the specific test and call specific MAESTRO routines directly.

`test_advect`

This test initializes a Gaussian density field (no other scalar quantities are used) and a uniform velocity field in any one of the coordinate directions. The Gaussian profile is advected through the period domain exactly once and the error in the density profile (L2 norm) is computed. The driver for this problem does this for every dimension twice (once with a positive velocity and once with a negative velocity), and loops over all advection methods (`ppm_type = 0,1,2` and `bds_type = 1`). After all coordinate directions are tested, the norms are compared to ensure that the error does not show any directional bias.

Note: the BDS advection method does not guarantee that the error be independent of the advection direction—small differences can arise. What's happening here is that within each cell BDS is trying to define a tri-linear profile for rho subject to the constrants in the BDS paper [30] (constraints 1 and 2 on p. 2044 after eq. 3.4). We do not solve the L2 minimization problem exactly so we iterate up to 3 times using a simple heuristic algorithm designed to work toward the constraint. The iteration will give different results depending on orientation since we work through the corners in arbitrary order.

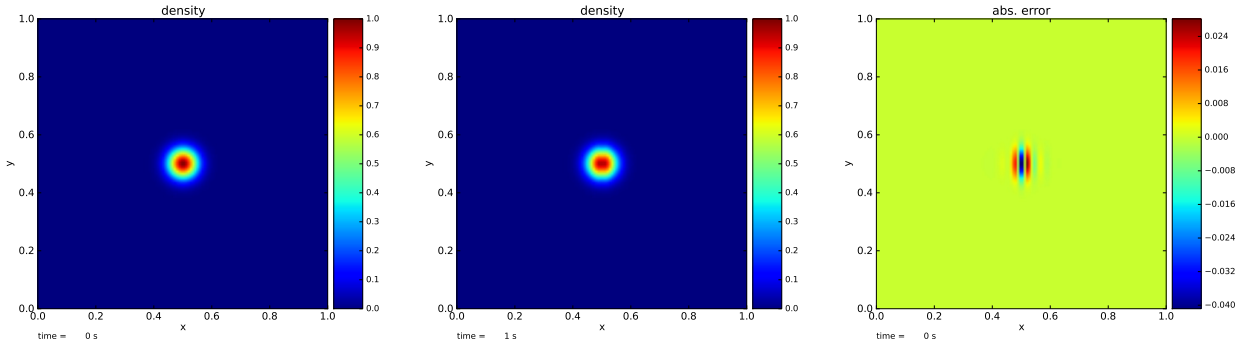


Figure 4.1: Initial Gaussian density profile (left); density profile after advecting to the right for one period (center); absolute error between the final and initial density fields, showing the error in the advection scheme (right).

`test_average`

This test initializes a 1D radial base state quantity with a Gaussian distribution, maps it into the 3D domain (assuming a spherical geometry) using the routines provided by the `fill_3d_module` module, and then calls `average` to put it back onto a 1D radial array. This way we test the accuracy of our procedure to map between the 1D radial and 3D Cartesian states. The output from this test was described in detail in [27].

`test_basestate`

This test initializes the base state to contain a hydrostatic model and then evolves the state with heating to watch the hydrostatic adjustment of the atmosphere. In particular, the base state velocity, w_0, is computed in response to the heating and this is used to advect the base state density and compute the new pressure, p_0. An early version of this routine was used for the plane-parallel expansion test in [6]. This version of the test was also shown for a spherical, self-gravitating star in [27].

`test_diffusion`

This test initializes a Gaussian temperature profile and calls the thermal diffusion routines in MAESTRO to evolve the state considering only diffusion. The driver estimates a timestep based on the explicit thermal diffusion timescale and loops over calls to the thermal diffusion solver. A Gaussian remains Gaussian when diffusing, so an explicit error can be computed by comparing to the analytic solution. This test is described in [25].

`test_eos`

This test sets up a 3-d cube with ρ varying on one axis, T on another, and the composition on the third. The EOS is then called in every zone, doing $(\rho, T) \rightarrow p, h, s, e$ and stores those quantities. Then it does

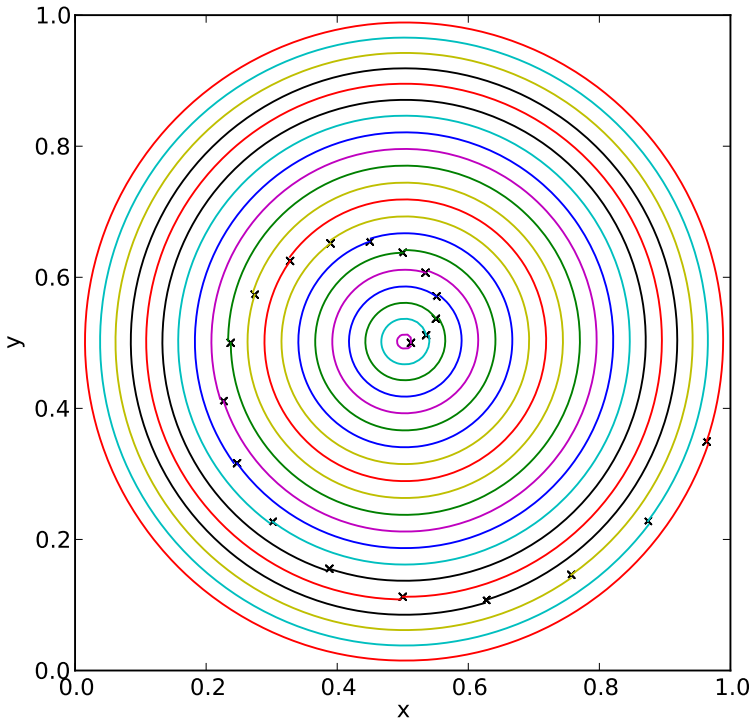


Figure 4.2: Particle paths for the test_particles problem. The initial position of the particles is marked with an $\times$.

each of the different EOS types to recover either T or ρ (depending on the type), and stores the new T (or ρ) and the relative error with the original value. A plotfile is stored holding the results and errors. This allows us to determine whether the EOS inversion routines are working right.

test_particles

This test exercises the particle advection routine. A simple circular velocity field, with the magnitude increasing with radius from the center is initialized. A number of particles are then initialized at various radii from the center and they are advected for one period. The particle paths should be perfect circles, and the final particle position should overlap with the initial position.

Particle data is stored separately from the fluid data. Instead of being part of the plotfiles, the particle data is outputted each timestep into files named timestamp_*, where the number indicates which processor did the writing. These particle files can be processed and the particle data plotted using the python routines in data_processing/python/.

The output from this test can be visualized with the script plot.py in the test directory. The output shows the particle paths (see figure 4.2).

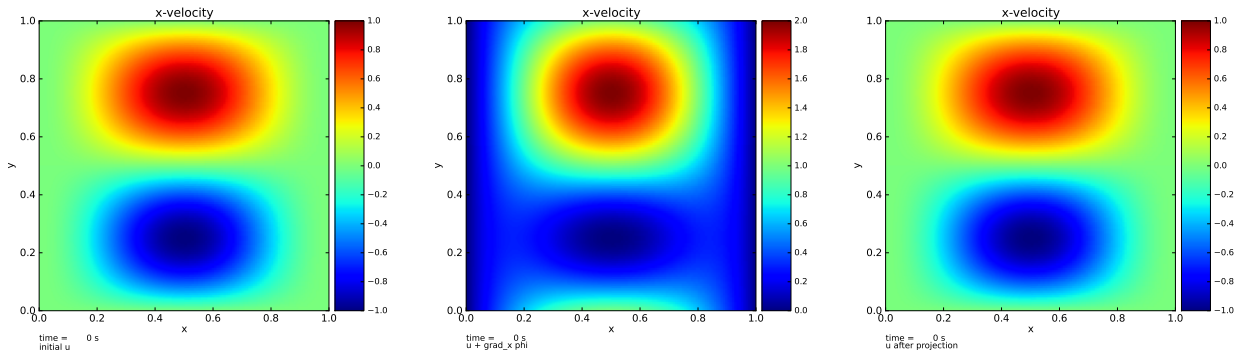


Figure 4.3: Initial divergence free velocity field (x-component; left); Velocity field plus gradient of a scalar (x-component; center); and resulting velocity after projecting out the non-divergence free portion (x-component; right). This is with slipwall boundary conditions on all sides, a 2-level grid with the left half refined and right half coarse, and the hgprojection tested.

`test_projection`

This tests the projection routines in 2- and 3-d—either the hgprojection (`project_type = 1`) or the MAC projection (`project_type = 2`). A divergence-free velocity field is initialized and then "polluted" by adding the gradient of a scalar. The form of the scalar differs depending on the boundary conditions (wall and periodic are supported currently). Finally, the hgproject routine is called to recover the initial divergence-free field. Figure 4.3 shows the initial field, polluted field, and result of the projection for the hgproject case.

`test_react`

This simply tests the reaction network by calling the MAESTRO `react_state` routine directly. The network is selected in the `GNUmakefile` by setting the `NETWORK_DIR` variable. A 3d cube is setup with density varying on one axis, temperature varying on another, and the composition varying on the third. The density and temperature ranges are set in the inputs file. The composition is read in via an input file.

A good use of this test is to test whether a burner is threadsafe. This is accomplished by compiling with OpenMP (setting `OMP=t`) and the running with 1 thread and multiple threads (this can be done by setting the environment variable `OMP_NUM_THREADS` to the desired number of threads). Since each zone is independent of the others, the results should be identical regardless of the number of threads. This can be confirmed using the `fcompare` tool in `BoxLib/Tools/Postprocessing/F_Src/`.

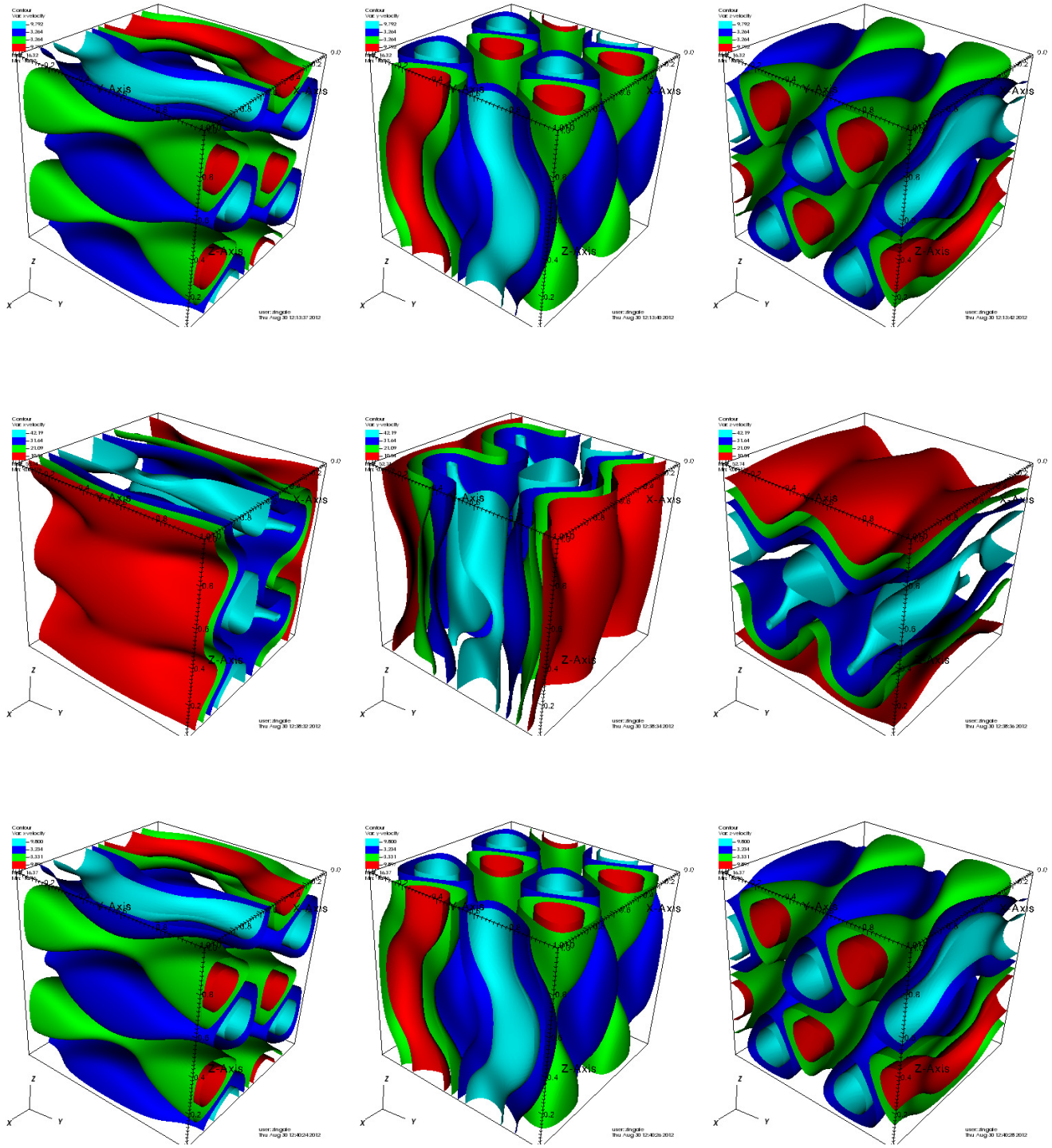


Figure 4.4: Projection test in 3-d showing the x-velocity (left), y-velocity (middle), and z-velocity (right) initially (top row), after the gradient of a scalar is added (center row), and the resulting velocity after the projection. This is with slipwall boundary conditions on all sides, a 2-level grid with an octant refined, and the hgprojection.

CHAPTER 5

Architecture

MAESTRO **Basics**

MAESTRO models problems that are in tight hydrostatic equilibrium. The fluid state is decomposed into a 1D radial base state that describes the hydrostatic structure of the star or atmosphere, and a 2- or 3D Cartesian full state, that captures the departures from hydrostatic equilibrium. Two basic geometries are allowed. A *plane-parallel* geometry assumes that the domain is thin compared to the radius of curvature of the star, and therefore the 1D base state is perfectly aligned with the Cartesian state. A *spherical* geometry is for modeling an entire star[1]. Here, the 1D base state is not aligned with the Cartesian state. Figure 5.1 shows these geometries.

MAESTRO can use adaptive mesh refinement to focus resolution on complex regions of flow. For Cartesian/plane-parallel geometries, all cells at the same height must be at the same level of refinement. This restriction is to allow for the base state to directly align with the Cartesian state everywhere. For spherical geometries, there is no such restriction (again, see Figure 5.1). The MAESTRO grids are managed by the AMReX library, which is distributed separately.

The MAESTRO **'Ecosystem'**

Building MAESTRO requires both the MAESTRO-specific source files (distributed in the `MAESTRO/` directory), and the AMReX library (distributed separately, consisting of the `amrex/` directory). AMReX provides both a C++ and a Fortran framework. MAESTRO only uses the Fortran portions of AMReX. Figure 5.2 shows the relationship between the different packages, highlighting what goes into defining a specific MAESTRO problem.

[1]Spherical geometry only exists for 3-d. This is a design decision—convection is 3-d. You can however run as an octant

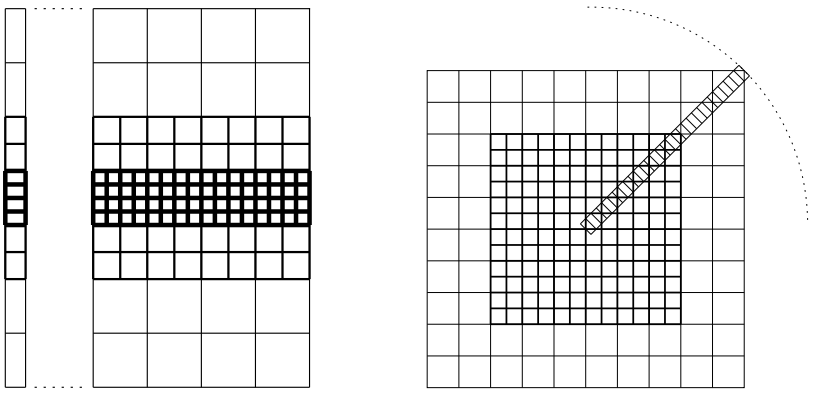


Figure 5.1: MAESTRO geometries, showing both the 1D base state and the full Cartesian state. (Left) For multi-level problems in planar geometry, we force a direct alignment between the radial array cell centers and the Cartesian grid cell centers by allowing the radial base state spacing to change with space and time. (Right) For multi-level problems in spherical geometry, since there is no direct alignment between the radial array cell centers and the Cartesian grid cell centers, we choose to fix the radial base state spacing across levels. Figure taken from [27].

Problems piece together various MAESTRO directories, choosing a reaction network, equation of state, and conductivity routine to build an executable. Briefly, the MAESTRO sub-directories are:

- `MAESTRO/`

  The main MAESTRO algorithm directory.

  Important directories under `MAESTRO/` include:

  - `Docs/`

    Documentation describing the basic algorithm (including this document).

  - `Exec/`

    The various problem-setups. Each problem in MAESTRO gets it own sub-directory under `SCIENCE/`, `TEST_PROBLEMS/`, or `UNIT_TESTS/`. The `GNUmakefile` in the problem directory includes the instructions on how to build the executable, including what modules in `Microphysics/` are used. Any file that you place in a sub-directory here takes precedence over a file of the same name in `MAESTRO/`. This allows problems to have custom versions of the main MAESTRO routines (e.g. initial conditions via `initdata.f90`. See § 5.3.1 and Chapter 10 for details on the build system).

    * `SCIENCE/`

      MAESTRO problem directories for science studies. These are the setups that have been used for science papers in the past, or are the basis for current science studies.

    * `TEST_PROBLEMS/`

      MAESTRO problem directories for simple test problems that have been used while developing MAESTRO. Many of these problems have appeared in the various papers describing the MAESTRO algorithm.

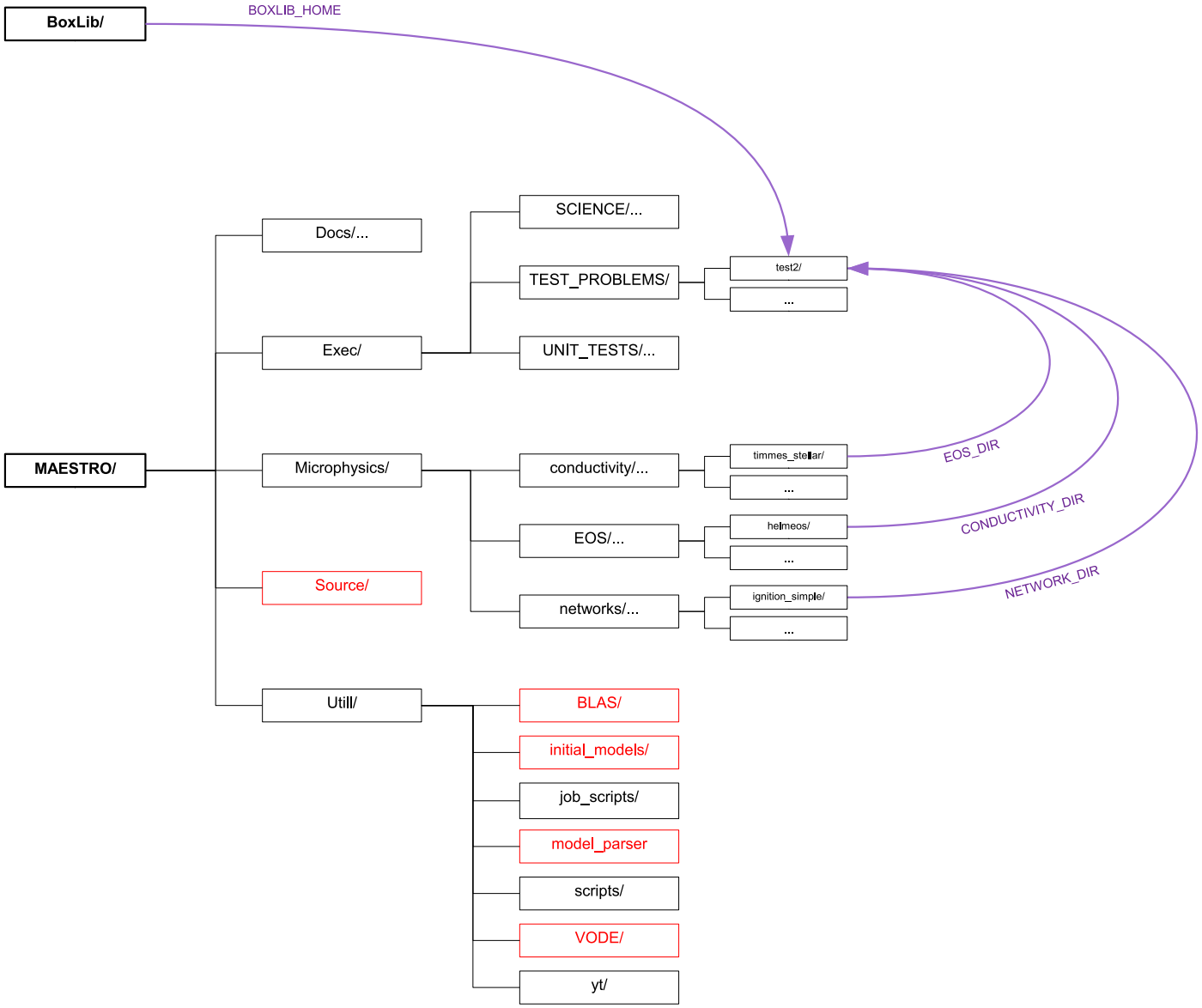


Figure 5.2: The basic MAESTRO 'ecosystem'. Here we see the different packages that contribute to building the `reacting_bubble` problem in MAESTRO. The red directories are part of most standard MAESTRO build. The purple lines show the directories that are pulled in through various Makefile variables (`AMREX_HOME`, `NETWORK_DIR`, `EOS_DIR`, and `CONDUCTIVITY_DIR`).

    * UNIT_TESTS/

      Special MAESTRO problem directories that test only a single component of the MAESTRO algorithm. These often have their own main drivers (varden.f90) that setup and initialize some data structures and then call only a few of the MAESTRO routines. See Chapter 4 for details.

- Microphysics/[2]

  The basic microphysics routines used by MAESTRO. These are organized into the following sub-directories.

    * conductivity/

      Various routines for computing the thermal conductivity used in the thermal diffusion part of the algorithm.

    * EOS/

      The gamma_law_general/.

    * networks/

      The basic general_null network that defines arbitrary non-reacting species.

- Source/

  The main MAESTRO source. Here you will find the driver routine, the advection routines, etc. All MAESTRO problems will compile this source.

- Util/

  Various helper routines exist in this directory. Some of these are externally developed.

    * BLAS/

      Linear algebra routines.

    * initial_models/

      Simple routines for generating toy initial models in hydrostatic equilibrium.

    * model_parser/

      A simple Fortran module for reading in 1D initial model files. This is used by the initialization routines to get the initial model data.

    * random/

      A random number generator.

    * VODE/

      The VODE [15] package for integrating ODEs. At the moment, this is used for integrating various reaction networks.

The AMReX directory structure is shown in Figure 5.3. The subset of the directories that are used by MAESTRO are:

[2]Note: many more compatible routines are available in the separate Microphysics git repo

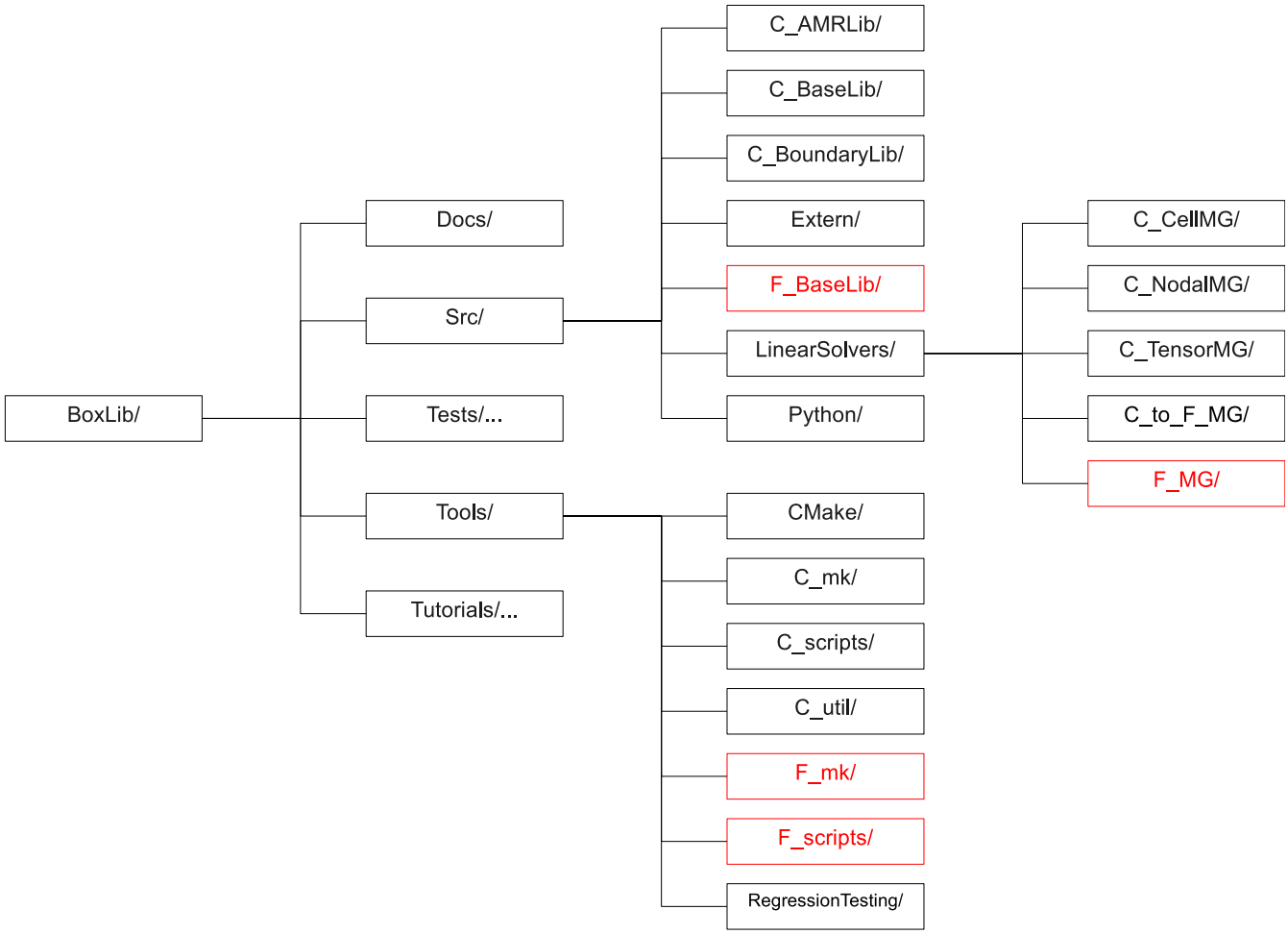


Figure 5.3: The basic AMReX directory structure. The directories used by MAESTRO are indicated in red. (this figure is likely out of date)

- Src/

  The main AMReX source directory. In MAESTRO, we only use the Fortran source files. The core directories are:

  - `F_BaseLib/`

    The Fortran AMReX files. This is a library for describing meshes consisting of a union of boxes. The AMReX modules define the basic datatypes used in MAESTRO. AMReX also provides the routines that handle the parallelization and I/O.

  - `LinearSolvers/`

    The AMReX linear solvers—these are used to solve elliptic problems in the MAESTRO algorithm.

    * `F_MG`

      The Fortran multigrid solver, with support for both cell-centered and node-centered data.

- Tools/

  Various tools used for building AMReX applications. Here we use:

  - F_mk/

    The generic Makefiles that store the compilation flags for various platforms. Platform/compiler-specific options are stored in the comps/ sub-directory.

  - F_scripts/

    Some simple scripts that are useful for building, running, maintaining MAESTRO.

Finally the amrex/Tools/Postprocessing/F_Src package provides simple Fortran-based analysis routines (e.g. extract a line from a multidimensional dataset) that operate on AMReX datasets. These are described in § 9.1. Several sub-directories with python-based routines are also here. These are described both in § 9.1 and § 8.4.

Adding A New Problem

Different MAESTRO problems are defined in sub-directories under Exec/ in SCIENCE, TEST_PROBLEMS, or UNIT_TESTS. To add a problem, start by creating a new sub-directory—this is where you will compile your problem and store all the problem-specific files.

The minimum requirement to define a new problem would be a GNUmakefile which describes how to build the application and an input file which lists the runtime parameters. The problem-specific executable is built in the problem directory by typing make. Source files are found automatically by searching the directories listed in the GNUmakefile. Customized versions of any source files placed in the problem-directory override those with the same name found elsewhere. Any unique source files (and not simply a custom version of a file found elsewhere) needs to be listed in a file called GPackage.mak in the problem-directory (and this needs to be told to the build system—see below).

The GNUmakefile

A basic GNUmakefile begins with:

```
NDEBUG := t
MPI    :=
OMP    :=
```

Here, NDEBUG is true if we are building an optimized executable. Otherwise, the debug version is built—this typically uses less optimization and adds various runtime checks through compiler flags. MPI and OMP are set to true if we want to use either MPI or OpenMP for parallelization. If MPI := t, you will need to have the MPI libraries installed, and their location may need to be specified in MAESTRO/mk/GMakeMPI.mak.

The next line sets the compiler to be used for compilation:

```
COMP := gfortran
```

The MAESTRO build system knows what options to use for various compiler families. The COMP flag specifies which compiler to use. Allowed values include Intel, gfortran, PGI, PathScale, and Cray. The specific details of these choices are defined in the MAESTRO/mk/comps/ directory.

MKVERBOSE set to true will echo the build commands to the terminal as the are executed.

```
MKVERBOSE := t
```

The next line defines where the top of the MAESTRO source tree is located.

```
MAESTRO_TOP_DIR := ../../..
```

A MAESTRO application is built from several packages (the multigrid solver, an EOS, a reaction network, etc.). The core MAESTRO packages are always included, so a problem only needs to define the EOS, reaction network, and conductivities to use, as well as any extra, problem-specific files.

```
EOS_DIR := helmholtz
CONDUCTIVITY_DIR := constant
NETWORK_DIR := ignition_simple

EXTRA_DIR := Util/random
```

Note that the microphysics packages are listed simply by the name of the directory containing the specific implementation (e.g. helmholtz). By default, the build system will look in Microphysics/EOS/ for the EOS, Microphysics/conductivity/ for the conductivity routine, and Microphysics/networks/ for the reaction network. To override this default search path, you can set EOS_TOP_DIR, CONDUCTIVITY_TOP_DIR, and NETWORK_TOP_DIR respectively.

Generally, one does not need to include the problem directory itself in EXTRA_DIR, unless there are unique source files found there, described in a GPackage.mak file. These variables are interpreted by the GMaestro.mak file and used to build a master list of packages called Fmdirs. The build system will attempt to build all of the files listed in the various GPackage.mak files found in the Fmdirs directories. Furthermore, Fmdirs will be will be added to the make VPATH, which is the list of directories to search for source files. The problem directory will always be put first in the VPATH, so any source files placed there override those with the same name found elsewhere in the source tree.

Some packages (for instance, the helmholtz EOS) require Fortran include files. The Fmincludes variable lists all those directories that contain include files that are inserted into the Fortran source at compile time via the include statement. Presently, the only instance of this is with the Helmholtz general equation of state found in Microphysics/EOS/helmholtz/. This is automatically handled by the GMaestro.mak instructions.

Runtime parameters listed in the MAESTRO/_parameters file are parsed at compile time and the file probin.f90 is written and compiled. This is a Fortran module that holds the values of the runtime parameters and makes them available to any routine. By default, the build system looks for a file called _parameters in the problem directory and adds those parameters along with the master list of MAESTRO parameters (MAESTRO/_parameters) to the probin_module.

The final line in the GNUmakefile includes the rules to actually build the executable.

```
include $(MAESTRO_TOP_DIR)/GMaestro.mak
```

Handling Problem-Specific Source Files

As mentioned above, any source files placed in the problem directory override a files with the same name found elsewhere in the source tree. This allows you to create a problem-specific version of any routine. Source files that are unique to this problem (i.e. there is no file with the same name elsewhere in the source tree) need to be listed in a file `GPackage.mak` in the problem directory, and the problem-directory needs to be explicitly listed in the `EXTRA_DIR` list in the `GNUmakefile`.

Defining Runtime Parameters

The runtime parameters for the core MAESTRO algorithm are listed in `MAESTRO/_parameters`. That file is parsed at compile-time by the `MAESTRO/write_probin.py` script (along with any problem-specific parameters). The script outputs the `probin.f90` source file. Each line in the `_parameters` file has the form:

parameter *data-type* *value*

where *parameter* is the name of the runtime parameter, *data-type* is one of {`character`, `real`, `integer`, `logical`}, and the *value* specifies the default value for the runtime parameter. Comments are indicated by a '`#`' character and are used to produce documentation about the available runtime parameters. For the documentation, runtime parameters are grouped together in the `_parameters` file into categories. The category headings are defined by comments in the `_parameters` file and any comments following that heading are placed into that category. The documentation (Chapter 6.1) is produced by the script `MAESTRO/docs/runtime_parameters/rp.py`.

At runtime, the default values for the parameters can be overridden either through the inputs file (by adding a line of the form: `parameter = value`) or through a command-line argument (taking the form: `--parameter value`). The `probin_module` makes the values of the runtime parameters available to the various functions in the code (see § 5.6.7).

Problem-specific runtime parameters should be defined in the problem-directory in a file called `_parameters`. This file will be automatically found at compile time.

Preparing the Initial Model

MAESTRO models subsonic flows that are in hydrostatic equilibrium. The solution in MAESTRO is broken up into a 1D base state and the 2- or 3D full state. The job of the 1D base state in the algorithm is to represent the hydrostatic structure. The full, Cartesian state carries the departures from hydrostatic equilibrium. The underlying formulation of the low Mach number equations assumes that the base state is in hydrostatic equilibrium. At the start of a simulation, the initial model is read in and taken as the base state. Therefore, any initial model needs to already be in hydrostatic equilibrium.

The routines in `Util/initial_models/` prepare an initial model for MAESTRO. In general, there are two different proceduces that are needed. The first type modify an existing 1D initial model produced somewhere else (e.g. a 1D stellar evolution code), and map it onto a uniform grid, at the desired resolution, using the equation of state in MAESTRO, and using MAESTRO's discretization of hydrostatic equilibrium. The second type generate the initial model internally, by integrating the condition

of hydrostatic equilibrium together with a simplifying assumption on the energy (e.g. isothermal or isentropic). In both cases hydrostatic equilibrium is enforced as:

$$\frac{p_{i+1} - p_i}{\Delta r} = \frac{1}{2}(\rho_i + \rho_{i+1}) g_{i+1/2} \tag{5.1}$$

Here, $g_{i+1/2}$ is the edge-centered gravitational acceleration.

The `toy_atm` example provides a simple approximation for a thin (plane-parallel) convectively-unstable accreted layer on the surface of a star. This can be used as the starting point for a more complex model.

MAESTRO initial models are read in by the `Util/model_parser` routines. This expects the initial model to contain a header giving the number of variables and their names, followed by rows of data giving the coordinate and data values at that coordinate. The initial model should contain the same species data (in the form of mass fractions) as defined in the `network` module used by the MAESTRO problem.

Full details on which initial model routine matches each problem and how the initial models are used to initialize the full state data can be found in § 7.1.

Customizing the Initialization

The best way to customize the initialization (e.g. add perturbations) is to copy from one of the existing problems. The file `initveldata.f90` controls the velocity field initialization and `initscaldata.f90` controls the initialization of the scalars (ρ, ρX_k, ρh). The `reacting_bubble` problem is a good starting point for plane-parallel and `wdconvect` is a good starting point for full stars.

AMReX **Data Structures**

MAESTRO's gridding is handled by the AMReX library, which contains the most fundamental objects used to construct parallel block-structured AMR applications—different regions of the domain can have different spatial resolutions. At each level of refinement, the region covered by that level is divided into grids, or boxes. The entire computational domain is covered by the coarsest (base) level of refinement, often called level $\ell = 0$, either by one grid or divided into many grids. Higher levels of refinement have cells that are finer by a "refinement ratio" (typically 2). The grids are properly nested in the sense that the union of grids at level $\ell + 1$ is contained in the union of grids at level ℓ. Furthermore, the containment is strict in the sense that, except at physical boundaries, the level ℓ grids are large enough to guarantee that there is a border at least n_{buffer} level ℓ cells wide surrounding each level $\ell + 1$ grid (grids at all levels are allowed to extend to the physical boundaries so the proper nesting is not strict there). For parallel computations, the boxes are spread across processors, in a fashion designed to put roughly equal amounts of work on each processor (load balancing).

ghost cells?

On a grid, the data can be stored at cell-centers, on a face/edge, or on the corners. In AMReX, data that is on an edge is termed 'nodal' in that direction (see Figure 5.4). Data that is on the corners is nodal in all spatial directions. In MAESTRO, the state data (density, enthalpy, velocity, ...) is generally cell-centered. Fluxes are nodal in the direction they represent. A few quantities are nodal in all directions (e.g. ϕ used in the final velocity projection).

To simplify the description of the underlying AMR grid, AMReX provides a number of Fortran types. We briefly summarize the major data types below. A more extensive introduction to AMReX is provided by the AMReX User's Guide, distributed with the library.

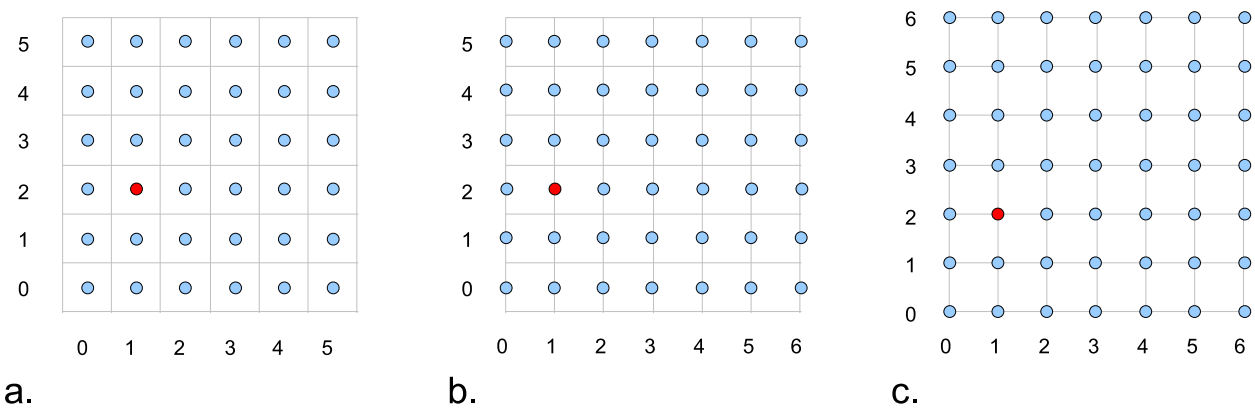


Figure 5.4: Some of the different data-centerings: (a) cell-centered, (b) nodal in the x-direction, and (c) nodal in both the x- and y-directions. Note that for nodal data, the integer index corresponds to the lower boundary in that direction. In each of these centerings, the red point has the same indices: (1,2). Not shown is the case where data is nodal in the y-direction only.

`box`

A `box` is simply a rectangular domain in space. Note that boxes do not hold the state data themselves. A `box` has a `lo` and `hi` index in each coordinate direction that gives the location of the lower-left and upper-right corner with respect to a global index space.

The computational domain is divided into boxes. The collection of boxes with the same resolution comprise a level. Figure 5.5 shows three boxes in the same level of refinement. The position of the boxes is with respect to the global index space at that level. For example, box 1 in the figure has `lo` = (3,7) and `hi` = (9,12). Note that the global indexing is 0-based.

The global index space covers the entire domain at a given resolution. For a simulation setup with `n_cellx = 32` and `n_celly = 32`, the coarsest level (level 1) has 32×32 zones, and the global index space will run from $0, \ldots, 31$ in each coordinate direction. Level 2 will have a global index space running from $0, \ldots, 63$ in each coordinate direction (corresponding to 64×64 zones if fully refined), and level 3 will have a global index space running from $0, \ldots, 127$ in each coordinate direction (corresponding to 128×128 zones if fully refined).

Common Operations on a `box`

A `box` declared as:

```
type(box) :: mybox
```

The upper and lower bounds of the box (in terms of the global index space) are found via:

- `lo = lwb(mybox)` returns an array, `lo(dm)`, with the box lower bounds
- `hi = upb(mybox)` returns an array, `hi(dm)`, with the box upper bounds

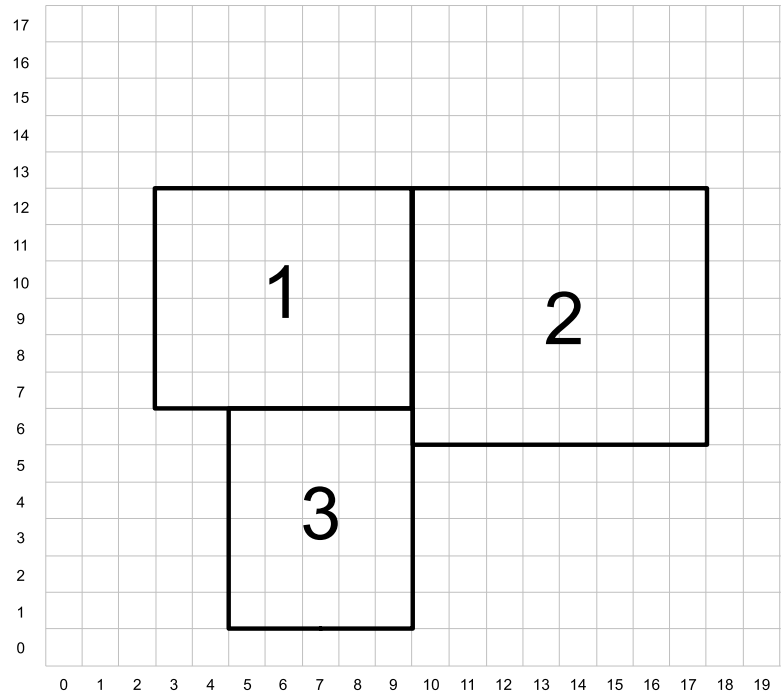


Figure 5.5: Three boxes that comprise a single level. At this resolution, the domain is 20×18 zones. Note that the indexing in AMReX starts with 0.

`boxarray` and `ml_boxarray`

A `boxarray` is an array of boxes. A `ml_boxarray` is a collection of `boxarrays` at different levels of refinement.

`layout` and `ml_layout`

A `layout` is basically a `boxarray` that knows information about other boxes, or box "connectivity." It contains additional information that is used in filling ghost cells from other fine grids or from coarser grids. This information is stored as long as the layout exists so that we don't have to recompute intersections every time we do some operation with two `multifabs` that have that layout, for example.

By separating the layout from the actual data, we can allocate and destroy data that lives on the grid as needed.

`fab`

A `fab` is a "Fortran Array Box". It contains the state data in a multidimensional array and several `box`-types to describe where in the global index-space it lives:

```
type fab
   ...
   type(box) :: bx
   type(box) :: pbx
```

```
    type(box) :: ibx
  end type fab
```

bx represents the box in the global index-space over which the fab is defined, pbx represents the "physical" box in the sense that it includes bx plus ghost cells, and ibx is the same as bx unless the fab is nodal. As can be seen in Figure 5.4, for the same grid nodal data requires one more array element than cell-centered data. To address this ibx is made by growing bx by one element along all nodal dimensions.

It's important to note that all state data is stored in a four-dimensional array *regardless of the problem's dimensionality*. The array is (nx,ny,nz,nc) in size, where nc is the number of components, for instance representing different fluid variables, and (nx,ny,nz) are the number of cells in each respective spatial dimension. For 2D problems, nz=1.

A fab would represent the data for a single box in the domain. In MAESTRO, we don't usually deal with fabs alone, but rather we deal with multifabs, described next.

multifab

A multifab is a collection of fabs at the same level of refinement. This is the primary data structure that MAESTRO routine operate on. A multilevel simulation stores the data in an array of multifabs, where the array index refers to the refinement level.

All fabs in a given multifab have the same number of ghost cells, but different multifabs can have different numbers of ghost cells (or no ghost cells).

Working with multifabs

To build a multifab, we need to provide a layout, the number of components to store in the multifab and the number of ghostcells. In MAESTRO the hierarchy of grids will be described by a single ml_layout. A multifab can be declared and built at any time in a simulation using the ml_layout, thereby allocating space at every grid location in the simulation. The sequence to build a multifab appears as

```
  type(multifab) :: mfab(nlevs)
  ...
  do n = 1, nlevs
     call multifab_build(mfab(n), mla%la(n), nc, ng)
  enddo
```

Here, nc is the number of components and ng is the number of ghostcells. The multifab is built one level at a time, using the layout for that level taken from the ml_layout, mla.

A common operation on a multifab is to initialize it to 0 everywhere. This can be done (level-by-level) as

```
call setval(mfab(n), ZERO, all=.true.)
```

where ZERO is the constant 0.0 from bl_constants_module.

The procedure for accessing the data in each grid managed by the `multifab` is shown in § 5.8. Subroutines to add, multiply, or divide two `multifabs` exist, as do subroutines to copy from one `multifab` to another—see `amrex/Src/F_BaseLib/multifab.f90` for the full list of routines that work with `multifabs`.

When you are done working with a `multifab`, its memory can be freed by calling `multifab_destroy` on the `multifab`.

`bc_tower`

A `bc_tower` holds the information about what boundary conditions are in effect for each variable in a MAESTRO simulation. These are interpretted by the ghost cell filling routines. See § 5.10 for more detail.

MAESTRO **Data Organization**

The state of the star in MAESTRO is described by both a multidimensional state and the 1D base state. The full multidimensional state is stored in `multifabs` while the base state is simply stored in Fortran arrays. Here we describe the major MAESTRO data-structures.

‘s’ `multifab`**s (fluid state)**

The fluid state (density, enthalpy, species, temperature, and tracer) are stored together in a cell-centered multi-component `multifab`, typically named `sold`, `s1`, `s2`, or `snew` (depending on which time-level it represents). The enthalpy is stored as (ρh), and the species are stored as partial-densities (ρX_k). The tracer component is not used at present time, but can describe an arbitrary advected quantity.

Individual state variables should be indexed using the integer keys provided by the `variables` module (see §5.6.8). For example, the integer `rho_comp` will always refer to the density component of the state.

Note: the pressure is not carried as part of the ‘`s`’ `multifabs`.

‘u’ `multifab`**s (fluid velocity)**

The fluid velocity at time-levels n and $n+1$ is stored in a cell-centered multi-component `multifab`, typically named `uold` or `unew`. Here the `dm` components correspond to each coordinate direction.

`umac` **(the MAC velocity)**

In creating the advective fluxes, we need the time-centered velocity through the faces of the zone—the x-velocity on the x-edges, the y-velocity on the y-edges, etc. (see figure 5.6). This type of velocity discretization is termed the MAC velocity (after the “marker-and-cell” method for free boundaries in incompressible flows [22]).

The MAC velocities are allocated at each level of refinement, `n`, by making a `multifab` array where each of the `dm` components is nodal in its respective direction:

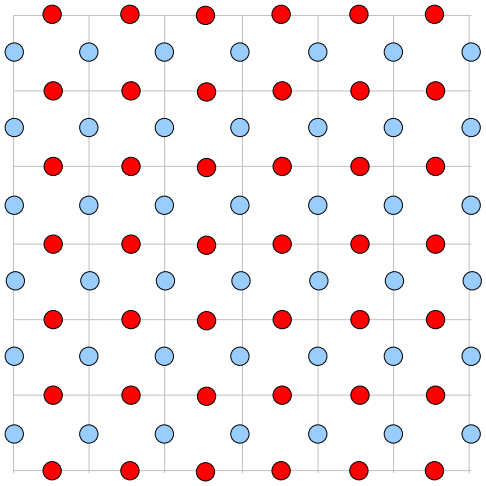


Figure 5.6: The MAC grid for the velocity. Here the x-velocity is on the x-edges (shown as the blue points) and the y-velocity is on the y-edges (shown as the red points).

```
type(multifab) :: umac(nlevel,dm)

do n=1,nlevel
   do comp=1,dm
      call multifab_build_edge(umac(n,comp), mla%la(n),1,1,comp)
   enddo
enddo
```

Base State Arrays

The base state is defined by ρ_0, p_0, and w_0. There is no base state composition. Other arrays are defined as needed, such as h_0, the base state enthalpy.

The base state arrays are 2-dimensional, with the first dimension giving the level in the AMR hierarchy and the second the radial index into the base state. For spherical geometries, the base state only exists at a single level, so the first index will always be 1. The radial index is 0-based, to be consistent with the indexing for the Cartesian state data. For example, the base state density would be dimensioned: `rho0(nlevs,0:nr_fine-1)`. Here, `nlevs` is the number of levels of refinement and `nr_fine` is the number of cells in the radial direction at the finest level of refinement.

For multilevel, plane-parallel geometry, all grids at the same height will have the same resolution so that the full state data is always aligned with the base state (see Figure 5.1). Base state data on coarse grids that are covered by fine grids is not guaranteed to be valid.

For spherical problems, the base state resolution, Δr, is generally picked to be finer than the Cartesian grid resolution, Δx, i.e. $\Delta r < \Delta x$. The ratio is controlled by the parameter `drdxfac`.

Note there are no ghost cells for the base state outside of the physical domain. For plane-parallel, multilevel simulations, there are ghostcells at the jumps in refinement—these are filled by the `fill_code_base` routine. The convention when dealing with the base state is that we only access it inside of the valid physical domain. Any multi-dimensional quantity that is derived using the base state then has its ghost cells filled by the usually `multifab` ghost cell routines.

MAESTRO **Helper Modules**

A number of MAESTRO modules appear frequently throughout the source. Below, we describe some of the more common functionality of the most popular modules.

`average_module`

The `average_module` module provides a routine `average` that takes a multilevel `multifab` array and averages the full Cartesian data onto the 1D base state.

`eos_module`

The `eos_module` provides the interface to the equation of state to connect the state variables thermodynamically. It gets the information about the fluid species from the `network` module (for example, the atomic number, Z, and atomic weight, A, of the nuclei).

Presently there is a single EOS that comes with MAESTRO, tt gamma_law_general, but many more are available through the external Microphysics repo[3]. The Microphysics EOSs share the same interface and can be compiled into MAESTRO directly. Here are the more popular EOSs:

- `helmholtz` represents a general stellar equation of state, consisting of nuclei (as an ideal gas), radiation, and electrons (with arbitrary degeneracy and degree of relativity). This equation of state is that described in [32].

  A runtime parameter, `use_eos_coulomb`, is defined in this EOS to enable/disable Coulomb corrections.

- `gamma_law_general` assumes an ideal gas with a mixed composition and a constant ratio of specific heats, γ:

  $$p = \rho e(\gamma - 1) = \frac{\rho k_B T}{\mu m_p} \tag{5.2}$$

  where k_B is Boltzmann's constant and m_p is the mass of the proton. The mean molecular weight, μ, is computed assuming electrically neutral atoms:

  $$\mu = \left(\sum_k \frac{X_k}{A_k} \right)^{-1} \tag{5.3}$$

  An option in the source code itself exists for treating the species as fully-ionized, but there is no runtime-parameter to make this switch.

- `multigamma` is an ideal gas equation of state where each species can have a different value of γ. This mainly affects how the internal energy is constructed as each species, represented with a mass fraction X_k will have its contribution to the total specific internal energy take the form of

[3] Microphysics is available at `https://github.com/starkiller-astro/Microphysics`. MAESTRO will find it via the `MICROPHYSICS_HOME` environment variable

$e = p/\rho/(\gamma_k - 1)$. The main thermodynamic quantities take the form:

$$p = \frac{\rho k T}{m_u} \sum_k \frac{X_k}{A_k} \tag{5.4}$$

$$e = \frac{kT}{m_u} \sum_k \frac{1}{\gamma_k - 1} \frac{X_k}{A_k} \tag{5.5}$$

$$h = \frac{kT}{m_u} \sum_k \frac{\gamma_k}{\gamma_k - 1} \frac{X_k}{A_k} \tag{5.6}$$

We recognize that the usual astrophysical $\bar{A}^{-1} = \sum_k X_k/A_k$, but now we have two other sums that involve different γ_k weightings.

The specific heats are constructed as usual,

$$c_v = \left.\frac{\partial e}{\partial T}\right|_\rho = \frac{k}{m_u} \sum_k \frac{1}{\gamma_k - 1} \frac{X_k}{A_k} \tag{5.7}$$

$$c_p = \left.\frac{\partial h}{\partial T}\right|_p = \frac{k}{m_u} \sum_k \frac{\gamma_k}{\gamma_k - 1} \frac{X_k}{A_k} \tag{5.8}$$

and it can be seen that the specific gas constant, $R \equiv c_p - c_v$ is independent of the γ_i, and is simply $R = k/m_u\bar{A}$ giving the usual relation that $p = R\rho T$. Furthermore, we can show

$$\Gamma_1 \equiv \left.\frac{\partial \log p}{\partial \log \rho}\right|_s = \left(\sum_k \frac{\gamma_k}{\gamma_k - 1} \frac{X_k}{A_k}\right) \Big/ \left(\sum_k \frac{1}{\gamma_k - 1} \frac{X_k}{A_k}\right) = \frac{c_p}{c_v} \equiv \gamma_{\text{effective}} \tag{5.9}$$

and $p = \rho e(\gamma_{\text{effective}} - 1)$.

This equation of state takes several runtime parameters that can set the γ_i for a specific species:

- `eos_gamma_default`: the default γ to apply for all species
- `species_X_name` and `species_X_gamma`: set the γ_i for the species whose name is given as `species_X_name` to the value provided by `species_X_gamma`. Here, `X` can be one of the letters: `a`, `b`, or `c`, allowing us to specify custom γ_i for up to three different species.

The thermodynamic quantities are stored in a Fortran type `eos_t`, which has fields for all the thermodynamic inputs and outputs. The type definition is brought in through `eos_type_module`. [4]

The first argument to the `eos` call is an integer key that specifies which thermodynamic variables (in addition to the mass fractions) are used as input. EOS input options are listed in table 5.1.

`fill_3d_module`

The `fill_3d_module` provides routines that map from the 1D base state to the full Cartesian 2- or 3D state. Variations in the routines allow for cell-centered or edge-centered data on either the base state or full Cartesian state.

[4] Note: an older interface to the EOS exists, but is deprecated. In this mode, the `eos_old_interface` module declares the variables that need appear in the old-style `eos` call argument list. MAESTRO routines use these module variables in the EOS call to avoid having to declare each quantity in each routine that calls the EOS. Most code has been updated to use the new interface.

Table 5.1: EOS input flags

key	input quantities
`eos_input_rt`	ρ, T
`eos_input_rh`	ρ, h
`eos_input_tp`	T, p
`eos_input_rp`	ρ, p
`eos_input_re`	ρ, e
`eos_input_ps`	p, s

`fundamental_constants_module`

The `fundamental_constants_module` provides a simple list of various fundamental constants (e.g. Newton's gravitational constant) in CGS units.

`geometry`

`network`

The `network` module defines the number species advected by the code (`nspec`), their ordering, and gives their basic properties (like atomic number, Z, and atomic mass, A). All MAESTRO problems require a `network` module, even if there are no reactions modeled. Many different reaction modules (containing different sets of isotopes) exist in `Microphysics/networks`. The particular network used by a problem is defined in the problem's `GNUmakefile`.

To find the location of a particular species (for instance, "carbon-12") in the allowed range of `1:nspec`, you do the following query:

```
ic12 = network_species_index("carbon-12")
```

If the resulting index is `-1`, then the requested species was not found.

`probin_module`

`probin_module` provides access to the runtime parameters. The runtime parameters appear simply as module variables. To get the value of a parameter, one simply needs to '`use probin_module`'. The preferred method is to add the '`only`' clause to the `use` statement and explicitly list only those parameters that are used in the routine. Defining new runtime parameters is described in § 5.3.2.

`variables`

The `variables` module provides integer keys to index the state multifabs and other arrays dealing with the scalar quantities. The most commonly used keys are are list in table 5.2.

The species indices are contiguous in the state array, spanning `spec_comp:spec_comp-1+nspec`. To find a particular species, a query can be made through the `network` module, such as:

Table 5.2: Common `variables` module keys

`rho_comp`	density
`rhoh_comp`	density $\times$ specific enthalpy, (ρh)
`spec_comp`	first species partial density, (ρX_1)
`temp_comp`	temperature

```
ic12 = network_species_index("carbon-12")
```

and then the `fab` can be indexed using `spec_comp-1+ic12` to get "carbon-12". The `variables` module also provides keys for the plotfile variables and boundary condition types.

Other keys in the `variables` modules are reserved for boundary conditions (`foextrap_comp` and `hoextap_comp`), the projection of the pressure (`press_comp`), or constructing the plotfile.

AMReX Helper Modules

There are a large number of modules in `amrex/` that provide the core functionality for managing grids. Here we describe the most popular such modules.

`bl_types`

The main purpose of this module is to define the Fortran kind `dp_t` which is used throughout the code to declare double precision variables.

`bl_constants`

This module provides descriptive names for a number of common double precision numbers, e.g. `ONE = 1.0_dp_t`. This enhances the readability of the code.

`parallel`

All MPI calls are wrapped by functions in the `parallel` module. For serial jobs, the wrappers simply do the requested operation on processor. By wrapping the calls, we can easily switch between serial and parallel builds.

Example: Accessing State and MAC Data

In MAESTRO, the state data is stored in a cell-centered multifab array (the array index refers to the AMR level) and the MAC velocities are stored in a 2D nodal multifab array (with indices referring to the AMR level and the velocity component). Here we demonstrate a typical way to extract the state and MAC velocity data.

All MAESTRO routines are contained in a module, to allow for compile-time argument checking.

```
module example_module

contains
```

The main interface to our routine is called `example`—this will take the `multifab`s containing the data and then pass them to the work routines, `example_2d` or `example_3d`, depending on the dimensionality.

```
  subroutine example(mla,s,umac,dx,dt)

    use bl_types
    use multifab_module
    use ml_layout_module
    use variables, only: rho_comp
```

Here, the `bl_types` and `multifab_module` modules bring in the basic AMReX data types. Specifically, here, `bl_types` defines `dp_t` which is the `kind` used for declaring double precision data, and `multifab_module` defines the `multifab` data type. The `ml_layout_module` defines the datatype for a `ml_layout`—many routines will take an `ml_layout`to allow us to fill ghostcells. The `variables` module is a MAESTRO module that provides integer keys for indexing the state arrays. In this case the integer `rho_comp` refers to the location in the state array corresponding to density.

Next we declare the subroutine arguments:

```
    type(ml_layout), intent(in   ) :: mla
    type(multifab) , intent(inout) :: s(:)
    type(multifab) , intent(inout) :: umac(:,:)
    real(kind=dp_t), intent(in   ) :: dx(:,:),dt
```

Here, `s(:)` is our `multifab` array that holds the state data. with the array index in `s` refers to the AMR level. The MAC velocities are held in the multifab `umac`, with the array indices referring to the AMR level and the component.

Local variable declarations come next:

```
    ! Local variables
    real(kind=dp_t), pointer :: sp(:,:,:,:)
    real(kind=dp_t), pointer :: ump(:,:,:,:), vmp(:,:,:,:), wmp(:,:,:,:)
    integer :: i,n,dm,nlevs,ng_sp,ng_um
    integer :: lo(mla%dim),hi(mla%dim)
```

Amongst the local variables we define here are a pointer, `sp`, that will point to a single `fab` from the `multifab` `s`, and a pointer for each component of the MAC velocity, `ump`, `vmp`, and `wmp` (for a 2D run, we won't use `wmp`). We note that regardless of the dimensionality, these pointers are 4-dimensional: 3 spatial + 1 component.

Next we get the dimensionality and number of levels

```
    dm = mla%dim
    nlevs = mla%nlevel
```

Each multifab can have their own number of ghostcells, so we get these next:

```
    ng_sp = nghost(s(1))
    ng_um = nghost(umac(1,1))
```

By convention, all levels in a given multifab have the same number of ghostcells, so we use level 1 in the `nghost()` call. We also use the same number of ghostcells for each component of the velocity, so we only need to consider the first component in the `nghost()` call. The ghostcells will be needed to access the data stored in the `fabs`.

To access the data, we loop over all the levels, and all the boxes in the given level.

```
    do n=1,nlevs
       do i = 1, nfabs(s(n))
```

`nfabs(s(n))` is simply the number of boxes in level `n` on the current processor. Each processor knows which `fabs` in its `multifab`are local to that processor, and this loop will only loop over those.

