Chapter 5

Hydrodynamics

> Reality continues to ruin my life.
>
> Bill Watterson

Gas and fluid dynamics play an important role in many astrophysical processes. When addressing these physical phenomena, we refer to the collection of solvers by the generic term *hydrodynamics*. Gas dynamics forms the basis for our understanding of star and planet formation; it drives X-ray sources and accretion onto supermassive black holes; it dictates the appearance of galaxies. In AMUSE, hydrodynamics is addressed numerically using two quite distinct methods: the Lagrangian formalism with smoothed particles, and Eulerian grid-based finite volume formalisms. These approaches are complementary. Hybrid approaches that combine the advantages of both methods also exist, but are (currently) unavailable in AMUSE. Simulating hydrodynamics is an art, but one with a solid mathematical background.

5.1 In a Nutshell

Hydrodynamics deals with the motion of fluids driven by internal and external forces, for example due to pressure gradients and gravity. Although fluid dynamics is important for everyday life, we discuss it here in the context of astronomical phenomena. That means we will largely ignore many important fluid properties, such as viscosity and surface tension, that are critical to Earth-bound applications.

In AMUSE, we confine ourselves to a continuum description of the gas—we do not attempt to describe the microscopic behavior of its constituent particles. In doing so, we are assuming that we can sensibly define thermodynamic quantities, such as density and temperature, on the spatial scales of interest. In numerical simulations, the spatial variation of fluid properties may be represented in a variety of different ways—on a regular grid, or an irregular tessellation, or in terms of a collection of macroscopic particles that sample the fluid at different locations. In all cases, if the sampling scale is λ, the gas has number density n, and we are interested in a physical flow with characteristic length scale L, then we require $\lambda \ll L$ and $n\lambda^3 \gg 1$ (that is, a resolution element should contain many particles but still be much smaller than the length scales

of interest). Fortunately, these requirements are often met and we can make reliable quantitative statements about the fluid flow.

Some astrophysical processes—for example, jets, stellar winds, and supernova explosions—have symmetries that may allow a good approximation by reducing the dimensionality of the problem, but often no such simplification exists. In addition, there is a wide range of instabilities for which the typical length scale depends on the dimensionality of the problem, as in the Rayleigh–Taylor (dense fluid on top of a low-density fluid) and Kelvin–Helmholtz (shear between two fluids) instabilities. For these reasons, we address here only the hardest problem—fluid dynamics in three dimensions.

Solving the fundamental equations of hydrodynamics is one of the hardest numerical problems in computational science. Three-dimensional fluid flows are very computationally demanding. The underlying equations are nonlinear and the dynamic range may exceed algorithmic capabilities by orders of magnitude. At the same time, there is only a very limited set of analytic solutions against which numerical implementations can be tested (Coggeshall, 1991). Laboratory experiments are possible, but it is often hard to compare them quantitatively with simulations, especially in astrophysics. There are a number of excellent resources on hydrodynamics, ranging from basic but comprehensive introductions (Landau & Lifshitz, 1959) to more astronomically oriented texts (Shore, 2007; Regev *et al.*, 2016).

5.1.1 Underlying Equations

Astrophysical fluids can be described by a relatively small number of variables. A fluid element has position $\mathbf{x}$ and velocity $\mathbf{v}$, as well as internal thermodynamic properties, such as density ρ, composition, pressure P, temperature T, entropy s, and so on. For astrophysical systems, the composition is usually no more complicated than specifying just the hydrogen mass fraction X, the helium fraction Y, and/or a "metal" fraction $Z = 1 - X - Y$ that accounts for all the heavier elements. The connection among the thermodynamic variables is provided by the equation of state (see Section 5.1.1.3). The dynamics of a fluid can be described by three independent partial differential equations, which express the conservation of mass, momentum, and energy.

5.1.1.1 Conservation of Mass

The rate of change of the mass m in some volume V is equal to the rate at which mass flows into or out of V across its surface S:

$$\frac{dm}{dt} = -\int_S \rho \mathbf{v} \cdot d^2\mathbf{S}, \tag{5.1}$$

where $d^2\mathbf{S}$ is the (outward-pointing) vector element of area on S. Writing

$$m = \int_V \rho \, d^3V \tag{5.2}$$

and using the divergence theorem on the right side of Equation 5.1, we find

$$\frac{dm}{dt} = \int_V \frac{\partial \rho}{\partial t} \, d^3V = -\int_V \nabla \cdot (\rho \mathbf{v}) \, d^3V. \tag{5.3}$$

Because this must be true for any volume V, we immediately obtain the *continuity equation*

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0. \tag{5.4}$$

In the frame of a moving fluid element, we can recast this in terms of the so-called Lagrangian (or comoving) derivative

$$\frac{d}{dt} \equiv \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla \tag{5.5}$$

to write

$$\frac{d\rho}{dt} + \rho \nabla \cdot v = 0. \tag{5.6}$$

5.1.1.2 Conservation of Momentum

Newton's second law (see also Equation 3.1), applied to a fluid element, is simply

$$\rho \frac{d\mathbf{v}}{dt} = \mathbf{F}, \tag{5.7}$$

where $\mathbf{F}$ is the net force per unit volume and "d" represents a Lagrangian derivative. In an astrophysical context, the most important forces contributing to the right side of Equation 5.7 are pressure ($\mathbf{F} = -\nabla P$) and gravity ($\mathbf{F} = \rho \mathbf{g} = -\rho \nabla \phi$), so

$$\frac{d\mathbf{v}}{dt} = -\frac{\nabla P}{\rho} + \mathbf{g}. \tag{5.8}$$

Equation 5.8 is known as the *Euler equation*. Other forces (e.g., radiation pressure) can be included in an analogous way. Expanding out the Lagrangian derivative, we find

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{\nabla P}{\rho} + \mathbf{g}. \tag{5.9}$$

Note the nonlinear term on the left side, which is the origin of many of the most intractable problems in computational fluid dynamics.

Thus far, we have neglected internal viscous stresses ($\mathbf{F} = \rho \, \mathbf{f}_{\rm visc}$) in the Euler equation. By adopting a specific model for the inclusion of including viscous stresses in Equations 5.8 and 5.9, we obtain the *Navier–Stokes equation*. For an incompressible flow, the viscous term can be written

$$\mathbf{f}_{\rm visc} = \nu \nabla^2 \mathbf{v}, \tag{5.10}$$

where ν is the kinematic viscosity. Although physical viscosities are often too small to be important for large-scale astrophysical flows, similar *artificial viscosity* terms are used in many simulations to suppress short-wavelength features that would otherwise fall below the resolution of the simulation (Von Neumann & Richtmyer, 1950; see Section 5.1.2.2).

5.1.1.3 The Equation of State

We now have four equations (continuity and Euler) involving five fluid quantities: $\mathbf{v}$, ρ, and P. In order to solve this system of equations, we need to close it. In principle, this can be accomplished by adopting an equation of state, which connects ρ and P.

We discussed some possible equations of state in Section 4.1.2.1. Many of the hydrodynamics modules in AMUSE adopt a simple expression of the form

$$P = A(s)\rho^{\gamma}, \tag{5.11}$$

where γ is the adiabatic index and s is specific entropy. For a monatomic adiabatic ideal gas, $\gamma = \frac{5}{3}$ and s is constant, so P is a function of ρ only. In these cases, the entropy s can only change in shocks (see Section 5.1.2.2), so the flow is adiabatic except when shocks are encountered (Longair, 2011). Sometimes, the index γ is allowed to vary in a density-dependent way to model heating and cooling processes. An isothermal equation of state is modeled simply by forcing the temperature to the desired value at the end of every step.

In reality, however, the true equation of state usually involves other thermodynamic variables, such as temperature $T = T(P, \rho)$ or entropy $s = s(P, \rho)$, and we need additional equations to complete the system. For example, to determine the temperature, we may need to take into account heating and cooling processes in the gas, adding an energy equation to close the set of equations:

$$\Delta e = Q - P\Delta V. \tag{5.12}$$

Here, e is the internal energy per unit mass of the gas, $V = 1/\rho$ is specific volume, and $Q = T\Delta s$ is the net heating, which may have contributions from a variety of sources, such as dissipation, nuclear reactions, cosmic rays, radiative heating and cooling, etc. If the gas cools at a rate $\mathcal{C}$ and is heated at a rate $\mathcal{H}$ (both per unit mass), then the energy equation is

$$\frac{de}{dt} = \mathcal{H} - \mathcal{C} - \frac{P}{\rho}\nabla \cdot \mathbf{v}. \tag{5.13}$$

The simplified equations of state described previously are approximations to the solutions of this equation.

The full equation of state is determined by a number of processes, including the degree of ionization of the medium, its chemistry and composition, the presence of nuclear reactions, heating by cosmic rays, and various radiative processes. The most sophisticated (and costly) treatments model these processes explicitly; the temperature and ionization structure of the gas is determined by equating the heating and cooling rates, or it may be provided by an additional module that calculates the radiative structure of the system (see Chapter 6). Once the temperature T and ionization fraction of the gas have been determined, the pressure P in a star-forming region of density ρ is accurately given by

$$P = \frac{\rho kT}{\mu} + \tfrac{1}{3}aT^4 \tag{5.14}$$

(cf. Equation 4.16), where k is the Boltzmann constant, μ is the mean molecular weight with the dimension of mass (rather than another common definition in which it is normalized to the mass of hydrogen), and a is the radiation constant.

5.1.2 Turbulence and Shocks

Fluids are commonly characterized by several dimensionless parameters that describe the overall state of the flow. Two of the most important are the *Reynolds number*, which quantifies the effect of viscosity, and the *Mach number*, which specifies the speed of the flow.

5.1.2.1 Reynolds Number

The Reynolds number Re is defined as the ratio of inertial to viscous forces in a fluid flow. If the flow has length scale L, velocity scale v, and kinematic viscosity ν, then dimensional analysis leads to the expression

$$Re = \frac{vL}{\nu}. \tag{5.15}$$

For low Reynolds number, $Re \ll 1$, viscous forces dominate and the flow is smooth. Fluids with $Re \gg 1$ tend to be turbulent.

For $Re \to \infty$, the flow behaves as a perfect inviscid fluid ($\nu = 0$), but this does not mean that viscosity is irrelevant. Turbulent flow is characterized by motion on a wide range of scales, from the largest to the smallest turbulent eddies. The nonlinear term in the Euler equation sees to it that energy in the large-scale motion with $Re \gg 1$ is systematically transferred to smaller and smaller scales, eventually reaching a scale where $Re \sim 1$ and viscosity can efficiently dissipate the energy. These small scales are often out of reach for numerical simulations, with the result that turbulence acts as an effective viscosity, and high Reynolds number flows are very difficult to model numerically.

5.1.2.2 Mach Number

The Mach number M is the ratio of the fluid speed to the local sound speed

$$M = \frac{v}{c_s}. \tag{5.16}$$

For an adiabatic flow, each fluid element has constant entropy and the sound speed is

$$c_s = \sqrt{\frac{\gamma P}{\rho}} = \sqrt{\frac{\gamma k_B T}{\mu}}, \tag{5.17}$$

where γ is the adiabatic index. Thus, the Mach number depends on temperature through c_s. The last equality of Equation 5.17 assumes an ideal gas.

Flows with $M \lesssim 0.3$ are well described by the simplified incompressible fluid equations. Supersonic flows ($M > 1$) lead to shock waves. Simply put, given that information in a fluid travels at the speed of sound, material in front of a supersonic

disturbance cannot receive any warning before the disturbance arrives, so it can't "get out of the way." As a result, material piles up in front of the disturbance as it propagates through the fluid, forming a shock. Astrophysics abounds with situations in which shock waves can form. Sometimes, as with stellar outflows and supernovae, the disturbance may move at thousands of kilometers per second through the surrounding medium. In other circumstances, such as the collision of two interstellar clouds, the speed of the disturbance may be quite modest—only a few kilometers per second—but still much greater than the local sound speed in a cool gas.

As illustrated in Figure 5.1, in the frame moving with the shock, high-speed material (index 1) flows in from the left while low-speed material (index 2) flows out to the right. In the vicinity of a shock, the Mach number is always $M > 1$ before the shock and $M < 1$ behind it. Within this simple one-dimensional geometry, we can write the equations describing the flow in flux-conservative form:

$$\begin{aligned} \frac{\partial \rho}{\partial t} &= -\frac{\partial}{\partial x}(\rho u), \\ \frac{\partial}{\partial t}(\rho u) &= -\frac{\partial}{\partial x}(\rho u^2 + P), \\ \frac{\partial E}{\partial t} &= -\frac{\partial}{\partial x}\left[u(E+P)\right], \end{aligned}$$

where u is the fluid speed and $E = \rho e + \frac{1}{2}\rho u^2$ is the total energy density of the fluid, with e the specific internal energy.

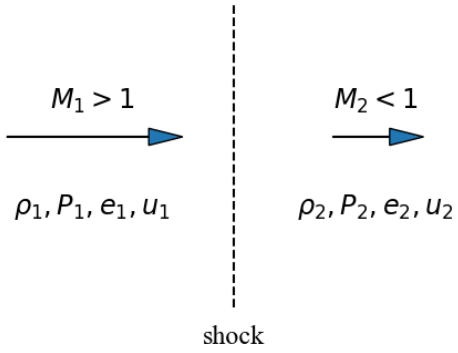

Figure 5.1: Geometry of a plane shock wave, in the frame of the shock.

Integrating these equations across the shock discontinuity and assuming a steady state, for a stationary shock in its frame of reference, we obtain the *Rankine–Hugoniot conditions* (Rankine, 1870; Hugoniot, 1885), expressing conservation of mass, momentum, and energy across the shock:

$$\begin{aligned} \rho_1 u_1 &= \rho_2 u_2, \\ \rho_1 u_1^2 + P_1 &= \rho_2 u_2^2 + P_2, \\ \rho_1 u_1 \left(e_1 + \tfrac{1}{2}u_1^2 + P_1/\rho_1\right) &= \rho_2 u_2 \left(e_2 + \tfrac{1}{2}u_2^2 + P_2/\rho_2\right) \end{aligned}$$

Note that we can use the first condition to simplify the third and give Bernoulli's principle (Bernoulli, 1738): $e_1 + 0.5u_1^2 + P/\rho_1 = e_2 + 0.5u_2^2 + P/\rho_2$.

These equations can be solved for the jump in various quantities of interest across the shock, expressed here in terms of the pre-shock Mach number $M_1 = u_1/c_{s1}$ (Landau & Lifshitz, 1959):

$$\begin{aligned}
\frac{\rho_2}{\rho_1} &= \frac{u_1}{u_2} = \frac{(\gamma+1)M_1^2}{(\gamma-1)M_1^2+2}, \\
\frac{P_2}{P_1} &= \frac{2\gamma M_1^2-(\gamma-1)}{\gamma+1}, \\
\frac{T_2}{T_1} &= \frac{[2\gamma M_1^2-(\gamma-1)]\,[(\gamma-1)M_1^2+2]}{(\gamma+1)^2 M_1}, \\
M_2^2 &= \frac{2+(\gamma-1)M_1^2}{2\gamma M_1^2-(\gamma-1)}
\end{aligned}$$

The thickness of the shock is ultimately determined by the viscosity of the fluid (see Equation 5.10), which turns pre-shock kinetic energy into post-shock thermal energy and entropy. For systems without large-scale viscosity or explicit heating and cooling mechanisms, shocks are the only regions where the gas can increase its entropy. In many astrophysical contexts, the fluid viscosity is very low, so the shock is well-approximated as a discontinuous jump in fluid properties from left to right, as just described. However, in simulations, numerical and artificial viscosity can exceed the true physical viscosity by many orders of magnitude, spreading out the shock and limiting the resolution of the flow.

5.1.3 Hydrodynamical Time Scales

To further develop our intuition, we define some time scales relevant to astrophysical phenomena. In Table 5.1, we provide an overview of various hydrodynamical regimes and their characteristic parameters, such as temperature and density. The range of parameters spans many orders of magnitude, and the physics may vary considerably across this range of astronomical objects, but practically all of these can be (and are) addressed with the hydrodynamical models described in this chapter.

5.1.3.1 Free-fall Time Scale

As with a self-gravitating stellar system, we can define a collapse (or freefall[**freefall or free-fall? – Steven**]) time scale for a gaseous cloud. This time scale can be written

$$t_{\rm ff} = \frac{1}{\sqrt{G\rho}}. \tag{5.18}$$

Note that, because $\rho \sim M/R^3$, this is essentially the dynamical time scale defined in Equation 3.5. For a 1 $M_\odot$ molecular cloud in a cubic parsec volume, $\rho \simeq 7\times10^{-23}\,{\rm g/cm^3}$ and $t_{\rm ff} \simeq 15\,{\rm Myr}$.

The self-gravity of a gas is often ignored in hydrodynamical simulations, particularly when the gravity is dominated by another object, as in a relatively low-mass circumstellar disk, or when the gas is extremely hot and expanding, as in an expanding

Table 5.1: Physical characteristics of some typical regimes in which hydrodynamics plays a role. Estimates of the relevant fundamental parameters are given, including the typical temperature (T), number density (n), scale height (h), and the state of the gas. For the circumstellar disks we adopt the mid-plane densities and radial scales.

Typical environment	T [K]	n [cm^{-3}]	h [pc]	state of hydrogen
cold neutral medium	50–100	20–50	100–300	atomic
warm neutral medium	10^4	0.2–0.5	300–400	atomic
warm ionized medium	8000	0.2–0.5	10^3	ionized
H II regions	8000	0.2–0.5	70	ionized
stellar coronæ	10^6–10^7	10^{-4}–10^{-2}	10^{-7}	ionized
stellar interiors	10^4–10^7	10^{29}	10^{-9}	ionized
molecular clouds	10–20	10^2–10^6	80	molecular
circumstellar disks	3000–7000	10^{11}	10^{-5}–10^{-3}	molecular

supernova shell. In other circumstances, of course, such as star formation, self-gravity is critical.

5.1.3.2 Sound-crossing Time Scale

The sound-crossing time scale of a region of gas with length scale R is

$$t_{\rm sound} = \frac{R}{c_s}, \tag{5.19}$$

where the sound speed c_s is given by Equation 5.17. For a star in virial equilibrium (see Section 4.1.1.3), $t_{\rm sound} \approx t_{\rm ff}$.

5.1.3.3 Viscous Time Scale

We can estimate the shortest unstable wavelength of a flow if we know the kinematic viscosity ν. The time scale on which a disturbance of wavenumber k is damped is

$$t_{\rm visc} \sim \frac{1}{k^2\nu}. \tag{5.20}$$

This can be seen by estimating $|\mathbf{v}|/|\mathbf{f}_{\rm visc}|$ in Equations 5.8 and 5.10, or by writing $L = 1/k$, $v = L/t_{\rm visc}$ and requiring that the Reynolds number $Re \sim 1$. Stabilization occurs if this time scale is shorter than the growth time scale of an instability. Comparing with the instability growth times given by Equations 5.25 and 5.21, we see that viscosity results in an upper limit on the unstable wavenumber k.

5.1.4 Hydrodynamical Instabilities

Hydrodynamical flows are subject to a number of instabilities, the best known being the Kelvin–Helmholtz, Rayleigh–Taylor, and Jeans instabilities. The first two operate in (for example) boundary layers between fluids; the second and third occur in external or

self-gravitating potentials, respectively. The growth of these instabilities is eventually arrested by viscosity, the effect of which increases as spatial scales become small and velocities become high.

5.1.4.1 Rayleigh–Taylor Instability

The Rayleigh–Taylor instability arises when a dense fluid (density ρ_1) lies above a less dense fluid (density ρ_2) in a gravitational field (which defines "above"). The growth time scale for a mode with wavenumber k in a uniform field g is

$$t_{\mathrm{RT}} = \left(\frac{\rho_1 - \rho_2}{\rho_1 + \rho_2}\, gk \right)^{-1/2} \tag{5.21}$$

The instability is driven by the fractional difference between the two densities. The time scale $(gk)^{-1/2}$ in Equation 5.21 is the time needed for a point on the boundary to accelerate (with acceleration g) through a distance comparable to the wavelength $(1/k)$. Shorter wavelengths grow more quickly, and they become nonlinear sooner, but the long wavelengths are ultimately crucial for mixing the fluids.

5.1.4.2 Shear and Kelvin–Helmholtz Instabilities

The Kelvin–Helmholtz instability arises in the interface layer between fluids with different velocities. Consider a fluid of constant mean density ρ, which is subject to shear due to the presence of another fluid with which it shares a boundary at $z = 0$. Near the boundary, the relative flow velocity is

$$\mathbf{v} = v(z)\,\mathbf{e}_x = Sz\,\mathbf{e}_x, \tag{5.22}$$

where $S = dv/dz|_0$ is the local shearing rate of the fluid. Equation 5.22 can be viewed as a first-order Taylor expansion of the actual shear flow near $z = 0$.

Now consider two parcels of fluid, one at $z = 0$ and the other at $z = \varepsilon$, where ε is small. The kinetic energy (per unit volume) in the two parcels is

$$E = \tfrac{1}{2}\rho v(0)^2 + \tfrac{1}{2}\rho v(\epsilon)^2 = \tfrac{1}{2}\rho S^2\epsilon^2 \tag{5.23}$$

Moving in the frame of reference of the fluid at $z = 0$, the momentum of the lower parcel is zero, while a parcel in the upper layer has a momentum (per unit volume) of $\rho v = \rho S\varepsilon$. Now imagine switching the two parcels, so the top one becomes the bottom and vice versa, mixing and equalizing their momenta. If total momentum is conserved in the process, then the momentum of each of the two parcels becomes $\frac{1}{2}\rho S\epsilon$ and they both move with the average speed $\frac{1}{2}S\epsilon$. The kinetic energy of the switched parcels is then

$$E_s = \tfrac{1}{4}\rho S^2\epsilon^2. \tag{5.24}$$

Thus, the new flow consisting of the two switched parcels has lower energy than the original: the difference is $\Delta E = E - E_{\mathrm{s}} = 1/4\rho S^2\varepsilon^2 > 0$. This energy is available for amplifying initial perturbations and driving instabilities. This is the fundamental

Figure 5.2: Kelvin-Helmholtz instability observed in the atmosphere of Jupiter, as photographed by the *Juno* mission on 2017 December 16 at 12:43 EST at a distance of 13,345 km. Image by NASA/JPL-Caltech/SwRI/MSSS/Gerald Eichstädt/Seán Doran with permission.

reason why shear instabilities are so common: the shear itself is a reservoir of kinetic energy that can be used to drive the instability. Figure 5.2 presents an actual example of observed Kelvin–Helmholtz instability.

If gravity can be neglected, then the shear flow is demonstrably unstable on all length scales. For a shear flow involving fluids of densities ρ_1 and ρ_2 and velocities v_1 and v_2, the growth time scale for wavenumber k is (Bradt, 2014)

$$t_{\rm KH} = \frac{\rho_1 + \rho_2}{\sqrt{\rho_1 \rho_2}} \frac{1}{k \Delta v}. \tag{5.25}$$

Here, $\Delta v \equiv |v_1 - v_2|$ and $\lambda = 2\pi/k$. When the densities are comparable, $t_{\rm KH} \sim 1/k\Delta v$, so the growth time scale is comparable to the time taken for the fluids to move one wavelength relative to one another. Again, the shortest wavelengths grow most rapidly. In Section 5.4.2, we present an example to test the AMUSE hydrodynamics solvers on a Kelvin–Helmholtz instability.

5.1.4.3 Jeans Instability

The Rayleigh–Taylor and Kelvin–Helmholtz instabilities appear along the boundaries between fluids. The Jeans instability can arise in a single fluid, so long as the fluid is sufficiently dense.

