Massive stars shaping the ISM
Simulations and application to the Orion-Eridanus Superbubble

Katharina Maria Fierlinger

Garching 2014

Massive stars shaping the ISM Simulations and application to the Orion-Eridanus Superbubble

Katharina Maria Fierlinger

Dissertation
an der Fakultät für Physik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Katharina Maria Fierlinger
aus Graz, Österreich

München, den 13. Oktober 2014

Erstgutachter: Prof. Dr. Andreas Burkert
Zweitgutachter: PD Dr. Roland Diehl
Tag der mündlichen Prüfung: 30. Oktober 2014

Contents

List of Figures

List of Tables

Listings

Zusammenfassung

Diese Arbeit befasst sich mit dem Einfluss von Sternen, deren Masse acht Sonnenmassen übersteigt, auf das Interstellare Medium in ihrer Umgebung. Solche massereiche Sterne beenden ihr Dasein mit einer Supernovaexplosion und verlieren im Laufe ihrer – verglichen mit massearmen Sternen – raschen Entwicklung einen großen Teil ihrer Masse über ihre starken Sternwinde. Beispielsweise gibt ein Stern mit 60 Sonnenmassen Anfangsmasse mehr als die doppelte Supernovaenergie über die kinetische Energie seiner Winde in seine Umgebung ab.
Sterne entstehen in Regionen mit kaltem, dichtem Gas, den sogenannten Molekülwolken. Beobachtungen zeigen, dass diese Gaswolken turbulent sind. Es ist allerdings noch ungeklärt, woher die beobachtete Turbulenz im Interstellaren Medium ihre Energie bezieht. Die Energieabgabe von massereichen Sternen ist – neben großskaligen gravitativen Instabilitäten in der Scheibe der Milchstraße – eine der möglichen Erklärungen. Beobachtungen erlauben Rückschlüsse auf die eingebrachte Energiemenge und die Längenskalen des Energie liefernden Prozesses. Daher ist es relevant, zu bestimmen, wie viel kinetische Energie ein massereicher Stern in der ihn umgebenden Molekülwolke deponieren kann.
Der Schwerpunkt dieser Arbeit sind hydrodynamische Simulationen, die diese Energieeffizienz testen. Dazu wurden aktuelle Sternentwicklungsmodelle in die frei zugänglichen Eulerschen Gittercodes PLUTO und RAMSES eingebaut. Die Simulationen verwenden das von Eva Ntormousi erstellte Modul für die Berechnung der Heiz- und Kühlprozesse eines Multiphasenmediums.
Die Modellrechnungen führten zur Erkenntnis, dass in jener Phase der Simulation, in der die räumliche Auflösung der Modellrechnung die Eneergieeffizienz stark beeinflusst, der größte Energieverlust durch Strahlung an jener Stelle auftritt, an der das vom Stern ausgestoßene Material auf das aufgesammelte Umgebungsgas trifft. An dieser Kontaktfläche treten Mischungsprozesse auf, welche die Energieverluste steigern. Somit können unsere Simulationen in Kombination mit einer Abschätzung der Effizienz und Skalenlänge dieser Mischprozesse eine Aussage treffen, wie viel Energie massereiche Sterne zur Aufrechterhaltung der Turbulenz beitragen können. Für diese Abschätzung der Mischprozesse liefert die Literatur auf Beobachtungen und numerischen Simulationen basierende Richtwerte.
Als Anwendungsbeispiel wird in dieser Arbeit die Orion-Eridanus Region diskutiert. In dieser Region wird das radioaktive Isotop ^{26}Al beobachtet. Dieses Isotop wird vorrangig in massereichen Sternen gebildet. Es kann daher als Indikator für von Sternen ausgestoßene Materie verwendet werden. Interessanterweise zeigen die Beobachtungen dieser Region nur in einem Teil des Gebiets mit Röntgenemission ein ^{26}Al Signal. Unsere RAMSES Modelle berücksichtigen ^{26}Al und können daher auf Gebiete mit (fehlenden) Korrelationen zwischen Röntgenemission und ^{26}Al Signal durchsucht werden.

Chapter 1

Motivation

This work simulates the effects of massive stars on their surroundings. Groups of massive stars, so-called "OB associations", form in molecular clouds. A nice, illustrative study of massive stars shaping their environment is the Milky Way Project (Kendrew et al., 2012; Simpson et al., 2012; Beaumont et al., 2014, http://www.milkywayproject.org), where citizens are asked to help scientists identifying bubbles in observational data from the Spitzer Space Telescope. While we know, that massive stars have a dramatic effect on their direct surroundings, since they burn fast and hot and eject much of their material, it is less clear to which extent they are involved in driving turbulence.

This leads us to the question: "What is turbulence?". We can loosely describe turbulence as a highly irregular flow in space and time. Energy is injected at large scales and cascades down to smaller scales, where it is dissipated. This can also be seen in everyday life. For example, stirring a glass of caffè latte will mix coffee and milk, nicely illustrating turbulence at work. Of course, as a physicist, one has to analyze the results of this little experiment. And there even exist computer simulations of this process: e.g. Volker Springel published a simulation called "stirring a coffee mug" which makes use of his AREPO code (snapshots can be found e.g. in Fig. 39 in Springel, 2010). In this experiment the large scale motion of the spoon causes many small whirls. When we analyze our data, turbulence is usually visualized with a Kolmogorov energy spectrum showing the energy contained in coffee and milk blobs (these elements will be called "eddies" later on) of different sizes. If turbulence has developed, a characteristic slope of $-5/3$ is observed in this spectrum.

Technically, the onset of turbulence can also be parametrized via the Reynolds number (relating velocity, scale length and viscosity) and the Prandtl number (relating momentum diffusivity and thermal diffusivity), which are larger than unity in turbulent flows.

So, how does the process in the caffè latte relate to astrophysical fluid flows in the interstellar medium (ISM) and this thesis? It is obvious, that a spoon created the motions in the caffè latte experiment. However, in molecular clouds the origin of the energy injection, which creates and sustains turbulence, is still a matter of debate. Basically, observations of the density and velocity structure of the ISM can be compared to simulations. This gives a hint on the amount of injected energy and the energy injection scales.

Possible processes creating turbulence in the ISM are accretion of gas of extragalactic origin, magneto-rotational instability in the galactic disk, convergent flows of atomic gas triggered by spiral density waves, supernovae (SNe), expanding H II regions, or stellar outflows. These processes differ by the length scale on which energy is injected.

Most probably, turbulence is driven by a mixture of all these processes. While the local impact of supernova explosions is obvious, their impact on galactic turbulence remains an open question. In this work, we will thus study, how much of the stellar feedback energy can be converted to kinetic energy of the cold gas in the surroundings of the star. We call this "feedback energy efficiency (ϵ)". Another similarity between the caffè latte and the processes studied in this thesis is, that after stirring a caffè latte, milk and coffee become well mixed. In this work, we are also interested in the distribution of heavy elements. The reader might be aware of the fact, that most (about 90% of the mass) of the chemical elements in a human being were not created in the Big Bang. Thus, the spreading of heavy elements in the cosmos (sometimes called "chemodynamics" or "the cosmic matter cycle") is an interesting process of evident importance for mankind. Our work also touches this question. For this work, the spatial distribution of the radioactive isotope ^{26}Al, which is created in massive stars, is of interest. Due to its radioactive decay, it can serve as a tracer to identify matter that was newly ejected from massive stars. ^{26}Al can be used to study the spatial distribution as well as the velocities of these ejecta.
In the next chapter we will discuss the Orion-Eridanus region, which is a prototypical example of a region with interactions between young, massive stars and star-forming molecular clouds. Fortunately, a ^{26}Al signal has been observed in this region and – due to a successful INTEGRAL proposal of R. Diehl – more observational data of ^{26}Al in this region will become available in the near future. The spread of ^{26}Al might also help to shed light on the question, if the Orion-Eridanus Superbubble (OES) is a monolithic bubble of possibly[1] peculiar shape (Reynolds and Ogden, 1979; Burrows et al., 1993; Diehl et al., 2004; Pon et al., 2014a) or a superposition of individual superbubbles (Boumis et al., 2001; Ryu et al., 2008; Jo et al., 2011) created by the Orion OB I associations. Presently, the available observational data for the OES (see Sect. 2.4) can be interpreted in both ways and this question is still under debate. In this work, we will use the term "OES" for both interpretations of the data from the Orion-Eridanus region.

[1] Pon et al. (2014a) fit a symmetric model, the other authors assume a less regular shape.

Chapter 2

Background: massive stars and their surroundings

Our current understanding of astrophysics sees the universe as a constantly evolving very dynamic system. In computational astrophysics, when we try to simulate the cosmos, we are faced with the problem that processes on very different length scales seem to be coupled, which makes a self consistent treatment of a subsystem challenging. An example for such a coupling between small scales and large scales is chemical enrichment, where heavy elements are produced in stars and distributed throughout galaxies. Vice versa, also large scales can influence small scales, for example via turbulence, which cascades energy from large scales down to the smallest scales where it is dissipated. Another interesting aspect of this system is that many astrophysical processes appear to be cyclic. For instance, the processes studied in this work are often subsumed under the concept of the matter cycle of stars. In this cycle, stars form in gas clouds, start nucleosynthesis, produce heavy elements and finally, when they have consumed their fuel for nucleosynthesis, give a large fraction of their gas back into the interstellar medium (ISM), possibly triggering the birth of a new generation of stars. From this plethora of interesting processes we will now pick one – namely the interaction of massive stars with their environments – and look at it in detail.
The benefit of gaining insight on the influences of stellar feedback onto the surrounding ISM from small-scale high-resolution studies is twofold: On the one hand we can simulate regions small enough to treat them in high-resolution and compare our results to observations like data from the Orion-Eridanus Superbubble (OES) and on the other hand we can try to draw conclusions which will hopefully be useful for investigations of processes on larger length scales. More precisely, simulations of galaxies have a hard time resolving stellar feedback. This problem is usually assessed with sub-grid models, and such models can be improved with our findings.
In this section we will discuss some key agents in the problem of stellar feedback energy efficiency and present the terminology[1] – for example "ISM" or "superbubble", which we already used in the preceding paragraphs – before we delve into the simulations in the next chapters. We will start with the physics and the composition of the ISM which encompasses – as its name already indicates – the gas and dust between stars (Sect. 2.1). In this context we will also introduce Giant Molecular Clouds (GMCs, Sect. 2.5) and discuss observational evidence of the ISM (Sect. 2.4). Since we are most interested in the Orion-Eridanus region, we will briefly introduce it and focus on the observational evidence from this region. Obviously the other important topic are massive

[1] To make the text a bit shorter and easier to read, some of the terminology (highlighted in blue in the electronic version) can also be found in the glossary.

stars, which will be discussed in Sect. 2.6 including their occurrence in the Orion-Eridanus region. Since the dynamics of the ISM involve the exchange of mass and energy between the constituents of the ISM we will briefly mention cooling and heating processes in the ISM in Sect. 2.2.6. The mixing of newly produced elements into the surrounding GMC gas will be discussed in the rest of Sect. 2.2.

2.1 Theories of the interstellar medium (ISM)

Our current picture of the interstellar medium (ISM) is that of a complex dynamic mixture of several gas phases (Cox, 2005; de Avillez and Breitschwerdt, 2005). After reviewing the classic models of the ISM (Field et al., 1969; McKee and Ostriker, 1977), which can be assumed to be a zero order approximation, we will proceed to the present day dynamic picture of the ISM.

2.1.1 Classic equilibrium models for the ISM

This class of models of the ISM (Field et al., 1969; McKee and Ostriker, 1977) postulates the existence of several gas phases in pressure equilibrium. In this context a "phase of the ISM" is a stable combination of number density and temperature (n, T) where the heating rate (Γ) equals the cooling rate (Λ, see also Sect. 2.2.6). An important tool in this context is Field's stability criterion (Field, 1965), which states that a gas phase is stable, if the slope of the cooling-heating equilibrium curve ($\frac{\mathrm{d}\log p/k_\mathrm{B}}{\mathrm{d}\log n}$, see Fig. 2.1) in the p/k_B, n diagram is positive. A point in this space is called stable, if a perturbation in density or temperature leads to a change of the cooling-heating function, which counteracts this perturbation.

The classic model of Field et al. (1969) applies this concept to two phases: to a cold phase with a temperature of 100 K and a warm phase with a temperature of 10 000 K. The motivation for this model was the observed stability of cold H I clouds. This finding can be explained by assuming that H I clouds are immersed in a hot, rarefied medium, which is heated by cosmic rays and which is in pressure equilibrium with the H I regions. In contrast to this model, which emphasizes the impact of cosmic rays, the equilibrium model of McKee and Ostriker (1977) identifies supernova explosions as the key agent. These supernovae lead to a third thermal phase: a dilute hot medium. The general picture presented in the McKee and Ostriker (1977) model consists of three components in rough pressure equilibrium. This model predicts that 70% to 80% of the volume are filled with the hot inter-cloud medium (HIM, $T \sim 5 \times 10^5$ K, $n \sim 0.003$ particles cm^{-3}) produced by supernovae. The cold neutral medium (CNM, $T \sim 80$ K, $n \sim 40$ particles cm^{-3}) forms small dense spheres with average diameters of 3.2 pc, which are embedded in the hot medium and occupy about 2% to 4% of the volume. The remaining $\sim$ 20% of the volume are filled with the coronae ($T \sim 8\,000$ K, $n \sim 0.25$ to 0.37 particles cm^{-3}) of the cold clouds. The model expects two layers in these coronae: an inner layer of warm neutral medium (WNM) and an outer ionized layer, containing the so-called warm ionized medium (WIM).

An interesting aspect for our study – which focuses on massive stars in molecular clouds – is the role of molecular clouds in this model. The cloud masses in the McKee and Ostriker (1977) model are chosen to stay below 10^4 $M_\odot$ to avoid self gravity of the clouds. McKee (1990) states that molecular clouds are self-gravitating and thus not in pressure equilibrium with the phases of the ISM. Consequently, molecular clouds do not form a fourth component of the model.

Nowadays, the three phase model is considered as a zero-order approximation only and numerical

simulations as well as observations suggest a more dynamic, turbulent ISM. Also the important role of conduction in the McKee and Ostriker (1977) model has been criticized. Moreover, the concept of spherical clouds does not fit well to the observed filamentary structure of the ISM. Therefore, we will move on and discuss the concept of a dynamic ISM.

2.1.2 Dynamic multi-phase ISM

Cox (2005) suggests that dynamics in the ISM have a larger effect on the constituents of the ISM than the thermal instability, arguing that the time to adjust to the equilibrium is rather long (Sect. 2.2.6). Also numerical simulations (e.g. Korpi et al., 1999; de Avillez and Breitschwerdt, 2005; Joung and Mac Low, 2006; Hennebelle and Audit, 2007; Koyama and Ostriker, 2009; de Avillez and Breitschwerdt, 2012; Hill et al., 2012; Gent et al., 2013) show a more dynamic picture of the ISM: Generally, these models do not find an ISM becoming saturated by SN impacts. Several studies find volume filling factors of the hot gas much lower than 70% (Joung and Mac Low, 2006; Hill et al., 2012; Hill et al., 2012; de Avillez and Breitschwerdt, 2012). Recently, de Avillez and Breitschwerdt (2012) also showed that the assumption of collisional ionization equilibrium (CIE) below 10^6 K is problematic, and that non-equilibrium models can find O VI emission at lower temperatures than previously expected (see Sect. 2.4.4). All models observe a dynamic medium with large variations in pressure. Turbulence also seems to lead to a tightly interwoven CNM and WNM with a continuously varying density and temperature structure. Some authors (e.g. Hennebelle and Audit, 2007) claim that the CNM and WNM are locally in pressure equilibrium in their simulations. To summarize, whereas also simulations that take turbulent motions of the ISM into account, find much of the gas mass close to the cooling-heating equilibrium, the gas phases observed in simulations of a turbulent ISM differ from the two phases formed by thermal instability. In a dymanic ISM, pressure gradients lead to gas phases in the unstable regime in Fig. 2.1, where the thermal instability is slowly working on restoring stable phases.

2.2 Mass and energy exchange

In the following, some processes, which lead to an exchange of mass and energy between gas phases or to a removal of energy from the system, are briefly discussed. The motivation for the brief excursion into radiative cooling (Sect. 2.2.6) is that a large fraction of the feedback energy of massive stars (discussed in Sect. 2.7) in GMCs is removed from this environment via radiative cooling processes (see e.g. Tab. 6.2). The importance of mixing of material of different gas phases (treated in Sect. 2.2.2 to 2.2.5) for our work is twofold: On the one hand, obviously, the spread of our trace element ^{26}Al and all other newly produced heavy elements will be influenced. As a consequence, also the predicted ^{26}Al velocities are affected, as motions in the swept-up GMC material are substantially lower than the velocities observed inside the superbubble. On the other hand, mixing of gas phases can enhance radiative losses and change the feedback energy efficiency. More generally speaking, mixing of stellar ejecta with the ambient medium is important for models of the the cosmic matter cycle. Due to the large range of scales, a hydrodynamical treatment of these mixing processes is beyond reach in most simulations. Therefore many chemical evolution models assume an immediate mixing of the SN ejecta in the walls of superbubbles. However, it is unclear if this is realistic. As pointed out by e.g. Tenorio-Tagle (1996) stellar winds and supernova explosions lead to a two shock structure with a contact discontinuity (CD) separating the well

mixed hot material inside the bubble from the swept up, compressed, heated, radiatively cooling (and thus cold) ambient medium.
The efficiency of mixing across the CD still remains an open question. Presently the mechanism of mixing via droplets produced in the SN receives most attention (Stasińska et al., 2007; Gounelle et al., 2009; Gounelle and Meynet, 2012; Boss and Keiser, 2012; Pan et al., 2012).
In the literature the stability of the CD in wind-blown bubbles is debated: Tenorio-Tagle (1996) reports Rayleigh-Taylor instabilities followed by Kelvin-Helmholtz instabilities due to the collision of SN ejecta with the wind material in his 2D simulations, whereas Pan et al. (2012) report a stable CD for isotropic ejecta. However, Pan et al. (2012) note that the omnipresent turbulence in the ISM will lead to instabilities, which in turn enhance the mixing across the CD by increasing the CD surface.
In our brief discussion of processes capable of degrading the CD, we will start from kinetic gas theory, where such degradations are caused by particle motion smearing out a gradient. We can look at different manifestations of this diffusion process. To do so, we consider two distinct gas phases in pressure equilibrium that are separated by a CD. After a few words on the mean free path (λ), we will estimate in the rest frame of the CD how many hot particles will flow into the cold gas and vice versa. This ultimately leads to heat conduction down a temperature gradient (Sect. 2.2.2). Another manifestation of such mixing processes is molecular diffusion (Sect. 2.2.3). In this case the CD separates two different gas species and diffusion will try to level a concentration gradient. Taking a step back from the microscopic level to the macroscopic level, gas blobs can mix via turbulenceturbulent diffusion (Sect. 2.2.4). And last but not least one can rely on ambipolar diffusion caused by magnetic fields (Sect. 2.2.5).