For a given `box`, we get the data and the bounds of the `box`.

```
          sp  => dataptr(s(n), i)
          ump => dataptr(umac(n,1),i)
          vmp => dataptr(umac(n,2),i)
          lo =  lwb(get_box(s(n), i))
          hi =  upb(get_box(s(n), i))
```

The actual data array is accessed through the `dataptr` function, which takes a `multifab` (e.g. `s(n)`) and the index of the `box` (`i`) we want. We see that the x MAC velocity for the current box is stored in `ump` and the y MAC velocity is stored in `vmp`. We don't get the z velocity data here, since that would not be available for a 2D run—we defer that until we test on the dimensionality below.

Finally, the index bounds of the box (just the data, not the ghostcells) are stored in the `dm`-dimensional arrays `lo` and `hi`. These indices refer to the current box, and hold for both the state, `sp`, and the MAC velocity, `ump` and `vmp`. However, since the MAC velocity is nodal in the component direction, the loops over the valid data will differ slight (as we see below).

With the data extracted, we call a subroutine to operate on it. We use different subroutines for the different dimensionalities (and many times have a separate routine for spherical geometries).

```
          select case (dm)
          case (2)
             call example_2d(sp(:,:,1,rho_comp),ng_sp, &
                             ump(:,:,1,1),vmp(:,:,1,1),ng_um, &
                             lo,hi,dx(n,:),dt)
          case (3)
             wmp => dataptr(umac(n,3),i)
             call example_3d(sp(:,:,:,rho_comp),ng_sp, &
                             ump(:,:,:,1),vmp(:,:,:,1),wmp(:,:,:,1),ng_um, &
                             lo,hi,dx(n,:),dt)
          end select
       enddo    ! end loop over boxes

    enddo    ! end loop over levels

  end subroutine example
```

We call either the function example_2d for two-dimensional data or example_3d for three-dimensional data. Note that in the two-dimensional case, we index the data as sp(:,:,1,rho_comp). Here a '1' is used as the 'z'-coordinate spatial index, since this is a 2D problem, and the density component of the state is selected (using the integer key rho_comp). The 3D version accesses the data as sp(:,:,:,rho_comp)—only the component regarding the variable is needed here. Notice that we also pass through the number of ghostcells for each of the quantities.

This routine will be supplimented with example_2d and example_3d, which actually operate on the data. The form of the 2D function is:

```
  subroutine example_2d(density,ng_sp, &
                        umac,vmac,ng_um, &
                        lo,hi,dx,dt)

    use bl_constants_module
    use probin_module, only: prob_lo

    integer        , intent(in) :: lo(:),hi(:), ng_sp, ng_um
    real(kind=dp_t), intent(in) :: density(lo(1)-ng_sp:,lo(2)-ng_sp:)
    real(kind=dp_t), intent(in) ::    umac(lo(1)-ng_um:,lo(2)-ng_um:)
    real(kind=dp_t), intent(in) ::    vmac(lo(1)-ng_um:,lo(2)-ng_um:)

    real(kind=dp_t), intent(in) :: dx(:),dt

    integer         :: i, j
    real(kind=dp_t) :: x, y
    real(kind=dp_t) :: dens, u, v

    do j = lo(2), hi(2)
       y = prob_lo(2) + (dble(j) + HALF)*dx(2)

       do i = lo(1), hi(1)
          x = prob_lo(1) + (dble(i) + HALF)*dx(1)

          dens = density(i,j)

          ! compute cell-centered velocity
          u = HALF*(umac(i,j) + umac(i+1,j))
          v = HALF*(vmac(i,j) + vmac(i,j+1))

          ! operate on the data
          ! ...

       enddo
    enddo

  end subroutine example_2d

end module example_module
```

In this function, the bounds of the density array take into account the ng_sp ghostcells and the index space of the current box. Likewise, the MAC velocities refer to the ng_um ghostcells. The j and i loops loop over all the valid zones. Coordinate information is computed from dx and prob_lo which

is the physical lower bound of the domain. `bl_constants_module` declares useful double-precision constants, like `HALF` (0.5). Here, we see how to access the density for the current zone and compute the cell-centered velocities from the MAC velocities. By convection, for a nodal array, the indices refer to the *lower* interface in the nodal direction, so for `umac`, `umac(i,j)` and `umac(i+1,j)` are the x MAC velocities on the lower and upper edge of the zone in the x-direction.

The three-dimensional case is similar, with the `density` array declared as

```
density(lo(1)-ng_sp:,lo(2)-ng_sp:,lo(3)-ng_sp:)
```

and an additional loop over the 'z' coordinate (from `lo(3)` to `hi(3)`).

In this example, we looped over the valid zones. If we wished to loop over the interfaces bounding the valid zones, in the x-direction, we would loop as

```
do j = lo(2), hi(2)
   do i = lo(1), hi(1)+1
      ! access umac(i,j)
   enddo
enddo
```

Filling Ghostcells

Ghostcells are filled through a variety of different routines, depending on the objective.

- `multifab_fill_boundary` fills ghost cells for two adjacent grids at the same level, which als includes periodic domain boundary ghost cells.
- `multifab_physbc` fills ghostcells at the physical boundaries.
- `multifab_fill_ghost_cells` is used for multilevel problems, and fills ghostcells in the finer grid (level `n`) by interpolating from data in the coarser grid (level `n-1`). This function, by default, will also call `multifab_fill_boundary` and `multifab_physbc` for both levels `n` and `n-1` (you can override this behavior for speed optimization purposes). This call is usually preceded by a call to `ml_cc_restriction_c` which sets the level `n-1` data to be the average of the level `n` data covering it.

You generally won't see calls in the MAESTRO source code to these subroutines, as there is now a special AMReX subroutine, `ml_restrict_and_fill`, that takes an array of multifabs at different level, and in order calls: (1) `ml_cc_restriction_c`, (2) `multifab_fill_boundary`, (3) `multifab_physbc`, and (4) `multifab_fill_ghost_cells`. These four subroutines are called in such a way to avoid extra ghost-cell filling, saving on communication time. You can specify the starting component, starting boundary condition component, the number of components, the number of ghost cells, and whether or not you want to use the same boundary condition component for all variables.

Boundary Conditions

When MAESTRO is run, the boundary condition parameters are read in from the input file and used to build the `bc_tower`. The `bc_tower` consists of a `bc_level` object for each level of resolution

in the simulation. The bc_level contains 3 different descriptions of the boundary conditions for each box in the domain at that level of refinement: phys_bc_level_array, adv_bc_level_array, and ell_bc_level_array. In all cases, the boundary conditions are specified via integer values that are defined in bc_module (part of AMReX).

Each level has a phys_bc_level_array(0:ngrids,dim,2) array, where ngrids is the number of boxes on that level, dim is the coordinate direction, and the last index refers to the lower (1) or upper (2) edge of the box in the given coordinate direction. This stores the *physical desciption* of the boundary type (outlet, inlet, slipwall, etc.)—this description is independent of the variables that live on the grid. The phys_bc_level_array(0,:,:) 'box' refers to the entire domain. If an edge of a box is not on a physical boundary, then it is set to a default value (typically INTERIOR). These boundary condition types are used to interpret the actual method to fill the ghostcells for each variable, as described in adv_bc_level_array and ell_bc_level_array.

Whereas phys_bc_level_array provides a physical description of the type of boundary, the array adv_bc_level_array describes the *action* taken (e.g. reflect, extrapolate, etc.) for each variable when filling boundaries. adv_bc_level_array specifically describes the boundary conditions that are in play for the advection (hyperbolic) equations. The form of this array is adv_bc_level_array(0:ngrids,dim,2,nvar) where the additional component, nvar, allows for each state variable that lives on a grid to have different boundary condition actions associated with it. The convention is that the first dm variables in bc_level(where dm is the dimensionality of the simulation) refer to the velocity components, and the subsequent slots are for the other standard variables described in the variables_module. For instance, to reference the boundary condition for density, one would index with dm+rho_comp. For temporary variables that are created on the fly in the various routines in MAESTRO there may not be a variable name in variables_module that describes the temporary variable. In this case, the special variables foextrap_comp and hoextrap_comp (first-order and high-order extrapolation) are used.

ell_bc_level_array is the analog to adv_bc_level_array for the elliptic solves in MAESTRO. This will come into play in the multigrid portions of the code. The actions that are used for ell_bc_level_array are either Dirichlet or Neumann boundary condtions. For the velocity projections, we are dealing with a pressure-like quantity, ϕ, so the pressure boundary conditions here reflect the behavior we want for the velocity. After the projection, it is $\nabla\phi$ that modifies the velocity field. At a wall or for inflow conditions, we already have the velocity we want at the boundary, so we want the velocity to remain unchanged after the projection. This requires $d\phi/dn = 0$ on those boundaries. For outflow, we impose a condition that we do not want the boundaries to introduce any tangental acceleration (or shear), this is equivalent to setting $\phi = 0$ (then $\partial\phi/\partial t = 0$, with t meaning 'tangental'). This allows the velocity to adjust as needed to the domain (see, for example, [7]).

The actual filling of the ghostcells according to the descriptions contained in the bc_tower is carried out by the multifab_physbc routine. When you have an EXT_DIR condition in multifab_physbc (an specified in the inputs file as inlet), the advection solver (via the slope routine) and linear solvers will then assume that the value in the ghost cells is equal to the value that actually lies on the wall.

Multigrid

MAESTRO uses the multigrid solver to enforce the divergence constraint both on the half-time edge-centered advective velocities (the "MAC projection") and on the final cell-centered velocities (the "HG projection"). For the MAC projection, since the velocity data is edge-centered (the MAC grid), the

hould draw a figure

projection is cell-centered. For the HG projection, since the velocity data is cell-centered, the projection is node-centered. The multigrid solver performs a number of V-cycles until the residual drops by 10-12 orders-of-magnitude. There are several options that affect how the multigrid solver behaves, which we describe below. More detail on the multigrid solvers is given in Chapter 22.

Multilevel and Refinement Criteria

Particles

MAESTRO has support for Lagrangian particles that are passively advected with the velocity field. These are useful for diagnostics and post-processing. To use particles, particles must be seeded into the domain by writing a problem-specific `init_particles.f90` routine. This routine is called at the start of the simulation. The `init_particles` routines add particles at specific locations by calling the `particle_module`'s `add` routine when a given criteria is met by the fluid state.

When you run the code, particles are enabled by setting `use_particles = T`. At the end of each timestep the locations of all the particles are written out into a series of files called `timestamp_NN`, where `NN` is the CPU number on which the particle *currently* resides. Particles are always kept on the processor containing the state data corresponding to their present location. Several bits of associated data (density, temperature, and mass fractions) are stored along with the particle ID and position.

Some simple python scripts allow for the plotting of the particle positions. See § 9.1.3 for details.

Regression Testing

There is an extensive regression test suite for AMReX that works with MAESTRO. Full details, and a sample MAESTRO configuration file are provided in the AMReX User's Guide and source.

CHAPTER 6

Runtime Parameters

Introduction to Runtime Parameters

The runtime parameters are defined in the `MAESTRO/_parameters` files, and any other problem-specific parameter files. These parameters are then made available to the code through the `probin_module`.

Any runtime parameters defined by the microphysics (either in the `MAESTRO/Microphysics/` source or the external `Microphysics/` source are also parsed at build time and read in via the same namelist in `probin.f90`. These microphysics runtime parameters can be accessed via `extern_probin_module`.

Parameter definitions take the form of:

```
# comment describing the parameter
name              data-type       default-value      priority
```

Here, the `priority` is simply an integer. When two directories define the same parameter, but with different defaults, the version of the parameter with the highest priority takes precedence. This allows specific implementations to override the general parameter defaults.

The following tables list all of the general MAESTRO runtime parameters. These tables are generated automatically using the `rp.py` script in `MAESTRO/docs/runtime_parameters/` by parsing the `MAESTRO/_parameters` file. The problem-specific parameters are not shown here.

Table 6.1: EOS parameters.

parameter	description	default value
small_dens		1.d-5
small_temp		5.d6
use_eos_e_instead_of_h	In deriving the temperature from the h, first subtract off p_0/ρ to define e, and use that as the input to the EOS.	.false.
use_pprime_in_tfromp		.false.
use_tfromp	When updating temperature, use $T = T(\rho, p_0, X)$ rather than $T = T(\rho, h, X)$.	.false.

Table 6.2: SDC parameters.

parameter	description	default value
sdc_couple_mac_velocity	recompute MAC velocity at the beginning of each SDC iter	.false.
sdc_iters	how many SDC iteractions to take (requires the code be compiled with SDC := t	1

Table 6.3: algorithm initialization parameters.

parameter	description	default value
do_initial_projection	Do the initial projection.	.true.
init_divu_iter	Number of initial divu iterations.	4
init_iter	Number of initial pressure iterations.	4
restart	which file to restart from. -1 means do not restart	-1
restart_into_finer	restart and add a level of refinement	.false.

Table 6.4: base state mapping parameters.

parameter	description	default value
s0_interp_type	The interpolation for filling a cell-centered multifab from a 1D bin-centered array. 1 = piecewise constant; 2 = piecewise linear; 3 = quadratic	3
s0mac_interp_type	The interpolation for filling an edge based multifab from a 1D bin-centered array. 1 = Interpolate s0 to cell centers (with s0_interp_type), then average to edges; 2 = Interpolate s0 to edges directly using linear interpolation; 3 = Interpolate s0 to edges directly using quadratic interpolation.	1
w0_interp_type	The interpolation for filling a cell-centered multifab from a 1D edge-centered array. 1 = piecewise constant; 2 = piecewise linear; 3 = quadratic	2
w0mac_interp_type	The interpolation for putting w0 on edges. We only compute the normal component. 1 = Interpolate w0 to cell centers (with w0_interp_type), then average to edges; 2 = Interpolate w0 to edges directly using linear interpolation; 3 = Interpolate w0 to edges directly using quadratic interpolation; 4 = Interpolate w0 to nodes using linear interpolation, then average to edges.	1

Table 6.5: burning parameters.

parameter	description	default value
burner_threshold_cutoff	Mass fraction cutoff for burner_threshold_species used in burner threshold	1.d-10
burner_threshold_species	Name of the species to be used in burner threshold	""
do_burning	turn on (.true.) or off (.false.) burning	.true.
do_subgrid_burning	break a zone into subzones, call the burner in each subzone and then average the result to the original cell	.false.
reaction_sum_tol	mass fraction sum tolerance (if they don't sum to 1 within this tolerance, we abort)	1.d-10

Table 6.6: debugging parameters.

parameter	description	default value
boxlib_fpe_invalid	enable floating point exception trapping for invalid	.false.
boxlib_fpe_overflow	enable floating point exception trapping for overflow	.false.
boxlib_fpe_zero	enable floating point exception trapping for divide by zero	.false.

Table 6.7: general MAESTRO parameters.

parameter	description	default value
barrier_timers	Call a parallel_barrier before each of the simple timers we have coded in advance.f90. Might cause a very slight overall performance hit.	.false.
job_name	job name printed in output	""
the_knapsack_verbosity	Verbosity of the knapscak processor-to-grid algorithm.	.false.
verbose	General verbosity	0

Table 6.8: grid parameters.

parameter	description	default value
amr_buf_width	the number of buffer zones surrounding a cell tagged for refinement. note that this needs to be ¿= regrid_int	-1
bcx_hi	$+x$ boundary condition (valid values are listed in boxlib/bc.f90)	SLIP_WALL
bcx_lo	$-x$ boundary condition	SLIP_WALL
bcy_hi	$+y$ boundary condition	SLIP_WALL

continued on next page

Table 6.8—continued

parameter	description	default value
bcy_lo	$-y$ boundary condition	SLIP_WALL
bcz_hi	$+z$ boundary condition	SLIP_WALL
bcz_lo	$-z$ boundary condition	SLIP_WALL
blocking_factor	grids must be an integer multiple of this number, if possible	8
change_max_grid_size_1	Change the max grid size on the base level on restart	.false.
dm_in	dimensionality (valid values are 2 or 3)	2
do_2d_planar_octant	Set to 1 if using the 2D simplified (planar) model of an octant.	0
drdxfac	ratio of radial base state zones to Cartesian full state zones for spherical geometry	1
dump_grid_file	dump out a file, named grids.out, containing the number of grids a each level. A new line is added each time regrid is called.	.false.
max_grid_size	The largest grid size that will be created using make_new_grids.	64
max_grid_size_1	The largest grid size that will be created using make_new_grids for the coarsest level. Defaults to max_grid_size.	-1
max_grid_size_2	The largest grid size that will be created using make_new_grids for level 2 (the first refined level). Defaults to max_grid_size.	-1
max_grid_size_3	The largest grid size that will be created using make_new_grids for level 3 (the second refined level) and beyond. Defaults to max_grid_size.	-1
max_levs	Total number of levels. 1 = single level.	1
min_eff	parameter for cluster algorithm for making new grids in adaptive problems	0.9d0
minwidth	The minimum size on a side for a grid created using make_new_grids.	8
n_cellx	Number of cells for the base level in the x-direction	-1
n_celly	Number of cells for the base level in the y-direction	-1
n_cellz	Number of cells for the base level in the z-direction	-1
octant	set octant = T if you just want to model an octant of a sphere (note: only takes effect for spherical geometry)	.false.
prob_hi_x	physical coordinates of hi-x corner of problem domain	1.d0

continued on next page

Table 6.8—continued

parameter	description	default value
prob_hi_y	physical coordinates of hi-y corner of problem domain	1.d0
prob_hi_z	physical coordinates of hi-z corner of problem domain	1.d0
prob_lo_x	physical coordinates of lo-x corner of problem domain	ZERO
prob_lo_y	physical coordinates of lo-y corner of problem domain	ZERO
prob_lo_z	physical coordinates of lo-z corner of problem domain	ZERO
ref_ratio	Refinement ratio for multilevel problems	2
regrid_int	How often we regrid.	-1
spherical_in	Set to 1 if you are doing a spherical problem.	0
test_set	Fixed grid file.	″
the_copy_cache_max	Number of boxassoc layouts we keep in memory to avoid having to recompute the boxassoc, which is computationally expensive.	128
the_layout_verbosity		0
the_ml_layout_strategy	This prarmeter has no impact on single-level runs! 0: This uses either knapsack or sfc on each level. Before boxes are distributed, MPI ranks are sorted according to the memory usage by all levels. This is our traditional approach. 1: This uses sfc on each level. MPI ranks are sorted according to the memory usage by lower levels.	0
the_sfc_threshold	When assigning processors for grids, this determines whether we use the sfc algorithm or knapsack algorithm. If the total number of grids divided by the number of processors is greater than this number, use sfc.	5
use_tpert_in_tagging	pass T' into the tagging routines as the auxillary multifab instead of the default ρH_{nuc}.	.false.
xhi_boundary_type	$+x$ boundary condition name—if this is set, it overrides the integer value set through bcx_hi	″″
xlo_boundary_type	$-x$ boundary condition name—if this is set, it overrides the integer value set through bcx_lo	″″
yhi_boundary_type	$+y$ boundary condition name—if this is set, it overrides the integer value set through bcy_hi	″″

continued on next page

Table 6.8—continued

parameter	description	default value
ylo_boundary_type	$-y$ boundary condition name—if this is set, it overrides the integer value set through bcy_lo	""
zhi_boundary_type	$+z$ boundary condition name—if this is set, it overrides the integer value set through bcz_hi	""
zlo_boundary_type	$-z$ boundary condition name—if this is set, it overrides the integer value set through bcz_lo	""

Table 6.9: heating parameters.

parameter	description	default value
do_heating	use analytic heating	.false.

Table 6.10: hydrodynamics parameters.

parameter	description	default value
anelastic_cutoff	The density below which we modify the constraint to look like the anelastic constraint, instead of the low Mach constraint. This prevents velocities from getting out of hand at the edge of the star. Refer to Section 13.3.1.	3.d6
base_cutoff_density	The density below which we keep the initial model constant. Refer to Section 13.3.2	3.d6
bds_type	0 = use ppm instead for multi-d integrator 1 = bilinear	0
beta_type	what type of coefficient to use inside the velocity divergence constraint. beta_type = 1 uses β_0; beta_type = 2 uses ρ_0 (anelastic); beta_type = 3 uses 1 (small-scale combustion).	1

continued on next page

Table 6.10—continued

parameter	description	default value
buoyancy_cutoff_factor	The multiplicative factor (over base_cutoff_density) below which we do zero out the buoyancy term in the momentum equation.	5.0
co_latitude	latitude, in radians, for problems with rotation where the domain is only a subset of a full star.	ZERO
do_eos_h_above_cutoff	After the advective enthalpy update, recompute the enthalpy if we are below the base cutoff density.	.true.
do_planar_invsq_grav	are we doing $1/r^2$ gravity for plane-parallel	.false.
do_smallscale	force $\rho_0 = (\rho h)_0 = 0$, evolve_base_state = F and beta_type = 3	.false.
do_sponge	Use sponging.	.false.
dpdt_factor	factor in front of the volume discrepancy term (0.0 = off)	0.d0
drive_initial_convection	freeze the temperature used in the reaction network to the initial value. This is useful for developing an initial convective field to carry away the energy, while preventing the reactions from going nonlinear.	.false.

enthalpy_pred_type	predict_rhoh = 0;
	predict_rhohprime = 1;
	predict_h = 2;
	predict_T_then_rhohprime = 3;
	predict_T_then_h = 4;
	predict_hprime = 5;
	predict_Tprime_then_h = 6.
		predict_rhohprime

evolve_base_state	turn on (.true.) or off (.false.) basestate evolution	.true.
fix_base_state	if true, don't call average to reset the base state at all, even during initialization	.false.
grav_const	the gravitational acceleration (cm s^{-2}) for plane-parallel geometry	-1.5d10
mach_max_abort	maximum mach number before the code aborts	-1.d0
planar_invsq_mass	the point mass for planar $1/r^2$ gravity	0.d0
plot_sponge_fdamp	plot fdamp rather than sponge assumes sponge has the form 1/(1+dt*sponge_kappa*fdamp)	.false.

continued on next page

Table 6.10—continued

parameter	description	default value
ppm_trace_forces	if 1, then perform parabolic reconstruction on the forces used in the prediction and trace under the parabola to the interfaces the amount that can reach the interface over dt	0
ppm_type	0 = no ppm (piecewise linear slopes instead) 1 = 1984 ppm 2 = Hybrid Sekora/Colella and McCorquodale/Colella 2009/2010 ppm	1
restart_with_vel_field	restart the simulation using a result from a drive_initial_convection = T run note that this uses the restart variable to specify which file to restart from. After reading in the velocity information from the restart file, the time and timestep number are zeroed.	.false.
rotation_radius	radius used for computing centrifugal term in rotation problems	1.0d6
rotational_frequency	rotational frequency used for computing centrifugal term in rotation problems.	ZERO
slope_order	order of slopes in piecewise linear Godunov algorithm. Options are 0, 2, or 4.	4
species_pred_type	Which quantities do we predict to the edges for computing the (ρX) edge states? species_pred_type = 1 means predict ρ' and X separately. species_pred_type = 2 means predict the full (ρX) itself. species_pred_type = 3 means predict ρ and X separately.	1
sponge_center_density	Center of the inner sponge.	3.d6
sponge_kappa	Parameter for sponge. Problem dependent.	10.d0
sponge_start_factor	The sponge begins at sponge_center_density * sponge_start_factor.	3.333d0
stop_initial_convection	timestep beyond which we set drive_initial_convection = F	-1
use_alt_energy_fix	modify the momentum equation to have $(\beta_0/\rho)\nabla(\pi/\beta_0)$ instead of just $(1/\rho)\nabla(\pi)$	.true.
use_delta_gamma1_term	turns on second order correction to delta gamma1 term	.false.
use_etarho	turn on the etarho term as described in flow chart	.true.
use_linear_grav_in_beta	how to represent gravity in the β_0 integration: .true. = piecewise linear .false. = piecewise constant	.false.

Table 6.11: linear solvers parameters.

parameter	description	default value
cg_verbose	Verbosity of bottom solver	0
hg_bottom_solver	valid values are ≥ 0	-1
hg_cycle_type	Type of cycle used in the nodal multigrid – 1 = F-cycle, 2 = W-cycle, 3 = V-cycle	3
hg_dense_stencil	In hgproject, in 2D, use a 9 point Laplacian (.true.) or 5-point Laplacian (.false.). In 3D, use a 27 point Laplacian (.true.) or 7-point Laplacian (.false.).	.true.
max_mg_bottom_nlevels	if mg_bottom_solver == 4, then how many mg levels can the bottom solver mgt object have	1000
mg_bottom_nu	number of smoothing iterations to do after the multigrid bottom solver	10
mg_bottom_solver	valid values are ≥ 0	-1
mg_cycle_type	Type of cycle used in the MAC multigrid – 1 = F-cycle, 2 = W-cycle, 3 = V-cycle	3
mg_nu_1	number of smoothing iterations to do going down the V-cycle	2
mg_nu_2	number of smoothing iterations to do going up the V-cycle	2
mg_verbose	Verbsoity of the multigrid solver, but not the bottom solver.	0
use_hypre	use the hypre library	.false.

Table 6.12: output parameters.

parameter	description	default value
check_base_name	prefix to use in checkpoint file names	”chk”
chk_int	Number of timesteps between writing a checkpoint file	0
diag_buf_size	number of timesteps to buffer diagnostic output information before writing (note: not implemented for all problems)	10

continued on next page

Table 6.12—continued

parameter	description	default value
lUsingNFiles	If .true., use nOutFiles processors to write checkpoint and plotfiles. Fortran has the unfortunate feature of each processor only being able to write out 1-2GB each without crashing.	.true.
mini_plot_base_name	basename for the mini plotfiles	"miniplt"
mini_plot_deltat	rather than use a mini plot interval, output a mini plotfile every mini_plot_deltat in time	-1.d0
mini_plot_int	mini plot interval	-1
mini_plot_var1	names of specific variables to store in the mini plotfile	""
mini_plot_var2		""
mini_plot_var3		""
mini_plot_var4		""
mini_plot_var5		""
mini_plot_var6		""
mini_plot_var7		""
mini_plot_var8		""
mini_plot_var9		""
nOutFiles	If lUsingNFiles = .true., use this many processors to write checkpoint and plotfiles. Fortran has the unfortunate feature of each processor only being able to write out 1-2GB each without crashing.	64
plot_Hext	plot external heating (Hext) in plotfile	.false.
plot_Hnuc	plot nuclear energy generation rate (Hnuc) in plotfile	.true.
plot_ad_excess	plot the adiabatic excess	.false.
plot_base	plot w0_x, w0_y, w0_z, divw0, rho0, rhoh0, h0, and p0 in plotfile	.false.
plot_base_name	prefix to use in plotfile file names	"plt"
plot_cs	plot soundspeed	.false.

continued on next page

Table 6.12—continued

parameter	description	default value
plot_deltat	rather than use a plot interval, plot a file after the solution has advanced past plot_deltat in time	-1.d0
plot_eta	plot η_ρ in plotfile	.false.
plot_gpi	plot pi and grad(pi)	.true.
plot_h_with_use_tfromp	Turn on storing of enthalpy-based quantities in the plotfile when we are running with use_tfromp	.true.
plot_int	plot interval	0
plot_omegadot	plot omegadot in plotfile	.true.
plot_pidivu	plot pi * div(U) – this is a measure of conservation of energy	.false.
plot_processors	create a field in the plotfile storing the processor number for each zone	.false.
plot_spec	plot species and omegadot in plotfile	.true.
plot_trac	plot tracers in plotfile	.false.
single_prec_plotfiles	store the state data in single precision	.false.

Table 6.13: particles parameters.

parameter	description	default value
store_particle_vels	store the velocity of the particle?	.false.
use_particles	call the particle initialization, advection, etc. routines.	.false.

Table 6.14: problem initialization parameters.

parameter	description	default value
model_file	input model file	"model.hse"
perturb_model	Turn on a perturbation in the initial data. Problem specific.	.false.
print_init_hse_diag	print out HSE diagnostics as a function of r for the initial model	.false.

Table 6.15: thermal diffusion parameters.

parameter	description	default value
limit_conductivity	apply the conductivity limiting—if T, then we set the thermal coefficients all to 0 for $\rho <$ buoyancy_cutoff_factor * base_cutoff_density	.false.
temp_diffusion_formulation	How to compute the explicit thermal diffusion term. 1 = in terms of T; 2 = in terms of ρ, p_0, X.	2
thermal_diffusion_type	In the thermal diffusion solver, 1 = Crank-Nicholson; 2 = Backward Euler.	1
use_thermal_diffusion	Use thermal diffusion.	.false.

Table 6.16: timestepping parameters.

parameter	description	default value
cflfac	CFL factor to use in the computation of the advection timestep constraint	0.5d0
fixed_dt	Fix the time step. If -1.0, then use the standard time step.	-1.0d0
init_shrink	the multiplicative factor (≤ 1) to reduce the initial timestep as computed by the various timestep estimators	1.d0
max_dt	This is the maximum dt that is allowed	1.0d33

continued on next page

Table 6.16—continued

parameter	description	default value
max_dt_growth	The maximum scale factor that the time step is allowed to grow by per time step.	1.1d0
max_step	Maximum number of steps in the simulation.	1
nuclear_dt_fac	If $T^n_{max} > T^{n-1}_{max}$ set the new dt = min[dt, dt*nuclear_dt_fac*(T^{n-1}_{max} / ($T^n_{max} - T^{n-1}_{max}$))] for example, nuclear_dt_fac = 0.01 means don't let the max temp grow more than approximately 1 percent not checkpoint-compatible yet since it wouldn't be backwards compatible	-1.0d0
small_dt	the minimum allowed timestep – we abort if dt drops below this value	1.d-10
stop_time	simulation stop time	-1.d0
use_divu_firstdt	Use the divu constraint when computing the first time step.	.false.
use_soundspeed_firstdt	Use the soundspeed constraint when computing the first time step.	.false.

Table 6.17: gamma law general EOS parameters.

parameter	description	default value
eos_assume_neutral	do we assume that all the species are neutral atoms? otherwise, we assume complete ionization	.true.
eos_gamma	the constant ratio of specific heats (γ)	5.d0/3.d0

Table 6.18: generic network parameters parameters.

parameter	description	default value
small_x	cutoff for species mass fractions	0.0

Table 6.19: constant conductivity parameters.

parameter	description	default value
conductivity_constant	the constant conductivity (only applies if the conductivity/constant module is used)	1.0d0

CHAPTER 7

Initial Models

Here we briefly describe the various routines for generating initial models for the main MAESTRO problems and how the initial model is used to initialize both the base state and the full Cartesian state.

I think it would be a good idea to mentio some where in here that MAESTRO outputs info that's useful to check –dr c base state and dr of the input file, which should be the same at the finest level; and Maximum HSE Error, which should be some small number.

Creating the Model Data from Raw Data

We have found that for the best performance, the MAESTRO initialization procedure should be given model data with the same resolution as the base state resolution, Δr. For planar problems, $\Delta r = \Delta x$. For multilevel planar problems, we use Δr equal to Δx at the finest resolution. For spherical problems we set $\Delta r = \Delta x/\texttt{drdxfac}$. We generate our initial model either analytically or from raw data, $\rho^{\rm raw}$, $T^{\rm raw}$, $p^{\rm raw}$, and $X^{\rm raw}$. Here is the raw data file for each test problem:

$$\begin{aligned} \texttt{reacting\_bubble} &\rightarrow \text{none} - \text{it is generated analytically} \\ \texttt{test\_convect} &\rightarrow \text{none} - \text{it is generated analytically} \\ \texttt{wdconvect} &\rightarrow \texttt{model\_6.e8.raw} \\ \texttt{spherical\_heat} &\rightarrow \text{none} - \text{it is generated analytically} \end{aligned} \tag{7.1}$$

We use a fortran subroutine to interpolate the raw data, yielding the model data, $\rho^{\rm model}$, $T^{\rm model}$, $p^{\rm model}$, and $X^{\rm model}$. The fortran subroutine then uses an iterative procedure to modify the model data so that it is thermodynamically consistent with the MAESTRO equation of state (EOS), and also satisfies our chosen hydrostatic equilibrium (HSE) discretization,

$$\frac{p_r - p_{r-1}}{\Delta r} = \frac{\rho_r + \rho_{r-1}}{2} g_{r-1/2}, \tag{7.2}$$

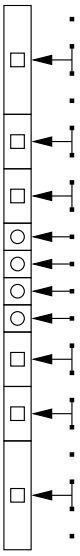


Figure 7.1: Multilevel Initialization. The data from the initial model is represented by the dots on the right. The initial data at the finest level is represented by the circles. The initial data at non-finest levels is represented by the squares. We copy data from the dots directly into the circles. We use linear interpolation using the two nearest points to copy data from the dots to the squares.

to a user defined tolerance. Here are the fortran subroutines for each test problem:

$$\begin{aligned} \texttt{reacting\_bubble} &\rightarrow \texttt{Util/initial\_models/test2/init\_1d.f90} \\ \texttt{test\_convect} &\rightarrow \texttt{Util/initial\_models/test2/init\_1d.f90} \\ \texttt{wdconvect} &\rightarrow \textit{none}^1 \\ \texttt{spherical\_heat} &\rightarrow \texttt{Util/initial\_models/spherical/init\_1d.f90} \end{aligned} \tag{7.3}$$

The model data is not generated at run-time—it must be generated in advance of running any MAESTRO examples. The inputs file should point to the file containing the model data.

Creating the Initial Data from the Model Data

In `base_state.f90`, using the subroutine `init_base_state` (which is actually a terrible name since we are computing 1D initial data, which is not quite the same thing as the base satte), we set the initial data $\rho^{\text{init}}, T^{\text{init}}, p^{\text{init}}$ and X^{init} equal to the model data. Then, we set $p^{\text{init}}, h^{\text{init}} = p, h(\rho^{\text{init}}, T^{\text{init}}, X^{\text{init}})$ through the EOS. Note that we overwrite the existing value of p^{init} but this should not change the value since we called $p^{\text{model}} = p^{\text{model}}(\rho^{\text{model}}, T^{\text{model}}, X^{\text{model}})$ at the end of the initial model generation routines in § 7.1.

For multilevel planar problems, we generate initial data at the non-finest levels by using linear interpolation from the two nearest model points (see Figure 7.2). Note that the initial data is not in HSE nor is it thermodynamically consistent on non-finest cells. We will enforce these on the base state and full state later.

We deal with the edge of the star by tracking the first coarse cell in which the density falls below `base_cutoff_density`. We note the radius of this cell center, and the values of ρ, ρh, X_k, p, and T in this cells. Then, at every level, if current cell-center is above this radius, we set the state equal to this stored state. This ensures a consistent treatment of the edge of the star at all levels.

[1] the original model routines are not distributed, but the `Util/initial_models/spherical/init_1d.f90` routines can work as a simple substitute.

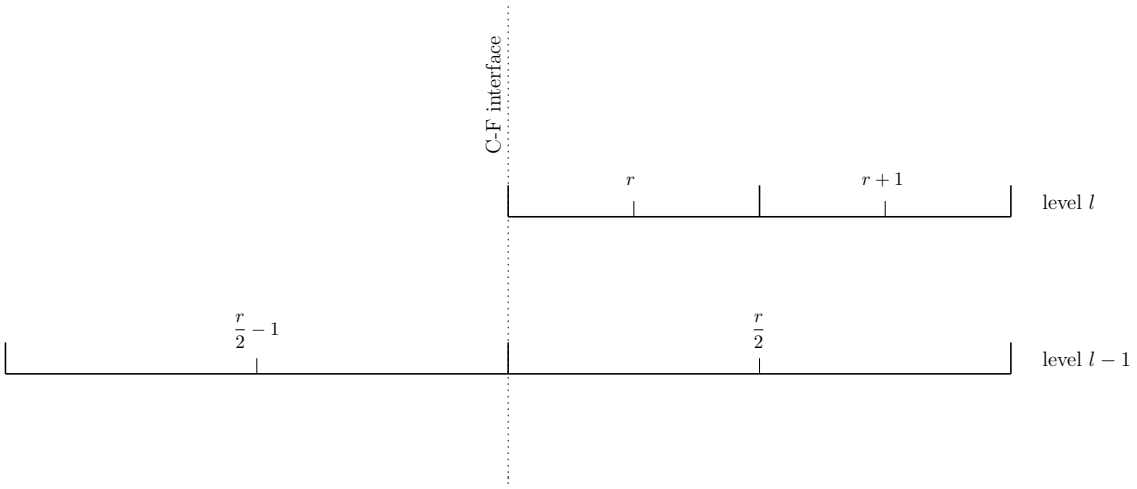


Figure 7.2: A coarse-fine interface in the 1-d base state

Creating the Base State and Full State

Given $p^{\text{init}}, \rho^{\text{init}}, T^{\text{init}}$, and X^{init}, this section describes how the base state (ρ_0 and p_0) and full state (ρ, h, X, and T) are computed. The base state is, general, not simply a direct copy of the initial model data, since we require that $\rho_0 = \overline{\rho}$. Additionally, we require thebase state to be HSE according to equation (7.2), and that the full state is thermodynamically consistent with p_0. Overall we do:

1. Fill $\rho^{\text{init}}, h^{\text{init}}, X^{\text{init}}$, and T^{init} onto a multifab to obtain the full state ρ, h, X, and T.
2. If `perturb_model = T`, a user-defined perturbation is added. This routine should make sure that the EOS is called so that there is some sense of thermodynamic consistency.
3. Set $\rho_0 = \overline{\rho}$.
4. Compute p_0 using `enforce_HSE`.
5. Compute $T, h = T, h(\rho, p_0, X_k)$.
6. Set $(\rho h)_0 = \overline{(\rho h)}$.
7. Compute $\overline{T}$. Note that we only use $\overline{T}$ as a diagnostic and as a seed for EOS calls.

Now $\rho_0 = \overline{\rho}$, the base state is in HSE, and the full state is thermodynamically consistent with p_0.

Coarse-Fine `enforce_HSE` Discretization

When integrating the HSE discretization upward, we must use a different differencing procedure at coarse-fine interfaces. Figure 7.2 shows the transition from coarse (level $l-1$) to fine (level l), with the zone center indices noted.

To find the zone-centered pressure in the first fine zone, p^l_r, from the zone-centered pressure in the coarse zone just below the coarse-fine interface, $p^{l-1}_{r/2-1}$, we integrate in 2 steps. We allow for a spatially changing gravitational acceleration, for complete generality.

First we integrate up to the coarse-fine interface from the coarse-cell center as:

$$\frac{p^l_{r-1/2} - p^{l-1}_{r/2-1}}{\Delta r^{l-1}/2} = \frac{\rho^l_{r-1/2} + \rho^{l-1}_{r/2-1}}{2} \frac{g^l_{r-1/2} + g^{l-1}_{r/2-1}}{2} \tag{7.4}$$

We can rewrite this as an expression for the pressure at the coarse-fine interface:

$$p^l_{r-1/2} = p^{l-1}_{r/2-1} + \frac{\Delta r^{l-1}}{8} \left(\rho^l_{r-1/2} + \rho^{l-1}_{r/2-1}\right) \left(g^l_{r-1/2} + g^{l-1}_{r/2-1}\right). \tag{7.5}$$

Next we integrate up from the coarse-fine interface to the fine-cell center:

$$\frac{p^l_r - p^l_{r-1/2}}{\Delta r^l/2} = \frac{\rho^l_r + \rho^l_{r-1/2}}{2} \frac{g^l_r + g^l_{r-1/2}}{2} \tag{7.6}$$

We can rewrite this as an expression for the pressure at the fine-cell center:

$$p^l_r = p^l_{r-1/2} + \frac{\Delta r^l}{8} \left(\rho^l_r + \rho^l_{r-1/2}\right) \left(g^l_r + g^l_{r-1/2}\right). \tag{7.7}$$

Combining equations 7.5 and 7.7 gives

$$\begin{aligned} p^l_r = p^{l-1}_{r/2-1} &+ \frac{\Delta r^{l-1}}{8} \left(\rho^l_{r-1/2} + \rho^{l-1}_{r/2-1}\right) \left(g^l_{r-1/2} + g^{l-1}_{r/2-1}\right) \\ &+ \frac{\Delta r^l}{8} \left(\rho^l_r + \rho^l_{r-1/2}\right) \left(g^l_r + g^l_{r-1/2}\right). \end{aligned} \tag{7.8}$$

We can simplify using

$$\Delta r^{l-1} = 2\Delta r^l, \tag{7.9}$$

and by interpolating the cell-centered densities to the coarse-fine interface as:

$$\rho^l_{r-1/2} = \frac{2}{3}\rho^l_r + \frac{1}{3}\rho^{l-1}_{r/2-1}. \tag{7.10}$$

Because we carry both the cell- and edge-centered gravitational accelerations, we do not need to interpolate g to the interface. Simplifying, we have

$$\begin{aligned} p^l_r = p^{l-1}_{r/2-1} &+ \frac{\Delta r^l}{4} \left(\frac{2}{3}\rho^l_r + \frac{4}{3}\rho^{l-1}_{r/2-1}\right) \left(g^l_{r-1/2} + g^{l-1}_{r/2-1}\right) \\ &+ \frac{\Delta r^l}{8} \left(\frac{5}{3}\rho^l_r + \frac{1}{3}\rho^{l-1}_{r/2-1}\right) \left(g^l_r + g^l_{r-1/2}\right) \quad . \end{aligned} \tag{7.11}$$

Finally, we note for constant g, this simplifies to:

$$p^l_r = p^{l-1}_{r/2-1} + \frac{3\Delta r^l g}{4} \left(\rho^{l-1}_{r/2-1} + \rho^l_r\right). \tag{7.12}$$

When integrating across a fine-coarse interface (see Figure 7.3), the proceduce is similar. The expression

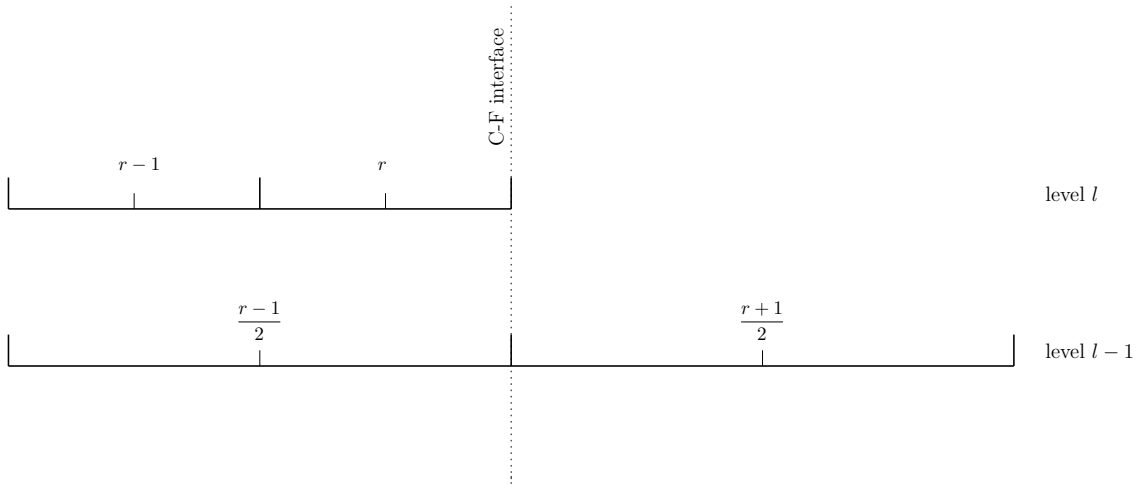


Figure 7.3: A fine-coarse interface in the 1-d base state

for general gravity becomes:

$$\begin{aligned} p^{l-1}_{(r+1)/2} = p^l_r \quad &+ \quad \frac{\Delta r^l}{4}\left(\frac{2}{3}\rho^l_r + \frac{4}{3}\rho^{l-1}_{(r+1)/2}\right)\left(g^{l-1}_{(r+1)/2-\frac{1}{2}} + g^{l-1}_{(r+1)/2}\right) \\ &+ \quad \frac{\Delta r^l}{8}\left(\frac{5}{3}\rho^l_r + \frac{1}{3}\rho^{l-1}_{(r+1)/2}\right)\left(g^l_r + g^{l-1}_{(r+1)/2-\frac{1}{2}}\right) \quad . \end{aligned} \tag{7.13}$$

and for spatially-constant gravity, it simplifies to:

$$p^{l-1}_{(r+1)/2} = p^l_r + \frac{3\Delta r^l g}{4}\left(\rho^l_r + \rho^{l-1}_{(r+1)/2}\right). \tag{7.14}$$

CHAPTER 8

Visualization

Plotfiles

MAESTRO outputs plotfile specifically for visualization and analysis. Table 8.1 lists the quantities stored in a plotfile. Not all of these may be present, dependent on what options were used in creating the plotfile.

By default, plotfiles store double precision data, but if `single_prec_plotfiles = T` is set, then the data is converted to single precision before outputting—this is done to reduce file sizes.

Table 8.1: Plotfile quantities

plotfile variable name	description	runtime parameter controlling output
`x_vel`	$\tilde{u}$	–
`y_vel`	$\tilde{v}$	–
`z_vel`	$\tilde{w}$	3-d runs only
`density`	ρ	–
`rhoh`	(ρh)	`use_tfromp = F` or (`use_tfromp = T` and `plot_h_with_use_tfromp = T`)
`h`	$(\rho h)/\rho$	`use_tfromp = F` or (`use_tfromp = T` and `plot_h_with_use_tfromp = T`)
`X(*)`	$(\rho X_k)/\rho$	`plot_spec`
`tracer`	tracers	`plot_trac`
`w0_x`	$w_0\mathbf{e}_r \cdot \mathbf{e}_x$	`plot_base`
`w0_y`	$w_0\mathbf{e}_r \cdot \mathbf{e}_y$	`plot_base`
`w0_z`	$w_0\mathbf{e}_r \cdot \mathbf{e}_z$	`plot_base`
`divw0`	$\nabla \cdot w_0$	`plot_base`

continued on next page

Table 8.1—continued

plotfile variable name	description	runtime parameter controlling output
rho0	ρ_0	plot_base
rhoh0	$(\rho h)_0$	plot_base
h0	$(\rho h)_0/\rho_0$	plot_base
p0	p_0	plot_base
radial_velocity	$\widetilde{\mathbf{U}}\cdot\mathbf{e}_r + w_0$	spherical == 1
circum_velocity	$\lvert\widetilde{\mathbf{U}} - (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\mathbf{e}_r\rvert$	spherical == 1
magvel	$\lvert\widetilde{\mathbf{U}} + w_0\mathbf{e}_r\rvert$	–
momentum	$\rho\lvert\widetilde{\mathbf{U}} + w_0\mathbf{e}_r\rvert$	–
vort	$\lvert\nabla\times\widetilde{\mathbf{U}}\rvert$	–
S	S	–
rhopert	$\rho - \rho_0$	–
rhohpert	$(\rho h) - (\rho h)_0$	use_tfromp = F or (use_tfromp = T and plot_h_with_use_tfromp = T)
tfromp	$T(\rho, p_0, X_k)$	–
tfromh	$T(\rho, h, X_k)$	use_tfromp = F or (use_tfromp = T and plot_h_with_use_tfromp = T)
deltaT	$[T(\rho,h,X_k) - T(\rho,p_0,X_k)]/T(\rho,h,X_k)$	use_tfromp = F or (use_tfromp = T and plot_h_with_use_tfromp = T)
deltap	$\lvert p(\rho,h,X_k) - p_0\rvert/p_0$	use_tfromp = F or (use_tfromp = T and plot_h_with_use_tfromp = T)
tpert	$T(\rho,h,X_k) - \overline{T}$ if use_tfromp = F; $T(\rho,p_0,X_k) - \overline{T}$ otherwise	–
Machnumber	$\lvert\widetilde{\mathbf{U}} + w_0\mathbf{e}_r\rvert/c(\rho,p_0,X_k)$	–
soundspeed	$c(\rho,p_0,X_k)$	plot_cs
deltagamma	$\Gamma_1(\rho,p_0,X_k) - \overline{\Gamma}_1$	–
entropy	$s(\rho,p_0,X_k)$	–
entropypert	$[s(\rho,p_0,X_k) - \overline{s}]/\overline{s}$	–
sponge or sponge_fdamp	$1/(1+\Delta t\kappa f_{\rm damp})$ by default; $f_{\rm damp}$ if plot_sponge_fdamp = T	–
pi	π	–
gpi_x	$\nabla\pi\cdot\mathbf{e}_x$	plot_gpi
gpi_y	$\nabla\pi\cdot\mathbf{e}_y$	plot_gpi
gpi_z	$\nabla\pi\cdot\mathbf{e}_z$	plot_gpi
pioverp0	π/p_0	plot_base
p0pluspi	$p_0 + \pi$	plot_base
omegadot(*)	$\dot{\omega}_k$	plot_omegadot
enucdot	$(\rho H_{\rm nuc})/\rho$	plot_Hnuc
Hext	$(\rho H_{\rm ext})/\rho$	plot_Hext
eta_rho	η_ρ	plot_eta
thermal	$\nabla\cdot\kappa\nabla T$	use_thermal_diffusion
conductivity	$\kappa(\rho,T,X_k)$	use_thermal_diffusion
ad_excess	$\nabla - \nabla_{\rm ad}$	plot_ad_excess
particle_count	number of particles in a cell	use_particles

continued on next page

Table 8.1—continued

plotfile variable name	description	runtime parameter controlling output
processor_number	processor number containing the cell's data	plot_processors
pi_divu	$\pi\nabla \cdot \tilde{\mathbf{U}}$ (a measure of energy conservation)	plot_pidivu

Mini vs. regular plotfiles

MAESTRO can manage two independent sets of plotfiles. This allows you to output the default plotfiles, which contain a lot of variables, sparsely, and output mini-plotfiles much more frequently. A mini-plotfile is controlled by the analogous runtime parameters as the main plotfiles:

- mini_plot_int is the interval in steps between successive plotfiles
- mini_plot_deltat is the interval in time between successive plotfiles
- mini_plot_base_name is the base name that prefixes the plotfiles. The default is miniplt

To set the fields that are stored in the mini plotfile, a second set of runtime parameters is used: mini_plot_var1, mini_plot_var2, . . ., mini_plot_var9. These can be set to any of the following:

- "density"
- "species": this gets all of the mass fractions
- the name of an individual species in the network (like "helium-4")
- "radvel": this gets both the radial and circumferential velocity
- "velocity": all three componets
- "temperature": this is either $T(\rho, p_0)$ or $T(\rho, h)$, depending on the value of use_tfromp
- "enuc": the nuclear energy generation rate
- "mach": the Mach number

the AMReX file format

MAESTRO stores the plotfile data in a hierarchical directory format, with each level's data stored in a separate subdirectory. Some meta-data is stored in the top-level to help interpret the structure. The basic format is:

- plt00000/
  - Header
  - job_info
  - Level_00/
    - Cell_D_00000

```
        * ...
        * Cell_H
    - model_cc_00000
    - model_ec_00000
```

In the main directory, `Header` contains the information required to interpret the data stored on disk. We describe this below. The `job_info` file is a plaintext file that contains a lot of information about the run (where it was run, when it was run, compiler options, runtime options, etc.). It is not needed to interpret the data. Finally, the `model_cc_00000` and `model_ec_00000` contain the MAESTRO basestate information for the cell-centered and edge-centered basestate quantities respectively. These files are not typically used for visualization.

`Header`

The main `Header` is written by `fabio_ml_multifab_write_d` by processor 0. The information contained is the following:

NavierStokes-V1.1
number of variables
variable 1 name
variable 2 name
⋮
last variable name
number of dimensions
simulation time
number of levels (0-based)
physical domain minimum coordinate (dm numbers)
physical domain maximum coordinate (dm numbers)
jump in refinement between level 0 and 1
⋮
jump in refinement between level $n-2$ *and* $n-1$

Visualizing with Amrvis

Amrvis is a tool developed together with AMReX to visualize datasets from codes built around the AMReX library. You can download the Amrvis source from:
`https://ccse.lbl.gov/Downloads/downloadAmrvis.html`

Amrvis exists in the C++ AMReX framework, so the build system is slightly different. A different executable is needed for 2- vs. 3-d datasets. Edit the `GNUmakefile` and set the compilers (probably `g++` and `gfortran`) and the dimensionality, and turn off any of the volume rendering options. You will need to have the Motif library installed on your system (or a replacement, such as `lesstif`.

Once the code is built, you visualize a dataset as:

```
amrvis3d.Linux.g++.gfortran.ex pltfile
```

where `pltfile` is the name of the plotfile directory. Different variables can be selected from the drop down menu at the top. Middle and right clicking in 3-d select the slice planes, and shift + middle or right will extract 1-d lines through the data. In 2-d, middle and right clicking alone extract 1-d lines.

If Amrvis cannot find the `Palette` file, then the plots will be in grayscale. To fix this, copy the `amrvis.defaults` and `Palette` files to your home directory and edit `amrvis.defaults` so that the `palette` line points to the `Palette` file, e.g.:

```
palette                /home/username/Palette
```

Visualizing with VisIt

Python visualization scripts

`AmrPostprocessing/python` provides some simple commandline tools for doing visualizations of AMReX plotfiles (note: a subset of these are distributed directly with AMReX in `amrex/Tools/Py_util/`). The main drivers are written in python and use a set of Fortran routines, compiled with `f2py` to interface with the plotfile data. To use the routine, you will need to have `matplotlib` and `f2py` installed. On a machine running Fedora linux, you can install these packages via

```
yum install python-matplotlib f2py
```

The library required by the python routines can be built by typing '`make`' in that directory. If successful, you should find a library `fsnapshot.so`.

The path to `fsnapshot.so` should be included in your `PYTHONPATH` environment variable. This can be done by adding:

```
export PYTHONPATH="${PYTHONPATH}:/home/user/AmrPostprocessing/python}
```

to your `.bashrc`.

It is recommended that you use `matplotlib` version 1.2.0 or higher. If the fonts look strange in the output files, you can try installing the `lyx-fonts` package and deleting your `.matplotlib` directory, and trying again.

`plotsinglevar.py`

`plotsinglevar.py` does visualizations of 2-d AMReX plotfiles, and slices through 3-d AMReX plotfiles. A simple plot can be made via:

```
plotsinglevar.py --log -o test.png plt00000/ tfromp
```

This will make a plot of "`tfromp`" from the plotfile `plt00000` with log scaling, and store the output in `test.png`. See Figure 8.1. If you don't do '`-o`', then a default output filename consisting of the plotfile name + component will be used.

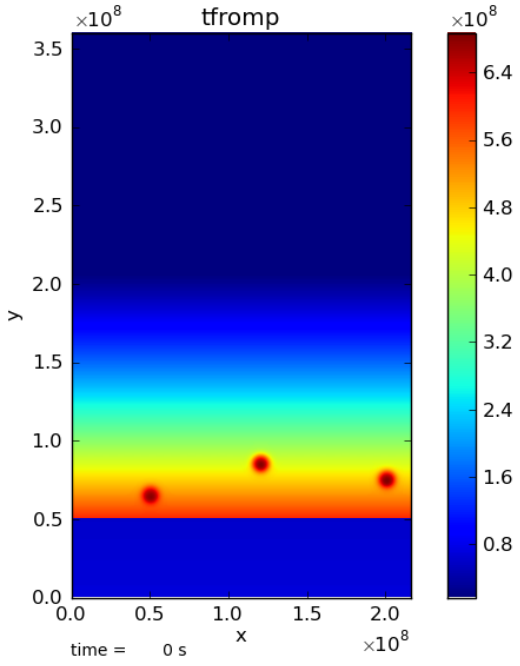


Figure 8.1: Plot of `reacting_bubble` done with the python script `plotsinglevar.py`.

If you list 2 different variables after the plotfile name, then they will be plotted side-by-side in a single figure. For example,

```
plotsinglevar.py plt00000/ tfromp enucdot
```

produces the output shown in figure 8.2.

Additional options include '`-m`' to specify the minimum data value, '`-M`' to specify the maximum data value, and '`--eps`' to make an EPS plot instead of PNG. Running the script with no parameters will give the full list available options.

Limited 3-d support is available. When run as with a plotfile name and variable, it will plot slices (x-y, x-z, and y-z) through the center of the domain. The option '`--origin`' will put the slices through the origin.

`contourcompare.py`

`contourcompare.py` takes two or three plotfiles and a single variable as arguments and plots contours of the datasets on the same set of axes. This is form comparisons of different runs. Running the script with no parameters will give the full list available options.

For example:

```
contourcompare.py tfromp plt00000 other_plt00000
```

will make a contour plot of the variable `tfromp` from the data in `plt00000` and `other_plt00000` shown on the same axes.

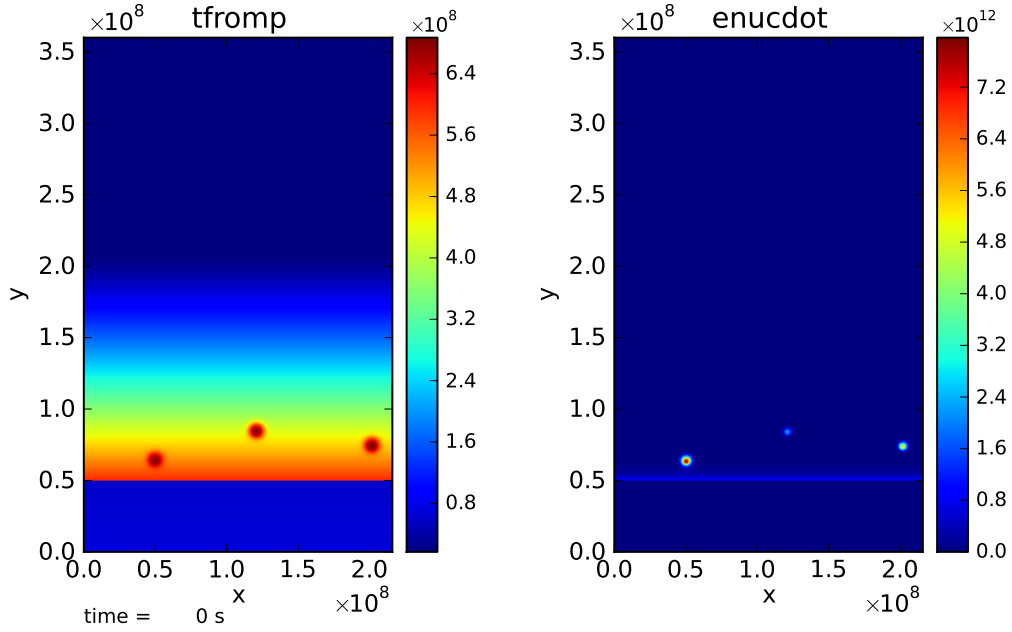


Figure 8.2: Plot of `reacting_bubble` done with the python script `plotsinglevar.py` showing 2 variables plotted from a single plotfile.

`runtimevis.py`

The `runtimevis.py` script is designed to be run from a submission script to produce plots from plotfiles as they are produced. This is accomplished by hooking it into the `process` scripts described in Chapter 11.

The script itself reads in an inputs file, `vis.in`, that describes the variables to plot. From 1 to 6 variables can be plotting from a plotfile. The script does its best to organize them in columns and rows to maximize the plot area. The image is always output at 1280×720 pixels, corresponding to 720p HD resolution. For each variable, a block of the form:

```
[varname]
min = 1
max = 2
log = 1
```

is supplied. If `min` or `max` are omitted, then the data limits are computed automatically. If `log` is omitted, then no log is taken of the data before plotting. The script is then run as:

```
runtimevis.py plt00000
```

Visualizing with yt

ytis a Python package for analyzing and visualizing simulation data, and understand that AMReX data from MAESTRO and CASTRO (along with many other simulation codes). For more information, see

the ythomepage at `http://yt-project.org/` and [33].

Note: for MAESTRO **data, you need to use** yt**3.0 or later.**

Some sample scripts that use yt with MAESTRO data are contained in `MAESTRO/Util/yt/`.

Installing yt

The easiest way to obtain ytis through the use of an installation script:

```
> wget http://hg.yt-project.org/yt/raw/yt-3.0/doc/install_script.sh
> bash install_script.sh
```

By default, this ytinstall script will download and install, in its own isolated environment, all the secondary utilities needed to get ytup and running. Note that this currently includes installing *hdf5, zlib, bzip2, libpng, freetype, python, numpy, matplotlib, mercurial, ipython, h5py, Cython, Forthon* as well as a yt*mercurial* bundle of changes. You can turn off the automatic installation of any of these particular packages by setting the appropriate `INST_*` variable to zero in the install script; you may have to then change some of `*_DIR` variables to point to your own particular installation of that package. It is usually best to just let ytinstall its own stuff, which ensures things are working properly.

After the install script has finished, and assuming you let ytinstall its own packages, you'll need to *prepend* some environment variables with ytlocations so that your system finds those first and stops looking. ytprovides a simple script to do this, which will be announced upon successful completion.