If the freefall time scale $t_{\rm ff}$ (Equation 5.18) in the fluid exceeds the sound-crossing time scale $t_{\rm sound}$ (Equation 5.19), pressure forces can overcome gravity and the fluid is stable against small perturbations. However, if $t_{\rm sound} > t_{\rm ff}$, the fluid is unstable against gravitational collapse. The resulting length scale is called the Jeans length and can be estimated by conflating Equations 5.18 and 5.19, which results in

$$\lambda_J = \frac{2\pi}{k_J} \sim \frac{c_s}{\sqrt{G\rho}}. \tag{5.26}$$

A more careful analysis of the collapse, based on a study of the stability of one-

dimensional sound waves (see Binney & Tremaine, 1987), gives

$$\lambda_J = c_s \sqrt{\frac{\pi}{G\rho}}. \tag{5.27}$$

All scales larger than λ_J are unstable against gravitational collapse, whereas smaller scales are stable. An associated mass, called the Jeans mass, is conventionally taken to be the mass contained within a sphere with diameter equal to the Jeans length:

$$M_J = \tfrac{4}{3}\pi\rho(\tfrac{1}{2}\lambda_J)^3 = \frac{\pi^{5/2}}{6}\frac{c_s^3}{\sqrt{G^3\rho}}. \tag{5.28}$$

For a typical dense molecular cloud with temperature 10 K and density 10^3 hydrogen molecules cm^{-3}, the sound speed is $c_s = 260$ m s^{-1}, so $\lambda_J = 1$ pc and $M_J = 26$ M$_\odot$.

5.1.5 Physics of the Integrator

5.1.5.1 Underlying Assumptions

Several assumptions must be made about the state of the fluid, depending on accuracy requirements and our knowledge of the underlying physics. We mention here just some of the most important assumptions made in astronomical fluid dynamical simulations.

1. **Fluid approximation**. As mentioned previously, to derive the fluid equations (Equations 5.9 and 5.8) solved by all of the hydrodynamics modules in AMUSE, it is necessary to assume that we can find a length scale that is small compared to any scale of interest, but large enough to contain sufficient particles for density, temperature, etc. to be well-defined. For example, imagine a giant molecular cloud of mass 10^4M$_\odot$ and radius 3 pc. The mean density is $\rho \approx 6.0 \times 10^{-21}$ g cm^{-3}. If the composition is pure molecular hydrogen, $m_{\rm H2} = 3.35\times 10^{-24}$ g, then a cubic meter contains 1.8×10^{12} molecules, more than enough to accurately define local thermodynamic quantities. At the same time, 1 m is far smaller than any scale of interest for studies of interstellar gas flow or star formation.

   Note that, even if the gas can be represented well as a continuous fluid, a similar problem recurs in simulation methods that treat the fluid as an ensemble of particles (see Section 5.2.1). It is important to ensure that the number of particles is large enough that the Poisson noise introduced by their finite number is small, so that the particles accurately represent the underlying distribution function. In principle, we could use a huge number of small particles to represent a cloud, but this would be numerically infeasible. However, if we make the particles too massive, we run the risk of averaging too much, thereby artificially smoothing the solution. One of the most important ingredients of hydrodynamical simulations is choosing the proper resolution to address the problem of interest.

2. **Classical fluid**. All of the AMUSE hydrodynamics modules assume that the fluid at any point can be described by its velocity, temperature, and composition, and that quantum mechanical and relativistic effects can be ignored.

3. **Local thermodynamic equilibrium**. We assume that every fluid element is in local thermodynamic equilibrium—that is, all particles in the element can interact locally and reach equilibrium with each other (see Section 4.1.2). As a consequence, there is only one degree of freedom: the temperature. This assumption is violated in shocks, in which the temperature is formally discontinuous (as are the related thermodynamic quantities). All AMUSE modules contain prescriptions to handle such sharp features, either by incorporating explicit shock-capturing schemes or by spreading shocks out over a few resolution elements.

All of these assumptions are built into the hydrodynamics modules in AMUSE. It is probably worth your time to check that they actually apply before launching into a large-scale AMUSE simulation.

5.2 Hydrodynamics in AMUSE

The hydrodynamics modules in AMUSE can be divided into two broad categories, each having its own fan base and sets of advantages and disadvantages. Experts will argue (interminably) the merits of each. Here, we try to take an agnostic view.

5.2.1 Types of Hydrodynamics Code

1. **Lagrangian methods: SPH.** In the Lagrangian description of a fluid flow, we follow the motion of a fluid element as described by Equation 5.8. One strength of Lagrangian methods is that they preserve Galilean invariance, i.e., the laws of motion are the same in all inertial frames.

   In most numerical implementations of this approach, the fluid is represented by particles. Each particle represents a finite mass of the fluid; density and other thermodynamic quantities are computed by kernel estimation from nearby particles. Because each particle may be thought of as being smoothed out over some region, these methods are generically referred to as Smoothed Particle Hydrodynamics, or SPH. The amount of smoothing determines the accuracy and resolution of the method, so it must be chosen carefully. Larger smoothing means that many particles contribute to the density at any given point, reducing statistical fluctuations but also decreasing resolution. Interactions among particles include gravity, implemented much as in an N-body code (see Section 3.2), as well as pressure forces due to a particle's neighbors.

   SPH codes are relatively straightforward to implement. The use of particles naturally makes them well-suited to astrophysical phenomena involving complex morphologies and/or poorly defined boundaries. They tend to be very stable, although the results can be adversely affected by noise. One significant problem is the lack of a natural prescription for dissipation and diffusion, which are generally controlled by free parameters. Because they put the particles where the mass is, SPH codes generally provide a poor description of low-density regions in a flow.

The smoothing formulation leads to difficulties in resolving environments with high density or temperature contrasts. In particular, shocks may be difficult to resolve, although there are many variations on the basic SPH method in which such complex phenomena can be handled effectively (e.g., Monaghan & Gingold, 1983). Variations and extensions to the fundamental SPH description allow for the study of complex processes, including, for example, magnetohydrodynamics, turbulence, and various relativistic effects (Iwasaki & Inutsuka, 2011; Price *et al.*, 2018).

2. **Eulerian methods: Grid-based and finite volume schemes.** Instead of following the motion of individual fluid elements, the equations of motion (as expressed in Equation 5.9) can instead be solved on a grid. Most high-performance Eulerian grid-based codes rely on so-called Godunov (1959) schemes, which solve the underlying partial differential equations using a conservative finite-volume method at each inter-cell boundary. The original Godunov method was accurate only to first order in space and time, but higher-order methods have since been developed (van Leer, 1979; Colella & Woodward, 1984), and are now the norm.

   In the simplest implementation of an Eulerian scheme, a fixed three-dimensional grid is imposed on the system and the fluid dynamical equations are solved on the grid vertices. The geometry can vary from run to run, but it remains fixed at run time. Common grid choices include Cartesian, spherical, cylindrical, or toroidal coordinates. Ideally, the problem under study has some symmetry, allowing it to map naturally onto some grid topology, and this symmetry should not change greatly during the simulation.

   In contrast to Lagrangian methods, grid-based methods are not Galilean-invariant, because the algorithm to calculate the fluxes has preferred directions. This may pose a problem when studying Kelvin–Helmholtz or other shear instabilities. For such studies, grid codes tend to give different solutions for the same relative velocity between the two layers when the entire system is moving (see Section 5.1.4.2). This can be solved (at considerable expense in computer resources) by reducing the truncation error in the flux calculation algorithm and increasing resolution until the hydrodynamical instabilities are resolved (Godunov & Kulikov, 2014). This is rarely done because of its computational cost. As a consequence, Eulerian fixed-grid codes are not really suitable for studying discontinuities and multi-phase media. These problems seem to be solved with moving-mesh methods, but further study is required.

   The fixed-grid topology allows grid-based hydrodynamics codes to perform excellently on large parallel architectures. In addition, it is relatively straightforward to extend the fluid dynamical equations to include radiative transfer, magnetic fields, or relativistic effects. Despite the resolution problems introduced by adopting a fixed grid, these methods are still quite popular.

   Fixed-grid methods can be improved by allowing the grid to be refined, depending on local resolution requirements. The resulting adaptive mesh refinement

(AMR) codes are suited particularly well to addressing the development of discontinuities, such as shocks and gravitational collapse, whose appearance in a simulation is hard to predict in advance (MacNeice *et al.*, 2000; Adams, 2014). The main disadvantages of grid-based methods are their relatively large memory requirements and the complexity of coding and optimization. In addition, the grid topology may be reflected in the solution, and boundary conditions may produce unwanted artifacts in the flow.

3. **Unstructured mesh schemes.** Particle- and grid-based methods are slowly being superseded by moving-mesh methods that solve the fluid dynamical equations on unstructured grids that evolve with the fluid (Robertson *et al.*, 2010; Springel, 2010). In many ways, these methods combine the best properties of Lagrangian and Eulerian methods. They are efficiently parallelizable by domain decomposition, and are suitable for solving systems with steep temperature and/or density gradients. A major advantage of these methods is that the solution is independent of grid structure, removing one of the major disadvantages of AMR methods. The largest disadvantage of these codes is their current unavailability in the public domain. This is disappointing because it prevents us from comparing them within the AMUSE framework to other similar algorithms. However, as these codes become publicly available, they will be incorporated into AMUSE.

Table 5.2 lists the most important hydrodynamics codes currently available in AMUSE. Here, we draw a distinction between particle-based (TreeSPH), fixed grid (grid), adaptive mesh refinement (AMR) codes and moving mesh (MM). Each of the codes is parallelized, because hydrodynamics is expensive in terms of computing time, and high resolution requires significant computer memory. The interface to Arepo is relatively new, and not used much yet, whereas the Ramses, though present, is hardly used in production. AMUSE further contains interfaces for additional codes, such as FLASH (Fryxell *et al.*, 2000), but these codes require registration or permission to use, so they are not routinely distributed with AMUSE. To be fair, the best codes to use are Gadged, Fi, or Phantom Price *et al.* (2018). Ideally, we would like to interface with BonsaiSPH Bedorf & Portegies Zwart (2020), but twe still have not resolved how to address multiple GPU's across a distributed memory machine in the AMUSE interface.

We can run a hydrodynamics code in AMUSE in much the same way as an N-body or stellar evolution simulation would be run. Because the two main algorithms (grid and SPH) used to solve the fluid equations differ so greatly, the user must decide in advance which one to use. Ideally, we would like to be able to switch back and forth between the two approaches as needed. Regrettably, that is not currently possible in AMUSE because we do not know how to uniquely convert a particle distribution to an adaptive grid, and generating a low-noise particle distribution from an adaptive mesh is challenging. The SPH interface is developed better and used most widely, so most of the discussion that follows will assume that choice. In Section 5.2.3, we will present some simple examples using grid codes.

Table 5.2: Hydrodynamics codes distributed within AMUSE. Each of these codes compiles by default with an adiabatic equation of state. An isothermal equation of state is implemented, but requires the code to be recompiled, except for Fi where it can be set as a code parameter. The global code characteristics are indicated in the third columns, where TreeSPH indicatess the SPH codes for which the interacting forces are calculated using a tree-code, grid and AMR refer to cartesian or refined grid methods, and MM to moving mesh. The last column gives the available boundary conditions accessible directly from AMUSE: "cont," "inflow," "interface," "outflow," "periodic," and "reflective" (see §5.2.3). For Athena3D see also https://trac.princeton.edu/Athena/wiki/AthenaEqns. References: 1: Springel (2000, 2005); 2: Hernquist & Katz (1989); Pelupessy *et al.* (2004); Gerritsen & Icke (1997); 3: Price *et al.* (2018); 4: Mellema *et al.* (1991); 5: Gardiner & Stone (2005, 2008); Stone *et al.* (2008, 2010); 6: Meliani *et al.* (2007); Keppens *et al.* (2012); van der Holst *et al.* (2012); Porth *et al.* (2014); 7: Teyssier (2002, 2010); 8: Fryxell *et al.* (2000); 9: Pakmor *et al.* (2016).

name	ref.	type of code	language	parallel	boundaries
Gadget-2	1	TreeSPH	C	yes	ciiopr
Fi	2	TreeSPH	F95	yes	ciiopr
Phantom	3	TreeSPH	F95	yes	ciiopr
Capreole	4	grid	F95	yes	--iopr
Athena3D	5	AMR	C	yes	--iopr
mpiAMRVAC	6	AMR	F95	yes	--iopr
ramses	7	AMR	F90	yes	--i-pr
FLASH	8	AMR	F90	yes	--i-pr
Arepo	9	MM	C	yes	ciiopr

5.2.2 Smoothed Particle Hydrodynamics

Listing 5.1 presents a simple function for setting up and solving a hydrodynamics problem in AMUSE. This script doesn't do much, but in the next few paragraphs we will demonstrate that, as with gravity and stellar evolution, it is possible to do some interesting experiments with just hydrodynamics.

The function first sets up a virialized gas Plummer (1911) sphere of specified mass and radius, realized as N particles:

```
converter = nbody_system.nbody_to_si(Mtot , Rvir)
gas = new_plummer_gas_model(N, convert_nbody=converter)
```

Note that, because the function new_plummer_gas_model is written in N-body units, we need a converter (see Section 3.3.5) to define the physical units used. Most hydrodynamics solvers work in CGS units, unlike the N-body and stellar units used in previous chapters. However, as discussed in Section 3.3.5, AMUSE keeps track of such details, so we don't have to do it explicitly.

We initialize the SPH code Gadget2 as our hydrodynamics code and commit the gas particles, much as we did in Chapter 3:

```
hydro = Gadget2(converter)
hydro.gas_particles.add_particles(gas)
```

```
import sys
import argparse
from amuse.units import nbody_system, units
from amuse.ic.gasplummer import new_plummer_gas_model
from amuse.community.gadget2 import Gadget2
from amuse.io import write_set_to_file


def hydro_minimal(
    number_of_particles=100,
    total_mass=1 | units.MSun,
    virial_radius=1 | units.RSun,
    time_end=6 | units.hour,
):
    converter = nbody_system.nbody_to_si(total_mass, virial_radius)
    gas = new_plummer_gas_model(number_of_particles, convert_nbody=converter)

    hydro = Gadget2(converter)
    hydro.gas_particles.add_particles(gas)
    energy_total_init = (
        hydro.kinetic_energy + hydro.potential_energy + hydro.thermal_energy
    )
    hydro.evolve_model(time_end)
    write_set_to_file(hydro.particles, "hydro.amuse")

    energy_kinetic = hydro.kinetic_energy
    energy_potential = hydro.potential_energy
    energy_thermal = hydro.thermal_energy
    energy_total = energy_kinetic + energy_potential + energy_thermal
    virial_ratio = (energy_kinetic + energy_thermal) / energy_potential
    energy_difference = (energy_total_init - energy_total) / energy_total
    com = hydro.gas_particles.center_of_mass()
    print(
        f"T= {hydro.get_time()} M= {hydro.gas_particles.mass.sum()} "
        f"E= {energy_total} Q= {virial_ratio} "
        f"dE= {energy_difference} CoM= {com.in_(units.RSun)}"
    )

    hydro.stop()
```

Listing 5.1: Minimal routine for evolving a Plummer gas cloud of mass Mtot and virial radius Rvir to some end time t_end.

We are now ready to evolve the hydrodynamics solver to some end time t_end:

```
    hydro.evolve_model(t_end)
```

Eventually, we could dump the data to a file with the name filename

```
    write_set_to_file(hydro.particles, filename, "hdf5")
```

We might improve this hydrodynamical calculation by running it in small time increments and printing diagnostics after each, just as we did for the gravity solver (Listing 3.2). Hydrodynamics simulations are often used in this way to make animations. Many of these can be quite instructive, but beware of convincing-looking animations that actually present erroneous results. Validating results is an important but difficult aspect of these simulations.

5.2.3 Grid-based Methods

Setting up a grid code is not much different from setting up an SPH code, except that defining the boundary conditions and generating the initial conditions are somewhat more elaborate. The fundamental calling sequence is similar to the SPH version. First, we declare the converter and start the code:

```
    converter = nbody_system.nbody_to_si(1|units.MSun, 1|units.RSun)
    hydro_code = Athena(converter, number_of_workers=16)
```

Grid-based hydro codes parallelize quite well, and the calculations are often computationally intensive. We therefore recommend always running them in parallel. In this example, we run the code using 16 cores.

We can tune the working of the hydrodynamics solver by specifying some of the fundamental parameters, such as the adiabatic index, for which we adopt $\gamma = 5/3$ for the ideal monatomic gas. Another important parameter is the Courant–Friedrichs–Lewy condition, which is used to control the convergence while solving the hyperbolic partial differential equations in the finite-difference scheme (see Courant *et al.*, 1928), for which we choose a value of 0.3.

```
    hydro_code.parameters.gamma = 5/3.0
    hydro_code.parameters.courant_number=0.3
```

More detail on the important parameters used by this AMUSE module (and others) are presented in Appendix B.

We further specify the grid resolution in each of the three Cartesian coordinates, and the length of the computational domain

```
    hydro_code.parameters.nx = resolution
    hydro_code.parameters.ny = resolution
    hydro_code.parameters.nz = resolution

    hydro_code.parameters.length_x = grid_size
    hydro_code.parameters.length_y = grid_size
    hydro_code.parameters.length_z = grid_size
```

We subsequently specify the boundary conditions, which are attributes of the running hydro code and can be set as parameters. For example, to apply open outflow boundary conditions along the Cartesian coordinates,

```
    hydro_code.x_boundary_conditions = ("outflow", "outflow")
```

```
    hydro_code.y_boundary_conditions = ("outflow", "outflow")
    hydro_code.z_boundary_conditions = ("outflow", "outflow")
```

Alternatives to the "outflow" boundary condition are "cont", "inflow", "interface", "periodic", and "reflective", although not all of these may be defined for each of the available hydrodynamics solvers in AMUSE. The keywords refer to the different ways in which grid boundaries are handled, and they can be set separately for each of the three Cartesian coordinates. In this case, we opt for "outflow" boundary conditions. Table 5.2 presents a brief overview of the boundary conditions supported by each code; Appendix B gives more detail on the other boundary conditions.

After the hydrodynamics solver has been initialized, we construct the grid, which is created with a specified geometry and filled with distributions of density, momentum, and energy. Initializing a three-dimensional grid can be realized with the built-in function from the datamodel package

```
    from amuse.datamodel import new_regular_grid
    grid = new_regular_grid((100, 100, 100), [10, 10, 10] | units.RSun)
```

which generates a three-dimensional grid of 10^6 ($= 100 \times 100 \times 100$) cells, with total lengths of 10 $R_\odot$ in each of the three Cartesian coordinates (the computational domain runs from zero to 10 $R_\odot$ in each). Densities (grid.rho), energies (grid.energy), and momenta (grid.momentum) can then be set in the individual grid cells. Global grid properties can be added to each of the grid cells in much the same way as we add properties to a particle set, e.g.,

```
    grid.rho = 1.e-10 * units.MSun/units.RSun**3
```

Other parameters can be set similarly.

Filling the grid with useful initial conditions is complicated, and only a limited number of built-in functions exist to assist you in the task. One of these functions is fill_grid_with_spherical_cloud, which adds a homogeneous spherical density distribution to grid. Thus, to add a homogeneous spherical cloud with density equal to the mean solar density,

```
    from amuse.ext import cloud
    cloud.fill_grid_with_spherical_cloud(grid,
        center = [5.0, 5.0, 5.0] | units.RSun,
        radius = 1 | units.RSun,
        rho = 1.0 | units.MSun / units.RSun**3,
        rhovx = 0.0 | units.kg / (units.s * units.m**2),
        rhovy = 0.0 | units.kg / (units.s * units.m**2),
        rhovz = 0.0 | units.kg / (units.s * units.m**2),
        energy = 1.e+40 | units.erg / units.RSun**3
    )
```

Sometimes, one needs to add additional attributes, such as chemical composition or magnetic fields, to the grid. This can be accomplished by adding a vector attribute to each of the grid cells. The function add_global_vector_attribute adds a vector quantity

to a local particle set or grid. For example, to add a three-dimensional vector called composition with components cx, cy, and cz describing the composition in hydrogen, helium, and metals, we could write:

```
grid.add_global_vector_attribute("composition", ["cx","cy","cz"])
grid.composition = (0.7, 0.29, 0.01)
```

The grid itself is eventually copied to the running hydro code using a channel.

```
    channel = grid.new_channel_to(hydro_code.grid)
    channel.copy()
```

Running the hydrodynamics solver to some end time t_end is no more elaborate than running any other code:

```
    channel = hydro_code.grid.new_channel_to(grid)
    dt = t_end/10.
    while hydro_code.model_time<t_end:
        hydro_code.evolve_model(hydro_code.model_time + dt)
        channel.copy()
        write_set_to_file(grid, filename, ``amuse'')
    hydro_code.stop()
```

A fully working example of this procedure can be found in ${AMUSE_DIR}/examples/textbook/supernov

5.2.4 Managing Shocks and Discontinuities

Efficient and accurate treatment of shocks is a key component of all modern hydrodynamics simulation codes (see Section 5.1.2.2). Simply put, finite-difference schemes don't handle discontinuities well, and when the discontinuity is too steep (spread over too many resolution elements), a shock wave produces unphysical density oscillations behind the advancing front.

Simple finite-difference codes (like the SPH codes in AMUSE), rely on artificial viscosity to prevent the steep gradients and unphysical oscillations associated with shocks. There are many variations, but the basic idea goes back to Von Neumann & Richtmyer (1950), who replaced the pressure P in the Euler (Equation 5.8) and energy (Equation 5.13) equations by $P + q$, where

$$q = \begin{cases} \alpha\rho h^2(\nabla\cdot\mathbf{v})^2, & \nabla\cdot\mathbf{v} < 0 \\ 0, & \nabla\cdot\mathbf{v} > 0. \end{cases} \tag{5.29}$$

Here, h is the particle smoothing length in an SPH calculation or the local grid spacing in a grid code. The method artificially increases the local pressure in regions of strongly convergent flows to limit large density gradients. The parameter $\alpha \sim 1$ is tuned to spread shocks out over a few (4–5) resolution elements, maintaining stability but losing precision.

More sophisticated *shock-capturing schemes* (incorporated into the grid-based AMUSE modules) leverage our knowledge of basic 1D shock physics to model the flux of mass,

momentum, and energy from one grid cell to the next (see Section 5.4.1). These schemes provide much better resolution of fine structure, albeit at the expense of significantly increased code complexity. Comparable schemes have been introduced into the SPH formalism (Inutsuka, 2002; Murante *et al.*, 2011), with only limited success.

5.2.5 Initializing a Grid from a Particle Distribution

Sometimes it is easier to generate a particle distribution with certain properties and then convert it to a grid. For example, we might start by generating an initial Plummer sphere (see Chapter 3) particle distribution, as described in Section 5.2.2. We can then convert this distribution to a grid by using the function convert_SPH_to_grid:

```
from amuse.ext.sph_to_grid import convert_SPH_to_grid
sph_code = setup_sph_code(Fi, N, Rvir, rho, u)
grid = convert_SPH_to_grid(sph_code, (100, 100, 100))
```

Here, we adopt the SPH code Fi with N particles distributed in a sphere with radius Rvir, density rho, and internal energy u. The SPH code is then used to fill the grid cells from the particle distribution. Bear in mind that some AMUSE data structures naturally map onto grids. For example, the one-dimensional internal structure of a star in a Henyey stellar evolution code (see Section 5.2) is easily translated into a three-dimensional grid for use in a hydro code.

Figure 5.3 presents the result of a grid-based hydrodynamical simulation of an exploding core-halo "star". Initially, the star had a central 1.4 $M_\odot$ core, of radius 0.1 $R_\odot$, surrounded by a homogeneous spherical halo of mass 3 $M_\odot$ and radius 1 $R_\odot$. We then uniformly injected 10^{51} erg of energy into the core, causing it to explode. The adopted grid resolution was fixed at 256^3. The symmetric blobs visible at the corners are artifacts of this Cartesian grid.