2.2.1 Mean free path

A crucial length scale for diffusive processes is the mean free path (λ), which denotes the average distance a particle travels between two scatterings. Processes at the scales of the mean free path and below have to be modeled taking plasma physics into account. As will be discussed in Sect. 3, our hydrodynamic simulations are based on the fluid approach, which assumes that λ is much smaller than a cell size. In other words, the underlying assumptions of our simulation method imply a maximal "meaningful" resolution, which is connected to λ. The mean free path

$$\lambda = \frac{1}{\sigma n} \tag{2.1}$$

for elastic scattering of neutral hydrogen with an elastic collision cross section $\sigma_{\mathrm{H-H}}$ of 5.7×10^{-15} cm^2 (Godard et al., 2009) becomes larger than a cell size of e.g. 0.01 pc (turbulent diffusion length scale estimate of Gounelle et al., 2009) if the density falls below 10^{-26} g cm^{-3}, which corresponds to a number density of 0.006 cm^{-3}. With the mean molecular velocity

$$v_{\mathrm{rms}}^2 = \frac{3k_{\mathrm{B}}T}{m_{\mathrm{H}}} = \frac{3RT}{\mu_{\mathrm{mol}}} \tag{2.2}$$

the average time between collisions is

$$\tau = \sqrt{\frac{m_{\mathrm{H}}}{3k_{\mathrm{B}}T\sigma^2 n^2}} \quad .$$

In ionized gases the scattering cross section is the area in which the electrostatic energy becomes comparable to the relative kinetic energy of the two charged particles. The electron mean free path

$$\lambda_e = \frac{0.290\,(k_\mathrm{B}T_e)^2}{n_e e^4 \ln\Lambda}$$

(Eq. 5-26 Spitzer, 1956; Shu, 1992, Eq. 1.5) with the thermal velocity of the electrons

$$v_{T_e}^2 = \frac{k_\mathrm{B}T_e}{m_e}$$

and the Coulomb logarithm

$$\Lambda = \frac{3}{2e^3}\sqrt{\frac{k^3T_e^3}{\pi n_e}}$$

is larger than 0.01 pc for temperatures above $10^{5.36}$ K for densities below 10^{-26} g cm^{-3}.

2.2.2 Evaporation due to thermal conduction

In the PLUTO code (Mignone et al., 2007, see also Sect. 5.1.1 of this work), thermal evaporation is facilitated with an additional divergence term for heat conduction in the energy equation:

$$\frac{\partial E}{\partial t} + \vec{\nabla}\cdot[(E+p)\,\vec{v}] = -\vec{\nabla}\cdot\vec{F}_\mathrm{c} \quad .$$

Due to the inverse dependence on the particle mass (evident from the mean molecular velocity, Eq. 2.2), conduction is electron dominated. If the scale length of the temperature gradient

$$l_\mathrm{T} \equiv \frac{T_\mathrm{e}}{|\nabla T_\mathrm{e}|}$$

is much larger than the mean free path of the electrons λ_e, the heat flux conducting heat down the electron temperature gradient in a plasma is given by

$$F_\mathrm{c} = -\kappa\nabla T_\mathrm{e} \quad .$$

We use a thermal conduction coefficient for a hydrogen plasma of $\kappa = 5.6\times10^{-7}T^{5/2}$ erg s^{-1} cm^{-1} K^{-1} (Spitzer, 1962) within the PLUTO code (Mignone et al., 2007). The relaxation time

$$t_\mathrm{relax} = \frac{nc_v}{\kappa}(\Delta x)^2 = \frac{(\Delta x)^2}{D} = \frac{3}{v_\mathrm{rms}\lambda}(\Delta x)^2$$

describes how fast heat conduction in the classic heat flux is. For a gas with a density of 10^{-26} g cm^{-3} and a temperature of 10^6 K on the scales of $\Delta x = 0.01$ pc the relaxation time is $\sim 1.8\times10^7$ years. For steep temperature gradients with scales shorter than the mean free path the code switches to the saturated heat flux, estimated to be

$$F_\mathrm{sat} = 5\phi\rho c_\mathrm{s,iso}^3\,[\mathrm{erg\,s^{-1}\,cm^{-2}}] \quad ,$$

with $\phi = 0.3$ (Balbus and McKee, 1982) and $c^2_{\rm s,iso} = k_{\rm B}T/m$, because in this regime the classic heat flux equation overestimates conduction. In the case of a CD we expect such a very steep temperature gradient. For a hydrogen gas with $\rho = 10^{-26}$ g cm^{-3} and $T = 10^6$ K this flux is 1.1×10^{-20} erg s^{-1} cm^{-2}, which can be compared to the loss via radiative cooling $\Lambda \sim 10^{-22} n^2$ erg s^{-1} cm^3= 10^{-26} erg s^{-1} cm^{-3} of a slab with a width of 10^6 cm, which is way below our maximal resolution. The heat flux is thus not an important agent near the CD in this problem.
In our simulations thermal conduction saturated near the CD. The kinetic feedback energy efficiency is only slightly lowered, if thermal conduction is taken into account (Tab. 6.2, Fig. 6.9), which is in agreement with the aforementioned order of magnitude estimates.
A more important aspect is the change in particle density, which affects the radiative cooling losses. Tenorio-Tagle (1996) find 10% of shell mass mixed into the cavity due to thermal evaporation. The efficiency of mixing of particles of different temperature is discussed in the section below.

2.2.3 Molecular diffusion

Molecular diffusion levels concentration gradients. If a diaphragm between two gaseous species in pressure equilibrium is removed, random movement of all gas particles starts to mix the two species. This process is described with the diffusion equation

$$\frac{\partial n}{\partial t} = D \frac{\partial^2 n}{\partial x^2} \quad ,$$

with the solution

$$n(x,t) = \frac{N}{\sqrt{4\pi D t}} \exp -x^2/4Dt \quad .$$

The diffusion coefficient $D \sim \bar{v}\lambda/3$, with the thermal velocity $\bar{v}$, is the same as for heat conduction. The diffusion length

$$\Delta x = \sqrt{2Dt} \sim \sqrt{v_{\rm rms}\lambda t}$$

is a measure over which physical scales mixing has occurred. This relation can also be used to estimate the timescale of this process:

$$t_{\rm d} \sim \frac{(\Delta x)^2}{v_{\rm rms}\lambda} \tag{2.3}$$

with the mean free path λ (Eq. 2.1) and the rms-velocity $v_{\rm rms}$ (Eq. 2.2) .
Equation 2.3 shows that molecular diffusion mixes chemical species efficiently in the hot dilute gas inside the bubble: In a gas with $n = 10^{-2}$ cm^{-3}, $T = 10^7$ K and $\mu \sim 1$ g mol^{-1} we find $v_{\rm rms} \sim 500$ km s^{-1} and a time of ~ 33 years for mixing on the scales of $\Delta x = 0.01$ pc. Diffusion inside the swept up medium is inefficient ($n = 1$ cm^{-3}and $T = 100$ K leads to a time of ~ 1.5 Myr for mixing on the scales of $\Delta x = 0.01$ pc).
All particles within a mean free path from the CD can penetrate into the other gas phase and one sixth of them will have a velocity vector appropriate to do so.[2] For two gas phases with

[2]The number of particles crossing the CD in the time interval t are thus a sixth of the particles within the volume Avt where A is the unit area.

$n = 0.01$ cm^{-3}, $T = 10^6$ K and $n = 1$ cm^{-3}, $T = 100$ K, respectively, the same number of hot and cold particles cross the CD. There is no change in density and thus no change in the mean free path, but there is a change in temperature. The hot particles in the cold medium undergo their first collision with cold particles after $t = \lambda_{\rm cold}/v_{\rm hot} = 0.35$ yr. This means that after 0.35 years a region of a length of 6×10^{-5} pc ($\lambda_{\rm cold}$) has a mean temperature of $T_{\rm hot}/6+5T_{\rm cold}/6 = 1.7\times10^5$ K. To estimate how much thermal energy has been carried into the cold medium we find the number of diffused particles from $\Delta n = A\lambda_{\rm cold}n_{\rm hot}/6 = 2.9\times10^{11}A$ cm^{-2} (with $n_{\rm hot} = 0.01$ cm^{-3}, $\lambda_{\rm cold} = 1.7\times10^{14}$ cm). The energy transfer caused by particle motion is $\dot{E} = \dot{n}k_{\rm B}T = n_{\rm hot}/6v_{\rm hot}k_{\rm B}T_{\rm hot} = 3.6\times10^{-6}$ erg s^{-1} cm^{-2}. With a cooling rate of $\Lambda_{\rm cool} = 10^{-22}n^2$ erg s^{-1} cm^{-3}, the energy flowing through an area A of the CD would be lost in a cell with a number density of 1 cm^{-3} and a volume of $A\times0.01$ pc.

Tenorio-Tagle (1996) reports that 10% of the ambient medium ended up in the bubble via thermal conduction and dense clumplets originating from the ambient medium penetrating the bubble wall. From kinetic gas theory, we would expect that in each collision time a sixth of the density in the first mean free path of the shell is lost into the bubble. In the example given above, the number of particles was conserved both in the cavity and in the shell, but if the density of the shell is enhanced, there will be a net flux of particles into the cavity.

2.2.4 Turbulent diffusion

In this process random and chaotic motions mix eddies of size $l_{\rm turb}$ with the velocity $v_{\rm turb}$. The turbulent velocity fields may be created, for example, by steep gradients, the overstability of radiative shocks (Chevalier and Imamura, 1982), stellar feedback impinging on a clumpy medium or instabilities like the nonlinear thin shell instability (Vishniac, 1994, NTSI). For example, convection can produce eddies and large scale perturbations that are mixed into a different gas phase. Such mixing processes do not necessarily lead to a homogeneous mixture. For the turbulent diffusion in a turbulent ISM, some authors (for a summary see Pan et al., 2012) rather expect an oil-in-water-like process leading to cold clumps immersed in hot zones, whereas other authors assume that the gas phases fully mix (e.g. Gounelle et al., 2009). The diffusion coefficient of turbulent mixing is

$$D_{\rm turb} = v_{\rm turb}l_{\rm turb} \quad .$$

Diffusion rises linearly below the size of turbulent eddies and saturates due to turn-over as soon as the eddy size is reached.

The assumed efficiencies of mixing in a SN shell range from a few percent (Boss and Keiser, 2012, mixing via clumps and RT fingers), over a range from 2% to 70% (Gounelle and Meynet, 2012), to the full range of few percent to full mixing in the study of Pan et al. (2012, clumplets and turbulent diffusion).

The estimates for the eddie size range from $l_{\rm turb} \sim 0.1 - 1$ pc (Stasińska et al., 2007, dispersion of metal-rich droplets in a H II region via molecular diffusion and turbulent mixing) to $l_{\rm turb} \sim 0.01$ pc (Gounelle et al., 2009, highly turbulent mixing process with 100% mixing efficiency and the characteristic length-scale of the thermal instability). Turbulent diffusion is thus likely to act on length scales comparable to the resolution of our simulations.

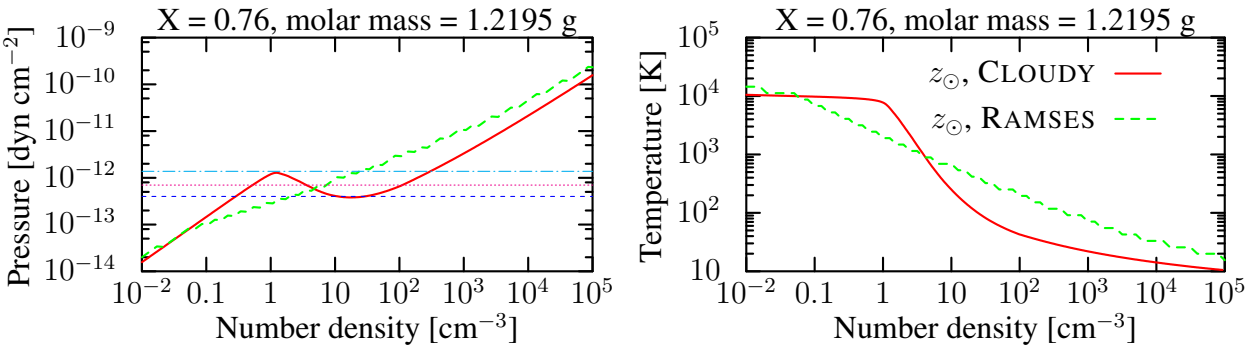


Figure 2.1: Comparison of the cooling–heating equilibrium for solar abundances computed with the RAMSES code (green) to the equilibrium found by the CLOUDY code (red) [data extracted from CLOUDY by Ntormousi & Heitsch]. The absence of a maximum in the RAMSES cooling–heating equilibrium curve (left plot) prevents the existence of two stable ISM phases. In contrast to this, the CLOUDY cooling heating equilibrium curve allows for a multi-phase medium. This is caused by multiple regions with positive slopes for the same pressure in the equilibrium curve. The missing multi-phase problem was fixed artificially by switching off cooling and heating below 100 K in dense regions with a number density larger than 5 particles cm^{-3} and by applying a similar procedure at 10 000 K in less dense regions.

2.2.5 Ambipolar diffusion

Ambipolar diffusion is a process that can remove magnetic fields from molecular clouds: The magnetic fields are tied to the ionized gas component, and this component drifts relative to the cold, neutral component of the gas, which is accelerated by gravity. E.g. Jijina et al. (1999) noted that ambipolar diffusion takes place more rapidly than the simple laminar description predicts. For a dense core with the size r the time scale for ambipolar diffusion is $\tau_{\mathrm{AD}} = \frac{r}{v_{\mathrm{D}}}$ with ion-neutral drift speed v_{D} (Mouschovias, 1987, eq. 81). This can be approximated by

$$\tau_{\mathrm{AD}} \sim 3 \times 10^6\,\mathrm{yr} \left(\frac{n_{\mathrm{H_2}}}{10^4\,\mathrm{cm^{-3}}}\right)^{1.5} \left(\frac{30\,\mu\mathrm{G}}{B}\right)^2 \left(\frac{r}{0.1\,\mathrm{pc}}\right)^2 \quad .$$

For a density of 1 cm^{-3} and a magnetic field strength of 10 μG (Crutcher, 2012) this leads to a time of about three months for 0.01 pc. This process rather acts to separate the gas phases than to mix them.

2.2.6 Cooling and heating processes in the ISM

For the work presented in this thesis, radiative losses are important, since they substantially lower the feedback energy efficiency and thus increase the GMC lifetimes. In a medium with solar metallicity, $\sim$ 100 particles cm^{-3} and a temperature of $\sim$ 100 K typical energy losses via radiative cooling amount to about 90% of the feedback energy (see e.g. Tab. 6.2). A similar energy loss was reported by Thornton et al. (1998).

The default cooling routine in the RAMSES code (see Sect. 5.1.2) uses Sutherland and Dopita (1993) cooling for all elements except H and He, Compton heating from CMB and Compton cooling according to Theuns et al. (1998, Tab. B1) with an amplitude of the radiation spectrum

at the hydrogen Lyman-alpha edge of 5×10^{-21} erg cm^{-2} sr^{-1}. For H and He the amount of (doubly) ionized particles is calculated. Based on the result of this iteration the code calculates ionization cooling for H and He according to Cen (1992, Eq. 12), recombination cooling for H and He according to Cen (1992, Eq. 13), dielectric recombination cooling for He according to Cen (1992, Eq. 14), line cooling for H and He according to Cen (1992, Eq. 15), Bremsstrahlung for H and He according to Cen (1992, Eq. 16) and radiative heating for H and He according to Theuns et al. (1998, Tab. B4). In our simulations solar abundances[3] are assumed.
In our study the existence of two gas phases in pressure equilibrium is desired, because this makes a "static background model" feasible: Our study is easier to analyze (1) if thermal energy of the medium, which is not influenced by the stellar feedback, stays constant, and (2) if no motions arise at the cloud surface, because of a pressure imbalance caused by cooling or heating processes. Since the standard RAMSES cooling–heating curve (Fig. 2.1) has no maximum that would allow for the existence of a two-phase region with a stable cold dense phase ($T = 100$ K, $\rho = 1.66 \times 10^{-22}$ g cm^{-3}) and a stable warm phase ($T = 10^4$ K, $\rho = 1.66 \times 10^{-24}$ g cm^{-3}) these phases are created artificially by switching off cooling and heating below 100 K in regions with a number density larger than 5 particles cm^{-3} and by applying a similar procedure at 10 000 K in less dense regions. In this prescription temperatures below 100 K can only be reached via expansion of the gas, not via radiative cooling. This is of course a crude approximation to the cooling process. However, Fig. 7.2 in Sect. 7.3 shows that it leads similar feedback energy efficiencies as the more elaborate cooling model described in Ntormousi et al. (2011), which uses detailed cooling tables extracted from CLOUDY.

Cooling time

The definition of the kinetic temperature of atomic gas uses the theorem of equipartition of energy, which in turn states that in thermal equilibrium on average an equal amount of energy is associated with each independent degree of freedom of the motion:

$$E = n_{\mathrm{H}} \frac{3}{2} kT \quad .$$

The change of energy can then be expressed as:

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \frac{3}{2} n_{\mathrm{H}} k \frac{\mathrm{d}T}{\mathrm{d}t} \quad .$$

The cooling time is:

$$t_{\mathrm{cool}} = \frac{\frac{3}{2} n_{\mathrm{H}} kT}{n_{\mathrm{H}}^2 \Lambda} \quad ,$$

with the the cooling rate Λ. After a cooling time, the gas will return to the cooling-heating equilibrium. Since cooling times in the low density phase of the ISM can be much longer than the time between SN events, this was an argument to develop the current dynamic picture of the ISM.

2.3 Multi-Messenger Astronomy

In Sect. 2.2.6 we mentioned large energy losses via radiative cooling. This radiation can help us gathering observational evidence on the ISM. In Fig. 2.2 we sketch the interaction of massive stars

[3] X=0.711, μ_{mol} = 1.2195 g

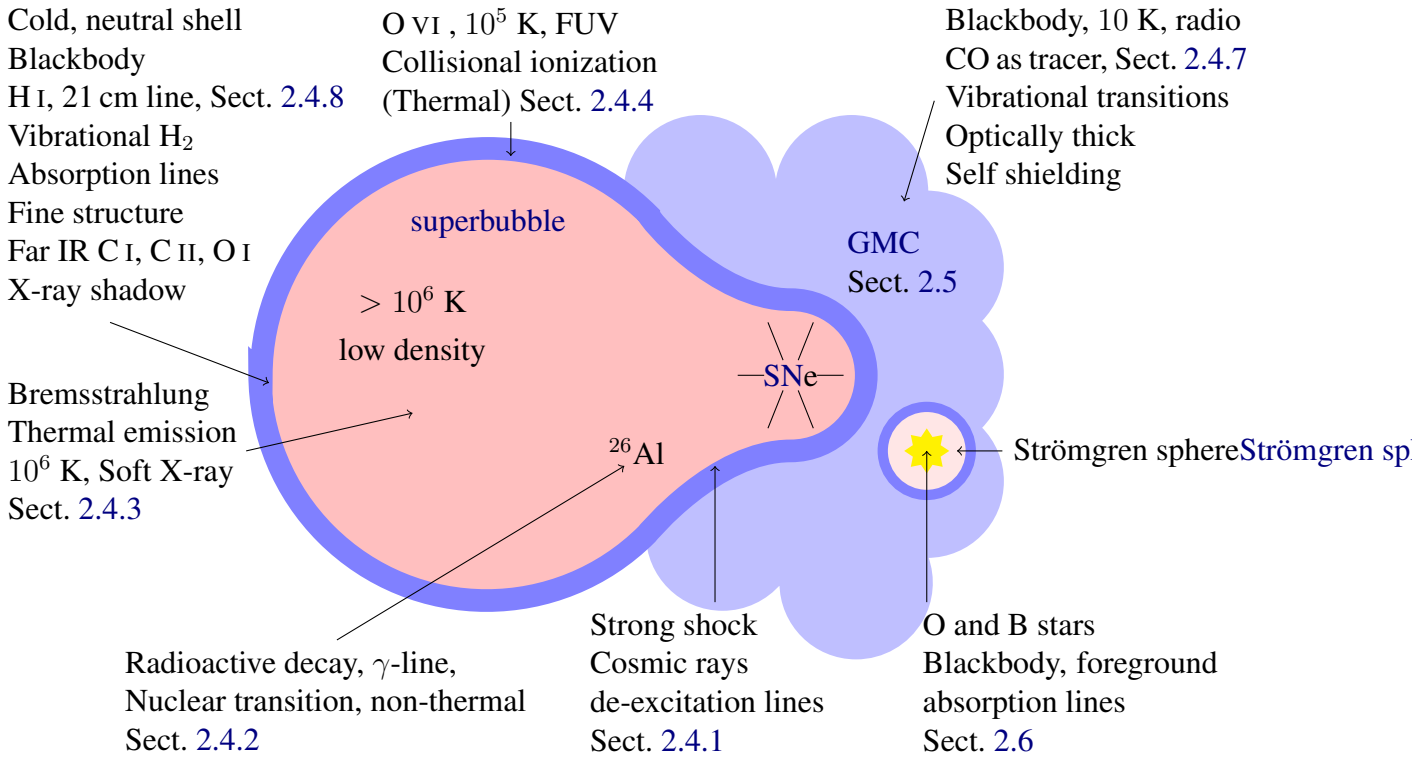


Figure 2.2: Sketch of a superbubble and its messengers. (Temperatures are orders of magnitude.)

with the ISM and label the regions suffering radiative energy losses and the processes leading to this emission of photons. This is of relevance for our work, since one of the aims of this work is comparing our numerical models containing gas with a large range of densities and temperatures to observational data from the Orion-Eridanus region.