Working with yt

The ytinstallation provides both an interactive, *iPython*-like, interface or the ability to import ytmodules for use in a batch script. The interactive interface should be in your `$PATH` if you've followed the instructions in the previous section; to start it, simply type `iyt` in a terminal.

```
> iyt

Welcome to yt!


In [1]:
```

Codes like `Enzo` use what are called *parameter files* to describe the general information—number of levels, domain dimensions, time, etc.—of a a dataset. ytlikes to grab this information before working on any specific data; this is accomplished via the convenience method `load`:

```
In [1]: pf = load("plt00166")
```

Note that some older versions of ytneeded an `inputs` file in the same directory as the plotfile, but as of yt3.0, all the necessary metadata is obtained from the `job_info` file inside the plotfile directory. The `load` method returns an instance of the `StaticOutput` class. One of the easiest ways of handling plots is via a `PlotCollection` object

```
In [2]: center = (pf.domain_right_edge + pf.domain_left_edge) / 2.0
In [3]: pc = PlotCollection(pf,center)
```

By default, the `PlotCollection` constructor places the center of the plot to be at the location of peak density. Here we have calculated the center of the domain by accessing the lower and upper domain boundaries via the `numpy` arrays `pf.domain_left_edge` and `pf.domain_right_edge`, respectively. Note that up until this point, ythas not actually loaded the AMR dataset.

Now we would like to take some slices of tfromp in the dataset and generate some 2-d plots. To do this, we will use the `PlotCollection`'s `add_slice` method:

```
In [4]: pc.add_slice("tfromp",0)
In [5]: pc.add_slice("tfromp",1)
In [6]: pc.add_slice("tfromp",2)
```

The first call to `add_slice` builds an `AMRHierarchy` object associated with `pf`. The `AMRHierarchy` object contains information about the actual dataset, such as its layout in the simulation domain or on disk. Building the hierarchy is expensive, but once it is built the data it contains can be accessed via attributes and dictionary lookup. In other words, the subsequent `add_slice` operations are relatively cheap. The first parameter to the `add_slice` method is obviously the variable we want; the second (optional) parameter specifies an coordinate axis orthogonal to the slice being made—0 for x, 1 for y, 2 for z. Now we want to save the plots from the `PlotCollection`; this is done with the `save` method, which takes an optional basename for the generated files:

```
In [7]: pc.save("my_cool_images")
```

This generates the following image files:

```
Out[7]:
['my_cool_images_Slice_x_tfromp.png',
 'my_cool_images_Slice_y_tfromp.png',
 'my_cool_images_Slice_z_tfromp.png']
```

Figure 8.5.2 shows an example of one of the slice images. Note that this was a quick and dirty generation of the image—there is a lot of space around the figure, which can be removed with various options to the ytmethods. Also, the user can specify derived variables, log-scaling, annotations, etc. For more information see the documentation at `http://yt-project.org/docs/dev/` .

When writing a script to use in batch mode, one has to manually import the import modules needed to work with yt. As such, all scripts must import from the `yt.mods` module, which is essentially a convenience module that sets up the appropriate ytnamespace. For completeness, below is a script containing our example above.

```
# load our namespace
from yt.mods import *

# the plotfile I'm interested in
fn = "plt00166"

# load it into a StaticOutput object
pf = load(fn)

# find the center of the domain
center = (pf.domain_right_edge + pf.domain_left_edge) / 2.0

# associate a PlotCollection with the pf
```

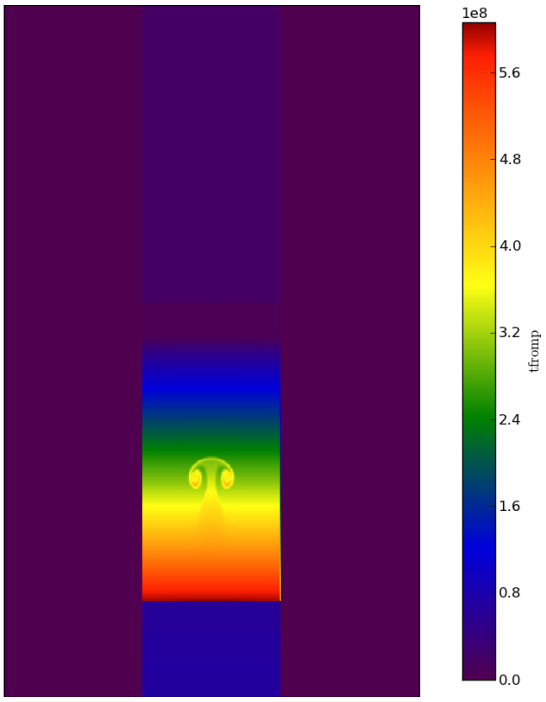


Figure 8.3: Example slice through 3-d dataset with yt.

```
pc = PlotCollection(pf,center)

# add some slices of tfromp
pc.add_slice("tfromp",0)
pc.add_slice("tfromp",1)
pc.add_slice("tfromp",2)

# save our plots to a files with basename
# "my_cool_images"
pc.save("my_cool_images")
```

2-d datasets

To visualize 2-d data, you can use the `SlicePlot` function, picking the normal direction to be `"z"`:

```
from yt.mods import *
 pf = load("plt00085")
s = SlicePlot(pf, "z", "tfromp")
s.save("tfromp.eps")
```

This generates the figure shown in Figure 8.4.

Volume Rendering

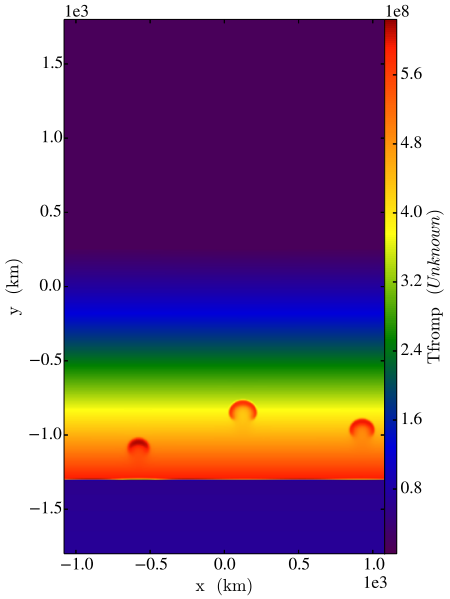


Figure 8.4: Example slice through 3-d dataset with yt.

CHAPTER 9

Analysis Routines

The `Postprocessing` Routines

The `BoxLib/Tools/Postprocessing/F_Src/` directory contains a large number of Fortran-based analysis routines for BoxLib datasets. Many of these can be used with both MAESTRO and the compressible astrophysics code, CASTRO.

To compile any of the individual routines, edit the `GNUmakefile` add uncomment the line beginning with '`programs +=`' containing the routine you want to build.

General Analysis Routines

The following routines are generally applicable for any BoxLib-based plotfile. Typing the executable names without any arguments will provide usage examples.

- `faverage.f90`

  Laterally average each of the variables in a plotfile (works for both 2-d and 3-d). This is written with MAESTRO plane-parallel geometry plotfiles in mind, and the averaging is done over the coordinate direction(s) perpendicular to gravity.

- `fboxinfo.f90`

  Print out some basic information about the number of boxes on each refinement level and (optionally) the bounds of each of the boxes.

- `fcompare.f90`

  Compare two plotfiles, zone-by-zone to machine precision, and report the L2-norm of the error (both absolute and relative) for each variable. This assumes that the grids are identical.

With the optional `--zone_info var` argument, where `var` is the name of a variable, it will also report the full state for the zone where `var` has the largest error.

This is used by in the regression test suite in `Parallel/util/regtests/`.

- `fextract.f90`

  Extract a 1-d line through a dataset (1-, 2-, or 3-d). This works with both uniformly-gridded or AMR datasets. For multi-dimensional datasets, the coordinate direction to extract along can be specified. The line is always taken through the center of the domain. Either a single variable or all variables, along with the coordinate information, are output to a file.

- `fextrema.f90`

  Report the min and max of each variable (or only a single variable) in one or more plotfiles.

- `fsnapshot2d.f90`, `fsnapshot3d.f90`

  Create an image (PPM file) of a single variable in a plotfile. For 3-d, the slice plane through the center of the domain is specified. Separate routines exist for 2-d and 3-d datasets.

- `ftime.f90`

  For each plotfile, simply print the simulation time.

- `fvarnames.f90`

  Simply print out the list of variables stored in a plotfile.

Data Processing Example

The routine `fspeciesmass2d.f90` in `F_src/tutorial` serves as a well-commented example of how to work with MAESTRO plotfile data. This routine simply computes the total mass of a particular species on the entire domain for a 2-d dataset. It is written to understand a multilevel (AMR) dataset, and only considers the finest-available data at any physical location in the computational domain.

`fspeciesmass2d.f90` should provide a good starting point for writing a new analysis routine for BoxLib data.

Particle routines

The `parseparticles.py` routine in the `python/` subdirectory can read in MAESTRO particle files containing particle histories (usually named `timestamp_??`). See the discussion in § 5.13 for details on initializing particles in MAESTRO. The driver `test_parseparticles.py` shows shows how to use this module to plot particle histories. Additional documentation is available from the module itself. In the python environment, type:

```
import parseparticles
help(parseparticles)
```

to get information on the classes and functions provided by the `parseparticles` module.

As a concrete example, running `reacting_bubble` with particles enabled will seed particles in the initial hotspot. To plot the results, first set your `PYTHONPATH` environment variable to point to the `AmrPostprocessing/python/` directory, for example:

```
export PYTHONPATH="/home/username/development/AmrPostprocessing/python"
```

This will allow python to see the `parseparticles.py` routine. For the `reacting_bubble` problem, the `plotparticles.py` routine shows how to plot the particle histories and make an animation of the particles colored by the ash mass fraction. This script is run as:

```
./plotparticles.py timestamp_*
```

Note, these python routines require the NumPy and matplotlib packages. On a Fedora Linux system, the necessary routines can be installed via:

```
yum install python-matplotlib lyx-fonts stix-fonts
```

CHAPTER 10

Build System

Build Process Overview

The MAESTRO build system uses features of GNU make (version 3.81 or later), which is typically the default on systems. The MAESTRO executable is built in the problem's directory (one of the directories under `SCIENCE/`, `TEST_PROBLEMS`, or `UNIT_TESTS`). This directory will contain a makefile, `GNUmakefile`, that includes all the necessary information to build the executable.

The main macros that define the build process are split across several files. The 4 main files are:

- `${AMREX_HOME}/Tools/F_mk/GMakedefs.mak`:

  This setups the basic macros, includes the options for the selected compiler, builds the list of object and source files, and defines the compile and link command lines.

- `${AMREX_HOME}/Tools/F_mk/GMakeMPI.mak`:

  This implements any changes to the compiler names and library locations necessary to build a parallel executable with MPI.

- `${AMREX_HOME}/Tools/F_mk/GMakerules.mak`:

  This creates the various build targets and specifies the rules for building the object files, the list of dependencies, and some other lesser-used targets (tags for editors, documentation, etc.)

- `MAESTRO/GMaestro.mak`:

  This is a MAESTRO-specific file that gathers all of the various modules that are used to build a typical MAESTRO application and integrates with the AMReX build system. Every MAESTRO problem's `GNUmakefile` will include this file.

MAESTRO gets the location of the AMReX library through the AMREX_HOME variable. This should be set as an environment variable in your shell start-up files (e.g. .bashrc or .cshrc).

The AMReX build system separates the compiler-specific information from the machine-specific information—this allows for reuse of the compiler information. The only machine-specific parts of the build system are for the MPI library locations, contained in GMakeMPI.mak. The compiler flags for the various compilers are listed in the files in ${AMREX_HOME}/Tools/F_mk/comps/. The compiler is set via the COMP variable in the problem's GNUmakefile.

There are several options in addition to the compiler that affect the build process: MPI, OMP, and NDEBUG—these turn on/off MPI parallelization, OpenMP parallelization, and debugging. Together, these choices along with the compiler name are reflected in the name of the executable.

When the 'make' command is run, the object and module files are written into a directory t/*OS*.*COMP*.*other*/, where *OS* is the operating system detected by the build system, *COMP* is the compiler used, and *other* reflects any other build options (MPI, OpenMP, debugging, etc.) used. Separating each build into a separate subdirectory under the problem directory allows for multiple builds of MAESTRO to exist side-by-side in the problem directory.

Finding Source Files

The MAESTRO executable is built from source distributed across a number of directories. In each of these directories containing source files, there is a GPackage.mak file. This file has a number of lines of the form:

```
f90sources += file.f90
```

where file.f90 is a source file that should be built when this directory is added to the list of build directories. For old fixed-form Fortran files, the files should be added to the fsources variable instead of f90sources.

The AMReX build system relies on the vpath functionality of make. In a makefile, the vpath variable holds search path used to locate source files. When a particular file is needed, the directories listed in vpath are searched until the file is found. The first instance of a file is used. We exploit this feature by always putting the build directory first in vpath (this is done in GMakerules.mak). This means that if a source file is placed in the build directory, that copy will override any other version in the source path.

In MAESTRO, the vpath variable is set using the macros defined in GMaestro.mak. A user does not need to set this variable explicitly. Additional source locations are added in the manner described below (see § 10.2.1).

Dependencies

There is no need to explicitly define the dependencies between the source files for Fortran modules. The scripts in AMREX_HOME/Tools/F_scripts/ are run at the start of the build process and parse all the source files and make an explicit list of the dependency pairs. The execution of these scripts is triggered by including makefiles of the form *.depends. On a fresh build these will not exist. When GNU make cannot find an included makefile it will first attempt to build it using any relevant targets before issuing an error. The targets for the *.depends files contain the recipe for executing the dependency scripts.

Once these makefiles are built by the scripts GNU make will then read the dependencies for the current build from them. This process is defined in `GMakerules.mak`.

A few files use explicit '`include`' statements to include Fortran source in other source files. Any include file locations should be added to `Fmincludes` variable in the problem's `GNUmakefile`. This does not occur frequently. For the case of the `helmholtz` equation of state, this is done automatically in `GMaestro.mak`.

Files Created at Compile-time

Several files are created at build-time:

- `probin.f90`:

  This is the module that controls runtime parameters. This is created by the script `write_probin.py` in `${AMREX_HOME}/Tools/F_scripts/`. The makefile logic for creating it is in `GMaestro.mak`. At compile time, the problem, main `MAESTRO/`, and any microphysics directories (set from the `EOS_DIR`, `CONDUCTIVITY_DIR`, and `NETWORK_DIR` parameters in the `GNUmakefile` are searched for `_parameter` files. These files are parsed and the `probin.f90` file is output containing the runtime parameters, their default values, and the logic for reading and interpreting the inputs file.

- `build_info.f90`:

  This is a module that contains basic information about the build environment (compiler used, build machine, build directory, compiler flags, etc.). This is created by the script `makebuildinfo.py` in `${AMREX_HOME}/Tools/F_scripts/` from `GMaestro.mak` by passing in a number of makefile variables. This is rewritten everytime the code is compiled. The primary use of this module is writing out the `job_info` file in the plotfiles.

- (`network.f90`):

  This is generated at compile time *only* for the `general_null` network. The `general_null` network allows the user to define a list of non-reacting species builds the `network.f90` based on this list. The makefile logic for building the `network.f90` is in the `GPackage.mak` in `Microphysics/networks/general_n` The script `write_network.py` in that directory does the actual parsing of the species file and outputs the `network.f90`.

MAESTRO **Problem Options**

Problem-specific Files

If a problem has a unique file that is needed as part of the build, then that file should be added to a `GPackage.mak` file in the problem's directory. Since, by default, problems don't have a `GPackage.mak`, the build system needs to be told to look in the problem directory for unique sources. This is accomplished by adding the problem's directory to the `EXTRA_DIR` variable in the problem's `GNUmakefile`.

Note that this is not necessary if you place a custom version of a source file in the problem's directory. Since that file is already listed in the `GPackage.mak` in its original location, the build system will know

that it needs to be built. Since the vpath variable puts the problem's directory at the start of the search path, the version of the file in the problem's directory will be found first.

Defining EOS, Network, and Conductivity Routines

Each MAESTRO problem needs to define an equation of state, a reaction network, and a routine to compute the conductivities (for thermal diffusion). This is true even if the problem is not doing reactions of thermal diffusion. These definitions are specified in the problem's GNUmakefile.

- EOS_DIR:

  This variable points to the directory (by default, relative to Microphysics/EOS/) of the equation of state used for the build. Choices that work with MAESTRO are:

  - helmholtz
  - gamma_law_general
  - multigamma

  To use an EOS contained in a different location, set the variable EOS_TOP_DIR to point to the directory above the alternate EOS directory.

- CONDUCTIVITY_DIR:

  This variable points to the conductivity routine used for the build (by default, relative to Microphysics/conductivity/). Choices that work with MAESTRO are:

  - constant
  - timmes_stellar

  If diffusion is not being used for the problem, this should be set to constant. To use an alternate conductivity routine, set the variable CONDUCTIVITY_TOP_DIR to point to the directory above the alternate conductivity directory.

- NETWORK_DIR:

  This variable points to the reaction network used for the build (by default, relative to Microphysics/networks/. Several options are present in Microphysics/networks/. A network is required even if you are not doing reactions, since the network defines the species that are advected and interpreted by the equation of state.

  A special choice, Microphysics/networks/general_null is a general network that simply defines the properties of one or more species. This requires an inputs file, specified by GENERAL_NET_INPUTS. This inputs file is read at compile-time and used to build the network.f90 file that is compiled into the source.

  To use an alternate reaction network, set the variable NETWORK_TOP_DIR to point to the directory above the alternate network.

Core MAESTRO modules

Several modules are included in all MAESTRO builds by default. From AMReX, we alway include:

- ${AMREX_HOME}/Src/F_BaseLib
- ${AMREX_HOME}/Src/LinearSolvers/F_MG

From Util, we always include

- Util/model_parser
- Util/random

The microphysics, as described above is also included. For the networks, we include a file called NETWORK_REQUIRES into GMaestro.mak that tells us whether to also include Util/VODE (if NEED_VODE := t). It is assumed in this case that we need BLAS and LINPACK, so these are compiled in from Util/BLAS and Util/LINPACK.

You can instead link in a system-wide optimized BLAS library by setting SYSTEM_BLAS := t in the GNUmakefile. This adds -lblas to the link line, and assumes that the library is in your path. Note that for some systems, you should have the static BLAS libraries available.

From MAESTRO/, we add

- MAESTRO/constants
- MAESTRO/Source

(although see the unit tests section below regarding MAESTRO/Source.

Finally, any extra directories listed in the EXTRA_DIR variable are included.

For each of these included directories, GMaestro.mak adds the list of source files defined in their GPackage.mak to the list of files to be compiled. It also adds each of these directories to the vpath as a directory for the build process to search in for source files.

Unit Tests

Sometimes we only want to use a few of the standard MAESTRO routines, for example in a unit test where we are testing only a small part of the MAESTRO algorithm indepenedently. In this case, we don't want to comple all of the files in MAESTRO/Source. If we set UNIT_TEST := t in our problem's GNUmakefile, then the GPackage.mak in MAESTRO/Source is not read, so those files are not automatically put into the list of files to compile. Instead, the problem should create its own GPackage.mak listing only the subset of files that are to be compiled. MAESTRO/Source is put into the vpath search path for sources, so those files will still be found as needed.

AMReX-only Tests

An even more restrictive setting than UNIT_TEST := t is invoked by setting AMREX_ONLY := t. This is like the unit test flag, but does not include MAESTRO/Source in the vpath search path for sources. So this is intended for cases where we don't want to use any MAESTRO source files. Typically, this is used in the small unit tests that live under the various microphysics solvers. If a probin.f90 is built for these tests, it will not include all the MAESTRO-specific parameters, but will include any parameters from the various microphysics routines.

Special Targets

Debugging

(print-*)

To see the contents of any variable in the build system, you can build the special target `print-`*`varname`*, where *varname* is the name of the variable. For example, to see what the Fortran compiler flags are, you would do:

```
make print-FFLAGS
```

This would give (for `gfortran`, for example):

```
FFLAGS is -Jt/Linux.gfortran/m -I t/Linux.gfortran/m -O2 -fno-range-check
```

This functionality is useful for debugging the makefiles.

file_locations

Source files are found by searching through the `make vpath`. The first instance of the file found in the `vpath` is used in the build. To see which files are used and their locations, do:

```
make file_locations
```

This will also show any files that aren't found. Some are expected (e.g., `build_info.f90` and `probin.f90` are created at compile time), but other files that are not found could indicate an incomplete `vpath`.

clean and realclean

Typing '`make clean`' deleted the object and module files for the current build (i.e., the current choice of `MPI`, `NDEBUG`, `COMP`, and `OMP`). This also removes any of the compile-time generated source files. Any other builds are left unchanged.

Typing '`make realclean`' deletes the object and module files for all builds—i.e., the entire `t/` directory is removed.

Special Debugging Modes

AMReX has several options that produce executables that can help track down memory issues, uninitialized variables, NaNs, etc.

- `NDEBUG`

  To generate an executable with debugging information included in the executable (e.g., to be interpreted by the debugger, `gdb`), compile with `NDEBUG := .` This will usually add `-g` to the compile line and also lower the optimization. For `gfortran` it will add several options to catch uninitialize variables, bounds errors, etc.

- TEST

  Setting TEST := t will enable routines in AMReX initialize multifabs and arrays allowed via bl_allocate to signalliing NaNs. This behavior is the same as NDEBUG :=, but TEST := t uses the same compiler optimizations as a normal build.

  This can be useful with compiler flags that trap floating point exceptions (FPEs), but checks on floating point exceptions can also be enabled through runtime parameters passed to AMReX's backtrace functionlity:

  - boxlib_fpe_invalid: enabling FPE trapping for invalid operations (e.g. 0 * inf, sqrt(-1))
  - boxlib_fpe_zero: enable FPE trapping for divide-by-zero
  - boxlib_fpe_overflow: enable FPE trapping for overflow

- backtracing

  When exception trapping is enabled (either via AMReX or the compiler), the code will abort, and the backtrace information will be output to a file Backtrace.N, where N is the processor number. AMReX will also initialize multifabs with signaliing NaNs to help uncover any floating point issues.

  This is also useful to diagnose deadlocks in parallel regions. If the code is hanging, doing "control-C" will be intercepted and the code will generate a backtrace which will identify where in the code there was a deadlock.

  Behind the scenes, AMReX implements this capability via the Linux/Unix feenableexcept function (this is in backtrace_c.cpp in AMReX).

- FSANITIZER

  For gfortran, gcc, g++, setting FSANITIZER := t will enable the address sanitizer support built into GCC. This is enabled through integration with https://github.com/google/sanitizers in GCC.

  Note: you will need to have the libraries libasan and libubsan installed on your machine to use this functionality.

Extending the Build System

Adding a Compiler

Properties for different compilers are already defined in ${AMREX_HOME}/Tools/F_mk/comps/. Each compiler is given its own file. The appropriate file is included into GMakedefs.mak by looking at the COMP variable and the operating system. These compiler files define the compiler flags for both optimized and debug compiling. Additionally, the variable FCOMP_VERSION should be defined there, based on the output from the compiler, to provide the compiler version for output into the job_info file at runtime.

Parallel (MPI) Builds

When building with MPI, the build system needs to know about the location of the MPI libraries. If your local MPI has the `mpif90` and `mpicc` wrappers installed and working, then MAESTRO will attempt to use these. Otherwise, you will need to edit `GMakeMPI.mak` and add a section specific to your machine with the compiler and library location. It is best to simply copy an existing similar portion of the makefile and adjust it to your system. Most national supercomputing facilities are already supported, and parallel builds on them should work out of the box.

CHAPTER 11

Compilers and Managing Jobs

General Info

All plotfile directories have a `job_info` file which lists as host of parameters about the simulation, including:

- A descriptive name for the simulation (the `job_name` runtime parameter
- The number of MPI tasks and OpenMP threads
- The total amount of CPU-hours used up to this point
- The data and time of the plotfile creation and the directory it was written to.
- The build date, machine, and directory
- The MAESTRO, AMReX, and other relevant git hashes for the source
- The directories used in the build
- The compilers used and compilation flags
- The number of levels and boxes for the grid
- The properties of the species carried
- The tolerances for the MG solves
- Any warnings from the initialization procedure (note: these are not currently stored on restart).
- The value of all runtime parameters (even those that were not explicitly set in the inputs file), along with an indicator showing if the default value was overridden.

This file makes it easy to understand how to recreate the run that produced the plotfile.

Linux boxes

gfortran

gfortran is probably the best-supported compiler for MAESTRO. Here are some version-specific notes:

- *gfortran 4.8.x*: This typically works well, but sometimes we get an error allocating memory in `cluster_f.f90`. This is a compiler bug (affecting atleast 4.8.2 and 4.8.3):

  The code runs without any problem if it is compiled with `-O2 -ftree-vectorize -fno-range-check` (our default) but with `cluster_f.f90` compiled with `-O2 -ftree-vectorize -fno-range-check -fno-tree-pre`. The "`fno-tree-pre`" option turns off "`ftree-pre`" that is turned on by "`O2`"

  GCC manual says,

  > `-ftree-pre`
  > Perform partial redundancy elimination (PRE) on trees. This flag is enabled by default at `-O2` and `-O3`.

  gfortran 4.8.5 appears to work without issues

- *gfortran 5.1.1*: These compilers have no known issues.

- *gfortran 5.3.x*: These compilers have no known issues.

- *gfortran 6.2*: These compilers work without any known issues.

  gfortran 6.2.1 is used for nightly regression testing.

PGI compilers

The AMReX floating point exception trapping is disabled with PGI compilers earlier than version 16, due to problems with PGI using the system header files. From version 16 onward, things should work.

There are no known issues with PGI 16.5 compilers—these are used for nightly regression testing.

Working at OLCF (ORNL)

Titan Compilers

The preferred compilers on Titan are the Cray compilers. Cray 8.4.0 works well on titan/OLCF with MPI/OpenMP.

Monitoring Allocations

The `showusage` and `showusage -f` commands give an overview of the usage.

Automatic Restarting and Archiving of Data

The submission script `titan.run` and shell script `process.titan` in `Util/job_scripts/titan/` are designed to allow you to run MAESTRO with minimal interaction, while being assured that the data is archived to HPSS on the OLCF machines.

To use the scripts, first create a directory in HPSS that has the same name as the directory on lustre you are running in (just the directory name, not the full path). E.g. if you are running in a directory call `wdconvect_run`, then do:

```
hsi
mkdir wdconvect_run
```

(Note: if the hsi command prompts you for your password, you will need to talk to the OLCF help desk to ask for password-less access to HPSS).

The script `process.titan` is called from `titan.run` and will run in the background and continually wait until checkpoint or plotfiles are created (actually, it always leaves the most recent one alone, since data may still be written to it, so it waits until there are more than 1 in the directory).

Then the script will use `htar` to archive the plotfiles and checkpoints to HPSS. If the `htar` command was successful, then the plotfiles are copied into a `plotfile/` subdirectory. This is actually important, since you don't want to try archiving the data a second time and overwriting the stored copy, especially if a purge took place.

Additionally, if the `ftime` executable is in your path (`ftime.f90` lives in `amrex/Tools/Postprocessing/F_src/`), then the script will create a file called `ftime.out` that lists the name of the plotfile and the corresponding simulation time.

Finally, right when the job is submitted, the script will tar up all of the diagnostic files created by `diag.f90` and `ftime.out` and archive them on HPSS. The `.tar` file is given a name that contains the date-string to allow multiple archives to co-exist.

The `titan.run` submission script has code in it that will look at the most recently generated checkpoint files, make sure that they were written out correctly (it looks to see if there is a Header file, since that is the last thing written), and automatically set the `--restart` flag on the run line to restart from the most recent checkpoint file. This allows you to job-chain a bunch of submission and have them wait until the previous job finished and then automatically queue up:

```
qsub -W depend=afterany:<JOB-ID>  <QSUB SCRIPT>
```

where `<JOB-ID>` is the id number of the job that must complete before the new submission runs and `QSUB SCRIPT` is the submission script (e.g. `titan.run`). This way you can queue up a bunch of runs and literally leave things alone and it will restart from the right place automatically and store the data as it is generated.

When `process.titan` is running, it creates a lockfile (called `process.pid`) that ensures that only one instance of the script is running at any one time. Sometimes if the machine crashes, the `process.pid` file will be left behind, in which case, the script aborts. Just delete that if you know the script is not running.

The `chainsub.sh` script can be used to automatically launch a number of jobs depending on a single, currently queued (or running) job.

Profiling and Debugging on GPUs

To get an idea of how code performs on Titan's GPUs, there are a few tools available. We'll overview a few here.

Score-P with CUBE and vampir

Score-P is a profiling and tracing tool that can be used to instrument code to generate data for other tools to analyze, such as CUBE and vampir. These tools have been developed to analyze performance of HPC codes that run on several nodes, not specifically for analyzing GPU usage. Still, they do support some GPU analysis. In the next section, we'll discuss NVIDA's tools specifically for analyzing GPU usage. At the time of writing, Score-P usage is fairly well documented on OLCF's website here: https://www.olcf.ornl.gov/kb_articles/software-scorep/. We'll review the essentials here, but please see the link for more details.

To instrument a code with Score-P you must re-compile. First, your desired modules will need to be loaded. Please note that *order is important* — you need to load modules needed for compilation before loading Score-P. The Score-P module is designed to detect the loaded environment and will configure itself based on that. These tools have been tested with the PGI 16.3.0 compilers, and we will use them in our examples. One possible set of module loads is

```
$ module load PrgEnv-pgi
$ module swap pgi/15.7.0 pgi/16.3.0
$ module load cudatoolkit
$ module load scorep/3.0-rc1
```

In the above we've loaded version 3.0, release candidate 1, which added some support for analyzing OpenACC code. The next step is to compile. You essentially preface your normal compile (and link) line with the Score-P executable and options. As an example using the Fortran wrapper required on Titan, we have

```
$ scorep --cuda --openacc -v ftn gpuprogram.f90
```

One way to achieve this in MAESTRO is to modify the appropriate make file. For PGI, this would be $AMREX_HOME/Tools/F_mk/comps/Linux_pgi.mak. If this proves useful, it may be worth it to build Score-P into the build infrastructure.

Once compiled, we are ready to generate profiling and tracing data. Among those that develop these tools, note that they draw a distinction between profiling and tracing. Profiling generates a timing (or perhaps other metric) summary of the entire program's execution while tracing produces a time-line of the execution. Score-P's analysis is configured with environment variables. Some of the key configuration variables used in testing include

```
export SCOREP_ENABLE_PROFILING=yes
export SCOREP_ENABLE_TRACING=yes
export SCOREP_EXPERIMENT_DIRECTORY=profile-trace
export SCOREP_CUDA_ENABLE=yes,kernel_counter,flushatexit
export SCOREP_CUDA_BUFFER=200M
export SCOREP_TOTAL_MEMORY=1G
export SCOREP_OPENACC_ENABLE=yes
```

For a full listing and definition of configuration variables, execute

```
$ scorep-info config-vars --full
```

Except for very simple codes, you will never want to enable both tracing and profiling. The overhead is too high, and the code will likely crash or be excessively slow. Typically, it's best to profile first and then trace. The profiling data can be used to help configure tracing (as we'll see shortly).

Once the configuration is set, simply run the code as you normally would. Experience suggests you will need to load the same modules that were loaded for compilation when executing. If analysis is being done through a batch script, note that you cannot do a simple `module load ...` in the script. First you need to do `source /opt/modules/default/init/bash` in the script, and then module loads will work as usual.

After executing, analysis data will be stored in the specified `SCOREP_EXPERIMENT_DIRECTORY`. With profiling, you will see a file like `profile.cubex`. This can be opened with cube (`module load cube`).

As mentioned, the profiling data can be used to get recommended settings for tracing. Running

```
$ scorep-score -r profile.cubex
```

will yield output showing estimated sizes for e.g. `SCOREP_TOTAL_MEMORY`. It also list functions that are called many times. If you don't care about them and they're slowing `Score-P` down (or making an outrageously large output file), you can configure `Score-P` to ignore them in its analysis. To filter a set of functions, you need to provide a filter file, for example

```
$ export SCOREP_FILTERING_FILE=scorep.filter
```

where

```
$ cat scorep.filter
SCOREP_REGION_NAMES_BEGIN
 EXCLUDE
   matmul_sub
   matvec_sub
SCOREP_REGION_NAMES_END
```

This would tell `Score-P` not to trace the routines `matmul_sub` and `matvec_sub`. See the OLCF KnowledgeBase article and/or `Score-P`'s help for more, but this doesn't seem to be the best-documented aspect of the program.

Running with tracing enabled will generate a traces.otf2 file that can be inspected with vampir (module load vampir)

`nvprof` and `nvvp`

NVIDIA provides tools for specifically analyzing how your code utilizes their GPUs. `Score-P` is a fully-featured profiler with some CUDA and OpenACC support. It can be useful for providing context for GPU execution and it allows you to, for example, see line numbers for OpenACC directives that are executed. `nvprof` will only analyze GPU execution, but in exchange you get much more detail than is available with `Score-P`. `nvvp` is NVIDIA's visual profiler. It can be used to read data generated

by nvprof. Most useful is the guided analysis it will perform, which analyzes your code's GPU performance for bottlenecks and suggests ways to improve performance. Both are provided when you load the cudatoolkit module.

With nvprof, no instrumentation is necessary. Instead, you compile normally and then run nvprof on the executable. As before, be sure when executing to load the modules used at compile-time. Executing nvprof on Titan's compute nodes requires some unexpected options having to do with how aprun and nvprof interact.

To get a basic overview printed to the terminal on Titan's compute node, execute

```
$ aprun -b nvprof --profile-child-processes ./gpuprogram.exe arg1 arg2...
```

To generate tracing data for nvvp, execute

```
$ aprun -b nvprof --profile-child-processes -o nvprof.timeline.out%p
  ./gpuprogram.exe arg1 arg2...
```

nvvp can then be used to read nvprof.timeline.out%p, where the %p will be replaced with the process ID. You *must* include %p in the output file's name or the code will crash, even if you're not running a multi-process code.

To generate profile-like metric data for nvvp, execute

```
$ aprun -b nvprof --profile-child-processes --analysis-metrics
  -o nvprof.metrics.out%p ./gpuprogram.exe arg1 arg2...
```

This is the output needed for nvvp's guided analysis.

Target Metrics

The output from profilers may be difficult to makes sense of. The purpose of this section is to note different metrics and reasonable targets for them. Note that these may be specific to the k20x hardware in Titan.

- Threads per block: 256-512. Note that if your code requires many registers per thread, then this will limit the number of threads per block.
- Occupancy: 60% is a reasonable target. We have had success with codes even achieving only 23% occupancy.

One very useful tool for determining target metrics and what is limiting your performance is a spreadsheet developed by NVIDIA to calculate occupancy. Every installation of the CUDA Toolkit should have this occupancy calculator in a tools subdirectory. At time of writing, the calculator is also available at this link: http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls. The document is actually more than a simple calculator. It contains quite a bit of interesting insight into optimizing a GPU code. More on occupancy can be found here: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#occupancy.

Batch Submission of yt Visualization Scripts

Rhea—preferred method

this section needs to be updated. See the titan section

The best way to do visualization is to use rhea, the OLCF vis machine. You need to build yt via the `install_script.sh` script *on rhea*. It also must be on a *Lustre filesystem*, so it is seen by the compute node. It is best to build it in your `$PROJWORK` directory, since that has a longer time between purges. Once the installation is complete, it will tell you what script to source to define the necessary paths.

The scripts in `MAESTRO/Util/job_scripts/rhea/` will handle the visualization. On rhea, the job script gives you access to the compute node, and then you can run serial jobs or a parallel job with `mpirun`. The `process-rhea.run` script will request the resources and run the `parallel-yt-rhea` script. `parallel-yt-rhea` will launch the visualization process (defined via the variables at the top of the script) on all the plotfiles found to match the prefix defined in the script. Several serial jobs are run together, with the script using lock files to keep track of how many processors are in use. When processors free, the next file in the list is processed, and so on, until there are no more plotfiles left to process. If a `.png` file matching the plotfile prefix is found, then that plotfile is skipped.

Note: the line in `parallel-yt-rhea` that `sources` the yt `activate` script may need to be modified to point to the correct yt installation path.

Titan

You can run yt python scripts in the titan batch queues to do your visualization. You need to install yt and all its dependencies manually somewhere on a *Lustre filesystem*—this ensures that the compute nodes can see it. A good choice is the project space, since that has a longer purge window. The following procedure will setup the development version of yt (from source)

- create a directory in your `$PROJWORK` directory named `yt/`
- in `yt/`, down load the yt install script:

  ```
  wget https://bitbucket.org/yt_analysis/yt/raw/yt/doc/install_script.sh
  ```

- edit the script to use Conda to get the necessary dependencies and to build yt from source. This is accomplished by setting the following variables near the top of the script:

  ```
  INST_CONDA=1
  INST_YT_SOURCE=1
  ```

- run the script:

  ```
  source install_script.sh
  ```

When the script is done, you will have a new python installation in a sub-directory called `yt-conda/` and the script will tell you how to modify your path in your `.bashrc`

Important: make sure that you are not loading any other python environments in your `.bashrc`, e.g., via modules.

To test thing out, start up the python shell, and try doing `import yt`. If there are no errors, then you are good.

The python code `vol.py` and submission script `yt-vis-titan.run` in `MAESTRO/Util/job_scripts/titan/vis/` show how to do a simple visualization in the batch queue using yt. Note that `vol.py` is executable, and that we run it via `aprun` to ensure that it is launched on the compute node.

The scripts `vis-titan.run` and `parallel-yt-new` in that same directorywill manage the yt jobs by calling a python script for each file that matches a pattern. Note that the actual visualization command itself is launched by `parallel-yt-new`, again using the `aprun` command. But `aprun` can only launch a single job at a time, so this means we cannot easily do (trivally) parallel visualization on a node. For this reason, running on rhea is preferred.

Remote VisIt Visualization on Lens

Note: this information may be out-of-date. It is recommended that yt *be used instead.*

For large data files, visualization with VisIt should be done with a client running on your local machine interfacing with VisIt running on the remote machine. For the `lens` machine at NCCS, the proper setup is described below.

First, on `lens`, in your `.bashrc`, add:

```
export PATH="/sw/analysis-x64/visit/bin/":$PATH
```

(you would think that you could just add `module load visit` but this does not seem to work with VisIt.

On your local machine, launch VisIt. Note: this procedure seems to work with VisIt 2.4.2, but not VisIt 2.5.0 for some reason.

- First we setup a new host
  - From the menu, select *options* → *host profiles*
  - Create a new host by clicking on the *New Host* button.
  - Enter the *Host nickname* as `lens`
  - Enter the *Remote host name* as `lens.ccs.ornl.gov`
  - Enter the *Path to Visit installation* as `/sw/analysis-x64/visit` (not sure if this is needed)
  - Make sure that your *username* is correct
  - Check *Tunnel data connections through SSH*
- Setup the *Launch Profiles*
  - Click on the *Launch Profiles* tab
  - Click on the *New Profile* button
  - Enter the *Profile name* as `parallel`
  - Click on the *Parallel* tab

  - Check *Launch parallel engine*
  - Select the *Parallel launch method* as `qsub/mpirun`
  - Set the *Partition / Pool / Queue* to `computation`
  - Change the *Default number of processors* to 8
  - Set the *Default number of nodes* to 2
  - Set the *Default Bank / Account* to `AST006`
  - Set the *Default Time Limit* to `00:30:00`
- Click on *Apply* and *Post*
- Save your changes by selecting *Options* → *Save Settings*

To do remote visualization, select *File* → *Open*. From the drop down list at the top, select `lens`. You will be prompted for your password. After that, you can navigate to the directory on lens with the data.

To make a movie (output sequence of images):

- save a view in VisIt you like as a session file (File → Save session).
- On lens, create a file called `files.visit` which lists all of the files you want to visualize, one per line, with `/Header` after the filename. This can be done simply as:

  ```
  ls -1 | grep -v processed | awk '{print $1"/Header"}' > files.visit
  ```

  (note: the `processed` bit is for when you used the script above to automatically archive the data).
- Edit the session file, searching for the name of the plotfile you originally visualized, and change it to read `files.visit`. Make sure that the path is correct. This may appear multiple times.
- Relaunch VisIt locally and restore the session (File → Restore session). It will render the first image. Then reopen (File → ReOpen file). After this is done, the buttons that allow you to move through the files should become active (black).
- Resave the session file
- To generate the frames, you have 2 options:

  1. File → Save movie. Pick *New simple movie*, then set the format to *PNG* and add this to the output box by clicking the right arrow, then in the very last screen, select: Later, tell me the command to run.

     VisIt will pop up a box showing the command to run. Trying to get the currently running session of VisIt to generate the frames seems problamatic. Note: you will probably want to edit out the `-v x.x.x` argument in the commandline to not have it force to use a specific version.

  2. If the session file successfully does the remote visualization as desired, you can run the movie via the commandline with something like:

     ```
     visit -movie -format png -geometry 1080x1080 -output subchandra_cutoff3_ \
         -start 0 -end 44 -sessionfile subchandra_radvel.session
     ```

Working at NERSC

edison compilers

The default compilers on edison are the Intel compilers, but PGI and Cray also work well

- Intel 15.0.1 works well on edison/NERSC with MPI/OpenMP
- Intel 16.0.2 works fine.
- Cray 8.4.x has worked in the past, but it has not been used at NERSC in a while.

this may have changed with the migration of NERSC to anaconda?

Note: in order to compile, you will need to ensure that both the `python` and `python_base` modules are loaded (via the `module` command).

Running Jobs

edison is configured with 24 cores per node split between two Intel IvyBridge 12-core processors. Each processor connects to 1/2 of the node's memory and is called a NUMA node, so there are 2 NUMA nodes per edison node. Best performance is seen when running with 6 or 12 threads.

Note: edison switched to SLURM as the batch system. Your job is submitted using the `sbatch` command. Options to `sbatch` are specified at the top of your submission script with `#SBATCH` as the prefix. These options can be found on the `sbatch manpage`. For instance,

```
#SBATCH -N 2
#SBATCH -J myjob
#SBATCH -A repo-name
#SBATCH -p regular
#SBATCH -t 12:00:00
```

will request 2 nodes (`-N`), under the account `repo-name` (`-J`), in the regular queue, and for a 12-hour window `-t`.

If you are using OpenMP, then your script should set `OMP_NUM_THREADS`, e.g.,

```
export OMP_NUM_THREADS=12
```

By default, SLURM will change directory into the submission directory. The job is launched from your script using `srun`, e.g.:

```
srun -n 48 ./main.Linux.Cray.mpi.exe inputs_3d
```

to run 48 MPI tasks (across the 2 nodes), or

```
export OMP_NUM_THREADS=6
srun -n 8 -c 6 ./main.Linux.Cray.mpi.omp.exe inputs_3d
```

to use 8 MPI tasks each with 6 threads.

The scripts in `Util/job_scripts/edison/` provides some examples.

To chain jobs, such that one queues up after the previous job finished, use the `chainslurm.sh` script in that same directory. You can view the job dependency using:

```
squeue -l -j job-id
```

where `job-id` is the number of the job.

Jobs are submitted with `sbatch`. A job can be canceled using `scancel`, and the status can be checked using `squeue -u` *`username`*.

Automatic Restarting and Archiving of Data

The same set of submission scripts described for titan are available for edison at NERSC in `Util/job_scripts/ediso` In particular, the job submission script will set the restart command line parameters to restart from the most recent checkpoint file in the output directory.

Note: NERSC does not allow for the `process` script that archives to HPSS to run in the main job submission script. Instead, a separate job needs to be run in the "`xfer`" queue. The script `edison.xfer.slurm` in `Util/job_scripts/edison/` shows how this works.

Jobs in the `xfer` queue start up quickly. The best approach is to start one as you start your main job (or make it dependent on the main job). The sample `process.xrb` script will wait for output and then archive it as it is produced, using the techniques described for titan above.

To check the status of a job in the `xfer` queue, use:

```
squeue -u username -M all
```

Batch visualization using yt

yt can be built using the `install_script.sh`. It has been tested using the build of `yt` from source and dependencies via conda, by setting:

```
INST_CONDA=1
INST_YT_SOURCE=1
```

in the `install_script.sh`. Once these are set, run:

```
source install_script.sh
```

Note: installation was done in the home directory.

This way of building yt installs it's own python and support libraries in a directory, `yt-conda`. **Important:** You need to make sure that your start-up files (typically `.bashrc.ext` at NERSC) don't `module load` python or any python libraries, as this will interfere with the conda installation. The install script will direct you to add the install location to your path.

The scripts `parallel-yt` and `process-edison.slurm` in `Util/job_scripts/edison` show how to invoke yt to loop over a series of plotfiles and do visualization. A number of tasks are run at once on the node, each operating on a separate file. The `parallel-yt` script then calls `vol.py` to do the volume rendering with yt. Note: it is important that `srun` be used to launch the yt script to ensure that it is run on the compute node.

A simple `test-yt.slurm` script shows how to just call the yt python script directly, using one node and 24 threads, again using `srun` to execute on the compute node.

If you want to keep up with the development version of yt, then you can update the source in `yt-conda/bin/src/yt-hg`, using:

```
hg pull
hg update yt
```

and then rebuild it via:

```
python setup.py develop --user
```

Using the `AmrPostprocesing` python plotting scripts on hopper

To build the `fsnapshot.so` library, you need to do:

```
module load gcc
```

`f2py` is already in the path, so the library should then build without issue.

Then edit your `.bashrc.ext` file to set the `PYTHONPATH` to the `python_plotfile` directory, e.g.:

```
export PYTHONPATH="/global/homes/z/zingale/AmrPostprocessing/python"
```

and set the `PATH` to that directory,

```
export PATH="/global/homes/z/zingale/AmrPostprocessing/python:$PATH"
```

To run the script, you need to do:

```
module load matplotlib
module load python
```

Remote visualization on hopper

VisIt is already configured to work with hopper. If the host does not appear in your local version of visit, copy the `host_nersc_hopper.xml` file from the `.visit/allhosts/` directory under the system's VisIt install path to your ∼`/.visit/hosts/` directory.

Working at NCSA (Blue Waters)

Overview

Blue Waters consists of 22,640 Cray XE6 compute nodes and 4,224 Cray XK7 compute nodes.

Each XE node has two AMD Interlagos model 6276 compute units, each of which has 16 integer cores (thus, a single node has a total of 32 integer cores). Two integer cores share a multithreaded, 256-bit wide floating point unit (FPU). If both integer cores have their own thread, each has access to 128-bit floating point processing, whereas if only one thread is assigned the process can access all 256 bits. In one major science application on Blue Waters it was found that having an OpenMP thread for each

integer core gave the best performance, but when starting a new application it's best to experiment. One OpenMP thread per FPU may be better in some cases.

Each compute unit is divided into two NUMA nodes. Cores in the same NUMA region share a pool of L3 cache. For the same science application as before it was found that the best performance was achieved by assigning an MPI task to each NUMA node. Thus, each physical node has four MPI tasks.

The XK nodes consist of one AMD Interlagos model 6276 compute unit and an NVIDIA GK110 "Kepler" GPU accelerator (Tesla K20X). The GPU is configured with 14 streaming multiprocessor units (SMXs), each of which has 192 single-precision or 64 double-precision CUDA cores. Thus there are a total of 2688 SP CUDA cores or 896 DP CUDA cores.

For more details, please see `https://bluewaters.ncsa.illinois.edu/user-guide`

BW Compilers

The Cray compilers are the default on blue waters, and version 8.3.3 works well with MAESTRO.

Monitoring Allocations

The `usage` command will list the current user's usage and `usage -P` *`project`* will list the usage for all users in a project allocation named "project".

CHAPTER 12

FAQ

Coding

1. *Why is everything in its own module?*

   If a subroutine is in a Fortran module, then at compile time, there is argument checking. This ensures that the right number and data types of arguments are present. It makes the code safer.

2. *How do tags work when editing source?*

   Tags allow the editor to follow function/subroutine names and automatically switch you to the source code corresponding to that function. Using tags in MAESTRO depends on the editor:

   - `vi`:

     In the build directory, type '`make tags`'. Then in `vi`, move the cursor over a function name and use `^-]` to bring up the source corresponding to that function. Use `^-t` to go back. (Here, `^-` refers to the Control key.)

   - `emacs`:

     In the build directory, type '`make TAGS`'. Then in `emacs`, move the cursor over a function name and use `M-.` to bring up the source corresponding to that function. Use `M-*` to go back. (Here, `M-` refers to the META key.)

Compiling

1. *What version of the Fortran standard is needed?*

Some features of Fortran 2003 are used, in particular, the `ISO_C_BINDING` feature of Fortran 2003 is needed to define a long integer type for some MPI operations in Fortran. This is supported by most Fortran compilers even if they don't support the entire Fortran 2003 standard.

We also rely on the Fortran 95 standard that specifies that any local allocated arrays are automatically deallocated when a subroutine ends. Fortran 90 does not do this. Most MAESTRO routines rely on this Fortran 95 feature.

2. *The code doesn't compile, but complains right away that there is "*`No rule to make target 'fabio_c.c', needed by 't/Linux.gfortran.mpi/c.depends'`*"*

   The environment variable `AMREX_HOME` needs to be the full path to the `amrex/` directory. You cannot use '∼' as a shortcut for your home directory.

3. `make` *issues an error like:*

```
.../BoxLib/Tools/F_mk/GMakeMPI.mak:40: Extraneous text after 'else' directive
.../BoxLib/Tools/F_mk/GMakeMPI.mak:47: *** only one 'else' per conditional.  Stop
```

   You need to use GNU make version 3.81 or later. Earlier versions do not support an *else-if* clause.

Running

1. *How do we turn off all the initial projections to look at the initial velocity field as specified in initdata, instead of as modified by the velocity constraint?*

```
max_step  = 1
init_iter = 0
init_divu_iter = 0
do_initial_projection = F
```

2. MAESTRO *crashes because the multigrid algorithm fails to converge—how do I get around this?*

   Setting general convergence criteria for multigrid is as much art as science. First, it is important to determine if the multigrd solver is close to convergence and just dancing around near the desired tolerance, but never reaching it, or if it is no where near convergence. For the latter, it may be that the multigrid solver was fed bad data and the problem arose in one of the earlier steps. To get more detail information from the multigrid solver, set `mg_verbose` to a positive integer from 1-4 (the higher the number the more information you receive.

   If the multigrid solver is failing during one of the initial "divu" iterations, it may be because the velocity is initially zero, so there is no velocity magnitude to use as a reference for convergence, and that $(S - \bar{S})$ is very small (or zero). In this case, it is usually a good idea to perturb the initial state slightly, so the righthand side is non-zero.

   The tolerances used for the various multigrid solves in the code can be overridden on a problem-by-problem basis by putting a copy of `MAESTRO/Source/mg_eps.f90` into the problem directory and resetting the tolerances. The role of each of these tolerances is described in `MAESTRO/docs/mg/`.

3. *Why do the initial projection and "divu" iters sometimes have a harder time converging than the multigrid solves in the main algorithm?*

The initial projection and "divu" solve sets the density to 1 (see § 2.3), so the coefficients in the elliptic solve are $O(\beta_0) \sim O(\rho)$. But in the main algorithm, the coefficients are $O(\beta_0/\rho) \sim O(1)$. Since ρ can vary a lot, the variation in the coefficients in the initial projection and "divu" solve present a harded linear system to solve.

4. *How can I obtain profiling infomation for my run?*

   The code is already instrumented with timers. Simply compile with `PROF=TRUE` in the `GNUmakefile`, or equvalently do `make PROF=t`. A file containing a summary of the timings will be output in the run directory.

   An alternate way to get single-processor timings, when using the GCC compilers is to add `-pg` to the compilation flags for both `gfortran` and `gcc`. This can be accomplished by setting:

   ```
   GPROF := t
   ```

   in your `GNUmakefile`. Upon completion, a file names `gmon.out` will be produced. This can be processed by `gprof` running

   ```
   gprof exec-name
   ```

   where *exec-name* is the name of the executable. More detailed line-by-line timings can be obtained by using the `-l` argument to `gprof`.

5. *How can I force* MAESTRO *to abort?*

   In the output directory, do '`touch .abort_maestro`'. This will cause the code to write out a final checkpoint file, free up any allocated memory, and gracefully exit. Be sure to remove the `.abort_maestro` file before restarting the code in the same directory.

Debugging

1. *How can we dump out a variable to a plotfile from any point in the code?*

   ```
   use fabio_module

   call fabio_ml_multifab_write_d(uold,mla%mba%rr(:,1),"a_uold")
   call fabio_ml_multifab_write_d(umac(:,1),mla%mba%rr(:,1),"a_umacx")
   ```

2. *How can I print out a multifab's contents from within the code?*

   There is a `print` method in `multifab_module`. This can be simply called as

   ```
   call print(a)
   ```

   where `a` is a multifab (single-level).

3. *How can I debug a parallel (MPI) job with gdb?*

   If you only need to use a few processors, the following command will work:

   ```
   mpiexec -n 4 xterm -e gdb ./main.Linux.gfortran.mpi.exe
   ```

where the executable needs to be created with the "-g" flag to the compiler. This will pop up multiple xterms with gdb running in each. You need to then issue:

```
run inputs
```

where inputs is the desired inputs file *in each* xterm.

4. *How can I get more information about floating point exceptions?*

   AMReX can intercept floating point exceptions and provide a helpful backtrace that shows you where they were generated. See §10.4.

I/O

1. *How can I tell from a plotfile what runtime parameters were used for its run? or when it was created?*

   In each plotfile directory, there is a file called job_info (e.g. plt00000/job_info) that lists the build directory and date, as well as the value of every runtime parameter for the run.

2. *How can I force the code to output a plotfile / checkpoint file at the next step?*

   In the output directory (where the code is running) do 'touch .dump_plotfile'. This will create an empty file called .dump_plotfile. At the end of each step, if the code finds that file, it will output a plotfile. Simply delete the file to restore the code to its normal plotfile behavior.

   Similarly, creating the file .dump_checkpoint will force the output of a checkpoint file.

Algorithm

1. *Why is* MAESTRO *so "hard" to use (e.g. as compared to a compressible code)?*

   There are several complexities to the algorithm that don't have straightforward compressible counterparts. These mainly involve the role of the base state and the constraint equation.

   Care must be taken to setup an initial model/initial base state that respects the thermodynamics in MAESTRO and is in hydrostatic equilibrium. Best results are attained when the model is processed with the MAESTRO EOS and reset into HSE, as is done in the initial_model routines. Because MAESTRO builds off of the base state, any flaws in that initial state will influence the subsequent behavior of the algorithm.

   The constraint equation brings another complexity not seen in compressible codes—information is instantly communicated across the grid. In compressible codes you can track down a problem by watching where it starts from and watching it move one cell per dt. In MAESTRO things can go wrong in multiple places without it being obvious where the root problem is.

2. *In the final projection in the algorithm, we project U^{n+1}, using a time-centered β_0, a time-centered ρ_0, but an "$n+1$"-centered S. Why then is the resulting ϕ (which then defines π) is at "$n+1/2$"?*

   The short answer to this question is that you should think of this as really projecting $(U^{n+1} - U^n)$ and the right hand side as having $(S^{n+1} - S^n)$. This is because the pressure enters the dynamic

equations as $(U^{n+1} - U^n) = \ldots + \frac{1}{\rho^{n+1/2}} \nabla \pi^{n+1/2}$. (We approximate $\pi^{n+1/2}$ by $\pi^{n-1/2}$ then do the projection to fix the π as well as the U.)

So everything is in fact time-centered.

3. *Why is $\overline{\Gamma}_1$ computed as the average of the full state Γ_1 instead of computed from the base state density and pressure via the equation of state?*

   The primary reason is that there is no base state composition. The base state density is simply the average of the full state density, and the base state pressure is the pressure required for hydrostatic equilibrium. There is no thermodynamic relationship enforced between these base state quantities.

4. *Can I run a full star in 2-d axisymmetric geometry?*

   No. This is a design decision. There is no support for axisymmetric coordinates in MAESTRO. Spherical problems must be run in 3-d.

5. *Why did we switch all the equations over to the $\tilde{\mathbf{U}}$ form instead of just working with $\mathbf{U}$?*

   This is basically a numerical discretization issue. Whenever the base state aligns with the grid, you should be able to show that you get exactly the same answer each way.

   When you do a spherical star on a 3d Cartesian grid, though, the w_0 is defined on the radial mesh and the $\tilde{\mathbf{U}}$ on the Cartesian mesh, and the w_0 part never experiences the Cartesian projection, for example. So there are differences in exactly how the w_0 component appears (projected on the Cartesian mesh vs. interpolated from the radial mesh)—we made the decision at the time to separate the components for that reason.

6. *Why does "checkerboarding" appear in the velocity field, especially in regions where the flow is stagnant?*

   Checkerboarding can arise from the projection—it doesn't see that mode (because it is an approximate projection) so it is unable to remove it. This allows the pattern to slowly build up. There are filtering techniques that can be used to remove these modes, but they are not implemented in MAESTRO.

Analysis

1. *I want to open a plotfile, derive a new quantity from the data stored there, and write out a new plotfile with this derived data. How do I do this?*

   One implementation of this can be found in `amrex/Tools/Postprocessing/F_Src/tutorial/fwrite2d.f90`. This reads in the plotfile data using the `plotfile_module` that the `data_processing` routines rely on, but then builds a multifab and writes the data out to a plotfile using the AMReX write routines.

Part III

MAESTRO Technical Details

CHAPTER 13

Notes on the Low Density Parameters in MAESTRO

These are working notes for the low density parameters in MAESTRO. In low density regions, we modify the behavior of the algorithm. Here is a summary of some parameters, and a brief description of what they do.

- `base_cutoff_density`, ρ_{base}, (real):

  Essentially controls the lowest density allowed in the simulation and modifies the behavior of several modules.

- `base_cutoff_density_coord(:)` (integer array):

  For each level in the radial base state array, this is the coordinate of the first cell where $\rho_0 < \rho_{\text{base}}$. Slightly more complicated for multilevel problems.

- `anelastic_cutoff`, $\rho_{\text{anelastic}}$, (real):

  If $\rho_0 < \rho_{\text{anelastic}}$, we modify the computation of β_0 in the divergence constraint.

- `anelastic_cutoff_coord(:)` (integer array):

  Anelastic cutoff analogy of `base_cutoff_density_coord(:)`.

- `burning_cutoff_density`, ρ_{burning}, (real):

  If $\rho < \rho_{\text{burning}}$, don't call the burner in this cell.

- `burning_cutoff_density_coord(:)` (integer array):

  Burning cutoff analogy of `base_cutoff_density_coord(:)`.

- `buoyancy_cutoff_factor` (real):

  When computing velocity forcing, set the buoyance term $(\rho - \rho_0)$ to 0 if $\rho <$ `buoyancy_cutoff_factor` * `base_cutoff_density`.

- `do_eos_h_above_cutoff` (logical):

  If true, at the end of the advection step, for each cell where $\rho < \rho_{\text{base}}$, recompute $h = h(\rho, p_0, X)$.

Computing the Cutoff Values

We compute `anelastic_cutoff_coord`(:), `base_cutoff_density_coord`(:), and `burning_cutoff_density_coord`(:) in analogous fashion.

Single-Level Planar or any Spherical

Here the base state exists as a single one-dimensional array with constant grid spacing Δr. Basically, we set the corresponding coordinate equal to r as soon as $\rho_0(r)$ is less than or equal to that particular cutoff value. See Figure 13.1 for a graphical representation.

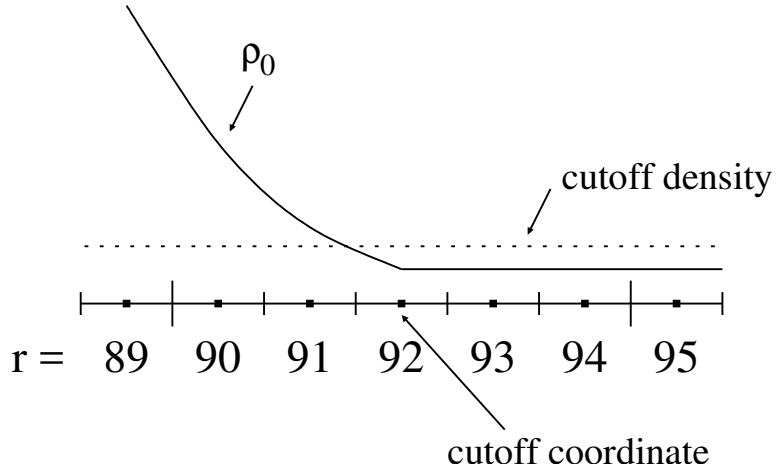


Figure 13.1: Image of how the cutoff density and cutoff coordinates are related for single-level planar and all spherical problems.

Note that for single-level planar or any spherical problem, saying $r \geq$ `anelastic_cutoff_coord` is analogous to saying $\rho_0(r) \leq$ `anelastic_cutoff`. Also, saying $r <$ `anelastic_cutoff_coord` is analogous to saying $\rho_0(r) >$ `anelastic_cutoff`. Ditto for `base_cutoff_density` and `base_cutoff_density_coord`.

Multilevel Planar

In this case, the base state exists as several one-dimensional arrays, each with different grid spacing. Refer to Figure 13.2 in the following examples. The guiding principle is to check whether ρ_0 falls below

ρ_{cutoff} on the finest grid first. If not, check the next coarser level. Continue until you reach the base grid. Some examples are in order:

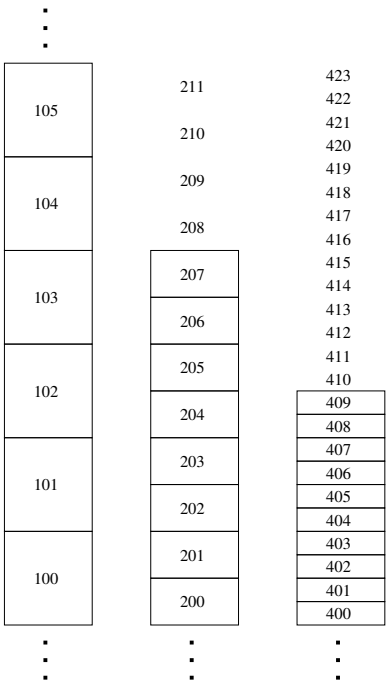


Figure 13.2: Multilevel cutoff density example.