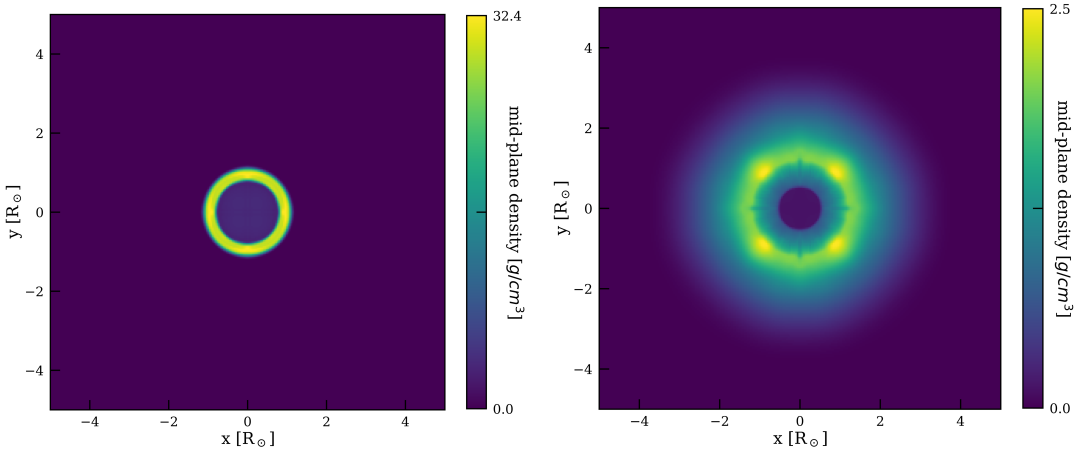

Figure 5.3: Mid-plane density distribution of a $4\,M_\odot$ core–halo "star" whose $1.4\,M_\odot$ core explodes at time 0 with an energy of 10^{51} erg. Resolution is $256 \times 256 \times 256$ grid cells. The two snapshots are taken at 100 s (left) and 300 s (right) after the explosion. The calculation was performed using `Athena`. The figure was made using the script `${AMUSE_DIR}/examples/textbook/supernova_grid.py`; it takes a couple of hours to run on a laptop.

Figure 5.4 presents the result of an SPH calculation using exactly the same initial conditions as for the grid calculation just described Figure 5.3. The presentation in Figure 5.4 is slightly different from Figure 5.3, because generating an image from a particle distribution entails interpolating the particle densities onto the image plane, rather than simply taking slices from a grid, but the two images show interesting similarities and discrepancies. In this case, we first performed the calculation using one script, then wrote the data to a file, and later made the plot using another script.

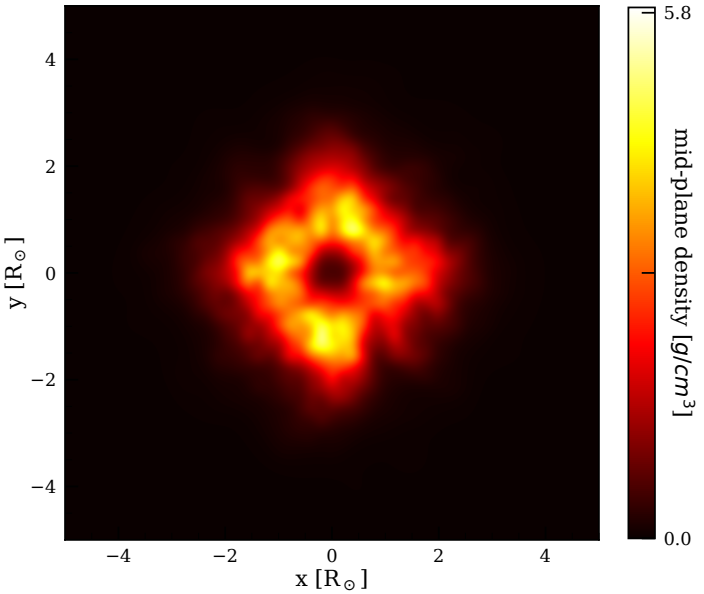

Figure 5.4: Projected density distribution of the same core-halo exploding object as in Figure 5.3. This run was performed using the SPH code Gadget2 with 10^5 equal-mass particles. The image is taken 300 s after the injection of 10^{51} erg of energy into the core. The data to make the figure were generated using the script `${AMUSE_DIR}/examples/textbook/supernova_sph.py`, requiring a few minutes to run; the figure itself was made using `${AMUSE_DIR}/examples/textbook/plot_supernova_sph.py`.

To make the figure, we first construct a regular grid of x- and y-coordinates:

```
def setup_grid(N):
    x,y = numpy.indices((N+1,N+1))
    x = L*(x.flatten()-N/2.)/N
    y = L*(y.flatten()-N/2.)/N
    z = 0.*x
    vx = 0.*x
    vy = 0.*x
    vz = 0.*x
    x = units.parsec(x)
    y = units.parsec(y)
    z = units.parsec(z)
    vx = units.kms(vx)
    vy = units.kms(vy)
    vz = units.kms(vz)
    return x, y, z, vx, vy, vz
```

and fill this using interpolated values from a running SPH code:

```
def make_hydromap_and_show_picture(sph_particles, N=100, L=10):
    hydro = Gadget2(converter)
    hydro.gas_particles.add_particles(sph_particles)
    x, y, z, vx, vy, vz = setup_grid(N)
    rho,rhovx,rhovy,rhovz,rhoe = hydro.get_hydro_state_at_point(x,y,z,vx,vy,vz)
    rho = rhoe.reshape((N+1,N+1))
    pyplot.imshow(numpy.log10(rho.value_in(units.erg/units.RSun**3)))
    pyplot.show()
```

When generating initial conditions for a grid-based hydrodynamics code, we would ideally like to be able to run the same initial conditions using an SPH code. The translation from a grid to a particle representation is not unique, but the routine convert_grid_to_SPH performs such a conversion. The resulting particle distribution still reflects the structure of the grid—even in an SPH code, the structure of the grid is not erased until the system has evolved for a time comparable to the freefall time scale (Equation 5.1.3.1).

5.2.6 Using a Hydro Code to Simulate a Stellar Merger

In Section 7.4.4, we demonstrated how to use a separate code to resolve a stellar collision using a parameterization of the collision process. Instead of a parameterized code, we could also use a hydrodynamics solver for the collision. Here we do just that, using an SPH code to resolve the merger.

5.2.6.1 Generating Stellar Structure in a Particle Set

To start the merger simulation, we first need the structure of two stars. We saw in Chapter 4 how this can be realized. Listing 5.2 demonstrates the process again.

```
def return_evolved_star_hydro(mass, time, number_of_sph_particles):
    star = Particle(mass=mass)
    stellar = Evtwin()
    star = stellar.particles.add_particle(star)
    stellar.evolve_model(time)
    number_of_sph_particles = (
        number_of_sph_particles * int(mass.value_in(units.MSun))
    )
    star_in_sph = convert_stellar_model_to_sph(
        star, number_of_sph_particles
    ).gas_particles
    stellar.stop()
    return star_in_sph
```

Listing 5.2: Using a Henyey code to construct an SPH realization of a single star of mass mass evolved to time time. The complete listing is in `${AMUSE_DIR}/examples/textbook/merge_two_stars_sph.py`

In addition to evolving the star to the desired time, this code also converts from the stellar structure provided by the Henyey code to an SPH particle representation:

```
    star_in_sph = convert_stellar_model_to_SPH(star, N_sph).gas_particles
```

The method convert_stellar_model_to_SPH returns a Python class having gas_particles as an attribute. We run the same procedure for the primary and the secondary stars, then add the two SPH realizations together after offsetting the secondary with respect to the primary and giving them some relative velocity. This is illustrated in Listing 5.3. The script looks similar to Listing 4.7, except that here we convert the evolved stars into N smoothed gas particles.

```
def merge_two_stars_sph(
    mass_primary, mass_secondary, time_collision, number_of_sph_particles
):
    primary_in_sph = return_evolved_star_hydro(
        mass_primary, time_collision, number_of_sph_particles
    )
    primary_in_sph = relax_sph_realization(primary_in_sph)
    secondary_in_sph = return_evolved_star_hydro(
        mass_secondary, time_collision, number_of_sph_particles
    )
    secondary_in_sph = relax_sph_realization(secondary_in_sph)
    radius = primary_in_sph.x.max() + secondary_in_sph.x.max()
    total_mass = primary_in_sph.mass.sum() + secondary_in_sph.mass.sum()
    secondary_in_sph.x += 0.8*radius
    secondary_in_sph.y += 0.6*radius
    secondary_in_sph.vx -= (constants.G*total_mass/radius).sqrt()

    converter = nbody_system.nbody_to_si(mass_primary, 1.0 | units.AU)
    hydro = Gadget2(converter)
    hydro.gas_particles.add_particles(primary_in_sph)
    hydro.gas_particles.add_particles(secondary_in_sph)
    hydro.evolve_model(2.0 | units.hour)
    hydro.gas_particles.new_channel_to(primary_in_sph).copy()
    hydro.gas_particles.new_channel_to(secondary_in_sph).copy()
    hydro.stop()
    return primary_in_sph, secondary_in_sph
```

Listing 5.3: Using an SPH code to merge two stars. For the full listing see ${AMUSE_DIR}/examples/textbook/merge_two_stars_sph.py

The conversion from Henyey to SPH is done on a grid, which introduces artifacts and generally leaves the SPH star slightly out of equilibrium. We address this problem by relaxing the SPH models (in this case, using a separate SPH code), as shown in Listing 5.4. Here, we slowly reduce the velocities of the SPH particles while running the hydrodynamics code with short steps. This is the same process as for creating a "glass" solution for initial conditions as is done in many simulations (see, for example, Section 3.3 on page 40 in White, 1994). In this example, we have opted to relax the star for 2.5 dynamical times in 250 steps per dynamical time scale, which is rather short. It is better to have the star relax for far longer: J. Lombardi (2010, private communication) recommends at least 25 dynamical time scales.

By default, the output of convert_stellar_model_to_SPH represents a star at rest at the origin. Hence, before combining the two SPH realizations of the stars in a hydro

```
63 def relax_sph_realization(sph_star):
64     dynamical_timescale = sph_star.dynamical_timescale()
65     converter = nbody_system.nbody_to_si(dynamical_timescale, 1 | units.RSun)
66     hydro = Gadget2(converter, number_of_workers=2)
67     hydro.gas_particles.add_particles(sph_star)
68 
69     to_framework = hydro.gas_particles.new_channel_to(sph_star)
70 
71     ts_factor = 2.5
72     time_end = ts_factor * sph_star.dynamical_timescale(mass_fraction=0.9)
73     n_steps = ts_factor * 100
74     velocity_damp_factor = 1.0 - (ts_factor*2*np.pi)/n_steps
75     time_step = time_end/float(n_steps)
76     time = 0 | units.day
77     while time < time_end:
78         time += time_step
79         hydro.evolve_model(time)
80         hydro.gas_particles.velocity = (
81             velocity_damp_factor * hydro.gas_particles.velocity
82         )
83     to_framework.copy()
84     hydro.stop()
85     return sph_star
```

Listing 5.4: Relaxing the SPH realization of a star in order to prepare it for a hydrodynamical collision simulation. This code fragment is from `${AMUSE_DIR}/examples/textbook/merge_two_stars_sph.py`. Here we introduced a scaling factor ts_factor

solver, their positions and velocities must be changed to reflect the initial condition of the collision. This can be done by shifting each of the stars to its desired center-of-mass position, with all particles within each star being given the same velocity. In Listing 5.3, we position the secondary star such that it is almost touching the primary star, with a transverse velocity of roughly the escape speed in a direction that results in a grazing collision.

In order to prevent the two stars from moving to the left due to the momentum added to the system, one could move all the SPH particles to their center-of-mass frame after merging.

```
1     all_sph_particles = ParticlesSuperset([primary_in_sph, secondary_in_sph])
2     all_sph_particles.move_to_center()
```

Given that SPH is Galilean-invariant, the shift to the center-of-mass frame is unnecessary, but it makes graphical presentations of the encounter much more convenient.

5.2.6.2 Resolving the Collision with the Hydrodynamics Solver

With the initial conditions established, we instantiate the hydrodynamics solver Gadget2

```
1     hydro = Gadget2()
```

and add the SPH particle sets of the two stars to the community code:

```
    hydro.gas_particles.add_particles(primary_in_sph)
    hydro.gas_particles.add_particles(secondary_in_sph)
```

Here, we add the particle sets for each of the two stars separately, in order to keep track of which star each SPH particle initially belonged to.

In Listing 5.3, we run the hydrodynamics code for a simulation time of only two hours without saving any diagnostic data. For production simulations, this is not desirable. In many cases, it is better to run the specific code in a loop over time, testing conservation of energy and angular momentum, printing some diagnostics, and writing the raw data to a file at the end of each step. Such an loop might look like

```
    t_end = 10 | units.hour
    dt = 1 | units.hour
    while hydro.model_time<t_end:
        hydro.evolve_model(hydro.model_time + dt)
        channel_from_hydro_to_framework.copy()
        print_diagnostics(hydro)
        write_set_to_file(primary_in_sph, 'hydro.amuse', 'amuse')
        write_set_to_file(secondary_in_sph, 'hydro.amuse', 'amuse')
    hydro.stop()
```

Here, print_diagnostics(hydro) might print time, energy, angular momentum, and other useful diagnostic information reflecting the progress of the simulation.

5.2.6.3 Recognizing a Star in the Collision Product

It is not completely trivial to recognize the stellar merger product in the messy hydrodynamical blob that results from a collision. Such collisions may be highly asymmetric, rotate rapidly, and have a giant envelope of barely bound low-density gas. Figure 5.5 demonstrates the asymmetries one can expect from a rather violent collision, in this case between a 80 $M_\odot$ and a 20 $M_\odot$ star, both at an age of about 0.49 Myr. In this calculation, we adopted Gadget2 for performing the hydrodynamical simulation using 10^5 equal-mass SPH particles. The amount of mixing you see in Figure 5.5 is somewhat surprising when compared to earlier results, particularly because such collisions tend to lead to relatively little mixing (see the discussion in Lombardi *et al.*, 1996). The use of more realistic stellar structure models cannot be the reason for the discrepancy (see Sills & Lombardi, 1997), but it is possible that the very high stellar masses adopted, compared to the initial conditions in Lombardi *et al.* (1996), could lead to more mixing.

Here, we adopt a rather simple approach to the problem of identifying the SPH particles making up the merger product. We analyze the results of the SPH simulation using the hop clump finder (see Section 7.3.1). The three control parameters for peak density threshold, saddle point density cutoff, and the outer density threshold are specified to help hop decide which particles belong to a coherent structure. Each of the groups in the list of particle sets (hop.groups) identified by hop belongs to an identified structure. For our stellar merger calculation, we are interested in the largest one.

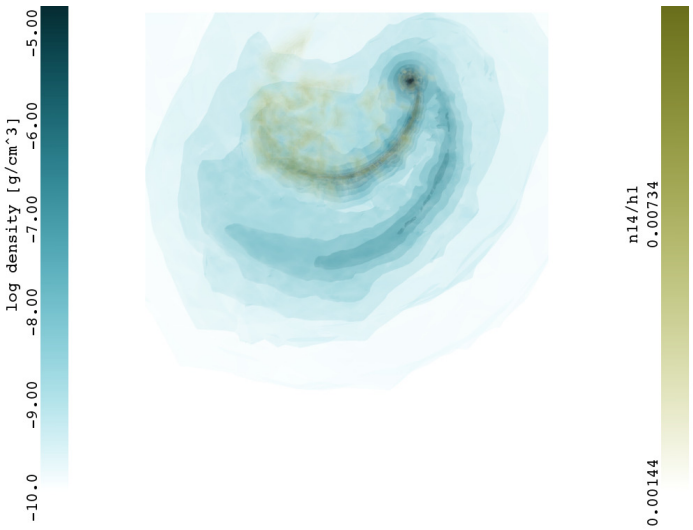

Figure 5.5: Projected density distribution of the gas ejected shortly after the merger between two massive stars. The results of the hydrodynamical simulation are shown 8.8 days after the merger between a $M_{\rm in} = 80\,{\rm M}_\odot$ star and a $m_{\rm in} = 20\,{\rm M}_\odot$ star in an $a_{\rm in} = 77\,{\rm R}_\odot$ orbit with an eccentricity of $e_{\rm in}^{\rm B} = 0.87$. Projected density is shown in blue, while the nitrogen/hydrogen ratio is presented in mustard. The origin is at the center of mass of the original two-body system. The two massive loops of ejected gas (at left) are characteristic of a merger. We do not present the plotting routine for this figure, but instead refer to Portegies Zwart & van den Heuvel (2016) for more details.

```
def find_clumps_with_hop(sph_particles, unit_converter):
    hop = Hop(unit_converter)
    hop.particles.add_particles(sph_particles)
    hop.calculate_densities()

    mean_density = hop.particles.density.mean()
    hop.parameters.peak_density_threshold = mean_density
    hop.parameters.saddle_density_threshold = 0.99*mean_density
    hop.parameters.outer_density_threshold = 0.05*mean_density
    hop.do_hop()

    blob = [x.get_intersecting_subset_in(sph_particles) for x in hop.groups()]
    hop.stop()
    return blob
```

Finding clumps in an SPH simulation is commonly done in post-processing, but it can also be used to monitor the progress of the merger simulation. In the simulation presented in Figure 5.5, we monitored the clumpiness of the particle distribution at run time, and continued until only a single blob of particles remained. After the detection of this final blob, we ran the hydrodynamics for a further 5.5 years to ensure that it had come into dynamical equilibrium. This final blob is considered to be the merger product, and the residual material is identified as ejecta from the merger process. In this simulation, the merger resulted in a single $83\,{\rm M}_\odot$ star.

Once coherent structures have been identified, we often are interested in the original properties of particles that are part of a particular group. For performance and memory optimization, the particles in hop are given only the attributes necessary to identify

individual clumps, whereas the original particle set (sph_particles in the above example) may contain additional information, such as chemical composition. We therefore want to find the intersecting subset of particles with the same key among the original sph_particles and the particles in hop. This is done in the line

```
blob = [x.get_intersecting_subset_in(sph_particles) for x in hop.groups()]
```

Finally, we stop hop and return the appropriate subset of particles.

5.2.6.4 Converting from Hydro back to Stars

In Section 7.4.4, we discussed how a separate code can be used to manage a head-on collision between a 3 $M_\odot$ and a 1 $M_\odot$ star, each evolved to an age of 150 Myr, continuing the evolution of the resulting merged star afterwards. There we adopted the output of the MMAMS (Gaburov *et al.*, 2008; Lombardi & Sawyer Warren, 2014) package to resolve the collision, but now we can adopt the methods discussed in the previous subsection, using SPH rather than MMAMS to determine the structure of the merger product.

As before, we first evolve the stars for 150 Myr, which is half of the primary's main sequence lifetime. Both stars are then converted to SPH realizations and relaxed (see Section 5.2.6.1). The resulting particle sets are combined, with the stellar surfaces just touching and zero relative velocity at infinity. We then use Fi to resolve the hydrodynamics of the encounter. The freefall time scale at this moment is just a few minutes, but to ensure that the gas distribution is in equilibrium, we continue the hydrodynamical calculation for 10^3 times longer. After the SPH calculation, we identify the merged star by using hop to search for the most massive blob (see Section 5.2.6.3), and then we strip away any low-density unbound splash material. The leftover merger product has a mass of 3.73 $M_\odot$, very close to the MMAMS result for the same encounter (see Section 7.4.4).

After stripping away the envelope and relaxing the SPH system again, we put the star back into the Henyey stellar evolution code. This latter step is a rather delicate process, because the stellar evolution code expects a stellar structure that is in thermal equilibrium. The hydrodynamics blob will be in dynamical equilibrium, but thermal equilibrium cannot be guaranteed. The procedure for converting the SPH star into a Henyey model is

```
from amuse.ext.sph_to_star import convert_SPH_to_stellar_model
new_stellar_model = convert_SPH_to_stellar_model(star_in_SPH)
```

This function call can be quite expensive in terms of computer time, and may even fail after an hour or so of crunching. If it succeeds, however, you have a good chance of being able to evolve the star using MESA or EVtwin. After this conversion, we continue the evolution of the collision product to study its track in the H–R diagram.

Figure 5.6 displays the same information for this SPH/hop merger as was presented in Figure 7.5 for the MMAMS merger (see Section 7.4.4). The red, blue, and green tracks show, respectively, the evolution of main sequence stars of masses 3 $M_\odot$, 4 $M_\odot$, and 3.73$M_\odot$ (the value returned by SPH/hop). The yellow dashed line is the

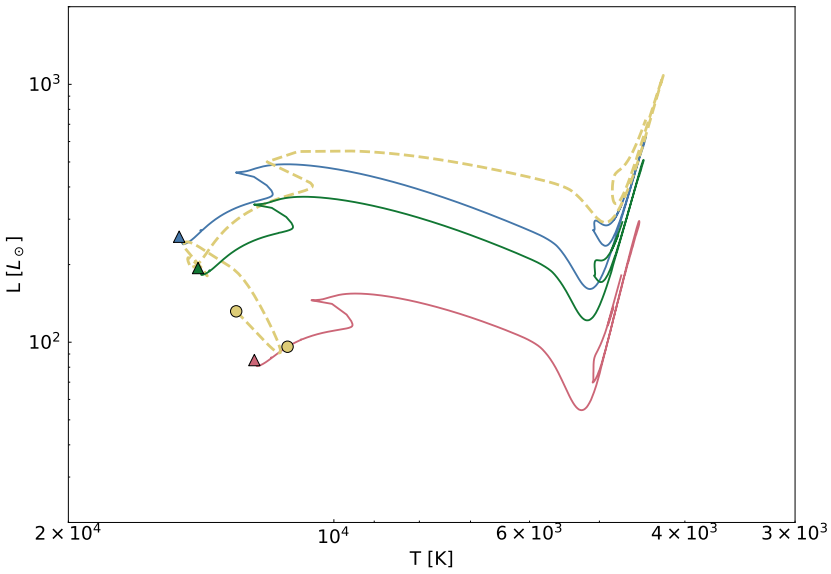

Figure 5.6: Evolutionary tracks for a 3 $M_\odot$ star (red curve), a 4 $M_\odot$ star (blue), and a 3.73 $M_\odot$ star (green) from the main sequence (filled triangles) to the bottom of the AGB. The yellow dashed curve shows the evolutionary track of the merger product of a 3 $M_\odot$ star and a 1 $M_\odot$ star that collide halfway along the primary's main sequence track, with the merger structure determined using SPH and hop. The yellow circles represent the evolutionary states of the primary immediately before the collision and of the merger product immediately afterward. This figure was produced using the script ${AMUSE_DIR}/examples/textbook/merge_two_stars_and_evolve.py, which should run in 10–15 minutes on a laptop.

evolutionary track of the merged object. As with the merger discussed in Section 7.4.4, the evolutionary track of the merger product makes several rapid excursions in temperature and luminosity before settling (after $\sim 10^5$ yr, again comparable to the Kelvin–Helmholtz time scale; see Equation 4.6) onto a new stable track. Its subsequent evolution takes it along a significantly more luminous track than either the MMAMS equivalent or the 4 $M_\odot$ star.

Some cautionary notes are in order. First, recognizing the bound blob after the hydrodynamical simulation depends on several parameters in the hop algorithm (see Section 5.2.6.3), and may require some experimentation. Second, we have used a relatively small number of SPH particles (100 equal-mass particles per solar mass) here to model the merging stars. In part, this is to reduce the cost of the SPH simulation, but it also reflects the fact, already noted, that converting from an SPH to a Henyey representation is a very fragile process that may easily fail to produce a workable Henyey model, and whose outcome depends sensitively on the details of how SPH particles are combined into Henyey shells. In particular, the initial location of the merger product on the H–R diagram can vary dramatically with these details, and probably should not be taken too seriously. As with many advanced topics in AMUSE, these details are still under active development; see the updates page `${AMUSE_DIR}/examples/textbook/updates/AMUSE_updates.pdf` for new material.

It is hard to validate the evolution of the merger product. Understanding the

causes of these deviations from the standard evolutionary track would require a detailed analysis of the stellar structure, which could easily expand into a PhD project. We will not dwell further on the dark art of stellar astrophysics. It is clear, however, that it is worth exploring the possibility of resolving stellar collisions using hydrodynamical models. At this point however, we consider MMAMS a viable option that at least leads to predictable behavior (see Section 7.4.4).

5.2.7 Continuing with Hydrodynamics after a Henyey Code Crash

In Section 5.2.5 we compared grid and SPH hydro simulations of a simplified exploding star. The initial conditions for these simulations were quite far from reality, if one is really interested in supernova explosions. Specifically, the density, temperature, and momentum structure of the two-component model in Section 5.2.5 could easily be improved if we adopted a Henyey stellar evolution code to generate the stellar structure at the moment of the explosion.