The basic idea behind the "Multi-Messenger Astronomy" is to gather information on the same object – in our case the OES – via different physical processes. The "messengers" can be photons, but in principle also neutrinos and cosmic ray particles. However, IceCube (IceCube Collaboration et al., 2013) reports no neutrinos from supernova remnant shocks and also in the COMPTEL (Bloemen et al., 1999) data cosmic ray induced de-excitation lines fell below the significance limit.

We will therefore focus on physical processes leading to the emission of photons. These processes can be subdivided into sources of line emission and continuum emission. We can further subdivide the continuous radiation into thermal and non-thermal radiation. Whereas thermal radiation is characterized via the temperature, since intense interaction leads to an identical energy density of the radiation and the radiating material, non-thermal radiation results from interaction processes far from global energy equilibrium.

Processes leading to line radiation or line absorption are intrinsically quantum phenomena. The quantization of the energy levels in the nucleus and in the shell leads to the emission or absorption of photons in very narrow wavelength ranges. To measure the velocity of the photon-emitting gas, one uses lines with small natural line widths and small thermal broadening. In the Orion-Eridanus region radial velocities are derived from radio data and absorption lines in the spectra from background stars. Also ^{26}Al data from INTEGRAL can measure velocities (Kretschmer et al., 2013) – we are awaiting a result of the ongoing ^{26}Al observations of the OES in the near future.

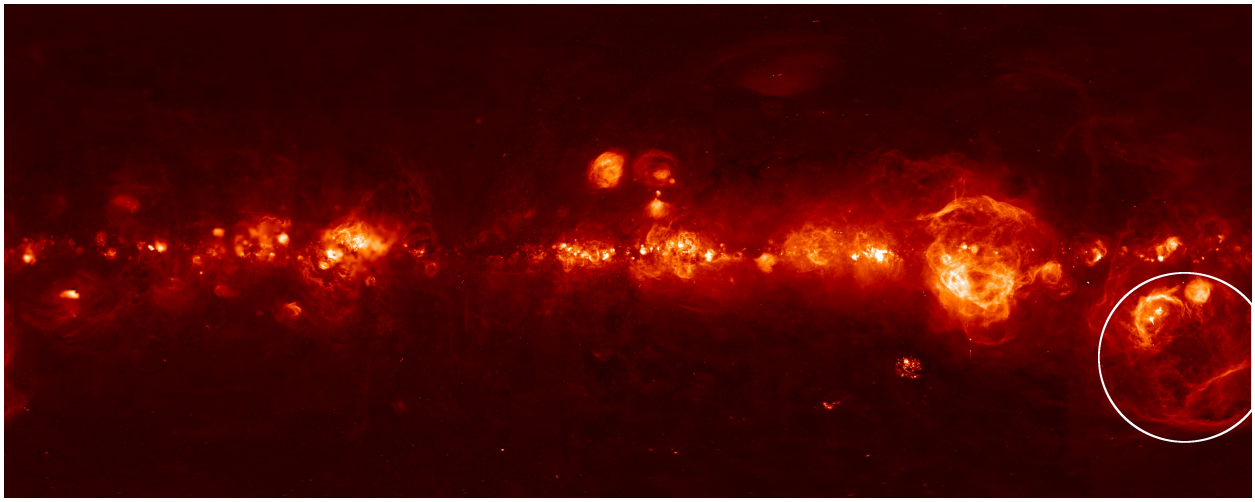


Figure 2.3: Milky Way in Hα. The Orion-Eridanus region is highlighted with a white ellipse. Data source: http://astrometry.fas.harvard.edu/skymaps/halpha .

2.4 Messengers from the Orion-Eridanus region

We will now move from the multi-phase ISM in simulations to multi-wavelength observations. For this discussion, the Orion-Eridanus region will serve as an example, since ^{26}Al data from this region were the motivation for this PhD project. The Orion-Eridanus region extends from the Galactic coordinates $l = 185°$ to $210°$ and $b = -16°$ to $-50°$ and it harbors the Orion-Eridanus Superbubble (OES). Superbubbles are large cavities filled with hot tenuous gas, which were created by the combined feedback of several massive stars. In Fig. 2.3, showing the Milky Way in Hα, the Orion-Eridanus region is highlighted with a white ellipse (this is not to be confused with the assumed boundaries of the OES). The lower part of this region features two strong filamentary Hα shells, called "Arc A" and "Arc B" (see also Fig. 2.4 and Sect. 2.4.5). As we will see in this section, the OES is a particularly good example of a region revealing interactions between young, massive stars and star-forming molecular clouds. In the following subsections the reader will get a glimpse on the observational evidence from this well observed region.
Fig. 2.5 shows an interpretation of the observed data and the position of the OES in the Milky Way. This figure was originally drawn by Burrows et al. (1993) and augmented with ^{26}Al by Diehl (2002). Already Reynolds and Ogden (1979) proposed a similar de-projection of the observed data. For this thesis the distance of the molecular clouds and the locations of the massive stars in this sketch were adapted to the distances used by Voss et al. (2010). Fig. 2.5 also addresses a possible interaction of the OES with the local bubble, which makes this zone even more interesting: With the cloud shadowing technique (presented e.g. in Burrows and Mendenhall, 1991) Burrows et al. (1993) find the molecular cloud L1569 near the interface of the Local Bubble and the OES. This view is strengthened by FUV data of Jo et al. (2011). The position of the H I layer was derived from observed filaments. However, Ryu et al. (2008) favor a different geometry of the OES consisting of two superbubbles both originating in the Orion molecular cloud complex. In this alternative interpretation "Arc A" is not the back side of a single cavity but the front layer of a second cavity. This model is sketched in green in Fig. 2.5. A detailed discussion of the nature of "Arc A" can be found in the appendix of Pon et al. (2014b). Recently the "single cavity" approach was revived by Pon (2013); Pon et al. (2014a) who fitted Kompaneets models to the OES.

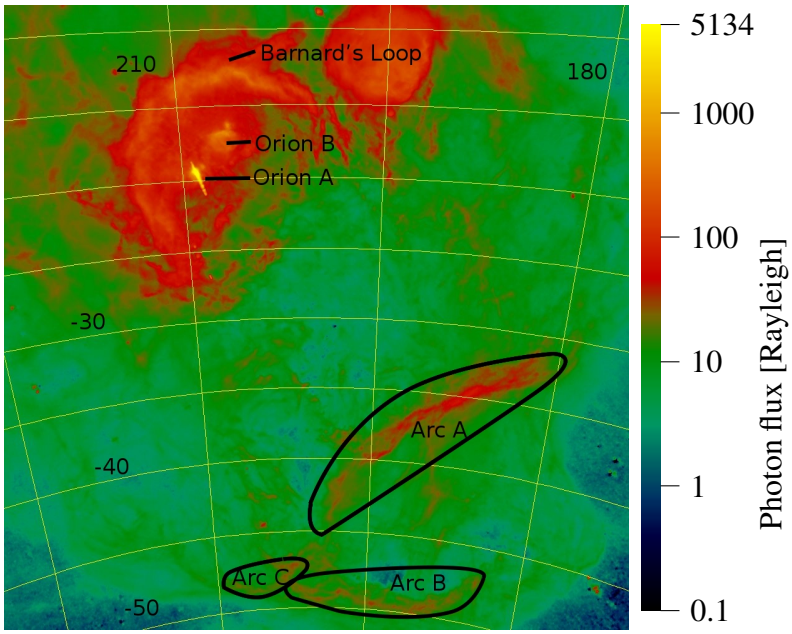


Figure 2.4: Hα features in the OES. The filaments "Arc A" and "Arc B" are also shown in Fig. 2.5. The Hα data of Finkbeiner (2003) was downloaded from skyview.gsfc.nasa.gov. The image size (45°) and the image center (Galactic coordinates $l = 200$, $b = -30$) are the same as in Fig. 2.6. Also the same color bar and the same projection were used.

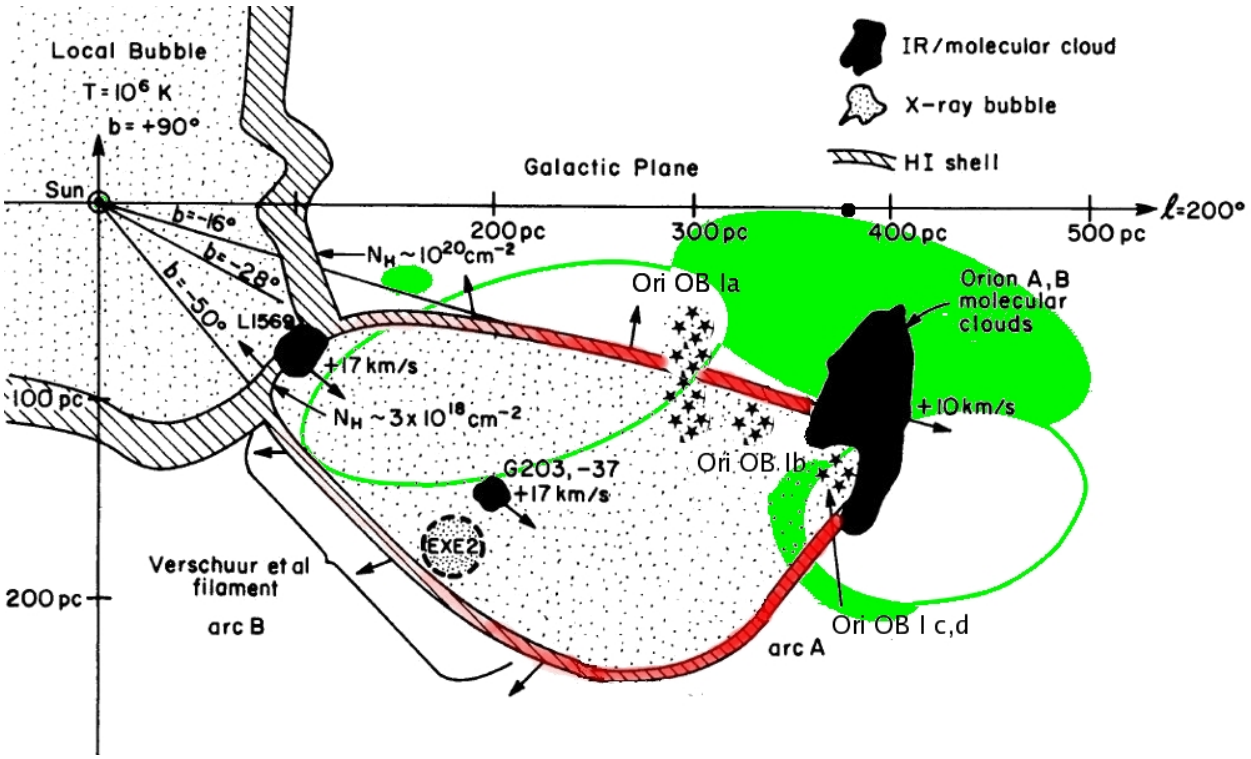


Figure 2.5: This sketch of the Orion-Eridanus Superbubble is a variant of the sketch of Burrows et al. (1993). In this plot the ^{26}Al distribution (red) was added according to Diehl (2002). Moreover the shape of the bubble and the locations of the OB associations use the distance estimates compiled in Voss et al. (2010). In this model, the Orion-Eridanus Superbubble is an adjacent bubble of the Local Bubble. It is located from $l = 185°$ to 210° and $b = -16°$ to $-50°$. The different interpretation of Ryu et al. (2008) is shown with green ellipses. In this alternative model, two separated superbubbles originating from different parts of the Orion GMC complex are assumed.

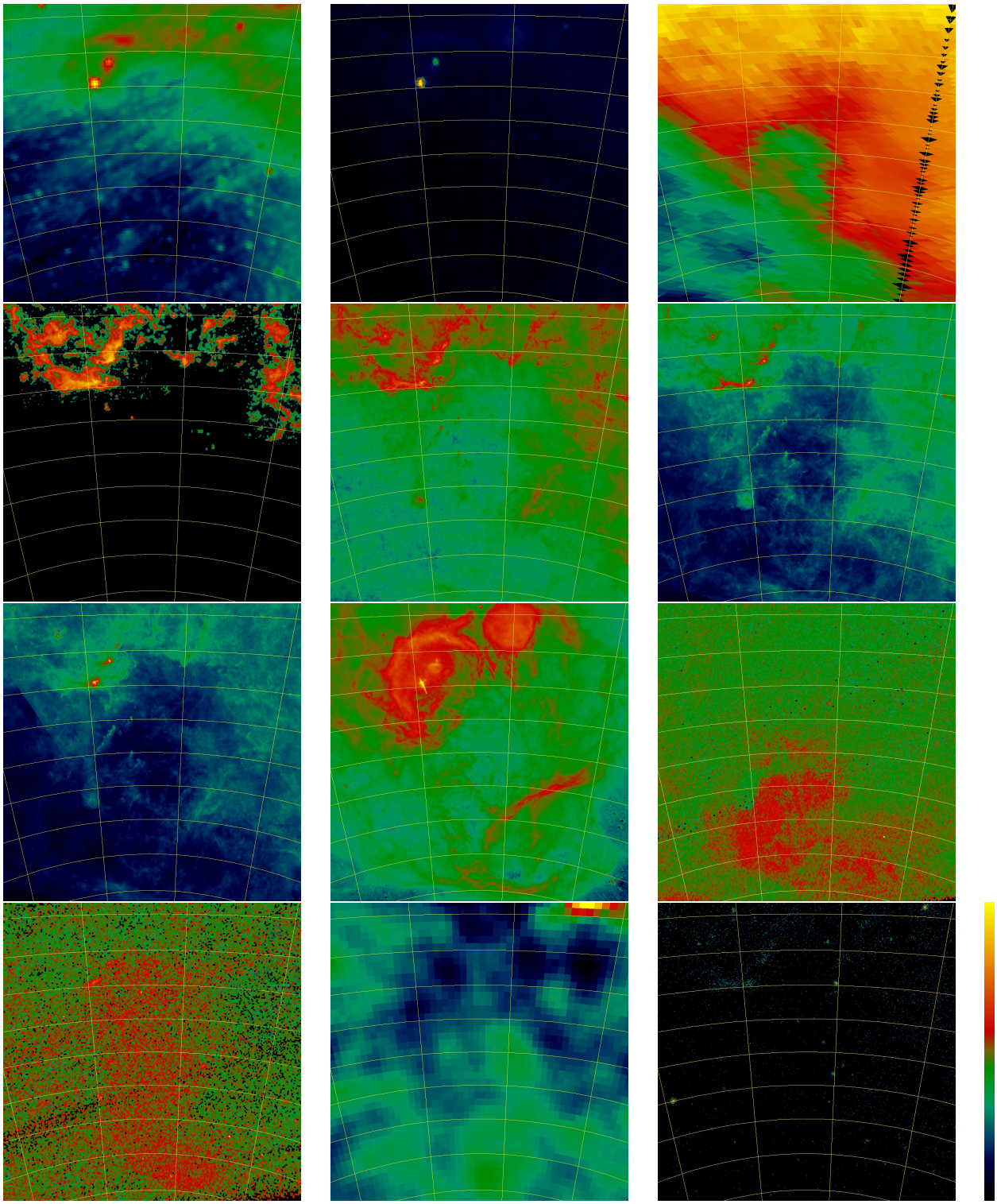


Figure 2.6: Multi-wavelength observations of the Orion-Eridanus region. Sorted by wavelength, from top left: 1st row: 408 MHz, Bonn H I, Dickey and Lockman H I, 2nd row: CO 115 GHz, Planck 353 GHz, Planck 857 GHz, 3rd row: IRIS 100 micron, Hα, ROSAT 0.25 keV, 4th row: ROSAT 0.75 keV, CGRO Comptel $1-30$ MeV, Fermi $3-300$ GeV, Data obtained from skyview.gsfc.nasa.gov. Image size ($45°$), map projection ("Tan") and center ($l = 200°, b = -30°$) as in Fig. 2.4. For references and details (e.g. color bar ranges) see text.

We will now discuss observational evidence for the gas phases in the ISM starting from the data with the shortest wavelengths and ending with radio data. Fig. 2.4 shows a Hα picture of the OES with labeled cold gas structures. In Fig. 2.6 the same region of the sky is depicted in various wavelengths. Labels and coordinates, which are shown in Fig. 2.4, are not shown again in Fig. 2.6.

2.4.1 Cosmic rays: γ-ray data

Parizot (1998) reports that the 3–7 MeV maximum likelihood map of CGRO COMPTEL shows a correlation with the GMCs and might trace the walls of the OES. He argues that the Gamma-ray emission is induced by the interaction of energetic (cosmic ray) particles from inside the super-bubble with the Orion molecular cloud complex, thereby causing non-thermal C and O nuclear de-excitation lines. In Fig. 2.6 we see emission near the GMCs in Band 5 of Fermi (data in Fig. 2.6 from Atwood et al., 2009, band pass 3–300 GeV, color bar: log, values range from 0 to 38 counts). However, this correlation is not visible in the 3 band maximum likelihood map of COMPTEL (data in Fig. 2.6 from Strong, 1994, band passes 1–3 MeV, 3–10 MeV, 10–30 MeV, color bar: log, values range from 2.312×10^{-5} to 2.46×10^{-3} counts s^{-1} cm^{-2} steradian^{-1}). This is in accordance with Bloemen et al. (1999), who report that likely a superimposed signal of the instrument caused a false detection in the data of Parizot (1998). After re-analysis of the data the signal fell below the significance level.

2.4.2 Nucleosynthesis yields: ^{26}Al

^{26}Al is a radioactive trace element for stellar nucleosynthesis and decays approximately a million years after being ejected from the stars (Project, 2004, $\tau_{1/2} \sim 0.72$ Myr). The radioactive decay of ^{26}Al produces an excited ^{26}Mg nucleus. The photon produced at the de-excitation of ^{26}Mg can be observed in Gamma-rays at 1.809 MeV (Project, 2004). Since ^{26}Al decays after ejection from massive stars, ^{26}Al observations provide information on the time scales of the interaction process of massive stars and the ISM. For example the spread of ^{26}Al in different hydrodynamic models can help to understand the potentially peculiar shape (if the model of Burrows et al. (1993) is correct, see Fig. 2.5) of the OES. Further advantages of this tracer are that extinction is no problem and that it shows only a weak dependence on the state of the ISM, since the ^{26}Al -decay is a non-thermal process. However, via the line-shape velocities of the stellar ejecta can be measured, as Kretschmer et al. (2013) have shown for the Galactic center.
Existing COMPTEL data (Diehl et al., 2003, ^{26}Al contours from this publication are overlaid in Fig. 2.7) of ^{26}Al and successful INTEGRAL proposals of R. Diehl and the Gamma group at MPE were the main motivation for this thesis. For the interpretation of this data, it is an interesting question whether ^{26}Al is more likely found inside the bubble or in the cavity walls (see also Sect. 8).