- **Example 1:** $\rho_{0,104} > \rho_{\text{cutoff}}$ and $\rho_{0,105} < \rho_{\text{cutoff}}$.

  `cutoff_density_coord`(1) = 105
  `cutoff_density_coord`(2) = 210
  `cutoff_density_coord`(3) = 420

  This is the simplest case in which the cutoff transition happens on the coarsest level. In this case, the cutoff coordinates at the finer levels are simply propagated from the coarsest level, even though they do not correspond to a valid region.

- **Example 2:** $\rho_{0,403} > \rho_{\text{cutoff}}$ and $\rho_{0,404} < \rho_{\text{cutoff}}$.

  `cutoff_density_coord`(1) = 101
  `cutoff_density_coord`(2) = 202
  `cutoff_density_coord`(3) = 404

  In this case, the cutoff transition happens where the finest grid is present. Happily, the transition occurs at a location where there is a common grid boundary between all three levels. Therefore, we simply propagate the cutoff density coordinate from the finest level downward.

- **Example 3:** $\rho_{0,404} > \rho_{\text{cutoff}}$ and $\rho_{0,405} < \rho_{\text{cutoff}}$.

  `cutoff_density_coord`(1) = 102
  `cutoff_density_coord`(2) = 203

cutoff_density_coord(3) = 405

In this case, the cutoff transition happens where the finest grid is present. However, the transition occurs at a location where there NOT is a common grid boundary between all three levels. We choose to define the cutoff transition at the coarser levels as being at the corresponding boundary that is at a larger radius than the location on the finest grid.

Note: if ρ_0 does not fall below $\rho_{\rm cutoff}$ at any level, we set the cutoff coordinate at the fine level to be first first cell above the domain and propagate the coordinate to the coarser levels.

When are the Cutoff Coordinates Updated?

At several points in the algorithm, we compute anelastic_cutoff_coord(:), base_cutoff_density_coord(:), and burning_cutoff_density_coord(:):

- After we call initialize in varden.
- After reading the base state from a checkpoint file when restarting.
- After regridding.
- After advancing ρ_0 with advect_base_dens.
- After advancing ρ and setting $\rho_0 = \overline{\rho}$.
- At the beginning of the second-half of the algorithm (**Step 6**), we reset the coordinates to the base-time values using ρ_0^n.

Usage of Cutoff Densities

anelastic_cutoff

The anelastic_cutoff is the density below which we modify the constraint.

- In probin, anelastic_cutoff is set to 3×10^6 by default.
- In make_div_coeff, for $r \geq$ anelastic_cutoff_coord, we set div_coeff$(n,r) =$ div_coeff$(n,r-1) * \rho_0(n,r)/\rho_0(n,r-1)$.
- in make_S, we set delta_gamma1_term and delta_gamma1 to zero for $r \geq$ anelastic_cutoff_coord. This is only relevant if you are running with use_delta_gamma1_term = T.
- Some versions of sponge, use anelastic_cutoff in a problem dependent way.

base_cutoff_density

The base_cutoff_density is the lowest density that we model.

- In probin, base_cutoff_density is set to 3×10^6 by default.

- In base_state, we compute a physical cutoff location, base_cutoff_density_loc, which is defined as the physical location of the first cell-center at the coarsest level for which $\rho_0 \leq$ base_cutoff_density. This is a trick used for making the data consistent for multiple level problems. When we are generating the initial background/base state, if we are above base_cutoff_density_loc, just use the values for ρ, T, and p at base_cutoff_density_loc. When we check whether we are in HSE, we use base_cutoff_density_loc.
- In make_hgrhs, make_macrhs, and make_w0, we only add the volume discrepancy for $r <$ base_cutoff_density_coord (in plane parallel) and if $\rho_0^{\text{cart}} >$ base_cutoff_density (in spherical).
- In mkrhohforce for plane-parallel, for $r \geq$ base_cutoff_density_coord, we compute ∇p_0 with a difference stencil instead of simply setting it to $\rho_0 g$.
- In update_scal, if $\rho \leq$ base_cutoff_density and do_eos_h_above_cutoff, we call the EOS to compute h.
- In update_scal, if $\rho \leq$ base_cutoff_density/2 we set it to base_cutoff_density/2.
- In make_grav for spherical, we only add the enclosed mass if $\rho_0 >$ base_cutoff_density.
- In enforce_HSE, we set $p_0(r+1) = p_0(r)$ for $r \geq$ base_cutoff_density_coord.
- In make_psi for plane-parallel, we only compute ψ for $r <$ base_cutoff_density_coord.

burning_cutoff

The burning cutoff determines where we call the reaction network to get the nuclear energy generation rate and composition changes. For densities below the burning cutoff, we do not call the network.

- In probin, burning_cutoff_density is set to base_cutoff_density. There is no option to set burning_cutoff_density using the inputs file.
- In react_state, we only call the burner if $\rho >$ burning_cutoff_density.

buoyancy_cutoff_factor

The buoyancy_cutoff_factor is used to zero out the forcing terms to the velocity equation at low densities.

- In init_base_state we print out the value of the the density at which the buoyancy cutoff would take effect, buoyancy_cutoff_factor * base_cutoff_density.
- In mk_vel_force, we zero out rhopert, the perturbational density used in computing the buoyancy force, if $\rho <$ buoyancy_cutoff_factor $*$ base_cutoff_density.
- In mk_vel_force, for spherical problems, we zero out centrifugal_term, the centrifugal force for rotating stars, if $\rho <$ buoyancy_cutoff_factor $*$ base_cutoff_density.
- In make_explicit_thermal, if limit_conductivity = T, then for $\rho <$ buoyancy_cutoff_factor $*$base_cutoff_density, we zero out the thermal coefficients, effectively turning off thermal diffusion there.

CHAPTER 14

Notes on the Volume Discrepancy Term

The volume discrepancy term is used in the constraint equation to force the system back to the equation of state. We write our velocity constraint equation as

$$\nabla\cdot(\beta_0\mathbf{U}) = \beta_0\left(S - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - p_{\rm EOS}}{\Delta t}\right) \quad . \tag{14.1}$$

Here, f is the volume discrepancy factor and ranges from 0 to 1, and $p_{\rm EOS}$ is the thermodynamic pressure as returned by the EOS, using the full state as input. In practice we evaluate this as

$$p_{\rm EOS} = p(\rho, h, X_k) \tag{14.2}$$

The idea behind this forcing term is that if $p_{\rm EOS} > p_0$ then $\nabla\cdot(\beta_0\mathbf{U}) > 0$, and the cell will evacuate. This has the effect of returning us to $p_{\rm EOS} = p_0$.

In MAESTRO, we decomponse the velocity into a base state component and a local component. The base state constraint equation is simply the horizontal average of the full constraint. Starting with $\mathbf{U} = \widetilde{\mathbf{U}} + w_0\mathbf{e}_r$ in equation 14.1, we have

$$\nabla\cdot(\beta_0 w_0\mathbf{e}_r) + \nabla\cdot(\beta_0\widetilde{\mathbf{U}}) = \beta_0\left(S - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - p_{\rm EOS}}{\Delta t}\right) \quad . \tag{14.3}$$

Averaging this over a layer, we note that $\overline{\nabla\cdot(\beta_0\widetilde{\mathbf{U}})} = 0$, yielding

$$\nabla\cdot(\beta_0 w_0\mathbf{e}_r) = \beta_0\left(\overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - \overline{p_{\rm EOS}}}{\Delta t}\right) \tag{14.4}$$

and

$$\nabla\cdot(\beta_0\widetilde{\mathbf{U}}) = \beta_0\left(S - \overline{S} + \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_{\rm EOS} - \overline{p_{\rm EOS}}}{\Delta t}\right) \quad . \tag{14.5}$$

In solving the w_0 evolution equation (Eq. [14.4]), we expand the divergence, giving

$$\nabla\cdot(w_0\mathbf{e}_r) = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial t} - w_0\mathbf{e}_r\cdot\frac{1}{\beta_0}\nabla\beta_0 - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - \overline{p_{\rm EOS}}}{\Delta t} \quad . \tag{14.6}$$

Recalling that

$$\frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial r} = \frac{1}{\beta_0}\frac{\partial \beta_0}{\partial r}$$

(see Paper I), and that $\psi \equiv D_0 p_0/Dt \equiv \partial p_0/\partial t + w_0 \partial p_0/\partial r$, we have

$$\nabla\cdot(w_0\mathbf{e}_r) = \overline{S} - \frac{1}{\overline{\Gamma}_1 p_0}\psi - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - \overline{p_{\rm EOS}}}{\Delta t} \quad . \tag{14.7}$$

This is the form that is solved in `make_w0`.

CHAPTER 15

EOS and Temperature Notes

EOS Calls

Initialization

`advance_timestep`

Step 1. *Define the average expansion at time $t^{n+1/2}$ and the new w_0.*

if `dpdt_factor` > 0 then

- In `makePfromRhoH`, we compute a thermodynamic p^n for the volume discrepancy term using $(\rho, h, X)^n$.

end if

Step 2. *Construct the provisional edge-based advective velocity, $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$.*

Step 3. *React the full state and base state through the first time interval of $\Delta t/2$.*

- In `react_state` $\rightarrow$ `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^n$.
- In `react_state`, we compute $T^{(1)}$ using $(\rho, h, X)^{(1)}$ (if `use_tfromp = F`) or $(\rho, X)^{(1)}$ and p_0^n (if `use_tfromp = T`)

if `evolve_base_state = T` then

- In `make_gamma`, we compute Γ_1 using $(\rho, X)^{(1)}$ and p_0^n.

end if

Step 4. *Advect the base state, then the full state, through a time interval of* Δt.

if `use_thermal_diffusion = T` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{(1)}$.
- In `enthalpy_advance` $\rightarrow$ `update_scal`, we compute h above the `base_cutoff_density` using $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$.
- In `thermal_conduct`, we compute $T^{(2),*}$ using $(\rho, h, X)^{(2),*}$ (if `use_tfromp = F`) or $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

else

- In `enthalpy_advance` $\rightarrow$ `update_scal`, we compute h above the `base_cutoff_density` using $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$.
- In `enthalpy_advance`, we compute $T^{(2),*}$ using $(\rho, h, X)^{(2),*}$ (if `use_tfromp = F`) or $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

end if

Step 5. *React the full state through a second time interval of* $\Delta t/2$.

- In `react_state` $\rightarrow$ `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^{(2),*}$.
- In `react_state`, we compute $T^{n+1,*}$ using $(\rho, h, X)^{n+1,*}$ (if `use_tfromp = F`) or $(\rho, X)^{n+1,*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

if `evolve_base_state` then

- In `make_gamma`, we compute Γ_1 using $(\rho, X)^{n+1,*}$ and $p_0^{n+1,*}$.

end if

Step 6. *Define a new average expansion rate at time* $t^{n+1/2}$.

if `use_thermal_diffusion` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{n+1,*}$.

end if

- In `make_S`, we compute thermodynamic variables using $(\rho, T, X)^{n+1,*}$.

if `dpdt_factor` > 0 then

- In `makePfromRhoH`, we compute a thermodynamic $p^{n+1,*}$ for the volume discrepancy term using $(\rho, h, X)^{n+1,*}$.

end if

Step 7. *Construct the final edge-based advective velocity,* $\widetilde{\mathbf{U}}^{\rm ADV}$.

Step 8. *Advect the base state, then the full state, through a time interval of* Δt.

if `use_thermal_diffusion` = T then

- In `enthalpy_advance` $\rightarrow$ `update_scal`, we compute h above the `base_cutoff_density` using $(\rho, X)^{(2)}$ and p_0^{n+1}.
- In `advance` before `thermal_conduct`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{(2),*}$.
- In `thermal_conduct`, we compute $T^{(2)}$ using $(\rho, h, X)^{(2)}$ (if `use_tfromp = F`) or $(\rho, X)^{(2)}$ and p_0^{n+1} (if `use_tfromp = T`).

else

- In `enthalpy_advance` $\rightarrow$ `update_scal`, we compute h above the `base_cutoff_density` using $(\rho, X)^{(2)}$ and p_0^{n+1}.
- In `enthalpy_advance`, we compute $T^{(2)}$ using $(\rho, h, X)^{(2)}$ (if `use_tfromp = F`) or $(\rho, X)^{(2)}$ and p_0^{n+1} (if `use_tfromp = T`).

end if

Step 9. *React the full state and base state through a second time interval of* $\Delta t/2$.

- In `react_state` $\rightarrow$ `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^{(2)}$.
- In `react_state`, we compute T^{n+1} using $(\rho, h, X)^{n+1}$ (if `use_tfromp = F`) or $(\rho, X)^{n+1}$ and p_0^{n+1} (if `use_tfromp = T`).

if `evolve_base_state = T` then

- In `make_gamma`, we compute Γ_1 using $(\rho, X)^{n+1}$ and p_0^{n+1}.

end if

Step 10. *Compute* S^{n+1} *for the final projection.*

if `make_explicit_thermal` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{n+1}$.

end if

- In `make_S`, we compute thermodynamic variables using $(\rho, T, X)^{n+1}$.

Step 11. *Update the velocity.*

if `dpdt_factor` > 0 then

- In `makePfromRhoH`, we compute a thermodynamic p^{n+1} for the volume discrepancy term using $(\rho, h, X)^{n+1}$.

end if

Step 12. *Compute a new* Δt.

`make_plotfile`

Temperature Usage

`advance_timestep`

Step 1. *Define the average expansion at time $t^{n+1/2}$ and the new w_0.*

Step 2. *Construct the provisional edge-based advective velocity, $\widetilde{\mathbf{U}}^{\mathrm{ADV},\star}$.*

Step 3. *React the full state and base state through the first time interval of $\Delta t/2$.*

- In `react_state` → `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^n$.
- In `react_state`, we compute $T^{(1)}$ using $(\rho, h, X)^{(1)}$ (if `use_tfromp = F`) or $(\rho, X)^{(1)}$ and p_0^n (if `use_tfromp = T`).

Step 4. *Advect the base state, then the full state, through a time interval of Δt.*

if `use_thermal_diffusion = T` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\mathrm{th}}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{(1)}$.
- In `thermal_conduct`, we compute $T^{(2),*}$ using $(\rho, h, X)^{(2),*}$ (if `use_tfromp = F`) or $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

else

- In `enthalpy_advance`, we compute $T^{(2),*}$ using $(\rho, h, X)^{(2),*}$ (if `use_tfromp = F`) or $(\rho, X)^{(2),*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

end if

Step 5. *React the full state through a second time interval of $\Delta t/2$.*

- In `react_state` → `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^{(2),*}$.
- In `react_state`, we compute $T^{n+1,*}$ using $(\rho, h, X)^{n+1,*}$ (if `use_tfromp = F`) or $(\rho, X)^{n+1,*}$ and $p_0^{n+1,*}$ (if `use_tfromp = T`).

Step 6. *Define a new average expansion rate at time $t^{n+1/2}$.*

if `use_thermal_diffusion = T` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\mathrm{th}}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{n+1,*}$.

end if

- In `make_S`, we compute thermodynamic variables using $(\rho, T, X)^{n+1,*}$.

Step 7. *Construct the final edge-based advective velocity, $\widetilde{\mathbf{U}}^{\mathrm{ADV}}$.*

Step 8. *Advect the base state, then the full state, through a time interval of* Δt.

if `use_thermal_diffusion = T` then

- In `advance` before `thermal_conduct`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{(2),*}$.
- In `thermal_conduct`, we compute $T^{(2)}$ using $(\rho, h, X)^{(2)}$ (if `use_tfromp = F`) or $(\rho, X)^{(2)}$ and p_0^{n+1} (if `use_tfromp = T`).

else

- In `enthalpy_advance`, we compute $T^{(2)}$ using $(\rho, h, X)^{(2)}$ (if `use_tfromp = F`) or $(\rho, X)^{(2)}$ and p_0^{n+1} (if `use_tfromp = T`).

end if

Step 9. *React the full state and base state through a second time interval of* $\Delta t/2$.

- In `react_state` $\rightarrow$ `burner`, we compute c_p and ξ_k for inputs to `VODE` using $(\rho, T, X)^{(2)}$.
- In `react_state`, we compute T^{n+1} using $(\rho, h, X)^{n+1}$ (if `use_tfromp = F`) or $(\rho, X)^{n+1}$ and p_0^{n+1} (if `use_tfromp = T`).

Step 10. *Compute* S^{n+1} *for the final projection.*

if `make_explicit_thermal` then

- In `advance` before `make_explicit_thermal`, we compute the coefficients for $\nabla \cdot (k_{\rm th}/c_p)\nabla h + \cdots$ using $(\rho, T, X)^{n+1}$.

end if

- In `make_S`, we compute thermodynamic variables using $(\rho, T, X)^{n+1}$.

Step 11. *Update the velocity.*

Step 12. *Compute a new* Δt.

CHAPTER 16

Networks

Introduction to MAESTRO Networks

MAESTRO models multiple species, described by the mass density of the fluid, ρ, and the mass fraction of the species, $X_k \equiv \rho_k/\rho$, where ρ_k is the mass density of species k. All MAESTRO problems, regardless of whether they model reactions, need a network. In its most basic form, the network supplies the properties of the species (atomic mass, atomic number) that are interpreted by the equation of state to compute.

Additional networks will be made available in the Microphysics repo[1]. These will have interfaces for both MAESTRO and CASTRO.

Notes of Specific Networks

`general_null`

This is a 'null' network – i.e. no burning, just define the properties of the species for thermodynamics. The twist is that you can create an inputs file to define what species you want to carry. For example, the extern/networks/null/ network defines C12, O16, and Mg24. To replicate this in `general_null`, we have the file `ignition.net` with contents:

```
# name        short name    aion     zion
 carbon-12       C12          12.0     6.0
 oxygen-16       O16          16.0     8.0
 magnesium-24    Mg24         24.0    12.0
```

[1] https://github.com/starkiller-astro/Microphysics

To use this set of species in your problem, you would set:

```
NETWORK_DIR := extern/networks/general_null
GENERAL_NET_INPUTS := ignition.net
```

It is assumed that the *.net files live in `extern/networks/general_null/`

Then at compile time, the network.f90 is created using these species and compiled. (For the curious, the rule to build `network.f90` lives in `extern/networks/general_null/GPackage.mak`)

`ignition_chamulak`

This network was introduced in our paper on convection in white dwarfs as a model of Type Ia supernovae [38]. It models carbon burning in a regime appropriate for a simmering white dwarf, and captures the effects of a much larger network by setting the ash state and energetics to the values suggested in [17].

`ignition_simple`

This is the original network used in our white dwarf convection studies [37]. It includes a single reaction, $^{12}\mathrm{C}(^{12}\mathrm{C},\gamma)^{24}\mathrm{Mg}$, using the rate from Caughlin and Fowler [16].

`rprox`

This is a network introduced in a paper modeling mixed H/He X-ray bursts [26]. `rprox` that has 10 species approximates hit CNO burning, triple-α, and rp-process breakout up through $^{56}\mathrm{Ni}$. Updated rates from ReacLib [18] are used. The overall ideas in this network are based on Appendix C of [35].

CHAPTER 17

MESA Microphysics

Introduction

Modules for Experiments in Stellar Astrophysics (MESA) is a collection of open source, robust and efficient modules developed for a wide range of applications in computational stellar astrophysics. The modules include macroscopic and microscopic physics such as nuclear reaction rates, opacity, equation of state, element diffusion data and atmosphere boundary conditions in addition to numerical routines. This document describes how to link the MESA modules to an outside program such as MAESTRO[1].

Setting Up MESA

To install MESA, download the source code from the MESA website, `http://mesa.sourceforge.net` . Check the website for the latest version, which is 5271 when the last upgrades to this document were added. The codes and algorithms discussed here were created when the current version of MESA was version 4088 and have been modified accordingly to keep up with the quickly channging source code. It might be possible to use older versions of MESA with the information in this document, although there are minor changes and upgrades between MESA versions that may cause the routines and algorithms discussed here to crash. These notes are aimed at working with version 4942, to download the source code associated with this version, issue the following command:

```
svn co -r 4942 svn://svn.code.sf.net/p/mesa/code/trunk mesa
```

This will download the source into a directory called `mesa` located in the directory where this command was run. This step might take some time ($\sim$1-3 min depending on the internet connection) possibly because the final `mesa` directory will take up roughly 1.8 GB.

[1] This interface work was done by Ryan Orvedahl—questions (and credit) should be directed to him

Rich Townsend has developed a Software Development Kit (SDK) to help install MESA called MESASDK. The SDK will install certain software packages that MESA uses and it will also change the PATH environment variable such that these programs become the default. A tarball of the SDK software and indepth instructions for its use are available on Townsend's website:

```
http://www.astro.wisc.edu/~townsend/static.php?ref=mesasdk
```

The instructions in this document ***do not*** use the SDK in order to avoid automatically changing the PATH variable.

A quick outline for compiling MESA for use with MAESTRO is shown below. Detailed instructions are discussed in Section 17.2.1.

1. Set environment variables, MESA_DIR and OMP_NUM_THREADS
2. Change name of makefile
3. Makefile: choose compiler, comment out second instance of CC = gcc, turn off USE_PGSTAR and LOAD_PGPLOT
4. Modify the const/public/const_def.f to fix the omp_get_max_threads() line
5. Run install script
6. Turn off OMP
7. Recompile using the compile-all-mods.sh script

Setup MESA Without MESASDK

Environment Variables (NOT using SDK)

A few environment variables need to be set (commands are shown for the BASH shell):

1. Set location of mesa: export MESA_DIR=<absolute-path-to-mesa-directory>
2. Set OpenMP number of threads: export OMP_NUM_THREADS=n, a suggested value is the number of cores in the computer

Modify the Makefile (NOT using SDK)

The main makefile that contains compiler information including flags is located in the mesa/utils directory and is called makefile_header. All MESA modules have a makefile that includes this main makefile. The file that needs to be edited is named makefile_header_non_mesasdk and so the name must be changed in order for the edits to take effect. This can be done as follows:

```
cp makefile_header_non_mesasdk makefile_header
```

All of the changes listed below apply to variables that already exist. There is no need to explicitly add new code unless you intend to compile MESA with something other than gfortran or ifort. If you want to compile using something different, then the only code that needs to be explicitly added are the definitions of the compiler flags.

Open the `makefile_header` file and change the following:

1. Compilers: First change the compilers labeled `CC` (C compiler) and `FC` (Fortran compiler) to the compilers of your choice (MESA defaults are `gcc` and `ifort`).
2. Remove (or comment out) the second instance of `CC` located directly under the `SPECIAL_FC_FLAGS` variable
3. `UTILS_ISNAN`: Leave this as `utils_isnan_okay`, if MESA complains upon compiling the `mesa/utils` module, then change it to `utils_isnan_nope`.
4. `BLAS` and `LAPACK`: Make sure the location of `LAPACK` and `BLAS` (provided with the MESA source tree) are set using the following definitions (these should be the default settings):

```
LOAD_LAPACK = -lmesalapack
WHICH_BLAS = USE_SRCS
LOAD_BLAS = -lmesablas
MKL_INCLUDE =
```

5. `PGPLOT`: Set the variable `USE_PGSTAR` to `NO` and set the variable `LOAD_PGPLOT` to be empty in the if-construct corresponding to the compiler that was chosen. The default for `gcc` and `ifort` is `YES`, but for any other compiler the default is `NO`. `PGPLOT` controls the pop up images that are used in `mesa/star` (1D stellar evolution code).
6. `USE_SE`: Make sure the variable `USE_SE` is set to `NO` and both `LOAD_SE` and `INCLUDE_SE` are empty. These also need to be set in the correct if-construct according to which compiler was chosen. `SE` is a library of formats used for writing/reading data in stellar evolution codes.
7. Compiler Flags: Find the if-construct corresponding to the chosen compiler. This is where compiler options are defined. It is not necessary to edit any of these if you are using `gfortran` or `ifort`. These are the variables to edit if you wish to add more (or less) debugging flags for example. If using a different compiler, the variables in the last `else` statement (which are all empty except for `FCbasic = UNKNOWN COMPILER`) are what must be edited to hold the compiler flag definitions.

   Installing MESA without OpenMP will result in a failed install. MESA is capable of being compiled in serial, but it must happen after the initial install.
8. Edit 1 source file: An undefined reference to `omp_get_max_threads` occurs in the `const_def.f` file located in `const/public/const_def.f` when running the code in serial.

   In the `const_def.f` file, around line ∼168, in the `do_const_init` subroutine, there is a line that reads:

```
omp_num_threads = omp_get_max_threads()
```

   Our first change is to declare a temporary character variable in the (`do_const_init` subroutine):

```
character(len=16) :: temp_max_threads
```

   Then we comment out the above line (`omp_num_threads = ...`) and add the following (all in the `do_const_init` subroutine):

```
!omp_num_threads = omp_get_max_threads()
call get_environment_variable("OMP_NUM_THREADS", temp_max_threads)
```

```
read(temp_max_threads, *) omp_num_threads
```

At this point, MESA is ready to be installed, see Section 17.2.2.

Install MESA

If the environment variables were added to the `.bashrc` (or equivalent) file, do not forget to source that file before continuing. Now MESA is ready to be installed. Change directories to the `mesa` directory. Run the `install` script by typing `./install`. The permissions of the `install` script may need to be changed in order for it to be executable. This will untar many data tables used by the equation of state, network and chemistry modules. It will also compile and export all modules.

Following a successful execution of the `./install` script, MESA has been installed and the following has completed:

- Data tables have been untarred
- Linear Algebra packages have been compiled and linked
- All modules have been compiled (in parallel)
- All modules and libraries have been exported to the `mesa/include` and `mesa/lib` directories

If you want to run MESA in parallel, nothing further needs to be done. If you want to make sure all modules are compiled in serial, see Section 17.3, which addresses how to compile the modules.

Compile Modules Without OpenMP

In the `mesa/utils` directory, open the `makefile_header` file. Change the `FCopenmp` flag (`gfortran` default = `-fopenmp`) to the appropriate value so as to disable OpenMP, e.g. with `gfortran` `FCopenmp` should be empty. Now all modules must be recompiled. This is most easily accomplished with the scripts described in Section 17.7.

Anytime a change is made to a makefile or any source code, the module must be recompiled, which is a three step process involving the `clean`, `mk` and `export` scripts provided in each module directory. The first step is to clean out any old files using the `./clean` script. Step two is to make the module using the `./mk` script. The last step is to export the `.mod` files and the library to the right directories using the `./export` script. Section 17.7 discusses a few simple scripts that are meant to help with compiling the modules.

Optional: Turn Off `mesa/star` OpenMP

There are four other places to turn off OpenMP, but they are all contained in the `mesa/star` directory, which is the 1-D stellar evolution code. If only using a few MESA modules that do not include the `mesa/star` stellar evolution code, then these last few places do not need to be modified in any way.

To find where all to set the `use_omp` flag to False, execute the following command:

```
find $(MESA_DIR) -name "*.f" -exec grep -Hn --color use_omp {} \;
```

This will return fortran files located in the mesa/star/private directory.

Compiling MAESTRO to Use MESA

Compiling code that depends on MESA modules involves linking the code with the .mod files and libraries that are generated when MESA is compiled. After the export script has been run in all compiled directories, the .mod files and other supporting routines are copied to the mesa/include directory while the libraries are copied to the mesa/lib directory.

The MESA libraries have the name lib(module-name).a, e.g. libeos.a or libnet.a. The libraries need to be linked using the -L flag with the mesa/lib directory and a list of which libraries to load, e.g. -L$(MESA_DIR)/lib -leos -lnet -lnum. MESA .mod files are linked to the code using the -I flag and the mesa/include directory, e.g. -I$(MESA_DIR)/include.

The order in which the libraries appear is important. If compilation errors related to undefined references to a particular MESA module such as eos_lib occur, check to make sure that the libraries are in the correct order. Of course it is also possible that an undefined reference really does exist.

The wrapper routines that allow MAESTRO to use MESA modules are located in the AstroDev/EOS/MESA, AstroDev/networks/MESA and AstroDev/networks/MESA_SDC directories. In each directory there is a GPackage.mak file that specifies the proper -I and -L compiler flags along with what source files to include in the build. To use the EOS, open the GNUmakefile in the MAESTRO problem directory and add the following lines:

```
EOS_TOP_DIR := $(ASTRODEV_DIR)/EOS
EOS_DIR := MESA
```

To use the Strang-Split version of the network, add the following lines:

```
NETWORK_TOP_DIR := $(ASTRODEV_DIR)/networks
NETWORK_DIR := MESA
GENERAL_NET_INPUTS := $(NETWORK_TOP_DIR)/$(NETWORK_DIR)/Net-Files/net_input.net
```

To use the Spectral Deferred Corrections (SDC) version of the network add these lines instead:

```
SDC := t

NETWORK_TOP_DIR := $(ASTRODEV_DIR)/networks
NETWORK_DIR := MESA_SDC
GENERAL_NET_INPUTS := $(NETWORK_TOP_DIR)/$(NETWORK_DIR)/Net-Files/net_input.net
```

The GENERAL_NET_INPUTS variable holds the path to the net_input.net file, which does not need to be kept in the Net-Files directory. The net_input.net file should have the following format:

```
# name              short-name    aion   zion
hydrogen-1             H1          1.0    1.0
helium-4               He4         4.0    2.0
carbon-12              C12         12.0   6.0
oxygen-16              O16         16.0   8.0
neon-20                Ne20        20.0   10.0
sodium-23              Na23        23.0   11.0
```

```
magnesium-23          Mg23      23.0  12.0
magnesium-24          Mg24      24.0  12.0
silicon-28            Si28      28.0  14.0
```

Using MESA Modules

This section describes what needs to take place in order to use MESA modules such as setting up the module, invoking certain routines and shutting down the module. This discussion will only include information pertaining to the network and equation of state modules.

The routines that carry out these processes (setting up EOS, setting up isotope map, shutdown routines, etc.) ***have already been written*** and live in `AstroDev/networks/MESA`, `AstroDev/networks/MESA_SDC` and `AstroDev/EOS/MESA`. This section outlines the necessary steps taken by those routines in order to set up the EOS and network.

Equation of State

Set Up

There are several steps to setting up the equation of state:

1. Constant Initialization: This is done by calling the `const_init` subroutine
2. Data Directory: Define where the `mesa/data` directory is: This can be done by using the `MESA_DIR` environment variable that was set at installation
3. Chem Initialization: This is done by calling the `chem_init` subroutine: `call chem_init(data_dir, 'isotopes.data_real', ierr)`. The second argument can also be `'isotopes.data_approx'`. The "real" file contains actual values for atomic quantities such as atomic weight. The "approx" file contains approximate values for these quantities, e.g. the mass of the proton and neutron are both equal to 1.0. The `ierr` argument is an integer that holds the exit status of the subroutine, non-zero means there was an error
4. EOS Initialization: This is done by calling the `eos_init` subroutine: `call eos_init(data_dir, eos_file_prefix, use_cache, ierr)`. `eos_file_prefix` is a character variable and should be declared and set equal to 'mesa'. `use_cache` is a logical variable that also needs to be declared and set equal to `.true.`
5. Handle: Obtain a handle associated with the equation of state by calling the `alloc_eos_handle` function: `handle = alloc_eos_handle(ierr)`. `handle` should be declared as an integer

Calling the EOS

The call to the generic MESA equation of state looks like:

```
call eosDT_get(handle_eos, Z, X, abar, zbar, nspec, chem_id, net_iso, Xa, &
               rho, log10(rho), temp, log10(temp), res, d_dlnd, d_dlnT, ierr)
```

Where `Z` is the metallicity, `X` is the mass fraction of hydrogen, `abar` is the average atomic weight, `zbar` is the average atomic number, `nspec` is the number of species in the network, `Xa` is the array of mass fractions. See Section 17.5.2 for a complete description of `chem_id` and `net_iso`. The results of the EOS call are stored in the arrays `res`, `d_dlnd` and `d_dlnT`, which are all of size `num_eos_basic_results`. The `d_dlnd` and `d_dlnT` arrays are the derivatives of the `res` array with respect to density and temperature, respectively. The values held in the `res` array are shown in Table 17.1.

Output	Definition	Units
P_{gas}	gas pressure	ergs cm^{-3}
E	internal energy	ergs g^{-}1
S	entropy per gram	ergs g^{-1} K^{-1}
$dE/d\rho\|_T$		ergs cm^3 g^{-2}
C_V	specific heat at constant V $\equiv 1/\rho$	ergs g^{-1} K^{-1}
$dS/d\rho\|_T$		ergs cm^3 g^{-2} K^{-1}
$dS/dT\|_\rho$		ergs g^{-1} K^{-2}
χ_ρ	$\equiv d\ln P/d\ln\rho\|_T$	none
χ_T	$\equiv d\ln P/d\ln T\|_\rho$	none
C_P	specific heat at constant pressure	ergs g^{-1} K^{-1}
∇_{ad}	adiabatic T gradient with pressure	none
Γ_1	$\equiv d\ln P/d\ln\rho\|_S$	none
Γ_3	$\equiv d\ln T/d\ln\rho\|_S + 1$	none
η	ratio of electron chemical potential to $k_B T$	none
μ	mean molecular weight per gas particle	none
$1/\mu_e$	mean number of free electrons per nucleon	none

Table 17.1: `eos` output quantities and units

It is possible to call the HELM equation of state directly. This is done as follows:

```
call eos_get_helm_results(handle_eos, Z, X, abar, zbar, dens, log10(dens), &
                          temp, log10(temp), helm_res, ierr)
```

The `helm_res` array is of size `num_helm_results` and holds all of the output values from the call, including all derivatives. There are slight differences between the generic EOS call and the HELM EOS call: HELM does not need the mass fractions or the maps between MESA isotope indices and the mass fraction array indices (`chem_id` and `net_iso`, see Section 17.5.2). The HELM EOS returns 409 different quantities which include the values shown in Table 17.1 as well as quantities such as $dP/d(abar)$ and $dE/d(zbar)$.

Shutdown the EOS

There are two steps to shutting down the equation of state:

1. Free the handle: Call the `free_eos_handle` subroutine: `call free_eos_handle(handle_eos)`.

2. Shutdown: Call the `eos_shutdown` subroutine: `call eos_shutdown`

Mapping MAESTRO to MESA

MAESTRO and MESA refer to isotopes in a very similar way; using short names of the isotopes such as he4 and ni56. MAESTRO tends to use capital letters (He4 and Ni56) while MESA uses lowercase letters (he4 and ni56). A simple routine to convert from uppercase to lowercase will be needed.

In order to use MESA modules and obtain the correct result, it is important to make sure that MESA and MAESTRO agree on what isotope each entry in the mass fraction array refers to. This is done by using the `chem_id` and `net_iso` variables. Both variables are integer pointer arrays of size `num_chem_isos`, which is defined in one of the chemistry modules. `chem_id` maps from the index of an isotope in the current net to the MESA data table position that holds the properties of that isotope. `net_iso` is used to map from the MESA data table position to the index of an isotope in the current network.

For example, assume there are 3 isotopes in the current net: H1, Be7 and C12. The `chem_id` array should look like this: `chem_id(:) = -1, chem_id(1) = 2, chem_id(2) = 18, chem_id(3) = 38`, and the `net_iso` array should look like this: `net_iso(:) = -1, net_iso(2) = 1, net_iso(18) = 2, net_iso(38) = 3`. Where 2, 18 and 38 refer to the MESA data table location of H1, Be7 and C12, respectively.

Care should be taken when setting up these arrays in order to avoid disagreement between the MESA mass fraction array and the MAESTRO mass fraction array.

Network

There are currently two different algorithms that integrate the nuclear reaction network in MAESTRO; Strang Splitting and Spectral Deferred Corrections (SDC). The routines to go from MAESTRO to MESA using Strang are stored in `AstroDev/networks/MESA` while the routines that implement the SDC method are stored in `AstroDev/networks/MESA_SDC`.

The setup/shutdown procedures for both methods are very similar. The largest difference is that the Strang method uses MESA to obtain the right hand sides of the network equations *and* do the integration, while the SDC method uses MESA to obtain the right hand sides and uses MAESTRO to integrate the equations using `VODE`.

Set Up

There are significantly more steps involved when initializing the MESA network. Here is a brief outline of what happens:

1. Get the MESA and AstroDev directory locations
2. Convert the short species names to MESA format
3. Generate a file that will tell MESA which reactions to add based on the isotopes in the current network
4. Initialize various MESA modules
5. Setup the map between MAESTRO and MESA (see Section 17.5.2 for a more in depth explanation)
6. Initialize MESA `rates` and `net` modules

7. Allocate a network handle
8. Call the `net_start_def`, `read_net_file`, `net_ptr` and `net_finish_def` subroutines
9. Output a log file containing the species in the current net and which reactions were chosen by MESA based on those species
10. Set `which_rates_choice` and allocate `which_rates` array
11. Set entire array equal to `which_rates_choice` and call `net_set_which_rates`
12. Call `net_setup_tables`
13. Set which linear algebra solver to use
14. Set the ODE solver to use for the network integration (Strang-Split version only)

Calling the Network

In the case of Strang Splitting the call to the MESA burner looks like this:

```
call net_1_zone_burn( &
      handle, solver_choice, species, num_reactions, 0.0_dp_t, burn_tend, &
      xin, clip, num_times, times, log10Ts_f1, log10Rhos_f1, etas_f1, &
      dxdt_source_term, rate_factors, category_factors, std_reaction_Qs, &
      std_reaction_neuQs, screening_mode, theta_e_for_graboske_et_al, &
      h, max_step_size, max_steps, rtol, atol, itol, &
      decsol_choice, caller_id, burn_solout, iout, ending_x, &
      nfcn, njac, nstep, naccpt, nrejct, time_doing_net, ierr)
```

The inputs are density, temperature, how long to burn, starting mass fractions, various rates and parameters as well as some things to control the integration. The outputs are the energy (`burn_ergs`) in units of ergs/g, the ending mass fractions and various counters such as number of jacobian evaluations. This routine will calculate the right hand sides, pass them to a stiff ODE integrator and return the final results.

In the case of the SDC method, the call to the MESA routine that returns the instantaneous right hand sides looks like this:

```
call burn_derivs(nspec, t, xmass(1:nspec), ydot(1:nspec), lrpar, rpar, &
                 lipar, ipar, ierr)
```

The inputs are mass fraction, work arrays and the time. The output is the time derivative of the mass fraction. This routine is only responsible for calculating the right hand sides of the network equations, `VODE` is tasked with the integration in this algorithm.

Shutdown the Network

Shutting down the MESA network requires only one step:

1. Deallocate `chem_id`, `net_iso` and `which_rates`

Install MESA on Hopper

These notes are somewhat dated because they pertain only to MESA version 4088. In my experience, version 4088 is very easy to install and follows the same steps as outlined above. When using the scripts in AstroDev/MESA/networks/scripts, make sure the proper list of modules to compile (lines containing `do_one ....`) is used. Those lists are version dependant.

Once MESA was installed and working properly on a local machine (e.g. Bender), I tried installing it on Hopper, which is a Cray XE6 machine. On Bender, which is an Intel(R) Xeon(R) X5650 machine, the compiler was `gcc` version 4.7.0, on Hopper I tried compiling MESA with the Cray Compiler using version 8.0.5. The module that was loaded to enable this was `PrgEnv-cray`.

There are several steps needed to compile MAESTRO with MESA on Hopper:

1. Download the `MESA` source code (Section 17.2) and be sure to use the correct version: `svn co -r 4088 ...`
2. Edit the `makefile_header` in the `utils` directory
3. Edit the `build_and_test` script in the `utils` directory
4. Edit the `chem_lib.f` file in the `chem` directory
5. Edit the `mtx` makefile
6. Edit the makefiles in `screen`, `utils`, `num` and `interp_1d`
7. Edit the `install` script in the main MESA directory
8. Edit the `BoxLib/Tools/F_mk/GMakedefs.mak` makefile
9. Edit the `BoxLib/Tools/F_mk/comps/Linux_cray.mak` file
10. Install MESA
11. Compile MAESTRO

`makefile_header`

In following the setup outlined in this document, I changed the compilers, used the source version of `LAPACK` and `BLAS`, turned off `PGPLOT` and turned off the `SE` formatting library. I also added the appropriate compiler flags. The results of what I changed are shown below:

```
# Cray C-compiler
CC = cc

# Cray Fortran-Compiler
FC = ftn

# if you need special flags for the compiler, define them here:
SPECIAL_FC_FLAGS = -h mpi0 -target=linux -emf
# -h mpi0 disables mpi optimization
# -target=linux specifies the build machine as linux based
# -emf: the m flag says create NAME.mod files, the f flag says
#         convert those to name.mod files
```

```
# must explicity define all compilation flags
FCbasic = $(SPECIAL_FC_FLAGS)
FCimpno = -eI
FCchecks =
FCwarn = -m 3
FCfixed = -f fixed -N 132
FCfixed72 = -f fixed -N 72
FCfree = -f free
FCopt = -O 1
FCdebug = -g -O0 -R bps
FCstatic =

FC_fixed_preprocess = -eZ
FC_free_preprocess = -eZ

# Cray compilers have omp ON by default
FCopenmp =
# to disable omp:
#FCopenmp = -h noomp
```

`build_and_test` script

The `build_and_test` script in the `utils` directory is responsible for compiling the main module directory as well as compiling and running the test directory. We want to turn off the compiling and running of the test directory to ensure that MESA compiles successfully (it failed to install otherwise). A portion of the `build_and_test` script looks like this:

```
cd make
make
check_okay
cd ../test
./mk
check_okay
if [ -x test_quietly ]
then
        ./test_quietly
        check_okay
fi
./ck >& diff.txt
```

we wish to change it to this:

```
cd make
make
check_okay
#cd ../test
#./mk
check_okay
if [ -x test_quietly ]
then
#        ./test_quietly
```

```
        check_okay
fi
#./ck >& diff.txt
```

This disables the compiling and running of the test directory.

`chem_lib.f` Source File

Open the `chem_lib.f` file located in `chem/public`. In the `generate_nuclide_set` subroutine, the implied do loop must be removed and replaced with an actual do loop. For an unknown reason, the implied do loop was causing a Segmentation Fault on Hopper with the Cray compiler. The implied do loop:

```
set = [(nuclide_set(names(i), i), i=1, size(names))]
```

should be changed to an explicit do loop:

```
do i=1, size(names)
   set(i) = nuclide_set(names(i), i)
enddo
```

`mtx` makefile

In the `mtx/make/makefile`, there are a few compiler flags that were hard coded in. There is a `-w` flag in five places and a `-fno-common` flag in one place. The `-w` flag instances look like this:

```
%.o: $(MODULE_DIR)/lapack_src/%.f
      $(COMPILE_XTRA) -w $<

# must turn off optimization for dlamch or can get infinite loop!!!
dlamch.o: $(MODULE_DIR)/blas_src/dlamch.f
      $(COMPILE_XTRA_NO_OPT) -w $<
```

The `-w` should be deleted. There are three more instances immediately after these lines. The `-fno-common` flag appears here:

```
KLU_C = $(CC) -O3 -fno-common -fexceptions
KLU_I = -I$(KLU_DIR)
```

All three of the manually added flags (`-O3`, `-fno-common` and `-fexceptions`) should be deleted from the `KLU_C` line.

makefiles in `screen`, `num`, `utils` and `interp_1d`

When creating the library for these four modules, there is a `LIB_DEFS` variable in the makefile that holds the `*_def.o` file, e.g. `utils_def.o` or `num_def.o`. This file is never loaded into the library as seen below for the `utils` directory. Certain modules have a `*_def.f` file with both definitions and declarations of data that will be needed at runtime, so they must be included in the library. Other

modules have a `*_def.f` file that contains only definitions and no data declarations and therefor only needs to be included at compile time. The Cray compiler on Hopper for whatever reason complained that there were undefined references to certain `*_def.o` files. The undefined references only occurred for modules whose `*_def.f` file did not need to be included in the library. The fix was to manually include the `*_def.o` file in the library for all of the necessary module directories. The `utils` directory is shown here:

```
LIB = libutils.a
LIB_DEFS = utils_def.o
LIB_OBJS = $(UTILS_ISNAN).o utils_nan.o utils_dict.o utils_lib.o

$(LIB) : $(LIB_DEFS) $(LIB_OBJS)
      $(LIB_TOOL) $(LIB) $(LIB_OBJS)
```

To include the `LIB_DEFS` file, change the above code to the following:

```
LIB = libutils.a
LIB_DEFS = utils_def.o
LIB_OBJS = $(UTILS_ISNAN).o utils_nan.o utils_dict.o utils_lib.o

$(LIB) : $(LIB_DEFS) $(LIB_OBJS)
      $(LIB_TOOL) $(LIB) $(LIB_DEFS) $(LIB_OBJS)
```

This must be done for each makefile in the following directories: `screen/make`, `num/make`, `utils/make` and `interp_1d/make`.

`install` script

Certain modules that are unneccessary for use with MAESTRO do not need to be installed (and also cause some sort of installation error). In the main MESA directory, edit the `install` script and comment out the following four lines: `do_one sample`, `do_one star`, `do_one adipls` and `do_one astero`.

BoxLib Linux_cray.mak file

Two flags used in the compilation of MESA with the Cray compiler are important; the `-e m` or `-em` flag and the `-e f` or `-ef` flag. The `-em` flag tells the compiler to create `.mod` files of the form `NAME.mod`. The `-e f` flag, which must be used with the `-em` flag, instead outputs `.mod` files of the form `name.mod`. This is important for MESA because the `export` scripts copy the various libraries and `.mod` files to the correct directories. The `export` script in the `const` directory for example looks like this:

```
cp make/const_lib.mod ../include
cp make/const_def.mod ../include

cp make/libconst.a ../lib
cd ../lib
ranlib libconst.a
```

If the `-ef` flag is not enabled, the `.mod` file will be `CONST_LIB.mod` and will not get copied to the `include` directory (the library is unaffected). The default for MAESTRO is to not use the `-ef` flag, but it does use

the -em flag. To make MAESTRO and MESA compatible, I added the -ef flag to the Linux_cray.mak file located in BoxLib/Tools/F_mk/comps. I also added a few more debugging flags:

```
FFLAGS   += -J $(mdir) -I $(mdir) -emf
F90FLAGS += -J $(mdir) -I $(mdir) -emf

ifdef NDEBUG
  FFLAGS   += -O 1
  F90FLAGS += -O 1
  CFLAGS   += -O 1
else
  FFLAGS   += -g -O0 -R bps
  F90FLAGS += -g -O0 -R bps
  CFLAGS   += -g -O0 -h bounds
endif
```

BoxLib GMakedefs.mak **makefile**

In MAESTRO, .mod files are located in the $(tdir)/m directory, where tdir = t/$(suf) and $(suf) was in this case Linux.Cray. This means the MAESTRO routines are expecting to find module files with the name t/Linux.Cray/m/network.mod. The addition of the -ef flag affects the entire name of the module so the Cray compiler was creating files with the name t/linux.cray/m/network.mod. To solve this, I modified what tdir is defined to be when using the Cray compiler. This was done in the BoxLib/Tools/F_mk/GMakedefs.mak makefile by using a simple if statement:

```
ifeq ($(COMP), Cray)
  tdir = t/$(shell echo $(suf) | tr A-Z a-z)
else
  tdir = t/$(suf)
endif
odir = $(tdir)/o
mdir = $(tdir)/m
```

The shell command echo $(suf) | tr A-Z a-z has the effect of converting $(suf) to lowercase.

Install MESA **and Compile** MAESTRO

At this point the install script in the top MESA directory can be executed to install MESA. This will compile the source code, generate the libraries and copy the necessary files to the correct directories. Once, MESA is installed, MAESTRO can be compiled and linked such that it can use MESA's EOS and/or network. Make sure the makefile variable COMP is set to Cray in the MAESTRO problem directory so the Cray compiler is used.

Compilation Scripts

Some useful scripts have been written in BASH to help with compiling MESA modules. They live in the AstroDev/networks/MESA/scripts directory. Currently, there are two scripts; one that compiles all modules (which unfortunately is version dependant), and one that compiles a single module.

The compile-one-module script can be run from any directory because the calling sequence involves the relative path to the MESA module:

```
compile-one-mod.sh ../../../mesa/net
```

The above command will compile the `net` module without needing to be in the MESA directory.

The compile-all-modules script will clean the module directory, compile the source code and export the libraries and `.mod` files to the appropriate directories for every module. To run this script, you must be in the MESA directory. The calling sequence would be something like:

```
cd $MESA_DIR
compile-all-mods.sh
```

CHAPTER 18

Notes on η_ρ

We carry around three different 1D radial quantities: $\eta_\rho^{\rm ec}$ (edge-centered), $\eta_\rho^{\rm cc}$ (cell-centered), and $\nabla\cdot(\eta_\rho \mathbf{e}_r)$ (cell-centered). These notes discuss when each of these is used, and how they are computed, in both plane-parallel and spherical.

The Mixing Term, η_ρ

The base state evolves in response to heating and mixing in the star. The density evolution is governed by

$$\frac{\partial \rho_0}{\partial t} = -\nabla\cdot(\rho_0 w_0 \mathbf{e}_r) - \nabla\cdot\left(\eta_\rho \mathbf{e}_r\right) \ , \tag{18.1}$$

with

$$\eta_\rho(r) = \overline{\left(\rho'\widetilde{\mathbf{U}}\cdot\mathbf{e}_r\right)} = \frac{1}{A(\Omega_H)}\int_{\Omega_H}(\rho'\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\,dA \ , \tag{18.2}$$

designed to keep the average value of the full density, ρ, over a layer of constant radius in the star equal to ρ_0. To complete the update of the base state, we need evolution equations for the pressure, p_0, and velocity, w_0. For spherical geometry, the derivation of w_0 constraint equation is shown in the multilevel paper, resulting in the following system

$$w_0 = \overline{w}_0 + \delta w_0 \tag{18.3}$$

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\overline{w}_0\right) = \overline{S} \tag{18.4}$$

$$\frac{\partial}{\partial r}\left[\frac{\overline{\Gamma}_1 p_0}{r^2}\frac{\partial}{\partial r}(r^2\delta w_0)\right] = -\frac{g}{r^2}\frac{\partial(r^2\eta_\rho)}{\partial r} - \frac{4(\overline{w}_0+\delta w_0)\rho_0 g}{r} - 4\pi G\rho_0\eta_\rho \tag{18.5}$$

In paper III, we introduced a mixing term, η_ρ, to the density evolution equation (eq. [18.1]), with the objective of keeping the base state density equal to the average of the density over a layer. For a spherical base state, it is best to define the average in terms of spherical coordinates,

$$\overline{q} = \frac{1}{4\pi}\int_{\Omega_H} q(r,\theta,\phi)\, d\Omega \tag{18.6}$$

where $\int_{\Omega_H} d\Omega = 4\pi$ represents the integral over the spherical θ and ϕ angles at constant radius.

Recall from Paper III that if initially $\overline{\rho'} = 0$, there is no guarantee that $\overline{\rho'} = 0$ will hold at later time. To see this, recall equation (2.55), written in a slightly different form:

$$\frac{\partial \rho'}{\partial t} + \nabla\cdot\left(\rho'\mathbf{U}\right) = -\nabla\cdot\left(\rho_0\widetilde{\mathbf{U}}\right). \tag{18.7}$$

We integrate this over a spherical shell of thickness $2h$ at radius r_0, i.e., $\Omega_H \times (r_0-h, r_0+h)$, and normalize by the integration volume, which is $\sim 4\pi r_0^2 2h$ for small h, to obtain:

$$\begin{aligned}
\frac{1}{4\pi r_0^2 2h}\int_{r_0-h}^{r_0+h} r^2 dr \int_{\Omega_H}\left[\frac{\partial\rho'}{\partial t} + \nabla\cdot\left(\rho'\mathbf{U}\right)\right] d\Omega &= -\frac{1}{4\pi r_0^2 2h}\int_{r_0-h}^{r_0+h} r^2 dr \int_{\Omega_H}\nabla\cdot\left(\rho_0\widetilde{\mathbf{U}}\right) d\Omega \\
&= -\frac{1}{4\pi r_0^2 2h}\int_{\Omega_H}\left[\rho_0\left(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r\right)\right] r^2 d\Omega\Big|_{r_0-h}^{r_0+h} \\
&= 0,
\end{aligned} \tag{18.8}$$

where we have used the divergence theorem in spherical coordinates to transform the volume integral on the right hand side into an area integral over Ω_H. We see that the right hand side disappears since $\int_{\Omega_H}\rho_0(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)d\Omega = 0$, which follows from the definition of $\widetilde{\mathbf{U}}$. Now, expanding the remaining terms and taking the limit as $h\to 0$, we can write

$$\begin{aligned}
0 &= \lim_{h\to 0}\frac{1}{4\pi r_0^2 2h}\int_{r_0-h}^{r_0+h} r^2 dr\int_{\Omega_H}\left[\frac{\partial\rho'}{\partial t} + \nabla\cdot(\rho'\mathbf{U})\right] d\Omega \\
&= \frac{\partial}{\partial t}\left(\lim_{h\to 0}\frac{1}{4\pi r_0^2}\frac{1}{2h}\int_{r_0-h}^{r_0+h} r^2 dr\int_{\Omega_H}\rho' d\Omega\right) + \lim_{h\to 0}\left[\frac{1}{4\pi r_0^2 2h}\int_{r_0-h}^{r_0+h} r^2 dr\int_{\Omega_H}\nabla\cdot(\rho'\mathbf{U})d\Omega\right] \\
&= \frac{\partial}{\partial t}\left(\frac{1}{4\pi}\int_{\Omega_H}\rho' d\Omega\right) + \lim_{h\to 0}\left\{\frac{1}{r_0^2 2h}\left[\frac{1}{4\pi}\int_{\Omega_H}\rho'(\mathbf{U}\cdot\mathbf{e}_r)d\Omega\right] r^2\Big|_{r_0-h}^{r_0+h}\right\} \\
&= \frac{\partial}{\partial t}\overline{\rho'} + \lim_{h\to 0}\left\{\frac{1}{r_0^2 2h}\left[\overline{\rho'(\mathbf{U}\cdot\mathbf{e}_r)}\right] r^2\Big|_{r_0-h}^{r_0+h}\right\} \\
&= \frac{\partial}{\partial t}\overline{\rho'} + \lim_{h\to 0}\frac{1}{r_0^2 2h}\int_{r_0-h}^{r_0+h}\nabla\cdot\left[\overline{\rho'(\mathbf{U}\cdot\mathbf{e}_r)}\mathbf{e}_r\right] r^2 dr \\
&= \frac{\partial}{\partial t}\overline{\rho'} + \nabla\cdot\left[\overline{\rho'(\mathbf{U}\cdot\mathbf{e}_r)}\mathbf{e}_r\right]
\end{aligned} \tag{18.9}$$

again using the divergence theorem, extracting the time derivative from the spatial integral, and switching the order of operations as appropriate.

In short,

$$\frac{\partial}{\partial t}\overline{\rho'} = -\nabla\cdot\left[\overline{\rho'\left(\mathbf{U}\cdot\mathbf{e}_r\right)}\mathbf{e}_r\right] = -\nabla\cdot\left(\eta_\rho\mathbf{e}_r\right), \tag{18.10}$$

and thus, $\eta_\rho = \overline{(\rho' \mathbf{U} \cdot \mathbf{e}_r)}$.

We need both η_ρ alone and its divergence for the various terms in the construction of w_0 and the correction to ρ_0. The quantity η_ρ is edge-centered on our grid, and for Cartesian geometries, we constructed it by averaging the appropriate fluxes through the grid boundaries. For a spherical base state, this does not work, since the spherical shells do not line up with the Cartesian grid boundaries.

Therefore, we take a different approach. We compute η_ρ by constructing the quantity $\rho' \widetilde{\mathbf{U}} \cdot \mathbf{e}_r$ in each cell, and then use our average routine to construct a 1-d, cell-centered $\eta_{\rho,r}$ (this is essentially numerically solving the integral in Eq. [18.2]). The edge-centered values of η_ρ, $\eta_{\rho,r+1/2}$ are then constructed by simple averaging:

should we be doing a volume-weighted average?

$$\eta_{\rho,r+1/2} = \frac{\eta_{\rho,r} + \eta_{\rho,r+1}}{2} \quad . \tag{18.11}$$

Instead of differencing $\eta_{\rho,r+1/2}$ to construct the divergence, we instead use equation (18.10) directly, by writing:

$$\left[\nabla \cdot (\eta_\rho \mathbf{e}_r)\right]^{n+1/2} = -\frac{\overline{\rho'^{n+1}} - \overline{\rho'^n}}{\Delta t} = -\frac{\overline{\rho'^{n+1}}}{\Delta t} \quad , \tag{18.12}$$

where we have made use of the fact that $\overline{\rho'^n} = 0$ by construction.

η Flow Chart

1. Enter `advance_timestep` with $[\eta_\rho^{\rm ec}, \eta_\rho^{\rm cc}]^{n-1/2}$.
2. Call `make_w0`. The spherical version uses uses $\eta_\rho^{{\rm ec},n-1/2}$ and $\eta_\rho^{{\rm cc},n-1/2}$.
3. Call `density_advance`. The plane-parallel version computes $\eta_\rho^{{\rm flux},n+1/2,*}$.
4. Call `make_etarho` to compute $[\eta_\rho^{\rm ec}, \eta_\rho^{\rm cc}]^{n+1/2,*}$. The plane-parallel version uses $\eta_\rho^{{\rm flux},n+1/2,*}$.
5. Call `make_psi`. The plane-parallel version uses $\eta_\rho^{{\rm cc},n+1/2,*}$.
6. Call `make_w0`. The spherical version uses uses $\eta_\rho^{{\rm ec},n+1/2,*}$ and $\eta_\rho^{{\rm cc},n+1/2,*}$.
7. Call `density_advance`. The plane-parallel version computes $\eta_\rho^{{\rm flux},n+1/2}$.
8. Call `make_etarho` to compute $[\eta_\rho^{\rm ec}, \eta_\rho^{\rm cc}]^{n+1/2}$. The plane-parallel version uses $\eta_\rho^{{\rm flux},n+1/2}$.
9. Call `make_psi`. The plane-parallel version uses $\eta_\rho^{{\rm cc},n+1/2}$.

Computing $\eta_\rho^{\rm ec}$ and $\eta_\rho^{\rm cc}$

This is done in `make_eta.f90`.

Plane-Parallel

We first compute a radial edge-centered multifab, $\eta_\rho^{\rm flux}$, using

$$\eta^{\rm flux}_{\rho,\mathbf{i}+1/2\mathbf{e}_r} = \left[\left(\widetilde{\mathbf{U}}^{n+1/2}_{\mathbf{i}+1/2\mathbf{e}_r}\cdot\mathbf{e}_r\right) + w^{n+1/2}_{0,r+1/2}\right]\rho^{n+1/2}_{\mathbf{i}+1/2\mathbf{e}_r} - w^{n+1/2}_{0,r+1/2}\rho^{n+1/2,\rm pred}_{0,r+1/2} \tag{18.13}$$

$\eta_\rho^{\rm ec}$ is the edge-centered "average" value of $\eta_\rho^{\rm flux}$,

$$\eta^{\rm ec}_{\rho,r+1/2} = \overline{\eta^{\rm flux}_{\rho,\mathbf{i}+1/2\mathbf{e}_r}} \tag{18.14}$$

$\eta_\rho^{\rm cc}$ is a cell-centered average of $\eta_\rho^{\rm ec}$,

$$\eta^{\rm cc}_{\rho,r} = \frac{\eta^{\rm ec}_{\rho,r+1/2}+\eta^{\rm ec}_{\rho,r-1/2}}{2}. \tag{18.15}$$

Spherical

First, construct $\eta_\rho^{\rm cart} = [\rho'(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)]^{n+1/2}$ using:

$$\left[\frac{\rho^n+\rho^{n+1}}{2} - \left(\frac{\rho_0^n+\rho_0^{n+1}}{2}\right)^{\rm cart}\right]\sum_d\left(\frac{\widetilde{\mathbf{U}}^{n+1/2}_{\mathbf{i}+1/2\mathbf{e}_d}\cdot\mathbf{e}_d + \widetilde{\mathbf{U}}^{n+1/2}_{\mathbf{i}-1/2\mathbf{e}_d}\cdot\mathbf{e}_d}{2}\right)n_d. \tag{18.16}$$

Then, $\eta_\rho^{\rm cc}$ is the cell-centered average of $\eta_\rho^{\rm cart}$,

$$\eta^{\rm cc}_\rho = \overline{\eta^{\rm cart}_\rho}. \tag{18.17}$$

On interior faces, $\eta_\rho^{\rm ec}$ is the average of $\eta_\rho^{\rm cc}$,

$$\eta^{\rm ec}_{\rho,r-1/2} = \frac{\eta^{\rm cc}_{\rho,r-1}+\eta^{\rm cc}_{\rho,r}}{2}. \tag{18.18}$$

At the upper and lower boundaries, we use

$$\eta^{\rm ec}_{\rho,-1/2} = 0, \tag{18.19}$$

$$\eta^{\rm ec}_{\rho,{\rm nr}-1/2} = \eta^{\rm cc}_{\rho,{\rm nr}-1}. \tag{18.20}$$

Using $\eta_\rho^{\rm ec}$

Plane-Parallel

NOT USED.