For our more realistic star, we will use the stellar structure from Section 7.4.5 and continue from the point when the stellar evolution model failed to converge, leading to a crash. As noted earlier, the termination of the stellar evolution code does not crash the AMUSE script, so we can pick up the last converged stellar model as input to a hydrodynamical model of the supernova explosion that crashed the Henyey code. For definiteness, we continue the calculation using an SPH code. (It would have been equally easy to adopt a grid-based code.)

We first convert the stellar model to an SPH particle realization, much as in Section 5.3.3. We then inject energy into the innermost SPH particles to start the explosion. Listing 5.5 shows how we inject 10^{51} erg into the core of the SPH realization of the Henyey stellar model (see also Section 5.2.5). Note that selecting the right number of SPH particles is rather tricky and can easily lead to artificial numerical effects in the simulations. If too few are selected, individual SPH particles may penetrate the stellar envelope and leave the star without transferring their energy. On the other hand, if the supernova energy is divided among too many SPH particles, it may not be sufficient to drive the supernova explosion. Ideally, the energy should be divided among the particles corresponding to the degenerate core of the stellar model, and this may be achieved with a sufficiently high resolution in the hydrodynamical simulation. After the energy is injected, we start the hydro code, feed it the SPH particle set, and let it run. The resulting evolution of the thermal, kinetic, and potential energies is presented in Figure 5.7.

```
1 def setup_stellar_evolution_model():
2     out_pickle_file = os.path.join(get_path_to_results(),
3                                    "super_giant_stellar_structure.pkl")
4     if os.path.exists(out_pickle_file):
5         return out_pickle_file
```

Listing 5.5: Routine to inject 10^{51} erg into the innermost regions of an SPH realization of an exploding star.

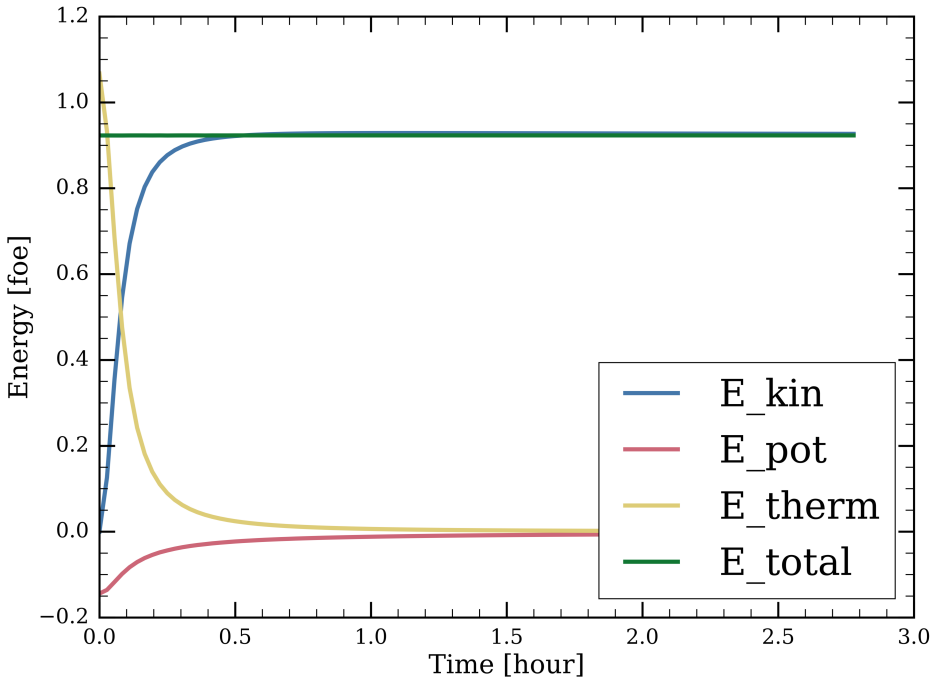

Figure 5.7: Evolution of the energy of an exploding 10 $M_\odot$ star. The star was first evolved using MESA to the tip of the asymptotic giant branch. After this the 1-dimensional Henyey model was converted to a particle (SPH) representation. At that point we injected 10^{51} erg into the stellar core and continued the hydrodynamical evolution using Gadget2. The figure was produced with `${AMUSE_DIR}/examples/textbook/explode_evolved_star.py`, which should run in about 10 minutes on a laptop.

The function get_path_to_results referenced in Listing 5.5 is imported from from amuse.test.amusetest, to provide a standard directory in which temporary files can be stored.

5.2.8 Extending the Hydrodynamics Solver

Many hydrodynamical studies focus on the evolution of a fluid flow with specified initial conditions, as discussed in Sections 5.3.1 and 5.3.3 below. In some cases, however, we need to understand what happens when gas enters or leaves the system. In this section, we explore how such processes are implemented in AMUSE.

5.2.8.1 Generating New SPH Particles

New gas may be introduced as part of the boundary conditions in the computational domain (such as the inflow of a gas stream from one side), or from a source within the simulated volume (such as a stellar wind or a supernova). In the SPH formalism, new gas is introduced simply by initializing new particles on some surface or in some volume around a source. The location of the source and the state of the gas are typically provided by another code, such as a stellar evolution calculation that generates a wind from an evolving star.

The code fragment below demonstrates how to create n_new new SPH particles in

an existing system, add them to the existing gas particles (gas_particles), and subsequently synchronize the gas particles in the main Python script with the particles in the hydro code.

```
new_gas = Particles(n_new)
give_particles_some_properties(new_gas)
gas_particles.add_particles(new_gas)
gas_particles.synchronize_to(hydro.gas_particles)
```

The astronomical magic is hidden in the routine give_particles_some_properties(), which provides the newly created particles with position, velocity, temperature, etc. The function synchronize_to ensures that the particle set in the main script is consistent with the particles in the hydro module. Note that we could have accomplished the same result by adding new_gas directly to the hydro particle set:

```
hydro.gas_particles.add_particles(new_gas)
```

Listing 5.6 presents an example of how to set the characteristics of the SPH particles. In this case, we are modeling the colliding winds of a binary star system. The positions and velocities of these stars typically come from a gravity code, while the stellar parameters are determined by a stellar evolution code. The amount of mass lost by each star is included in the star particles as a free parameter (star.MWind), which can be set in the main script from the stellar evolution code. The attribute MWind is not standard, but is temporarily added to the stellar particle set just for this purpose.

The new particles created in Listing 5.6 can be incorporated into the SPH code in two basic ways. At one extreme, they could be given high radial velocities but low temperatures, in which case the SPH code will establish a wind-like outflow that will shock and thermalize as it impacts the surrounding medium. Alternatively, at the other extreme, they could be given low velocities and high temperatures, consistent with the post-shock thermalized gas, as calculated by the routine in Listing 5.7. The wind speed and/or shocked temperatures can be taken from a stellar evolution code. In reality, the proper prescription lies somewhere in between.

We must make these choices because SPH is incapable of resolving the small-scale details of the true outflow, as the wind streams out, shocks, and thermalizes, so we must inject the energy as some combination of kinetic and thermal energy. The details can be resolved by some AMR grid models, but at the cost of very short time steps. The best choice between pure kinetic and pure thermal energy input is generally resolution-dependent. It is interesting to try both extreme methods to see the effect of these different strategies on the results of the simulation. In Section 5.3.4, we discuss an example in which a star accretes from a companion wind.

5.2.8.2 Removing SPH Particles

It is sometimes also necessary to remove gas from a system, for example when it leaves the computational domain, falls into a black hole, or is accreted by another object. Removing gas is not much harder than adding gas, as discussed in Section

```
def new_sph_particles_from_stellar_wind(stars, mass_per_gas_particle):
    new_sph = Particles(0)
    for i, star in enumerate(stars):
        number_of_gas = int(-star.Mwind / mass_per_gas_particle)
        if number_of_gas == 0:
            continue
        total_mass_of_gas = mass_per_gas_particle * number_of_gas
        star.Mwind += total_mass_of_gas
        new_gas = Particles(number_of_gas)
        new_gas.mass = mass_per_gas_particle
        new_gas.h_smooth = 0.0 | units.parsec

        dx, dy, dz = uniform_unit_sphere(number_of_gas).make_xyz()
        new_gas.x = star.x + (dx * star.radius)
        new_gas.y = star.y + (dy * star.radius)
        new_gas.z = star.z + (dz * star.radius)
        for j, gas_particle in enumerate(new_gas):
            r = gas_particle.position - star.position
            r = r / r.length()
            v_wind = (
                constants.G
                * star.mass
                / (gas_particle.position - star.position).length()
            ).sqrt()
            gas_particle.u = 0.5 * (v_wind) ** 2
            gas_particle.vx = star.vx + r[0] * star.terminal_wind_velocity
            gas_particle.vy = star.vy + r[1] * star.terminal_wind_velocity
            gas_particle.vz = star.vz + r[2] * star.terminal_wind_velocity
        new_sph.add_particles(new_gas)
    return new_sph
```

Listing 5.6: Routine to generate new SPH particles modeling the stellar wind of a set of stars. This listing is a fragment from the script ${AMUSE_DIR}/examples/textbook/hydro_outflow_particles.py. The full script models a colliding-wind binary system and will take several weeks to run on a laptop. In Section 8, we present an example of how this code works when coupled to gravity.

```
def v_terminal_teff(star):
    t4 = (np.log10(star.temperature.value_in(units.K)) - 4.0).clip(0.0, 1.0)
    return (30 | units.km / units.s) + ((4000 | units.km / units.s) * t4)
```

Listing 5.7: Determining the terminal velocity of an SPH particle for a star with a given temperature. This expression was used by Pelupessy & Portegies Zwart (2012) to determine the terminal velocity of the winds of massive stars.

5.2.8.1. We simply delete the unwanted gas particles from the local particle set, via particles_to_remove, and synchronize the particle set in the hydro code.

```
    removed_particles = particles_to_remove.copy()
    gas_particles.remove_particles(particles_to_remove)
    gas_particles.synchronize_to(hydro.gas_particles)
```

We store a copy of the removed particles in a local particle set called removed_particles. Later, we might use this particle set to check that we properly removed all particles, and that conserved quantities, such as mass, are actually conserved.

In many applications, particles are removed when they are accreted by a sink parti-

cle, which swallows gas particles when they approach within some specified distance. A sink particle has mass, position, radius, velocity, and possibly more parameters, such as some notion of the number of gas particles accreted, the accreted angular momentum, or the chemical composition of the accreted material. A sink particle can be declared just like any other particle (refer to Section 3.3.1 for an example). Here is a simple example of a non-moving sink particle with zero mass at the origin of the coordinate system:

```
    sink = Particle()
    sink.mass = 0 | units.MSun
    sink.radius = 1 | units.AU
    sink.position = (0, 0, 0) | units.AU
    sink.velocity = (0, 0, 0) | units.kms
    sink.accreted_mass = 0 | units.MSun
```

The radius of the sink particle is arbitrarily set here to 1 | units.AU, but in practice this radius reflects the distance from which it accretes.

Listing 5.8 presents a simple example of how we might decide which particles should be accreted by a collection of sink particles and removed from the calculation. Although simple, it shows the fundamental way in which gas accretion can be managed. This method is used, for example, in molecular-cloud collapse simulations, when sinks are formed in a Jeans unstable clump. The equations of motion of the sink can be integrated using an N-body code, while at the same time it can continue to accrete from the surrounding gas. The techniques needed to couple the gravity solver with a hydro code are discussed in Chapters 7 and 8. In Section 5.3.4, we present a simple example of a small experiment with a sink particle; a more elaborate example is discussed in Chapter 9.

```

def hydro_sink_particles(sinks, gas):
    removed_particles = Particles()
    for s in sinks:
        xs, ys, zs = s.x, s.y, s.z
        radius_squared = s.radius**2
        insink = gas.select_array(
            lambda x, y, z: (x - xs) ** 2 + (y - ys) ** 2 + (z - zs) ** 2
            < radius_squared,
            ["x", "y", "z"],
        )
        if len(insink) == 0:
            return insink

        cm = s.position * s.mass
        p = s.velocity * s.mass
```

Listing 5.8: A sink particle consumes gas particles. This is a fragment from the script `${AMUSE_DIR}/examples/textbook/hydro_sink_particles.py`

Stars (or other objects) may emit and accrete gas particles at the same time. For example, a protostar might accrete matter from the inner edge of a surrounding disk

while simultaneously emitting jets perpendicular to the disk. A simple way to realize this is to place the sink over the center of mass of the star, with an accretion radius smaller than the radius at which wind particles are released.

5.2.8.3 Sinks and Dark Matter Particles

In the previous section, we discussed the possibility of allowing sinks to form and to accrete material. Once a sink forms, it behaves like a point particle that interacts gravitationally but not hydrodynamically with the rest of the system, unlike a parcel of gas. Sinks are widely used in hydrodynamics codes as a way to avoid the need to follow star formation to small and very costly scales once a pocket of gas becomes Jeans unstable and starts to collapse. Combining the collapsing mass into a sink that continues to accrete but no longer behaves as a fluid particle realizes substantial savings in computer time. In many ways, sink particles in the AMUSE hydro modules are the fluid analogs of the binary and multiple systems managed by the multiples module (see Section 7.6.1)—except that interactions among sinks are far easier to handle, as we will see (Section 5.3.1.2).

Most hydrodynamics solvers can accommodate sinks because they are designed to include non-gas particles in the calculation. Some codes use built-in "sink particles" to follow the formation and accretion of protostars. Others include "dark matter particles" intended to model dark matter in galactic or cosmological contexts, but they too can be used to model sinks. When they exist in an SPH code, AMUSE refers to them as dm_particles to distinguish then from the gas_particles we have already encountered. Dark matter particles can be quite handy and are used in a wide variety of applications, including compact objects or as the degenerate core of a star on the giant branch. In the latter case, the giant's envelope would be comprised of gas particles. The more generic particle set particles is also available, and contains both the gas and the dm particles.

On adding a new particle to an SPH code, it is important to indicate whether it is a gas or a dm particle. The former is evolved according to the fluid equations of motion, possibly including self-gravity; the latter is affected only by gravity. Used as sinks, dark matter particles can continue to accrete from the surrounding gas, under the control of the top-level AMUSE script.

Dark matter particles in a hydro code are integrated according to Newton's equations of motion. In most codes, the integration itself is realized using a second-order Leapfrog scheme, and the gravitational potential is softened (see Section 3.1.4). This is probably sufficient for many hydrodynamical problems in which one is not interested in the subtleties of the gravitational dynamics. However, in Chapter 3, we discussed far more accurate gravity solvers, and it may be desirable to integrate the dark matter particles using one of those methods. This requires code coupling, which we discuss in Chapter 8.

5.3 Examples

In this section, we present four basic AMUSE experiments related to hydrodynamics. Some require only relatively simple initial conditions. Others require more complex initial conditions, as well as some notion of outflow and how to generate a disk around a star. Some of these simplified examples can easily form the basis for much more elaborate and rewarding experiments.

5.3.1 Collapsing Molecular Cloud

Figure 5.8 presents the results of a simple experiment that can be used for pedagogical purposes or to study how a model molecular cloud collapses. Similar simulations have been performed by many authors, but new simulations, even at relatively low resolution, often give interesting results if novel initial conditions are tried. One might, for example, start with a more complex cloud structure, perhaps based on an observed density distribution for which velocity, temperature, and/or composition information is available.

5.3.1.1 Initial Conditions

As a starting point, we adopt a homogeneous gas sphere with a power-law velocity power spectrum ($P(k) \propto k^{-4}$, where k is wavenumber) to mimic large-scale turbulence (Bate *et al.*, 2003). This is a popular initial model for simulations of star-forming molecular clouds. At the start of the simulation, we scale the internal energy such that the cloud begins to collapse, resulting in a cloud with a mean temperature of about 23 K. We recognize that these initial conditions are not very realistic, but they are convenient and widely used in the community. The resolution of our simulation is low (only 1000 particles, i.e. 10 $M_\odot$ per SPH particle), and the softening is large (~ 0.1 pc). We can see the cloud form interesting substructure as it evolves. The substructure in the collapsing cloud originates entirely from the initial velocity field, and while the initial model is idealized, the clumps and filaments that form are thought to be key ingredients in the collapse of real star-forming regions.

In the next few subsections, we extend this simple calculation in two key ways: adding a more realistic description of the equation of state of the gas, and allowing the possibility of star formation as the collapse proceeds. Both are implemented at the framework (Python) level.

1. In the previous section, we adopted an adiabatic equation of state, which is basically a cheap way of approximating the result of the interplay between heating and cooling processes. More complex hydrodynamical problems generally require some form of explicit heating and cooling. Not all hydro codes have explicit treatments of heating or cooling, but even if they don't, we can still heat and cool the gas via AMUSE. In Section 5.3.1.2, we describe how to introduce explicit cooling into the hydrodynamics solver to simulate the collapse of an isothermal molecular cloud.

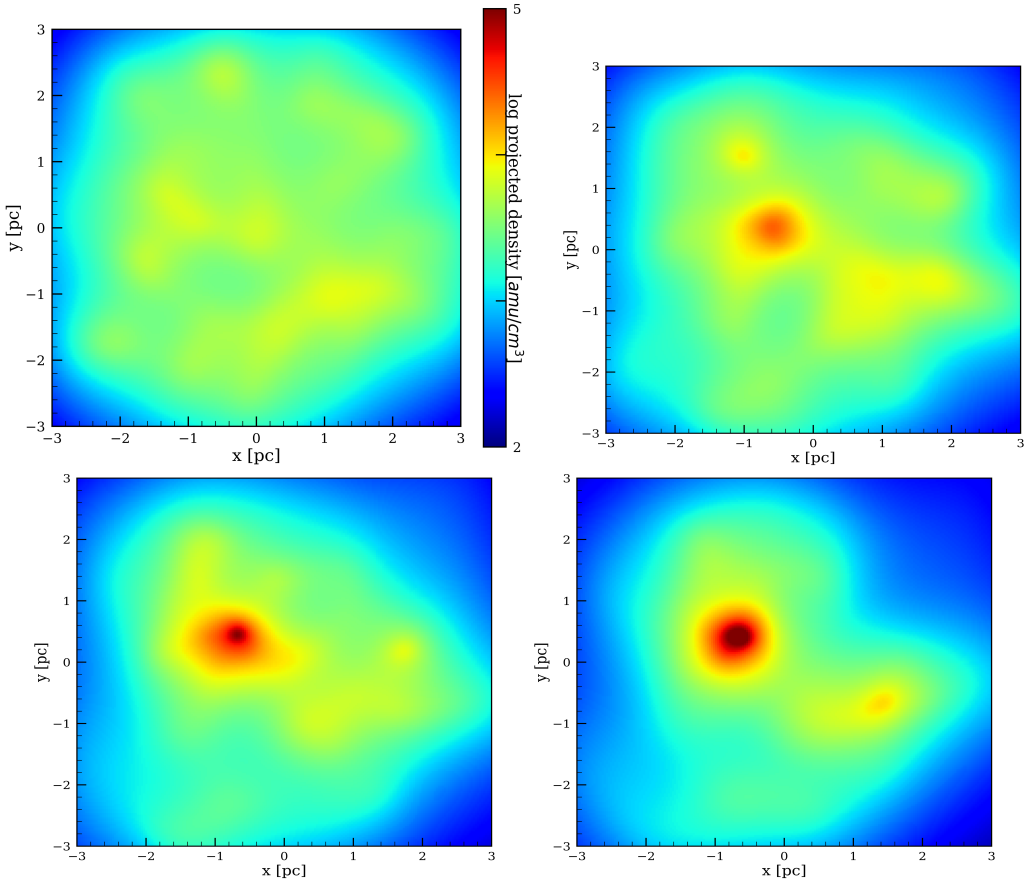

Figure 5.8: Collapse of a homogeneous $10^4\,\mathrm{M}_\odot$ gas cloud with an initial radius of 3 pc and a turbulent (power-law) velocity field. With these parameters, the free-fall time is $t_{\mathrm{ff}} \approx 0.86$ Myr, and the central density is 6.0×10^{-18} kg/m^3. The hydrodynamical evolution was carried out using the SPH code Fi using just 1000 SPH particles. The four panels show $x - y$ projections of the cloud at 0.25 Myr 0.50, 0.75, and 1.0 Myr. The same color scheme is used for each panel ($\log_{10}$ projected density). The simulation was performed using the script `${AMUSE_DIR}/examples/textbook/molecular_cloud_collapse.py`.

2. When individual parcels of gas become sufficiently dense to collapse, we turn them into sinks (see Section 5.2.8.3). These sinks can be integrated using the internal hydrodynamics N-body integrator (usually a Leapfrog or Verlet code; see Section 3.2.2). They can continue to accrete, or can merge if two sinks approach sufficiently closely. The details are described in Section 5.3.1.3. In Chapter 8, we will demonstrate how a high-order direct N-body code can be incorporated to integrate the equations of motion of the sink particles more accurately.

In these cases, we will find generally it desirable to significantly increase the resolution of our calculations. We now consider these enhancements in turn.

5.3.1.2 Cooling the Gas Using an External Routine

A script that performs the entire simulation with cooling and sinks can be found in ${AMUSE_DIR}/examples/textbook/molecular_cloud_collapse_with_sinks.py. Since this code is a bit more complicated than anything we have encountered so far, we have

split some of the ingredients into Python classes. New to this script is a hydrodynamics class hydro_model, defined in hydrodynamics_class.py, and a rudimentary cooling class cooling_model, in cooling_class.py.

Heating and cooling are computed and applied by the evolve_model() member function in the cooling class, which adjusts the internal energy and temperature of the gas according to a simple physical model, forcing the gas toward an isothermal equation of state on a time scale consistent with actual cooling processes (see Section 5.3.1.2). It was used by Lützgendorf *et al.* (2016) (see also Section 9.1) to cool the hot winds of the S-stars orbiting the supermassive black hole in the Galactic center. The results are applied directly to particles or grid cells (Gerritsen & Icke, 1997); the hydro code then takes care of the consequences.

The calling sequence between hydro_model and cooling_model is

```
    cooling = cooling_model()
    hydro = hydro_model()
    dt = 0.05 | units.Myr
    while time < t_end:
        time += dt/2
        channel_to_cooling.copy()
        cooling.evolve_model(time)
        channel_from_cooling_to_hydro.copy()
        time += dt/2
        hydro.evolve_model(time)
        channel_from_hydro_to_cooling.copy()
        cooling.evolve_model(time)
        channel_to_framework.copy()
```

Note that we advance the cooling for half a step, then use that temperature structure to advance the hydro for a full step, then complete the cooling step. This interlaced stepping strategy results in second-order coupling between the cooling and hydro codes. It is inspired by the second-order code coupling strategy discussed in Chapter 8. The same time-stepping strategy was adopted when combining stellar evolution and gravitational dynamics in Chapter 7. The choice of time step dt is a matter of experimentation and some art. Overly large time steps will lead to errors, but small time steps can make the code very slow. The proper time step is determined using convergence arguments (see Section 5.4 for a discussion).

5.3.1.3 Star Formation in a Collapsing Molecular Cloud

Star formation is implemented by creating sink particles in regions of gas that exceed a specified critical density density_threshold (see Section 5.2.8.2). The stopping condition for detecting high-density particles or cells is set by

```
    code.parameters.stopping_condition_maximum_density = density_threshold
    density_limit_detection = self.code.stopping_conditions.density_limit_detection
    density_limit_detection.enable()
```

The exception is raised when any high-density gas is detected. Loosely, “high-density” generally means something substantially above the average, although there is a more

formal prescription based on the resolution of the simulation, related to smoothing length, internal energy, and particle mass (Federrath *et al.*, 2010; Bleuler & Teyssier, 2014; Sormani *et al.*, 2017).

In Section 5.2.8.2, we used a simple particle as a sink, but we could do better by importing the AMUSE sink particle module. This internal module is not much different from a simple particle set, except that it takes care of some rudimentary aspects of conservation following accretion. It is accessed by

```
from amuse.ext.sink import new_sink_particles
```

Sink particles can be initialized from a host particle set host_particles:

```
    sink_particles = new_sink_particles(host_particles, sink_radius)
```

The host_particles set can be stars in a gravity code or dark matter particles in an SPH code. The parameter sink_radius here is the radius of the sink particle, which is used to determine when ambient gas is accreted and when sinks merge.