2.4.3 Hot ISM: X-ray data

The ROSAT soft X-ray background data traces the emission of Bremsstrahlung in hot ionized medium (HIM): Fig. 2.6 shows X-ray-emitting hot, diffuse plasma detected with ROSAT (Snowden et al., 1997). The 0.25 keV emission (ROSAT Band 1; band pass: 0.11–0.284 keV; color bar: log; values range from -191 to 50290 in units of 10^{-6} counts s^{-1}) peaks near 10^6 K, whereas the

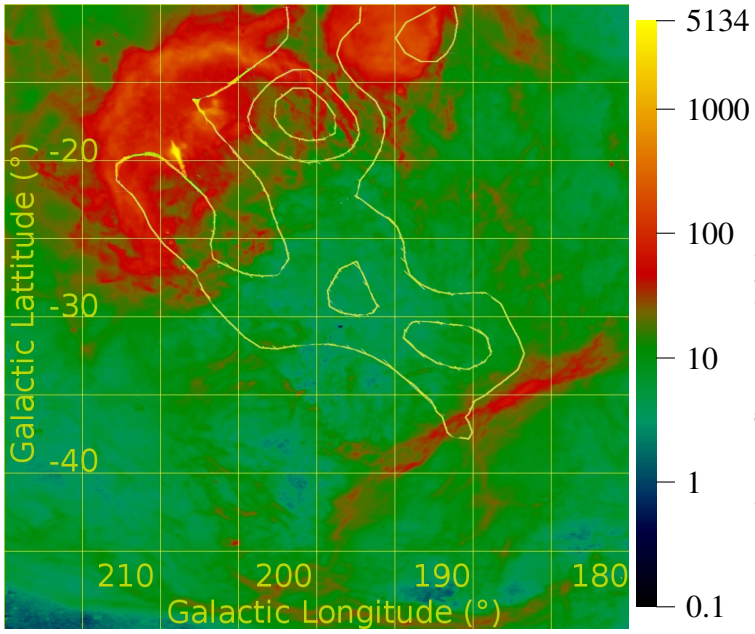


Figure 2.7: ^{26}Al in the Orion-Eridanus region. Total significance $\sim 5\sigma$ contours for the 1.8 MeV COMPTEL data (Diehl et al., 2003), are overlaid on Hα data of (Finkbeiner, 2003, downloaded from skyview.gsfc.nasa.gov). The ^{26}Al emission does not extend beyond "Arc A" (for labeled Hα features see Fig. 2.4). In the two-bubble model (see Fig. 2.5) the bubble bounded by "Arc A" is powered by the feedback of the younger Orion associations OB Ib, OB Ic and OB Id, whereas the larger bubble bounded by "Arc B" contains Ori OB Ia, where all O stars have already exploded.

0.75 keV emission (ROSAT Band 5; band pass: 0.56–1.21 keV; color bar: log; values range from -49 to 20312 in units of 10^{-6} counts s^{-1}) can trace plasma up to 2×10^6 K.

The X-ray emission of the OES was studied by several authors (Burrows et al., 1993; Guo et al., 1995; Snowden et al., 1995; Guo and Burrows, 1996; Burrows and Guo, 1996; Heiles et al., 1999). The common interpretation is a cavity-like region, filled with 2×10^6 K plasma glowing in X-rays due to thermal emission. The energy needed to heat the plasma is believed to originate from winds of hot stars. These winds can collide and shock-heat gas. The X-ray shadow method (presented e.g. in Burrows and Mendenhall, 1991) was used to extract information on the relative distances of the structures visible in different wavebands.

Recent modeling efforts of the X-ray emission of the OES have been published by Krause et al. (2014); Krause and Diehl (2014).

2.4.4 Hot ISM: O VI

More evidence for hot gas in the Orion-Eridanus region is found from UV emission lines of high-stage ions like O VI. Since O VI (five times ionized oxygen) line emission leads to large radiative losses, a high temperature collisionally ionized plasma would quickly cool upon emission. Therefore, O VI emission in the diffuse ISM indicates, that hot gas is replenished. E.g. near the contact discontinuity in a superbubble. The lines of O VI are found at 103.193 nm and 103.762 nm. Kregenow et al. (2006) find that the O VI emission peaks at the thermal interface in "Arc B". The estimated gas temperature is 3×10^5 K. However, de Avillez and Breitschwerdt (2012) showed that in simulations with collisional ionization equilibrium O VI traces higher temperatures than in non-equilibrium models, where 70% of the O VI mass is found in regions with temperatures below 10^5 K. In contrast to O VI in regions with temperatures above 10^5 K, where collisional ionization dominates, the main production channel of O VI at lower temperatures is photoionization.

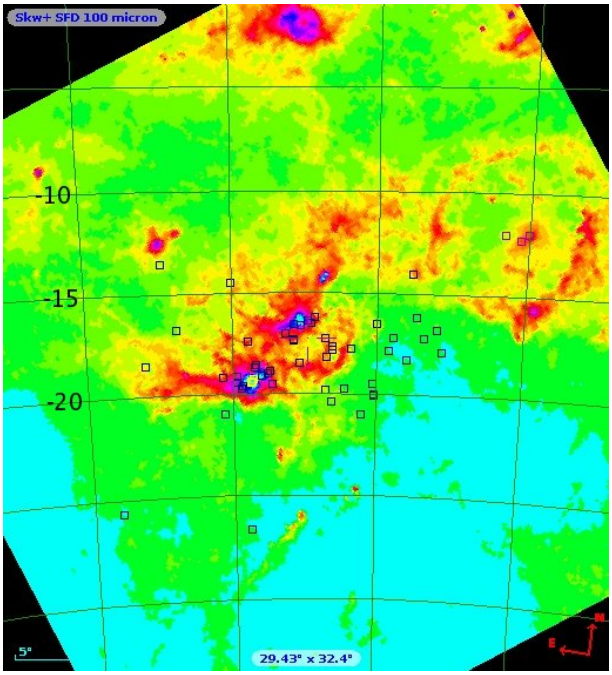


Figure 2.8: 100 micron (Skw+SFD) and the OB stars considered in Voss et al. (2010). This plot was created with the ALADIN interactive sky atlas (Bonnarel et al., 2000). Unfortunately, the astrometric evidence for Ori OB I is limited, because the relative velocities of the stars are mostly directed away from the sun. Thus, de Zeeuw et al. (1999) could not use the Hipparcos parallaxes and velocities to determine the membership of the stars in the field.

2.4.5 Warm ionized interstellar gas: Hα

The warm ionized component of the ISM ($\sim 10^4$ K) can be traced by Hα. This line at 656.28 nm is part of the Balmer series and the brightest spectral line of ionized hydrogen in visible light. It results from the recombination of a proton and an electron to a hydrogen atom. In this process, the electron cascades to the ground state and can pass the $n = 3$ to $n = 2$ transition that leads to the emission of a Hα photon. Fig. 2.3 shows the Milky Way in Hα. A zoom with labeled Hα features is shown in Fig. 2.4. Additionally Hα is shown in Fig. 2.6 (data from Finkbeiner, 2003, band pass: 456.2–457.38 THz; color bar: log; values range from 0.1 to 5134 Rayleighs). Reynolds and Ogden (1979) already reported an ionized shell with a mass of $\sim 8 \times 10^4\ M_\odot$ and a velocity of 15 km s^{-1}. Since Hα cannot trace column densities (it traces its emission measure N(H$^+$)n_e), Reynolds and Ogden (1979) used multi-wavelength data for this result. Boumis et al. (2001) argue that "Arc B" might be closer to the observer, in a distance of $\sim$ 150 pc, whereas "Arc A" could also be at $\sim$ 530 pc. Moreover they also discuss the idea that the two arcs might be boundaries of more than a single hot cavity.

2.4.6 Total number density of warm, cool and cold gas: infrared emission

The 100 micron data (Miville-Deschenes and Lagache, 2008; Miville-Deschênes and Lagache, 2005, band pass: 2.5–3.6 THz; color bar: log; values range from 10^6 to 2.4358×10^{10} Jansky steradian^{-1}) shown in Fig. 2.6 trace the total number density of H I, H II and H_2 via thermal emission. Therefore we use this data to overlay the positions of the massive stars of the Orion OB I associations (Fig. 2.8). A more recent all-sky observation of dust is the Planck 857 GHz Survey (Planck Team, 2013, 857 GHz; color bar: log; values range from 0.4 to 8916 counts (native map units are in million Jansky per steradian)), also shown in Fig. 2.6.

2.4.7 Molecular gas: CO and H_2 fluorescence

In the OES, molecular gas can be found in the Orion A and B molecular clouds (see Sect. 2.5) and in the filamentary structure called "Arc A" (Ryu et al., 2008).
Whereas cold H_2 does not have radio emission, CO, the second most abundant molecule after H_2, shows a strong signal from rotational transitions. The physics behind this is that, in contrast to the homonuclear H_2 molecule, CO has a small dipole moment and can thus absorb or emit radiation on vibra-rotational transitions. The CO(1-0) line[4] at 2.6 mm (115.271 GHz) is shown in Fig. 2.6 (Dame et al., 2001, band pass: 114.89–115.12 GHz; color bar: log of velocity-integrated main beam brightness temperature; values range from -32768 to 180 K km s^{-1}). The Planck filter centered around 353 GHz (Planck Team, 2013, 353 GHz; color bar: log; values range from -1.2×10^{-5} to 1.9 counts (native map units T_{CMB})) can trace the CO(3-2) line at 345.796 GHz. To infer the distribution of H_2 from CO observations, the H_2 to CO ratio has to be calibrated via UV absorption lines of CO and H_2. The CO to H_2 conversion factor is still debated. For a recent review see Bolatto et al. (2013).
In the Orion-Eridanus region Ryu et al. (2006, 2008) have observed H_2 fluorescence in far-ultraviolet (135–175 nm) with the SPEAR/FIMS mission. They find a correlation with Hα emission and suggest that UV radiation from the Ori OB I associations might be responsible for both, the fluorescence and the recombination emission. They conclude that "Arc A" is likely to be at a distance of $\sim$ 500 pc, whereas "Arc B" could be on the near side of the cavity at $\sim$ 150 pc. In both regions excitation temperatures can reach up to 1000 K. Ryu et al. (2008) argue that "Arc A" is mostly associated with molecular and dust components while "Arc B" can be more or less characterized by atomic origins. Based on these findings, they suggest two unrelated bubbles instead of one peculiar shaped superbubble. This model is shown with green ellipses in Fig. 2.5.

2.4.8 H I: 21 cm line

The 21 cm line of neutral hydrogen (1420.4 MHz) results from a transition between the hyperfine levels of the hydrogen 1s ground state. Since the relative orientation of electron spin and nuclear spin is causing these energy levels, the 21 cm line traces H I column density for a wide range of temperatures. Already Clark (1965) pointed out that the difference between 21 cm absorption and emission indicates contributions from a cold neutral medium (CNM, optically thick seen in emission and absorption) and a warm neutral medium (WNM, optically thin, only seen in emission). These two phases are the two co-existing phases in the cooling-heating equilibrium, which is discussed in Sect. 2.2.6. In the WNM number densities n_H of 0.03 to 1.3 cm^{-3} are observed. The kinetic temperatures lie between 4000 and 8800 K. Recently, (Murray et al., 2014) measured an excitation temperature of $\sim 7200^{+1800}_{-1200}$ K in their survey of the Galactic WNM. They conclude, that resonant Lyman-α scattering in addition to collisional excitation leads to this temperature. The CNM has higher number densities ($n_H \sim 5$ to 120 cm^{-3}) and lower temperatures (kinetic temperature of the order of $40 - 200$ K) than the WNM. In contrast to the diffuse distribution of the WNM, the CNM has a filamentary structure, possibly originating from turbulence, seen as absorption peaks in spectra.
In the review of Kalberla and Kerp (2009), a local exponential vertical scale height above the Galactic plane of $\sim$ 150 pc for the CNM and $\sim$ 400 pc for the WNM is reported.

[4] CO(1-0) is a transition between the ground state and the first excited level. A rough estimate of the levels can be found by using the rigid rotor model and solving for the Eigenvalues of the Schrödinger equation.

In the H I velocity profiles of the THINGS galaxies Ianjamasimanana et al. (2012) find a broad component with a mean velocity dispersion of 16.8 ± 4.3 km s^{-1} and a narrow component with a mean velocity dispersion of 6.5 ± 1.5 km s^{-1}. They discuss an indication that the narrow component is associated with molecular gas, which is in accordance with the theoretical expectation that atomic gas passes through the CNM phase before turning into molecular gas.
In the vicinity of the OES 21 cm radiation of neutral hydrogen ($T \sim 10^2$ to 10^3 K) outside the hot bubble was reported e.g. by Brown et al. (1995) who estimate a H I shell mass of $\sim 2.3 \times 10^5\ M_\odot$ (combining H I with 100 micron observations).
The first line of Fig. 2.6 shows two surveys using the 21 cm line of H I: the Bonn 1.4 GHz Survey (Reich, 1982; Reich and Reich, 1986, band pass: 1418.8–1421.2 MHz; color bar: log; values range from 0 to 15324 mK) and the Dickey and Lockman H I map (Dickey and Lockman, 1990, band pass: 1418.8–1421.2 MHz; color bar: log; values range from 0 to 3.964×10^{21} atoms cm^{-2}). An alternative approach is used in the "H I All-Sky Continuum Survey" (Haslam et al., 1982, band pass: 406.25–409.75 MHz; color bar: log; values range from 12.5 to 108.3 K), which shows mostly synchrotron radiation. All of these H I surveys show an anti-correlation with X-ray data.

2.5 Giant Molecular Clouds (GMCs)

For our study GMCs are of major importance, since stars form in such cold, dense molecular gas. Despite the small volume fraction of molecular clouds, about a half of the Milky Way's ISM mass inside the orbit of the sun is found in molecular gas (Williams and McKee, 1997).
Typically, molecular clouds with masses above $\sim 10^4\ M_\odot$ (Blitz, 1993; Williams et al., 2000) are called GMCs. The distinct features in the upper part of the CO observations shown in Fig. 2.6 are Orion's Giant Molecular Clouds. Distance estimates of Brown et al. (1994) find the near edge of the Orion A and Orion B molecular clouds at a distance of 320 pc and the far edge at a distance of 500 pc. The properties of the Orion A and Orion B molecular clouds are summarized in Tab. 2.1.
The Milky Way also harbors more massive GMCs than Orion's GMCs. Williams and McKee (1997) list GMCs with up to $\sim 6 \times 10^6\ M_\odot$ and Murray (2011) finds masses up to $\sim 1.3 \times 10^7\ M_\odot$. The diameters of GMCs are in the range of tens of parsecs to a few hundred parsecs (Murray, 2011, up to 210 pc).
For molecular cloud complexes in the Milky Way an average density of 1.7×10^{-22} g cm^{-3} (corresponding to $\sim 100\ m_\mathrm{H}$ cm^{-3} or $\Sigma \sim 100\ M_\odot$ pc^{-2}) can be found from column 14 and 15 in Table 1 of Murray (2011). Tan et al. (2013) also find that typical, ^{12}CO defined GMCs have a mass surface density of $\Sigma \sim 100\ M_\odot$ pc^{-2}. The median value of the number density of H_2 in the Galactic ring survey (Roman-Duval et al., 2010) is 230 ± 21 cm^{-3}. However, this survey is likely biased towards high density regions, since it is based on 4σ ^{13}CO contours. Similar techniques led to a factor of 10 lower densities in Heyer et al. (2009).
As Larson (1981) showed, GMC mass and size correlate with the velocity dispersion. Kritsuk et al. (2013) conclude that these relations can be interpreted as an empirical signature of supersonic turbulence. Turbulence also leads to a very inhomogeneous, sponge-like self similar density structure of molecular clouds. Molecular clouds contain clumps that may form star clusters and cores that may form single stars or binaries (Tan et al., 2013). Dense cores with densities of $10^4 - 10^6$ particles cm^{-3} are considered the most important environment for star formation. Whereas these dense cores and clumps are gravitationally bound, it is debated whether GMCs are also gravitationally bound, since kinetic energy in turbulent motions can balance or even outweigh self gravity. For

	Mass $[M_\odot]$	Diameter [pc]	Density [g cm^{-3}]	Reference
Orion A GMC	10^5	65	$< \rho > \sim 10$ H$_2$	[1]
Orion B GMC	6×10^4	25	$< \rho > \sim 110$ H$_2$	[1]
Orion A GMC	$\sim 10^5$	40×2 (29 deg^2)	$\rho_{\max} > 10^{-20}$	[2]
Orion B GMC	$\sim 10^5$	(19 deg^2)	max. $\rho_{\max} > 10^{-20}$	[2]
Orion A GMC	$\sim 10^5$	110×20 (31.5 deg^2)		[3]
Orion B GMC	$\sim 8 \times 10^4$	~ 40 (25.7 deg^2)		[3]
Orion A GMC	8.1×10^4			[4]
Orion B GMC	4.0×10^4			[4]
Realistic SPH cloud	2.8×10^5	~ 40	$< 9 \times 10^{-22}$ $> 1.66 \times 10^{-24}$	[5]
Homogeneous cloud	1.6×10^5	~ 50	1.66×10^{-22}	This work

[1] Larson (1981), [2] Genzel and Stutzki (1989), [3] Wilson et al. (2005), [4] Okumura et al. (2009), [5] Dobbs et al. (2011)

Table 2.1: Giant Molecular Clouds (GMCs, Sect. 2.5).

example Tan et al. (2013) argue that most GMCs are indeed gravitationally bound whereas Blitz et al. (2007) claim that GMCs are only marginally stable.

2.5.1 Simulated clouds

Based on these findings, we will use an average number density in the order of $\sim$ 100 cm^{-3} which lies well in the plausible region of average densities in molecular clouds, for the cold, dense clouds in this work. Tab. 2.1 also contains two toy-models, which we use in our simulations: a simplified, homogeneous GMCs and a cloud from a large scale simulation of Dobbs et al. (2011). Both artificial cloud models are of comparable size and mass as Orion's GMCs.

Due to the masses, the velocity dispersions and the magnetic fields inferred from observations, GMCs are believed to be self gravitating (Pon et al., 2012; Tan et al., 2013) and supported against collapse by turbulence and magnetic fields. However, in this work gravity is not considered in the simulations, since its aforementioned antagonists (turbulence, magnetic fields) are absent. Moreover, the free-fall time ($\tau_{\rm ff} = \sqrt{3\pi/(32G\rho)}$) is about 5 Myr for $n \sim 100$ cm^{-3}. Thus, the timescales on which self-gravity acts are rather large. The internal velocity dispersion (σ) that would lead to gravitational stability of the homogeneous GMC in Tab. 2.1 can be found from the virial theorem $M_{\rm vir}/[1\,M_\odot] = 1160 R/[1\,pc](\sigma/[1\,km\,s^-1])^2$ to be twice the sound speed in this cloud. The virial mass $M_{\rm vir}$ balances the internal motions if external pressure and magnetic fields are not taken into account.

Another reason not to include gravity is that the focus of this work in not calculating GMC lifetimes but rather to testing if stellar feedback is efficient enough to play a role in driving turbulence in GMCs. The advantage of our minimalist toy model is that simple relations between the feedback energy efficiency and the depth of embedding of the stars or the porosity of the GMC can be seen.