Spherical

In `make_w0`, $\eta_\rho^{\rm ec}$ is used in the construction of the RHS for the δw_0 equation.

Using η_ρ^{cc}

Plane-Parallel

In `make_psi`, $\psi = \eta_\rho^{cc} g$.

Spherical

In `make_w0`, η_ρ^{cc} is used in the construction of the RHS for the δw_0 equation.

CHAPTER 19

Notes on Prediction Types

Predicting interface states

The MAESTRO hyperbolic equations come in two forms: advective and conservative. The procedure for predicting interface states differs slightly depending on which form we are dealing with.

Advective form

Most of the time, we are dealing with equations in advective form. Here, a scalar, ϕ, obeys:

$$\frac{\partial \phi}{\partial t} = -\mathbf{U} \cdot \nabla \phi + f \tag{19.1}$$

where f is the force. This is the form that the perturbation equations take, as well as the equation for X_k itself.

A piecewise linear prediction of ϕ to the interface would appear as:

$$\begin{aligned}
\phi_{i+1/2,j}^{n+1/2} &= \phi_{i,j}^n + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x}\bigg|_{i,j} + \frac{\Delta t}{2}\frac{\partial \phi}{\partial t}\bigg|_{i,j} && (19.2)\\
&= \phi_{i,j}^n + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x}\bigg|_{i,j} + \frac{\Delta t}{2}\left(-u\frac{\partial \phi}{\partial x} - v\frac{\partial \phi}{\partial y} + f\right) && (19.3)\\
&= \phi_{i,j}^n + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\frac{\partial \phi}{\partial x} \underbrace{-\frac{\Delta t}{2}v\frac{\partial \phi}{\partial y}}_{\text{“transverse term”}} + \frac{\Delta t}{2}f && (19.4)
\end{aligned}$$

(see the Godunov notes section for more details). Here, the “transverse term” is accounted for in `make_edge_scal`. Any additional forces should be added to f. For the perturbation form of equations, we add additional advection-like terms to f by calling `modify_scal_force`. This will be noted below.

Conservative form

A conservative equation takes the form:

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot (\phi \mathbf{U}) + f \tag{19.5}$$

Now a piecewise linear prediction of ϕ to the interface is

$$\begin{aligned}
\phi^{n+1/2}_{i+1/2,j} &= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x}\bigg|_{i,j} + \frac{\Delta t}{2}\frac{\partial \phi}{\partial t}\bigg|_{i,j} && (19.6)\\
&= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x}\bigg|_{i,j} + \frac{\Delta t}{2}\left(-\frac{\partial(\phi u)}{\partial x} - \frac{\partial(\phi v)}{\partial y} + f\right) && (19.7)\\
&= \phi^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\frac{\partial \phi}{\partial x} \underbrace{-\frac{\Delta t}{2}\phi\frac{\partial u}{\partial x}}_{\text{“non-advective term”}} \underbrace{-\frac{\Delta t}{2}\frac{\partial(\phi v)}{\partial y}}_{\text{“transverse term”}} + \frac{\Delta t}{2} f && (19.8)
\end{aligned}$$

Here the “transverse term” is now in conservative form, and an additional term, the non-advective portion of the x-flux (for the x-prediction) appears. Both of the underbraced terms are accounted for in `make_edge_scal` automatically when we call it with `is_conservative = .true.`.

Density Evolution

Basic equations

The full density evolution equation is

$$\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla \cdot (\rho \mathbf{U}) \\
&= -\mathbf{U} \cdot \nabla \rho - \rho \nabla \cdot \mathbf{U}\,. && (19.9)
\end{aligned}$$

The species are evolved according to

$$\begin{aligned}
\frac{\partial (\rho X_k)}{\partial t} &= -\nabla \cdot (\rho \mathbf{U} X_k) + \rho \dot{\omega}_k \\
&= -\mathbf{U} \cdot \nabla (\rho X_k) - \rho X_k \nabla \cdot \mathbf{U} + \rho \dot{\omega}_k\,. && (19.10)
\end{aligned}$$

In practice, only the species evolution equation is evolved, and the total density is set as

$$\rho = \sum_k (\rho X_k) \tag{19.11}$$

To advance (ρX_k) to the next timestep, we need to predict a time-centered, interface state. Algebraically, there are multiple paths to get this interface state—we can predict (ρX) to the edges as a single quantity, or predict ρ and X separately (either in full or perturbation form). In the notes below, we use the subscript ‘`edge`’ to indicate what quantity was *predicted* to the edges. In MAESTRO, the different methods of computing (ρX) on edges are controlled by the `species_pred_type` parameter. The quantities predicted to edges and the resulting edge state are shown in the table 19.1

We note the labels `predict_rhoprime_and_X`, `predict_rhoX`, and `predict_rho_and_X` are provided by the `pred_parameters` module.

Table 19.1: Summary of species edge state construction

`species_pred_type`	quantities predicted in `make_edge_scal`	(ρX_k) edge state
1 / `predict_rhoprime_and_X`	$\rho'_{\rm edge}$, $(X_k)_{\rm edge}$	$\left(\rho_0^{n+1/2,\rm avg} + \rho'_{\rm edge}\right)(X_k)_{\rm edge}$
2 / `predict_rhoX`	$\sum(\rho X_k)_{\rm edge}$, $(\rho X_k)_{\rm edge}$	$(\rho X_k)_{\rm edge}$
3 / `predict_rho_and_X`	$\rho_{\rm edge}$, $(X_k)_{\rm edge}$	$\rho_{\rm edge}(X_k)_{\rm edge}$

Method 1: `species_pred_type` **=** `predict_rhoprime_and_X`

Here we wish to construct $(\rho_0^{n+1/2,\rm avg} + \rho'_{\rm edge})(X_k)_{\rm edge}$.

We predict both ρ' and ρ_0 to edges separately and later use them to reconstruct ρ at edges. The base state density evolution equation is

$$\frac{\partial \rho_0}{\partial t} = -\nabla \cdot (\rho_0 w_0 \mathbf{e}_r) = -w_0 \frac{\partial \rho_0}{\partial r} \underbrace{-\rho_0 \frac{\partial w_0}{\partial r}}_{\text{“}\rho_0 \text{ force”}}. \tag{19.12}$$

Subtract (19.12) from (19.9) and rearrange terms, noting that $\mathbf{U} = \widetilde{\mathbf{U}} + w_o \mathbf{e}_r$, to obtain the perturbational density equation,

$$\frac{\partial \rho'}{\partial t} = -\mathbf{U} \cdot \nabla \rho' \underbrace{-\rho' \nabla \cdot \mathbf{U} - \nabla \cdot (\rho_0 \widetilde{\mathbf{U}})}_{\rho' \text{ force}}. \tag{19.13}$$

We also need X_k at the edges. Here, we subtract $X_k \times$ Eq. 19.9 from Eq. 19.10 to obtain

$$\frac{\partial X_k}{\partial t} = -\mathbf{U} \cdot \nabla X_k + \dot{\omega}_k \tag{19.14}$$

When using Strang-splitting, we ignore the $\dot{\omega}_k$ source terms, and then the species equation is a pure advection equation with no force.

Predicting ρ' at edges

We define $\rho' = \rho^{(1)} - \rho_0^n$. Then we predict ρ' to edges using `make_edge_scal` in `density_advance` and the underbraced term in Eq. 19.13 as the forcing. This force is computed in `modify_scal_force`. This prediction is done in advective form.

Predicting ρ_0 at edges

There are two ways to predict ρ_0 at edges.

1. We call `make_edge_state_1d` using the underbraced term in (19.12) as the forcing. This gives us $\rho_0^{n+1/2,\rm pred}$. This term is used to advect ρ_0 in **Advect Base Density**. In plane-parallel geometries, we also use $\rho_0^{n+1/2,\rm pred}$ to compute η_ρ, which will be used to compute ψ.

2. We define $\rho_0^{n+1/2,\mathrm{avg}} = (\rho_0^n + \rho_0^{(2)})/2$. We compute $\rho_0^{(2)}$ from ρ_0^n using **Advect Base Density**, which advances equation (19.12) through Δt in time. The (2) in the superscript indicates that we have not called **Correct Base** yet, which computes ρ_0^{n+1} from $\rho_0^{(2)}$. We use $\rho_0^{(2)}$ rather than ρ_0^{n+1} to construct $\rho_0^{n+1/2,\mathrm{avg}}$ since ρ_0^{n+1} is not available yet. $\rho_0^{n+1/2,\mathrm{avg}}$ is used to construct ρ at edges from ρ' at edges, and this ρ at edges is used to compute fluxes for ρX_k.

We note that in essence these choices reflect a hyperbolic (1) vs. elliptic (2) approach. In MAESTRO, if we setup a problem with $\rho = \rho_0$ initially, and enforce a constraint $\nabla \cdot (\rho_0 \mathbf{U}) = 0$ (i.e. the anelastic constraint), then analytically, we should never generate a ρ'. To realize this behavior numerically, we use $\rho_0^{n+1/2,\mathrm{avg}}$ in the prediction of (ρX_k) on the edges to be consistent with the use of the average of ρ to the interfaces in the projection step at the end of the algorithm.

Computing ρ at edges

For the non-radial edges, we directly add $\rho_0^{n+1/2,\mathrm{avg}}$ to ρ' since $\rho_0^{n+1/2,\mathrm{avg}}$ is a cell-centered quantity. For the radial edges, we interpolate to obtain $\rho_0^{n+1/2,\mathrm{avg}}$ at radial edges before adding it to ρ'.

Predicting X_k at edges

Predicting X_k is straightforward. We convert the cell-centered (ρX_k) to X_k by dividing by ρ in each zone and then we just call `make_edge_scal` in `density_advance` on X_k. The force seen by `make_edge_scal` is 0. The prediction is done in advective form.

Method 2: `species_pred_type` = `predict_rhoX`

Here we wish to construct $(\rho X_k)_{\mathrm{edge}}$ by predicting (ρX_k) to the edges as a single quantity. We recall Eq. 19.10:

$$\frac{\partial(\rho X_k)}{\partial t} = -\nabla \cdot (\rho \mathbf{U} X_k) + \rho \dot{\omega}_k \,.$$

The edge state is created by calling `make_edge_scal` in `density_advance` with `is_conservative = .true.`. We do not consider the $\rho\dot{\omega}_k$ term in the forcing when Strang-splitting.

We note that even though it is not needed here, we still compute $\rho_{\mathrm{edge}} = \sum(\rho X_k)_{\mathrm{edge}}$ at the edges since certain enthalpy formulations need it.

Method 3: `species_pred_type` = `predict_rho_and_X`

Here we wish to construct $\rho_{\mathrm{edge}}(X_k)_{\mathrm{edge}}$ by predicting ρ and X_k to the edges separately.

Predicting X_k to the edges proceeds exactly as described in § 19.2.2.

Predicting the full ρ begins with Eq. 19.9:

$$\frac{\partial \rho}{\partial t} = -\mathbf{U} \cdot \nabla \rho \underbrace{-\rho \nabla \cdot \mathbf{U}}_{\text{“}\rho\text{ force”}} \,. \tag{19.15}$$

Using this, ρ is predicted to the edges using `make_edge_scal` in `density_advance`, with the underbraced force computed in `modify_scal_force` with `fullform = .true.`.

we need to switch this over to doing th conservative prediction

Advancing ρX_k

The evolution equation for ρX_k, ignoring the reaction terms that were already accounted for in `react_state`, and the associated discretization is

`species_pred_type = predict_rhoprime_and_X`:

$$\frac{\partial \rho X_k}{\partial t} = -\nabla \cdot \left\{ \left[\left({\rho_0}^{n+1/2,\mathrm{avg}} + \rho'_{\mathrm{edge}} \right) (X_k)_{\mathrm{edge}} \right] (\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r) \right\}. \tag{19.16}$$

`species_pred_type = predict_rhoX`:

$$\frac{\partial \rho X_k}{\partial t} = -\nabla \cdot \left\{ \left[(\rho X_k)_{\mathrm{edge}} \right] (\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r) \right\}. \tag{19.17}$$

`species_pred_type = predict_rho_and_X`:

$$\frac{\partial \rho X_k}{\partial t} = -\nabla \cdot \left\{ \left[\rho_{\mathrm{edge}} (X_k)_{\mathrm{edge}} \right] (\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r) \right\}. \tag{19.18}$$

Energy Evolution

Basic equations

MAESTRO solves an enthalpy equation. The full enthalpy equation is

$$\begin{aligned} \frac{\partial (\rho h)}{\partial t} &= -\nabla \cdot (\rho h \mathbf{U}) + \frac{D p_0}{D t} + \nabla \cdot k_{\mathrm{th}} \nabla T + \rho H_{\mathrm{nuc}} + \rho H_{\mathrm{ext}} \\ &= \underbrace{-\mathbf{U} \cdot \nabla (\rho h) - \rho h \nabla \cdot \mathbf{U}}_{-\nabla \cdot (\rho h \mathbf{U})} + \underbrace{\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r) \frac{\partial p_0}{\partial r}}_{\frac{D p_0}{D t}} + \nabla \cdot k_{\mathrm{th}} \nabla T + \rho H_{\mathrm{nuc}} + \rho H_{\mathrm{ext}}. \end{aligned} \tag{19.19}$$

Due to Strang-splitting of the reactions, the call to `react_state` has already been made. Hence, the goal is to compute an edge state enthalpy starting from $(\rho h)^{(1)}$ using an enthalpy equation that does not include the ρH_{nuc} and ρH_{ext} terms, where were already accounted for in `react_state`, so our equation becomes

$$\frac{\partial (\rho h)}{\partial t} = -\mathbf{U} \cdot \nabla (\rho h) - \rho h \nabla \cdot \mathbf{U} + \underbrace{\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r) \frac{\partial p_0}{\partial r} + \nabla \cdot k_{\mathrm{th}} \nabla T}_{\text{“}(\rho h) \text{ force”}} \tag{19.20}$$

We define the base state enthalpy evolution equation as

$$\begin{aligned}\frac{\partial(\rho h)_0}{\partial t} &= -\nabla\cdot[(\rho h)_0 w_0 \mathbf{e}_r] + \frac{D_0 p_0}{Dt} \\ &= -w_0\frac{\partial(\rho h)_0}{\partial r} - \underbrace{(\rho h)_0\frac{\partial w_0}{\partial r} + \psi}_{\text{“}(\rho h)_0\text{ force”}} .\end{aligned} \tag{19.21}$$

Perturbational enthalpy formulation

Subtracting (19.21) from (19.20) and rearranging terms gives the perturbational enthalpy equation

$$\begin{aligned}\frac{\partial(\rho h)'}{\partial t} &= -\nabla\cdot[(\rho h)'\mathbf{U}] - \nabla\cdot[(\rho h)_0\widetilde{\mathbf{U}}] + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla\cdot k_{\text{th}}\nabla T \\ &= -\mathbf{U}\cdot\nabla(\rho h)' \underbrace{-(\rho h)'\nabla\cdot\mathbf{U} - \nabla\cdot[(\rho h)_0\widetilde{\mathbf{U}}] + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla\cdot k_{\text{th}}\nabla T}_{\text{“}(\rho h)'\text{ force”}},\end{aligned} \tag{19.22}$$

Temperature formulation

Alternately, we can consider an temperature evolution equation, derived from enthalpy, as:

$$\frac{\partial T}{\partial t} = -\mathbf{U}\cdot\nabla T + \frac{1}{\rho c_p}\left\{(1-\rho h_p)\left[\psi + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r}\right] + \nabla\cdot k_{\text{th}}\nabla T - \sum_k \rho\xi_k\dot{\omega}_k + \rho H_{\text{nuc}} + \rho H_{\text{ext}}\right\}. \tag{19.23}$$

Again, we neglect the reaction terms, since that will be handled during the reaction step, so we can write this as:

$$\frac{\partial T}{\partial t} = -\mathbf{U}\cdot\nabla T + \underbrace{\frac{1}{\rho c_p}\left\{(1-\rho h_p)\left[\psi + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r}\right] + \nabla\cdot k_{\text{th}}\nabla T\right\}}_{\text{“}T\text{ force”}} . \tag{19.24}$$

Pure enthalpy formulation

A final alternative is to consider an evolution equation for h alone. This can be derived by expanding the derivative of (ρh) in Eq. 19.20 and subtracting off $h\times$ the continuity equation (Eq. 19.9):

$$\frac{\partial h}{\partial t} = -\mathbf{U}\cdot\nabla h + \underbrace{\frac{1}{\rho}\left\{\psi + (\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla\cdot k_{\text{th}}\nabla T\right\}}_{\text{“}h\text{ force”}} . \tag{19.25}$$

Prediction requirements

To update the enthalpy, we need to compute an interface state for (ρh). As with the species evolution, there are multiple quantities we can predict to the interfaces to form this state, controlled by

`enthalpy_pred_type`. A complexity of the enthalpy evolution is that the formation of this edge state will depend on `species_pred_type`.

The general procedure for making the (ρh) edge state is as follows:

1. predict $(rhoh)$, $(\rho h)'$, h, or T to the edges (depending on `enthalpy_pred_type`) using `make_edge_scal` and the forces identified in the appropriate evolution equation (Eqs. 19.22, 19.24, or 19.25 respectively).

   The appropriate forces are summaried in table 19.2.

2. if we predicted T, convert this predicted edge state to an intermediate "enthalpy" state (the quotes indicate that it may be perturbational or full enthalpy) by calling the EOS.

3. construct the final enthalpy edge state in `mkflux`. The precise construction depends on what species and enthalpy quantities are input to `mkflux`.

Finally, when MAESTRO is run with `use_tfromp = T`, the temperature is derived from the density, basestate pressure (p_0), and X_k. When run with reactions or external heating, `react_state` updates the temperature after the reaction/heating term is computed. In `use_tfromp = T` mode, the temperature will not see the heat release, since the enthalpy does not feed in. Only after the hydro update does the temperature gain the effects of the heat release due to the adjustment of the density (which in turn sees it through the velocity field and S). As a result, the `enthalpy_pred_types` that predict temperature to the interface (`predict_T_then_rhoprime` and `predict_T_then_h`) will not work. MAESTRO will abort if the code is run with this combination of parameters.

Table 19.3 gives a summary of the `enthalpy_pred_type` behavior.

Table 19.2: Forcing term into `make_edge_scal`

`enthalpy_pred_type`	advective force
0 / `predict_rhoh` (ρh)	$\left[\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla \cdot k_{\rm th} \nabla T\right]$
1 / `predict_rhohprime` $((\rho h)')$	$-(\rho h)' \nabla \cdot (\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r) - \nabla \cdot (\widetilde{\mathbf{U}}(\rho h)_0) + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla \cdot k_{\rm th} \nabla T$
2 / `predict_h` (h)	$\frac{1}{\rho}\left[\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r} + \nabla \cdot k_{\rm th} \nabla T\right]$
3 / `predict_T_then_rhohprime` (T)	$\frac{1}{\rho c_p}\left\{(1 - \rho h_p)\left[\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r}\right] + \nabla \cdot k_{\rm th} \nabla T\right\}$
4 / `predict_T_then_h` (T)	$\frac{1}{\rho c_p}\left\{(1 - \rho h_p)\left[\psi + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r}\right] + \nabla \cdot k_{\rm th} \nabla T\right\}$

Table 19.3: Summary of enthalpy edge state construction

species_pred_type	enthalpy_pred_type	cell-centered quantity predicted in make_edge_scal	intermediate "enthalpy" edge state	species quantity available in mkflux	final (ρh) edge state
1 / predict_rhoprime_and_X	0 / predict_rhoh	(ρh)	$(\rho h)_{\text{edge}}$	X_{edge}, ρ'_{edge}	$(\rho h)_{\text{edge}}$
1 / predict_rhoprime_and_X	1 / predict_rhohprime	$(\rho h)'$	$(\rho h)'_{\text{edge}}$	X_{edge}, ρ'_{edge}	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
1 / predict_rhoprime_and_X	2 / predict_h	h	h_{edge}	X_{edge}, ρ'_{edge}	$\left(\rho_0^{n+1/2,\text{avg}} + \rho'_{\text{edge}}\right) h_{\text{edge}}$
1 / predict_rhoprime_and_X	3 / predict_T_then_rhohprime	T	$(\rho h)'_{\text{edge}}$	X_{edge}, ρ'_{edge}	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
1 / predict_rhoprime_and_X	4 / predict_T_then_h	T	h_{edge}	X_{edge}, ρ'_{edge}	$\left(\rho_0^{n+1/2,\text{avg}} + \rho'_{\text{edge}}\right) h_{\text{edge}}$
2 / predict_rhoX	0 / predict_rhoh	(ρh)	$(\rho h)_{\text{edge}}$	$(\rho X)_{\text{edge}}$, $\sum(\rho X)_{\text{edge}}$	$(\rho h)_{\text{edge}}$
2 / predict_rhoX	1 / predict_rhohprime	$(\rho h)'$	$(\rho h)'_{\text{edge}}$	$(\rho X)_{\text{edge}}$, $\sum(\rho X)_{\text{edge}}$	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
2 / predict_rhoX	2 / predict_h	h	h_{edge}	$(\rho X)_{\text{edge}}$, $\sum(\rho X)_{\text{edge}}$	$\sum(\rho X)_{\text{edge}} h_{\text{edge}}$
2 / predict_rhoX	3 / predict_T_then_rhohprime	T	$(\rho h)'_{\text{edge}}$	$(\rho X)_{\text{edge}}$, $\sum(\rho X)_{\text{edge}}$	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
2 / predict_rhoX	4 / predict_T_then_h	T	h_{edge}	$(\rho X)_{\text{edge}}$, $\sum(\rho X)_{\text{edge}}$	$\sum(\rho X)_{\text{edge}} h_{\text{edge}}$
3 / predict_rho_and_X	0 / predict_rhoh	(ρh)	$(\rho h)_{\text{edge}}$	X_{edge}, ρ_{edge}	$(\rho h)_{\text{edge}}$
3 / predict_rho_and_X	1 / predict_rhohprime	$(\rho h)'$	$(\rho h)'_{\text{edge}}$	X_{edge}, ρ_{edge}	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
3 / predict_rho_and_X	2 / predict_h	h	h_{edge}	X_{edge}, ρ_{edge}	$\rho_{\text{edge}} h_{\text{edge}}$
3 / predict_rho_and_X	3 / predict_T_then_rhohprime	T	$(\rho h)'_{\text{edge}}$	X_{edge}, ρ_{edge}	$\left[(\rho h)_0^{n+1/2,\text{avg}} + (\rho h)'_{\text{edge}}\right]$
3 / predict_rho_and_X	4 / predict_T_then_h	T	h_{edge}	X_{edge}, ρ_{edge}	$\rho_{\text{edge}} h_{\text{edge}}$

Method 0: `enthalpy_pred_type` **=** `predict_rhoh`

Here we wish to construct $(\rho h)_{\rm edge}$ by predicting (ρh) to the edges directly. We use `make_edge_scal` with `is_conservative = .true.` on (ρh), with the underbraced term in Eq. 19.20 as the force (computed in `mkrhohforce`).

Method 1: `enthalpy_pred_type` **=** `predict_rhohprime`

Here we wish to construct $\left[(\rho h)_0^{n+1/2,\rm avg} + (\rho h)'_{\rm edge}\right]$ by predicting $(\rho h)'$ to the edges.

Predicting $(\rho h)'$ at edges

We define $(\rho h)' = (\rho h)^{(1)} - (\rho h)_0^n$. Then we predict $(\rho h)'$ to edges using `make_edge_scal` in `enthalpy_advance` and the underbraced term in (19.22) as the forcing (see also table 19.2 for the forcing term). The first two terms in $(\rho h)'$ force are computed in `modify_scal_force`, and the last two terms are accounted for in `mkrhohforce`. For spherical problems, we have found that a different representation of the pressure term in the $(\rho h)'$ force gives better results, namely:

$$(\widetilde{\mathbf{U}} \cdot \mathbf{e}_r)\frac{\partial p_0}{\partial r} \equiv \widetilde{\mathbf{U}} \cdot \nabla p_0 = \nabla \cdot (\widetilde{\mathbf{U}} p_0) - p_0 \nabla \cdot \widetilde{\mathbf{U}}. \tag{19.26}$$

Predicting $(\rho h)_0$ at edges

We use an analogous procedure described in Section 19.2.2 for computing $\rho_0^{n+1/2,\rm avg}$ to obtain $(\rho h)_0^{n+1/2,\rm avg}$, i.e., $(\rho h)_0^{n+1/2,\rm avg} = [(\rho h)_0^n + (\rho h)_0^{n+1}]/2$.

For spherical, however, instead of computing $(\rho h)_0$ on edges directly, we compute ρ_0 and h_0 separately at the edges, and multiply to get $(\rho h)_0$.

Computing ρh at edges

We use an analogous procedure described in Section 19.2.2 for computing ρ at edges to compute ρh at edges.

Method 2: `enthalpy_pred_type` **=** `predict_h`

Here, the construction of the interface state depends on what species quantities are present. In all cases, the enthalpy state is found by predicting h to the edges.

For `species_pred_types`: `predict_rhoprime_and_X`, we wish to construct $(\rho_0 + \rho'_{\rm edge})h_{\rm edge}$.

For `species_pred_types`: `predict_rho_and_X` or `predict_rhoX`, we wish to construct $\rho_{\rm edge} h_{\rm edge}$.

Predicting h at edges

We define $h = (\rho h)^{(1)}/\rho^{(1)}$. Then we predict h to edges using `make_edge_scal` in `enthalpy_advance` and the underbraced term in Eq. 19.25 as the forcing (see also table 19.2). This force is computed by `mkrhohforce` and then divided by ρ. Note: `mkrhoforce` knows about the different `enthalpy_pred_types` and computes the correct force for this type.

Computing ρh at edges

`species_pred_types`: `predict_rhoprime_and_X`:
We use the same procedure described in Section 19.2.2 for computing ρ_{edge} from ρ_0 and ρ'_{edge} and then multiply by h_{edge}.

`species_pred_types`: `predict_rhoX`:
We already have $\sum(\rho X_k)_{\text{edge}}$ and simply multiply by h_{edge}.

`species_pred_types`: `predict_rho_and_X`:
We already have ρ_{edge} and simply multiply by h_{edge}.

Method 3: `enthalpy_pred_type` = `predict_T_then_rhohprime`

Here we wish to construct $\left[(\rho h)_0 + (\rho h)'_{\text{edge}}\right]$ by predicting T to the edges and then converting this to $(\rho h)'_{\text{edge}}$ via the EOS.

Predicting T at edges

We predict T to edges using `make_edge_scal` in `enthalpy_advance` and the underbraced term in Eq. 19.24 as the forcing (see also table 19.2). This force is computed by `mktempforce`.

Converting T_{edge} to $(\rho h)'_{\text{edge}}$

We call the EOS in `makeHfromRhoT_edge` (called from `enthalpy_advance`) to convert from T_{edge} to $(\rho h)'_{\text{edge}}$. For the EOS call, we need X_{edge} and ρ_{edge}. This construction depends on `species_pred_type`, since the species edge states may differ between the various prediction types (see the "species quantity" column in table 19.3). The EOS inputs are constructed as:

After calling the EOS, the output of `makeHfromRhoT_edge` is $(\rho h)'_{\text{edge}}$.

Computing ρh at edges

The computation of the final (ρh) edge state is done identically as the `predict_rhohprime` version.

Table 19.4: EOS states in `makeHfromRhoT_edge`

`species_pred_type`	ρ edge state	X_k edge state
`predict_rhoprime_and_X`	$\rho_0^{n+1/2,\mathrm{avg}} + \rho'_{\mathrm{edge}}$	$(X_k)_{\mathrm{edge}}$
`predict_rhoX`	$\sum_k (\rho X_k)_{\mathrm{edge}}$	$(\rho X_k)_{\mathrm{edge}} / \sum_k (\rho X_k)_{\mathrm{edge}}$
`predict_rho_and_X`	ρ_{edge}	$(X_k)_{\mathrm{edge}}$

Method 4: `enthalpy_pred_type` **=** `predict_T_then_h`

Here, the construction of the interface state depends on what species quantities are present. In all cases, the enthalpy state is found by predicting T to the edges and then converting this to h_{edge} via the EOS.

For `species_pred_types`: `predict_rhoprime_and_X`, we wish to construct $(\rho_0 + \rho'_{\mathrm{edge}})h_{\mathrm{edge}}$.

For `species_pred_types`: `predict_rhoX`, we wish to construct $\sum(\rho X_k)_{\mathrm{edge}} h_{\mathrm{edge}}$.

For `species_pred_types`: `predict_rho_and_X`, we wish to construct $\rho_{\mathrm{edge}} h_{\mathrm{edge}}$.

Predicting T at edges

The prediction of T to the edges is done identically as the `predict_T_then_rhohprime` version.

Converting T_{edge} to h_{edge}

This is identical to the `predict_T_then_rhohprime` version, except that on output, we compute h_{edge}.

Computing ρh at edges

The computation of the final (ρh) edge state is done identically as the `predict_h` version.

Advancing ρh

We update the enthalpy analogously to the species update in Section 19.2.5. The forcing term does not include reaction source terms accounted for in **React State**, and is the same for all `enthalpy_pred_types`.

$$\frac{\partial(\rho h)}{\partial t} = -\nabla \cdot \left\{ \langle (\rho h) \rangle_{\mathrm{edge}} \left(\widetilde{\mathbf{U}} + w_0 \mathbf{e}_r \right) \right\} + (\widetilde{\mathbf{U}} \cdot \mathbf{e}_r) \frac{\partial p_0}{\partial r} + \psi \ . \tag{19.27}$$

where $\langle (\rho h) \rangle_{\mathrm{edge}}$ is the edge state for (ρh) computed as listed in the final column of table 19.3 for the given `enthalpy_pred_type` and `species_pred_type`.

Experience from `toy_convect`

Why is `toy_convect` Interesting?

The `toy_convect` problem consists of a carbon-oxygen white dwarf with an accreted layer of solar composition. There is a steep composition gradient between the white dwarf and the accreted layer. The convection that begins as a result of the accretion is extremely sensitive to the amount of mixing.

Initial Observations

With `use_tfromp = T` and `cflfac = 0.7` there is a large difference between `species_pred_type = 1` and `species_pred_type = 2,3` as seen in Figure 19.1. `species_pred_type = 1` shows quick heating (peak T vs. t) and there is ok agreement between `tfromh` and `tfromp`. `species_pred_type = 2,3` show cooling (peak T vs. t) and `tfromh` looks completely unphysical (see Figure 19.2). There are also strange filament type features in the momentum plots shown in Figure 19.3.

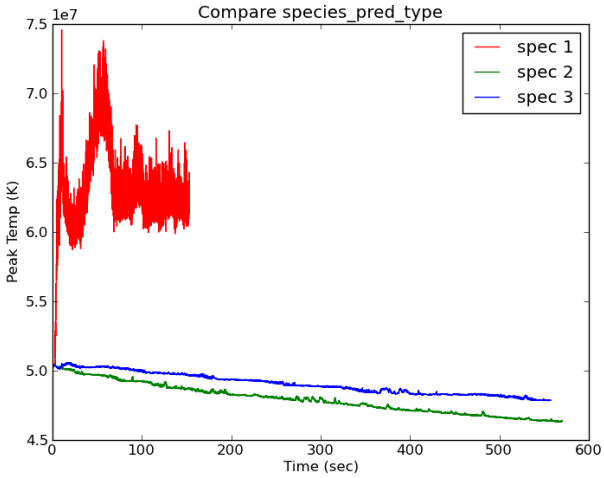


Figure 19.1: Compare `species_pred_type = 1,2,3` with `use_tfromp = T, enthalpy_pred_type = 1, cflfac = 0.7`

Using `use_tfromp = F` and `dpdt_factor` $>$ `0` results in many runs crashing very quickly and gives unphyiscal temperature profiles as seen in Figure 19.4.

Change `cflfac` and `enthalpy_pred_type`

With `species_pred_type = 1` and `cflfac = 0.1`, there is much less heating (peak T vs. t) than the `cflfac = 0.7` (default). There is also a lower overall Mach number (see Figure 19.6) with the `cflfac = 0.1` and excellent agreement between `tfromh` and `tfromp`.

`use_tfromp = F, dpdt_factor = 0.0, enthalpy_pred_type = 3,4 and species_pred_type = 2,3` shows cooling (as seen in `use_tfromp = T`) with a comparable rate of cooling (see Figure 19.7) to the `use_tfromp = T` case. The largest difference between the two runs is that the `use_tfromp = F` case

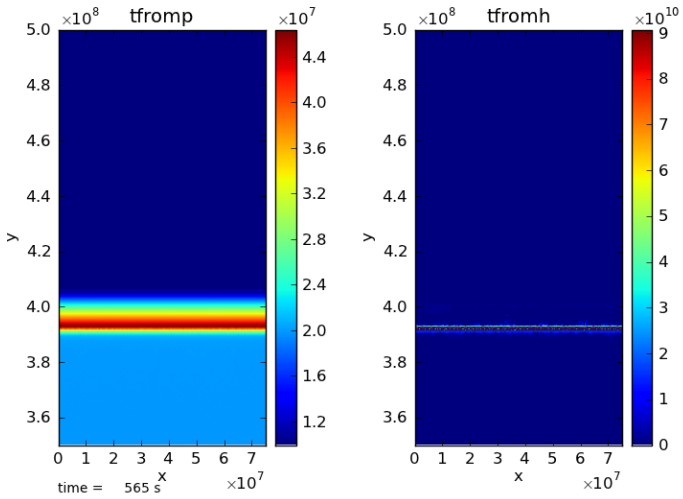


Figure 19.2: `tfromh` is unphysical when using `species_pred_type = 2,3`, `enthalpy_pred_type = 1`, `cflfac = 0.7`, `use_tfromp = T`. Shown above is `species_pred_type = 2`

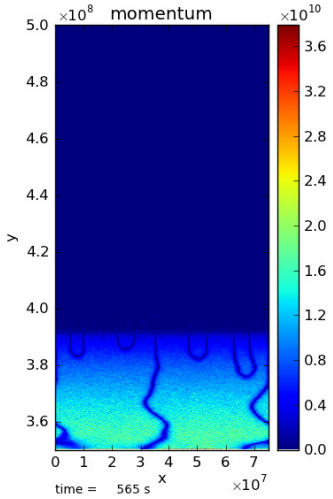


Figure 19.3: There are strange filament type features at the bottom of the domain. `species_pred_type = 2`, `enthalpy_pred_type = 1`, `cflfac = 0.7`, `use_tfromp = T`

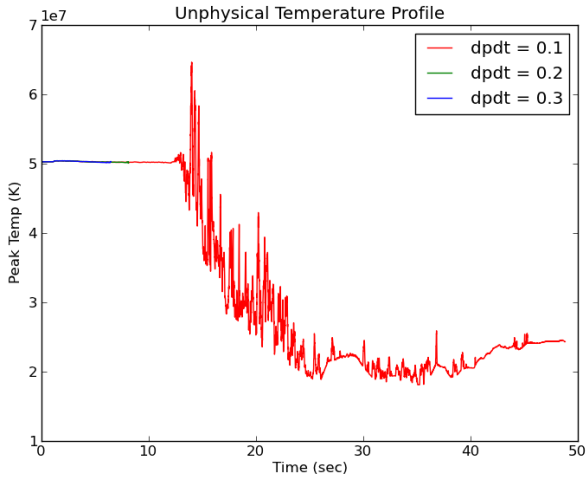


Figure 19.4: Unphysical temperature profile with `use_tfromp = F` and `dpdt_factor = 0.1`. `dpdt_factor = 0.2,0.3` lead to the code crashing.

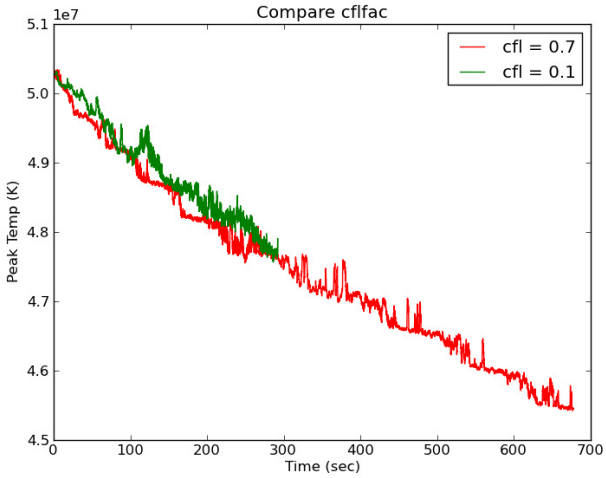


Figure 19.5: Compare `cflfac = 0.1` with `cflfac = 0.7` for `use_tfromp = F`, `dpdt_factor = 0.0`, `species_pred_type = 2`, `enthalpy_pred_type = 4`

shows excellent agreement between `tfromh` and `tfromp` with `cflfac = 0.7`. The filaments in the momentum plot of Figure 19.3 are still present.

For a given `enthalpy_pred_type` and `use_tfromp = F`, `species_pred_type = 2` has a lower Mach number (vs. t) compared to `species_pred_type = 3`.

Any `species_pred_type` with `use_tfromp = F`, `dpdt_factor = 0.0` and `enthalpy_pred_type = 1` shows significant heating, although the onset of the heating is delayed in `species_pred_type = 2,3`

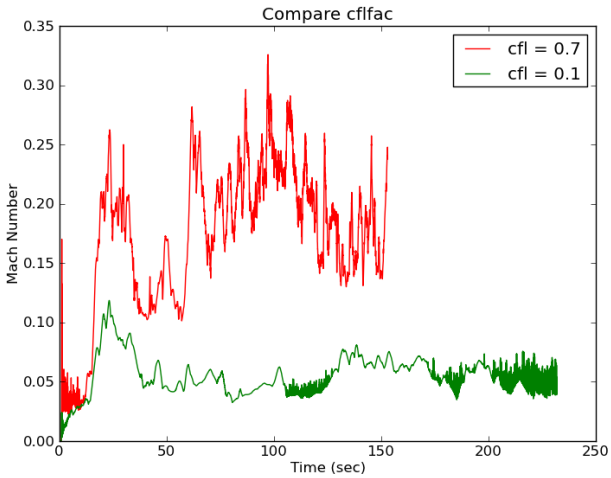


Figure 19.6: Comparing the Mach number of `cflfac = 0.1` and `cflfac = 0.7`. `species_pred_type = 1, enthalpy_pred_type = 1`

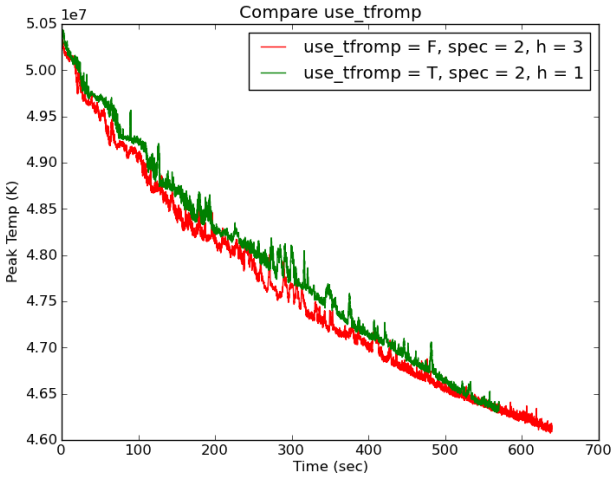


Figure 19.7: Illustrate the comparable cooling rates between `use_tfromp = T` and `use_tfromp = F` with `dpdt_factor = 0.0` using `species_pred_type = 2, enthalpy_pred_type = 3,1`

(see Figure 19.8). Only `species_pred_type = 1` gives good agreement between `tfromh` and `tfromp`.

Comparing `cflfac = 0.7` and `cflfac = 0.1` with `use_tfromp = F, dpdt_factor = 0.0, species_pred_type = 2` and `enthalpy_pred_type = 4` shows good agreement overall (see Figure 19.5).

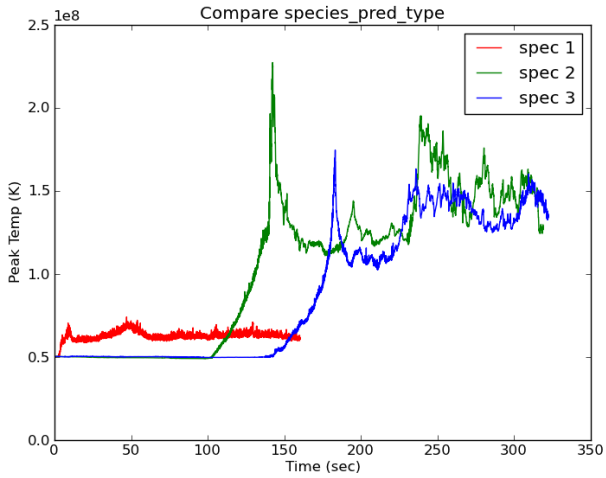


Figure 19.8: Compare various `species_pred_type` with `use_tfromp = F, dpdt_factor = 0.0, enthalpy_pred_type = 1`

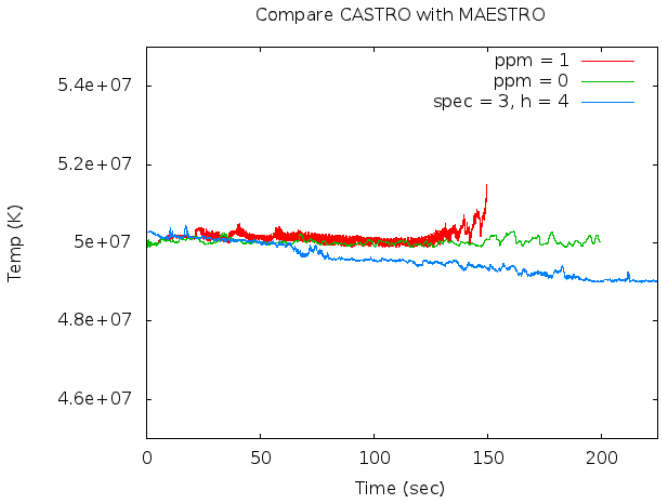


Figure 19.9: Compare the `castro.ppm_type` `CASTRO` runs with the `species_pred_type` `MAESTRO` runs.

Additional Runs

bds_type = 1

Using bds_type = 1, use_tfromp = F, dpdt_factor = 0.0, species_pred_type = 2, enthalpy_pred_type = 4 and cflfac = 0.7 seems to cool off faster, perhaps due to less mixing. There is also no momentum filaments in the bottom of the domain.

evolve_base_state = F

Using evolve_base_state = F, use_tfromp = F, dpdt_factor = 0.0, species_pred_type = 2 and enthalpy_pred_type = 4 seems to agree well with the normal evolve_base_state = T run.

toy_convect in CASTRO

toy_convect was also run using CASTRO with castro.ppm_type = 0,1. These runs show temperatures that cool off rather than increase (see Figure 19.9) which suggests using species_pred_type = 2,3 instead of species_pred_type = 1.

Recommendations

All of these runs suggest that running under species_pred_type = 2 or 3, enthalpy_pred_type = 3 or 4 with either use_tfromp = F and dpdt_factor = 0.0 or use_tfromp = T gives the most consistent results.

CHAPTER 20

Notes on SDC

SDC Overview

Spectral deferred corrections (SDC) is an iterative scheme used to integrate the thermodynamic variables, $(\rho X_k, \rho h)$, over a time step. It has been shown to be more accurate and efficient than Strang splitting in a terrestrial, non-stratified, low Mach number reacting flow solver [29], so we would like to develop an SDC version of MAESTRO.

MAESTRO integrates the following system of equations:

$$\frac{\partial \mathbf{U}}{\partial t} = -\mathbf{U} \cdot \nabla \mathbf{U} - \frac{1}{\rho} \nabla \pi - \frac{\rho - \rho_0}{\rho} g \mathbf{e}_r, \tag{20.1}$$

$$\frac{\partial (\rho X_k)}{\partial t} = -\nabla \cdot (\rho X_k \mathbf{U}) + \rho \dot{\omega}_k, \tag{20.2}$$

$$\frac{\partial (\rho h)}{\partial t} = -\nabla \cdot (\rho h \mathbf{U}) + \frac{D p_0}{D t} + \rho H_{\text{nuc}} + \nabla \cdot k_{\text{th}} \nabla T, \tag{20.3}$$

together with base state evolution equations and a constraint equation. By default, MAESTRO advances the thermodynamic variables by coupling the different physical processes together (advection, diffusion, reactions) using Strang splitting. Specifically, we integrate the reaction terms over half a time step ignoring contributions from advection and diffusion, then from this intermediate state we integrate the advection and diffusion terms over a full time step (while ignoring reactions), and finally integrate the reactions over a half time step (ignoring advection and diffusion). For problems where the reactions and/or diffusion greatly alter the energy balance or composition as compared to advection, this operator splitting approach can lead to large splitting errors and highly inaccurate solutions. This issue is can be particularly exasperating for low Mach number methods that can take large advection-based time steps.

An alternate approach to advancing the thermodynamic variables is SDC. SDC is an iterative approach to couple the various processes together, with each process seeing an approximation of the other pro-

cesses as a source term. The SDC algorithm converges to an integral representation of the solution in time that couples all of the processes together in a self-consistent fashion, see [29].

As a first attempt, we will work on coupling advection and reactions only via SDC, with a base state that is fixed in time, but not space.

Strang-Splitting Without Thermal Diffusion or Base State Evolution

In the Strang splitting version of MAESTRO, the reaction and advection processes operate independent of one-another. The species and enthalpy equations are integrated over Δt using the following sequence:

- React for $\Delta t/2$:

  In the Strang splitting version of MAESTRO, the reaction network solves just the reaction portion of the species evolution equations along with a temperature evolution equation. This is done using a standard stiff ODE solver package (like `VODE`) on the system:

$$\frac{dX_k}{dt} = \dot{\omega}_k(\rho, X_k, T), \tag{20.4}$$

$$\frac{dT}{dt} = \frac{1}{c_p}\left(-\sum_k \xi_k \dot{\omega}_k + H_{\rm nuc}\right). \tag{20.5}$$

  Here, T is evolved solely to evaluate the reaction rates, $\dot{\omega}_k(\rho, X_k, T)$. Furthermore, we simplify the problem "freezing" the thermodynamics—i.e., c_p and ξ_k are evaluated at the start of the integration and held constant thereafter. The density remains constant during this step, i.e., $\rho^{\rm new} = \rho^{\rm old}$, and we update the enthalpy at the end of the integration as:

$$h^{\rm new} = h^{\rm old} + \frac{\Delta t}{2} H_{\rm nuc}. \tag{20.6}$$

- Advect for Δt:

  Beginning with the results from the previous reaction half time step, we integrate that state using the equations

$$\frac{\partial(\rho X_k)}{\partial t} = -\nabla\cdot(\rho X_k \mathbf{U}), \tag{20.7}$$

$$\frac{\partial(\rho h)}{\partial t} = -\nabla\cdot(\rho h \mathbf{U}) + \frac{Dp_0}{Dt}. \tag{20.8}$$

  Note that no reaction terms appear here. Since the advection takes place using the state updated from the reaction step, the effect of the reactions is implicitly contained in the advective update.

- React for $\Delta t/2$:

  Finally, we react again, starting with the state left by the advection step.

Note that MAESTRO uses a predictor-corrector approach. After integrating $(\rho X_k, \rho h)$ over the time step, we use this time-advanced state to get a better estimate of a time-centered β_0 and S. We recompute the advective velocities using an updated divergence constraint and repeat the thermodynamic variable advance.

SDC Without Thermal Diffusion or Base State Evolution

In the SDC version, the VODE integration at the end of an SDC iteration is responsible for updating all the thermodynamic quantities including both the advection (incorporated via a piecewise constant advective flux divergence source term) and the reactions. This provides a much stronger coupling between the physical processes. In particular, our system now looks like:

$$\frac{d(\rho X_k)}{dt} = \rho\dot{\omega}_k(\rho, X_k, T) - \underbrace{\nabla \cdot (\rho X_k \mathbf{U})}_{A_{\rho X_k}} \tag{20.9}$$

$$\frac{d(\rho h)}{dt} = \rho H_{\rm nuc} - \underbrace{\nabla \cdot (\rho h \mathbf{U}) + \frac{Dp_0}{Dt}}_{A_{\rho h}} \tag{20.10}$$

Here, $A_{\rho X_k}$ and $A_{\rho h}$ are piecewise-constant (in time) approximations to the change in ρX_k and ρh (respectively) due to the advection. These are constructed by calling `density_advance` and `enthalpy_advance` in MAESTRO and passed into the network solver during the reaction step. A flowchart of the MAESTRO SDC algorithm is shown in figure 20.1.

Advective Update

In the advective update, our goal is to compute $A_{\rho X_k}$ and $A_{\rho h}$. These terms approximate the following:

$$A_{\rho X_k} = [-\nabla \cdot (\rho X_k \mathbf{U})]^{n+1/2} \tag{20.11}$$

$$A_{\rho h} = \left[-\nabla \cdot (\rho h \mathbf{U}) + \frac{Dp_0}{Dt}\right]^{n+1/2} \tag{20.12}$$

The construction of the interface states used in the advective terms uses either a time-lagged or iteratively-lagged approximation to the reaction terms ($I_{\rho X_k}$ and $I_{\rho h}$, see below) as a source term in the interface prediction. This explicitly couples the reaction process to the advection process.

Final Update

The RHS routine that the ODE solver operates on will first construct the density as:

$$\rho = \sum_k (\rho X_k) \tag{20.13}$$

It will then derive the temperature from the equation of state. If we are running with `use_tfromp = T`, then we do

$$T = T(\rho, p_0, X_k) \tag{20.14}$$

otherwise, we do

$$T = T(\rho, h, X_k) \tag{20.15}$$

Note that in contrast to the Strang splitting version, here we call the EOS every time we enter the RHS routine, but here we call the EOS to compute temperature rather than thermodynamic coefficients.

Finally we integrate the ODE system (Eqs. 20.9 and 20.10). At the end of the integration, we define $I_{\rho X_k}$ and $I_{\rho h}$. The actual form of these depends on what quantities we predict to edges during the construction of the advective fluxes. Note that we only need $I_{\rho X_k}$ and $I_{\rho h}$ for the prediction of the interface states, and not the `VODE` integration. This is because all we need from the advection solver is the approximation to $A_{\rho X_k}$ and $A_{\rho h}$ and not the final updated state.

Species Source Terms.

For the species prediction, the form of I depends on `species_pred_type` (see §19.2). We note that there is no I term for ρ or ρ' prediction, since the density evolution equation does not have a reaction source term.

- `species_pred_type` = 1 (`predict_rhoprime_and_X`) or 3 (`predict_rho_and_X`)

$$I_{X_k} = \frac{1}{\rho^{n+1/2}} \left[\frac{(\rho X_k)^{\rm new} - (\rho X_k)^{\rm old}}{\Delta t} - A_{\rho X_k} \right]. \tag{20.16}$$

(Andy's Idea) Define I_{X_k} using

$$I_{X_k} = \frac{X_k^{\rm new} - X_k^{\rm old}}{\Delta t} - A_{X_k}, \tag{20.17}$$

where we first define a state that has only been updated with advection:

$$\frac{(\rho X_k)^{(1)} - (\rho X_k)^{\rm old}}{\Delta t} = A_{\rho X_k}, \tag{20.18}$$

and then define the species mass fractions,

$$X_k^{(1)} = (\rho X_k)^{(1)} / \sum_k (\rho X_k)^{(1)}, \quad X_k^{\rm old} = (\rho X_k)^{\rm old} / \sum_k (\rho X_k)^{\rm old}, \quad X_k^{\rm new} = (\rho X_k)^{\rm new} / \sum_k (\rho X_k)^{\rm new}, \tag{20.19}$$

and finally define A_{X_k} using

$$\frac{X^{(1)} - X^{\rm old}}{\Delta t} = A_{X_k}. \tag{20.20}$$

- `species_pred_type` = 2 (`predict_rhoX`)

$$I_{\rho X_k} = \frac{(\rho X_k)^{\rm new} - (\rho X_k)^{\rm old}}{\Delta t} - A_{\rho X_k}. \tag{20.21}$$

Enthalpy Source Terms.

The appropriate constructions are:

- `enthalpy_pred_type` = 0 (`predict_rhoh`)

$$I_{\rho h} = \frac{(\rho h)^{\rm new} - (\rho h)^{\rm old}}{\Delta t} - A_{\rho h}. \tag{20.22}$$

- `enthalpy_pred_type` = 1 (`predict_rhohprime`, not implemented yet)

  (Andy's Idea) Here we need an $I_{\rho h}$ term for the $(\rho h)'$ evolution equation (see Eq. 19.22). In this case we will use $I_{(\rho h)'} = I_{\rho h}$. Since we are not evolving the base state, the PDE for $(\rho h)_0$ is simply $\partial(\rho h)_0/\partial t = 0$, and thus the evolution equation for $(\rho h)'$ is the same as the evolution equation for ρh.

  In the future, when we enable base state evolution, the base state enthalpy evolution equation may need to know about the $I_{\rho h}$ source term. In particular, should $(\rho h)_0$ see a $\overline{(\rho H_{\rm nuc})}$ term? what about an average thermal diffusion?

- `enthalpy_pred_type` = 2 (`predict_h`)

  This is the most straightforward prediction type. The SDC solver integrates the equation for (ρh):

$$\frac{\partial(\rho h)}{\partial t} = -\nabla\cdot(\rho h\mathbf{U}) + \frac{Dp_0}{Dt} + \rho H_{\rm nuc} \tag{20.23}$$

  (shown here without diffusion or external heat sources). Expanding the time derivative and divergence, and using the continuity equation we see:

$$\frac{\partial h}{\partial t} = -\mathbf{U}\cdot\nabla h + \frac{1}{\rho}\frac{Dp_0}{Dt} + \frac{1}{\rho}(\rho H_{\rm nuc}) \tag{20.24}$$

  Comparing these equations, we see that

$$I_h = \frac{1}{\rho^{n+1/2}}\left[\frac{(\rho h)^{\rm new} - (\rho h)^{\rm old}}{\Delta t} - A_{\rho h}\right] \tag{20.25}$$

  (Andy's Idea) Form I_h in the same way we would form I_{X_k} from above:

$$I_h = \frac{h^{\rm new} - h^{\rm old}}{\Delta t} - A_h, \tag{20.26}$$

  where we first define

$$\frac{(\rho h)^{(1)} - (\rho h)^{\rm old}}{\Delta t} = A_{\rho h}, \tag{20.27}$$

  and then define h,

$$h^{(1)} = (\rho h)^{(1)}/\sum_k(\rho X_k)^{(1)}, \quad h^{\rm old} = (\rho h)^{\rm old}/\sum_k(\rho X_k)^{\rm old}, \quad h^{\rm new} = (\rho h)^{\rm new}/\sum_k(\rho X_k)^{\rm new}, \tag{20.28}$$

  and finally define A_h using

$$I_h = \frac{h^{(1)} - h^{\rm old}}{\Delta t} = A_h. \tag{20.29}$$

- `enthalpy_pred_type` = 3 (`predict_T_then_rhoprime`) or `enthalpy_pred_type` = 4 (`predict_T_then_h`)

  Both of these `enthalpy_pred_types` predict temperature. Expressing $h = h(p_0, T, X_k)$ and differentiating along particle paths:

$$\frac{Dh}{Dt} = \left.\frac{\partial h}{\partial T}\right|_{p,X_k}\frac{DT}{Dt} + \left.\frac{\partial h}{\partial p}\right|_{T,X_k}\frac{Dp_0}{Dt} + \sum_k\left.\frac{\partial h}{\partial X_k}\right|_{p,T}\frac{DX_k}{Dt} \tag{20.30}$$

$$= c_p\frac{DT}{Dt} + h_p\frac{Dp_0}{Dt} + \sum_k \xi_k\dot{\omega}_k \tag{20.31}$$

where c_p, h_p, and ξ_k are as defined in the table of symbols (Table 1.1), and we substitute $DX_k/Dt = \dot{\omega}_k$ (from the species continuity equation, Eq. 19.10). Using Eq. 20.24, we have the familiar temperature evolution equation:

$$\rho c_p \frac{DT}{Dt} = \underbrace{(1 - \rho h_p)\frac{Dp_0}{Dt}}_{\substack{\text{already accounted for} \\ \text{in T prediction}}} - \sum_k \xi_k \rho \dot{\omega}_k + \rho H_{\text{nuc}} \tag{20.32}$$

where the underbraced term is already present in `mktempforce`. Recognizing that Eq. **??** is the SDC approximation to (ρH_{nuc}) and Eq. 20.21 is the SDC approximation to $(\rho \dot{\omega}_k)$, we can define

$$I_T = \frac{1}{\rho^{n+1/2} c_p^{n+1/2}} \left\{ \left[\frac{(\rho h)^{\text{new}} - (\rho h)^{\text{old}}}{\Delta t} - A_{\rho h} \right] - \sum_k \xi_k^{n+1/2} \left[\frac{(\rho X_k)^{\text{new}} - (\rho X_k)^{\text{old}}}{\Delta t} - A_{\rho X_k} \right] \right\} \tag{20.33}$$

(Andy's Idea) The idea is to advance the species and enthalpy with advection terms only, and compute the resulting temperature, $T^{(1)}$. Compare that temperature with the final temperature computed by the SDC VODE call. The difference between these values is I_T.

$$I_T = \frac{T^{\text{new}} - T^{\text{old}}}{\Delta t} - A_T, \tag{20.34}$$

with A_T given by

$$\frac{T^{(1)} - T^{\text{old}}}{\Delta t} = A_T, \tag{20.35}$$

and $T^{(1)}$ computed using the equation of state from $\rho^{(1)}$, $X_k^{(1)}$, and $h^{(1)}$ (or p_0, if `use_tfromp = T`).

Implementation

This is done in `advance.f90` just after the call to `react_state`, stored in the `multifab` called `intra`. These terms are used as the source terms for the advection step in the next SDC iteration.

Summary of Changes

The major changes from the non-SDC-enabled burners is the addition of the advective terms to the system of ODEs, the fact that we integrate (ρX_k) instead of just X_k, integrate (ρh) instead of T, and the need to derive the temperature from the input state for each RHS evaluation by `VODE`.

Note also that the SDC integration by `VODE` does not operate on the velocities at all. That update is handled in the same fashion as the Strang splitting version of the code.

The `ignition_simple_SDC` burner shows how to setup the system for `use_tfromp = T` or `F`. Presently, this implementation does not support `evolve_base_state = T` (in particular, we need to evolve p_0 in the RHS routine).

Algorithm Flowchart - ADR with Fixed Base State

We now include thermal diffusion and assume the base state is constant in time but not space:

$$\frac{\partial \mathbf{U}}{\partial t} = -\mathbf{U}\cdot\nabla\mathbf{U} - \frac{1}{\rho}\nabla\pi - \frac{\rho-\rho_0}{\rho}g\mathbf{e}_r, \tag{20.36}$$

$$\frac{\partial(\rho X_k)}{\partial t} = -\nabla\cdot(\rho X_k\mathbf{U}) + \rho\dot{\omega}_k, \tag{20.37}$$

$$\frac{\partial(\rho h)}{\partial t} = -\nabla\cdot(\rho h\mathbf{U}) + \underbrace{\mathbf{U}\cdot\nabla p_0}_{Dp_0/Dt} + \rho H_{\rm nuc} + \underbrace{\nabla\cdot\frac{k_{\rm th}}{c_p}\nabla h - \sum_k \nabla\cdot\frac{\xi_k k_{\rm th}}{c_p}\nabla X_k - \nabla\cdot\frac{h_p k_{\rm th}}{c_p}\nabla p_0}_{\nabla\cdot k_{\rm th}\nabla T}. \tag{20.38}$$

The time-advancement is divided into three major steps. The first step is the predictor, where we integrate the thermodynamic variables, $(\rho, \rho X_k, \rho h)$, over the full time step. The second step is corrector, where we use the results from the predictor to perform a more accurate temporal integration of the thermodynamic variables. The third step is the velocity and dynamic pressure update.