New sinks can be created after a hydrodynamics code stopping condition is reached, or (as here) by searching for high-density regions in the gas distribution. In either case, the Python script must search for high-density particles (or cells), then create new sinks and add them to the hydro code:

```
    highdens = gas_particles.select_array(lambda rho:rho>density_threshold, ["rho"])
    candidate_sinks = highdens.copy()
    gas_particles.remove_particles(highdens)
    newsinks_in_code = hydro.dm_particles.add_particles(candidate_sinks)
    newsinks = Particles()
    for nsi in newsinks_in_code:
        if nsi not in self.sink_particles:
            newsinks.add_particle(nsi)
    sink_particles.add_sinks(newsinks)
```

Here, the newly found sinks are added to the (previously created) particle set sink_particles.

When two sinks approach within sink_radius (typically the sum of the individual radii) of one another, they can be merged in much the same way as stellar mergers in the stellar evolution example in Listing 4.7 of Section 4.2.7. As soon as the mass of a sink exceeds some threshold, we could start a stellar evolution code and take stellar evolution into account in setting the mass and size of the growing star. However, increasing the stellar mass due to accretion onto the sink requires some more thought, which would take us beyond the scope of this chapter.

5.3.1.4 Incorporating Stellar Winds

To make the problem a little more realistic, we include stellar evolution using the parameterized stellar evolution code SeBa and take the winds generated by young stars (particle set stars) into account using the internal amuse.ext[**internal ext, seems contradictional – Steven**] stellar wind module:

```
from amuse.ext.stellar_wind import new_stellar_wind
```

This module generates new SPH particles near the mass-losing stars, with wind parameters (position, velocity, density, temperature) derived from the stellar evolution code, and hands these particles back to the SPH code. The SPH code later automatically takes care of incorporating the effect of the stellar wind into the residual gas.

```
    wind = new_stellar_wind(mgas, target_gas=gas, timestep=dt,
                            derive_from_evolution=True)
    wind.particles.add_particles(stars)
    channel_to_wind = stars.new_channel_to(wind.particles)
```

Here, mgas is the mass of an individual gas particle in the stellar wind generated over a time interval dt. Then the stars, which are also in the stellar evolution code, are added as particles to the wind module, and we make a channel from the stars to the wind module in order to ensure that changes to the stars due to stellar evolution are properly propagated to the wind module. In the main event loop, the stellar and wind modules are called and the newly generated wind particles are synchronized with the gas particles in the hydro code:

```
    stellar.evolve_model(time)
    channel_to_wind.copy()
    wind.evolve_model(time)
    gas_particles.synchronize_to(hydro_code.gas_particles)
```

In Section 9.1.1, we demonstrate how the wind module can be used in conjunction with stellar evolution, gravity and hydro codes to simulate the intricate coupling between strong stellar winds and the dynamics of the stars in orbit around the supermassive black hole in the Galactic center.

5.3.1.5 The Collapse Calculation

After implementing the above changes into the basic script discussed in Section 5.3.1, we can start to experiment with a more realistic collapsing gas cloud. (In Figure 8.7, we show another method to arrive an interesting initial conditions for the combined distributions of gas and stars.) Initially, the cloud mass was 10^3 $M_\odot$, distributed in 10^5 equal-mass particles, and the radius was 2 pc, with the same initial turbulent velocity field as before. Figure 5.9 presents a snapshot of the calculation 4.5 Myr after the start of the run. By this time, the age of the oldest star is almost 3.5 Myr, and the most massive star, born 1.5 Myr after the start of the simulation, is about 6.9 $M_\odot$.

Recently, Hacar *et al.* (2017) have measured the radio intensity of the molecular N_2H^+ emission in the embedded star cluster NGC 1333. This cluster lies at the center of a stellar clump with a diameter of $\sim$ 2 pc. Star formation has been going on there for some 3 Myr. Hacar *et al.* (2017) estimate the total mass in gas in NGC 1333 to be $\sim$580 $M_\odot$. The protostars are classified as type 0 to III (see Adams *et al.*, 1987; Lada & Lada, 2003), allowing an approximate age range to be associated with each. The radio

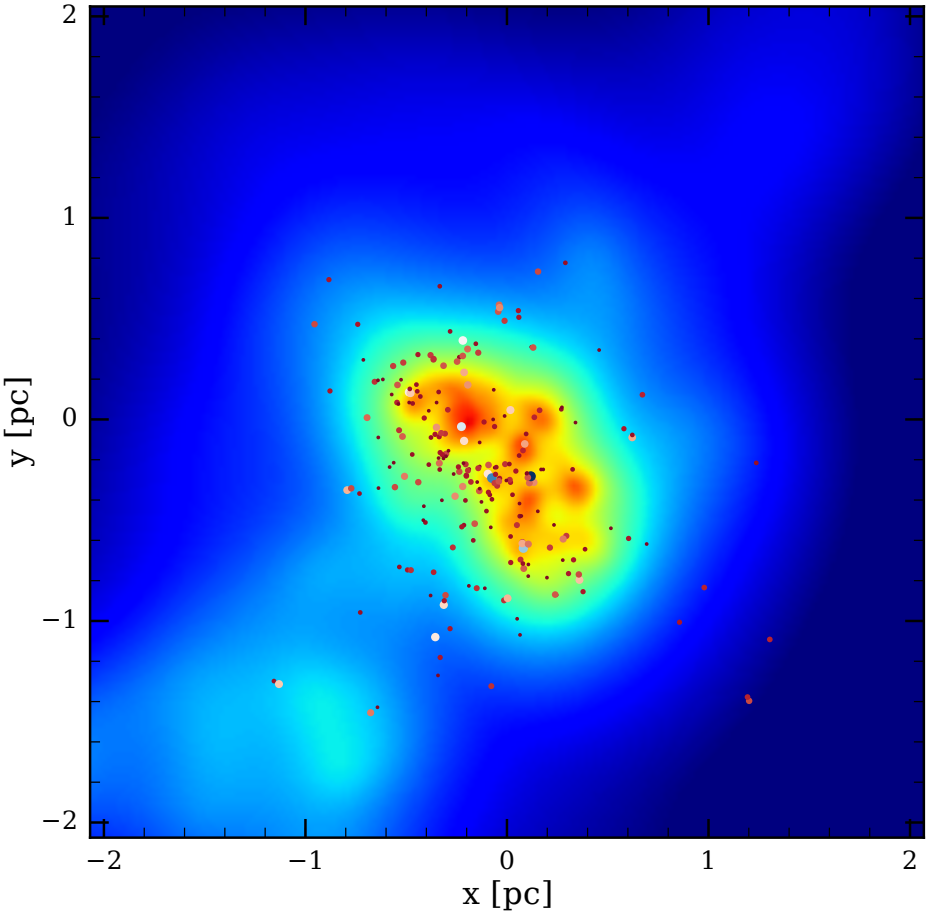

Figure 5.9: Projection of the central portion of a collapsing molecular cloud at an age of 4.5 Myr. The density scale is indicated to the right of the figure in $\log_{10}$ units of amu/cm^3. The colored bullet points are the stars formed from the densest gas (color coding from dark red for the low mass stars to bright white for the most massive stars). The size of the points is proportional to the logarithm of the mass; the most massive has mass 6.9 $M_\odot$. The script for this simulation is `${AMUSE_DIR}/examples/textbook/molecular_cloud_collapse_with_sinks.py`; it uses the two support routines in ${AMUSE_DIR}/examples/textbook/hydrodynamics_class.py and `${AMUSE_DIR}/examples/textbook/cooling_class.py`. This figure was generated using ${AMUSE_DIR}/examples/textbook/plot_molecular_cloud_collapse_with_sinks.py.

intensity of the observed lines is a proxy for surface density of the dense gas, giving us a (limited) view of the changing cluster environment. We can convert between N_2H^+ radio intensity I (in units of K·km/s) and projected molecular hydrogen density N (in cm^{-2}) using the approximate empirical prescription provided by Hacar *et al.* (2017):

$$N \simeq (7.9\, I + 8.2) \times 10^{21}. \tag{5.30}$$

In computing N for our simulation, we first remove gas with density exceeding 2.64×10^{-21} g/cm^3, because such gas would be unresolved in the observations. We then convert the projected density of the remaining gas in the model to intensity, in order to compare this with the observations.

Figure 5.10 shows the projected gas density profile in our simulation, relative to the center of mass of the stellar distribution, at an age of 4.5 Myr, and compares it with the inferred projected density distribution in NGC 1333. The various lines indicate the

densities in gas and stars. The squares are taken from Figure 3 of Hacar *et al.* (2017). The projected gas density in our simulation is somewhat smaller than the observed distribution, and our stellar density is somewhat higher. However, apart from the youngest stars, our model density distribution matches the observations reasonably well.

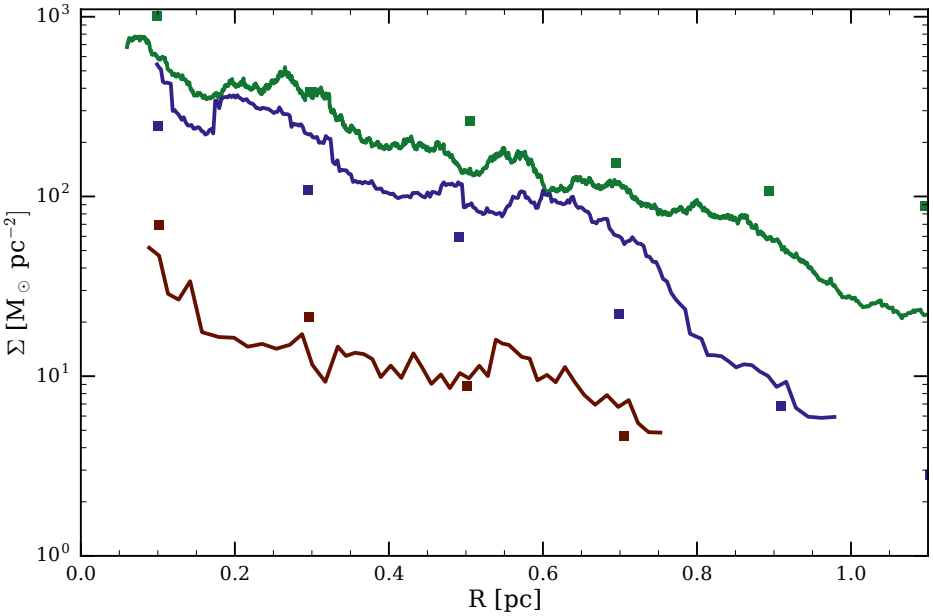

Figure 5.10: Projected density profile (on the x–y plane) of the simulated molecular cloud and stars at an age of 4.5 Myr (lines) and the observed data for NGC 1333 from (Hacar *et al.*, 2017; squares). The top curve (green) and squares represent the density distribution of the gas. The blue curve and squares are projected radial stellar density profiles; the red curve and squares show the density profile only of stars younger than 1.5 Myr (class 0 and class I objects). The observational data are taken directly from Figure 3 of Hacar *et al.* (2017). This figure was made using the simulation presented in Figure 5.9 and the script `${AMUSE_DIR}/examples/textbook/2017AandA...606A.123H_Fig3.py`.

Hacar *et al.* (2017) further argue that, on average, the younger stars (class 0 and I objects in the observations) are embedded in higher-density gas than the older stars (class II and III). They conclude that older and more massive stars blow bubbles around them, reducing the local gas density (as opposed to the alternative explanation that older stars have simply moved away from the sites where they formed). We test these assertions in Figure 5.11, where we plot the N_2H+ intensity of gas near young stellar objects in our simulations. The shaded area in Figure 5.11 is constructed from Figure 10 of Hacar *et al.* (2017). It represents the spread in the intensity distribution for each protostellar class. In this figure, we have attributed tentative ages to each observed class of object, from class 0 (~ 0.5 Myr) to class III (2–3 Myr).

The red crosses plotted in Figure 5.11 are from our cloud collapse simulation, including star formation, stellar evolution (using SeBa), and stellar winds. Because the simulation was performed with rather low mass resolution (0.5 $M_\odot$ per particle), we adopted a lower mass (0.005 $M_\odot$) for the wind particles; otherwise, an insufficient amount of wind would be liberated by the stars to generate even a single SPH particle during the simulation. Performing SPH calculations with a mixed mass resolution of the SPH particles can, under some circumstances, result in problems, such a spurious

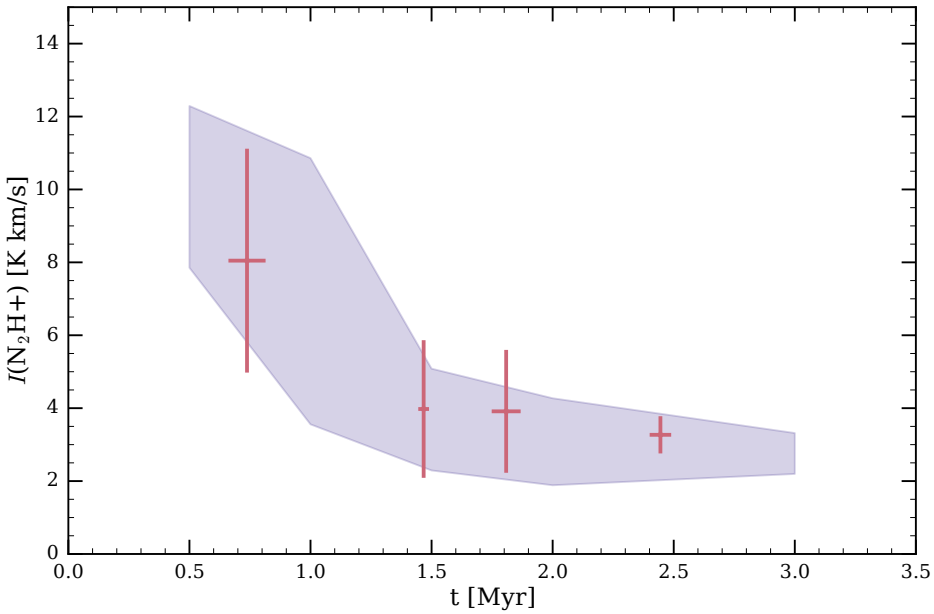

Figure 5.11: Intensity of emission from the molecule N_2H^+ in the direction of the newborn stars in our simulation. The horizontal axis is the age of the star; the vertical axis is (equivalent) intensity of N_2H^+ in units of K·km/s. The shaded region shows the observed mean intensity of N_2H^+ in NGC 1333 as a function of the ages of the protostars studied by Hacar *et al.* (2017). The error bars give the 90% confidence intervals for the N_2H^+ intensity of the simulated young stellar objects. We adopt the following age ranges: 0–1 Myr for class 0, 1–1.5 Myr for class I, 1.5–2 Myr for class II, and 2–3 Myr for class III. The figure was made using the simulation presented in Figure 5.9 and the script `${AMUSE_DIR}/examples/textbook/2017AandA...606A.123H_Fig10.py`.

mixing, if not handled carefully. However, when studying systems with small entropy gradients (Lombardi *et al.*, 1999) or large density contrasts, there may be no other option than to run with different particle masses. Some SPH codes are, in fact, designed specifically with these objectives in mind (Gaburov *et al.*, 2010). In Assignment 5.5.4, we encourage the reader to try this out by experimenting with multiple resolutions.

The comparison of the radial distribution of the projected stellar and gas densities (Figure 5.10 and of the observable intensity as a function of stellar age (Figure 5.11) match the observations reasonably well, supporting the Hacar *et al.* (2017) conclusions. This conclusion cannot be made if we start with a smaller ($\lesssim 1.4$ pc) or larger ($\gtrsim 3$ pc) molecular cloud, if we add more than 10% of break-up rotation to the cloud, or if we switch off the stellar-wind mass loss.

5.3.2 Circumstellar Disk with a Bump

Planets form in disks, but these disks tend to be lumpy. In this example, we show that a lump disperses in a few orbits. We start our simulations with an axisymmetric 1 $M_\odot$ disk, with an isothermal equation of state, around a 10 $M_\odot$ star. The density falls off toward the outer edge of the disk as a power law with exponent -3/2. The inner and outer edges are at 10 au and 100 au. These choices are largely motivated by the numerical cost of extending the disk closer to the central star. After generating the disk, we introduce a small perturbation in the form of a bump in a circular orbit

at a distance of 50 au from the star. The bump was generated using a Plummer gas sphere with a characteristic radius of 5 au and a mass of 0.5 $M_\odot$. Listing 5.9 presents a snapshot of the routine used to generate the disk and bump.

```
160
161
162 def new_disk_with_bump(
163     Mstar=10 | units.MSun,
164     Ndisk=100,
165     Mdisk=1.0 | units.MSun,
166     Rmin=1.0 | units.AU,
167     Rmax=100.0 | units.AU,
168     Mbump=0.1 | units.MSun,
169     Rbump=5.0 | units.AU,
170     abump=10 | units.AU,
171 ):
172
173     converter = nbody_system.nbody_to_si(Mstar, Rmin)
174     disk = ProtoPlanetaryDisk(
175         Ndisk,
176         convert_nbody=converter,
177         densitypower=1.5,
178         Rmin=1,
179         Rmax=Rmax / Rmin,
180         q_out=1.0,
181         discfraction=Mdisk / Mstar,
182     ).result
183     com = disk.center_of_mass()
184
185     # determine bump's local velocity
186
187     inner_particles = disk.select(lambda r: (com - r).length() < abump, ["position"
    ])
188     M_inner = Mstar + inner_particles.mass.sum()
189     v_circ = (
190         (constants.G * M_inner * (2.0 / abump - 1.0 / abump)).sqrt().value_in(units.
    kms)
191     )
192
193     # initialize bump
194
195     Nbump = int(Ndisk * Mbump / Mdisk)
196     bump = new_plummer_gas_model(
197         Nbump, convert_nbody=nbody_system.nbody_to_si(Mbump, Rbump)
198     )
199     bump.x += abump
200     bump.velocity += [0, v_circ, 0] | units.kms
201
202     disk.add_particles(bump)
203     disk.move_to_center()
204     return disk
```

Listing 5.9: Routine to generate a disk with a Plummer model bump. This listing is a fragment from the script `${AMUSE_DIR}/examples/textbook/hydro_disk_with_bump.py`.

The disk is constructed from a cylindrical slice in which all particles are in circular Keplerian orbits around the central star, with a radial density profile described by a power law and a scale height consistent with the adopted Safronov–Toomre Q-parameter (see below Safronov, 1960; Toomre, 1964). This is realized using the routine

```
disk = ProtoPlanetaryDisk(Ndisk, convert_nbody=converter,
                          densitypower=1.5, Rmin=1,
                          Rmax=Rmax/Rmin,
                          q_out=1.0, discfraction=1.0).result
```

which takes as arguments the number of disk particles, a converter, a density power law (here -1.5), the minimum and maximum disk radii, the Safronov–Toomre Q-parameter, and the mass fraction of the disk. The scaling parameters in this argument list (Rmin, Rmax, and discfraction) are relative to the converter parameters (in this example, 1 $M_\odot$ and 10 au).[1]

The Safronov–Toomre parameter provides a stability criterion for a differentially rotating disk. It can be derived from the Jeans stability criterion by comparing the self-gravity of the disk with its thermal pressure, and can be expressed in terms of the radial velocity dispersion σ_r (or the sound speed for gaseous disks), the epicyclic frequency κ, and the surface density Σ:

$$Q = \frac{\sigma_r \kappa}{\pi G \Sigma}. \tag{5.31}$$

Disks tend to be stable for $Q > 1$. In Listing 5.9, we generate the disk with $Q = 1$, which results in a barely stable, rather flat disk. (Later, in Section 6.6.1, we will present an example of a much more stable thick disk with $Q = 25$.)

The disk is generated using 50,000 SPH particles, with 25,000 more for the bump, and a softening length of $\varepsilon = 10$ au. We then select an SPH code for the hydrodynamics, in this case Gadget2. To keep things simple, we include the central star as a dark matter particle in the hydro code. We subsequently integrate the equations of motion of the star and the hydrodynamics of the gas for 2000 yr, which is long compared to the $\sim$ 100 yr Keplerian orbital period of the bump around the star. Figure 5.12 presents snapshots of the disk plus bump 2000 yr after the start of the simulation. By this time, the bump has been spread out over the disk, forming a nice spiral arm. We will study the survival time of the bump in Assignment 5.5.3

5.3.3 Colliding Stars

Collisions play an important role in the evolution of the stars in dense star clusters (Hills & Day, 1976), as well as for the formation of the Moon and other solar system objects (Benz *et al.*, 1986). In a pioneering study, Benz & Hills (1987) performed SPH simulations of the collisions of stars in a star cluster environment. Their simulations followed collisions of identical low-mass main sequence stars, modeled as simple $n = 3/2$ polytropes (Chandrasekhar, 1933), on both head-on and grazing trajectories. Their

[1] Instead of generating a standard protoplanetary disk, we could have achieved a similar initial condition using the more general routine to construct a uniform cylinder using uniform_unit_cylinder.

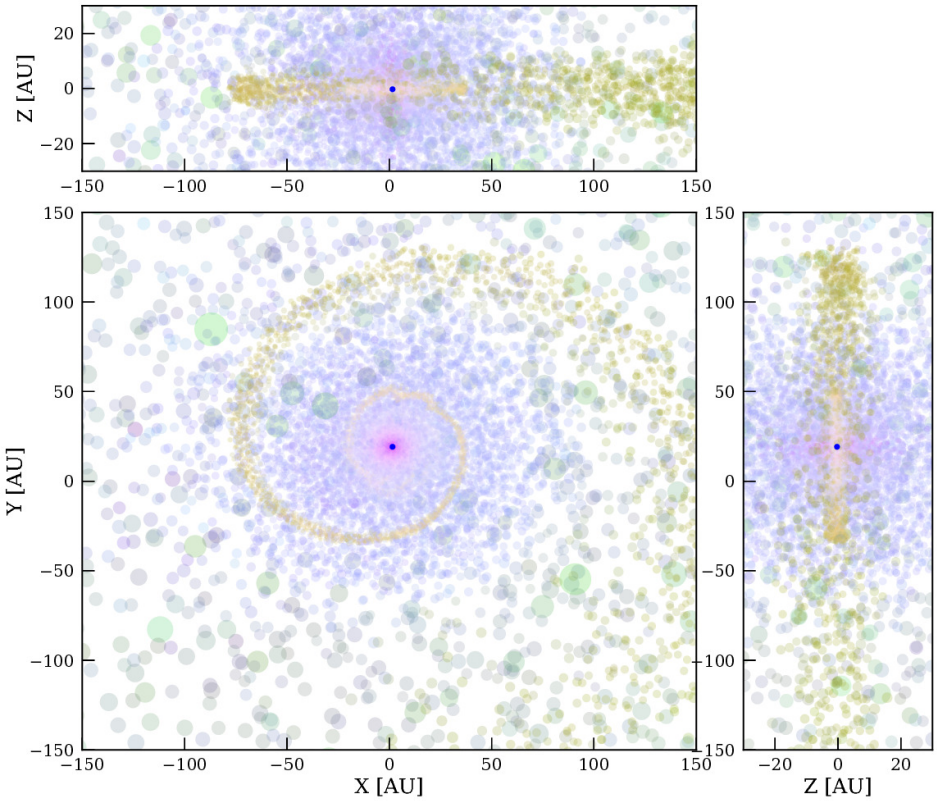

Figure 5.12: Face-on and edge-on views of the disk with a bump after 2000 yr. Individual SPH particles (75000 in total) are represented as semi-transparent bullet points. The sizes represent their height above the disk, relative to the z scale height, and the colors are generated from the pressure, velocity and temperature represented by red, green and blue, respectively. The lack of blue in the spiral arm, which is a result of the disrupted bump, is a result of the relatively low temperature of the bump with respect to the rest of the disk. The calculation was performed using the script `${AMUSE_DIR}/examples/textbook/hydro_disk_with_bump.py`, the figure was made with `${AMUSE_DIR}/examples/textbook/plot_hydro_disk_with_bump.py`.

main conclusion was that grazing collisions result in higher degrees of mixing of the inner and outer stellar layers than do head-on collisions.