Group	Location	Distance [pc]	Age [Myr]	M_{up} [$M_\odot$]	M_{max} [$M_\odot$]	SN	O stars	B stars
OB Ia	outside GMC	330	8-12	18.5	13	7.3	0	16
OB Ib	Ori B GMC	360	0.5-8	45	40	1.3	4	10
OB Ic	Ori A GMC	400	3-6	45	32	1.5	2	19
OB Id	Ori A GMC	410	0-2	120	36	0	5	2

Table 2.2: Massive stars in Ori OB I according to Voss et al. (2010).

	OB stars	ø [pc] long.	ø [pc] lat.
Ori OB I a	5	9.9	2.7
Ori OB I b	27	78.6	36.8
Ori OB I c	29	38.2	32.4

Table 2.3: Massive stars in Ori OB I according to Mel'Nik and Efremov (1995).

2.6 Massive stars

Already the data of the HD catalog showed that OB stars are not homogeneously distributed over the sky but rather grouped. OB stars form in loose groups – so-called OB associations. These associations can be detected kinematically, because they have a very small internal velocity dispersion of only a few km s^{-1}. Thus, they can be seen as coherent structures in velocity space. Since OB associations have lower masses than bound star clusters, OB associations disperse within a few million years and their stars spread over larger diameters (diameters in the Hipparcos sample de Zeeuw et al. (1999, Fig. 29) and Brown et al. (2000, table 1) are in the order of 50 pc) than clusters (order of 1 pc). The reader interested in the history of the research on OB associations is referred to the introduction of de Zeeuw et al. (1999).

2.6.1 Orion's OB associations

The massive star content of the Orion-Eridanus region has recently been summarized by Voss et al. (2010). The still existing OB stars in the four sub-groups of the Orion OB I associations are shown in Fig. 2.8. Their positions are also marked in Fig. 2.5. Tab. 2.2 combines the positions (shown in Fig. 2.8) of the massive stars with distance and mass estimates of Voss et al. (2010). M_{up} is the most massive not yet exploded star of the age given in column 4. M_{max} is the most massive observed star. Column "SN" shows an Initial Mass Function (IMF) based estimate of the number of already exploded stars. The last two columns list the observed O and B stars.

Unfortunately, the astrometric evidence for Ori OB I is limited, because the relative velocities of the stars are mostly directed away from the sun. Thus, de Zeeuw et al. (1999) could not use the Hipparcos parallaxes and velocities to determine the membership of the stars in the field. For the same reason the stellar content discussed in Voss et al. (2010) differs from the massive star content mentioned in Mel'Nik and Efremov (1995). The latter is listed in Tab. 2.3.

2.7 Stellar feedback

Basically, we have to distinguish between the feedback of individual stars (discussed in Sect. 2.7.2 to 2.7.4) and the feedback of groups of stars (Sect. 2.7.5). In this work, we are interested in the feedback of OB associations. We can find the global feedback of the associations by determining the number of stars in the association and their masses and then adding up the feedback models for these stars. However, this approach does not treat energy losses e.g. via colliding winds of individual stars in the group. The reader interested in this aspect is referred to Krause et al. (2012), where the interaction of the wind-blown bubbles of massive stars has been investigated.
The stellar feedback of OB associations including ^{26}Al has been modeled e.g. by Voss et al. (2009, 2010) via population synthesis. These models compute a Monte-Carlo sample of stars between 8 and 120 $M_\odot$ from a Salpeter (1955) Initial Mass Function (IMF). For each Monte-Carlo realization stellar wind velocities, stellar evolution models (mass loss rate, surface abundances) and SN models (remnant mass, trace elements ejected in the SN) are combined. If no stellar evolution model with the given mass exists, models are interpolated logarithmically. The resulting energy curve is an average of all Monte-Carlo realizations. Since a few percent of the Monte-Carlo realizations with extremely massive stars influence the average strongly, this average feedback is stronger than a "typical feedback" (the median). Moreover, the interpolation of models leads to unphysical additional discontinuities in the energy injection rate (Fig. 2.9 to 2.12).
Therefore, we took a step back and evaluated the contribution of the individual stars in OB associations. Unsurprisingly, the most massive still existing star dominates the feedback (Fig. 2.9, details in Sect. 2.7.5 and Fig. 2.18). Our problem has thus been reduced to finding the mass of the most massive still existing star in an OB association of given total mass.
A plausible mass of this star can be estimated from the molecular clouds in the Milky Way: Under the assumption that about 8% (Murray, 2011) of the molecular cloud mass are converted to stars, we expect a cluster mass of $8 \times 10^3\ M_\odot$ for a molecular cloud of $10^5\ M_\odot$. In the galactic ring survey (Roman-Duval et al., 2010) $\sim$ 18% and in the list of Heyer et al. (2009) $\sim$ 31% of the galactic molecular clouds are estimated to be more massive than $10^5\ M_\odot$. Weidner et al. (2013) find a most massive star of $\sim 60\ M_\odot$ for a cluster mass of $8 \times 10^3\ M_\odot$ with their polynomial fit to the observed most massive stars as a function of the cluster mass. This also fits well to the OES: The most massive star in Ori OB Ib is ϵ Ori A with about 40 $M_\odot$ (Voss et al., 2010). The older association Ori OB Ia is expected to be more massive than Ori OB Ib and Voss et al. (2010) assume seven already exploded stars with masses above 18 $M_\odot$. This would lead to a most massive star with approximately 60 $M_\odot$. Thus, we start with 1D spherically symmetric simulations focusing on the feedback energy efficiency of a 60 $M_\odot$ star (Sect. 6), since a good fraction of the GMCs can harbor most massive stars of this mass. The stellar winds in this model play an important role, since they insert 2.34 times the SN energy into the ambient ISM. This wind-to-SN ratio is larger than in Voss et al. (2009), since we consider individual massive stars, whereas Voss et al. (2009) are interested in OB associations. In groups of stars, less massive stars lower the ratio of wind energy to SN energy if a canonical SN energy of 10^{51} erg is assumed.

2.7.1 Mass loss rates and surface abundances

The mass loss rates and surface abundances for our wind models for individual stars are taken from the rotating models of Ekström et al. (2012). These surface abundances also supply us with information on the mass fraction of ^{26}Al in the wind. Voss et al. (2009, 2010) base their population

synthesis on the rotating stellar evolution models with solar metallicity (z=0.02) of Palacios et al. (2005) (= default mode "geneva05"). This is the first set of models in the Geneva grids with ^{26}Al surface abundances.
The Geneva grids of stellar evolution models cover stellar masses between 0.8 and 120 $M_\odot$ and metallicities from z=0.001 to 0.1 . A list of the models can be found on-line at
http://obswww.unige.ch/Recherche/evol/Geneva-grids-of-stellar-evolution.
For our work, we compared the following 5 sets of models:

- Meynet et al. (1994) models,
  http://cdsarc.u-strasbg.fr/cgi-bin/myqcat3?J/A+AS/103/97
  masses between 12 and 120 $M_\odot$, metallicities from z=0.001 to 0.04, mass loss rate increased by a factor of two during MS and WNL phases

- Meynet and Maeder (2003) models,
  http://obswww.unige.ch/Recherche/evol/tables_WR/
  masses between 9 and 120 $M_\odot$, solar metallicity (z=0.02), axial rotational velocity on the ZAMS[5] 0 or 300 km s^{-1}. In the latter models wind anisotropies are taken into account.

- Meynet and Maeder (2005) models,
  http://obswww.unige.ch/Recherche/evol/tables_WR_nosolar/
  masses between 9 and 120 $M_\odot$, metallicities from z=0.004 to 0.04, axial rotational velocity on the ZAMS 0 or 300 km s^{-1}, also models with metallicity dependent mass loss rates during the Wolf-Rayet (WR) stage.

- Palacios et al. (2005) models,
  http://www.aanda.org/articles/aa/full/2005/02/aa1757/aa1757.html
  masses between 25 and 120 $M_\odot$, axial rotational velocity on the ZAMS 0 or 300 km s^{-1}, metallicities 0.004, 0.008, 0.02 and 0.04, ^{26}Al surface abundances. These models are not available on-line – the stellar feedback data was provided by R. Voss (logarithmically interpolated models, which served as raw data for Voss et al. (2009, 2010)).

- Ekström et al. (2012) models,
  http://obswww.unige.ch/Recherche/evol/tables_grids2011/
  http://obswww.unige.ch/Recherche/evol/Grids-of-rotating-stellar-models
  masses between 0.8 and 120 $M_\odot$, new solar metallicity (z=0.014), no rotation or $v/v_{\rm crit} = 0.40$ [$v_{\rm crit}$ listed in column 37], ^{26}Al surface abundances.

Fig. 2.13 compares the time dependent stellar masses of all available models for a 120 $M_\odot$ star and Fig. 2.14 shows the estimates for the ^{26}Al content of the winds of such a star. The time dependent stellar masses of models from 9 to 120 $M_\odot$ are compared in Fig. 2.10. Fig. 2.11 shows the mass loss rates. In both figures the 32 $M_\odot$ model is not shown, since it is only available in Ekström et al. (2012). The raw data for Voss et al. (2009) shows saw tooth like features that are not present in the Geneva models. These features are artifacts from the interpolation between stellar tracks.

2.7.2 Stellar wind velocities

We estimate the time dependent wind velocity from the surface abundances as summarized in Table 2 of Voss et al. (2009). The surface abundances are taken from the Geneva models (details

[5]ZAMS stands for "zero age main sequence" and refers to the start of the hydrogen fusion in the stellar core.

Type	Classification criteria	Wind velocity	Reference
LBV	$3.75 < \log T_{\rm eff} < 4.4$ $\dot{M} > 10^{-3.5} M_\odot$ yr^{-1}	200 km s^{-1}	Leitherer et al. (1999)
WR	$\log T_{\rm eff} > 4.0$		
WR type	Surface abundances		
WNL	$0.4 > {\rm H}_s > 0.1$	1250 km s^{-1}	Niedzielski and Skorzynski (2002)
WNE	${\rm H}_s < 0.1$ ${\rm C}_s/{\rm N}_s < 10$	2000 km s^{-1}	Niedzielski and Skorzynski (2002)
WC6-9	${\rm H}_s < 0.1$ ${\rm C}_s/{\rm N}_s > 10$ $({\rm C}_s + {\rm O}_s)/{\rm He}_s < 0.5$	1760 km s^{-1}	Niedzielski and Skorzynski (2002)
WC4-5	${\rm H}_s < 0.1$ ${\rm C}_s/{\rm N}_s > 10$ $({\rm C}_s + {\rm O}_s)/{\rm He}_s < 1.0$	2650 km s^{-1}	Niedzielski and Skorzynski (2002)
WO	${\rm H}_s < 0.1$ ${\rm C}_s/{\rm N}_s > 100$ $({\rm C}_s + {\rm O}_s)/{\rm He}_s > 1.0$	3000 km s^{-1}	Niedzielski and Skorzynski (2002)
	Outside categories		
cool star	$\log T_{\rm eff} < 4.32$	$v_\infty = 1.3 v_{\rm esc}$	Lamers et al. (1995)
hot star	$\log T_{\rm eff} > 4.32$	$v_\infty = 2.6 v_{\rm esc}$	Lamers et al. (1995)

Table 2.4: Classification criteria for stellar winds. This is a modified version of Table 2 in Voss et al. (2009) for their default mode "`wind08`". Stellar types are defined according to Smith and Maeder (1991); Leitherer et al. (1999). ${\rm H}_s, {\rm C}_s, {\rm N}_s$ and ${\rm He}_s$ are the fractional surface abundances (mass fraction) of hydrogen, carbon, nitrogen and helium, respectively.

in Sect. 2.7.1). In the default mode "`wind08`", Voss et al. (2009, 2010) classify the stellar types according to Smith and Maeder (1991) and Leitherer et al. (1999). The assumed wind velocities are listed in Table 2.4. The escape velocity is computed from the effective temperature (column 5 in the Geneva models), the mass (column 3 in the Geneva models) and the luminosity (column 4 in the Geneva models) via

$$v_{\rm esc} = \sqrt{\frac{2GM}{r}} \overset{L=4\pi\sigma T_{\rm eff}^4 r^2}{=} 2T_{\rm eff}\sqrt{GM}\sqrt[4]{\frac{\pi\sigma}{L}} \quad . \tag{2.4}$$

The differences between the escape velocities of the Meynet and Maeder (2003) and Ekström et al. (2012) models shown in Fig. 2.12 are mostly due to different effective temperatures. The wind velocities in the interpolated models for the population synthesis of Voss et al. (2009, 2010) show similar interpolation artifacts as the mass loss rates.

2.7.3 Computed feedback momentum and kinetic energy

For our simulations we estimated the wind velocity (as explained in Sect. 2.7.2) for each data point in the Ekström et al. (2012) tables. The time dependent feedback energy and momentum were then computed on the fly during the hydrodynamic simulations by linear interpolation in these tabulated

mass loss rates and corresponding wind velocities. Comparing the stellar feedback computed with the mass loss rates and surface abundances in the models of Ekström et al. (2012) and Meynet and Maeder (2003) to the feedback extracted by Rasmus Voss from Palacios et al. (2005) shows that the latter has more scatter in the wind velocities and a varying slope of the energy input (see Fig. 2.9 to 2.12). These effects are probably caused by logarithmic interpolation on a coarse grid. The expected behavior would be between the rotating models of Ekström et al. (2012); Meynet and Maeder (2003), probably closer to Ekström et al. (2012), since – as explained in Voss et al. (2009) – the code had to be modified to include ^{26}Al. Since such features will lead to additional shock waves in the simulation, we decided to stick with the leading order term of the feedback instead of introducing effects from such features. Fig. 2.9 shows that the leading order term is the feedback of the most massive still existing star. This will be elaborated in Sect. 2.7.5.

2.7.4 Supernovae

All supernova (SN) energies are estimated to be 10^{51} erg. The mass loss during the SN is the difference between the mass at the last point of the stellar evolution model and the remnant mass. The remnants are assumed to be neutron stars with canonical masses of 1.4 $M_\odot$ for stars with initial masses below 25 $M_\odot$ and black holes with canonical masses of 7 $M_\odot$ for more massive stars. These estimates are also used by Voss et al. (2009, 2010). Figure 2.15 shows the mass loss per SN event against time between ZAMS and SN event (which is larger for smaller initial masses). In Sect. 6 we show that in the ambient densities considered in this work the actual amount mass loading of the SNe only leads to minor differences in the retained feedback energy.
The amount of radioactive tracers ^{26}Al and ^{60}Fe ejected during the SN event is estimated using Table 2 and 3 in Limongi and Chieffi (2006). This is relatively independent of the mass cut (i.e. the mass coordinate that separates ejected material from material forming the remnant), since ^{26}Al and ^{60}Fe are synthesized close enough to the surface. The SN mass loss and the amounts of released trace elements for different initial masses are summarized in Fig. 2.16.

2.7.5 Feedback of individual stars in an OB association

In this section we will compare the stellar feedback of individual stars in a typical OB association. Our approach is to use the cumulative distribution function (CDF) of a group of stars with a power law[6] initial mass function (IMF, $\xi(M)$) to get the mass distribution of the stars. $dN = \xi(M)dM$ quantifies how many stars are expected in a mass interval $[M, M + \mathrm{d}M]$. For the Salpeter IMF (Salpeter, 1955) the exponent $\alpha = 2.35$ for the power law $\xi(M) \propto M^{-\alpha}$ was derived from observations.
Predicting the mass of the most massive star in a cluster of given mass is still an active field of research, since random sampling from the IMF fails to match the observations. Already Elmegreen (2000a) discussed the problem of the most massive star for a Salpeter IMF. Basically, the mass interval for the most massive star contains only a single star

$$k_1 \int_{M_{\max}}^{\infty} n(M)dM = 1 \quad ,$$

and the cluster mass

$$M_{\text{cluster}} = k_1 \int_{M_{\min}}^{\infty} Mn(M)dM$$

[6] or multi part power law

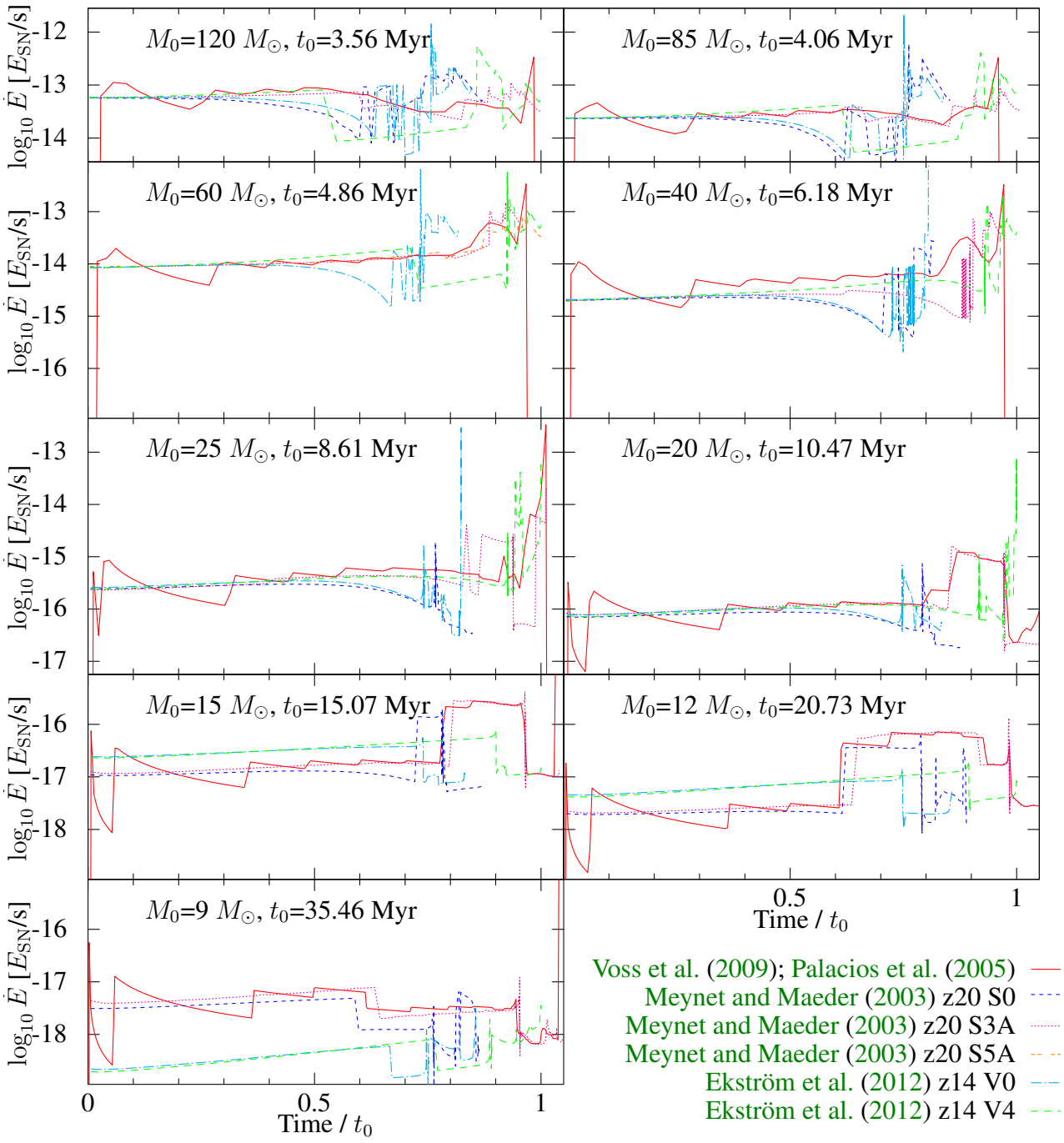


Figure 2.9: These plots combine the data in Fig. 2.11 and Fig. 2.12. The feedback energy rates are normalized to supernova energies ($E_{SN} = 10^{51}$ erg) per second. The times are normalized to the end times of the rotating Ekström et al. (2012) models (t_0). For better visibility line plots were used for the tabulated functions. The numbers of points per plot are: 51 points in models of Meynet et al. (1994), 350 points in models of Meynet and Maeder (2003), 400 points in models of Ekström et al. (2012) and (interpolated) points every 0.1 Myr in the time dependent masses used for Voss et al. (2009) which are based on stellar models of Palacios et al. (2005). The data points in the other models are placed non uniformly in time to ease the comparison of the similar evolution stages in different models. This figure also illustrates, why we focus on massive stars: these stars evolve faster and have more energetic winds.