Step 1: (*Compute advection velocities*)

Use $\mathbf{U}^n$ and a second-order Godunov method to compute time-centered edge velocities, $\mathbf{U}^{\rm ADV,\star}$, with time-lagged dynamic pressure and explicit buoyancy as forcing terms. The $\star$ superscript indicates that this field does not satisfy the divergence constraint. Compute $S^{n+1/2,\rm pred}$ by extrapolating in time,

$$S^{n+1/2,\rm pred} = S^n + \frac{\Delta t^n}{2}\frac{S^n - S^{n-1}}{\Delta t^{n-1}}, \tag{20.39}$$

and project $\mathbf{U}^{\rm ADV,\star}$ to obtain $\mathbf{U}^{\rm ADV,pred}$, which satisfies

$$\nabla\cdot\left(\beta_0^n\mathbf{U}^{\rm ADV,pred}\right) = S^{n+1/2,\rm pred}. \tag{20.40}$$

Step 2: (*Predictor*)

In this step, we integrate $(\rho, \rho X_k, \rho h)$ over the full time step. The quantities $(S, \beta_0, k_{\rm th}, c_p, \xi_k, h_p)^n$ are computed from the the thermodynamic variables at t^n. This step is divided into several sub-steps:

Step 2A: (*Compute advective flux divergences*)

Use $\mathbf{U}^{\rm ADV,pred}$ and a second-order Godunov integrator to compute time-centered edge states, $(\rho X_k, \rho h)^{n+1/2,(0)}$, with time-lagged reactions ($I^{\rm lagged} = I^{(j_{\rm max})}$ from the previous time step), explicit diffusion, and time-centered thermodynamic pressure as source terms. Define the advective flux divergences as

$$A^{(0)}_{\rho X_k} = -\nabla\cdot\left[(\rho X_k)^{n+1/2,(0)}\,\mathbf{U}^{\rm ADV,pred}\right], \tag{20.41}$$

$$A^{(0)}_{\rho h} = -\nabla\cdot\left[(\rho h)^{n+1/2,(0)}\,\mathbf{U}^{\rm ADV,pred}\right] + \mathbf{U}^{\rm ADV,pred}\cdot\nabla p_0. \tag{20.42}$$

Next, use these fluxes to compute the time-advanced density,

$$\frac{\rho^{n+1}-\rho^n}{\Delta t} = \sum_k A^{(0)}_{\rho X_k}. \tag{20.43}$$

Then, compute preliminary, time-advanced species using

$$\frac{\rho^{n+1}\widehat{X}_k^{n+1,(0)} - (\rho X_k)^n}{\Delta t} = A^{(0)}_{\rho X_k} + I^{\text{lagged}}_{\rho X_k}. \tag{20.44}$$

Step 2B: (*Compute diffusive flux divergence*)

Solve a Crank-Nicolson-type diffusion equation for $\widehat{h}^{n+1,(0)}$, using transport coefficients evaluated at t^n everywhere,

$$\begin{aligned}
\frac{\rho^{n+1}\widehat{h}^{n+1,(0)} - (\rho h)^n}{\Delta t} &= A^{(0)}_{\rho h} + I^{\text{lagged}}_{\rho h} \\
&\quad + \frac{1}{2}\left(\nabla\cdot\frac{k^n_{\text{th}}}{c^n_p}\nabla h^n + \nabla\cdot\frac{k^n_{\text{th}}}{c^n_p}\nabla\widehat{h}^{n+1,(0)}\right) \\
&\quad - \frac{1}{2}\left(\sum_k \nabla\cdot\frac{\xi^n_k k^n_{\text{th}}}{c^n_p}\nabla X^n_k + \sum_k \nabla\cdot\frac{\xi^n_k k^n_{\text{th}}}{c^n_p}\nabla\widehat{X}^{n+1,(0)}_k\right) \\
&\quad - \frac{1}{2}\left(\nabla\cdot\frac{h^n_p k^n_{\text{th}}}{c^n_p}\nabla p_0 + \nabla\cdot\frac{h^n_p k^n_{\text{th}}}{c^n_p}\nabla p_0\right),
\end{aligned} \tag{20.45}$$

which is equivalent to

$$\begin{aligned}
\left(\rho^{n+1} - \frac{\Delta t}{2}\nabla\cdot\frac{k^n_{\text{th}}}{c^n_p}\nabla\right)\widehat{h}^{n+1,(0)} &= (\rho h)^n + \Delta t\left[A^{(0)}_{\rho h} + I^{\text{lagged}}_{\rho h} + \left(\frac{1}{2}\nabla\cdot\frac{k^n_{\text{th}}}{c^n_p}\nabla h^n\right)\right. \\
&\quad - \frac{1}{2}\left(\sum_k \nabla\cdot\frac{\xi^n_k k^n_{\text{th}}}{c^n_p}\nabla X^n_k + \sum_k \nabla\cdot\frac{\xi^n_k k^n_{\text{th}}}{c^n_p}\nabla\widehat{X}^{n+1,(0)}_k\right) \\
&\quad \left. - \frac{1}{2}\left(\nabla\cdot\frac{h^n_p k^n_{\text{th}}}{c^n_p}\nabla p_0 + \nabla\cdot\frac{h^n_p k^n_{\text{th}}}{c^n_p}\nabla p_0\right)\right].
\end{aligned} \tag{20.46}$$

Step 2C: (*Advance thermodynamic variables*)

Define $Q^{(0)}_{\rho X_k}$ as the right hand side of (20.44) without the $I^{\text{lagged}}_{\rho X_k}$ term, and define $Q^{(0)}_{\rho h}$ as the right hand side of (20.45) without the $I^{\text{lagged}}_{\rho h}$ term. Use `VODE` to integerate (20.37) and (20.38) over Δt to advance $(\rho X_k, \rho h)^n$ to $(\rho X_k, \rho h)^{n+1,(0)}$ using the piecewise-constant advection and diffusion source terms:

$$\frac{\partial(\rho X_k)}{\partial t} = Q^{(0)}_{\rho X_k} + \rho\dot{\omega}_k \tag{20.47}$$

$$\frac{\partial(\rho h)}{\partial t} = Q^{(0)}_{\rho h} + \rho H_{\text{nuc}}. \tag{20.48}$$

At this point we can define $I^{(0)}_{\rho X_k}$ and $I^{(0)}_{\rho h}$, or whatever term we need depending on our species and enthalpy edge state prediction types, for use in the corrector step. In our first implementation, we are predicting ρX_k and ρh, in which case we define:

$$I^{(0)}_{\rho X_k} = \frac{(\rho X_k)^{n+1,(0)} - (\rho X_k)^n}{\Delta t} - Q^{(0)}_{\rho X_k} \tag{20.49}$$

$$I^{(0)}_{\rho h} = \frac{(\rho h)^{n+1,(0)} - (\rho h)^n}{\Delta t} - Q^{(0)}_{\rho h}. \tag{20.50}$$

Step 3: (*Update advection velocities*)

First, compute $S^{n+1/2}$ and $\beta_0^{n+1/2}$ using

$$S^{n+1/2} = \frac{S^n + S^{n+1,(0)}}{2}, \qquad \beta_0^{n+1/2} = \frac{\beta_0^n + \beta_0^{n+1,(0)}}{2}. \tag{20.51}$$

Then, project $\mathbf{U}^{\mathrm{ADV},\star}$ to obtain $\mathbf{U}^{\mathrm{ADV}}$, which satisfies

$$\nabla \cdot \left(\beta_0^{n+1/2}\mathbf{U}^{\mathrm{ADV}}\right) = S^{n+1/2}. \tag{20.52}$$

Step 4: (*Corrector Loop*)

We loop over this step from $j = 1, j_{\max}$ times. In the corrector, we use the time-advanced state from the predictor to perform a more accurate integration of the thermodynamic variables. The quantities $(S, \beta_0, k_{\mathrm{th}}, c_p, \xi_k, h_p)^{n+1,(j-1)}$ are computed from $(\rho, \rho X_k, \rho h)^{n+1,(j-1)}$. This step is divided into several sub-steps:

Step 4A: (*Compute advective flux divergences*)

Use $\mathbf{U}^{\mathrm{ADV}}$ and a second-order Godunov integrator to compute time-centered edge states, $(\rho X_k, \rho h)^{n+1/2}$, with iteratively-lagged reactions ($I^{(j-1)}$), explicit diffusion, and time-centered thermodynamic pressure as source terms. Define the advective flux divergences as

$$A^{(j)}_{\rho X_k} = -\nabla \cdot \left[(\rho X_k)^{n+1/2,(j)}\,\mathbf{U}^{\mathrm{ADV}}\right], \tag{20.53}$$

$$A^{(j)}_{\rho h} = -\nabla \cdot \left[(\rho h)^{n+1/2,(j)}\,\mathbf{U}^{\mathrm{ADV}}\right] + \mathbf{U}^{\mathrm{ADV}} \cdot \nabla p_0. \tag{20.54}$$

Then, compute preliminary, time-advanced species using

$$\frac{\rho^{n+1}\widehat{X}_k^{n+1,(j)} - (\rho X_k)^n}{\Delta t} = A^{(j)}_{\rho X_k} + I^{(j-1)}_{\rho X_k}. \tag{20.55}$$

Step 4B: (*Compute diffusive flux divergence*)

Solve a backward-Euler-type correction equation for $\widehat{h}^{n+1,(j)}$,

$$\begin{aligned}\frac{\rho^{n+1}\widehat{h}^{n+1,(j)} - (\rho h)^n}{\Delta t} &= A_{\rho h}^{(j)} + I_{\rho h}^{(j-1)} \\ &+\nabla\cdot\frac{k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla\widehat{h}^{n+1,(j)} + \frac{1}{2}\left(\nabla\cdot\frac{k_{\rm th}^n}{c_p^n}\nabla h^n - \nabla\cdot\frac{k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla h^{n+1,(j-1)}\right) \\ &-\frac{1}{2}\left(\sum_k\nabla\cdot\frac{\xi_k^n k_{\rm th}^n}{c_p^n}\nabla X_k^n + \sum_k\nabla\cdot\frac{\xi_k^{n+1,(j-1)}k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla\widehat{X}_k^{n+1,(j)}\right) \\ &-\frac{1}{2}\left(\nabla\cdot\frac{h_p^n k_{\rm th}^n}{c_p^n}\nabla p_0 + \nabla\cdot\frac{h_p^{n+1,(j-1)}k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla p_0\right), \end{aligned} \tag{20.56}$$

which is equivalent to

$$\begin{aligned}\left(\rho^{n+1} - \Delta t\nabla\cdot\frac{k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla\right)\widehat{h}^{n+1,(j)} &= (\rho h)^n + \Delta t\Bigg[A_{\rho h}^{(j)} + I_{\rho h}^{(j-1)} \\ &+\frac{1}{2}\left(\nabla\cdot\frac{k_{\rm th}^n}{c_p^n}\nabla h^n - \nabla\cdot\frac{k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla h^{n+1,(j-1)}\right) \\ &-\frac{1}{2}\left(\sum_k\nabla\cdot\frac{\xi_k^n k_{\rm th}^n}{c_p^n}\nabla X_k^n + \sum_k\nabla\cdot\frac{\xi_k^{n+1,(j-1)}k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla\widehat{X}_k^{n+1,(j)}\right) \\ &-\frac{1}{2}\left(\nabla\cdot\frac{h_p^n k_{\rm th}^n}{c_p^n}\nabla p_0 + \nabla\cdot\frac{h_p^{n+1,(j-1)}k_{\rm th}^{n+1,(j-1)}}{c_p^{n+1,(j-1)}}\nabla p_0\right)\Bigg]. \end{aligned} \tag{20.57}$$

Step 4C: (*Advance thermodynamic variables*)

Define $Q_{\rho X_k}^{(j)}$ as the right hand side of (20.55) without the $I_{\rho X_k}^{(j-1)}$ term, and define $Q_{\rho h}^{(j)}$ as the right hand side of (20.56) without the $I_{\rho h}^{(j-1)}$ term. Use `VODE` to integerate (20.37) and (20.38) over Δt to advance $(\rho X_k, \rho h)^n$ to $(\rho X_k, \rho h)^{n+1,(j)}$ using the piecewise-constant advection and diffusion source terms:

$$\frac{\partial(\rho X_k)}{\partial t} = Q_{\rho X_k}^{(j)} + \rho\dot{\omega}_k \tag{20.58}$$

$$\frac{\partial(\rho h)}{\partial t} = Q_{\rho h}^{(j)} + \rho H_{\rm nuc}. \tag{20.59}$$

At this point we can define $I_{\rho X_k}^{(j)}$, $I_{\rho h}^{(j)}$, and any other I terms we need depending on our species and enthalpy edge state prediction types, for use in the predictor in the next time step. In our first implementation, we are predicting ρX_k and ρh, in which case we define:

$$I_{\rho X_k}^{(j)} = \frac{(\rho X_k)^{n+1,(j)} - (\rho X_k)^n}{\Delta t} - Q_{\rho X_k}^{(j)} \tag{20.60}$$

$$I_{\rho h}^{(j)} = \frac{(\rho h)^{n+1,(j)} - (\rho h)^n}{\Delta t} - Q_{\rho h}^{(j)}. \tag{20.61}$$

Step 5: (*Advance velocity and dynamic pressure*)

Similar to the original MAESTRO algorithm, more to come.

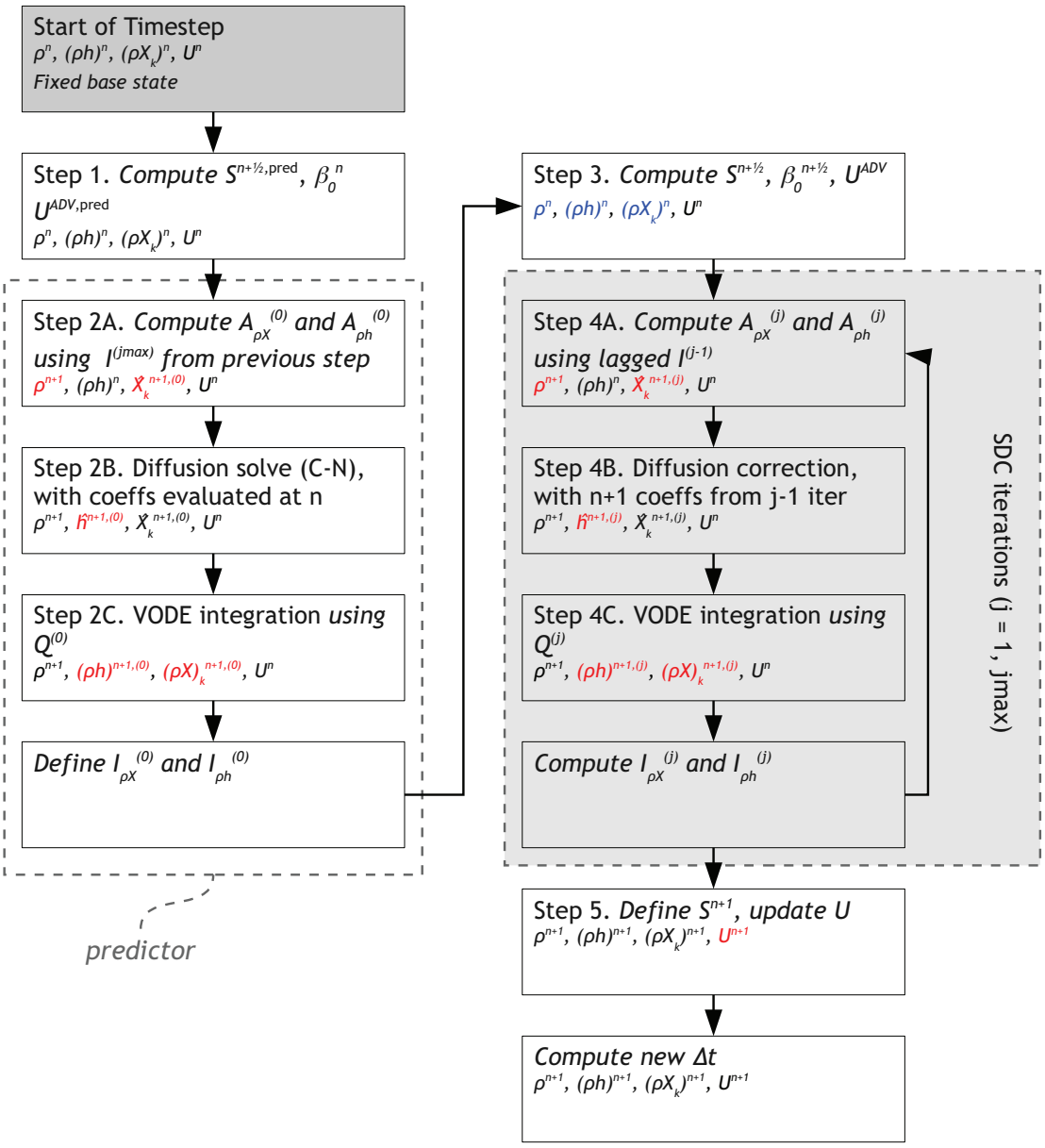


Figure 20.1: A flowchart of the MAESTRO SDC algorithm. The thermodynamic state variables and local velocity are indicated in each step. The base state is not shown as it is time-independent. Red text indicates that quantity was updated during that step. The predictor is outlined by the dotted box. The blue text indicates state variables that are the same in **Step 3** as they are in **Step 1**, i.e., they are unchanged by the predictor steps. The SDC loop is shown in the gray dotted box.

CHAPTER 21

Notes on Advection

These are working notes for the Godunov step in MAESTRO and VARDEN.

MAESTRO **Notation**

- For 2D, $\mathbf{U} = (u, w)$ and $\widetilde{\mathbf{U}} = (\tilde{u}, \tilde{w})$. Note that $u = \tilde{u}$. We will use the shorthand $\mathbf{i} = (x, r)$.
- For 3D plane parallel, $\mathbf{U} = (u, v, w)$ and $\widetilde{\mathbf{U}} = (\tilde{u}, \tilde{v}, \tilde{w})$. Note that $u = \tilde{u}$ and $v = \tilde{v}$. We will use the shorthand $\mathbf{i} = (x, y, r)$.
- For 3D spherical, $\mathbf{U} = (u, v, w)$ and $\widetilde{\mathbf{U}} = (\tilde{u}, \tilde{v}, \tilde{w})$. We will use the shorthand $\mathbf{i} = (x, y, z)$.

Computing U From $\widetilde{\mathbf{U}}$

For plane-parallel problems, in order to compute w from $\tilde{w}$, we use simple averaging

$$w^n_{\mathbf{i}} = \tilde{w}^n_{\mathbf{i}} + \frac{w_{0,r-\frac{1}{2}} + w_{0,r+\frac{1}{2}}}{2}. \tag{21.1}$$

For spherial problems, in order to compute $\mathbf{U}$ from $\widetilde{\mathbf{U}}$, we first map w_0 onto w_0^{MAC} using `put_w0_on_edges`, where w_0^{MAC} only contains normal velocities at each face. Then we construct $\mathbf{U}$ by using

$$u_{\mathbf{i}} = \tilde{u}_{\mathbf{i}} + \frac{w^{\mathrm{MAC}}_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_x} + w^{\mathrm{MAC}}_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}}{2}, \tag{21.2}$$

$$v_{\mathbf{i}} = \tilde{v}_{\mathbf{i}} + \frac{w^{\mathrm{MAC}}_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + w^{\mathrm{MAC}}_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}}{2}, \tag{21.3}$$

$$w_{\mathbf{i}} = \tilde{w}_{\mathbf{i}} + \frac{w^{\text{MAC}}_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_z} + w^{\text{MAC}}_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_z}}{2}. \tag{21.4}$$

To compute full edge-state velocities, simply add w_0 (for plane-parallel) or `w0mac` to the perturbational velocity directly since only edge-based quantities are involved.

Computing $\partial w_0/\partial r$

For plane-parallel problems, the spatial derivatives of w_0 are given by the two-point centered difference:

$$\left(\frac{\partial w_0}{\partial r}\right)_{\mathbf{i}} = \frac{w_{0,r+\frac{1}{2}} - w_{0,r-\frac{1}{2}}}{h}. \tag{21.5}$$

For spherical problems, we compute the radial bin centered gradient using

$$\left(\frac{\partial w_0}{\partial r}\right)_{r} = \frac{w_{0,r+\frac{1}{2}} - w_{0,r-\frac{1}{2}}}{\Delta r}. \tag{21.6}$$

Then we put $\partial w_0/\partial r$ onto a Cartesian grid using `put_1d_array_on_cart_3d_sphr`.

Computing $\widetilde{\mathbf{U}}^{\mathrm{TRANS}}$ in MAESTRO

In `advance_premac`, we call `mkutrans`, to compute $\widetilde{\mathbf{U}}^{\mathrm{TRANS}}$. We will only compute the normal component of velocity at each face. These transverse velocities do not contain w_0, so immediately following the call to `mkutrans`, we call `addw0` to compute $\mathbf{U}^{\mathrm{TRANS}}$ from $\widetilde{\mathbf{U}}^{\mathrm{TRANS}}$.

The evolution equation for the perturbational velocity is:

$$\frac{\partial \widetilde{\mathbf{U}}}{\partial t} = -\mathbf{U}\cdot\nabla\widetilde{\mathbf{U}} \underbrace{-(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial w_0}{\partial r}\mathbf{e}_r - \frac{1}{\rho}\nabla\pi + \frac{1}{\rho_0}\frac{\partial \pi_0}{\partial r}\mathbf{e}_r - \frac{(\rho-\rho_0)}{\rho}g\mathbf{e}_r}_{\text{forcing terms}}. \tag{21.7}$$

We extrapolate each velocity component to edge-centered, time-centered locations. For example,

$$\begin{aligned} \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &= \tilde{u}^n_{\mathbf{i}} + \frac{h}{2}\frac{\partial \tilde{u}^n_{\mathbf{i}}}{\partial x} + \frac{\Delta t}{2}\frac{\partial \tilde{u}^n_{\mathbf{i}}}{\partial t} \\ &= \tilde{u}^n_{\mathbf{i}} + \frac{h}{2}\frac{\partial \tilde{u}^n_{\mathbf{i}}}{\partial x} + \frac{\Delta t}{2}\left(-\tilde{u}^n_{\mathbf{i}}\frac{\partial \tilde{u}^n_{\mathbf{i}}}{\partial x} - \tilde{w}^n_{\mathbf{i}}\frac{\partial \tilde{u}^n_{\mathbf{i}}}{\partial r} + \text{forcing terms}\right) \end{aligned} \tag{21.8}$$

We are going to use a 1D Taylor series extrapolation in space and time. By 1D, we mean that we omit any spatial derivatives that are not in the direction of the extrapolation. We also omit the underbraced forcing terms. We also use characteristic tracing.

$$\tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}^n_{\mathbf{i}} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\min(0,\tilde{u}^n_{\mathbf{i}})\right]\partial\tilde{u}^n_{\mathbf{i}} \tag{21.9}$$

2D Cartesian Case

We predict $\tilde{u}$ to x-faces using a 1D extrapolation:

$$\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}^n_{\mathbf{i}-\mathbf{e}_x} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, u^n_{\mathbf{i}-\mathbf{e}_x})\right]\Delta_x \tilde{u}^n_{\mathbf{i}-\mathbf{e}_x}, \tag{21.10}$$

$$\tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}^n_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, u^n_{\mathbf{i}})\right]\Delta_x \tilde{u}^n_{\mathbf{i}}. \tag{21.11}$$

We pick the final trans states using a Riemann solver:

$$\tilde{u}^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} 0, & \left(\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \le 0 \text{ AND } \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \ge 0\right) \text{ OR } \left|\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon, \\ \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0, \end{cases} \tag{21.12}$$

We predict $\tilde{w}$ to r-faces using a 1D extrapolation:

$$\tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}^n_{\mathbf{i}-\mathbf{e}_r} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, w^n_{\mathbf{i}-\mathbf{e}_r})\right]\Delta_r \tilde{w}^n_{\mathbf{i}-\mathbf{e}_r}, \tag{21.13}$$

$$\tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}^n_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, w^n_{\mathbf{i}})\right]\Delta_r \tilde{w}^n_{\mathbf{i}}. \tag{21.14}$$

We pick the final TRANS states using a Riemann solver, noting that we upwind based on the full velocity.

$$\tilde{w}^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \begin{cases} 0, & \left(w_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} \le 0 \text{ AND } w_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} \ge 0\right) \text{ OR } \left|w_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + w_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right| < \epsilon, \\ \tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + w_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} > 0, \\ \tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + w_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} < 0, \end{cases} \tag{21.15}$$

3D Cartesian Case

We use the exact same procedure in 2D and 3D to compute $\tilde{u}^{\mathrm{TRANS}}$ and $\tilde{w}^{\mathrm{TRANS}}$. The procedure for computing $\tilde{v}^{\mathrm{TRANS}}$ is analogous to computing $\tilde{u}^{\mathrm{TRANS}}$. We predict $\tilde{v}$ to y-faces using the 1D extrapolation:

$$\tilde{v}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{v}^n_{\mathbf{i}-\mathbf{e}_y} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, v^n_{\mathbf{i}-\mathbf{e}_y})\right]\Delta_y \tilde{v}^n_{\mathbf{i}-\mathbf{e}_y}, \tag{21.16}$$

$$\tilde{v}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{v}^n_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, v^n_{\mathbf{i}})\right]\Delta_y \tilde{v}^n_{\mathbf{i}}, \tag{21.17}$$

$$\tilde{v}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \begin{cases} 0, & \left(v_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} \le 0 \ \text{ AND } \ v_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} \ge 0\right) \ \text{ OR } \ \left|v_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + v_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right| < \epsilon, \\ \tilde{v}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + v_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} > 0, \\ \tilde{v}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + v_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0. \end{cases} \tag{21.18}$$

3D Spherical Case

We predict the normal components of velocity to the normal faces using a 1D extrapolation. The equations for all three directions are identical to those given in the 2D and 3D plane-parallel sections. As in the plane-parallel case, make sure that the advection velocities, as well as the upwind velocity, is done with the full velocity, not the perturbational velocity.

Computing $\widetilde{\mathbf{U}}^{\mathrm{MAC},*}$ in MAESTRO

In `advance_premac`, we call `velpred` to compute $\widetilde{\mathbf{U}}^{\mathrm{MAC},*}$. We will only compute the normal component of velocity at each face.

For reference, here is the perturbational velocity equation from before:

$$\frac{\partial \widetilde{\mathbf{U}}}{\partial t} = -\mathbf{U}\cdot\nabla\widetilde{\mathbf{U}} \underbrace{-(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r)\frac{\partial w_0}{\partial r}\mathbf{e}_r \underbrace{-\frac{1}{\rho}\nabla\pi + \frac{1}{\rho_0}\frac{\partial \pi_0}{\partial r}\mathbf{e}_r - \frac{(\rho-\rho_0)}{\rho}g\mathbf{e}_r}_{\text{terms included in } \mathbf{f}_{\widetilde{\mathbf{U}}}}}_{\text{forcing terms}}. \tag{21.19}$$

Note that the $\partial w_0/\partial r$ term is treated like a forcing term, but it is not actually part of $\mathbf{f}_{\widetilde{\mathbf{U}}}$. We make use of the 1D extrapolations used to compute $\widetilde{\mathbf{U}}^{\mathrm{TRANS}}$ ($\tilde{u}_{L/R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $\tilde{v}_{L/R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, and $\tilde{w}_{L/R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}$), as well as the "TRANS" states ($\tilde{u}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $\tilde{v}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, and $\tilde{w}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}$)

2D Cartesian Case

1. Predict $\tilde{u}$ to r-faces using a 1D extrapolation.
2. Predict $\tilde{u}$ to x-faces using a full-dimensional extrapolation.
3. Predict $\tilde{w}$ to x-faces using a 1D extrapolation.
4. Predict $\tilde{w}$ to r-faces using a full-dimensional extrapolation.

Predict $\tilde{u}$ to r-faces using a 1D extrapolation:

$$\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{u}^n_{\mathbf{i}-\mathbf{e}_r} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, w^n_{\mathbf{i}-\mathbf{e}_r})\right]\Delta_r \tilde{u}^n_{\mathbf{i}-\mathbf{e}_r}, \tag{21.20}$$

$$\tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{u}_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, w^n_{\mathbf{i}})\right]\Delta_r \tilde{u}^n_{\mathbf{i}}. \tag{21.21}$$

Upwind based on w^{TRANS}:

$$\tilde{u}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \begin{cases} \frac{1}{2}\left(\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right), & \left|w^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right| < \epsilon \\ \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} > 0, \\ \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} < 0. \end{cases} \tag{21.22}$$

Predict $\tilde{u}$ to x-faces using a full-dimensional extrapolation,

$$\tilde{u}^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(w^{\text{TRANS}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + w^{\text{TRANS}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - \tilde{u}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\tilde{u},\mathbf{i}-\mathbf{e}_x}, \tag{21.23}$$

$$\tilde{u}^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(w^{\text{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + w^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - \tilde{u}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\tilde{u},\mathbf{i}}. \tag{21.24}$$

Solve a Riemann problem:

$$\tilde{u}^{\text{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} 0, & \left(u^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \le 0 \text{ AND } u^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \ge 0\right) \text{ OR } \left|u^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon, \\ \tilde{u}^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ \tilde{u}^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.25}$$

Predict $\tilde{w}$ to x-faces using a 1D extrapolation:

$$\tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{w}^n_{\mathbf{i}-\mathbf{e}_x} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, u^n_{\mathbf{i}-\mathbf{e}_x})\right]\Delta_x \tilde{w}^n_{\mathbf{i}-\mathbf{e}_x}, \tag{21.26}$$

$$\tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{w}_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, u^n_{\mathbf{i}})\right]\Delta_x \tilde{w}^n_{\mathbf{i}}. \tag{21.27}$$

Upwind based on u^{TRANS}:

$$\tilde{w}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(\tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + \tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ \tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ \tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.28}$$

Predict $\tilde{w}$ to r-faces using a full-dimensional extrapolation:

$$\begin{aligned}\tilde{w}^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} &- \frac{\Delta t}{4h}\left(u^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_r+\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_r-\frac{1}{2}\mathbf{e}_x}\right)\left(\tilde{w}_{\mathbf{i}-\mathbf{e}_r+\frac{1}{2}\mathbf{e}_x} - \tilde{w}_{\mathbf{i}-\mathbf{e}_r-\frac{1}{2}\mathbf{e}_x}\right)\\ &- \frac{\Delta t}{4h}\left(\tilde{w}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\tilde{w},\mathbf{i}-\mathbf{e}_r},\end{aligned} \tag{21.29}$$

$$\begin{aligned}\tilde{w}^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} &- \frac{\Delta t}{4h}\left(u^{\mathrm{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\left(\tilde{w}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} - \tilde{w}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\\ &- \frac{\Delta t}{4h}\left(\tilde{w}^{\mathrm{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\tilde{w},\mathbf{i}}.\end{aligned} \tag{21.30}$$

Solve a Riemann problem:

$$\tilde{w}^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \begin{cases} 0, & \left(w^{\mathrm{MAC},*}_L \le 0 \ \text{ AND } \ w^{\mathrm{MAC},*}_R \ge 0\right) \ \text{ OR } \ \left|w^{\mathrm{MAC},*}_L + w^{\mathrm{MAC},*}_R\right| < \epsilon, \\ \tilde{w}^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\mathrm{MAC},*}_L + w^{\mathrm{MAC},*}_R > 0, \\ \tilde{w}^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\mathrm{MAC},*}_L + w^{\mathrm{MAC},*}_R < 0. \end{cases} \tag{21.31}$$

3D Cartesian Case

This algorithm is more complicated than the 2D case since we include the effects of corner coupling.

1. Predict $\tilde{u}$ to y-faces using a 1D extrapolation.
2. Predict $\tilde{u}$ to r-faces using a 1D extrapolation.
3. Predict $\tilde{v}$ to x-faces using a 1D extrapolation.
4. Predict $\tilde{v}$ to r-faces using a 1D extrapolation.
5. Predict $\tilde{w}$ to x-faces using a 1D extrapolation.
6. Predict $\tilde{w}$ to y-faces using a 1D extrapolation.
7. Update prediction of $\tilde{u}$ to y-faces by accounting for r-derivatives.
8. Update prediction of $\tilde{u}$ to r-faces by accounting for y-derivatives.
9. Update prediction of $\tilde{v}$ to x-faces by accounting for r-derivatives.
10. Update prediction of $\tilde{v}$ to r-faces by accounting for x-derivatives.
11. Update prediction of $\tilde{w}$ to x-faces by accounting for y-derivatives.
12. Update prediction of $\tilde{w}$ to y-faces by accounting for x-derivatives.
13. Predict $\tilde{u}$ to x-faces using a full-dimensional extrapolation.
14. Predict $\tilde{v}$ to y-faces using a full-dimensional extrapolation.
15. Predict $\tilde{w}$ to r-faces using a full-dimensional extrapolation.

Predict $\tilde{u}$ to y-faces using a 1D extrapolation.

$$\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{u}^n_{\mathbf{i}-\mathbf{e}_y} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, v^n_{\mathbf{i}-\mathbf{e}_y})\right]\Delta_y \tilde{u}^n_{\mathbf{i}-\mathbf{e}_y}, \tag{21.32}$$

$$\tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{u}_{\mathbf{i}} - \left[\frac{1}{2} + \frac{\Delta t}{2h}\min(0, v^n_{\mathbf{i}})\right]\Delta_y \tilde{u}^n_{\mathbf{i}}. \tag{21.33}$$

Upwind based on v^{TRANS}:

$$\tilde{u}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \begin{cases} \frac{1}{2}\left(\tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right), & \left|v^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right| < \epsilon \\ \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} > 0, \\ \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0. \end{cases} \tag{21.34}$$

Predict $\tilde{u}$ to r-faces using a 1D extrapolation.

Predict $\tilde{v}$ to x-faces using a 1D extrapolation.

Predict $\tilde{v}$ to r-faces using a 1D extrapolation.

Predict $\tilde{w}$ to x-faces using a 1D extrapolation.

Predict $\tilde{w}$ to y-faces using a 1D extrapolation.

Update prediction of $\tilde{u}$ to y-faces by accounting for r-derivatives. The notation $\tilde{u}^{y|r}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ means state $\tilde{u}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ that has been updated to account for transverse derives in the r-direction.

$$\tilde{u}^{y|r}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} - \frac{\Delta t}{6h}\left(w^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_y+\frac{1}{2}\mathbf{e}_r} + w^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_y-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}_{\mathbf{i}-\mathbf{e}_y+\frac{1}{2}\mathbf{e}_r} - \tilde{u}_{\mathbf{i}-\mathbf{e}_y-\frac{1}{2}\mathbf{e}_r}\right), \tag{21.35}$$

$$\tilde{u}^{y|r}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} - \frac{\Delta t}{6h}\left(w^{\mathrm{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + w^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - \tilde{u}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right). \tag{21.36}$$

Upwind based on v^{TRANS}:

$$\tilde{u}^{y|r}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \begin{cases} \frac{1}{2}\left(\tilde{u}^{y|r}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + \tilde{u}^{y|r}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right), & \left|v^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right| < \epsilon \\ \tilde{u}^{y|r}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} > 0, \\ \tilde{u}^{y|r}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0. \end{cases} \tag{21.37}$$

Update prediction of $\tilde{u}$ to r-faces by accounting for y-derivatives.

Update prediction of $\tilde{v}$ to x-faces by accounting for r-derivatives.

Update prediction of $\tilde{v}$ to r-faces by accounting for x-derivatives.

Update prediction of $\tilde{w}$ to x-faces by accounting for y-derivatives.

Update prediction of $\tilde{w}$ to y-faces by accounting for x-derivatives.

Predict $\tilde{u}$ to x-faces using a full-dimensional extrapolation.

$$\begin{aligned} \tilde{u}^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(\tilde{u}^{y|r}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - \tilde{u}^{y|r}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + w^{\mathrm{TRANS}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}^{r|y}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - \tilde{u}^{r|y}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.38}$$

$$\begin{aligned} \tilde{u}^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \tilde{u}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\mathrm{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(\tilde{u}^{y|r}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - \tilde{u}^{y|r}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\mathrm{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + w^{\mathrm{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(\tilde{u}^{r|y}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - \tilde{u}^{r|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}}. \end{aligned} \tag{21.39}$$

Solve a Riemann problem:

$$\tilde{u}^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} 0, & \left(u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \le 0 \ \text{ AND } \ u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \ge 0\right) \ \text{ OR } \ \left|u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon, \\ \tilde{u}^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ \tilde{u}^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.40}$$

Predict $\tilde{v}$ to y-faces using a full-dimensional extrapolation.

Predict $\tilde{w}$ to r-faces using a full-dimensional extrapolation. In this step, make sure you account for the $\partial w_0/\partial r$ term before solving the Riemann problem:

$$\tilde{w}^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}^{\text{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{TRANS}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.41}$$

$$\tilde{w}^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \tilde{w}^{\text{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{TRANS}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{TRANS}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right) \tag{21.42}$$

3D Spherical Case

The spherical case is the same as the plane-parallel 3D Cartesian case, except the $\partial w_0/\partial r$ term enters in the full dimensional extrapolation for each direction. As in the plane-parallel case, make sure to upwind using the full velocity.

Computing $\rho^{\prime\mathrm{EDGE}}$, X_k^{EDGE}, $(\rho h)^{\prime\mathrm{EDGE}}$, and $\widetilde{\mathbf{U}}^{\mathrm{EDGE}}$ in MAESTRO

We call `make_edge_scal` to compute $\rho^{\prime\mathrm{EDGE}}$, X_k^{EDGE}, $(\rho h)^{\prime\mathrm{EDGE}}$, and $\widetilde{\mathbf{U}}^{\mathrm{EDGE}}$ at each edge. The procedure is the same for each quantitiy, so we shall simply denote the scalar as s. We always need to compute ρ' and X_k to faces, and the choice of energy prediction is as follows:

- For `enthalpy_pred_type` = 1, we predict $(\rho h)'$ to faces.
- For `enthalpy_pred_type` = 2, we predict h to faces.
- For `enthalpy_pred_type` = 3 and 4, we predict T to faces.
- For `enthalpy_pred_type` = 5, we predict h' to faces.
- For `enthalpy_pred_type` = 6, we predict T' to faces.

We are using `enthalpy_pred_type` = 1 for now. The equations of motion are:

$$\frac{\partial \rho'}{\partial t} = -\mathbf{U}\cdot\nabla\rho' \underbrace{-\rho'\nabla\cdot\mathbf{U} - \nabla\cdot\left(\rho_0\widetilde{\mathbf{U}}\right)}_{f_{\rho'}}, \tag{21.43}$$

$$\frac{\partial X_k}{\partial t} = -\mathbf{U}\cdot\nabla X_k \quad \text{(no forcing)}, \tag{21.44}$$

$$\frac{\partial (\rho h)'}{\partial t} = -\mathbf{U}\cdot\nabla(\rho h)' \underbrace{-(\rho h)'\nabla\cdot\mathbf{U} - \nabla\cdot\left[(\rho h)_0\widetilde{\mathbf{U}}\right] + \left(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r\right)\frac{\partial p_0}{\partial r} + \nabla\cdot k_{\mathrm{th}}\nabla T}_{f_{(\rho h)'}}, \tag{21.45}$$

$$\frac{\partial \widetilde{\mathbf{U}}}{\partial t} = -\mathbf{U}\cdot\nabla\widetilde{\mathbf{U}} \underbrace{\underbrace{-\left(\widetilde{\mathbf{U}}\cdot\mathbf{e}_r\right)\frac{\partial w_0}{\partial r}\mathbf{e}_r \underbrace{-\frac{1}{\rho}\nabla\pi + \frac{1}{\rho_0}\frac{\partial \pi_0}{\partial r}\mathbf{e}_r - \frac{(\rho-\rho_0)}{\rho}g\mathbf{e}_r}_{\text{terms included in } \mathbf{f}_{\widetilde{\mathbf{U}}}}}_{\text{forcing terms}}}. \tag{21.46}$$

2D Cartesian Case

1. Predict s to r-faces using a 1D extrapolation.
2. Predict s to x-faces using a full-dimensional extrapolation.
3. Predict s to x-faces using a 1D extrapolation.
4. Predict s to r-faces using a full-dimensional extrapolation.

Predict s to r-faces using a 1D extrapolation:

$$s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^n_{\mathbf{i}-\mathbf{e}_r} + \left(\frac{1}{2} - \frac{\Delta t}{2h} w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \Delta_r s^n_{\mathbf{i}-\mathbf{e}_r}, \tag{21.47}$$

$$s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h} w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \Delta_r s^n_{\mathbf{i}}. \tag{21.48}$$

Upwind based on w^{MAC}:

$$s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \begin{cases} \frac{1}{2}\left(s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right), & \left|w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right| < \epsilon \\ s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} > 0, \\ s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} < 0. \end{cases} \tag{21.49}$$

Predict s to x-faces using a full-dimensional extrapolation. First, the normal derivative and forcing terms:

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^n_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{2h} u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2} f^n_{\mathbf{i}-\mathbf{e}_x} \tag{21.50}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^n_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h} u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}} + \frac{\Delta t}{2} f^n_{\mathbf{i}} \tag{21.51}$$

Account for the transverse terms:

if `is_conservative` **then**

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{2h}\left[\left(w^{\text{MAC}}s\right)_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - \left(w^{\text{MAC}}s\right)_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right] - \frac{\Delta t}{2h} s^n_{\mathbf{i}-\mathbf{e}_x}\left(u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - u^{\text{MAC}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_x}\right) \tag{21.52}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{2h}\left[\left(w^{\text{MAC}}s\right)_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - \left(w^{\text{MAC}}s\right)_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right] - \frac{\Delta t}{2h} s^n_{\mathbf{i}}\left(u^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} - u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \tag{21.53}$$

else

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(s_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - s_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.54}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(w^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(s_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.55}$$

end if

Account for the $\partial w_0/\partial r$ term:

if `is_vel` **and** `comp` = 2 **then**

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.56}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.57}$$

$$\tag{21.58}$$

end if

Upwind based on u^{MAC}.

$$s^{\text{EDGE}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.59}$$

Predict s to x-faces using a 1D extrapolation:

$$s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^n_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{2h}u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta_x s^n_{\mathbf{i}-\mathbf{e}_x}, \tag{21.60}$$

$$s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h}u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta_x s^n_{\mathbf{i}}. \tag{21.61}$$

Upwind based on u^{MAC}:

$$s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.62}$$

Predict s to r-faces using a full-dimensional extrapolation. First, the normal derivative and forcing terms:

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^n_{\mathbf{i}-\mathbf{e}_r} + \left(\frac{1}{2} - \frac{\Delta t}{2h}w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\Delta_r s^n_{\mathbf{i}-\mathbf{e}_r} + \frac{\Delta t}{2}f^n_{\mathbf{i}-\mathbf{e}_r} \tag{21.63}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^n_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h}w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\Delta_r s^n_{\mathbf{i}} + \frac{\Delta t}{2}f^n_{\mathbf{i}} \tag{21.64}$$

Account for the transverse terms:

if `is_conservative` **then**

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{2h}\left[\left(u^{\text{MAC}}s\right)_{\mathbf{i}-\mathbf{e}_r+\frac{1}{2}\mathbf{e}_x} - \left(u^{\text{MAC}}s\right)_{\mathbf{i}-\mathbf{e}_r-\frac{1}{2}\mathbf{e}_x}\right] - \frac{\Delta t}{2h}s^n_{\mathbf{i}-\mathbf{e}_r}\left(w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - w^{\text{MAC}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right) \tag{21.65}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{2h}\left[\left(u^{\text{MAC}}s\right)_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} - \left(u^{\text{MAC}}s\right)_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right] - \frac{\Delta t}{2h}s^n_{\mathbf{i}}\left(w^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.66}$$

else

$$s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(u^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_r+\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_r-\frac{1}{2}\mathbf{e}_x}\right)\left(s_{\mathbf{i}-\mathbf{e}_r+\frac{1}{2}\mathbf{e}_x} - s_{\mathbf{i}-\mathbf{e}_r-\frac{1}{2}\mathbf{e}_x}\right) \tag{21.67}$$

$$s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(u^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\left(s_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} - s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \tag{21.68}$$

end if

Account for the $\partial w_0/\partial r$ term:

if `is_vel` **and** `comp` = 2 **then**

$$s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(\tilde{w}^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\mathrm{MAC}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{3}{2}\mathbf{e}_r}\right) \tag{21.69}$$

$$s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} - \frac{\Delta t}{4h}\left(\tilde{w}^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.70}$$

$$\tag{21.71}$$

end if

Upwind based on w^{MAC}:

$$s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} = \begin{cases} \frac{1}{2}\left(s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r} + s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right), & \left|w^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right| < \epsilon \\ u_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} > 0, \\ u_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}, & w^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r} < 0. \end{cases} \tag{21.72}$$

3D Cartesian Case

This algorithm is more complicated than the 2D case since we include the effects of corner coupling.

1. Predict s to x-faces using a 1D extrapolation.
2. Predict s to y-faces using a 1D extrapolation.
3. Predict s to r-faces using a 1D extrapolation.
4. Update prediction of s to x-faces by accounting for y-derivatives.
5. Update prediction of s to x-faces by accounting for r-derivatives.
6. Update prediction of s to y-faces by accounting for x-derivatives.
7. Update prediction of s to y-faces by accounting for r-derivatives.
8. Update prediction of s to r-faces by accounting for x-derivatives.
9. Update prediction of s to r-faces by accounting for y-derivatives.
10. Predict s to x-faces using a full-dimensional extrapolation.
11. Predict s to y-faces using a full-dimensional extrapolation.
12. Predict s to r-faces using a full-dimensional extrapolation.

Predict s to x-faces using a 1D extrapolation.

$$s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^n_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{2h} u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}-\mathbf{e}_x}, \tag{21.73}$$

$$s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h} u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}}. \tag{21.74}$$

Upwind based on u^{MAC}:

$$s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.75}$$

Predict s to y-faces using a 1D extrapolation.

Predict s to r-faces using a 1D extrapolation.

Update prediction of s to x-faces by accounting for y-derivatives. The notation $s^{x|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$ means "state $s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$ that has been updated to account for the transverse derivatives in the y-direction".

if `is_conservative` **then**

$$s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{3h}\left[(sv^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - (sv^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right], \tag{21.76}$$

$$s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{3h}\left[(sv^{\text{MAC}})_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - (sv^{\text{MAC}})_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right]. \tag{21.77}$$

else

$$s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{6h}\left(v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(s_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right), \tag{21.78}$$

$$s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{6h}\left(v^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(s_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right). \tag{21.79}$$

end if

Upwind based on u^{MAC}:

$$s^{x|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.80}$$

Update prediction of s to x-faces by accounting for r-derivatives.

Update prediction of s to y-faces by accounting for x-derivatives.

Update prediction of s to y-faces by accounting for r-derivatives.

Update prediction of s to r-faces by accounting for x-derivatives.

Update prediction of s to r-faces by accounting for y-derivatives.

Predict s to x-faces using a full-dimensional extrapolation.

if `is_conservative` **then**

$$\begin{aligned} s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[\left(s^{y|r}v^{\text{MAC}}\right)_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - \left(s^{y|r}v^{\text{MAC}}\right)_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2h}\left[\left(s^{r|y}w^{\text{MAC}}\right)_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - \left(s^{r|y}w^{\text{MAC}}\right)_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right] \\ &- \frac{\Delta t}{2h}s_{\mathbf{i}-\mathbf{e}_x}\left(u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - u^{\text{MAC}}_{\mathbf{i}-\frac{3}{2}\mathbf{e}_x}\right) + \frac{\Delta t}{2}f_{\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.81}$$

$$\begin{aligned} s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[\left(s^{y|r}v^{\text{MAC}}\right)_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - \left(s^{y|r}v^{\text{MAC}}\right)_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2h}\left[\left(s^{r|y}w^{\text{MAC}}\right)_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - \left(s^{r|y}w^{\text{MAC}}\right)_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right] \\ &- \frac{\Delta t}{2h}s_{\mathbf{i}}\left(u^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_x} - u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) + \frac{\Delta t}{2}f_{\mathbf{i}}. \end{aligned} \tag{21.82}$$

else

$$\begin{aligned} s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{y|r}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s^{y|r}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(s^{r|y}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} - s^{r|y}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.83}$$

$$\begin{aligned} s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{y|r}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s^{y|r}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(s^{r|y}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - s^{r|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) + \frac{\Delta t}{2} f_{\mathbf{i}}. \end{aligned} \tag{21.84}$$

end if

Account for the $\partial w_0/\partial r$ term:

if `is_vel` **and** `comp` = 2 **then**

$$s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.85}$$

$$s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(\tilde{w}^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_r} + \tilde{w}^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right)\left(w_{0,\mathbf{i}+\frac{1}{2}\mathbf{e}_r} - w_{0,\mathbf{i}-\frac{1}{2}\mathbf{e}_r}\right) \tag{21.86}$$

(21.87)

end if

Upwind based on u^{MAC}:

$$s^{\text{EDGE}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.88}$$

Predict s to y-faces using a full-dimensional extrapolation.

Predict s to r-faces using a full-dimensional extrapolation.

3D Spherical Case

The spherical case is the same as the plane-parallel 3D Cartesian case, except the $\partial w_0/\partial r$ term enters in the full dimensional extrapolation for each direction when predicting velocity to faces. As in the plane-parallel case, make sure upwind based on the full velocity.

Computing $\mathbf{U}^{\mathrm{MAC},*}$ in VARDEN

2D Cartesian Case

We do a 1D Taylor series extrapolation to get both components of velocity at the x-face:

$$u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u_{\mathbf{i}-\mathbf{e}_x} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, u_{\mathbf{i}-\mathbf{e}_x})\right]\Delta_x u_{\mathbf{i}-\mathbf{e}_x}, \tag{21.89}$$

$$u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u_{\mathbf{i}} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\min(0, u_{\mathbf{i}})\right]\Delta_x u_{\mathbf{i}}. \tag{21.90}$$

$$v^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = v_{\mathbf{i}-\mathbf{e}_x} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\max(0, v_{\mathbf{i}-\mathbf{e}_x})\right]\Delta_x v_{\mathbf{i}-\mathbf{e}_x}, \tag{21.91}$$

$$v^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = v_{\mathbf{i}} + \left[\frac{1}{2} - \frac{\Delta t}{2h}\min(0, v_{\mathbf{i}})\right]\Delta_x v_{\mathbf{i}}. \tag{21.92}$$

We obtain the normal velocity using the Riemann problem:

$$u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} 0, & \left(u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \le 0 \text{ AND } u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \ge 0\right) \text{ OR } \left|u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon, \\ u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.93}$$

We obtain the transverse velocity by upwinding based on $u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$:

$$v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(v^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + v^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ v^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ v^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.94}$$

We perform analogous operations to compute both components of velocity at the y-faces, $\mathbf{U}^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$.

Now we do a full-dimensional extrapolation to get the MAC velocity at the x-faces (note that we only compute the normal components):

$$u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(v^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(u^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - u^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}-\mathbf{e}_x}, \tag{21.95}$$

$$u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(v^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(u^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}}. \tag{21.96}$$

Then we solve a Riemann problem:

$$u^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} 0, & \left(u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \le 0 \text{ AND } u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} \ge 0\right) \text{ OR } \left|u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon, \\ u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.97}$$

We perform analogous operations to compute the normal velocity at the y-faces, $v^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$.

3D Cartesian Case

This is more complicated than the 2D case because we include corner coupling. We compute $\mathbf{U}^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $\mathbf{U}^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, and $\mathbf{U}^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$ in an analogous manner as equations (21.89)-(21.94). Then we compute an intermediate state, $u^{y|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, which is described as "state $u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ that has been updated to account for the transverse derivatives in the z direction", using:

$$u^{y|z}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} - \frac{\Delta t}{6h}\left(w^{1D}_{\mathbf{i}-\mathbf{e}_y+\frac{1}{2}\mathbf{e}_z} + w^{1D}_{\mathbf{i}-\mathbf{e}_y-\frac{1}{2}\mathbf{e}_z}\right)\left(u^{1D}_{\mathbf{i}-\mathbf{e}_y+\frac{1}{2}\mathbf{e}_z} - u^{1D}_{\mathbf{i}-\mathbf{e}_y-\frac{1}{2}\mathbf{e}_z}\right), \tag{21.98}$$

$$u^{y|z}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} - \frac{\Delta t}{6h}\left(w^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} + w^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right)\left(u^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} - u^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right). \tag{21.99}$$

Then upwind based on $v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$:

$$u^{y|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} = \begin{cases} \frac{1}{2}\left(u^{y|z}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + u^{y|z}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right), & \left|v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right| < \epsilon \\ u^{y|z}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} > 0, \\ u^{y|z}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_y}, & v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0. \end{cases} \tag{21.100}$$

We use an analogous procedure to compute five more intemediate states, $u^{z|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$, $v^{x|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $v^{z|x}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$, $w^{x|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, and $w^{y|x}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$. Then we do a full-dimensional extrapolation to get the MAC velocities at normal faces:

$$\begin{aligned} u^{\mathrm{MAC},*}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(u^{y|z}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - u^{y|z}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} + w^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right)\left(u^{z|y}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} - u^{z|y}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}-\mathbf{e}_x} \end{aligned} \tag{21.101}$$

$$\begin{aligned} u^{\mathrm{MAC},*}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = u^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(u^{y|z}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - u^{y|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} + w^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right)\left(u^{z|y}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} - u^{z|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right) + \frac{\Delta t}{2} f_{u,\mathbf{i}}. \end{aligned} \tag{21.102}$$

Then we use the Riemann solver given above for the 2D case (equation [21.97]) to compute $u^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$. We use an analogous procedure to obtain $v^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ and $w^{\mathrm{MAC},*}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$.

Computing $\mathbf{U}^{\mathrm{EDGE}}$ and ρ^{EDGE} in VARDEN

To compute $\mathbf{U}^{\mathrm{EDGE}}$, VARDEN uses the exact same algorithm as the s^{EDGE} case in MAESTRO. The algorithm for ρ^{EDGE} in VARDEN is slightly different than the s^{EDGE} case in MAESTRO since it uses a "conservative" formulation. Here, s is used in place of either ρ, u, v, or w (in 3D).

2D Cartesian Case

The 1D extrapolation is:

$$s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^n_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{2h} u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}-\mathbf{e}_x}, \tag{21.103}$$

$$s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{2h} u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right) \Delta_x s^n_{\mathbf{i}}. \tag{21.104}$$

Then we upwind based on u^{MAC}:

$$s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.105}$$

We use an analogous procedure to obtain $s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$. Now we do a full-dimensional extrapolation of s to each face. The extrapolation of a "non-conserved" s to x-faces is:

$$s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{s,\mathbf{i}-\mathbf{e}_x}, \tag{21.106}$$

$$s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{4h}\left(v^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{s,\mathbf{i}}. \tag{21.107}$$

The extrapolation of a "conserved" s to x-faces is:

$$\begin{aligned} s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[(s^{1D}v^{\mathrm{MAC}})_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - (s^{1D}v^{\mathrm{MAC}})_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2} s_{\mathbf{i}-\mathbf{e}_x}(\nabla\cdot\mathbf{U}^{\mathrm{MAC}})_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2h} s_{\mathbf{i}-\mathbf{e}_x}\left(v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{s,\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.108}$$

$$\begin{aligned} s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[(s^{1D}v^{\mathrm{MAC}})_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - (s^{1D}v^{\mathrm{MAC}})_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2} s_{\mathbf{i}}(\nabla\cdot\mathbf{U}^{\mathrm{MAC}})_{\mathbf{i}} + \frac{\Delta t}{2h} s_{\mathbf{i}}\left(v^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - v^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) + \frac{\Delta t}{2} f_{s,\mathbf{i}}. \end{aligned} \tag{21.109}$$

Then we upwind based on u^{MAC}.

$$s^{\mathrm{EDGE}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.110}$$

We use an analogous procedure to compute $s^{\mathrm{EDGE}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$.

3D Cartesian Case

This is more complicated than the 2D case because we include corner coupling. We first compute $s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, and $s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$ in an analogous manner to equations (21.103) and (21.104). Then we compute six intermediate states, $s^{x|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $s^{x|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, $s^{y|x}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, $s^{y|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$, $s^{z|x}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$, and $s^{z|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$. For the "non-conservative case", we use, for example:

$$s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{6h}\left(v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{1D}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s^{1D}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right), \tag{21.111}$$

$$s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{6h}\left(v^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{1D}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s^{1D}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right). \tag{21.112}$$

For the "conservative" case, we use, for example:

$$s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{3h}\left[(sv^{\mathrm{MAC}})_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - (sv^{\mathrm{MAC}})_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right], \tag{21.113}$$

$$s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} - \frac{\Delta t}{3h}\left[(sv^{\mathrm{MAC}})_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - (sv^{\mathrm{MAC}})_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right]. \tag{21.114}$$

Then we upwind based on u^{MAC}:

$$s^{x|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} \frac{1}{2}\left(s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} + s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right), & \left|u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right| < \epsilon \\ s^{x|y}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s^{x|y}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}, & u^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} < 0. \end{cases} \tag{21.115}$$

We use an analogous procedure to compute the other five intermediate states. Now we do a full-dimensional extrapolation of s to each face. The extrapolation of a "non-conserved" s to x-faces is:

$$\begin{aligned} s^{\mathrm{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{y|z}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s^{y|z}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} + w^{\mathrm{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right)\left(s^{z|y}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} - s^{z|y}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right) \\ &+ \frac{\Delta t}{2} f_{s,\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.116}$$

$$\begin{aligned} s^{\mathrm{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{4h}\left(v^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + v^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right)\left(s^{y|z}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s^{y|z}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right) \\ &- \frac{\Delta t}{4h}\left(w^{\mathrm{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} + w^{\mathrm{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right)\left(s^{z|y}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} - s^{z|y}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right) \\ &+ \frac{\Delta t}{2} f_{s,\mathbf{i}}. \end{aligned} \tag{21.117}$$

The extrapolation of a "conserved" s to x-faces is:

$$\begin{aligned} s^{\text{EDGE}}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{L,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[(s^{y|z}v^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - (s^{y|z}v^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2h}\left[(s^{z|y}w^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} - (s^{z|y}w^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right] \\ &- \frac{\Delta t}{2}s_{\mathbf{i}-\mathbf{e}_x}(\nabla\cdot\mathbf{U}^{\text{MAC}})_{\mathbf{i}-\mathbf{e}_x} \\ &+ \frac{\Delta t}{2h}s_{\mathbf{i}-\mathbf{e}_x}\left(v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - v^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y} + w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_z} - w^{\text{MAC}}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_z}\right) \\ &+ \frac{\Delta t}{2}f_{s,\mathbf{i}-\mathbf{e}_x}, \end{aligned} \tag{21.118}$$

$$\begin{aligned} s^{\text{EDGE}}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = s^{1D}_{R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x} &- \frac{\Delta t}{2h}\left[(s^{y|z}v^{\text{MAC}})_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - (s^{y|z}v^{\text{MAC}})_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}\right] \\ &- \frac{\Delta t}{2h}\left[(s^{z|y}w^{\text{MAC}})_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} - (s^{z|y}w^{\text{MAC}})_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right] \\ &- \frac{\Delta t}{2}s_{\mathbf{i}}(\nabla\cdot\mathbf{U}^{\text{MAC}})_{\mathbf{i}} \\ &+ \frac{\Delta t}{2h}s_{\mathbf{i}}\left(v^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - v^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} + w^{\text{MAC}}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_z} - w^{\text{MAC}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}\right) \\ &+ \frac{\Delta t}{2}f_{s,\mathbf{i}}. \end{aligned} \tag{21.119}$$

Then we upwind based on u^{MAC}, as in equation (21.110). We use an analogous procedure to compute both $s^{\text{EDGE}}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ and $s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_z}$.

ESTATE_FPU in GODUNOV_2D/3D.f

First, the normal predictor.

$$s_L^x = s_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{h_x}\mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}-\mathbf{e}_x} + \underbrace{\frac{\Delta t}{2}\mathrm{TFORCES}_{\mathbf{i}-\mathbf{e}_x}}_{\text{IF USE\_MINION}} \tag{21.120}$$

$$s_R^x = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{h_x}\mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \underbrace{\frac{\Delta t}{2}\mathrm{TFORCES}_{\mathbf{i}}}_{\text{IF USE\_MINION}} \tag{21.121}$$

If USE_MINION **and** ICONSERVE **then:**

$$s_L^x = s_L^x - \frac{\Delta t}{2} s_{\mathbf{i}-\mathbf{e}_x}\mathrm{DIVU}_{\mathbf{i}-\mathbf{e}_x} \tag{21.122}$$

$$s_R^x = s_R^x - \frac{\Delta t}{2} s_{\mathbf{i}}\mathrm{DIVU}_{\mathbf{i}} \tag{21.123}$$

Apply boundary conditions on s_L^x and s_R^x. Then,

$$\mathrm{s}^x_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} s_L^x, & \mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s_R^x, & \text{else.} \end{cases} \tag{21.124}$$

Then, if $|\mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}| \le \epsilon$, we set $s^x_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = (s_L^x + s_R^x)/2$. The procedure to obtain $s^y_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}$ is analogous.

Now, the transverse terms.