Instead of two polytropes, we adopt here a full Henyey stellar evolution model of two 0.6 $M_\odot$ stars evolved to 12 Gyr using MESA (see Section 5.2). The two evolved stellar models were converted to SPH particle representations using the routine presented in Listing 5.2 and described in Section 5.2.6. Our SPH realization has a mass resolution of $20M_{\rm Earth}$ per SPH particle (significantly higher than the original calculations). Our head-on collision has zero velocity at infinity, so the relative speed of the two stars is $v = v_{\rm esc} = \sqrt{2Gm/r} \simeq 440$ km s^{-1}. Here, $m = m_1 + m_2$ is the total mass and r is the initial separation. Following Benz & Hills (1987), we start the two stars touching, so $r = r_1 + r_2$. The grazing collision has initial relative speed $v = 1.67 v_{\rm esc} \simeq 740$ km s^{-1}. Because Benz and Hills neglected to mention the adopted impact parameter, we simply placed the stars in contact with a transverse offset of $0.5(r_1 + r_2)$ (see their Table 1).

The results are summarized in Figure 10 of Benz & Hills (1987). Figure 5.13 presents a similar figure showing the results of our simulations. The leftmost four bars indicate the initial structure of the stars. The second set of bars indicates the degree

of mixing in the merger product following the head-on collision. The rightmost set shows the result of a grazing collision. Comparing our Figure 5.13 with Figure 10 of Benz & Hills (1987), we observe differences in the degree of mixing. The head-on case compares reasonably well with the original calculation. The innermost quartile of the merger product is refreshed with about 15% of material from the next quartile out, whereas the outermost region (fourth quartile) is enriched with roughly 20% of material from below (third quartile). Overall, the degree of mixing in this case is less than reported by Benz and Hills. In contrast, the grazing collision seems to mix the stars more effectively in our simulation than in the original calculations. Virtually all of the mass in the first quartile of the incoming stars is lifted to higher zones, more than 50% of it finding its way into the outermost two quartiles of the merger product.

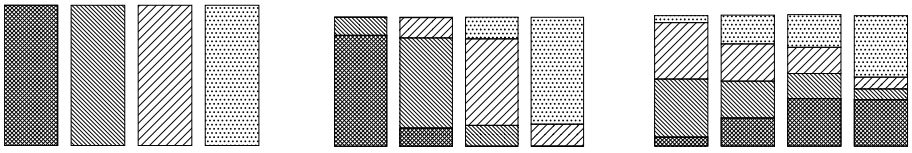

Figure 5.13: Recomputation of Figure 10 in Benz & Hills (1987), in which two 12 Gyr old 0.6 $M_\odot$ main-sequence collide with zero velocity at infinity. The left set of four vertical bars depicts the initial (pre-collision) structure of the stars; the various shading schemes represent the inner 25%, 50%, 75%, and the outer 25% of the mass. The middle set shows how the original stellar material is rearranged following a head-on collision, while the right-hand set shows the result of a grazing collision. The figure was made using the script `${AMUSE_DIR}/examples/textbook/plot_1987ApJ...323..614B.py` which depends on two other files `star_to_sph.py` and `1987ApJ...323..614B.py` in the same directory. Running the hydrodynamics of the collision takes a few hours on a laptop.

5.3.4 Accreting from the Wind of a Companion

In this example, we measure the amount of mass accreted by a 1 $M_\odot$ star from the wind of a 2 $M_\odot$ companion. Both stars have reached an age of 1.494 Gyr, at which time the 2 $M_\odot$ primary has lost some 0.0752 $M_\odot$ of its initial mass and has grown into a supergiant of radius 0.99 au and surface temperature 3300 K. Its wind speed is $v_w \sim 30$ km s^{-1}, with a mass-loss rate of $\dot{m} \approx 0.11 M_\odot$ Myr^{-1}. The 1 $M_\odot$ companion star has not lost much mass yet and is still on the main sequence. Its radius is 0.0043 au = 0.93 $R_\odot$.

We place the stars on a circular orbit at a separation of 10 au, and start the hydrodynamics solver. We adopt an SPH particle mass of $3 \times 10^{-11}\, M_\odot$ and integrate the hydrodynamics with a time step of one day, after which we launch new SPH particles using the simple wind module discussed in Section 5.2.8.1. Gas particles accreted by the 1 $M_\odot$ star, more than 200 au from the primary, are removed according to the method described in Section 5.2.8.2. Accretion is modeled by placing a sink particle (see Section 5.3.1.2) at the location of—and moving on the same orbit as—the 1 $M_\odot$ star. We include gravity by following the stars as dark matter particles in the hydro code.

Synchronization between wind particles and accreted particles is accomplished by

```
1    accreted = sink.accrete(interstellar_gas_particles)
```

```
    interstellar_gas_particles.remove_particles(accreted)
    interstellar_gas_particles.synchronize_to(hydro.gas_particles)
```

Here stellar_particles and sink_radius are, respectively, the stellar particles added to the sink and the radius of the sink.

Figure 5.14 presents the accretion rate onto the 1 $M_\odot$ star as a function of time. The red curve is calculated using a sink particle that did not move and had a radius equal to the star's Bondi–Hoyle–Lyttleton (Bondi & Hoyle, 1944; Hoyle & Lyttleton, 1939) radius of ∼2 au. [2] The blue curve is for a normal-sized star with self-gravity included. As can be seen, both approaches give the same results, within the statistical uncertainty, with a mean accretion rate of $\sim 0.0022 M_\odot$ Myr^{-1}. One might possibly argue that the Bondi–Hoyle–Lyttleton accretion starts a little earlier. This can be understood from the fact that particles have to travel less distance before being swallowed by the accreting star, simply because its accreting surface is closer to the companion.

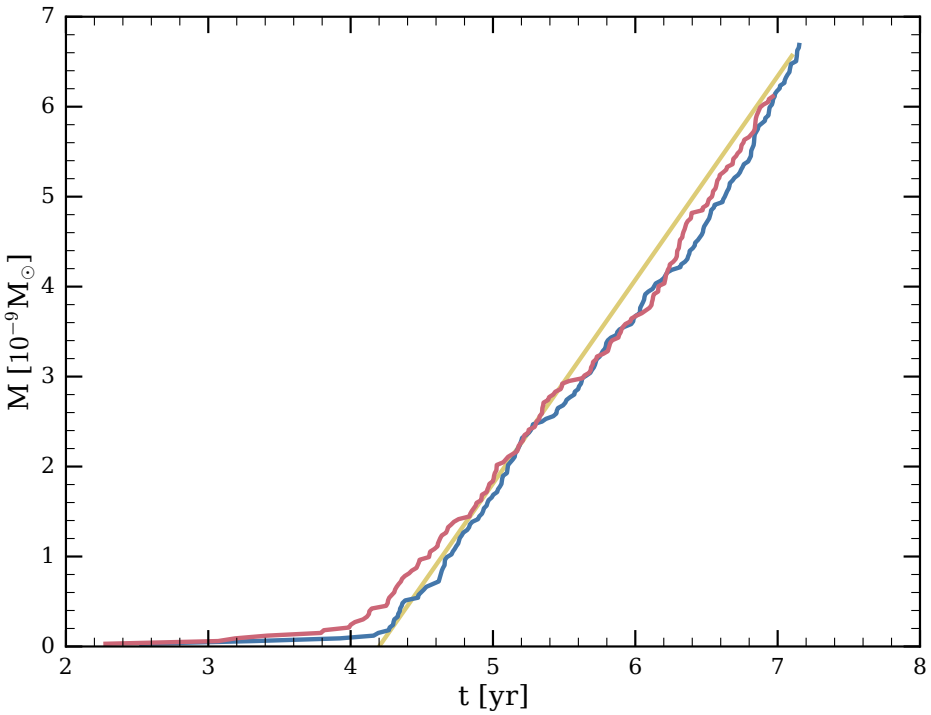

Figure 5.14: Accretion rate onto a 1 $M_\odot$ star embedded in the dense wind of a 2 $M_\odot$ companion on the asymptotic giant branch. The blue curve is calculated using the formal accretion radius from the stellar evolution code and the gravity solver in the hydrodynamics code. The red curve is calculated using the Bondi-Hoyle-Lyttleton accretion radius of the star (∼ 2 AU), without taking the primary's self-gravity into account. The yellow curve gives an estimate of the accretion rate using Eq. 5.32 and adopting a wind velocity of 17.3 km/s. The hydrodynamics was resolved using `${AMUSE_DIR}/examples/textbook/hydro_give_or_take.py`. The plot was made with `${AMUSE_DIR}/examples/textbook/plot_accretion_from_windy_star.py`.

The accretion rate from a steady wind can be approximated as (Theuns *et al.*, 1996)

$$\dot{M} \propto \mu^2 k^4 (1+k^2)^{-3/2} \dot{M}_{\mathrm{donor}}. \tag{5.32}$$

[2] The Bondi–Hoyle–Lyttleton radius R_{BHL} is the effective accretion radius of a gravitating body. For an object of mass M and incoming material moving at speed v, $R_{\mathrm{BHL}} \simeq 2GM/v^2$.

Here, $\mu \equiv m_2/(m_1+m_2)$ and $k \equiv v_{\rm orb}/v_{\rm wind}$. The mass ratio hardly changes during the integration and $\mu \simeq 0.34$, but the velocity in the wind at the location of the companion star is considerably lower than the terminal wind velocity used as input for generating the wind. The wind velocity measurement in the simulation that included self-gravity resulted in $\langle v_{\rm wind} \rangle \simeq 17.3$ km s^{-1}. With this wind velocity, the accretion rate calculated with Equation 5.32 is consistent with the simulations (the yellow line in Figure 5.14).

5.4 Validation

Validating gravity codes is relatively straightforward (see Section 3.6). Conservation of energy and angular momentum is generally the key to reliable results. Hydrodynamics is a very different realm. The most common way to validate the results of a hydrodynamics solver is by means of convergence tests, where one repeats a calculation with the same initial conditions and code settings, but with different resolution, i.e., we vary the grid size or the number of SPH particles, depending on the type of code used. Convergence tests provide quantitative information about the behavior of the code, and (with luck) on the correct solution of the studied system. However, even if the results converge, there is still no guarantee that the converged solution (or, more often, the solution approaching convergence) is correct. This requires additional validation.

Explicit validation is only possible in a few very limited cases where an analytic or semi-analytic solution is known (Landau & Lifshitz, 1959). More often, one has to compare new simulation results with previously computed "standard" solutions (Klingenberg *et al.*, 2007). Such comparisons have been undertaken for SPH (Thacker *et al.*, 2000) and for grid-based hydrodynamics, as in the Santa Barbara Cluster Comparison Project (Frenk *et al.*, 1999). Such comparisons reveal that both Eulerian mesh codes and standard SPH methods without artificial conductivity[3] (in addition to artificial viscosity) have significant problems even in rather idealized simulations (see also Saitoh & Makino, 2016). [**Are we mentioning STARBENCH here? – Steven**]

Thus, it is almost impossible to validate the results obtained by hydrodynamics codes, at least at the level of confidence typical in pure gravitational problems, and we advise caution in interpreting the results. In this section, we focus on validating codes via comparison with an exact solution (Section 5.4.1), and by comparing the results of a number of different codes. In the latter case, no analytic solution exists, but conventional wisdom states that, if several different codes give similar results, those results are probably qualitatively correct.

5.4.1 Riemann Shock Tube Problem

The one-dimensional Riemann shock tube test is one of the few nonlinear hydrodynamical problems for which an analytic solution exists (Landau & Lifshitz, 1959). The test setup consists of two fluids of different densities and pressures separated by a membrane at the center of the grid. The simulation starts when the membrane is suddenly

[3] Artificial conductivity is an additional dissipative term introduced into the energy equation in order to handle discontinuities in the fluid's thermal energy (Monaghan, 1997; Price, 2008).

removed. The solution shows three distinct discontinuities in the fluid: (1) a contact discontinuity that marks the current interface between the two fluids, (2) a shock wave that moves from the high-pressure fluid (left) into the low-pressure fluid (right), and (3) a expansion wave that moves in the opposite direction (Tasker *et al.*, 2008).

Figure 5.15 presents the exact Riemann solution at time 0.4 (in dimensionless code units; see Section B.2.5), as well as the solutions for two hydro codes—one grid-based and one SPH. The exact solution is taken from Toro (1997). The numerical setup follows Tasker *et al.* (2008). All modern grid solvers are designed to pass this test (in fact, they are built around this solution as the means of computing fluxes between adjacent cells), so it is not surprising that the grid code performs extremely well. However, while SPH in one dimension also accurately reproduces the analytic solution, the three-dimensional SPH code used here clearly has far greater difficulty than the grid-based code at comparable resolution. Note also that the way the initial conditions were set up favors the grid code. We started with analytic density and pressure distributions, which naturally map onto a grid-based representation, whereas in the SPH representation, random fluctuations in the particle's positions and velocities lead to Poisson and shot noise.

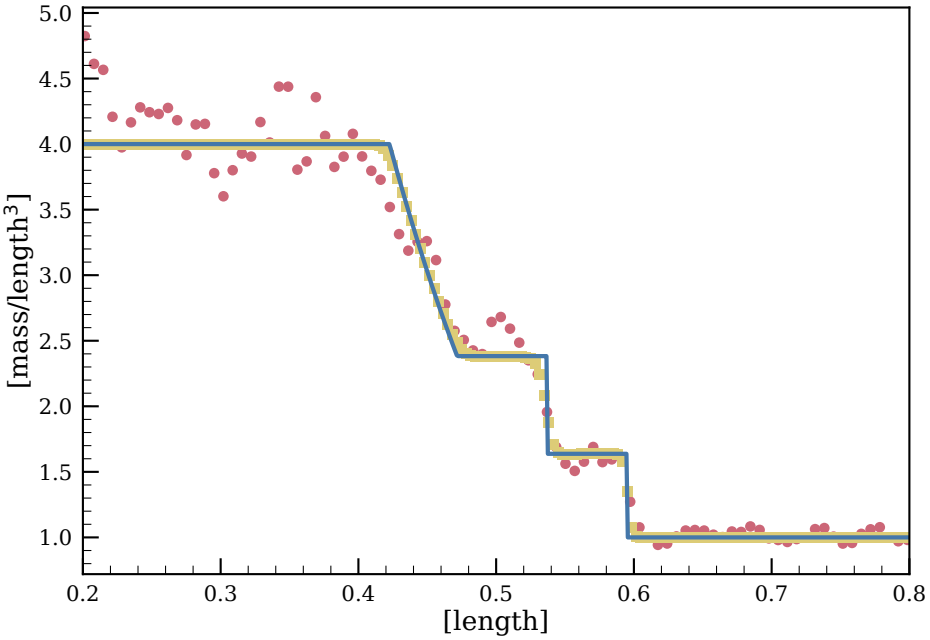

Figure 5.15: Riemann shock tube test using two fluids with a density ratio of 4 to 1, pressure ratio of 1 to 5.571, and with $\gamma = 5/3$. The solid blue line gives the analytic solution to this problem. The yellow squares are the results from the athena grid code in one-dimensional mode with 100 cells spanning the computational domain. The red bullets indicate the results using the SPH code Fi calculated in three-dimensions using 10^7 particles. To be fair we note that athena is working here in 1D, whereas the SPH code Fi operates in 3D, using a rather small number of particles for this test. The apparent poor result for the SPH code then is really a demonstration of how low resolution in SPH can still give reasonable results. The source code for this test can be found in `${AMUSE_DIR}/examples/applications/test_riemann_shock_tube_problem.py`. The figure was made using the script `${AMUSE_DIR}/examples/textbook/plot_riemann_shock_tube_rho.py`. Running the analytic code takes a few seconds, athena takes about 10 minutes and Fi about 18 hours (single core) on a laptop.

5.4.2 Kelvin–Helmholtz Test

The Kelvin–Helmholtz instability was discussed in Section 5.1.4.2. Because this instability is fundamental to fluid dynamics, any solver must be tested to determine whether the growth rate of the instability and its developing structure are consistent with earlier calculations (see also Section 5.4). However, it is not precisely known how the development of the instability is affected by the resolution—this is still an active area of research, involving both laboratory experiments and numerical simulations (Kajishima & Taira, 2017). A small script to perform a Kelvin–Helmholtz test is part of the AMUSE test suite, and can be found at `${AMUSE_DIR}/examples/simple/kelvin_helmholtz.py`.

Figure 5.16 presents the results of the Kelvin–Helmholtz instability test for three different grid-based hydrodynamics solvers. The red fluid moves to the left and has twice the density and half the speed of the blue fluid, which moves to the right. Even though the three images differ in detail, the characteristic structure of the Kelvin–Helmholtz instability is clearly evident. We did not perform this test for the SPH codes, because they tend to perform very poorly unless we go to extremely high resolution and introduce a special treatment for mixing (Read *et al.*, 2010).

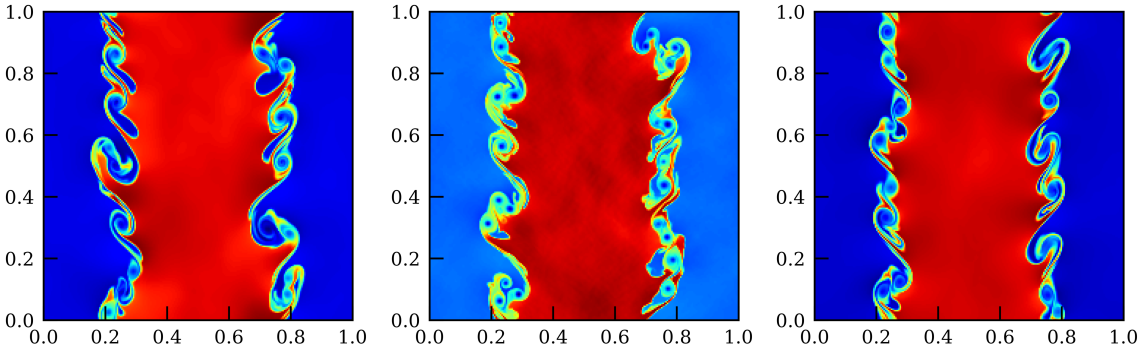

Figure 5.16: Result of the Kevin-Helmholtz test for three different grid-based hydrodynamics codes, athena (left), capreole (middle), and mpiamrvac (right), on a 2-dimensional 211 × 211 mesh (see Pelupessy *et al.*, 2013; for details). We seeded the initial system with random velocities of amplitude $0.01c_s$. Here we show the density distribution at $t = 1$. The source code took for this test about a minute to run and can be found in `${AMUSE_DIR}/examples/textbook/kelvin_helmholtz.py`.

5.4.3 Cloud-shock Test

Another hard test for hydro codes is the cloud-shock test, more popularly known as the blob test (Agertz *et al.*, 2007). This test aims to study interacting multi-phase fluids, which are generally poorly resolved by SPH techniques, whereas Eulerian grid-based methods are able to resolve the most important dynamical instabilities. In this test, a cloud is compressed by a high Mach number inflow, and forms a contact discontinuity between the high-density and the low-density material. The experiment is meant to mimic the shock-induced star formation process, except that there is no gravity. We present the results of this test using three different grid-based hydrodynamics solvers in AMUSE in Figure 5.17.

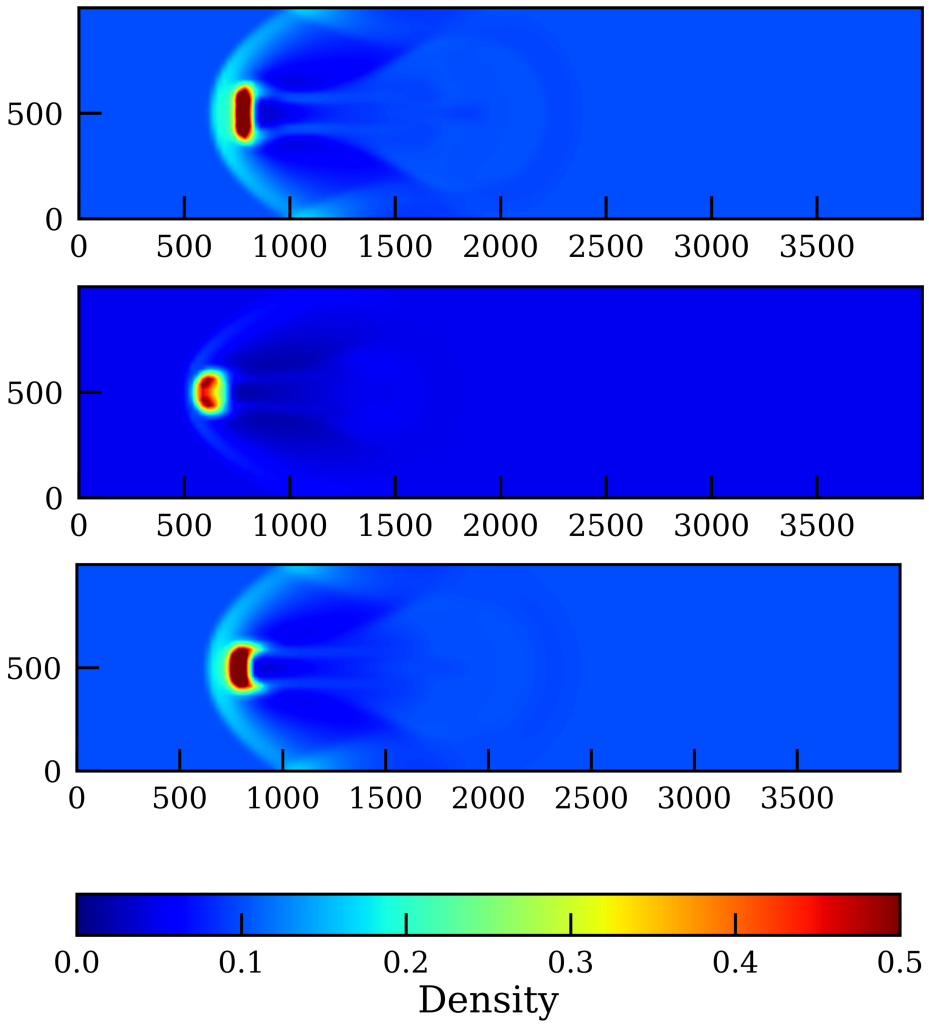

Figure 5.17: Comparison of the results of the cloud-shock test (Agertz *et al.*, 2007) for 3 grid-based hydrodynamics codes capreole (top), mpiamrvac (bottom), and athena (bottom). The color bar indicates the relative normalized density. Panels show slices through the density distribution at time $t = \tau_{\rm KH}$ (see Eq. 5.25). The resolution for athena and capreole is $320\times 320\times 1280$, for `mpiamrvac` it is $80\times 80\times 320$ with 2 levels of refinement for comparable effective resolution. The source code for this test is a standard application in AMUSE and can be found in `${AMUSE_DIR}/examples/textbook/cloudshock.py`. This image was first presented in Pelupessy *et al.* (2013). The various codes have a quite similar runtime of about 2 minutes.

Smoothed particle hydrodynamics codes use the local density distribution to derive the spatial derivatives of other quantities, which therefore should always remain differentiable. It is exactly this assumption that breaks down in the blob test, causing SPH codes to perform poorly. By working instead with the internal energy density (or pressure), which changes smoothly across the discontinuity boundary, as the volume element used for the kernel integration, the contact discontinuity can be resolved without numerical problems (Saitoh & Makino, 2013). However, these methods are currently not implemented in the SPH codes in AMUSE. [**Check if this is still true – Steven**]

5.4.4 Boss–Bodenheimer Test

Finally, we perform the Boss–Bodenheimer test for fragmentation in a rotating protostar (Boss & Bodenheimer, 1979). This test is often used to validate cloud collapse and the star formation process in hydrodynamical simulations. In this case, we use an SPH code, which provides a satisfactory solution. The experimental setup is a uniform spherical isothermal gas cloud of mass 1 $M_\odot$ and radius 0.01 pc, to which a bar-like density perturbation of amplitude 50% is added. The bar initially rotates as a rigid body with constant angular speed 1.56×10^{-12} rad s^{-1}. The calculation was performed using Fi with 50,000 SPH particles. The cloud collapses on a time scale of roughly the initial freefall time ($t_{\rm ff} \approx 0.175$ Myr), and forms two blobs containing approximately 80% of the total mass. The remaining gas is ejected.