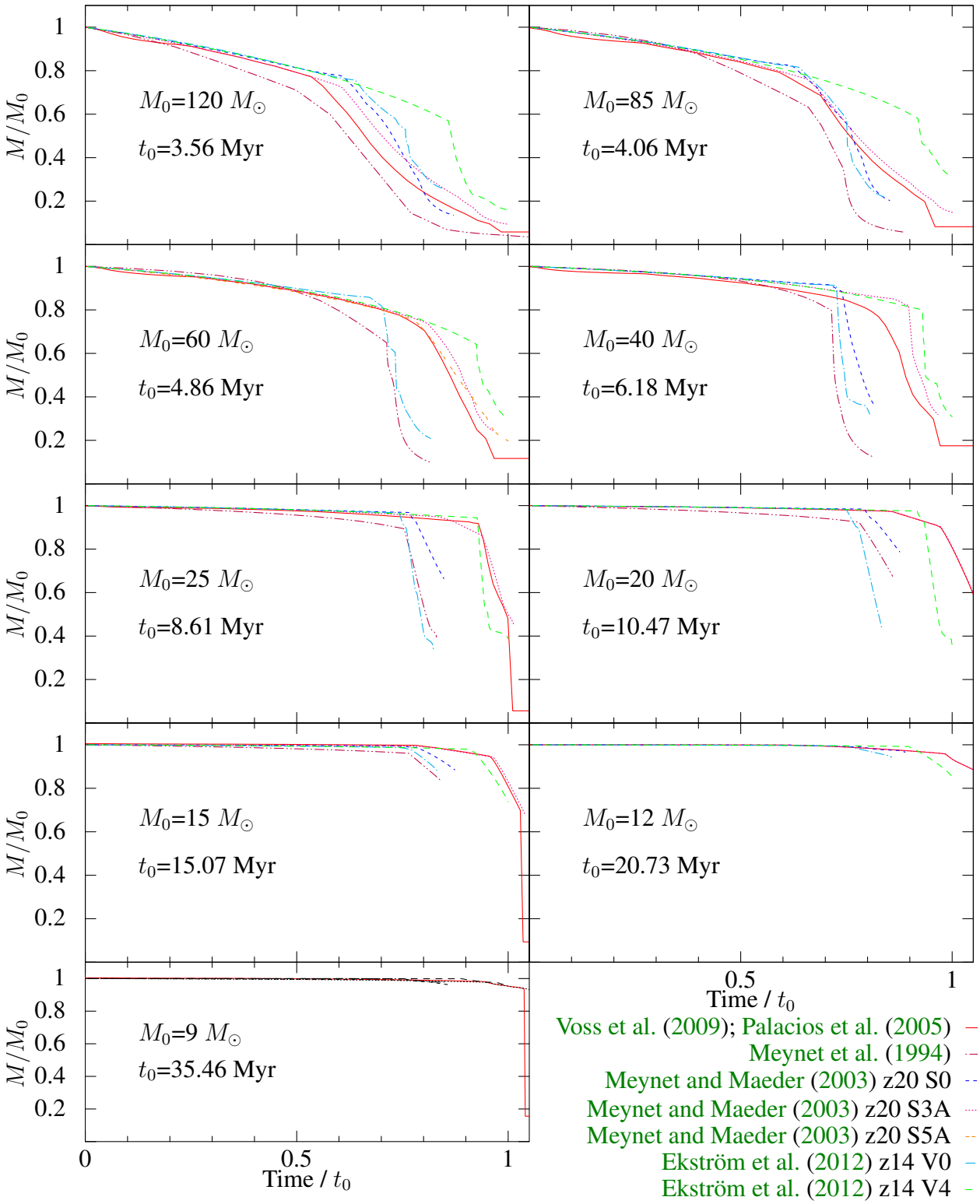


Figure 2.10: Time dependent stellar masses (M) in fractions of the initial mass (M_0). The times are normalized to the end times of the rotating Ekström et al. (2012) models (t_0).

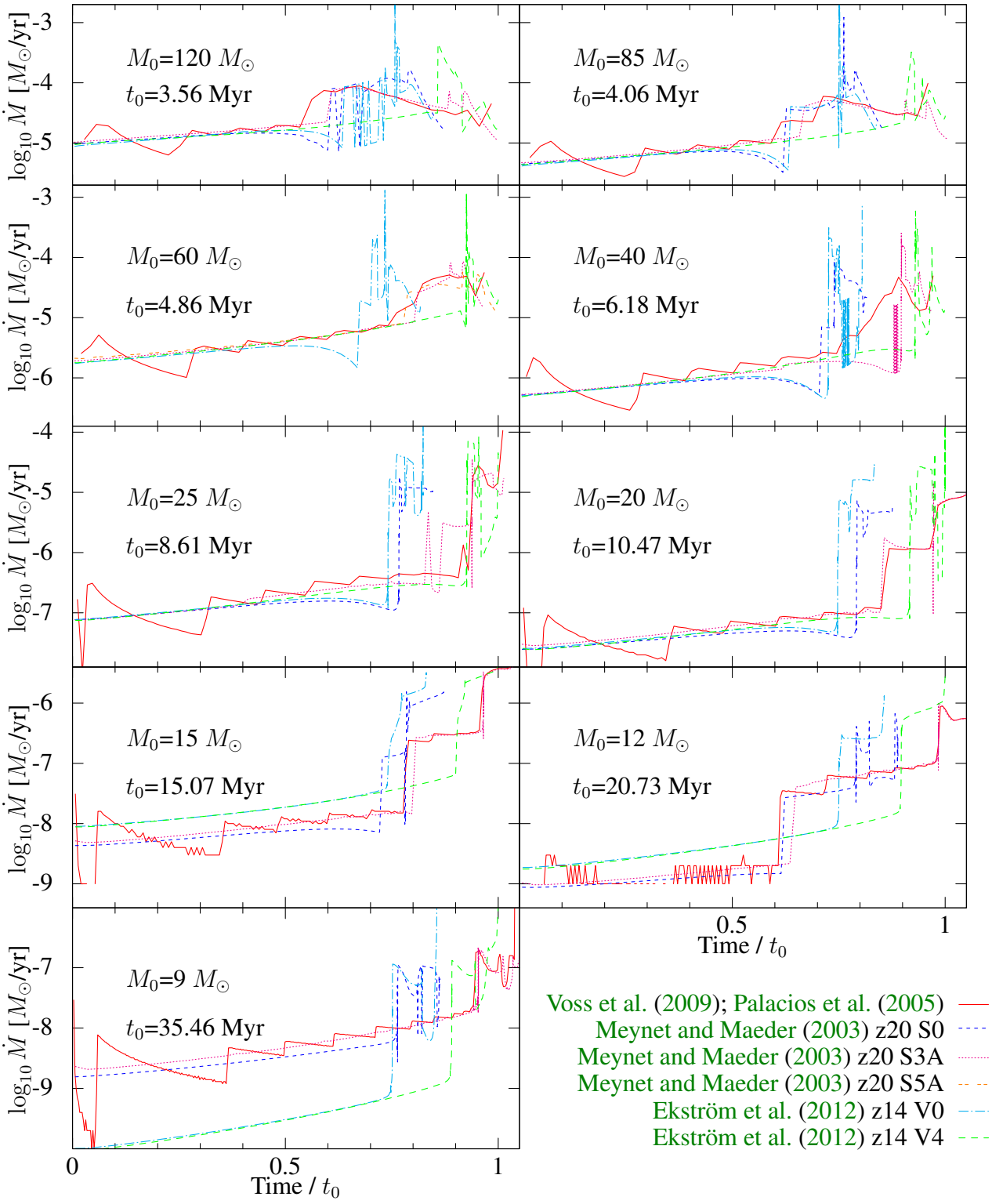


Figure 2.11: Logarithmic mass loss rates in solar masses per year. As in Fig. 2.10, times are normalized to the end times of the rotating Ekström et al. (2012) models (t_0) and a line plot was used for tabulated functions. Comparing the mass loss rates of Ekström et al. (2012); Meynet and Maeder (2003) to the mass loss rates used for Voss et al. (2009), which are based on stellar models of Palacios et al. (2005), shows artifacts of the interpolation between models in the latter. Therefore, we decided to preferentially use the rotating models of Ekström et al. (2012).

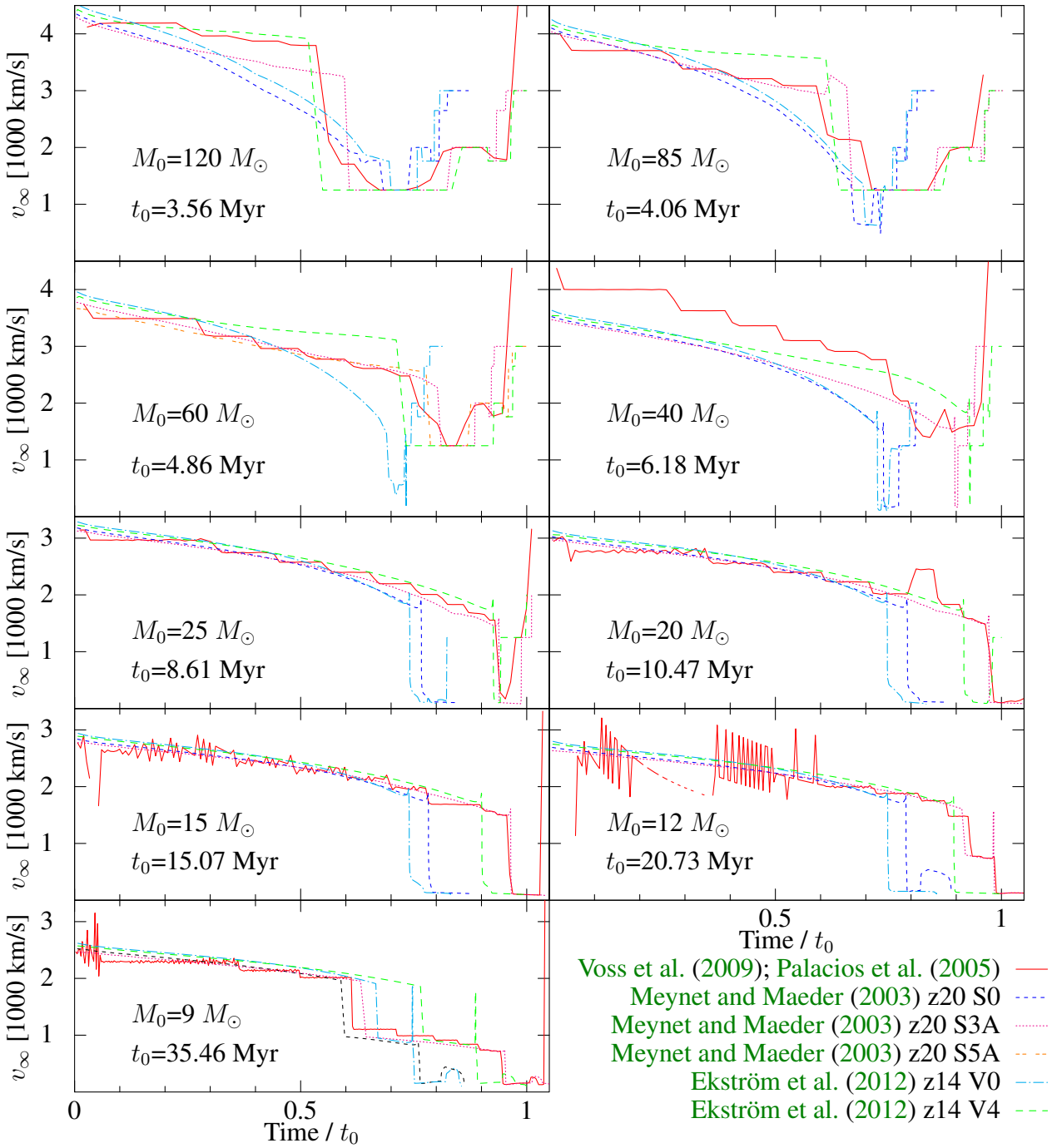


Figure 2.12: The terminal wind velocities (v_∞) shown in this figure were estimated from the surface abundances in the stellar evolution models with the classification criteria summarized in Table 2.4. Times were normalized to the end times of the rotating Ekström et al. (2012) models (t_0). The interpolated models used for Voss et al. (2009) show effects of interpolation between stellar tracks.

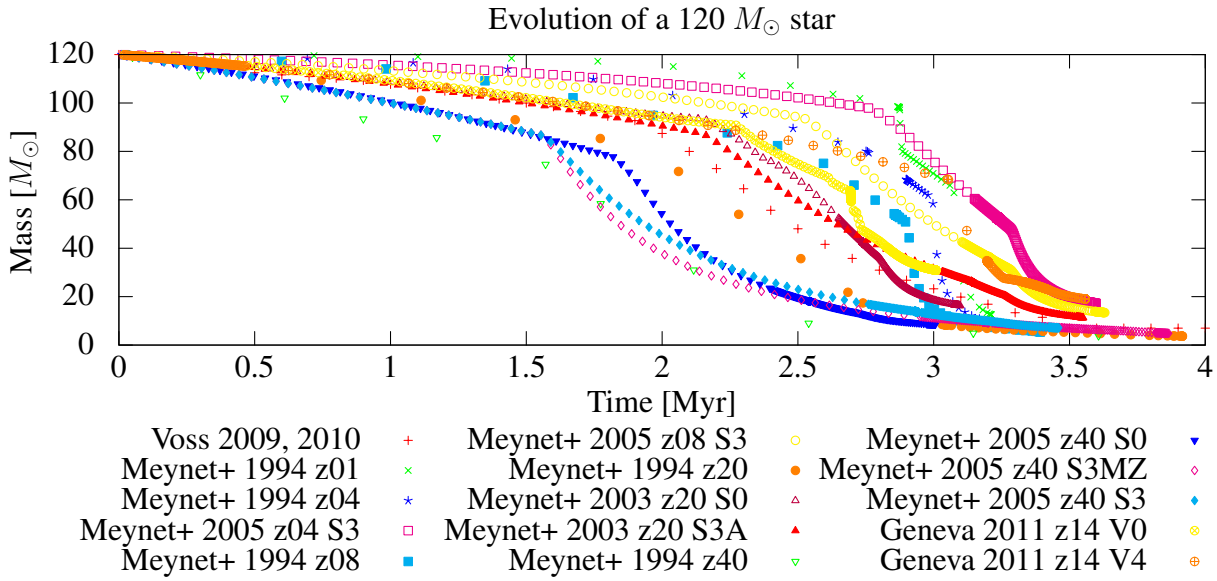


Figure 2.13: Evolution of the mass of an 120 $M_\odot$ star in the Geneva models (Sect. 2.7.1) with different metallicities and rotation velocities compared to the raw data used for Voss et al. (2009, 2010). The feedback of very massive stars plays a crucial role for the feedback of an OB association. Few models with very massive stars (1-2%) have a large influence the average feedback (factor 2-3).

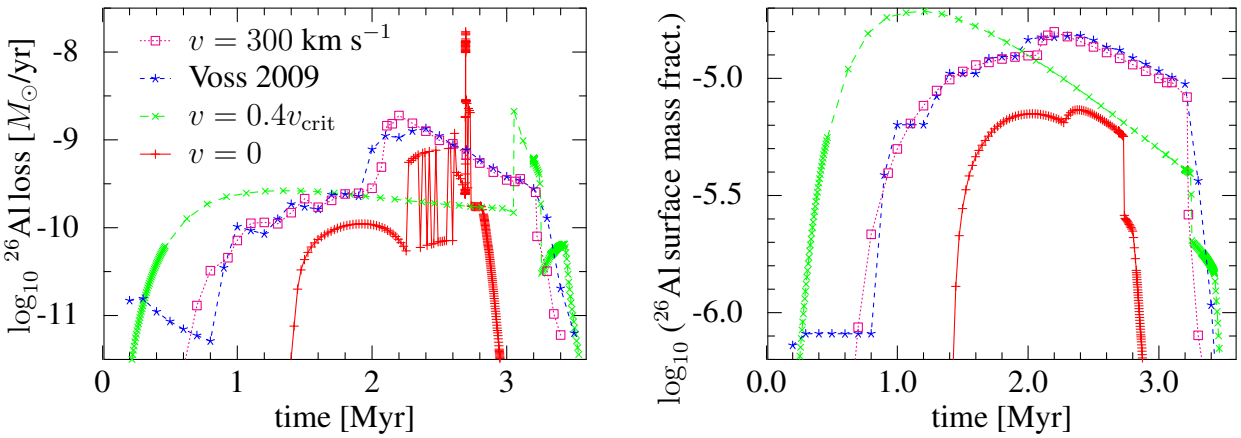


Figure 2.14: ^{26}Al feedback of a 120 $M_\odot$ star. The non-rotating model and the rotating model with $0.4v_{\rm crit}$ were taken from Ekström et al. (2012). The model with an initial rotation of 300 km s^{-1} was extracted from Palacios et al. (2005, Fig. 1). This shows that the mass fraction in the isochrones used in Voss et al. (2009) is relatively unaffected by the interpolation. The differences in the total output are caused by the problems with the mass loss rate.

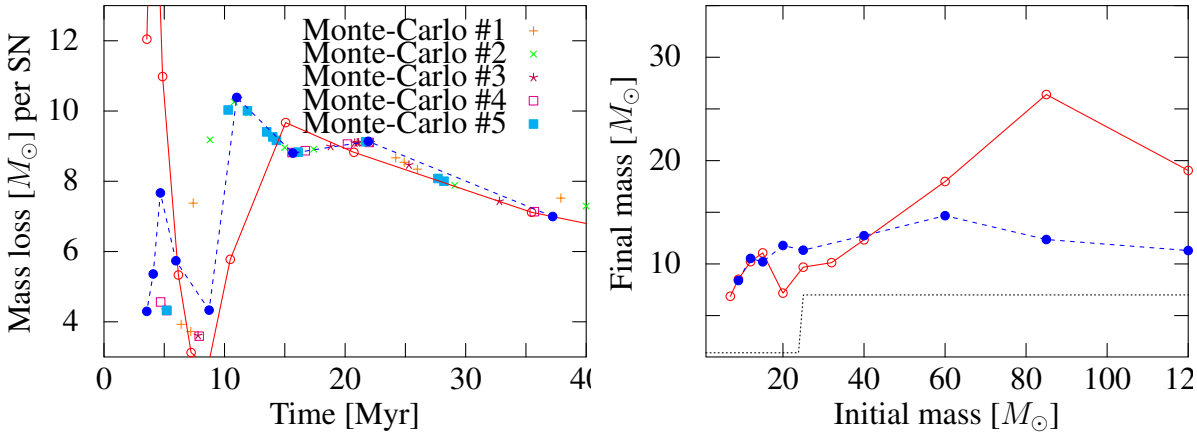


Figure 2.15: Left plot: mass loss per SN event against time between ZAMS and SN event (which is larger for smaller initial masses). The lines connect the rotating models from the stellar evolution grids and the dots are SN mass losses from individual Monte-Carlo realizations. The remnant masses are assumed to be 1.4 $M_\odot$ for stars with initial masses below 25 $M_\odot$ and 7 $M_\odot$ for more massive stars. The right plot shows the interrelation between the initial mass and the final mass or the remnant mass. The Monte-Carlo data is not displayed in this plot, since the information on the initial masses of the stars in the Monte-Carlo realization was not stored.

can be obtained by multiplying the number of stars with their mass. Elmegreen did not use an upper mass limit (i.e. $\infty^{-0.35} = 0$ and $\infty^{-1.35} = 0$ at the upper boundary of the intervals) and thus the constant can be eliminated from $1 = k_1(0 - M_{\rm max}^{-1.35})/(-1.35)$ and $M_{\rm cluster} = k_1(0 - M_{\rm min}^{-0.35})/(-0.35)$. This leads to

$$M_{\rm cluster} = \frac{1.35 M_{\rm min}^{-0.35}}{0.35 M_{\rm max}^{-1.35}} \quad .$$

This can be rewritten to

$$M_{\rm cluster} \sim 3 \times 10^3 \left(\frac{M_{\rm max}}{100 M_\odot} \right)^{-1.35} M_\odot \quad .$$

This mass limit cannot explain the high mass cut-off seen in observations.
A more recent multi component power law IMF has been published by Kroupa (2001). However, the most massive stars in clusters are still an unsolved problem (Weidner et al., 2010, 2013). The polynomial fit of Weidner et al. (2013) shows that for a cluster mass of 2 000 $M_\odot$ derived from a 8% star formation efficiency (Murray, 2011) and the median cloud mass in the Milky Way radio cloud survey of 2.5×10^4 $M_\odot$ the most massive star is expected to have ~ 60 $M_\odot$.