If ICONSERVE **then:**

$$\begin{aligned} \mathrm{sedge}_L^x = {} & s_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{h_x}\mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2}\mathrm{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \\ & - \frac{\Delta t}{2}\left[\frac{\mathrm{VEDGE}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} s^y_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - \mathrm{VEDGE}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y} s^y_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y}}{h_y}\right. \\ & \left. - \frac{s_{\mathbf{i}-\mathbf{e}_x}(\mathrm{VEDGE}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - \mathrm{VEDGE}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y})}{h_y} + s_{\mathbf{i}-\mathbf{e}_x}\mathrm{DIVU}_{\mathbf{i}-\mathbf{e}_x}\right] \end{aligned} \tag{21.125}$$

$$\begin{aligned} \mathrm{sedge}_R^x = {} & s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{h_x}\mathrm{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2}\mathrm{TFORCES}_{\mathbf{i}} \\ & - \frac{\Delta t}{2}\left[\frac{\mathrm{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} s^y_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - \mathrm{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} s^y_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y}}{h_y}\right. \\ & \left. - \frac{s_{\mathbf{i}}(\mathrm{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - \mathrm{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})}{h_y} + s_{\mathbf{i}}\mathrm{DIVU}_{\mathbf{i}}\right] \end{aligned} \tag{21.126}$$

Now, define $\mathrm{VBAR}_{\mathbf{i}} = (\mathrm{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + \mathrm{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})/2$.

If NOT ICONSERVE **and** $\text{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} \cdot \text{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0$ **and** $\text{VBAR}_{\mathbf{i}} < 0$ **then:**

$$\begin{aligned} \text{sedge}_L^x = s_{\mathbf{i}-\mathbf{e}_x} \quad + \quad & \left(\frac{1}{2} - \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \\ & -\frac{\Delta t}{2}\left[\frac{\text{VBAR}_{\mathbf{i}-\mathbf{e}_x}(s_{\mathbf{i}-\mathbf{e}_x+\mathbf{e}_y} - s_{\mathbf{i}-\mathbf{e}_x})}{h_y}\right] \end{aligned} \tag{21.127}$$

$$\begin{aligned} \text{sedge}_R^x = s_{\mathbf{i}} \quad - \quad & \left(\frac{1}{2} + \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}} \\ & -\frac{\Delta t}{2}\left[\frac{\text{VBAR}_{\mathbf{i}}(s_{\mathbf{i}+\mathbf{e}_y} - s_{\mathbf{i}})}{h_y}\right] \end{aligned} \tag{21.128}$$

Else If NOT ICONSERVE **and** $\text{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} \cdot \text{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} < 0$ **and** $\text{VBAR}_{\mathbf{i}} \geq 0$ **then:**

$$\begin{aligned} \text{sedge}_L^x = s_{\mathbf{i}-\mathbf{e}_x} \quad + \quad & \left(\frac{1}{2} - \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \\ & -\frac{\Delta t}{2}\left[\frac{\text{VBAR}_{\mathbf{i}-\mathbf{e}_x}(s_{\mathbf{i}-\mathbf{e}_x} - s_{\mathbf{i}-\mathbf{e}_x-\mathbf{e}_y})}{h_y}\right] \end{aligned} \tag{21.129}$$

$$\begin{aligned} \text{sedge}_R^x = s_{\mathbf{i}} \quad - \quad & \left(\frac{1}{2} + \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}} \\ & -\frac{\Delta t}{2}\left[\frac{\text{VBAR}_{\mathbf{i}}(s_{\mathbf{i}} - s_{\mathbf{i}-\mathbf{e}_y})}{h_y}\right] \end{aligned} \tag{21.130}$$

Else If NOT ICONSERVE **and** $\text{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} \cdot \text{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y} \geq 0$ **then:**

$$\begin{aligned} \text{sedge}_L^x \quad = \quad & s_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \\ & -\frac{\Delta t}{2}\left[\frac{(\text{VEDGE}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + \text{VEDGE}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y})(s_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y})}{2h_y}\right] \end{aligned} \tag{21.131}$$

$$\begin{aligned} \text{sedge}_R^x \quad = \quad & s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{h_x}\text{UEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}\right)\Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2}\text{TFORCES}_{\mathbf{i}} \\ & -\frac{\Delta t}{2}\left[\frac{(\text{VEDGE}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + \text{VEDGE}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})(s_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})}{2h_y}\right] \end{aligned} \tag{21.132}$$

Finally, upwind analogous to equation (21.124) to get $\text{sedge}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$.

ESTATE in GODUNOV_2D/3D.f

First, the normal predictor.

$$s_L^x = s_{\mathbf{i}-\mathbf{e}_x} + \left(\frac{1}{2} - \frac{\Delta t}{h_x} u_{\mathbf{i}-\mathbf{e}_x}\right) \Delta^x s_{\mathbf{i}-\mathbf{e}_x} \tag{21.133}$$

$$s_R^x = s_{\mathbf{i}} - \left(\frac{1}{2} + \frac{\Delta t}{h_x} u_{\mathbf{i}}\right) \Delta^x s_{\mathbf{i}} \tag{21.134}$$

If USE_MINION **then:**

$$s_L^x = s_L^x + \frac{\Delta t}{2} \text{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \tag{21.135}$$

$$s_R^x = s_R^x + \frac{\Delta t}{2} \text{TFORCES}_{\mathbf{i}} \tag{21.136}$$

Apply boundary conditions on s_L^x and s_R^x. Then,

$$\text{s}^x_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = \begin{cases} s_L^x, & \text{UAD}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} > 0, \\ s_R^x, & \text{else.} \end{cases} \tag{21.137}$$

Then, if $|\text{UAD}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}| \leq \epsilon$, we set $s^x_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x} = (s_L^x + s_R^x)/2$.

$$\begin{aligned} \text{sedge}_L^x = s_{\mathbf{i}-\mathbf{e}_x} \quad + \quad & \left(\frac{1}{2} - \frac{\Delta t}{h_x} u_{\mathbf{i}-\mathbf{e}_x}\right) \Delta^x s_{\mathbf{i}-\mathbf{e}_x} + \frac{\Delta t}{2} \text{TFORCES}_{\mathbf{i}-\mathbf{e}_x} \\ & - \frac{\Delta t}{2} \left[\frac{(\text{VAD}_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} + \text{VAD}_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y})(s_{\mathbf{i}-\mathbf{e}_x+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\mathbf{e}_x-\frac{1}{2}\mathbf{e}_y})}{2h_y} \right] \end{aligned} \tag{21.138}$$

$$\begin{aligned} \text{sedge}_R^x = s_{\mathbf{i}} \quad - \quad & \left(\frac{1}{2} + \frac{\Delta t}{h_x} u_{\mathbf{i}}\right) \Delta^x s_{\mathbf{i}} + \frac{\Delta t}{2} \text{TFORCES}_{\mathbf{i}} \\ & - \frac{\Delta t}{2} \left[\frac{(\text{VAD}_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} + \text{VAD}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})(s_{\mathbf{i}+\frac{1}{2}\mathbf{e}_y} - s_{\mathbf{i}-\frac{1}{2}\mathbf{e}_y})}{2h_y} \right] \end{aligned} \tag{21.139}$$

Note that the 2D and 3D algorithms are different - in 3D the transverse terms use upwinding analogous to equations (21.127)-(21.132), using UAD instead of UEDGE. Finally, upwind analogous to equation (21.137) to get $\text{sedge}_{\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, but use UEDGE instead of UAD.

Piecewise Parabolic Method (PPM)

Consider a scalar, s, which we wish to predict to time-centered edges. The PPM method is an improvement over the piecewise-linear method. Using our notation, we modify equations (21.73) and (21.74) in Section 21.4 to obtain better estimates for the time-centered 1D edge states, $s_{L/R,\mathbf{i}-\frac{1}{2}\mathbf{e}_x}$, etc.. Once these states are obtained, we continue with the full-dimensional extrapolations as described before.

The PPM method is described in a series of papers:

- Colella and Woodward 1984 - describes the basic method.
- Miller and Colella 2002 - describes how to apply PPM to a multidimensional unsplit Godunov method and generalizes the characteristic tracing for more complicated systems. Note that we only need to upwind based on the fluid velocity, so we don't need to use fancy characteristic tracing.
- Colella and Sekora 2008 - describes new fancy quadratic limiters. There are several errors in the printed text, so we have implemented a corrected version from Phil Colella.

Here are the steps for the x-direction. For simplicity, we replace the vector index notation with a simple scalar notation ($\mathbf{i} + \mathbf{e}_x \rightarrow i + 1$, etc.).

- **Step 1:** Compute $s_{i,+}$ and $s_{i,-}$, which are spatial interpolations of s to the hi and lo faces of cell i, respectively. See Sections 21.9.1 and 21.9.2 for the two options.
- **Step 2:** Construct a quadratic profile within each cell.

$$s_i^I(x) = s_{i,-} + \xi\left[s_{i,+} - s_{i,-} + s_{6,i}(1-\xi)\right], \tag{21.140}$$

$$s_{6,i} = 6s_i - 3\left(s_{i,-} + s_{i,+}\right), \tag{21.141}$$

$$\xi = \frac{x - (i - 1/2)h}{h},\ 0 \le \xi \le 1. \tag{21.142}$$

- **Step 3:** Integrate quadratic profiles to get the average value swept over the face over time.

  Define the following integrals, where $\sigma = |u|\Delta t/h$:

$$\mathcal{I}_{i,+}(\sigma) = \frac{1}{\sigma h}\int_{(i+1/2)h-\sigma h}^{(i+1/2)h} s_i^I(x)dx \tag{21.143}$$

$$\mathcal{I}_{i,-}(\sigma) = \frac{1}{\sigma h}\int_{(i-1/2)h}^{(i-1/2)h+\sigma h} s_i^I(x)dx \tag{21.144}$$

  Plugging in (21.140) gives:

$$\mathcal{I}_{i,+}(\sigma) = s_{j,+} - \frac{\sigma}{2}\left[s_{j,+} - s_{j,-} - \left(1 - \frac{2}{3}\sigma\right)s_{6,i}\right], \tag{21.145}$$

$$\mathcal{I}_{i,-}(\sigma) = s_{j,-} + \frac{\sigma}{2}\left[s_{j,+} - s_{j,-} + \left(1 - \frac{2}{3}\sigma\right)s_{6,i}\right]. \tag{21.146}$$

- **Step 4:** Obtain 1D edge states.

Perform a 1D extrapolation, without source terms, to get left and right edge states. Add the source terms later if desired/necessary.

$$s_{L,i-1/2} = \begin{cases} \mathcal{I}_{i-1,+}(\sigma), & u_{i-1} \text{ or } u^{\text{MAC}}_{i-1/2} > 0 \\ s_{i-1}, & \text{else.} \end{cases} \tag{21.147}$$

$$s_{R,i-1/2} = \begin{cases} \mathcal{I}_{i,-}(\sigma), & u_i \text{ or } u^{\text{MAC}}_{i-1/2} < 0 \\ s_i, & \text{else.} \end{cases} \tag{21.148}$$

Colella and Woodward Based Approach

Spatially interpolate s to edges. Use a 4th-order interpolation in space with van Leer limiting to obtain edge values:

$$s_{i+1/2} = \frac{1}{2}\left(s_i + s_{i+1}\right) - \frac{1}{6}\left(\delta s^{vL}_{i+1} - \delta s^{vL}_i\right), \tag{21.149}$$

$$\delta s_i = \frac{1}{2}\left(s_{i+1} - s_{i-1}\right), \tag{21.150}$$

$$\delta s^{vL}_i = \begin{cases} \text{sign}(\delta s_i)\min\left(|\delta s_i|,\, 2|s_{i+1} - s_i|,\, 2|s_i - s_{i-1}|\right), & \text{if } (s_{i+1} - s_i)(s_i - s_{i-1}) > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{21.151}$$

A more compact way of writing this is

$$s = \text{sign}(\delta s_i), \tag{21.152}$$

$$\delta s^{vL}_i = s\max\left\{\min\left[s\delta s_i, 2s(s_{i+1} - s_i), 2s(s_i - s_{i-1})\right], 0\right\} \tag{21.153}$$

Without the limiters, equation (21.149) is the familiar 4th-order spatial interpolation formula:

$$s_{i+1/2} = \frac{7}{12}\left(s_{i+1} + s_i\right) - \frac{1}{12}\left(s_{i+2} + s_{i-1}\right). \tag{21.154}$$

Next, we must ensure that $s_{i+1/2}$ lies between the adjacent cell-centered values:

$$\min\left(s_i, s_{i+1}\right) \le s_{i+1/2} \le \max\left(s_i, s_{i+1}\right). \tag{21.155}$$

In anticipation of further limiting, we set double-valued face-centered values:

$$s_{i,+} = s_{i+1,-} = s_{i+1/2}. \tag{21.156}$$

Modify $s_{i,\pm}$ using a quadratic limiter. First, we test whether s_i is a local extreumum with the condition:

$$\left(s_{i,+} - s_i\right)\left(s_i - s_{i,-}\right) \le 0, \tag{21.157}$$

If this condition is true, we constrain $s_{i,\pm}$ by setting $s_{i,+} = s_{i,-} = s_i$. If not, we then apply a second test to determine whether s_i is sufficiently close to $s_{i,\pm}$ so that a quadratic interpolate would contain a local extremum. We define $\alpha_{i,\pm} = s_{i,\pm} - s_i$. If one of $|\alpha_{i,\pm}| \ge 2|\alpha_{i,\mp}|$ holds, then for that choice of $\pm = +, -$ we set:

$$s_{i,\pm} = 3s_i - 2s_{i,\mp}. \tag{21.158}$$

Colella and Sekora Based Approach

Spatially interpolate s to edges. Use a 4th-order interpolation in space to obtain edge values:

$$s_{i+1/2} = \frac{7}{12}\left(s_{i+1} + s_i\right) - \frac{1}{12}\left(s_{i+2} + s_{i-1}\right). \tag{21.159}$$

Then, if $(s_{i+1/2} - s_i)(s_{i+1} - s_{i+1/2}) < 0$, we limit $s_{i+1/2}$ using a nonlinear combination of approximations to the second derivative. First, define:

$$\begin{aligned}
\left(D^2 s\right)_{i+1/2} &= 3\left(s_i - 2s_{i+1/2} + s_{i+1}\right) && (21.160)\\
\left(D^2 s\right)_{i+1/2,L} &= s_{i-1} - 2s_i + s_{i+1} && (21.161)\\
\left(D^2 s\right)_{i+1/2,R} &= s_i - 2s_{i+1} + s_{i+2} && (21.162)
\end{aligned}$$

Then, define

$$s = \text{sign}\left[\left(D^2 s\right)_{i+1/2}\right], \tag{21.163}$$

$$\left(D^2 s\right)_{i+1/2,\text{lim}} = s \max\left\{\min\left[Cs(D^2 s)_{i+1/2,L}, Cs(D^2 s)_{i+1/2,R}, s(D^2 s)_{i+1/2}\right], 0\right\}, \tag{21.164}$$

where $C = 1.25$ was used in Colella and Sekora. Then,

$$s_{i+1/2} = \frac{1}{2}\left(s_i + s_{i+1}\right) - \frac{1}{6}\left(D^2 s\right)_{i+1/2,\text{lim}}. \tag{21.165}$$

Now we implement Phil's new version of the algorithm to eliminate sensitivity to roundoff. First we need to detect whether a particular cell corresponds to an "extremum". There are two tests. For the first test, define

$$\alpha_{i,\pm} = s_{i\pm 1/2} - s_i. \tag{21.166}$$

If $\alpha_{i,+}\alpha_{i,-} \geq 0$, then we are at an extremum. We apply the second test if either $|\alpha_{i,\pm}| > 2|\alpha_{i,\mp}|$. Then, we define:

$$\begin{aligned}
(Ds)_{i,\text{face},-} &= s_{i-1/2} - s_{i-3/2} && (21.167)\\
(Ds)_{i,\text{face},+} &= s_{i+3/2} - s_{i+1/2} && (21.168)
\end{aligned}$$

$$(Ds)_{i,\text{face,min}} = \min\left[\left|(Ds)_{i,\text{face},-}\right|, \left|(Ds)_{i,\text{face},+}\right|\right]. \tag{21.169}$$

$$\begin{aligned}
(Ds)_{i,\text{cc},-} &= s_i - s_{i-1} && (21.170)\\
(Ds)_{i,\text{cc},+} &= s_{i+1} - s_i && (21.171)
\end{aligned}$$

$$(Ds)_{i,\text{cc,min}} = \min\left[\left|(Ds)_{i,\text{cc},-}\right|, \left|(Ds)_{i,\text{cc},+}\right|\right]. \tag{21.172}$$

If $(Ds)_{i,\text{face,min}} \geq (Ds)_{i,\text{cc,min}}$, set $(Ds)_{i,\pm} = (Ds)_{i,\text{face},\pm}$. Otherwise, set $(Ds)_{i,\pm} = (Ds)_{i,\text{cc},\pm}$. Finally, we are at an extreumum if $(Ds)_{i,+}(Ds)_{i,-} \leq 0$.

Now that we have finished the extremum tests, if we are at an extremum, we scale $\alpha_{i,\pm}$. First, we define

$$\begin{aligned}
\left(D^2 s\right)_i &= 6(\alpha_{i,+} + \alpha_{i,-}) && (21.173)\\
\left(D^2 s\right)_{i,L} &= s_{i-2} - 2s_{i-1} + s_i && (21.174)\\
\left(D^2 s\right)_{i,R} &= s_i - 2s_{i+1} + s_{i+2} && (21.175)\\
\left(D^2 s\right)_{i,C} &= s_{i-1} - 2s_i + s_{i+1} && (21.176)
\end{aligned}$$

Then, define

$$s = \text{sign}\left[(D^2 s)_i\right], \tag{21.177}$$

$$(D^2 s)_{i,\text{lim}} = \max\left\{\min\left[s(D^2 s)_i, Cs(D^2 s)_{i,L}, Cs(D^2 s)_{i,R}, Cs(D^2 s)_{i,C}\right], 0\right\}. \tag{21.178}$$

Then,

$$\alpha_{i,\pm} = \frac{\alpha_{i,\pm}(D^2 s)_{i,\text{lim}}}{\max\left[\left|(D^2 s)_i\right|, 1 \times 10^{-10}\right]} \tag{21.179}$$

Otherwise, if we are not at an extremum and $|\alpha_{i,\pm}| > 2|\alpha_{i,\mp}|$, then define

$$s = \text{sign}(\alpha_{i,\mp}) \tag{21.180}$$

$$\delta\mathcal{I}_{\text{ext}} = \frac{-\alpha_{i,\pm}^2}{4\left(\alpha_{j,+} + \alpha_{j,-}\right)}, \tag{21.181}$$

$$\delta s = s_{i\mp 1} - s_i, \tag{21.182}$$

If $s\delta\mathcal{I}_{\text{ext}} \geq s\delta s$, then we perform the following test. If $s\delta s - \alpha_{i,\mp} \geq 1 \times 10^{-10}$, then

$$\alpha_{i,\pm} = -2\delta s - 2s\left[(\delta s)^2 - \delta s \alpha_{i,\mp}\right]^{1/2} \tag{21.183}$$

otherwise,

$$\alpha_{i,\pm} = -2\alpha_{i,\mp} \tag{21.184}$$

Finally, $s_{i,\pm} = s_i + \alpha_{i,\pm}$.

CHAPTER 22

Multigrid

AMReX Multigrid Philosophy

Here we describe some of the general ideas behind the AMReX multigrid (MG).

We solve MG on an AMR hierarchy, which means in places we will encounter C-F interfaces. The AMReX MG will modify the stencil for the Laplacian at C-F interfaces to ensure the correct solution to the Poisson equation across these interfaces. There is no correction step needed in this case (this differs from [31]).

The MG solver always works with jumps of $2\times$ between levels. In some codes (but not MAESTRO) we can have jumps of $4\times$ between levels. AMReX uses a `mini_cycle` in these cases, effectively inserting a multigrid level between the two AMR levels and doing a mini V-cycle.

The MG solvers are located in `amrex/Src/LinearSolvers/F_MG/`. There are two MG solvers, for cell-centered and nodal data. Generally, the routines specific to the cell-centered solver will have `cc` in their name, and those specific to the nodal solver will have `nd` in their name.

Support for $\Delta x \neq \Delta y \neq \Delta z$

Data Structures

`mg_tower`

The `mg_tower` is a special Fortran derived type that carries all the information required for a AMReX multigrid solve.

The following parameters are specified when building the `mg_tower`:

- smoother: the type of smoother to use. Choices are listed in mg_tower.f90. Common options are MG_SMOOTHER_GS_RB for red-black Gauss-Seidel and MG_SMOOTHER_JACOBI for Jacobi.
- nu1: The number of smoothings at each level on the way down the V-cycle.
- nu2: The number of smoothings at each level on the way up the V-cycle.
- nub: The number of smoothing before and after the bottom solver.

what does this mean?

- gamma:
- cycle_type: The type of multigrid to do, V-cycles (MG_VCycle), W-cycles (MG_WCycle), full multigrid (MG_FCycle).

escription?

- omega:
- bottom_solver: the type of bottom solver to use. See the next section.
- bottom_max_iter: the maximum number of iterations for the bottom solver
- bottom_solver_eps: the tolerance used by the bottom solver. In MAESTRO, this is set via mg_eps_module.
- max_iter: the maximum number of multigrid cycles.
- max_bottom_nlevel: additional coarsening if you use bottom_solver type 4 (see below)
- min_width: minimum size of grid at coarsest multigrid level
- rel_solver_eps: the relative tolerance of the solver (in MAESTRO, this is set via mg_eps_module.
- abs_solver_eps: the absolute tolerance of the solver (in MAESTRO, this is set via mg_eps_module.
- verbose: the verbosity of the multigrid solver. In MAESTRO, this is set via the mg_verbose runtime parameter. Higher numbers give more verbosity.
- cg_verbose: the verbosity of the bottom solver. In MAESTRO, this is set via the cg_verbose runtime parameter. Higher numbers give more verbosity.

In addition to these parameters, the mg_tower carries a number of multifabs that carry the solution and stencil for the multigrid solve.

- ss: The stencil itself—for each zone, this gives the coefficients of the terms in the Laplacian, with the convection that the '0' term is located in the current cell and the other terms are the ± 1 off the current cell in each direction.
- cc: scratch space (?)
- ff: The source (righthand side) of the elliptic equation we are solving.
- dd: The residual/defect, $f - L\phi$
- uu: The solution variable (ϕ when on the finest level)
- mm: For cell-centered, mm takes a direction and tells you whether we are at a Dirichlet or Neumann boundary, or if we are skewed in that direction.

  For nodal, mm simply tells us whether a point is Dirichlet or Neumann. There is no skew in nodal.

`bndry_reg`

There are two types of `bndry_reg` objects, each of which is a set of `dim` multifabs. The first, which is built with the `bndry_reg_build` call, is defined on all cells immediately outside of each grid. Each multifab within the `bndry_reg` contains all the fabs on both the low and high faces for a given direction. In the context of multigrid, we fill the `bndry_reg` with coarse data before interpolating the data to the correct locations to be used in the multigrid stencil. The second type of `bndry_reg` object is defined on the coarse cells immediately outside each fine grid, and is defined with the `bndry_reg_rr_build` call, is defined on all cells immediately outside. For this latter type, the option is available to include only cells not covered by a different fine grid, but this is left as an option because it requires additional calculations of box intersections.

In multigrid, we use the latter type of `bndry_reg` to calculate the residual at coarse cells adjacent to fine grids that is used as the right hand side in the relaxation step at the coarse level on the way down the V-cycle (or other). The first type of `bndry_reg`is used to hold boundary conditions for the fine grids that are interpolated from the coarse grid solution; this is filled on the way back up the V-cycle.

To compute the residual in a coarse cell adjacent to a fine grid, we first compute the pure coarse residual, then subtract the contribution from the coarse face underlying the coarse-fine interface, and add the contribution of the fine faces at the coarse-fine interface. The `bndry_reg` holds the cell-centered contribution from the difference of these edge fluxes and is added to the coarse residual multifab.

Stencils

There are several different stencil types that we can use for the discretization. For cell-centered, these are:

- `CC_CROSS_STENCIL`: this is the standard cross-stencil—5 points in 2-d and 7 points in 3-d. For cell-centered MG, this is the default, and is usually the best choice.
- `HO_CROSS_STENCIL`: this is a cross-stencil that uses 9 points in 2-d and 11-points in 3-d. For instance, it will use (i, j); $(i \pm 1, j)$; $(i \pm 2, j)$; $(i, j \pm 1)$; $(i, j \pm 2)$. This is higher-order accurate that `CC_CROSS_STENCIL`.
- `HO_DENSE_STENCIL`: this is a dense-stencil—it uses all the points (including corners) in $(i \pm 1, j \pm 1)$, resulting in a 9-point stencil in 2-d and 27-point stencil in 3-d.

For the nodal solver, the choices are:

- `ND_CROSS_STENCIL`: this is the standard cross-stencil.
- `ND_DENSE_STENCIL`: this is a dense stencil, using all the points in $(i \pm 1, j \pm 1)$. The derivation of this stencil is based on finite-element ideas, defining basis functions on the nodes. This is developed in 2-d in [7].
- `ND_VATER_STENCIL`: this is an alternate dense stencil derived using a similar finite-element idea as above, but a different control volume.

For the cell-centered solve, the coefficients for the stencil are computed once, at the beginning of the solve. For the nodal solver, the coefficients are hard-coded into the smoothers.

Smoothers

The following smoothers are available (but not necessarily for both the cell-centered and nodal solvers):

- `MG_SMOOTHER_GS_RB`: a red-black Gauss-Seidel smoother
- `MG_SMOOTHER_JACOBI`: a Jacobi smoother (not implemented for the dense nodal stencil)
- `MG_SMOOTHER_MINION_CROSS`
- `MG_SMOOTHER_MINION_FULL`
- `MG_SMOOTHER_EFF_RB`

Cycling

The default cycling is a V-cycle, but W-cycles and full multigrid are supported as well.

Bottom Solvers

The multigrid cycling coarsens the grids as part of the solve. When the coarsest grid is reached, the individual boxes that comprise that level are coarsened as much as then can, down to 2^3 zones. Depending on the distribution of sizes of the grids, it may not be possible for everything to reach this minimum size. At this point, the bottom solver is invoked. Most of these will solve the linear system on this collection of grids directly. There is one special bottom solver that will define a new box encompassing all of the coarsened grids and then put the data on fewer boxes and processors and further coarsen the problem, again until we get as close to 2^3 as possible. At that point, one of the other bottom solvers will be called upon to solve the problem.

There are several bottom solvers available in AMReX. For MAESTRO. These are set through the `mg_bottom_solver` (MAC/cell-centered) and `hg_bottom_solver` (nodal) runtime parameters. The allowed values are:

- `mg_bottom_solver` / `hg_bottom_solver = 0`: smoothing only.
- `mg_bottom_solver` / `hg_bottom_solver = 1`: biconjugate gradient stabilized—this is the default.
- `mg_bottom_solver` / `hg_bottom_solver = 2`: conjugate gradient method
- `mg_bottom_solver` / `hg_bottom_solver = 4`: a special bottom solver that extends the range of the multigrid coarsening by aggregrating coarse grids on the original mesh together and further coarsening.

any simple discussion on why we might choose one of the other bottom solvers?

You should use the special bottom solver (4) whenever possible, even if it means changing your gridding strategy (as discussed below) to make it more efficient.

Special Bottom Solver

The special solver takes the data from the coarsest level of the original multigrid V-cycle and copies it onto a new grid structure with the same number of total cells in each direction, but with a fewer

number of larger grids. A new V-cycle begins from this point, so we are essentially coarsening this "new" problem. Now, the coarsest level of the multigrid V-cycle in the "new" problem has fewer cells and fewer grids as compared to the original coarsest level.

To enable this solver, set `hg_bottom_solver = 4` (for the nodal projections) and/or `mg_bottom_solver = 4` (for the cell-centered projections) in your inputs file.

To understand how this bottom solver works, the first thing you need to know is what the grid structure of the coarsest level of your multigrid V-cycle looks like. Next, figure out the size of the box you would need if you wanted it to fit all the data on the coarsest level. Finally, figure out what the largest integer n is so that you can evenly divide the length of this box by 2^n in every coordinate direction. If $n < 2$, the program will abort since the grid structure is not suitable for this bottom solver.

The code will set up a "new" problem, using the data at the coarsest level of the original problem as the initial data. The grid structure for this new problem has the same number of cells as the coarsest level of the original problem, but the data is copied onto a grid structure where each grid has 2^n cells on each side. The new V-cycle continues down to the new coarsest level, in which each grid has 2 cells on each side. If you wish to impose a limit on the maximum value that n can have, you can do so by setting `max_mg_bottom_nlevs` equal to that value.

Some grid examples help make this clear:

- **Example 1:** A 3D problem with 384^3 cells divided into 32^3 grids, i.e., there is a $12 \times 12 \times 12$ block of 32^3 grids. The coarsest level of the multigrid V-cycle contains $12 \times 12 \times 12$ grids that have 2^3 cells, so the entire problem domain has 24^3 cells. We see that $n = 3$, and create a new problem domain with a $3 \times 3 \times 3$ block of 8^3 grids. The coarsest level of the multigrid V-cycle for the "new" problem will be a $3 \times 3 \times 3$ block of 2^3 grids.
- **Example 2:** A 2D problem with 96×384 cells divided into 48^2 grids, i.e., there is a 2×8 block of 48^2 grids. The coarsest level of the multigrid V-cycle contains 2×8 grids that have 3^2 cells, so the entire problem domain has 6×24 cells. We see that $n = 0$, so the program aborts since this grid structure is not appropriate for the fancy bottom solver.

Flowchart

MAESTRO multigrid solves always involve the full AMR hierarchy.

Cell-Centered MG

The flowchart below shows the structure of a cell-centered multigrid solve using pure V-cycles.

- `stencil_fill_cc_all_mglevels` / `stencil_fill_cc`: Compute all of the stencil coefficients for the Laplacian operator at all cells. At the C-F interfaces, the stencil coefficients are modified to know this.
- `ml_cc`: The main driver for the cell-centered multigrid. Among other things, this computes the norm that will be used for convergence testing.
- `mg_tower_v_cycle` (recursive):
  - *recursively descend V-cycle*

        * `cc_mg_tower_smoother`: Smooth the problem at the current MG level using the desired smoother.
        * `compute_defect`: Construct $f - L\phi$.
        * `mg_tower_restriction`: Restrict the defect to the coarser level by conservative averaging.
    - `mg_tower_bottom_solve`: Solve the coarsened problem using the chosen bottom solver.
    - *ascend V-cycle*
        * `mg_tower_prolongation`: Take the solution at level $n-1$ and use it to correct the solution at level n by representing the data on the finer grid. This uses linear reconstruction for jumps by $2\times$ and piecewise-constant otherwise.
        * `cc_mg_tower_smoother`:
- `compute_defect`: This is called multiple times, checking for convergence at each level.

Nodal MG

The flowchart below shows the structure of a cell-centered multigrid solve using pure V-cycles.

odal does things ifferently than c—need more here

- `stencil_fill_cc_all_mglevels` / `stencil_fill_cc`: For the nodal solver, this applies the weights to the coefficients.
- `ml_nd`: The main driver for the nodal multigrid.
- `mg_tower_v_cycle` (recursive):
    - *recursively descend V-cycle*
        * `nodal_smoother_?d`: Smooth the problem at the current MG level using the desired smoother.
        * `compute_defect`: Construct $f - L\phi$.
        * `mg_tower_restriction`: Restrict the defect to the coarser level by simply taking the fine value that lies at the same place as the coarse data.
    - `mg_tower_bottom_solve`: Solve the coarsened problem using the chosen bottom solver.
    - *ascend V-cycle*
        * `mg_tower_prolongation`: For nodal data, the fine grid will have some points at exactly the same place as the coarse data—these are simply copied to the fine grid. The remain data is interpolated.
        * `modal_smoother_?d`:
- `compute_defect`: This is called multiple times, checking for convergence at each level.

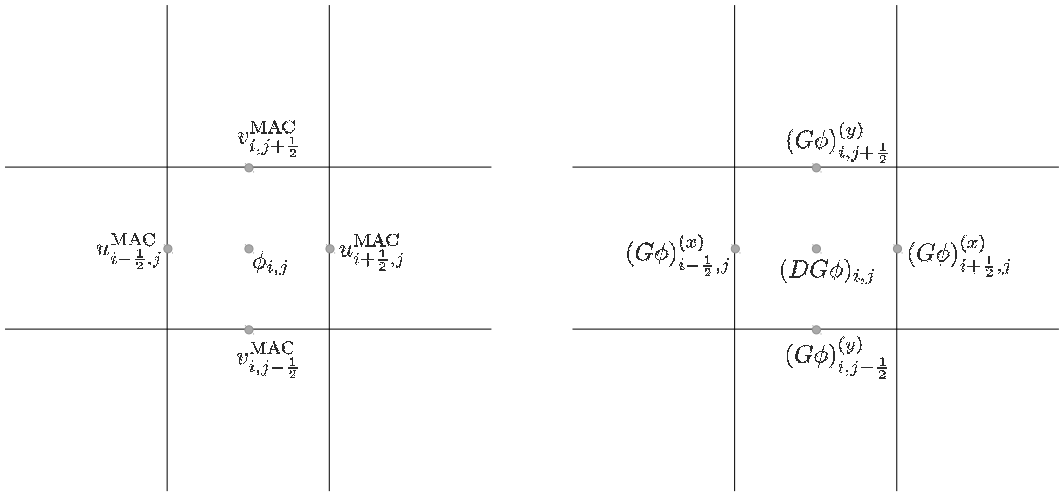


Figure 22.1: Data centerings for the MAC projection

MAESTRO's Multigrid Use

MAESTRO uses multigrid to enforce the velocity constraint through projections at the half-time (the MAC projection) and end of the time step (the HG projection). Two multigrid solvers are provided by AMReX—one for cell-centered data and one for node-centered (nodal) data. Both of these are used in MAESTRO.

The MAC projection operates on the advective velocities predicted at the cell-interfaces at the half-time. The edge-centered velocities are shown in Figure 22.1. If we consider purely incompressible flow, the projection appears as:

$$DG\phi = DU \tag{22.1}$$

where D is the divergence operator and G is the gradient operator. In this discretization, ϕ is cell-centered (see Figure 22.1). The remaining quantities are discretized as:

- DU is cell-centered,

$$(DU)_{i,j} = \frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{\Delta y} \tag{22.2}$$

- $G\phi$ is edge-centered, on the MAC grid, as shown in Figure 22.1.
- $DG\phi$ is cell-centered, also shown in Figure 22.1, computed from $G\phi$ using the same differencing as DU.

The HG projection projects the cell-centered velocities at the end of the timestep. Here, ϕ is node-centered. Figure 22.2 shows the locations of the various quantities involved in the HG projection. Again considering simple incompressible flow, we now solve:

$$L\phi = DU \tag{22.3}$$

where L is a discretization of the Laplacian operator. In this sense, the HG projection is an *approximate projection*, that is, $L \neq DG$ (in discretized form). The various operations have the following centerings:

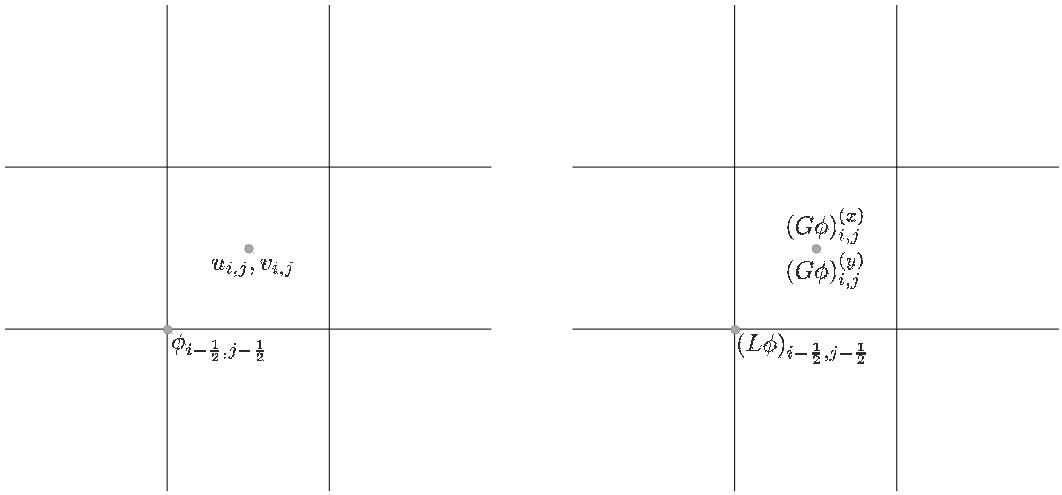


Figure 22.2: Data centerings for the HG projection

- DU is node-centered. This is computed as:

$$(DU)_{i-1/2,j-1/2} = \frac{\frac{1}{2}(u_{i,j} + u_{i,j-1}) - \frac{1}{2}(u_{i-1,j} + u_{i-1,j-1})}{\Delta x} + \frac{\frac{1}{2}(v_{i,j} + v_{i-1,j}) - \frac{1}{2}(v_{i,j-1} + v_{i-1,j-1})}{\Delta y} \tag{22.4}$$

- $G\phi$ is cell-centered, as shown in Figure 22.2.
- $L\phi$ is node-centered. This is a direct discretization of the Laplacian operator. By default, MAESTRO uses a dense stencil (9-points in 2-d, 27-points in 3-d). Alternately, a *cross* stencil can be used (by setting `hg_dense_stencil = F`). This uses 5-points in 2-d, 7-points in 3-d.

Convergence Criteria

All MAESTRO multigrid solves consist of pure V-cycles.

Multigrid Solver Tolerances

Beginning at the start of execution, there are several places where either cell-centered multigrid or node-centered multigrid solves are performed. The outline below lists the solves one encounters, in order, from the start of execution. The values of the tolerances lists here are defined in the `mg_eps` module. To set problem-specific values of these tolerances, place a local copy of `mg_eps.f90` in your problem directory.

In the initialization, multigrid comes in during the initial projection and the "divu" iterations.

- *initial projection* (`initial_proj` called from `varden`)

  The initial projection creates a first approximation to the velocity field by forcing the initial velocity field set by `initveldata` to satisfy the elliptic constraint equation. Since the initial velocity may be zero, there is no guarantee that a well-defined timestep can be computed at this point,

so the source term, S, used here only involves thermal diffusion and any external heating term, $H_{\rm ext}$—no reactions are included (see paper III, §3.3).

The initial projection can be disabled with the `do_initial_projection` runtime parameter.

The tolerances, `eps_init_proj_cart` and `eps_init_proj_sph` (for Cartesian and spherical respectively) are set in `mg_eps.f90` and have the default values of:

$$\begin{array}{ll} \text{Cartesian:} & \texttt{eps\_init\_proj\_cart} = 10^{-12} \\ \text{spherical:} & \texttt{eps\_init\_proj\_sph} = 10^{-10} \end{array}$$

- *"divu" iterations* (`divu_iter` called from `varden`)

The "divu" iterations projects the velocity field from the initial projection to satisfy the full constraint (including reactions). This is an iterative process since the reactions depend on the timestep and the timestep depends on the velocity field (see paper III, §3.3). The number of iterations to take is set through the `init_divu_iter` runtime parameter.

The overall tolerance, $\epsilon_{\rm divu}$ depends on the iteration, i. We start with a loose tolerance and progressively get tighter. The tolerances (set in `divu_iter`) are, for Cartesian:

$$\epsilon_{\rm divu} = \begin{cases} \min\{\ \texttt{eps\_divu\_cart} \cdot \texttt{divu\_iter\_factor}^2 \cdot \texttt{divu\_level\_factor}^{(\texttt{nlevs}-1)}, \\ \qquad \texttt{eps\_divu\_cart} \cdot \texttt{divu\_iter\_factor}^2 \cdot \texttt{divu\_level\_factor}^2\ \} & \text{for } i \le \texttt{init\_divu\_iter} - 2 \\ \min\{\ \texttt{eps\_divu\_cart} \cdot \texttt{divu\_iter\_factor} \cdot \texttt{divu\_level\_factor}^{(\texttt{nlevs}-1)}, \\ \qquad \texttt{eps\_divu\_cart} \cdot \texttt{divu\_iter\_factor} \cdot \texttt{divu\_level\_factor}^2\ \} & \text{for } i = \texttt{init\_divu\_iter} - 1 \\ \min\{\ \texttt{eps\_divu\_cart} \cdot \texttt{divu\_level\_factor}^{(\texttt{nlevs}-1)}, \\ \qquad \texttt{eps\_divu\_cart} \cdot \texttt{divu\_level\_factor}^2\ \} & \text{for } i = \texttt{init\_divu\_iter} \end{cases}$$

and for spherical:

$$\epsilon_{\rm divu} = \begin{cases} \texttt{eps\_divu\_sph} \cdot \texttt{divu\_iter\_factor}^2 & \text{for } i \le \texttt{init\_divu\_iter} - 2 \\ \texttt{eps\_divu\_sph} \cdot \texttt{divu\_iter\_factor} & \text{for } i = \texttt{init\_divu\_iter} - 1 \\ \texttt{eps\_divu\_sph} & \text{for } i = \texttt{init\_divu\_iter}\ ) \end{cases}$$

The various parameters are set in `mg_eps.f90` and have the default values of:

$$\begin{array}{ll} \texttt{eps\_divu\_cart} & = 10^{-12} \\ \texttt{eps\_divu\_sph} & = 10^{-10} \\ \texttt{divu\_iter\_factor} & = 100 \\ \texttt{divu\_level\_factor} & = 10 \end{array}$$

In the main algorithm, mulitgrid solves come in during the two MAC projections, two (optional) thermal diffusion solves, and the final velocity projection.

- *MAC projection*

The MAC projection forces the edge-centered, half-time advective velocities to obey the elliptic constraint. This is done both in the predictor and corrector portions of the main algorithm.

There are two tolerances here. The norm of the residual is required to be reduced by a relative tolerance of $\epsilon = \min\{\texttt{eps\_mac\_max}, \texttt{eps\_mac} \cdot \texttt{mac\_level\_factor}^{(\texttt{nlevs}-1)}\}$. A separate tolerance is used for the bottom solver, $\epsilon_{\rm bottom} = \texttt{eps\_mac\_bottom}$. These parameters are set in `mg_eps.f90` and have the default values:

eps_mac	$= 10^{-10}$
eps_mac_max	$= 10^{-8}$
mac_level_factor	$= 10$
eps_mac_bottom	$= 10^{-3}$

- *thermal diffusion*

  This uses the same mac_multigrid routine as the MAC projection, so it uses the same tolerances. The only difference is that the absolute tolerance is based on the norm of h now, instead of U^{ADV}.

- *velocity projection*

  The final velocity projection uses a tolerance of $\epsilon = \min\{\texttt{eps\_hg\_max}, \texttt{eps\_hg} \cdot \texttt{hg\_level\_factor}^{(\texttt{nlevs}-1)}\}$. This tolerance is set in hgproject using the parameter values specified in mg_eps.f90. A separate tolerance is used for the bottom solver, $\epsilon_{\mathrm{bottom}} = \texttt{eps\_hg\_bottom}$.

  The default parameter values are:

eps_hg	$= 10^{-12}$
eps_hg_max	$= 10^{-10}$
hg_level_factor	$= 10$
eps_hg_bottom	$= 10^{-4}$

General Remarks

If MAESTRO has trouble converging in the multigrid solves, try setting the verbosity mg_verbose or cg_verbose to higher values to get more information about the solve.

CHAPTER 23

Rotation

Rotation

To handle rotation in MAESTRO we move to the co-rotating reference frame. Time derivatives of a vector in the inertial frame are related to those in the co-rotating frame by

$$\left(\frac{D}{Dt}\right)_{\mathrm{i}} = \left[\left(\frac{D}{Dt}\right)_{\mathrm{rot}} + \mathbf{\Omega}\times\right], \tag{23.1}$$

where i(rot) refers to the inertial(co-rotating) frame and $\mathbf{\Omega}$ is the angular velocity vector. Using (23.1) and assuming $\mathbf{\Omega}$ is constant, we have

$$\frac{Dv_{\mathrm{i}}}{Dt} = \frac{Dv_{\mathrm{rot}}}{Dt} + 2\mathbf{\Omega}\times\mathbf{v}_{\mathrm{rot}} + \mathbf{\Omega}\times(\mathbf{\Omega}\times r_{\mathrm{rot}})\,. \tag{23.2}$$

Plugging this into the momentum equation and making use of the continuity equation we have a momentum equation in the co-rotating frame:

$$\frac{\partial(\rho\mathbf{U})}{\partial t} + \nabla\cdot(\mathbf{U}(\rho\mathbf{U}) + p) = \underbrace{-2\rho\mathbf{\Omega}\times\mathbf{U}}_{\text{Coriolis}} \underbrace{-\rho\mathbf{\Omega}\times(\mathbf{\Omega}\times\mathbf{r})}_{\text{Centrifugal}} - \rho\mathbf{g}. \tag{23.3}$$

The Coriolis and Centrifugal terms are force terms that will be added to right hand side of the equations in `mk_vel_force`. Note that the Centrifugal term can be rewritten as a gradient of a potential and absorbed into either the pressure or gravitational term to create an effective pressure or effective gravitational potential. Furthermore, because it can be written as a gradient, this term would drop out completely in the projection if we were doing incompressible flow. In what follows we will include the Centrifugal term explicitly.

Is this how John stated it?

Using Spherical Geometry

In spherical geometry implementing rotation is straightforward as we don't have to worry about special boundary conditions. We assume a geometry as shown in Figure 23.1.1 where the primed coordinate system is the simulation domain coordinate system, the unprimed system is the primed system translated by the vector $\mathbf{r}_c$ and is added here for clarity. The $\mathbf{r}_c$ vector is given by `center` in the `geometry` module. In spherical, the only runtime parameter of importance is `rotational_frequency` in units of Hz. This specifies the angular velocity vector which is assumed to be along the $\mathbf{k}$ direction:

$$\mathbf{\Omega} \equiv \Omega\mathbf{k} = 2\pi * (\texttt{rotational\_frequency})\,\mathbf{k}.$$

The direction of $\mathbf{r}$ is given as the `normal` vector which is passed into `mk_vel_force`; in particular

$$\cos\theta \equiv \frac{\mathbf{r}\cdot\mathbf{k}}{r} = \mathbf{normal}\cdot\mathbf{k} = \texttt{normal}(3).$$

The magnitude of $\mathbf{r}$ is calculated based on the current zone's location with respect to the `center`. Using this notation we can write the Centrifugal term as

$$\begin{aligned}\mathbf{\Omega}\times(\mathbf{\Omega}\times\mathbf{r}) &= (\mathbf{\Omega}\cdot\mathbf{r})\,\mathbf{\Omega} - (\mathbf{\Omega}\cdot\mathbf{\Omega})\,\mathbf{r} \\ &= \Omega^2 r * [\texttt{normal}(3)]\,\mathbf{k} - \Omega^2 r * \mathbf{normal} = \begin{pmatrix} \Omega^2 r * [\texttt{normal}(1)] \\ \Omega^2 r * [\texttt{normal}(2)] \\ 0 \end{pmatrix}.\end{aligned}$$

The Coriolis term is a straightforward cross-product:

$$\begin{aligned}\mathbf{\Omega}\times\mathbf{U} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & 0 & \Omega \\ u & v & w \end{vmatrix} \\ &= \begin{pmatrix} -\Omega v \\ \Omega u \\ 0 \end{pmatrix}.\end{aligned}$$

Using Plane-Parallel Geometry

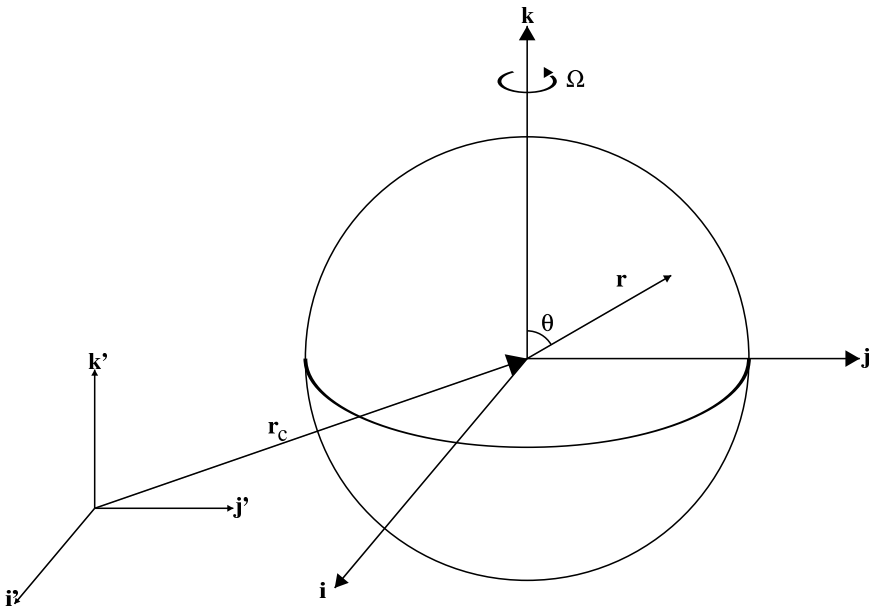


Figure 23.1: Geometry of rotation when `spherical_in` $= 1$. We assume the star to be rotating about the z axis with rotational frequency Ω.

CHAPTER 24

Radial Base State

This section used to contain notes about the "fill" and "average" routines. The multilevel paper has all the details now - if anything changes about fill and/or average, this is the place to put it.

CHAPTER 25

Spherical Expansion

Modifications for a Spherical Self-Gravitating Star

In papers II and III, we calculated the hydrostatic expansion of the base state in plane-parallel geometry under the assumption that the weight of the material above (or below) any given fluid parcel does not change during hydrostatic expansion. This assumption holds when the gravitational acceleration is independent of location. Here we discuss the modifications to the algorithm in paper III required to treat a spherical self-gravitating star.

One-dimensional Results

To test the spherical base state expansions, we inject heat at a steady rate into a one-dimensional white dwarf model. This is similar to the first test in paper II, except now in spherical coordinates. As in that test, the compressible method with which we compare the low Mach number method is the FLASH code's implementation of the piecewise-parabolic method (PPM) in a one-dimensional spherical geometry. The initial conditions for the white dwarf are those described in Section 4.1 of paper III for the central region.

In the expansion of a plane-parallel atmosphere, heating at a height r above the base does not affect the pressure or density below that height. By contrast, in a spherical symmetric self-gravitating star, heating at a radius r will lead to a pressure and density decrease at the center in addition to the expansion of the outer layers (see Schwarzchild & Harm, 1965, ApJ, 146, 855).

We apply a heating function of the form:

$$H_{\rm ext} = H_0 \exp\left[-(r - r_0)^2 / W^2\right] \quad , \tag{25.1}$$

with $r_0 = 4 \times 10^7$ cm, $W = 10^7$ cm, and $H_0 = 1 \times 10^{16}$ erg g^{-1} s^{-1}. This is the same functional form as used in the first test of paper II, but with a lower amplitude. Still, this heating rate is far higher

than what is expected during the convective phase of Type Ia SNe. The heating term is added to the enthalpy equation in the low Mach number equations in the same fashion as described in paper II. In this test, we do not consider reactions. Since this is a one-dimensional test all perturbational quantities, as well as $\widetilde{\mathbf{U}}$, are zero, so we are directly testing the computation of w_0 as and the base state update as described in the **Advect Base** procedure defined above. Both the PPM and low Mach calculation use 768 zones in a domain 5×10^8 cm high.

Figure 25.1 shows the structure of the star after heating for 10 s. The gray line is the initial star before any heating. We see that the compressible and low Mach number models agree extremely well. Both capture the decrease in the density and pressure at the center of the star and the considerable expansion in radius. Only at the surface of the star do the temperatures differ slightly. In all calculations, we set the minimum temperature to 5×10^6 K. The PPM simulation required 13488 steps and the low Mach (CFL $= 0.5$) calculation needed 203. Over the course of the simulation, the Mach number of the flow remained less than 0.35, with the maximum Mach number occurring at the surface of the star. This Mach number pushes the limits of validity of the low Mach number model; a smaller perturbation amplitude would result in a smaller Mach number.

Future improvements to the overall spherical base-state adjustment algorithm will address the expansion in a simulation where the medium outside the star is not brought down to arbitrarily low densities, but instead a "cutoff density" is applied, as in the case of the plane-parallel results presented in this paper. However, we expect the changes to the overall method shown here to be small.

ew

Add a figure showing that we retain the correct solution even when we place higher density material outside the star.

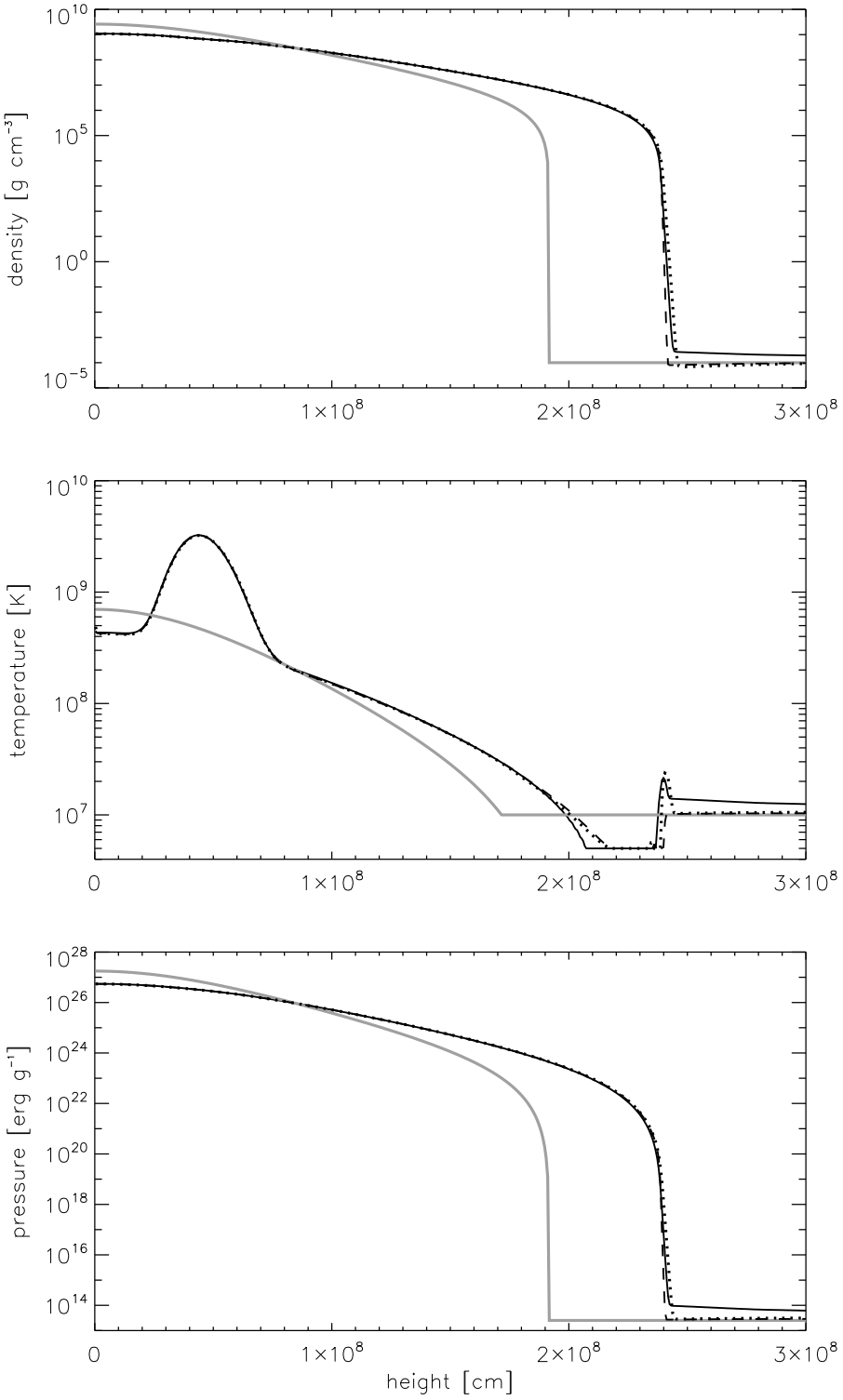


Figure 25.1: Hydrostatic adjustment of a spherically symmetric white dwarf with self-gravity. The gray line represents the initial model; all other lines are after 10 s of heating. The solid black line is the fully compressible solution, the dotted line is the low Mach number solution with a CFL number of 0.5, and the dashed line is the low Mach number solution with a CFL number of 0.1. All simulations used 768 equally spaced zones. We see excellent agreement between the compressible and low Mach number models. The only differences appear at the top of the atmosphere, where the outer boundary condition can influence the results.

CHAPTER 26

Planar $1/r^2$ Gravity Basestate

Modifications for a $1/r^2$ Plane-Parallel Basestate

In the plane parallel assumption, we model a layer of thickness ΔR a distance R_{base} from the center of a star, under the assumption the $\Delta R \ll R_{\text{base}}$. In this assumption, we can neglect the curvature of the atmosphere. Here, we extend that basic assumption to allow for the gravitational acceleration to still fall off as $1/r^2$ as we move outward in the envelope. We assume that the mass of the envelope is insignificant, and that only the mass of the underlying star contributes to the gravitational acceleration.