In the AMUSE test, illustrated in Figure 5.18, we can qualitatively see how the perturbation develops. A more useful validation is realized by making a comparison with other simulation codes. This test could, for example, be run relatively easily using a grid-based hydrodynamics solver, and the reader may take up this challenge in Section 5.5.2.

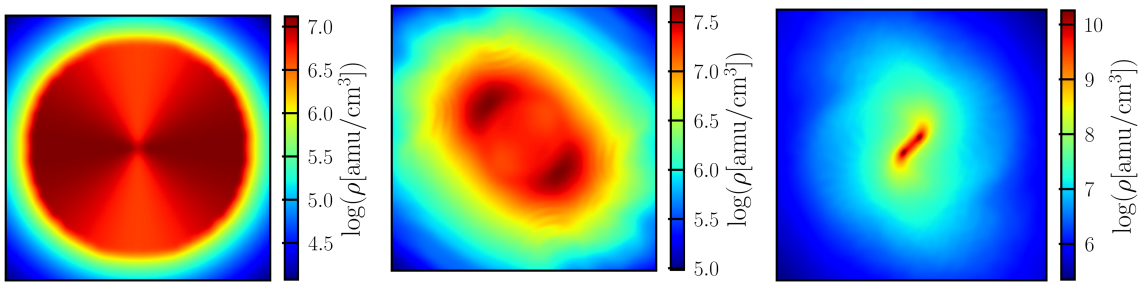

Figure 5.18: Three snapshots of the Boss–Bodenheimer test for a $1\,M_\odot$ uniform isothermal spherical cloud with radius 0.01 pc. Initially (left), a non-axisymmetric 50% $m = 2$ perturbation is visible. It rotates as a rigid body with constant angular speed of 1.56×10^{-12} rad/s. The middle image shows the projected gas distribution at an age of $0.836t_{\rm ff}$. The right panel shows the state of the system at $1.5t_{\rm ff}$. The calculation was performed using Fi with 50,000 SPH particles. The source code for this test is a standard simple AMUSE example and can be found in ${AMUSE_DIR}/examples/simple/boss_bodenheimer_test.py. It runs in about five minutes on a laptop.

There is no analytic solution to this problem, except that one can check conservation of mass, energy, and angular momentum throughout the test. We stopped the calculation after 1.5 $t_{\rm ff}$, because the local densities in the two blobs became rather high, dramatically slowing the SPH code. We could continue the calculation beyond this stage by introducing sink particles and allowing them to absorb some of the highest-density gas (see Section 5.3.1.2).

5.5 Assignments

5.5.1 Convergence Test

Ideally, aside from unphysical behavior of a code due to incorrect or absent physics, its behavior should be independent of the number of particles in the SPH code, the opening

angle in a tree code, the softening length, or the number of cells in the grid code. In reality, the results generally depend on these resolution and accuracy parameters. In practice, you will probably want to perform production calculations with the minimum number of grid cells or SPH particles that produce acceptable results, in order to have as efficient a simulation as possible. (Here, efficiency can be expressed in terms of computer time, wall-clock time, disk storage, memory usage, etc., and is generally problem-dependent.)

For hydrodynamics codes, the number of SPH particles or the number of grid cells are the most common handles for measuring convergence. It is generally best to start a convergence test with as few SPH particles or as coarse a grid as one dares. In subsequent simulations, the resolution can be increased, typically by factors of two, until the outcome becomes independent of the resolution. The outcome might be a graph with some measure of the resolution (number of grid cells in each dimension, or the number of SPH particles) along the x-axis, and the desired result on the y-axis.

In practice, simulations do not always reach convergence, because the required resolution may be beyond the available resources, such as computer time or total memory. In those cases, it is still possible to validate the result so long as the behavior of the system asymptotically approaches a certain value, and the performance curve can be extrapolated to the desired resolution.

In Section 5.2.6, we discussed how a collision between two stars can be resolved using an SPH code. Here, we use this example to carry out a convergence test for the SPH simulations. We perform the test by increasing the number of SPH particles from $10/\mathrm{M}_\odot$ to $10^4/\mathrm{M}_\odot$, then increasing the number of particles by a factor of 10 for each subsequent simulation. The results are presented in Figure 5.19. We will run the simulation as described in Section 5.2.6.2 and plot the temperature and density profiles of the merger product with varying resolution.

- Perform your own resolution test, starting with two stars of masses 6 $\mathrm{M}_\odot$ and 5 $\mathrm{M}_\odot$, each evolved to an age of 100 Myr.
- For the test case in Figure 5.19, the results seem to converge rather quickly. Is this a general result? Do your simulations also converge for $N \gtrsim 1000$ SPH particles per $\mathrm{M}_\odot$?
- Plot the wall-clock computer time as a function of resolution, and discuss how the cost in computer time grows with increasing resolution.
- Instead of using an SPH code, the same problem can also be done with a grid code (see Section 5.2.5). In that case, use the number of grid cells as a measure of resolution. Perform a similar analysis as for the SPH convergence test and address the same questions.

5.5.2 Testing the Boss–Bodenheimer Test

In Section 5.4.4, we performed the Boss–Bodenheimer test using the SPH code Fi. We continued the calculation up to 1.5 t_{ff}, after which two blobs of relatively high density

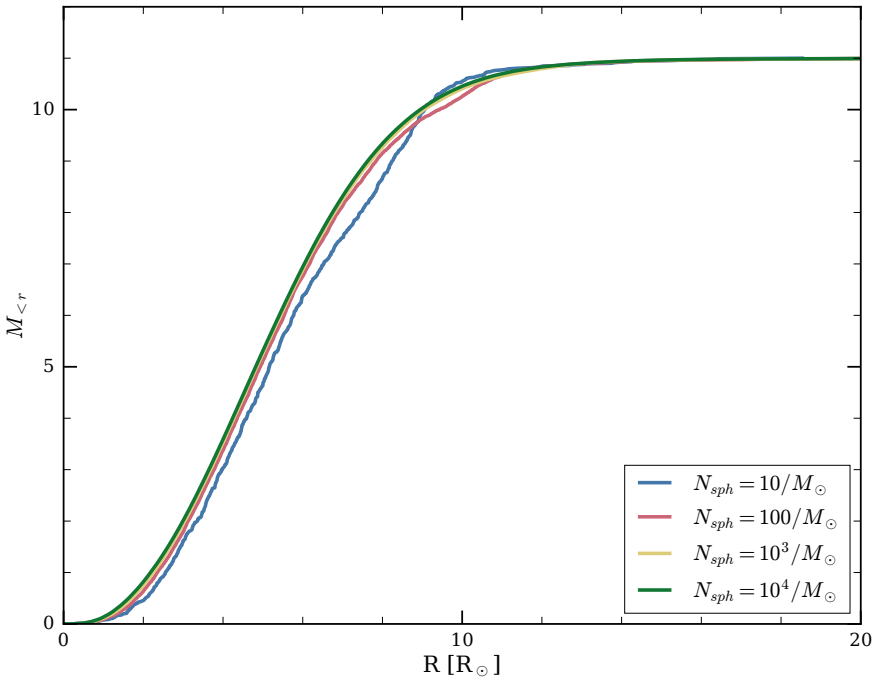

Figure 5.19: Convergence test for a hydrodynamical simulation of a merger of two stars of masses 10 $M_\odot$ and 1 $M_\odot$, with initial parameters otherwise identical to those discussed in Section 5.2.6. Here we adapt the calculation to perform a convergence test using 10, 100, 10^3 and 10^4 equal-mass SPH particles per solar mass to resolve the stars. The source code for this convergence test is `${AMUSE_DIR}/examples/textbook/merge_two_stars_sph_convergence.py`.

started to appear. This is the classic result of the Boss–Bodenheimer test. We stopped the calculation because the high density forced the SPH code to run very slowly.

We can imagine at least two solutions that enable us to continue the calculation. One is to introduce sink particles in the highest-density regions of the gas distribution and integrate them as dm_particles. Another is to repeat the test using one of the available grid-based codes in AMUSE. Specifically, the adaptive mesh refinement code Athena should do a reasonable job of following the formation of the binary system in this test calculation; mpiamrvac is less suitable because the test requires self-gravity. Running the SPH code with sink particles is straightforward, but converting the three-dimensional particle distribution including the velocities to a grid is somewhat more elaborate.

Do the grid and SPH codes give similar masses and orbital parameters for the two stars that form?

5.5.3 The Dissolving Bump

In Section 5.3.2 we discussed the dissolution of a massive perturbation in a protoplanetary disk around a 10 $M_\odot$ star. In this assignment, we will study the time scale on which such a bump dissolves. From the calculation in Section 5.3.2, we already know that the bump has dissolved after about 20 orbits, but we did not discuss the actual time scale for dissolution. The time scale can be derived from the calculations if we are able to recognize the bump at run time and measure its mass.

The bump in the hydro particle set can be identified by using hop (see Section 5.2.6.3). In this assignment you will repeat the calculation in Section 5.3.2, but identify and measure the bump's mass at run time. Make a plot of the mass of the bump as a function of time, and discuss the mass evolution. What happens to the bump-dissolution time scale if the mass of the central star and the mass of the disk and bump are all reduced by a factor of 10?

5.5.4 Collapsing Molecular Cloud with Sink Particles

In Section 5.3.1 we demonstrated the collapse of a giant molecular cloud. Even at this rather low resolution, the high densities reached in the simulation brought the code to a grinding halt. The solution was to introduce sink particles to reduce the computational load on the hydro solver. The size of a sink particle, and hence the limiting density that can be resolved, depends on the resolution of the simulation. This limiting density can be calculated by assuming that the volume containing $n_{\rm crit} = 64$ particles (a typical number of SPH neighbors) is barely Jeans unstable (see Section 5.1.4.3):

$$\rho_{\rm threshold} = \frac{\pi^5 e^3}{36 G^3 m^2 n_{\rm crit}^2}. \tag{5.33}$$

Here, $e \simeq c_{\rm s}^2$ is the specific internal energy of the gas, which in our case (a cold molecular cloud of temperature $T = 15$ K, similar to what we used in Section 5.3.1) is $e \simeq 0.08$ km^2 s^{-2}, and $m \simeq 2 \times 10^{29}$ kg is the mass of a single SPH particle in a somewhat more refined simulation capable of resolving solar-mass objects. We convert between e and T using

$$T = \frac{2e\mu}{3k}, \tag{5.34}$$

where

$$\mu = 2m_p \left(X + \tfrac{1}{2}Y\right)^{-1} \simeq 2.4 m_p \tag{5.35}$$

for molecular hydrogen gas and solar composition. In this case, we find $\rho_{\rm threshold} \simeq 8.5 \times 10^{-20}$ g cm^{-3} and a sink radius of $\lambda_J = 0.13$ pc (see Equation 5.27).

Run the hydro code and generate sink particles as soon as the local density (i.e., the density of a single SPH particle) exceeds the threshold density $\rho_{\rm threshold}$. Turn that particular particle into a sink particle by removing it from the gas and adding it to the hydro code as a dm_particle. We will have to do the mergers of the sink particles by hand, as we discussed in Section 7.4.1. Monitor the number of sink particles, and how their total mass increases with time. To validate the method, also plot the maximum gas density found in any of the SPH particles as a function of time. This maximum density should be limited by the adopted threshold density.

Measure, in several calculations, the moment when the first sink particle forms. Use the hydro code stopping condition to check for sink particle formation, then terminate the integration and print the time at which this occurs. This time is not always the same, but varies from run to run. Does this time also vary systematically for different resolutions? Why is it not the free-fall time scale?

5.5.5 A Star-forming Region

In Section 5.3.1.5 we discussed the correlation of the intensity of N_2H+ emission with the positions of protostars and young stellar objects. Our calculation was realized with dual resolution using two different masses for the particles in the SPH code, in order to allow a relatively low-resolution model of the star formation process. This is usually a bad idea (but see Section 5.3.1.5), and can cause a whole raft of other problems, but in that example we wanted to have some quick results in order to obtain a first impression of the physical processes and their consequences. Here, we encourage the reader to further discuss this resolution issue.

Perform the same simulation as in Section 5.3.1.5, with a total of 200,000 SPH particles (corresponding to a mass resolution of 0.005 $M_\odot$), and reproduce the two figures Figures 5.10 and 5.11. Was the observed numerical correlation a consequence of the dual resolution in the SPH simulation?

5.5.6 Neutron Star Hits Companion

Construct a hydrodynamical realization of a main sequence star of mass 10 $M_\odot$. Fire a neutron star at that star, from a distance equal to twice the stellar radius ($R_\star$) at a velocity of 1000 km s^{-1}. This is pretty fast, but during a supernova, the newly formed neutron star can acquire such a high speed due to asymmetries in the explosion. This problem can be run with a grid or an SPH code, although the latter makes it easier to include the neutron star as a point mass in the form of a dm_particle.

1. Perform a convergence test on the number of SPH particles for this problem where the neutron star is launched head-on into the star (see Section 5.5.1).

   Run a first test with a few (say, 100) SPH particles in the target star, and measure the behavior of the system. Does the neutron star stick to the companion star? How much mass is lost by the target star, and how much kinetic energy does the ejected mass carry? Run subsequent simulations with higher resolution, 500 SPH particles, 2000 articles, 4000, etc., until the results converge or your patience runs out.

   Make a figure of the resulting diagnostics of the convergence test as a function of resolution.

2. Repeat the convergence test for an impact parameter of $d = 0.5R_\star$. Was the resolution adopted after the initial convergence test (for $d = 0$) sufficient?

3. Finally, repeat the experiment with the same target star on the giant branch. Sadly, the convergence test will have to be redone, because the different structure of the target star is likely to change the behavior of the adopted code.

   For this second experiment, the same code used to generate Figure 5.13 (`${AMUSE_DIR}/examples`) can also be use used to make an SPH realization of a giant star. You may want to use a dm_particle to represent the stellar core, in order to avoid spending all the available computer time on the hydrodynamics of the giant's central region.

5.5.7 Supernova Explosion

In Section 5.2.5 we discussed the use of a simple two-component stellar model with constant density in each of the components to serve as a starting model for a hydrodynamical simulation of an exploding star. It is relatively straightforward to instead adopt the result of an actual Henyey code and convert the one-dimensional density structure of a star into a three-dimensional grid. In Section 5.2.5, we explored a similar approach but converted the stellar grid structure to a particle representation in order to resolve the hydrodynamics using an SPH code.

In this assignment, we will simulate a type Ic supernova using a grid code. We start with a 12 $M_\odot$ star, which we evolve to a radius of 100 $R_\odot$, which happens at an age of ∼14.6 Myr. We assume that, at this stage, the star overfills its Roche lobe in the binary system to which it belongs, and loses its envelope. The stripping of the hydrogen-rich layers has to be done in the stellar structure model, using a Henyey code; see Section 4.2.7. Listing 5.10 demonstrates how this can be done.

```
    number_of_zones = se_star.get_number_of_zones()
    composition = se_star.get_chemical_abundance_profiles()
    # first zone with X > 1.0e-9
    index = (composition[0] > 1.0e-9).nonzero()[0][0]

    print("Creating helium star, from the inner", index,
          "(out of", str(number_of_zones)+") shells.")
    helium_star = stellar_evolution.new_particle_from_model(
        {
            "mass": (se_star.get_cumulative_mass_profile() * se_star.mass)[:index],
            "radius": se_star.get_radius_profile()[:index],
            "rho": se_star.get_density_profile()[:index],
            "temperature": se_star.get_temperature_profile()[:index],
            "luminosity": se_star.get_luminosity_profile()[:index],
            "X_H": composition[0][:index],
            "X_He": composition[1][:index] + composition[2][:index],
            "X_C": composition[3][:index],
            "X_N": composition[4][:index],
            "X_O": composition[5][:index],
            "X_Ne": composition[6][:index],
            "X_Mg": composition[7][:index],
            "X_Si": composition[7][:index]*0.0,
            "X_Fe": composition[7][:index]*0.0
        },
        0.0 | units.Myr
    )
```

Listing 5.10: Snippet demonstrating how to strip the outer helium-rich layers of an evolved star. This is a fragment from the script `${AMUSE_DIR}/examples/textbook/helium_star.py`

Figure 5.20 shows the evolution of a 12 $M_\odot$ star in a Hertzsprung–Russell diagram, up to the time when it reaches a size of 100 $R_\odot$. The star is subsequently stripped of its hydrogen layers, using the procedure in Listing 5.10. This leaves us with a helium star of $\sim$ 1.85 $M_\odot$, which we continue to evolve using the Henyey code for another 0.21 Myr.

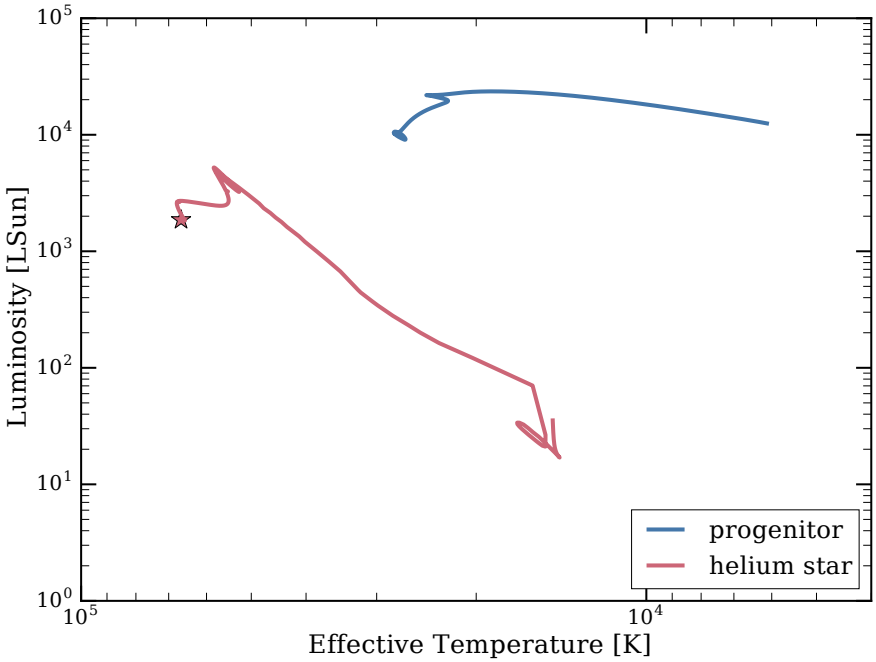

Figure 5.20: Blue curve: Evolutionary track of a 12 $M_\odot$ star for 14.6 Myr. The outer layers of the star are then stripped, as described in the text, and the helium core is evolved for a further 0.21 Myr (red track), eventually exploding as a supernova, marked by the star. The source code for this test is a standard simple AMUSE example and can be found in ${AMUSE_DIR}/examples/textbook/helium_star.py.

Ideally, we would like to continue evolving the star until just before the supernova occurs, but the AMUSE Henyey codes will likely encounter problems and crash at an earlier stage. Using a stopping_condition or exception handling to catch the right moment to ignite the supernova does not really work in these cases. Instead, we simply ignite the supernova at the moment at which the Henyey code fails to converge. In the example from `${AMUSE_DIR}/examples/textbook/helium_star.py`, we evolved the helium star for 10^3 internal time steps, because this brings the star sufficiently close to exploding.

Once the stellar evolution is finished, convert the one-dimensional Henyey representation of the star into a three-dimensional grid. Use this grid as the initial conditions for exploding the star, using the "thermal bomb" approach adopted earlier. Inject a total energy of $\sim 10^{51}$ erg into the inner 1.4 $M_\odot$ of the star, and divide that explosive energy across this volume, assuming constant energy per unit mass.

As a second objective, use an SPH code to simulate the supernova explosion. In that case, you will have to convert the initial three-dimension stellar realization into a representative particle set. There are two complementary ways to do this:

- Adopt the method described in Section 5.3.3, where the Henyey model is converted directly to a particle set, or from the grid generated from the Henyey code at the beginning of this exercise.

- Use the separate routine convert_grid_to_SPH to convert the grid you generated from the Henyey code directly to a particle set.

For the grid code and the SPH simulations, plot the temperature and the density in the supernova shell, as functions of time.

- Compare the temperature profile of the stellar evolution model with the result after the stellar model is converted to the grid.
- Run both initial realizations with the same grid-based hydro code, and compare the results.

5.5.8 Hoag's Object

In Section 3.4.4, we discussed how an SPH code can be used as a gravity solver. We presented an example in which two galaxies collided. There are a number of phenomena that can be simulated quite effectively this way. Examples include the impending collision between the Andromeda galaxy and the Milky Way, reproducing the Antennae galaxies, and Hoag's object. Figure 5.21 is an image of Hoag's object, as seen by the *Hubble Space Telescope* in July 2001.

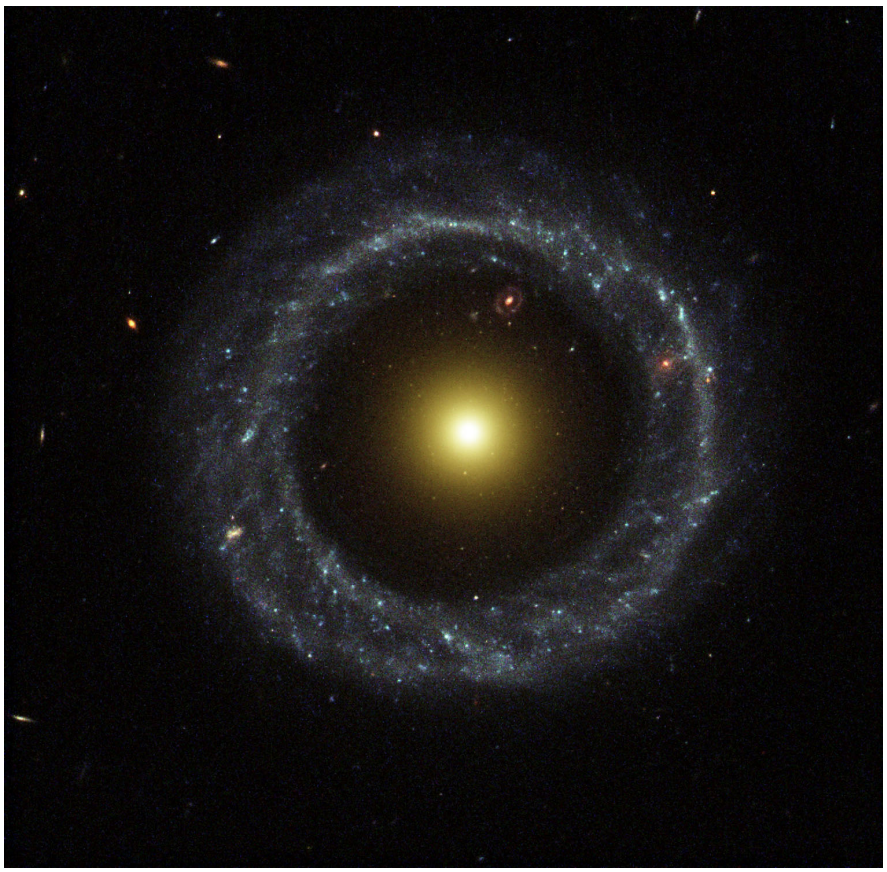

Figure 5.21: Hoag's object, from the Hubble Space Telescope. Image by NASA/ESA and The Hubble Heritage Team (STScI/AURA).

Hoag's object was discovered by Arthur Hoag (Hoag, 1950), who wrote that the object "*appears to be a perfectly symmetrical planetary nebula*". He ended the short

article with "*Since the object is unique, I suggest that a proper identification would be a worthwhile short-term project.*". By now, more than 60 papers have been written about it, and similar objects appear to be rather common (Buta, 2017). Even though the origin of its peculiar shape is still under debate, it appears to be quite stable (Bannikova, 2018). Two leading models include a vanishing bar (Freeman *et al.*, 2010) and the possibility of an interaction between a large, massive disk-like galaxy and a much smaller but denser elliptical galaxy. If the latter is aimed at the center of the former, it may simply penetrate, leaving the larger galaxy distorted. With the proper angles, velocities, mass ratio, and sizes of the two galaxies, this may be a way to make a resulting galaxy that looks like Hoag's object.