As explained in Weidner et al. (2013) random sampling of the IMF to find the most massive star in the association leads to results contradicting the observations. But even if our estimate of mass of the most massive star in the association would be too high, our conclusions on the individual contributions of association's stars to the combined feedback of the association will still stay valid.

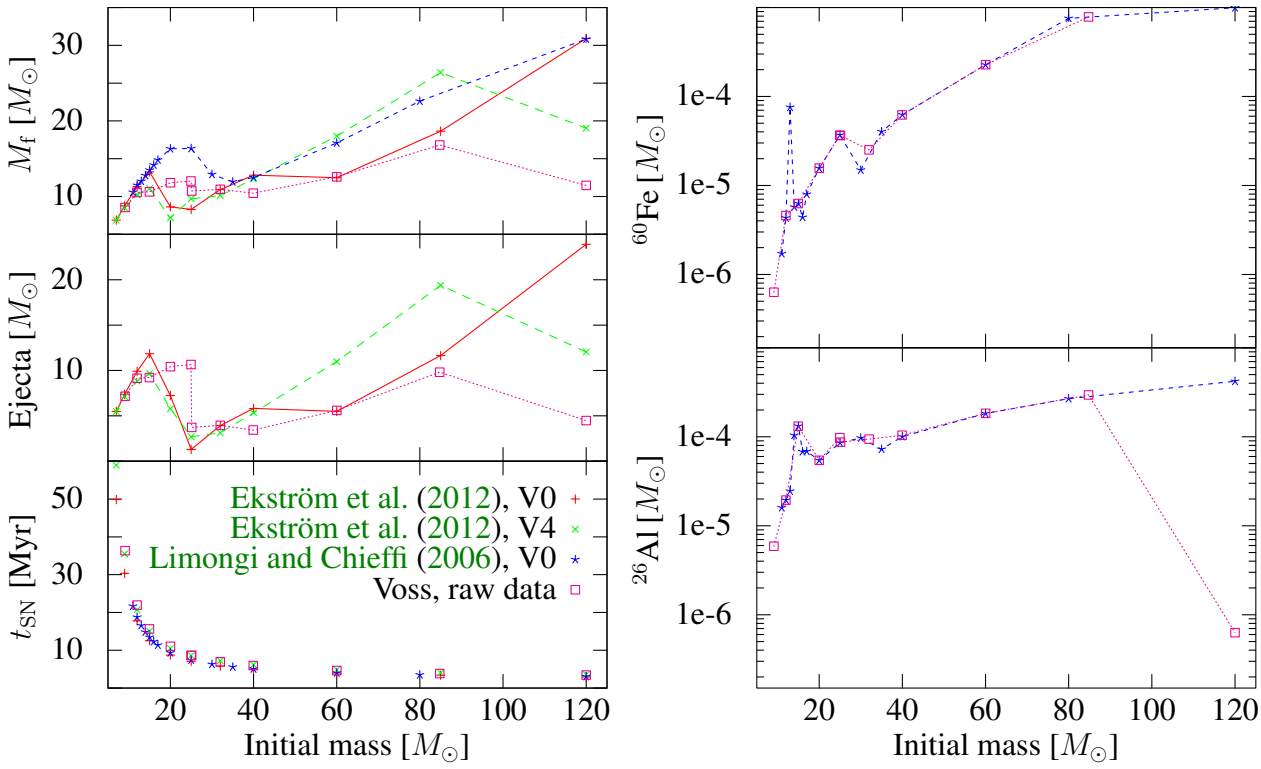


Figure 2.16: SN yields: final masses of the stellar evolution models (M_f, top left), SN ejecta (center, left) and times of the SN explosions (t_{SN}, bottom left). The data labeled Ekström et al. (2012) was combined with canonical remnant masses to compute the mass of the SN ejecta. These models do not include ^{60}Fe and ^{26}Al produced during the SN explosion. Thus, the amount of trace elements ejected in the SN (right panels) was taken from Tab. 2 and 3 of Limongi and Chieffi (2006). The data set "Voss, raw data" was used for the population synthesis models published in Voss et al. (2009, 2010).

We assume that our association of massive stars contains 10 stars in the mass range [8-120] $M_\odot$. According to Salpeter (1955) 10 stars in the mass range [8:120] $M_\odot$ have a total initial weight of 194 $M_\odot$:

$$\frac{10}{\int_8^{120} M^{-2.35}\mathrm{d}M}\int_8^{120} MM^{-2.35}\mathrm{d}M = 194\,M_\odot \quad .$$

Now it is possible to find 10 mass intervals $[n_i, n_{i+1}]$ with $i = 0, 1, ..., 10$ and $n_o = 8\ M_\odot$ in this range with the same cumulative distribution function (CDF) $F = 0.1 \times (8^{-1.35} - 120^{-1.35})$ via $n_i = (n_{i-1}^{-1.35} - F)^{-1/1.35}$. This is shown in Fig. 2.7.5. These intervals are the so called deciles. The masses M_i in these intervals are:

$$M_i = \frac{10}{\int_{n_i}^{n_{i+1}} M^{-2.35}\mathrm{d}M}\int_{n_i}^{n_{i+1}} MM^{-2.35}\mathrm{d}M \quad .$$

Table 2.5 lists M_i for the Salpeter (1955) IMF and the Kroupa (2001) IMF. Since there are only published Geneva models (Ekström et al., 2012) for distinct masses (7, 9, 12, 15, 20, 25, 32, 40, 60, 85 and 120 $M_\odot$) a first guess is to use $3 \times 9\ M_\odot$, $2 \times 12\ M_\odot$, $2 \times 15\ M_\odot$, $1 \times 20\ M_\odot$, $1 \times 32\ M_\odot$ and

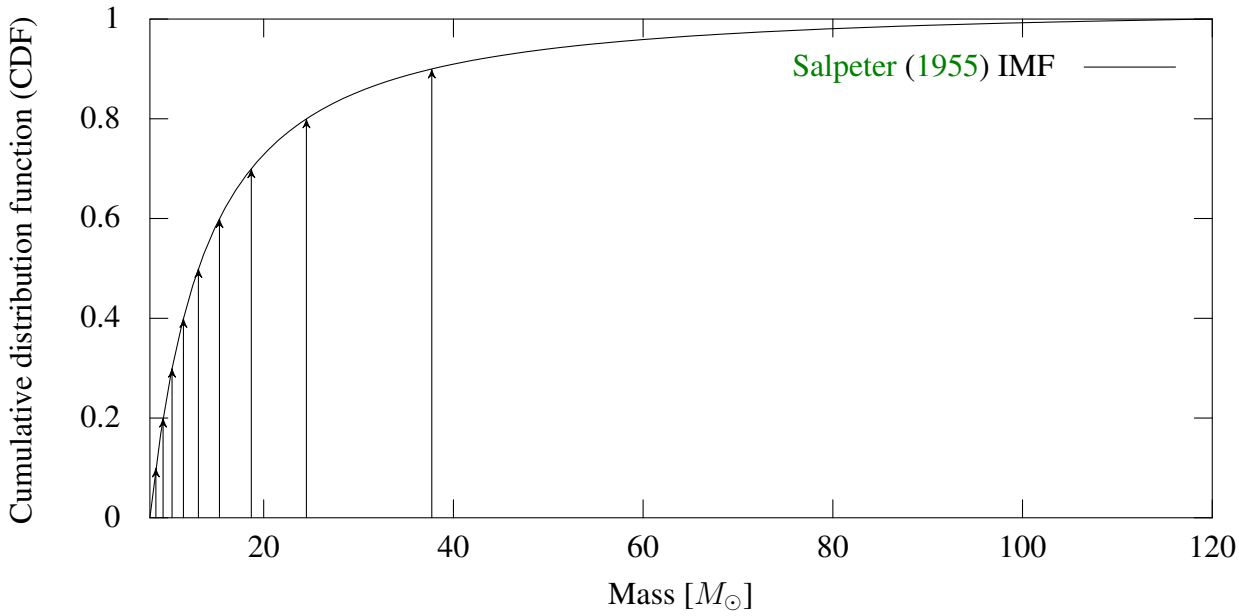


Figure 2.17: Deciles of the cumulative distribution function (CDF) of 10 stars in the mass range [8:120] $M_\odot$ with a Salpeter (1955) IMF. The vertical arrows indicate the boundaries between the mass intervals.

$1 \times 60\ M_\odot$. This leads to a total mass of 193 $M_\odot$ instead of the expected 194 $M_\odot$ from a Salpeter (1955) IMF or the expected 199 $M_\odot$ from a Kroupa (2001) IMF. The feedback of this model is shown in Fig. 2.18. From this figure it becomes evident that the most massive, still existing star dominates the feedback (e.g. Oey, 2005, also mentions this). Therefore, we conclude that during the first 4.8 Myr the feedback of a 60 $M_\odot$ star is a good first approximation for the feedback of our association.

Our first guess model assumes that no star in the association is more massive than 60 $M_\odot$. The probability for this is

$$p = \frac{\int_8^{60} M^{-2.35}\mathrm{d}M}{\int_8^{120} M^{-2.35}\mathrm{d}M} = 95\% \quad .$$

The most energetic feedback for stars with Geneva models and a total mass of approximately 200 $M_\odot$ can be obtained by using two stars: 120 $M_\odot$ and 85 $M_\odot$ (= 205 $M_\odot$), which is, however, not very likely. The probability that all stars with masses in [8:120] $M_\odot$ are more massive than 85 $M_\odot$ is

$$p = \frac{\int_{60}^{120} M^{-2.35}\mathrm{d}M}{\int_8^{120} M^{-2.35}\mathrm{d}M} = 1.6\% \quad .$$

Assuming that all mass is found in stars with 9 $M_\odot$ would lead to the weakest feedback. Also this scenario in not very likely. The probability that all stars are less massive than 9 $M_\odot$ is

$$p = \frac{\int_8^{9} M^{-2.35}\mathrm{d}M}{\int_8^{120} M^{-2.35}\mathrm{d}M} = 15\% \quad .$$

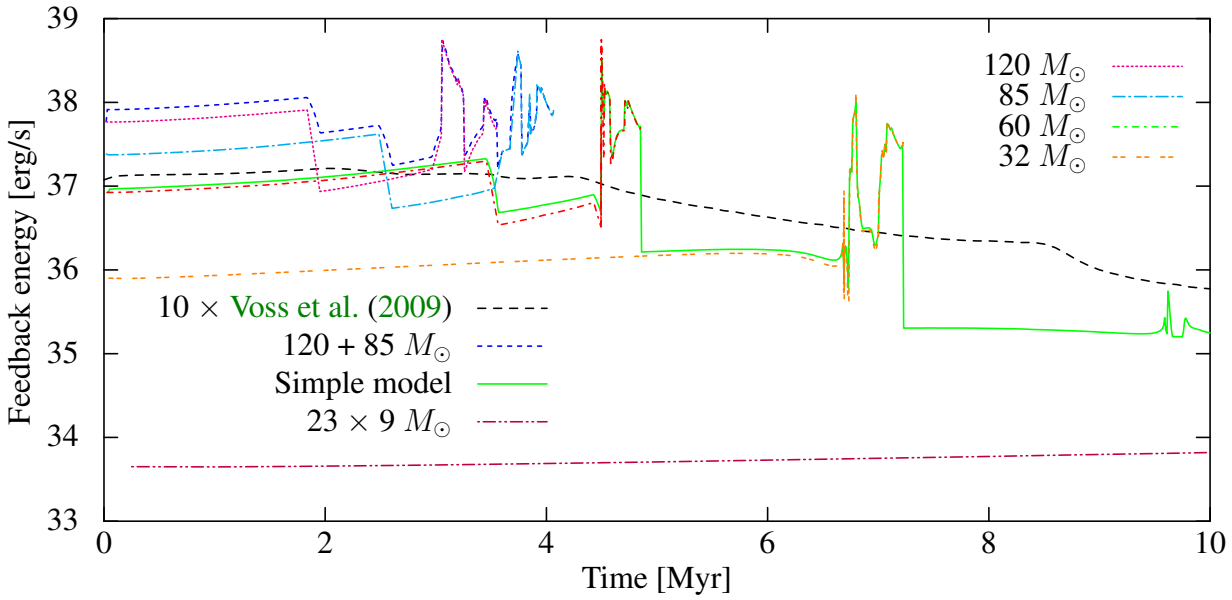


Figure 2.18: Comparison of the "first guess" model ($3\times9\ M_\odot$, $2\times12\ M_\odot$, $2\times15\ M_\odot$, $1\times20\ M_\odot$, $1\times32\ M_\odot$ and $1\times60\ M_\odot$) to the Voss et al. (2009) feedback model, feedback of individual massive stars as well as the highest energy model (85 $M_\odot$ and 60 $M_\odot$) and the lowest energy model (all mass in 9 $M_\odot$ stars). The "simple" model is much less continuous than the Voss et al. (2009) model. Also a realistic OB association with ~ 10 massive stars is expected to have a feedback that is strongly varying with time – the number of stars is too low to smooth the winds and there are peaks when the SN explode (smeared out over 0.1 Myr in the Voss et al. (2009) model).

This corresponds to ~ 21 stars if the expected mass for an IMF with 10 stars above 8 $M_\odot$ is used or ~ 23 stars if one prefers to use the a similar mass as for the most energetic case.
To summarize, we used the IMF to find a "first guess model", which can be constructed from the available data in the Geneva grid of stellar evolution models. Moreover, the feedback in this model is dominated by the most massive still existing star.

Kolmogorov Smirnov test for the "first guess" model

In the Kolmogorov Smirnov test the maximal deviation of the CDF of a sorted sample (S(M)) from the CDF of the IMF (F(M)) is computed. The result of this test is that our "first guess model" (Figure 2.18) for the mass distribution is in good agreement with the CDF of the IMF. Table 2.6 shows that the maximal deviation is so small that the null hypothesis cannot be rejected. As a comparison this test is also shown for the unlikely cases of high mass stars only or low mass stars only.

Salpeter (1955) IMF $\int_8^{120} M^{-2.35}\mathrm{d}M$		Kroupa (2001) IMF $\int_8^{120} M M^{-2.3}\mathrm{d}M$ $1 < m/M_\odot$		first guess model
Mass	Interval	Mass	Interval	Mass
M_1= 8.3 $M_\odot$	[8.0, 8.6] $M_\odot$	M_1= 8.3 $M_\odot$	[8.0, 8.7] $M_\odot$	9 $M_\odot$
M_2= 9.0 $M_\odot$	[8.6, 9.4] $M_\odot$	M_2= 9.0 $M_\odot$	[8.7, 9.4] $M_\odot$	9 $M_\odot$
M_3= 9.8 $M_\odot$	[9.4,10.3] $M_\odot$	M_3= 9.9 $M_\odot$	[9.4,10.4] $M_\odot$	9 $M_\odot$
M_4=10.9 $M_\odot$	[10.3,11.5] $M_\odot$	M_4=11.0 $M_\odot$	[10.4,11.7] $M_\odot$	12 $M_\odot$
M_5=12.3 $M_\odot$	[11.5,13.1] $M_\odot$	M_5=12.5 $M_\odot$	[11.7,13.3] $M_\odot$	12 $M_\odot$
M_6=14.2 $M_\odot$	[13.1,15.3] $M_\odot$	M_6=14.4 $M_\odot$	[13.3,15.7] $M_\odot$	15 $M_\odot$
M_7=16.9 $M_\odot$	[15.3,18.7] $M_\odot$	M_7=17.3 $M_\odot$	[15.7,19.2] $M_\odot$	15 $M_\odot$
M_8=21.3 $M_\odot$	[18.7,24.5] $M_\odot$	M_8=21.9 $M_\odot$	[19.2,25.3] $M_\odot$	20 $M_\odot$
M_9=30.0 $M_\odot$	[24.5,37.7] $M_\odot$	M_9=31.1 $M_\odot$	[25.3,39.2] $M_\odot$	32 $M_\odot$
M_{10}=61.3 $M_\odot$	[37.7,120.0] $M_\odot$	M_{10}=63.2 $M_\odot$	[39.2,120.0] $M_\odot$	60 $M_\odot$

Table 2.5: Columns 1-4 show the deciles of the IMF. Column 5 lists the "first guess model".

n	M	S(M)	F(M)	$S(n_{i-1}) - F(n_i)$	$S(n_i) - F(n_i)$
1	85	0.5	0.98	**-0.98**	-0.48
2	120	1.0	1.0	0.00	0.02
1	60	0.5	0.96	**-0.96**	-0.44
2	120	1.0	1.0	0.00	0.04
1	9	0.1	0.15	-0.15	-0.05
2	9	0.2	0.15	-0.05	0.05
3	9	0.3	0.15	0.05	0.15
4	12	0.4	0.43	-0.13	-0.03
5	12	0.5	0.43	-0.03	0.07
6	15	0.6	0.59	-0.09	0.01
7	15	0.7	0.59	0.01	0.11
8	20	0.8	0.73	-0.03	0.07
9	32	0.9	0.87	-0.07	0.03
10	60	1.0	0.96	-0.06	0.04
1	9	0.05	0.15	0.15	-0.10
⋮					
20	9	1.00	0.15	**-0.85**	0.80

Table 2.6: This table shows the results of the Kolmogorov Smirnov tests for the four models discussed in Sect. 2.7.5. The models are separated by double lines. On top the OB association consists of a 85 $M_\odot$ and a 120 $M_\odot$ star. The second model shows an association with a 60 $M_\odot$ and a 120 $M_\odot$ star. The next model is the "first guess model" and the last model assumes that the whole OB association consists of 9 $M_\odot$ stars. The first column is the sort index of the stars by mass. The second column shows the stellar mass. S(M) is the CDF of a sorted sample and F(M) is the CDF from the IMF. Column 4 shows the difference between F(M) in the same line and S(M) from the line above (or zero for the first star). Column 5 contains the differences between S(M) and F(M) from the same line. The interpretation of this data is that the "first guess model" passed the test and that the other models (shown as a reference) are quite unlikely (see highlighted values).