Constraint Equation

We begin with the w_0 constraint equation (derived elsewhere), including the volume-discrepancy term:

$$\nabla \cdot (w_0 \mathbf{e}_r) = \bar{S} - \frac{1}{\overline{\Gamma}_1 p_0}\psi - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - \overline{p_{0_{\text{EOS}}}}}{\Delta t} \tag{26.1}$$

where

$$\psi = \frac{D_0 p_0}{Dt} = \frac{\partial p_0}{\partial t} + w_0 \frac{\partial p_0}{\partial t}.$$

In Cartesian geometry, the divergence on the left hand side is simply $\partial w_0/\partial r$. Let us now define $\overline{w}_0$ and δw_0 such that

$$w_0 = \overline{w}_0 + \delta w_0$$

We take $\overline{w}_0$ to satisfy

$$\frac{\partial \overline{w}_0}{\partial r} = \bar{S} - \frac{f}{\overline{\Gamma}_1 p_0}\frac{p_0 - \overline{p_{0_{\text{EOS}}}}}{\Delta t}, \tag{26.2}$$

leaving the equation

$$\frac{\partial \delta w_0}{\partial r} = -\frac{1}{\overline{\Gamma}_1 p_0}\left(\frac{\partial p_0}{\partial t} + w_0 \frac{\partial p_0}{\partial r}\right). \tag{26.3}$$

If we multiply by $\overline{\Gamma}_1 p_0$, differentiate by r, then switch the order of ∂t and ∂r where they appear in the same term, we get

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right] = -\frac{\partial}{\partial t}\frac{\partial p_0}{\partial r} - \frac{\partial}{\partial r}\left(w_0 \frac{\partial p_0}{\partial r}\right). \tag{26.4}$$

We then substitute in the equation of hydrostatic equilibrium:

$$\frac{\partial p_0}{\partial r} = -\rho_0 g \qquad \text{where} \qquad g = \frac{Gm}{r^2}.$$

We will assume that the mass of the atmosphere is negligible relative to the mass of the core, which is outside of the simulation domain. This simplifies the equation by allowing us to assume that m, the enclosed mass, is constant. So we now have

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial}{\partial r}(\delta w_0)\right] = \frac{\partial}{\partial t}(\rho_0 g) + \frac{\partial}{\partial r}(w_0 \rho_0 g) = \rho_0\left(\frac{\partial}{\partial t} g + w_0 \frac{\partial}{\partial r} g\right) + g\left(\frac{\partial}{\partial t}\rho_0 + \frac{\partial}{\partial r}(w_0 \rho_0)\right). \tag{26.5}$$

We now recall Equation 29 from Paper III:

$$\frac{\partial}{\partial t}\rho_0 = -\nabla \cdot (w_0 \rho_0 \mathbf{e}_r) - \nabla \cdot (\eta_\rho \mathbf{e}_r),$$

which is, in Cartesian geometry,

$$\frac{\partial \rho_0}{\partial t} = -\frac{\partial}{\partial r}(w_0 \rho_0) - \frac{\partial \eta_\rho}{\partial r}. \tag{26.6}$$

Substituting this expression yields

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial}{\partial r}(\delta w_0)\right] = \rho_0 \frac{D_0 g}{Dt} + g\left(-\frac{\partial}{\partial r}(w_0 \rho_0) - \frac{\partial \eta_\rho}{\partial r} + \frac{\partial}{\partial r}(w_0 \rho_0)\right) = \rho_0 \frac{D_0 g}{Dt} - g\frac{\partial \eta_\rho}{\partial r}. \tag{26.7}$$

We then differentiate the gravitational acceleration:

$$\begin{aligned} \frac{D_0 g}{Dt} &= \frac{D_0}{Dt}\left(\frac{Gm}{r^2}\right) \\ &= Gm\left(\frac{\partial}{\partial t}(r^{-2}) + w_0 \frac{\partial}{\partial r}(r^{-2})\right) \\ &= -\frac{2Gmw_0}{r^3} \\ &= -\frac{2w_0 g}{r}. \end{aligned} \tag{26.8}$$

Substituting in this expression gives our final result:

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial}{\partial r}(\delta w_0)\right] = -\frac{2w_0 \rho_0 g}{r} - g\frac{\partial \eta_\rho}{\partial r} \tag{26.9}$$

Uniform Δr Discretization

Collecting all of the δw_0 terms on the left side, our constraint equation appears as:

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right] + \frac{2\rho_0 g \delta w_0}{r} = -\frac{2\rho_0 g \overline{w}_0}{r} - g\frac{\partial \eta_\rho}{\partial r} \tag{26.10}$$

On a uniform mesh (constant Δr, we would discretize this as:

$$\frac{1}{\Delta r}\left\{\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right]_j - \left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right]_{j-1}\right\} + \left[\frac{2\rho_0 g}{r}\delta w_0\right]_{j-\frac{1}{2}} = -\left[\frac{2\rho_0 g}{r}\overline{w}_0\right]_{j-\frac{1}{2}} - \frac{g_{j-\frac{1}{2}}}{\Delta r}\left[\eta_{\rho_j} - \eta_{\rho_{j-1}}\right] \tag{26.11}$$

Expanding the $\partial \delta w_0/\partial r$ terms, we have:

$$\frac{1}{(\Delta r)^2}\left\{\left[(\overline{\Gamma}_1 p_0)_j\left(\delta w_{0_{j+\frac{1}{2}}} - \delta w_{0_{j-\frac{1}{2}}}\right)\right] - \left[(\overline{\Gamma}_1 p_0)_{j-1}\left(\delta w_{0_{j-\frac{1}{2}}} - \delta w_{0_{j-3/2}}\right)\right]\right\} + \left[\frac{2\rho_0 g}{r}\delta w_0\right]_{j-\frac{1}{2}} = -\left[\frac{2\rho_0 g}{r}\overline{w}_0\right]_{j-\frac{1}{2}} - \frac{g_{j-\frac{1}{2}}}{\Delta r}\left[\eta_{\rho_j} - \eta_{\rho_{j-1}}\right] \tag{26.12}$$

As with the spherical case (multilevel paper, appendix B), we write this in the form:

$$A_j(\delta w_0)_{j-3/2} + B_j(\delta w_0)_{j-1/2} + C_j(\delta w_0)_{j+1/2} = F_j, \tag{26.13}$$

then:

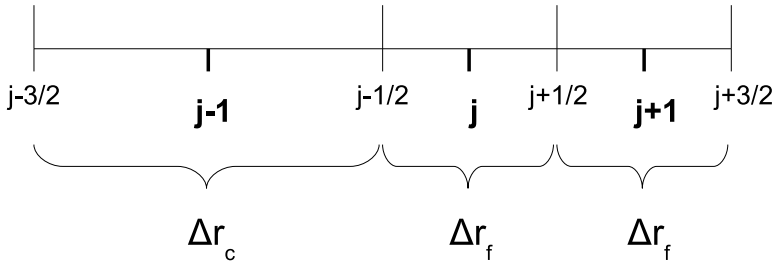
$$A_j = \frac{1}{\Delta r^2}\left(\overline{\Gamma}_1 p_0\right)_{j-1}, \tag{26.14}$$

$$B_j = -\frac{1}{\Delta r^2}\left[\left(\overline{\Gamma}_1 p_0\right)_j + \left(\overline{\Gamma}_1 p_0\right)_{j-1}\right] + \frac{2}{r_{j-1/2}}\left(\rho_0 g\right)_{j-\frac{1}{2}}, \tag{26.15}$$

$$C_j = \frac{1}{\Delta r^2}\left(\overline{\Gamma}_1 p_0\right)_j, \tag{26.16}$$

$$F_j = -\frac{2}{r_{j-1/2}}(\rho_0 g)_{j-\frac{1}{2}}(\overline{w}_0)_{j-\frac{1}{2}} - \frac{g_{j-\frac{1}{2}}}{\Delta r}\left[\left(\eta_\rho\right)_j - \left(\eta_\rho\right)_{j-1}\right] \tag{26.17}$$

Non-Uniform Δr Discretization



Consider the above non-uniform grid spacing, where $\Delta r_c = 2\Delta r_f$. Here, the discretization of the Laplacian-like term is more complex. We want to compute

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right]_{j-\frac{1}{2}} \tag{26.18}$$

This is to be centered at $j - \frac{1}{2}$, which we accomplish by averaging the two fine grids and then differencing:

$$\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right]_{j-\frac{1}{2}} = \frac{1}{\Delta r_c}\left\{\frac{1}{2}\left[\left(\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right)_{j+1} + \left(\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right)_j\right] - \left(\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right)_{j-1}\right\} \tag{26.19}$$

Expanding the $\partial\delta w_0/\partial r$ terms results in a equation depending on δw_0 at 4 different edge locations—this no longer fits into the tri-diagonal format used in the uniform grid case. In detail, it becomes:

$$\begin{aligned}\frac{\partial}{\partial r}\left[\overline{\Gamma}_1 p_0 \frac{\partial \delta w_0}{\partial r}\right]_{j-\frac{1}{2}} &= \frac{1}{\Delta r_c}\left\{\frac{1}{2}\left[\left(\overline{\Gamma}_1 p_0\right)_{j+1}\frac{(\delta w_0)_{j+3/2}-(\delta w_0)_{j+\frac{1}{2}}}{\Delta r_f}+\left(\overline{\Gamma}_1 p_0\right)_j\frac{(\delta w_0)_{j+\frac{1}{2}}-(\delta w_0)_{j-\frac{1}{2}}}{\Delta r_f}\right]\right. \\ &\qquad \left.-\left(\overline{\Gamma}_1 p_0\right)_{j-1}\frac{(\delta w_0)_{j-\frac{1}{2}}-(\delta w_0)_{j-3/2}}{\Delta r_c}\right\} \end{aligned} \tag{26.20}$$

which has terms proportional to $(\delta w_0)_{j-3/2}$, $(\delta w_0)_{j-\frac{1}{2}}$, $(\delta w_0)_{j+\frac{1}{2}}$, and $(\delta w_0)_{j+3/2}$

Boundary Conditions

Together with the assumption that the mass of the envelope does not contribute to the gravitational acceleration, we assume that as we move a fluid element in the atmosphere, it does not drive a velocity at the very base of the layer. Therefore, we take $w_0(r_{\text{base}}) = 0$.

CHAPTER 27

Thermodynamic Relations and Stability

Derivatives With Respect to Composition

In the following we assume that the molar mass of species i is given by its atomic mass number, $A_i = N_i + Z_i$ where N_i is the number of neutrons and Z_i is the number of protons in the isotope. This is a slight approximation that ignores the mass difference between protons and neutrons as well as some minor binding energy terms.

Note that this is common practice and that our EOS also makes this approximation.

The number density [cm$^-$3] of isotope i is can be formed from the mass density and the molar mass of that isotope as follows:

$$n_i = \frac{\rho_i N_\mathrm{A}}{A_i}, \tag{27.1}$$

where N_A is Avogadro's number [# / mole]. The molar abundance, Y_i, is a measure of the number of moles of species i per gram in the system:

$$Y_i = \frac{n_i}{\rho N_\mathrm{A}} = \frac{\rho_i}{\rho}\frac{1}{A_i} \equiv \frac{X_i}{A_i} \tag{27.2}$$

where we have defined the mass fraction, $X_i = \frac{\rho_i}{\rho}$. Note

$$\sum_i X_i = 1. \tag{27.3}$$

We write the average molar mass and average proton number as:

$$\bar{A} = \frac{\sum_i A_i n_i}{\sum_i n_i} = \left(\sum_i X_i\right)\left(\sum_i \frac{X_i}{A_i}\right)^{-1} \tag{27.4}$$

$$\bar{Z} = \frac{\sum_i Z_i n_i}{\sum_i n_i} = \left(\sum_i Z_i \frac{X_i}{A_i}\right)\left(\sum_i \frac{X_i}{A_i}\right)^{-1}. \tag{27.5}$$

Our algorithm requires terms involving the derivative thermodynamic variables (p or e, e.g.) with respect to composition. Our EOS does not return such derivatives but instead returns derivatives of these variables with respect to $\bar{A}$ and $\bar{Z}$. Using the chain rule, we have

$$\frac{\partial p}{\partial X_k} = p_{X_k} = \frac{\partial p}{\partial \bar{A}}\frac{\partial \bar{A}}{\partial X_k} + \frac{\partial p}{\partial \bar{Z}}\frac{\partial \bar{Z}}{\partial X_k}. \tag{27.6}$$

From (27.4) and (27.5) we have

$$\frac{\partial \bar{A}}{\partial X_k} = \left(\sum_i \frac{X_i}{A_i}\right)^{-1} - \frac{\bar{A}^2}{A_k} = -\frac{\bar{A}}{A_k}\left(\bar{A} - A_k\right) \tag{27.7}$$

$$\frac{\partial \bar{Z}}{\partial X_k} = \left(\frac{Z_k}{A_k}\right)\left(\sum_i \frac{X_i}{A_i}\right)^{-1} - \frac{\bar{Z}}{A_k}\left(\sum_i \frac{X_i}{A_i}\right)^{-1} = -\frac{\bar{A}}{A_k}\left(\bar{Z} - Z_k\right), \tag{27.8}$$

where after differentiation we have used (27.3) to write

$$\left(\sum_i \frac{X_i}{A_i}\right)^{-1} = \bar{A}.$$

We therefore have

$$p_{X_k} = -\frac{\bar{A}}{A_k}\left(\bar{A} - A_k\right)\frac{\partial p}{\partial \bar{A}} - \frac{\bar{A}}{A_k}\left(\bar{Z} - Z_k\right)\frac{\partial p}{\partial \bar{Z}}. \tag{27.9}$$

Before it was brought to our attention by Frank Timmes, we were missing the second term in (27.7). The only place where such terms appear in our algorithm is in a sum over all species, such as:

$$\sum_i p_{X_i}\dot{\omega}_i = -\bar{A}^2\frac{\partial p}{\partial \bar{A}}\sum_i \frac{\dot{\omega}_i}{A_i} + \bar{A}\frac{\partial p}{\partial \bar{A}}\sum_i \dot{\omega}_i - \bar{A}\bar{Z}\frac{\partial p}{\partial \bar{Z}}\sum_i \frac{\dot{\omega}_i}{A_i} + \bar{A}\frac{\partial p}{\partial \bar{Z}}\sum_i \frac{Z_i}{A_i}\dot{\omega}_i. \tag{27.10}$$

Note that if we are dealing with symmetric nuclei (as we have been up through paper III) where $A_k = N_k = \frac{1}{2}A_k$ for all k, then the last term in (27.10) is identically zero as well.

The second term in (27.10) is identically zero because

$$\sum_k \dot{\omega}_k \equiv 0.$$

This second term arises from what was added to (27.7) by Frank's correction. Therefore, although important for individual derivatives with respect to composition, this correction term has no effect on our solution.

Convective stability criterion

Here we look at the criterion for convective stability in the case of non-uniform chemical composition. This section follows Cox & Giuli [36] closely (see chapter 13).

Consider a fluid parcel that gets displaced upwards (against gravity) from a radial location r to $r + \Delta r$. The parcel is stable to convection if the displaced parcel's density is greater than the surrounding fluid and gravity pushes the parcel back towards where it came from. Then the criterion for stability should be

$$\rho_{parcel}(r + \Delta r) - \rho_{background}(r + \Delta r) > 0 \tag{27.11}$$

$$\left[\rho_{parcel}(r) + \left(\frac{d\rho}{dr}\right)_{parcel}\Delta r\right] - \left[\rho_{background}(r) + \left(\frac{d\rho}{dr}\right)_{background}\Delta r\right] > 0 \tag{27.12}$$

Since the parcel originates at r, $\rho_{parcel}(r) = \rho_{background}(r)$ and so the stability criterion is

$$\left(\frac{d\rho}{dr}\right)_{parcel} > \left(\frac{d\rho}{dr}\right)_{background} \tag{27.13}$$

Since the total pressure, P, always increases inward in a star in hydrostatic equilibrium, we can use P instead of r as the independent radial variable. Then condition for stability can be written as

$$\left(\frac{d\ln\rho}{d\ln P}\right)_{parcel} < \left(\frac{d\ln\rho}{d\ln P}\right)_{background}$$

Using the equation of state $P = P(\rho, T, \bar{\mu})$, where $\bar{\mu}$ is the average mass per molecule, we can write

$$d\ln P = \left.\frac{\partial \ln P}{\partial \ln \rho}\right|_{T,\bar{\mu}} d\ln\rho + \left.\frac{\partial \ln P}{\partial \ln T}\right|_{\rho,\bar{\mu}} d\ln T + \left.\frac{\partial \ln P}{\partial \ln \bar{\mu}}\right|_{\rho,T} d\ln\bar{\mu} \tag{27.14}$$

For convenience we introduce

$$\chi_\rho = \left.\frac{\partial \ln P}{\partial \ln \rho}\right|_{T,\bar{\mu}} \qquad \chi_T = \left.\frac{\partial \ln P}{\partial \ln T}\right|_{\rho,\bar{\mu}} \qquad \chi_{\bar{\mu}} = \left.\frac{\partial \ln P}{\partial \ln \bar{\mu}}\right|_{\rho,T}$$

Then we can rearrange 27.14 to get

$$\frac{d\ln\rho}{\partial \ln P} = \frac{1}{\chi_\rho} - \frac{\chi_T}{\chi_\rho}\frac{d\ln T}{d\ln P} - \frac{\chi_{\bar{\mu}}}{\chi_\rho}\frac{d\ln\bar{\mu}}{d\ln P} \tag{27.15}$$

Then the general stability criterion is

$$\left(\frac{1}{\chi_\rho} - \frac{\chi_T}{\chi_\rho}\frac{d\ln T}{d\ln P} - \frac{\chi_{\bar{\mu}}}{\chi_\rho}\frac{d\ln\bar{\mu}}{d\ln P}\right)_{parcel} < \left(\frac{1}{\chi_\rho} - \frac{\chi_T}{\chi_\rho}\frac{d\ln T}{d\ln P} - \frac{\chi_{\bar{\mu}}}{\chi_\rho}\frac{d\ln\bar{\mu}}{d\ln P}\right)_{background} \tag{27.16}$$

Here's where various assumptions/simplifications get used.

1. If no assumptions are made, you can't get any further than equation (27.16). Even in view of an infinitesimally small initial perturbation, you can't, in general, assume the χ's in parcel are the same as the χ's in the background. This applies in the case where nuclear reactions and/or ionization change the composition of the parcel. This case tends not to be of much interest for two reasons. Either composition effects get incorporated implicitly through assuming chemical equilibrium. Or both of these terms can be neglected in the rising parcel. This would be justified if the timescale for reactions is long compared to the convective timescale, and either the same is ture for ionization or the fluid is fully ionized.

2. If we assume that $\bar{\mu}$ remains constant in the parcel, then $\frac{d\ln\bar{\mu}}{d\ln P}$ drops out for the parcel. In this case, we can assume, in view of the arbitrarily small initial perturbation of the parcel, that χ_ρ and χ_T to have the same values in the parcel as in the background. Then the stability criterion becomes

$$\left(\frac{d\ln T}{d\ln P}\right)_{parcel} > \left(\frac{d\ln T}{d\ln P} + \frac{\chi_{\bar{\mu}}}{\chi_T}\frac{d\ln\bar{\mu}}{d\ln P}\right)_{background} \tag{27.17}$$

   The Ledoux stability criterion is obtained by assuming that the parcel moves adiabatically.

3. If we assume that the background is in chemical equilibrium and the parcel acheives instantaneous chemical equilibrium, then $\bar{\mu} = \bar{\mu}(\rho, T)$ for the background and the parcel. (Note that we aren't requiring constant composition in the parcel here.) The effect of variable composition are then absorbed into χ_ρ and χ_T. Again, we can take χ_ρ and χ_T to have the same values in the parcel as in the background. The criterion then is

$$\left(\frac{d\ln T}{d\ln P}\right)_{parcel} > \left(\frac{d\ln T}{d\ln P}\right)_{background} \tag{27.18}$$

We obtain the Schwarzchild criterion for stability if we also assume the parcel moves adiabatically.

The Scharwzchild criterion can be recast in terms of entropy if the EOS is taken as $P(\rho, S)$ instead of $P(\rho, T)$. Then, in place of equation (27.14) we have

$$d\rho = \left.\frac{\partial\rho}{\partial P}\right|_S dP + \left.\frac{\partial\rho}{\partial S}\right|_P dS \tag{27.19}$$

We can subsitute this into equation (27.13) for stability, and assuming the parcel moves adiabatically, we get

$$\left(\left.\frac{\partial\rho}{\partial S}\right|_P \frac{dS}{dr}\right)_{background} < 0 \tag{27.20}$$

One of Maxwell's relations is

$$\left.\frac{\partial\rho^{-1}}{\partial S}\right|_P = \left.\frac{\partial T}{\partial P}\right|_S \tag{27.21}$$

All thermodynamically stable substances have temperatures that increase upon adiabatic compression, i.e. $\left.\frac{\partial T}{\partial P}\right|_S > 0$. So Maxwell's relation implies that $\left.\frac{\partial\rho}{\partial S}\right|_P < 0$. The stability criterion then becomes

$$\left(\frac{dS}{dr}\right)_{background} > 0 \tag{27.22}$$

Determining which stability criterion we want to enforce in creating the initial model is complicated by the phenomenon of semiconvection, which occurs when the Ledoux criterion is statisfied but the Schwarzchild is not, i.e.

$$\left(\frac{d\ln T}{d\ln P}\right)_{parcel} < \left(\frac{d\ln T}{d\ln P}\right)_{background} < \left(\frac{d\ln T}{d\ln P}\right)_{parcel} - \left(\frac{\chi_{\bar{\mu}}}{\chi_T}\frac{d\ln\bar{\mu}}{d\ln P}\right)_{background} \tag{27.23}$$

(Note that $\chi_{\bar{\mu}}$ is negative, as pressure is inversely proportional to mass per particle, and $\frac{d\ln\bar{\mu}}{d\ln P}$ is positive, since nuclear reactions sythesize more massive particles in the center of the star.) In this case, when a rising parcel eventually reaches neutral buoyancy, it will have a temperature excess in comparision to it's surroundings. If the parcel can retain it's identitiy against diffusive mixing with the background long enough for significant heat exhange to occur, then the parcel's temperature will drop, it will contract increasing it's density, and the parcel will move inwards. The time scale of semiconvection is much longer than the timescale of traditional convection.

When we set up an initial model, we want to minimize any initial tendancy towards convective motions, as we want these to be driven by the heating due to nuclear reactions, not the initial configuration

we supply. Thus I think we want to guard against semiconvection as well as "traditional" convection by using the stability criterion

$$\left(\frac{d\ln T}{d\ln P}\right)_{parcel} = \left.\frac{d\ln T}{d\ln P}\right|_{S,\bar{\mu}} > \left(\frac{d\ln T}{d\ln P}\right)_{background} \tag{27.24}$$

Although this looks like the Schwarschild criterion (and, because I'm not entirely sure on vocabulary, it might even be called the Schwarzchild criterion), this does not simplify to equation (27.22) because we need to keep the explicit $\bar{\mu}$ dependence in the EOS.

The question of whether we're in chemical equilibrium or not might be a moot point since our EOS (or any other part of the code) doesn't enforce chemical equilibrium. Thus, even in the case of chemical equilbrium, we can't in general drop the explicit $\bar{\mu}$ dependence from our equations. If we wanted to do that, then we would need $\bar{\mu}(\rho, T)$ to be substituted for $\bar{\mu}$ inside the EOS.

Adiabatic Excess

The adiabatic excess, $\Delta\nabla$, is a quantity used to determine if a system is stable ($\Delta\nabla < 0$) or unstable ($\Delta\nabla > 0$) to convection under the Schwarzschild criterion (i.e. neglecting compositional gradients). Cox and Giuli (see chapter 9) define three different "adiabatic exponents" that we will use:

$$\begin{aligned}
\Gamma_1 &\equiv \left(\frac{d\ln p}{d\ln\rho}\right)_{\rm ad} \\
\frac{\Gamma_2}{\Gamma_2-1} &\equiv \left(\frac{d\ln p}{d\ln T}\right)_{\rm ad} \\
\Gamma_3 - 1 &\equiv \left(\frac{d\ln T}{d\ln\rho}\right)_{\rm ad},
\end{aligned}$$

where the subscript "ad" means along an adiabat. We can combine the exponents to get the following relation

$$\Gamma_1 = \left(\frac{\Gamma_2}{\Gamma_2-1}\right)(\Gamma_3 - 1)\,. \tag{27.25}$$

The adiabatic excess is defined as

$$\Delta\nabla = \nabla_{\rm actual} - \nabla_{\rm ad} \tag{27.26}$$

where

$$\nabla \equiv \frac{d\ln T}{d\ln P} \tag{27.27}$$

is the thermal gradient. It is important to note that these thermal gradients are only along the radial direction. The "actual" gradient can be found from finite differencing the data whereas the adiabatic term, $\nabla_{\rm ad} = (\Gamma_2 - 1)/\Gamma_2$, will need to be calculated at each point using thermodynamic relations. Our EOS only returns Γ_1 so we need find another relation to use with (27.25) to solve for the adiabatic excess.

The Schwarzschild criterion does not care about changes in composition and we therefore write $p = p(\rho, T)$ and

$$d\ln p = \chi_\rho d\ln\rho + \chi_T d\ln T \tag{27.28}$$

where

$$\chi_\rho = \left(\frac{d\ln p}{d\ln \rho}\right)_T, \qquad \chi_T = \left(\frac{d\ln p}{d\ln T}\right)_\rho.$$

Dividing (27.28) by $d\ln\rho$ and taking this along an adiabat we have

$$\left(\frac{d\ln p}{d\ln \rho}\right)_{\rm ad} = \chi_\rho + \chi_T \left(\frac{d\ln T}{d\ln \rho}\right)_{\rm ad}. \tag{27.29}$$

Using the Γ's, we have

$$\Gamma_1 = \chi_\rho + \chi_T \left(\Gamma_3 - 1\right). \tag{27.30}$$

Combining (27.25) and (27.30) to eliminate Γ_3, we have:

$$\nabla_{\rm ad} = \frac{\Gamma_1 - \chi_\rho}{\chi_T \Gamma_1} \tag{27.31}$$

which uses only terms which are easily returned from an EOS call.

CHAPTER 28

Notes on β_0

The goal of β_0 is to capture the expansion of a displaced fluid element due to the stratification of the atmosphere. MAESTRO computes β_0 as:

$$\beta_0(r,t) = \rho_0 \exp\left(\int_0^r \frac{1}{\overline{\Gamma}_1 p_0}\frac{\partial p_0}{\partial r'}dr'\right) \tag{28.1}$$

Constant Composition

Consider an isentropically stratified atmosphere, with a constant composition as a function of r. If you displace a parcel of fluid upwards, it will expand adabatically and continue to rise until its density matches the background density. Even if $\overline{\Gamma}_1$ is not constant in r, following from the definition of β_0,

$$\frac{1}{\beta_0}\frac{d\beta_0}{dr} = \frac{1}{\overline{\Gamma}_1 p_0}\frac{dp_0}{dr} \tag{28.2}$$

and the definition of Γ_1,

$$\Gamma_1 = \left.\frac{d\log p}{d\log \rho}\right|_s \tag{28.3}$$

So, at constant entropy, from the definition of Γ_1, it must hold that

$$\frac{1}{\rho}\frac{d\rho}{dr} = \frac{1}{\Gamma_1 p}\frac{dp}{dz} \quad . \tag{28.4}$$

Comparing to the defintion of β_0 then

$$\frac{1}{\beta_0}\frac{\beta_0}{dr} = \frac{1}{\overline{\Gamma}_1 p_0}\frac{dp_0}{dr} = \frac{1}{\rho_0}\frac{d\rho_0}{dr} \quad . \tag{28.5}$$

Therefore, $\beta_0 = \rho_0$.

This means that if we have a constant composition and an isentropically stratified atmosphere, as we displace a fluid element, it will always remain neutrally buoyant.

Composition Gradient

If there is a change in composition with r, the situation is more complicated. Consider again an isentropically stratified atmosphere, now with a composition gradient. If you displace a parcel of fluid upwards, it will rise. If there are no processes that change the composition (e.g. reactions), then the composition in the fluid element will remain fixed. As it rises, it will the ambient medium will have a different composition that it has. In this case, what is the path to equilibrium?

On the Effect of Chemical Potential

In MAESTRO, we do things in an operator split fashion — the hydro is de-coupled from the burning. This means that during the hydro parts of the algorithm (where β_0 is used), the system is fixed in chemical equilibrium. For completeness, however, here we describe the effects of the species' chemical potentials, which were neglected in the original derivation of β_0. Note that similar terms appear in the calculation of things such as specific heats, which *may* be important in the burning step — there appears to be very little about this in the literature, but everyone seems to assume it makes little difference.

Derivation of α

In paper I, α is defined as

$$\alpha \equiv -\left(\frac{(1-\rho h_p)p_T - \rho c_p}{\rho^2 c_p p_\rho}\right) \tag{28.6}$$

where

$$h_p \equiv \left(\frac{\partial h}{\partial p}\right)_{T,X}, \quad c_p \equiv \left(\frac{\partial h}{\partial T}\right)_{p,X}, \quad p_T \equiv \left(\frac{\partial p}{\partial T}\right)_{\rho,X}, \quad p_\rho \equiv \left(\frac{\partial p}{\partial \rho}\right)_{T,X}$$

where the subscript X means holding all X_i constant. In the absence of reactions, the X subscript can be dropped from all derivatives and with the use of the equation of state $p = p(\rho, T)$, α can be written as $\alpha = \alpha(\rho, T)$. Such a system without reactions and in thermal equilibrium could be either a pure system of one species, or a system of many species in chemical (and therefore *thermodynamic*) equilibrium. Cox and Giuli (hereafter CG) call the former type of system a "simple system" and therefore the latter a "non-simple system" in chemical equilibrium. The analysis in paper I that reduced (28.6) to

$$\alpha = \frac{1}{\Gamma_1 p_0} \tag{28.7}$$

used CG's discussion of the various adiabatic Γ's. However, their discussion only pertains to "simple systems" or "non-simple systems" in chemical equilibrium. In general, nuclear reactions will be important and therefore this analysis needs to be reformed.

Even in the presence of reactions, (28.6) can be rewritten as was done in paper I:

$$\alpha = -\frac{1}{p\chi_\rho c_p}\left[\left(\frac{1}{\rho\chi_\rho} - \frac{\rho e_\rho}{p\chi_\rho}\right)\frac{p\chi_T}{T} - c_p\right], \tag{28.8}$$

where

$$\chi_\rho \equiv \left(\frac{\partial \ln p}{\partial \ln \rho}\right)_{T,X}$$
$$\chi_T \equiv \left(\frac{\partial \ln p}{\partial \ln T}\right)_{\rho,X}.$$

Following the results of paper I, we want to find a relation between $p\chi_\rho$ and Γ_1.

For an equation of state $p = p(\rho, T, X)$ we have

$$d\ln p = \left(\frac{\partial \ln p}{\partial \ln \rho}\right)_{T,X} d\ln\rho + \left(\frac{\partial \ln p}{\partial \ln T}\right)_{\rho,X} d\ln T + \sum_i \left(\frac{\partial \ln p}{\partial \ln X_i}\right)_{\rho,T,(X_j,j\neq i)} d\ln X_i.$$

We define another logarithmic derivative

$$\chi_{X_i} \equiv \left(\frac{\partial \ln p}{\partial \ln X_i}\right)_{\rho,T,(X_j,j\neq i)}$$

and therefore

$$d\ln p = \chi_\rho\, d\ln\rho + \chi_T\, d\ln T + \sum_i \chi_{X_i}\, d\ln X_i.$$

From here we get the general statement

$$\frac{\partial \ln p}{\partial \ln \rho} = \chi_\rho + \chi_T \frac{\partial \ln T}{\partial \ln \rho} + \sum_i \chi_{X_i}\frac{\partial \ln X_i}{\partial \ln \rho}$$

which must hold for an adiabatic process as well, and therefore we have

$$\Gamma_1 = \chi_\rho + \chi_T\,(\Gamma_3 - 1) + \sum_i \chi_{X_i}\Gamma_{4,i} \tag{28.9}$$

where we use CG's definition of Γ_1 and Γ_3 and introduce a fourth gamma function:

$$\Gamma_1 \equiv \left(\frac{\partial \ln p}{\partial \ln \rho}\right)_{\rm AD}, \quad \Gamma_3 - 1 \equiv \left(\frac{\partial \ln T}{\partial \ln \rho}\right)_{\rm AD}, \quad \Gamma_{4,i} \equiv \left(\frac{\partial \ln X_i}{\partial \ln \rho}\right)_{\rm AD},$$

where the subscript AD means along an adiabat. We now derive an expression for Γ_3.

The first law of thermodynamics can be written as

$$dQ = dE + pdV - \sum_i \mu_i dN_i$$

where $\mu_i = \left(\frac{\partial E}{\partial N_i}\right)_{\rm AD,\rho,(N_j,j\neq i)}$ is the chemical potential; or per unit mass we have

$$dq = de - \frac{p}{\rho^2}d\rho - \sum_i \mu_i d\left(\frac{n_i}{\rho}\right)$$
$$= de - \frac{p}{\rho^2}d\rho - \sum_i \left(\frac{\partial e}{\partial X_i}\right)_{\rho,{\rm AD},(X_j,j\neq i)} dX_i$$

where we have used $X_i \equiv \rho_i/\rho = A_i n_i/\rho N_\mathrm{A}$ and the chemical potential has been replaced with $\mu_i = \frac{A_i}{N_\mathrm{A}}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}$. Using this and expressing the specific internal energy as $e = e(\rho, T, X)$ we then have

$$dq = c_v dT + \left[\left(\frac{\partial e}{\partial \rho}\right)_{T,X} - \frac{p}{\rho^2}\right] d\rho + \sum_i \left[\left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)} - \left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right] dX_i$$

and

$$\left(\frac{d\ln T}{d\ln \rho}\right)_\mathrm{AD} \equiv \Gamma_3 - 1 = \frac{1}{c_v T}\left[\frac{p}{\rho} - \left(\frac{\partial e}{\partial \ln \rho}\right)_{T,X} + \sum_i \left[\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)} - \left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)}\right] X_i \Gamma_{4,i}\right] \tag{28.10}$$

Now we need to evaluate $(\partial e/\partial \ln \rho)_{T,X}$. Again using the first law and the fact that $ds = dq/T$ is an exact differential (i.e. mixed derivatives are equal) we have

$$\left(\frac{\partial}{\partial \rho}\left[\frac{c_v}{T}\right]\right)_{T,X} = \left(\frac{\partial}{\partial T}\left[\frac{1}{T}\left(\frac{\partial e}{\partial \rho}\right)_{T,X} - \frac{p}{T\rho^2}\right]\right)_{\rho,X}$$

$$\frac{1}{T}\left(\frac{\partial}{\partial \rho}\left(\frac{\partial e}{\partial T}\right)_{\rho,X}\right)_{T,X} = -\frac{1}{T^2}\left(\frac{\partial e}{\partial \rho}\right)_{T,X} + \frac{1}{T}\left(\frac{\partial}{\partial T}\left(\frac{\partial e}{\partial \rho}\right)_{T,X}\right)_{\rho,X} + \frac{p}{T^2\rho^2} - \frac{1}{T\rho^2}\left(\frac{\partial p}{\partial T}\right)_{\rho,X}$$

$$\therefore \quad \left(\frac{\partial e}{\partial \ln \rho}\right)_{T,X} = \frac{p}{\rho}\left(1 - \chi_T\right), \tag{28.11}$$

exactly the same result if we were to exclude species information. Simlarly, we can find an expression for the derivative of energy with respect to composition

$$\left(\frac{\partial}{\partial X_i}\left[\frac{c_v}{T}\right]\right)_{\rho,T,(X_j,j\neq i)} = \left(\frac{\partial}{\partial T}\left[\frac{1}{T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)} - \frac{1}{T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right]\right)_{\rho,X}$$

$$\frac{1}{T}\left(\frac{\partial}{\partial X_i}\left(\frac{\partial e}{\partial T}\right)_{\rho,X}\right)_{\rho,T,(X_j,j\neq i)} = \frac{1}{T^2}\left[\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)} - \left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)}\right] + \frac{1}{T}\left[\left(\frac{\partial}{\partial T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)}\right)_{\rho,X} - \left(\frac{\partial}{\partial T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X}\right]$$

$$\therefore \quad \left(\frac{\partial e}{\partial X_i}\right)_{\rho,T,(X_j,j\neq i)} = \left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)} - \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X}.$$

Plugging these back into (28.10) we have

$$\Gamma_3 - 1 = \frac{1}{c_v T}\left[\frac{p}{\rho}\chi_T + \sum_i \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X} X_i \Gamma_{4,i}\right], \tag{28.12}$$

or

$$c_v = \frac{1}{T(\Gamma_3 - 1)}\left[\frac{p}{\rho}\chi_T + \sum_i \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X} X_i \Gamma_{4,i}\right]. \tag{28.13}$$

We can obtain an expression for the specific heat at constant pressure from the enthalpy

$$\begin{aligned} c_p \equiv \left(\frac{\partial h}{\partial T}\right)_{p,X} &= \left(\frac{\partial e}{\partial T}\right)_{p,X} - \frac{p}{\rho^2}\left(\frac{\partial \rho}{\partial T}\right)_{p,X} \\ &= \left(\frac{\partial e}{\partial T}\right)_{p,X} + \frac{p}{\rho^2}\left(\frac{\partial p}{\partial T}\right)_{\rho,X}\left(\frac{\partial \rho}{\partial p}\right)_{T,X} \\ &= \left(\frac{\partial e}{\partial T}\right)_{p,X} + \frac{p}{\rho T}\frac{\chi_t}{\chi_\rho}. \end{aligned}$$

The first term on the rhs can be obtained from writing $e = e(p, T, X)$ and $p = p(\rho, T, X)$:

$$\begin{aligned} de &= \left(\frac{\partial e}{\partial p}\right)_{T,X} dp + \left(\frac{\partial e}{\partial T}\right)_{p,X} dT + \sum_i \left(\frac{\partial e}{\partial X_i}\right)_{p,T,(X_j, j\neq i)} dX_i \\ dp &= \left(\frac{\partial p}{\partial \rho}\right)_{T,X} d\rho + \left(\frac{\partial p}{\partial T}\right)_{\rho,X} dT + \sum_i \left(\frac{\partial p}{\partial X_i}\right)_{\rho,T,(X_j, j\neq i)} dX_i \\ \therefore \left(\frac{\partial e}{\partial T}\right)_{\rho,X} &= \left(\frac{\partial e}{\partial p}\right)_{T,X}\left(\frac{\partial p}{\partial T}\right)_{\rho,X} + \left(\frac{\partial e}{\partial T}\right)_{p,X} \\ \Rightarrow \left(\frac{\partial e}{\partial T}\right)_{p,X} &= c_v - \left(\frac{\partial e}{\partial \rho}\right)_{T,X}\left(\frac{\partial \rho}{\partial p}\right)_{T,X}\left(\frac{\partial p}{\partial T}\right)_{\rho,X} \\ &= c_v - \frac{p\chi_T}{\rho T \chi_\rho}(1 - \chi_T) \end{aligned}$$

and

$$c_p = \frac{p}{\rho T}\frac{\chi_T^2}{\chi_\rho} + c_v$$

Dividing this by (28.13) and using the relation between the Γ's, (28.9), we then have

$$\begin{aligned} \gamma \equiv \frac{c_p}{c_v} &= 1 + \frac{p(\Gamma_3 - 1)}{\rho}\frac{\chi_T^2}{\chi_\rho}\left[\frac{p}{\rho}\chi_T + \sum_i \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j, j\neq i)}\right)_{\rho,X} X_i \Gamma_{4,i}\right]^{-1} \\ &= 1 + \frac{p\chi_T\left(\Gamma_1 - \chi_\rho - \sum_i \chi_{X_i}\Gamma_{4,i}\right)}{p\chi_\rho\chi_T + \rho\chi_\rho \sum_i \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j, j\neq i)}\right)_{\rho,X} X_i \Gamma_{4,i}} \\ &= \frac{p\chi_T\Gamma_1 + \sum_i \left[\rho\chi_\rho \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j, j\neq i)}\right)_{\rho,X} X_i - p\chi_T\chi_{X_i}\right]\Gamma_{4,i}}{p\chi_\rho\chi_T + \rho\chi_\rho \sum_i \left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j, j\neq i)}\right)_{\rho,X} X_i \Gamma_{4,i}} \\ \Rightarrow p\chi_\rho &= \frac{1}{\chi_T \gamma}\left[p\chi_T\Gamma_1 + \sum_i \left[\rho\chi_\rho(1-\gamma)\left(\frac{\partial}{\partial \ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j, j\neq i)}\right)_{\rho,X} X_i - p\chi_T\chi_{X_i}\right]\Gamma_{4,i}\right]. \end{aligned} \quad (28.14)$$

Plugging (28.14) into (28.8) and rewriting the partial derivative of e with the help of (28.11) we have

$$\begin{aligned}
\alpha &= -\frac{1}{p\chi_\rho c_p}\left[\left(\frac{1}{\rho\chi_\rho}-\frac{\rho e_\rho}{p\chi_\rho}\right)\frac{p\chi_T}{T}-c_p\right] \\
&= \frac{\gamma}{c_p}\frac{c_p\chi_T+\left(\rho\left(\frac{\partial e}{\partial \ln\rho}\right)_{T,X}-p\right)\frac{\chi_T^2}{T\rho\chi_\rho}}{p\chi_T\Gamma_1+\sum_i\left[\rho\chi_\rho\left(1-\gamma\right)\left(\frac{\partial}{\partial\ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X}X_i-p\chi_T\chi_{X_i}\right]\Gamma_{4,i}} \\
&= \frac{\gamma}{\Gamma_1 pc_p}\left[\frac{c_p-\frac{p\chi_T^2}{T\rho\chi_\rho}}{1+\sum_i\left[\frac{\rho\chi_\rho}{p\chi_T}\left(1-\gamma\right)\left(\frac{\partial}{\partial\ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X}X_i-\chi_{X_i}\right]\frac{\Gamma_{4,i}}{\Gamma_1}}\right] \\
&= \left(\frac{1}{\Gamma_1 p}\right)\left[1+\sum_i\left[\frac{\rho\chi_\rho}{p\chi_T}\left(1-\gamma\right)\left(\frac{\partial}{\partial\ln T}\left(\frac{\partial e}{\partial X_i}\right)_{\rho,\mathrm{AD},(X_j,j\neq i)}\right)_{\rho,X}X_i-\chi_{X_i}\right]\frac{\Gamma_{4,i}}{\Gamma_1}\right]^{-1}
\end{aligned}$$

$$\boxed{\alpha=\frac{1}{\Gamma_1 p}\left[1+\sum_i\left[\frac{\rho^2 p_\rho}{pp_T}(1-\gamma)\frac{N_\mathrm{A}}{A_i}\left(\frac{\partial\mu_i}{\partial T}\right)_{\rho,X}X_i-\chi_{X_i}\right]\frac{\Gamma_{4,i}}{\Gamma_1}\right]^{-1}}$$

Recalling Derivation of β_0

Recall from paper I that β_0 was derived from the equation

$$\nabla\cdot\mathbf{U}+\alpha\mathbf{U}\cdot\nabla p_0=\tilde{S} \tag{28.15}$$

in such a fashion that we ended up with an equation of the form

$$\nabla\cdot(\beta_0(r)\mathbf{U})=\beta_0\tilde{S}. \tag{28.16}$$

The derivation in Appendix B of paper I for a β_0 that satisfies (28.16) automatically assumed $\alpha = (\Gamma_{1_0}p_0)^{-1}$. This would have to be modified with the above derivation of α to be correct in a non-operator split fashion.

CHAPTER 29

Notes on Enthalpy

Evolution Equations

The compressible and low Mach number formulations of the governing equations both share the unapproximated continuity and momentum equations shown here (where $p = p_0 + \pi$).

$$\frac{\partial(\rho X_k)}{\partial t} = -\nabla \cdot (\rho X_k \mathbf{U}) + \rho \dot{\omega}_k, \tag{29.1}$$

$$\frac{\partial \mathbf{U}}{\partial t} = -\mathbf{U} \cdot \nabla \mathbf{U} - \frac{1}{\rho}\nabla \pi - \frac{\rho - \rho_0}{\rho} g \mathbf{e}_r, \tag{29.2}$$

In the compressible formulation we complete the system with an energy equation as well as an equation of state; in the low Mach number formulation we can derive a constraint on the velocity by setting $p_{EOS} = p(\rho, h, X_k) = p_0$ and differentiating the equation of state along particle paths. In this case adding the energy equation would over-constrain the system, but its solution can help us in providing a diagnostic capability for the solution. We can write the energy equation in terms of internal energy, e, or enthalpy, h; these two equations are analytically equivalent.

$$\begin{aligned} \frac{\partial(\rho h)}{\partial t} + \nabla \cdot (\rho h \mathbf{U}) &= \frac{Dp}{Dt} + \rho H_{\text{nuc}} \\ \frac{\partial(\rho e)}{\partial t} + \nabla \cdot (\rho e \mathbf{U}) &= -p \nabla \cdot \mathbf{U} + \rho H_{\text{nuc}} \end{aligned}$$

In low Mach number combustion, $Dp/Dt = 0$, which makes the enthalpy equation preferable to the internal energy equation because we don't need to evaluate $p \nabla \cdot \mathbf{U}$.

Derivation of Velocity Constraint

Differentiating the equation of state, written in the form, $p = p(\rho, T, X_k)$, along particle paths, we can write

$$\frac{Dp}{Dt} = p_\rho \frac{D\rho}{Dt} + p_T \frac{DT}{Dt} + \sum_k p_{X_k} \frac{DX_k}{Dt} \tag{29.3}$$

Then, by rearranging the terms, we get

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{U} = \frac{1}{p_\rho} \left(\frac{Dp}{Dt} - p_T \frac{DT}{Dt} - \sum_k p_{X_k} \dot{\omega}_k \right) \ , \tag{29.4}$$

with $p_\rho = \partial p / \partial \rho |_{X_k, T}$, $p_{X_k} = \partial p / \partial X_k |_{T, \rho, (X_j, j \neq k)}$, and $p_T = \partial p / \partial T |_{\rho, X_k}$.

Using the Enthalpy Equation

Now writing $h = h(p, T, X_k)$ and expanding Dh/Dt and using the enthalpy evolution equation:

$$\rho \frac{Dh}{Dt} = \rho \left(h_p \frac{Dp}{Dt} + c_p \frac{DT}{Dt} + \sum_k h_{X_k} \frac{DX_k}{Dt} \right) = \frac{Dp}{Dt} + \rho H_{\rm nuc} \tag{29.5}$$

we can express DT/Dt in terms of

$$\frac{DT}{Dt} = \frac{1}{\rho c_p} \left((1 - \rho h_p) \frac{Dp}{Dt} - \sum_k \rho \xi_k \dot{\omega}_k + \rho H_{\rm nuc} \right) \ , \tag{29.6}$$

where $c_p = \partial h / \partial T |_{p, X_k}$ is the specific heat at constant pressure, $\xi_k = \partial h / \partial X_k |_{p,T}$, and $h_p = \partial h / \partial p |_{T, X_k}$.

We could then write,

$$\begin{aligned} \nabla \cdot \mathbf{U} &= \frac{1}{\rho p_\rho} \left(-\frac{Dp}{Dt} + \frac{p_T}{\rho c_p} \left((1 - \rho h_p) \frac{Dp}{Dt} - \rho \sum_k \xi_k \dot{\omega}_k + \rho H_{\rm nuc} \right) + \sum_k p_{X_k} \dot{\omega}_k \right) \\ &= \frac{1}{\rho p_\rho} \left(\frac{p_T}{\rho c_p} (1 - \rho h_p) - 1 \right) \frac{Dp}{Dt} + \frac{1}{\rho p_\rho} \left(\frac{p_T}{\rho c_p} (\rho H_{\rm nuc} - \rho \sum_k \xi_k \dot{\omega}_k) + \sum_k p_{X_k} \dot{\omega}_k \right) \ . \end{aligned}$$

When we derived this expression we explicitly retained the dependence of h on p, as shown by the presence of the h_p term.

Then, replacing p by $p_0(r)$, Dp/Dt becomes $\mathbf{U} \cdot \nabla p_0$, and the divergence constraint can be written

$$\nabla \cdot \mathbf{U} + \alpha \mathbf{U} \cdot \nabla p_0 = \frac{1}{\rho p_\rho} \left(\frac{p_T}{\rho c_p} \left(-\sum_k \rho \xi_k \dot{\omega}_k + \rho H_{\rm nuc} \right) + \sum_k p_{X_k} \dot{\omega}_k \right) \equiv \tilde{S} \ , \tag{29.7}$$

where we define

$$\alpha(\rho, T) \equiv - \left(\frac{(1 - \rho h_p) p_T - \rho c_p}{\rho^2 c_p p_\rho} \right) \ . \tag{29.8}$$

Using the Energy Equation

Now writing $e = e(p, T, X_k)$ and expanding De/Dt and using the energy evolution equation:

$$\rho \frac{De}{Dt} = \rho \left(e_p \frac{Dp}{Dt} + e_T \frac{DT}{Dt} + \sum_k e_{X_k} \frac{DX_k}{Dt} \right) = -p\nabla \cdot \mathbf{U} + \rho H_{\text{nuc}} \tag{29.9}$$

we can express DT/Dt in terms of

$$\frac{DT}{Dt} = \frac{1}{\rho e_T} \left(-p\nabla \cdot \mathbf{U} + \rho H_{\text{nuc}} - \rho e_p \frac{Dp}{Dt} - \rho \sum_k e_{X_k} \dot{\omega}_k \right)$$

where $e_T = \partial e/\partial T|_{p,X_k}$ and $e_p = \partial e/\partial p|_{T,X_k}$. Then

$$-\rho\nabla \cdot \mathbf{U} = \frac{1}{p_\rho} \left(\frac{Dp}{Dt} - \frac{p_T}{\rho e_T} \left(-p\nabla \cdot \mathbf{U} + \rho H_{\text{nuc}} - \rho e_p \frac{Dp}{Dt} - \rho \sum_k e_{X_k} \dot{\omega}_k \right) - \sum_k p_{X_k} \dot{\omega}_k \right) \quad , \tag{29.10}$$

which leads to

$$\left(-\rho - \frac{p p_T}{\rho e_T p_\rho} \right) \nabla \cdot \mathbf{U} = \frac{1}{p_\rho} \left((1 + \frac{e_p p_T}{e_T}) \frac{Dp}{Dt} - \frac{p_T}{\rho e_T} \left(\rho H_{\text{nuc}} - \rho \sum_k e_{X_k} \dot{\omega}_k \right) - \sum_k p_{X_k} \dot{\omega}_k \right) \quad , \tag{29.11}$$

Note that we can replace p by p_0 in the coefficient on the l.h.s. as well as on the r.h.s.

Comparison of Constraints

If we set $\dot{\omega}_k = H_{\text{nuc}} = 0$ for simplicity, then the constraint as derived using h can be written

$$\nabla \cdot \mathbf{U} + \left(\frac{(1 - \rho h_p) p_T - \rho c_p}{\rho^2 c_p p_\rho} \right) \frac{Dp_0}{Dt} = 0 \tag{29.12}$$

and the constraint derived using e can be written

$$\nabla \cdot \mathbf{U} + \left(\frac{\rho e_T + \rho e_p p_T}{\rho^2 e_T p_\rho + p p_T} \right) \frac{Dp_0}{Dt} = 0 \tag{29.13}$$

We note that if we evaluate both constraints for $p = \rho RT$, with $h = c_p T$, $e = c_v T$, $c_p = c_v + R$ and $\gamma = c_p/c_v$, then both constraints reduce to

$$\nabla \cdot \mathbf{U} + \frac{1}{\gamma p} \frac{Dp_0}{Dt} = 0 \tag{29.14}$$

Enthalpy vs Energy Equation

The full enthalpy equation, with no approximations, appears as:

$$\frac{\partial(\rho h)}{\partial t} = -\nabla \cdot (\rho h \mathbf{U}) + \frac{Dp}{Dt} + \rho H_{\text{nuc}} \tag{29.15}$$

Here, $h = e + p/\rho$ is the specific enthalpy, with e the specific internal energy. In the low Mach number formulation, we replace p with p_0 in the Dp/Dt term, however, the definition of enthalpy implicitly contains a pressure. When calling the equation of state, we take h and ρ as inputs. The equation of state is expressed in terms of T and ρ, so it iterates until it finds the h that we desire. This h will be of the form $h = e + p_{\rm EOS}/\rho$, where $p_{\rm EOS}$ is the pressure returned from the EOS. Note that $p_{\rm EOS}$ may not be equal to p_0—this may be what causes us to drive off of the constraint.

The mismatch between the pressure implicit in the definition of h and p_0 can be seen by substituting $h = e + p/\rho$ into the enthalpy equation, where we replace p with p_0 in the Dp/Dt term:

$$\begin{aligned}
\frac{\partial(\rho h)}{\partial t} &= -\nabla\cdot(\rho h\mathbf{U}) + \frac{Dp_0}{Dt} + \rho H_{\rm nuc} \\
\frac{\partial(\rho e)}{\partial t} + \frac{\partial p}{\partial t} &= -\nabla\cdot(\rho e\mathbf{U}) - \nabla\cdot(p\mathbf{U}) + \frac{Dp_0}{Dt} + \rho H_{\rm nuc} \\
\frac{\partial(\rho e)}{\partial t} &= -\nabla\cdot(\rho e\mathbf{U}) - p\nabla\cdot\mathbf{U} + \rho H_{\rm nuc} + \left\{\frac{Dp_0}{Dt} - \frac{Dp}{Dt}\right\}
\end{aligned}$$

However, if we solve the evolution equation for e we would substitute p_0 for p in this equation as well. Thus, we can pose the situation as the following. If we solve the evolution equation for h then we effectively are solving

$$\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho e\mathbf{U}) = -p\ \nabla\cdot\mathbf{U} + \rho H_{\rm nuc} + \left\{\frac{Dp_0}{Dt} - \frac{Dp}{Dt}\right\}$$

but if we solve the evolution equation for e we are effectively solving

$$\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho e\mathbf{U}) = p_0\nabla\cdot\mathbf{U} + \rho H_{\rm nuc}$$

The second equation subtracted from the first gives:

$$\frac{D(p_0 - p)}{Dt} - (p_0 - p)\nabla\cdot\mathbf{U} = 0, \tag{29.16}$$

but this equation is only true, in general, if $p = p_0$.

Suppose we solve the current enthalpy equation, but when we call the EOS, we subtract p_0 from ρh and then call the EOS with e instead of h. This is equivalent to:

$$\begin{aligned}
\frac{\partial(\rho h)}{\partial t} &= -\nabla\cdot(\rho h\mathbf{U}) + \frac{Dp_0}{Dt} + \rho H_{\rm nuc} \\
\frac{\partial(\rho e)}{\partial t} + \frac{\partial p_0}{\partial t} &= -\nabla\cdot(\rho e\mathbf{U}) - \nabla\cdot(p_0\mathbf{U}) + \frac{Dp_0}{Dt} + \rho H_{\rm nuc} \\
\frac{\partial(\rho e)}{\partial t} &= -\nabla\cdot(\rho e\mathbf{U}) - p_0\nabla\cdot\mathbf{U} + \rho H_{\rm nuc}
\end{aligned}$$

which is identical to solving the energy equation with $p \to p_0$. This option is enabled in MAESTRO via `use_eos_e_instead_of_h = T`.

Constant γ Gas

Going back to the constant γ, ideal gas EOS, we can rewrite the enthalpy equation as a pressure evolution equation

$$\begin{aligned} \frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho h \mathbf{U}) &= \frac{Dp_0}{Dt} + \rho H \\ \frac{\gamma}{\gamma - 1}\frac{\partial p}{\partial t} + \frac{\gamma}{\gamma - 1}\nabla \cdot (p\mathbf{U}) &= \frac{Dp_0}{Dt} + \rho H \\ \frac{Dp}{Dt} &= -p\nabla \cdot \mathbf{U} + \frac{\gamma}{\gamma - 1}\left(\frac{Dp_0}{Dt} + \rho H\right). \end{aligned} \tag{29.17}$$

Similarly, we can derive a pressure evolution equation from the energy equation

$$\frac{Dp}{Dt} = -p\nabla \cdot \mathbf{U} - (\gamma - 1)\, p_0 \nabla \cdot \mathbf{U} + \rho H \tag{29.18}$$

Now, if we further make the assumption that p_0 is constant, $Dp_0/Dt = 0$, then the divergence constraint for such a gas reduces to

$$\nabla \cdot \mathbf{U} = \frac{\gamma - 1}{\gamma p_0}\rho H. \tag{29.19}$$

Plugging this back into either of (29.17) or (29.18) gives

$$\frac{Dp}{Dt} = \frac{\gamma - 1}{\gamma}\left(1 - \frac{p}{p_0}\right)\rho H. \tag{29.20}$$

If p_0 is assumed constant and using (29.19), the difference between the enthalpy equation and the energy equation, (29.16), can be rewritten as

$$-\frac{Dp}{Dt} - \frac{\gamma}{\gamma - 1}\left(1 - \frac{p}{p_0}\right)\rho H = 0,$$

where the equality holds from (29.20). In other words, for the constant γ gas we have $p = p_0$ as expected.

Outstanding Questions

1. Why do we want to start with enthalpy instead of internal energy?

   We believe that the original desire stems from our experience with smallscale combustion. There, stratification is not important and $Dp_0/Dt = 0$, so the enthalpy equation becomes a conservation equation for (ρh).

2. Should we call the EOS with h as is, or call the EOS with $e = h - p_0/\rho$?

3. When we stay on the constraint, i.e. $p_{EOS} = p_0$, then the equations for e and for h are identical. However, once we are off the constraint, do the terms in the current evolution equation for h serve to drive us back to the constraint? Recall our current "volume discrepancy factor" acts as a source term which modifies the divergence constraint, which effectively modifies both ρ and T (or e or h). The term in the enthalpy equation only modifies ρ. Is this relevant and/or useful??

Recall that the current "volume discrepancy factor" takes the form of adding to the r.h.s. of the constraint:

$$\nabla \cdot (\beta_0 \mathbf{U}) = \beta_0 \left(S - \frac{1}{\overline{\Gamma_1} p_0} \frac{\partial p_0}{\partial t} - \frac{f}{\overline{\Gamma_1} p_0} \frac{p_0 - p_{\mathrm{EOS}}}{\Delta t} \right) \tag{29.21}$$

4. Suppose we corrected the h (or e equation) by using the full p instead of p_0? Would this be more or less consistent (one could imagine doing this as a correction after solving for π earlier in the timestep).

5. In computing the thermodynamic coefficients in S for the projection, don't we need these to be in terms of p_0 instead of $p(h, \rho)$?

Part IV

References and Index

References

[1] A. Almgren. A new look at the pseudo-incompressible solution to Lamb's problem of hydrostatic adjustment. 57:995–998, April 2000.

[2] 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. *ApJ*, 715:1221–1238, June 2010.

[3] A. S. Almgren, J. B. Bell, and W. Y. Crutchfield. Approximate projection methods: Part I. Inviscid analysis. *SIAM J. Sci. Comput.*, 22(4):1139–59, 2000.

[4] A. S. Almgren, J. B. Bell, A. Nonaka, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. III. Reactions. *ApJ*, 684:449–470, September 2008.

[5] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. I. Hydrodynamics. *ApJ*, 637:922–936, February 2006.

[6] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. II. Energy Evolution. *ApJ*, 649:927–938, October 2006.

[7] A. S. Almgren, J. B. Bell, and W. G. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM J. Sci. Comput.*, 17(2):358–369, March 1996.

[8] A. J. Aspden, J. B. Bell, M. S. Day, S. E. Woosley, and M. Zingale. Turbulence-Flame Interactions in Type Ia Supernovae. *ApJ*, 689:1173–1185, December 2008.

[9] J. B. Bell, P. Colella, and H. M. Glaz. A Second Order Projection Method for the Incompressible Navier-Stokes Equations. *Journal of Computational Physics*, 85:257, December 1989.

[10] J. B. Bell, P. Colella, and L. H. Howell. An efficient second-order projection method for viscous incompressible flow. In *Proceedings of the Tenth AIAA Computational Fluid Dynamics Conference*, pages 360–367. AIAA, June 1991.

[11] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. Zingale. Direct numerical simulations of Type Ia supernovae flames I: The Landau-Darrieus instability. *ApJ*, 606:1029, 2004.

[12] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. Zingale. Direct numerical simulations of Type Ia supernovae flames II: The Rayleigh-Taylor instability. *ApJ*, 608:883, 2004.

[13] 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. *Journal of Computational Physics*, 195:677–694, April 2004.

[14] J. B. Bell and D. L. Marcus. A second-order projection method for variable-density flows. 101(2):334–348, August 1992.

[15] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. VODE: A variable coefficient ode solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989.

[16] G. R. Caughlan and W. A. Fowler. Thermonuclear reaction rates V. *Atomic Data and Nuclear Data Tables*, 40(2):283–334, 1988. see also `http://www.phy.ornl.gov/astrophysics/data/cf88/index.html`.

[17] D. A. Chamulak, E. F. Brown, F. X. Timmes, and K. Dupczak. The Reduction of the Electron Abundance during the Pre-explosion Simmering in White Dwarf Supernovae. *ApJ*, 677:160–168, April 2008.

[18] Richard H. Cyburt, A. Matthew Amthor, Ryan Ferguson, Zach Meisel, Karl Smith, Scott Warren, Alexander Heger, R. D. Hoffman, Thomas Rauscher, Alexander Sakharuk, Hendrik Schatz, F. K. Thielemann, and Michael Wiescher. The jina reaclib database: Its recent updates and impact on type-i x-ray bursts. *ApJS*, 189(1):240, 2010.

[19] M. S. Day and J. B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4(4):535–556, 2000.

[20] D. R. Durran. Improving the anelastic approximation. 46(11):1453–1461, 1989.

[21] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *ApJS*, 131:273–334, November 2000.

[22] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids*, 8:2182–2189, December 1965.

[23] Rupert Klein and Olivier Pauluis. Thermodynamic consistency of a pseudoincompressible approximation for general equations of state. *JAS*, March 2012.

[24] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[25] C. M. Malone, A. Nonaka, A. S. Almgren, J. B. Bell, and M. Zingale. Multidimensional Modeling of Type I X-ray Bursts. I. Two-dimensional Convection Prior to the Outburst of a Pure ^{4}He Accretor. *ApJ*, 728:118–+, February 2011.

[26] C. M. Malone, M. Zingale, A. Nonaka, A. S. Almgren, and J. B. Bell. Multidimensional Modeling of Type I X-Ray Bursts. II. Two-dimensional Convection in a Mixed H/He Accretor. *ApJ*, 788:115, June 2014.

[27] 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. *ApJS*, 188:358–383, June 2010.

[28] A. Nonaka, A. J. Aspden, M. Zingale, A. S. Almgren, J. B. Bell, and S. E. Woosley. High-resolution Simulations of Convection Preceding Ignition in Type Ia Supernovae Using Adaptive Mesh Refinement. *ApJ*, 745:73, January 2012.

[29] A Nonaka, J. B. Bell, M. S. Day, C. Gilet, A. S. Almgren, and M. L. Minion. A deferred correction coupling strategy for low mach number flow with complex chemistry. *Combustion Theory and Modeling*, 16(6):1053–1088, 2012.

[30] A. Nonaka, S. May, A. S. Almgren, and J. B. Bell. A Three-Dimensional, Unsplit Godunov Method for Scalar Conservation Laws. *SIAM J. Sci. Comput.*, 33(4):2039–2062, 2011.

[31] P. M. Ricker. A Direct Multigrid Poisson Solver for Oct-Tree Adaptive Meshes. *ApJS*, 176:293–300, May 2008.

[32] F. X. Timmes and F. D. Swesty. The Accuracy, Consistency, and Speed of an Electron-Positron Equation of State Based on Table Interpolation of the Helmholtz Free Energy. *ApJS*, 126:501–516, February 2000.

[33] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman. yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data. *ApJS*, 192:9, January 2011.

[34] Geoffrey M. Vasil, Daniel Lecoanet, Benjamin P. Brown, Toby S. Wood, and Ellen G. Zweibel. Energy conservation and gravity waves in sound-proof treatments of stellar interiors. ii. lagrangian constrained analysis. 773:169–, 2013.

[35] R. K. Wallace and S. E. Woosley. Explosive hydrogen burning. *ApJS*, 45:389–420, February 1981.

[36] A. Weiss, W. Hillebrandt, H.-C. Thomas, and H. Ritter. *Cox & Giuli's Principles of Stellar Structure*. Cambridge Scientific Publishers, 2004.

[37] M. Zingale, A. S. Almgren, J. B. Bell, A. Nonaka, and S. E. Woosley. Low Mach Number Modeling of Type IA Supernovae. IV. White Dwarf Convection. *ApJ*, 704:196–210, October 2009.

[38] M. Zingale, A. Nonaka, A. S. Almgren, J. B. Bell, C. M. Malone, and S. E. Woosley. The Convective Phase Preceding Type Ia Supernovae. *ApJ*, 740:8, October 2011.

[39] M Zingale, S. E. Woosley, C. A. Rendleman, M. S. Day, and J. B. Bell. Three-dimensional Numerical Simulations of Rayleigh-Taylor Unstable Flames in Type Ia Supernovae. *ApJ*, 632:1021, 2005.

Index