Construct two galaxies, one disk galaxy with a size comparable to Hoag's object's outer ring (diameter roughly 35 kpc), and a small spherical galaxy or massive globular cluster. Put both galaxies in an orbit such that the spherical galaxy hits the disk galaxy face on and along the axis of symmetry. For what parameters can a ring galaxy be created in this way? For guidance, refer to Fiacconi *et al.* (2012). Use GalactICs (Kuijken & Dubinski, 1995, 2011) to generate the disk galaxy and new_halogen_model (Zemp *et al.*, 2008; Zemp, 2014; Avila Perez & Murray, 2015) for the "bullet." Monitor how both galaxies are distorted in the interaction. Is this a promising way to make ring galaxies? Can you explain the central spheroid in Hoag's object with such an interaction?

An alternative way to construct something like Hoag's object is by allowing a single disk galaxy to evolve for a very long time. In that case, the bar, which continues to grow from the beginning, sweeps out the inner disk material. We found this process by accident when we were at the conference *Supercomputing 2014* in New Orleans. The run was performed on the Swiss CSCS Piz Daint supercomputer, and we streamed the data directly to the conference venue, where it was presented on a tiled display at the NVIDIA booth. The run was started and managed by Jeroen Bédorf, but we forgot the stop the run at night. It continued until, at an age of 24 Gyr, the image in Figure 5.22 appeared. We were immediately struck by its resemblance to Hoag's object.

Bibliography

Adams, Fred C., Lada, Charles J., & Shu, Frank H. 1987. Spectral Evolution of Young Stellar Objects. *ApJ* , **312**(Jan.), 788.

Adams, M. et. al. 2014. Chombo Software Package for AMR Applications Design Document. *In: Lawrence Berkeley National Laboratory: Lawrence Berkeley National Laboratory, LBNL Paper LBNL-6616E.*

Agertz, Oscar, Moore, Ben, Stadel, Joachim, Potter, Doug, Miniati, Francesco, Read, Justin, Mayer, Lucio, Gawryszczak, Artur, Kravtsov, Andrey, Nordlund, Åke, Pearce, Frazer, Quilis, Vicent, Rudd, Douglas, Springel, Volker, Stone, James, Tasker, Elizabeth, Teyssier, Romain, Wadsley, James, & Walder, Rolf. 2007. Fundamental differences between SPH and grid methods. *MNRAS* , **380**(3), 963–978.

Avila Perez, Santiago, & Murray, Steven. 2015 (May). *HALOGEN: Approximate synthetic halo catalog generator.* Astrophysics Source Code Library, record ascl:1505.017.

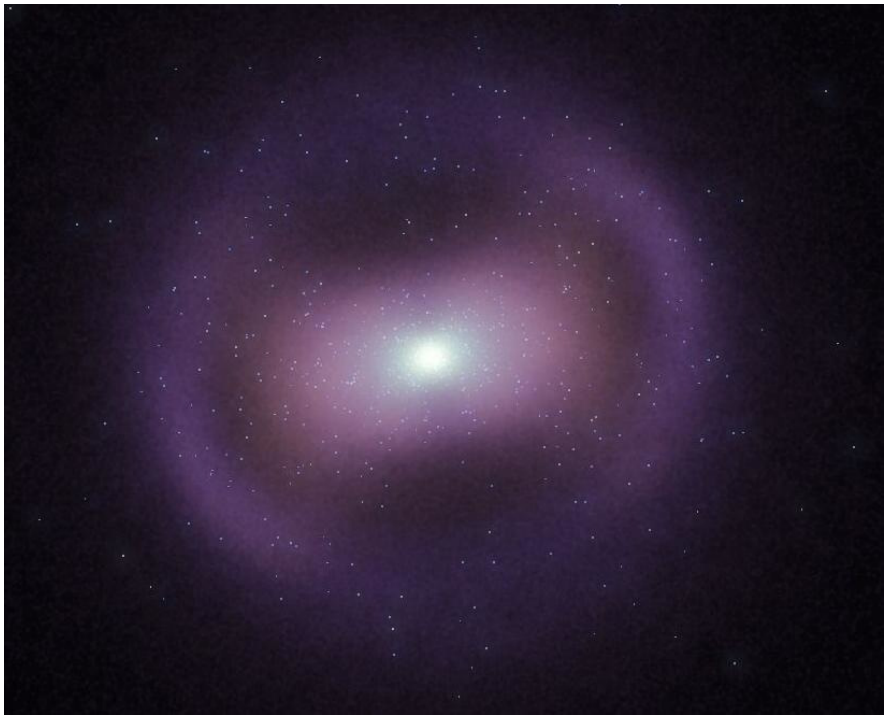

Figure 5.22: Result of a gravitational tree code simulation using Bonsai (Bédorf *et al.*, 2014). The initial condition was a disk galaxy but by accident we let the computer run to 24 Gyr after which this picture emerged, with some interesting resemblance to Hoag's object. The simulation was performed using 100 million particles initially distributed in a Milky-Way-like disk. This image was taken by one of the authors during the SC14 conference in New Orleans.

Bannikova, Elena Yu. 2018. The structure and stability of orbits in Hoag-like ring systems. *MNRAS* , **476**(3), 3269–3277.

Bate, Matthew R., Bonnell, Ian A., & Bromm, Volker. 2003. The formation of a star cluster: predicting the properties of stars and brown dwarfs. *MNRAS* , **339**(3), 577–599.

Bedorf, J., & Portegies Zwart, S. 2020. Bonsai-SPH: A GPU accelerated astrophysical Smoothed Particle Hydrodynamics code. *SciPost Astronomy*, **1**(May), 1.

Bédorf, Jeroen, Gaburov, Evghenii, Fujii, Michiko S., Nitadori, Keigo, Ishiyama, Tomoaki, & Portegies Zwart, Simon. 2014. 24.77 Pflops on a Gravitational Tree-code to Simulate the Milky Way Galaxy with 18600 GPUs. *Pages 54–65 of: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '14. Piscataway, NJ, USA: IEEE Press.

Benz, W., Slattery, W. L., & Cameron, A. G. W. 1986. The origin of the moon and the single-impact hypothesis I. *Icarus* , **66**(3), 515–535.

Benz, Willy, & Hills, Jack G. 1987. Three-dimensional Hydrodynamical Simulations of Stellar Collisions. I. Equal-Mass Main-Sequence Stars. *ApJ* , **323**(Dec.), 614.

Bernoulli, D. 1738. Specimen theoriae novae de mensura sortis. *Imp. Acad. Sci. St. Petersburg*, **5**, 175–192.

Binney, James, & Tremaine, Scott. 1987. *Galactic dynamics*.

Bleuler, Andreas, & Teyssier, Romain. 2014. Towards a more realistic sink particle algorithm for the RAMSES CODE. *MNRAS* , **445**(4), 4015–4036.

Bondi, H., & Hoyle, F. 1944. *MNRAS* , **104**, 273.

Boss, A. P., & Bodenheimer, P. 1979. Fragmentation in a rotating protostar: a comparison of two three-dimensional computer codes. *ApJ* , **234**(Nov.), 289–295.

Bradt, Hale. 2014. *Astrophysics Processes*.

Buta, Ronald J. 2017. Galactic rings revisited - I. CVRHS classifications of 3962 ringed galaxies from the Galaxy Zoo 2 Database. *MNRAS* , **471**(4), 4027–4046.

Chandrasekhar, S. 1933. The equilibrium of distorted polytropes. I. The rotational problem. *MNRAS* , **93**(Mar.), 390–406.

Coggeshall, S. V. 1991. Analytic solutions of hydrodynamics equations. *Physics of Fluids A*, **3**(5), 757–769.

Colella, P., & Woodward, Paul R. 1984. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics*, **54**(Sept.), 174–201.

Courant, R., Friedrichs, K., & Lewy, H. 1928. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, **100**(Jan.), 32–74.

Federrath, Christoph, Banerjee, Robi, Clark, Paul C., & Klessen, Ralf S. 2010. Modeling Collapse and Accretion in Turbulent Gas Clouds: Implementation and Comparison of Sink Particles in AMR and SPH. *ApJ* , **713**(1), 269–290.

Fiacconi, Davide, Mapelli, Michela, Ripamonti, Emanuele, & Colpi, Monica. 2012. Adaptive mesh refinement simulations of collisional ring galaxies: effects of the interaction geometry. *MNRAS* , **425**(3), 2255–2266.

Freeman, Tarsh, Howard, Sethanne, & Byrd, Gene G. 2010. Hoag's object, remnant of a vanished bar? *Celestial Mechanics and Dynamical Astronomy*, **108**(1), 23–34.

Frenk, C. S., White, S. D. M., Bode, P., Bond, J. R., Bryan, G. L., Cen, R., Couchman, H. M. P., Evrard, A. E., Gnedin, N., Jenkins, A., Khokhlov, A. M., Klypin, A., Navarro, J. F., Norman, M. L., Ostriker, J. P., Owen, J. M., Pearce, F. R., Pen, U. L., Steinmetz, M., Thomas, P. A., Villumsen, J. V., Wadsley, J. W., Warren, M. S., Xu, G., & Yepes, G. 1999. The Santa Barbara Cluster Comparison Project: A Comparison of Cosmological Hydrodynamics Solutions. *ApJ* , **525**(2), 554–582.

Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNeice, P., Rosner, R., Truran, J. W., & Tufo, H. 2000. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *ApJS* , **131**(1), 273–334.

Gaburov, E., Lombardi, J. C., & Portegies Zwart, S. 2008. Mixing in massive stellar mergers. *MNRAS* , **383**(1), L5–L9.

Gaburov, Evghenii, Lombardi, James C. Jr., & Portegies Zwart, Simon. 2010. On the onset of runaway stellar collisions in dense star clusters - II. Hydrodynamics of three-body interactions. *MNRAS* , **402**(1), 105–126.

Gardiner, Thomas A., & Stone, James M. 2005. An unsplit Godunov method for ideal MHD via constrained transport. *Journal of Computational Physics*, **205**(2), 509–539.

Gardiner, Thomas A., & Stone, James M. 2008. An unsplit Godunov method for ideal MHD via constrained transport in three dimensions. *Journal of Computational Physics*, **227**(8), 4123–4141.

Gerritsen, J. P. E., & Icke, V. 1997. Star formation in N-body simulations. I. The impact of the stellar ultraviolet radiation on star formation. *A&A* , **325**(Sept.),

972–986.

Godunov, S. K. 1959. A finite-difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics. *Math. Sb.*, **47**, 271–290.

Godunov, S. K., & Kulikov, I. M. 2014. Computation of discontinuous solutions of fluid dynamics equations with entropy nondecrease guarantee. *Computational Mathematics and Mathematical Physics*, **54**(6), 1012–1024.

Hacar, A., Tafalla, M., & Alves, J. 2017. Fibers in the NGC 1333 proto-cluster. *A&A* , **606**(Oct.), A123.

Hernquist, Lars, & Katz, Neal. 1989. TREESPH: A Unification of SPH with the Hierarchical Tree Method. *ApJS* , **70**(June), 419.

Hills, J. G., & Day, C. A. 1976. Stellar Collisions in Globular Clusters. *ApJL* , **17**(Feb.), 87.

Hoag, Arthur A. 1950. A peculiar object in Serpens. *AJ* , **55**(Oct.), 170–170.

Hoyle, F., & Lyttleton, R. A. 1939. The effect of interstellar matter on climatic variation. *Proceedings of the Cambridge Philosophical Society*, **35**(3), 405.

Hugoniot, H. 1885. Sur la propagation du mouvement dans les corps et spécialement dans les gaz parfaits,. *Comptes rendus hebdomadaires des séances de lâAcadémie des sciences*, **110**, 794–796.

Inutsuka, Shu-Ichiro. 2002. Reformulation of Smoothed Particle Hydrodynamics with Riemann Solver. *Journal of Computational Physics*, **179**(1), 238–267.

Iwasaki, Kazunari, & Inutsuka, Shu-Ichiro. 2011. Smoothed particle magnetohydrodynamics with a Riemann solver and the method of characteristics. *MNRAS* , **418**(3), 1668–1688.

Kajishima, Takeo, & Taira, Kunihiko. 2017. *Numerical Simulation of Turbulent Flows*. Cham: Springer International Publishing. Pages 207–235.

Keppens, R., Meliani, Z., van Marle, A. J., Delmont, P., Vlasis, A., & van der Holst, B. 2012. Parallel, grid-adaptive approaches for relativistic hydro and magnetohydrodynamics. *Journal of Computational Physics*, **231**(3), 718–744.

Klingenberg, Christian, Schmidt, Wolfram, & Waagan, Knut. 2007. Numerical comparison of Riemann solvers for astrophysical hydrodynamics. *Journal of Computational Physics*, **227**(1), 12–35.

Kuijken, K., & Dubinski, J. 1995. Nearly Self-Consistent Disc / Bulge / Halo Models for Galaxies. *MNRAS* , **277**(Dec.), 1341.

Kuijken, Konrad, & Dubinski, John. 2011 (Sept.). *GalactICS: Galaxy Model Building Package*. Astrophysics Source Code Library, record ascl:1109.011.

Lada, Charles J., & Lada, Elizabeth A. 2003. Embedded Clusters in Molecular Clouds. *ARA&A* , **41**(Jan.), 57–115.

Landau, Lev Davidovich, & Lifshitz, E. M. 1959. *Fluid mechanics*.

Lombardi, James, & Sawyer Warren, Jessica. 2014 (Dec.). *MMAS: Make Me A Star*. Astrophysics Source Code Library, record ascl:1412.010.

Lombardi, James C., Sills, Alison, Rasio, Frederic A., & Shapiro, Stuart L. 1999. Tests of Spurious Transport in Smoothed Particle Hydrodynamics. *Journal of Computational Physics*, **152**(2), 687–735.

Lombardi, James C. Jr., Rasio, Frederic A., & Shapiro, Stuart L. 1996. Collisions of Main-Sequence Stars and the Formation of Blue Stragglers in Globular Clusters.

ApJ , **468**(Sept.), 797.

Longair, Malcolm S. 2011. *High Energy Astrophysics*.

Lützgendorf, N., Helm, E. van der, Pelupessy, F. I., & Portegies Zwart, S. 2016. Stellar winds near massive black holes - the case of the S-stars. *MNRAS* , **456**(4), 3645–3654.

MacNeice, Peter, Olson, Kevin M., Mobarry, Clark, de Fainchtein, Rosalinda, & Packer, Charles. 2000. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, **126**(3), 330 – 354.

Meliani, Zakaria, Keppens, Rony, Casse, Fabien, & Giannios, Dimitrios. 2007. AMRVAC and relativistic hydrodynamic simulations for gamma-ray burst afterglow phases. *MNRAS* , **376**(3), 1189–1200.

Mellema, G., Eulderink, F., & Icke, V. 1991. Hydrodynamical models of aspherical planetary nebulae. *A&A* , **252**(Dec.), 718–732.

Monaghan, J. J. 1997. SPH and Riemann Solvers. *Journal of Computational Physics*, **136**(2), 298–307.

Monaghan, J.J, & Gingold, R.A. 1983. Shock simulation by the particle method SPH. *Journal of Computational Physics*, **52**(2), 374 – 389.

Murante, G., Borgani, S., Brunino, R., & Cha, S. H. 2011. Hydrodynamic simulations with the Godunov smoothed particle hydrodynamics. *MNRAS* , **417**(1), 136–153.

Pakmor, Rüdiger, Springel, Volker, Bauer, Andreas, Mocz, Philip, Munoz, Diego J., Ohlmann, Sebastian T., Schaal, Kevin, & Zhu, Chenchong. 2016. Improving the convergence properties of the moving-mesh code AREPO. *MNRAS* , **455**(1), 1134–1143.

Pelupessy, F. I., & Portegies Zwart, S. 2012. The evolution of embedded star clusters. *MNRAS* , **420**(2), 1503–1517.

Pelupessy, F. I., van der Werf, P. P., & Icke, V. 2004. Periodic bursts of star formation in irregular galaxies. *A&A* , **422**(July), 55–64.

Pelupessy, F. I., van Elteren, A., de Vries, N., McMillan, S. L. W., Drost, N., & Portegies Zwart, S. F. 2013. The Astrophysical Multipurpose Software Environment. *A&A* , **557**(Sept.), A84.

Plummer, H. C. 1911. On the problem of distribution in globular star clusters. *MNRAS* , **71**(Mar.), 460–470.

Portegies Zwart, S. F., & van den Heuvel, E. P. J. 2016. Was the nineteenth century giant eruption of Eta Carinae a merger event in a triple system? *MNRAS* , **456**(4), 3401–3412.

Porth, O., Xia, C., Hendrix, T., Moschou, S. P., & Keppens, R. 2014. MPI-AMRVAC for Solar and Astrophysics. *ApJS* , **214**(1), 4.

Price, Daniel J. 2008. Modelling discontinuities and Kelvin Helmholtz instabilities in SPH. *Journal of Computational Physics*, **227**(24), 10040–10057.

Price, Daniel J., Wurster, James, Tricco, Terrence S., Nixon, Chris, Toupin, Stéven, Pettitt, Alex, Chan, Conrad, Mentiplay, Daniel, Laibe, Guillaume, Glover, Simon, Dobbs, Clare, Nealon, Rebecca, Liptai, David, Worpel, Hauke, Bonnerot, Clément, Dipierro, Giovanni, Ballabio, Giulia, Ragusa, Enrico, Federrath, Christoph, Iaconi, Roberto, Reichardt, Thomas, Forgan, Duncan, Hutchison, Mark, Constantino, Thomas, Ayliffe, Ben, Hirsh, Kieran, & Lodato, Giuseppe. 2018. Phantom: A

Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics. *Proc. Astron. Soc. Aust.* , **35**(Sept.), e031.

Rankine, W. J. M. 1870. On the Thermodynamic Theory of Waves of Finite Longitudinal Disturbance. *Phil. Trans. R. Soc. Lond.*, **160**(Jan.), 277–288.

Read, J. I., Hayfield, T., & Agertz, O. 2010. Resolving mixing in smoothed particle hydrodynamics. *MNRAS* , **405**(3), 1513–1530.

Regev, O., Umurhan, O.M., & Yecko, P. 2016. *Modern Fluid Dynamics for Physics and Astrophysics*. Springer.

Robertson, Brant E., Kravtsov, Andrey V., Gnedin, Nickolay Y., Abel, Tom, & Rudd, Douglas H. 2010. Computational Eulerian hydrodynamics and Galilean invariance. *MNRAS* , **401**(4), 2463–2476.

Safronov, V. S. 1960. On the gravitational instability in flattened systems with axial symmetry and non-uniform rotation. *Annales d'Astrophysique*, **23**(Feb.), 979.

Saitoh, Takayuki R., & Makino, Junichiro. 2013. A Density-independent Formulation of Smoothed Particle Hydrodynamics. *ApJ* , **768**(1), 44.

Saitoh, Takayuki R., & Makino, Junichiro. 2016. Santa Barbara Cluster Comparison Test with DISPH. *ApJ* , **823**(2), 144.

Shore, Steven N. 2007. *Astrophysical Hydrodynamics: An Introduction*.

Sills, Alison, & Lombardi, James C. Jr. 1997. The Importance of Realistic Starting Models for Hydrodynamic Simulations of Stellar Collisions. *ApJL* , **484**(1), L51–L54.

Sormani, Mattia C., Treß, Robin G., Klessen, Ralf S., & Glover, Simon C. O. 2017. A simple method to convert sink particles into stars. *MNRAS* , **466**(1), 407–412.

Springel, Volker. 2000 (Mar.). *GADGET-2: A Code for Cosmological Simulations of Structure Formation*. Astrophysics Source Code Library, record ascl:0003.001.

Springel, Volker. 2005. The cosmological simulation code GADGET-2. *MNRAS* , **364**(4), 1105–1134.

Springel, Volker. 2010. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *MNRAS* , **401**(2), 791–851.

Stone, James M., Gardiner, Thomas A., Teuben, Peter, Hawley, John F., & Simon, Jacob B. 2008. Athena: A New Code for Astrophysical MHD. *ApJS* , **178**(1), 137–177.

Stone, James M., Gardiner, Thomas A., Teuben, Peter, Hawley, John F., & Simon, Jacob B. 2010 (Oct.). *Athena: Grid-based code for astrophysical magnetohydrodynamics (MHD)*. Astrophysics Source Code Library, record ascl:1010.014.

Tasker, Elizabeth J., Brunino, Riccardo, Mitchell, Nigel L., Michielsen, Dolf, Hopton, Stephen, Pearce, Frazer R., Bryan, Greg L., & Theuns, Tom. 2008. A test suite for quantitative comparison of hydrodynamic codes in astrophysics. *MNRAS* , **390**(3), 1267–1281.

Teyssier, R. 2002. Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. *A&A* , **385**(Apr.), 337–364.

Teyssier, Romain. 2010 (Nov.). *RAMSES: A new N-body and hydrodynamical code*. Astrophysics Source Code Library, record ascl:1011.007.

Thacker, R. J., Tittley, E. R., Pearce, F. R., Couchman, H. M. P., & Thomas, P. A. 2000. Smoothed Particle Hydrodynamics in cosmology: a comparative study of

implementations. *MNRAS* , **319**(2), 619–648.

Theuns, Tom, Boffin, Henri M. J., & Jorissen, Alain. 1996. Wind accretion in binary stars - II. Accretion rates. *MNRAS* , **280**(4), 1264–1276.

Toomre, A. 1964. On the gravitational stability of a disk of stars. *ApJ* , **139**(May), 1217–1238.

Toro, E. F. 1997. *Riemann solvers and numerical methods for fluid dynamics : a practical introduction*. Berlin, New York: Springer.

van der Holst, Bar, Keppens, Rony, Meliani, Zakaria, Porth, Oliver, van Marle, Allard Jan, Delmont, Peter, & Xia, Chun. 2012 (Aug.). *MPI-AMRVAC: MPI-Adaptive Mesh Refinement-Versatile Advection Code*. Astrophysics Source Code Library, record ascl:1208.014.

van Leer, B. 1979. Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method. *Journal of Computational Physics*, **32**(1), 101–136.

Von Neumann, J., & Richtmyer, R. D. 1950. A Method for the Numerical Calculation of Hydrodynamic Shocks. *Journal of Applied Physics*, **21**(3), 232–237.

White, Simon D. M. 1994. Formation and Evolution of Galaxies: Les Houches Lectures. *arXiv e-prints*, Oct., astro–ph/9410043.

Zemp, Marcel. 2014 (July). *Halogen: Multimass spherical structure models for N-body simulations*. Astrophysics Source Code Library, record ascl:1407.020.

Zemp, Marcel, Moore, Ben, Stadel, Joachim, Carollo, C. Marcella, & Madau, Piero. 2008. Multimass spherical structure models for N-body simulations. *MNRAS* , **386**(3), 1543–1556.

Chapter 6

Radiative Transfer

> Nel mezzo del cammin di nostra vita, mi ritrovai per una selva oscura, ché la diritta via era smarrita.
> When halfway through our life's journey, I found myself in a gloomy wood, because the path which led straight had been lost.
>
> Dante Alighieri

Our human perception of the universe is dominated by electromagnetic radiation. Radiation forms the basis for essentially all astronomical research, including all of the great images reported by observational astronomers. Radiative transport is the foundation for understanding how radiation interacts with matter. Solving the differential equations describing the interaction between radiation and matter is no trivial task. In AMUSE, radiative transfer is addressed numerically using two quite distinct techniques: by photon propagation through a grid, and by Monte Carlo ray-tracing. These approaches are complementary. In AMUSE, these codes are typically used to determine the temperature and ionization state of the gas. Radiative transfer is expensive in terms of computer time, particularly if the time scales are governed by the hydrodynamics or the gravitational dynamics of the environment.

6.1 In a Nutshell

Radiative transfer deals with the way electromagnetic radiation travels through space and interacts with matter. Radiation is mediated by photons, which have energy, direction, and polarization state. Because the number of photons is generally very large, we normally work with statistical quantities, such as mean intensity and flux, that describe the bulk properties of the radiation field. These quantities may be frequency-dependent, or integrated over some portion of the electromagnetic spectrum. Radiation travels freely through vacuum without loss of energy, but when a ray interacts with matter, energy can be injected or removed.

Radiative flux. The radiative flux F is defined as the total amount of energy passing through a unit of area of some surface per unit of time. Flux is measured in units of erg s^{-1} cm^{-2}, or the SI equivalent. If (as is usually the case) the energy is distributed