Chapter 3

Method: hydrodynamic simulations of the ISM

In this work we carry out numerical simulations of the interstellar medium (ISM). This approach is based on several assumptions, which are discussed in this chapter. First of all, when the ISM is modeled, it is usually treated as a perfect gas – often also called "ideal gas". This approach ignores intermolecular forces (i.e. dipole interactions, friction, Van der Waals, Joule-Thomson coefficients, molecular excitations, ...), which is problematic in high density environments like inside stars or planets, but which is a good approximation in the ISM, where densities are much lower than in earth-based laboratories. The best laboratory-vacua (e.g. in the LHC at CERN[1] or for gravitational wave interferometers) have of the order of a million particles per cubic centimeter. For comparison, in our simulations the highest number densities rarely surpass few thousand particles per cubic centimeter. In the ideal gas approach, the gas consists of individual particles with ballistic motions. These particles have localized interactions only and thus move on straight lines until they undergo perfectly elastic two particle collisions.

For our assumption of an ideal gas we have to decide, if the fluid approximation is a valid approach to facilitate handling areas spanning hundreds of cubic parsecs in space containing a vast numbers of particles (see Sect. 3.1).

If the fluid approach is valid, we have to choose a discretization scheme (Sect. 3.2) and to consider the limits of the time step size (Sect. 3.3).

Another approximation we have to make is the choice of the set of conservation laws we will employ. Our description of nature is always a simple approximation, which – hopefully – covers the leading order effects. In our case this means to decide which forces have to be taken into account in the simulation to follow the most important processes. This encompasses the evaluation if shear forces, magnetic fields or gravity are important agents. All these processes can in principle be included, however, at considerable computational cost. This is discussed in Sect. 3.4.

Also some numerical pitfalls (e.g. stability of numerical schemes, conservative vs. primitive variables, ...) are briefly sketched in this chapter.

[1] see e.g. http://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/components/vacuum.htm

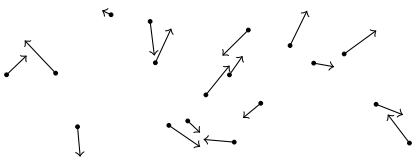


Figure 3.1: The hydrodynamic approach is based on the statistical treatment of large numbers of gas particles. This sketch shows individual particles (small dots) with individual velocities v_i (shown as vectors). A fluid approach can be applied if the particles in a given volume collide often enough to be described as a homogeneous gas blob with averaged quantities.

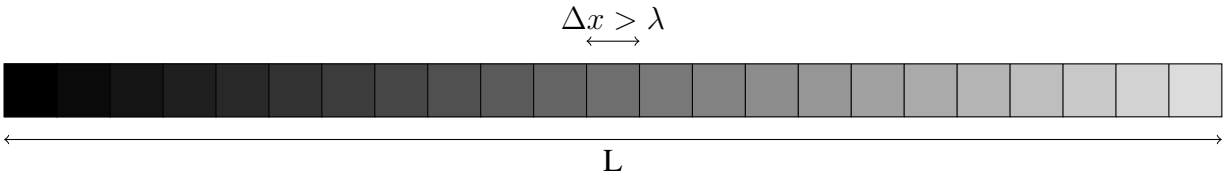


Figure 3.2: Discretization of continuous functions on a grid. The hydrodynamic approach is only applicable if the mean free path (λ) is much smaller than the scale of interest Δx and if the scale of the problem L is resolved with a sufficiently large number of grid cells.

3.1 Fluid approximation

To motivate the necessity of the hydrodynamic approach, we sketch a gas consisting of many particles with velocities $\vec{v}_i$ in Fig. 3.1. In our numerical hydrodynamic simulations we model large regions of space ($\sim 10^6$ pc^3) containing a huge number of gas particles. E.g. a number density of ~ 1 cm^{-3} leads to $> 10^{58}$ particles in a box of 10^6 pc^3. Thus, we cannot treat all gas particles individually. To tackle the evolution of a system with such a vast numbers of particles, we have to resort to the fluid approximation. This is an idealized concept, in which the gas is described statistically via averaged quantities, which vary with position (sketched in Fig. 3.2). For example, with a velocity field $\vec{v}\,(\vec{x}) = \langle \vec{v}_i \rangle\,(\vec{x})$, which is the average of the of individual particle velocities in a given volume, centered around a location $\vec{x}$ (at a given time). Later on, we will call such regions with averaged quantities and a length scale Δx "cells" or "fluid elements". The fluid approximation can only be used if the mean free path (λ) – i.e. the average distance a gas particle travels between two collisions – is much smaller than Δx:

$$\Delta x \gg \lambda = \frac{1}{\sigma n} \sim 10^6 \frac{T^2}{n} [\mathrm{cm}] \quad , \tag{3.1}$$

with the number density n and the interaction cross section σ.
For example, the mean free path of the warm ISM with a temperature of $T = 10^4$ K and a number density of $n = 1$ cm^{-3} is about $\lambda \sim 10^6 \frac{T^2}{n} = 10^{14}$ cm. This is several orders of magnitude below our resolution. An example where the fluid description of the gas should not be used is e.g. the solar wind near the earth. Assuming a temperature of $T = 10^5$ K and a number density of $n = 10$ cm^{-3}, this leads to a mean free path of $\lambda \sim 10^{15}$ cm (70 AU), which clearly shows that one should use a plasma description for this problem.
The length scales of the problem L, on which the continuous functions (e.g. density, pressure, ...) change, have to be even larger than the scales Δx, on which these averaged quantities are defined. To obtain meaningful averages, another important property is the saturation of forces. Forces which do not saturate e.g. gravity are treated as source terms in hydrodynamical simulations.

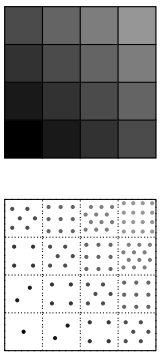


Figure 3.3: Illustration of the difference between the Eulerian (upper sketch) and the Lagrangian (lower sketch) discretization. The Eulerian view follows the evolution of averaged fluid quantities at fixed locations. The areas linked to these averaged quantities are color coded. In contrast, the smooth particle hydrodynamics (SPH) approach – which is a Lagrangian method – discretizes the initial conditions by grouping gas particles to fluid elements. During the simulation, it tracks the motions of these fluid elements. Regions of denser gas will thus be sampled with more fluid elements than sparse regions. However, this sketch is for illustration only, since an SPH simulation would set up the initial positions the fluid elements in a more sophisticated way (e.g. glass configuration).

3.2 Spatial discretization

Basically, there are two ways to follow the evolution of the fluid. The system properties (U), which are often gathered in a vector containing the information on the density, flow velocity and pressure of the fluid, can be followed in the Eulerian or Lagrangian picture (sketched in Fig. 3.3).
In the Lagrangian view, which is used in the smooth particle hydrodynamics (SPH) approach where individual fluid particles represent fluid elements, the observer moves along with the fluid element. The temporal changes of the system quantities are described by the "convective" or "Lagrangian" derivative $\left(\frac{\mathrm{D}U}{\mathrm{D}t}\right)$:

$$DU_i/Dt = \underbrace{\partial U_i/\partial t}_{\text{Eulerian}} + \underbrace{\vec{v}\cdot\vec{\nabla}U_i}_{\text{convective derivative}} \qquad \text{with } \vec{v}\cdot\vec{\nabla}U_i = (v_x\nabla_x + v_y\nabla_y + v_z\nabla_z)U_i \qquad .$$

The advantage of comoving coordinates is the feasibility of tracing the evolution of individual fluid elements. However, this method has problems with sparse regions, turbulence and artificial surface tension.
In the Eulerian view, which is the strategy in most grid codes[2], the temporal evolution of fluid is described by time dependent system properties at fixed locations (resp. in fixed volumes). Thus, temporal changes of the system quantities are followed via the partial time derivative $\frac{\partial U}{\partial t}$. The simulated region is subdivided into sub-volumes – which will be called "cells" further on. Individual gas particles can leave or enter such volumes. Technically, they are treated via averaged quantities. At every time step and at each cell interface, the Riemann problem (see Sect. 3.5) is solved. This checks, if fluxes across this interface occur and how large they are. Subsequently the averaged quantities are changed according to these fluxes. In grid codes, the resolution of the simulation can be either set before starting the simulation by a fixed mesh or it can be adjusted via adaptive mesh refinement (AMR), which can subdivide or merge cells during the run to better resolve "interesting areas"– e.g. near strong gradients (see Sect. 3.9).
This work uses grid codes, as we are interested in ^{26}Al in the dilute medium. Since the resolution in SPH follows the gas density, the SPH method is not well suited for the purpose of this study.

3.2.1 Setting up a grid code simulation

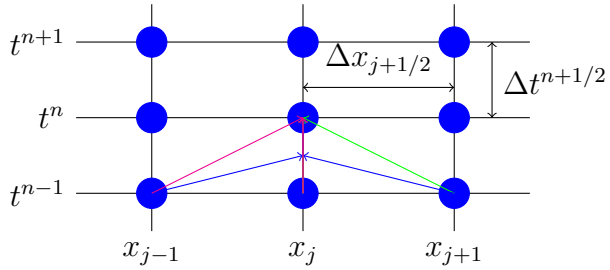
At the start of a grid code simulation, the initial state of the system has to be defined. This state of the medium is called "initial conditions (IC)" and it will typically be described by continuously

[2]Grid codes are also sometimes called "mesh codes". Moving mesh codes do not stick to the Eulerian view.

varying quantities. Most commonly, the IC are specified with so-called primitive variables: density, velocity and pressure. In principle, these quantities could also be used to calculate fluxes in the simulations. However, for the compliance with conservation laws, it is advantageous to use the conserved quantities (see Sect. 3.4), density, momentum and total energy, to calculate fluxes. This way, fluxes across cell interfaces will violate the conservation laws only at the level of number precision, which is much less than errors in fluxes based on primitive variables would violate the conservation laws. Typically, a grid code working with conservative quantities will convert them into primitive variables at each time step and for each cell. This is necessary for the treatment of ISM physics like radiative cooling (where densities and temperatures matter) and also for statistics like e.g. kinetic and thermal energy fractions. Moreover, the code will try to use units in which the conservative variables in most of the cells are in the order of unity, since this is advantageous for the numerics. The conversion factors between these units and cgs or SI units are taken account for the calculation of gravity or ISM physics like e.g. cooling.
Depicting these quantities on a mesh is called "discretization". It is connected with the loss of information, which causes discretization errors. Technically, the simulation will evolve the medium inside a box of given size and geometry. This volume has to be subdivided into grid cells. During this discretization process, averaged system properties – either conservative or primitive – are calculated for each cell: density, momentum or velocity, total energy or pressure. Additionally, it is possible to use so-called passive scalars for properties like metallicity. Passive scalars are advected across cell boundaries with same velocity as the carrier flow. However, the ratio of the concentrations of the passive scalar in the two cells can be different from the density ratio. There are several possible ways how the system quantities can be discretized: in finite difference (FD) methods, values at grid centers and at grid faces are stored. Typically, a so-called staggered mesh is employed, which usually stores scalars, like densities and pressures, at the grid cell centers and quantities containing directional information, like velocities, at the cell interfaces. We will use a more sophisticated approach, which makes use of the conservation laws: The discretization via finite volumes (FV) discretizes the conservative equations and stores cell averages. One can also store terms of the orthogonal series expansion of the velocity. This would then be a so-called finite element (FE) method. Thus, FV is a very low order FE method.
To specify the treatment of gas at the boundaries of the region, boundary conditions (BCs) have to be set. Typical choices are reflective boundaries, outflow boundaries or periodic boundaries. However, some problems might require more sophisticated BCs, e.g. with inflows or fixed states of boundary cells.

3.2.2 Geometry of grid code simulations

A good choice of the grid can substantially lower the computational cost. Since simulations become more computationally intensive when the number of dimensions in increased, symmetries in the problem setup can motivate the choice of a cylindrical or a spherical grid instead of a Cartesian grid. Such symmetries are also exploited in this work: we follow interesting processes first in spherically symmetric 1D models (if possible), since this way, sampling a large region in parameter space is feasible. Subsequently, interesting regions of parameter space are re-simulated in more dimensions to take deviations from spherical symmetry into account.
Generally, if very high resolution is required, problems are often treated in 1D, 2D or 2.5D first. Here, 2.5D means spherical polar coordinates (r, θ, ϕ), with reflecting BC at the equator and at the polar axis. This configuration assumes rotational symmetry around the polar axis. It performs 2D



Downwind $\partial_x (u)_j^n = \left(u_{j+1}^n - u_j^n\right) / (\Delta x)$

Centered $\partial_x (u)_j^n = \left(u_{j+1}^n - u_{j-1}^n\right) / (2\Delta x)$

Upwind $\partial_x (u)_j^n = \left(u_j^n - u_{j-1}^n\right) / (\Delta x)$

Figure 3.4: Finite differencing basics. The columns depict points in space (x_j) where the solution (u_j^n) is evaluated. The rows contain these solutions at different times (t^n). The sketched points are only a cutout of a larger grid. The full grid contains the initial conditions at the lowest row and boundary conditions in the first and last column. The sketch shows different discretizations of derivatives, which are mathematically equivalent in the limit of infinitely small grid spacing.

simulations in the meridional (r, θ) plane but 3D simulations in $\vec{v}, \vec{B}$.
When the number of dimensions is lowered, one has to keep the consequences in mind: e.g. hydrodynamic instabilities will develop differently and in Cartesian coordinates different numbers of dimensions will lead to different evolutions of e.g. a Sedov-Taylor blast (since the number of dimensions is part of the exponent in the formulas for the temporal evolution of radius and velocity, as shown in Sect. 4.3).
For many simulations an evenly spaced grid is a very good choice. It can, however, become necessary to better resolve regions with steep gradients if a high resolution of the whole volume becomes computationally too expensive. Possible solutions are AMR (see Sect. 3.9) or non-evenly spaced grids with (adaptive) cell sizes depending on gradients of e.g. pressure or density. In this work, we use evenly spaced grids and introduce – if necessary – evenly spaced meshes inside cells via AMR by splitting the cell into 2^ν cells, where ν is the number of dimensions.

3.3 Time discretization and von Neumann stability analysis

Figure 3.4 sketches the simulation process: the solution (u_j^n) at the locations (columns, x_j) is advanced in time (rows, t^n). The sketched points are a cutout of a larger net. In the full grid, the lowest row would contain the initial conditions. Furthermore the solution at the leftmost and rightmost points in the full grid (first and last column) follows from the boundary conditions. We now take a step back from finite volumes to finite differences and examine different ways to calculate derivatives. The colored lines in this sketch show different – in the limit of very high resolution mathematically equivalent – discretizations of derivatives. The points used in the derivative form the computational stencil of the method. At the first and last point in the grid, missing information in the stencil is replenished by the boundary condition. However, not all these discretizations of the derivative would lead to good results in a simulation. The suitability of these prescriptions can be checked by applying them to a wave solution. Schemes with growing amplification factors ($|\xi| > 1$), where $\xi(k)$ is the wave-number dependent amplification factor, are unstable and thus not suitable for numerical simulations. This is the basic concept of the von Neumann stability analysis.
For illustration, we will apply this method to two different discretizations: to the forward-time, centered-space method (blue lines in Fig. 3.4) and the forward-time, upwind scheme (purple line in

Fig. 3.4). In the von Neumann stability analysis solutions of the type $u_j^n = e^{ikj\Delta x \pm i\omega n\Delta t} = \xi^n e^{ikj\Delta x}$ are used. Two single frequency solutions to the 1D wave equation, $u(x,t) = \cos(kx \pm \omega t)$ and $u(x,t) = \sin(kx \pm \omega t)$ with the wave number $k = \omega/c$, are combined in this complex exponential form.
The Taylor series for the forward-time, centered-space method (blue lines in Fig. 3.4) leads to

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{2\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right) \quad .$$

Von Neumann stability analysis shows that this scheme is unconditionally unstable:

$$\begin{aligned}
\xi^{n+1}e^{ikj\Delta x} &= \xi^n e^{ikj\Delta x} - \frac{c\Delta t}{2\Delta x}\left(\xi^n e^{ik(j+1)\Delta x} - \xi^n e^{ik(j-1)\Delta x}\right) \\
\xi &= 1 - \frac{c\Delta t}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) \\
\xi &= 1 - i\frac{c\Delta t}{2\Delta x}\sin(k\Delta x) \\
|\xi|^2 &= 1 + \left(\frac{c\Delta t}{2\Delta x}\right)^2 \sin^2(k\Delta x) \quad .
\end{aligned}$$

The physics behind this is that information should only come from the direction of the flow.
An example for a scheme with stable regions is the forward-time, upwind scheme, where information is only allowed to propagate from the nearest upwind neighbors. The Taylor series is

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{\Delta x}\left(u_j^n - u_{j-1}^n\right) \quad .$$

Here von Neumann stability analysis leads to

$$\begin{aligned}
\xi &= 1 - \frac{c\Delta t}{\Delta x}\left(1 - e^{-ik\Delta x}\right) \\
\xi &= 1 - \frac{c\Delta t}{\Delta x}\left(1 - \cos(k\Delta x) + i\sin(k\Delta x)\right) \\
|\xi|^2 &= 1 - 2\frac{c\Delta t}{\Delta x}\left(1 - \cos(k\Delta x)\right) + \left(\frac{c\Delta t}{\Delta x}\right)^2\left(1 - 2\cos(k\Delta x) + \cos^2(k\Delta x) + \sin^2(k\Delta x)\right) \\
|\xi|^2 &= 1 - 2\frac{c\Delta t}{\Delta x}\left(1 - \frac{a\Delta t}{\Delta x}\right)\left(1 - \cos(k\Delta x)\right) \quad .
\end{aligned}$$

This is stable for $0 \leq \frac{c\Delta t}{\Delta x} \leq 1$. This can be rewritten as $\Delta t \leq \Delta x/c$, which is the Courant-Friedrichs-Lewy (CFL) condition (Courant et al.), used for the time step size in all our simulations in order to stay in the stable regime. The time step Δt must be less than the time to cross a cell at speed c. This is necessary to ensure that information from outside the stencil does not have enough time to reach the point x_j. The new solution u_j^{n+1} must take input from all points at t^n within the domain of dependence of x_j^{n+1} into account. The CFL also applies to finite volume methods, where similar arguments based on the domain of dependence can be made – in this case we would not draw the stencil but rather look at the characteristics (see Fig. 3.5 and its explanation in the text). However, also this would lead to the CFL and reflect the fact that gas must not cross more than a whole cell during one time step.
In conclusion, we have seen that schemes, which take the propagation of information properly into account, can lead to stable solutions. The next question is, which equations can describe the physics of the problem. We will delve into this problem in Sect. 3.4.

3.4 Hydrodynamic conservation laws (Euler equations)

The Euler equations[3] are a set of coupled non-linear hyperbolic[4] conservation laws, which can be used as a simplified model to describe the dynamics of compressible fluids. The Euler equations are only a first approximation. They neglect body forces (body forces – e.g. gravity – have to be added as source terms to the solution of the hyperbolic PDEs), viscosity (included in the Navier Stokes equations) or magnetic fields (treated in MHD). The Euler equations support sound waves and an entropy wave.
The derivation of the hydrodynamic conservation laws (Eq. 3.3 to 3.6) is based on the conservation of mass, energy and momentum. Using an integration of those quantities over a cell and Gauß's theorem, leads to the differential form of the Euler equations via the vanishing integrands. The detailed derivations can be found in any book on hydrodynamics, e.g. Shu (1992, Chapter 2 pages 20 to 23 and Chapter 4 pages 45 to 46).
In addition to these hydrodynamic conservation laws, pressure and energy have to be connected with an equation of state (EOS). This closure relation – i.e. the relation between pressure and energy – is required to solve the system of hydrodynamic conservation laws, since there are more unknowns than equations. A possible choice is the adiabatic[5] EOS of an ideal gas:

$$\text{Equation of State (EOS):} \qquad p = (\gamma - 1)\,\rho e_{\text{in}} \qquad \text{or} \qquad pV = N k_{\text{B}} T \quad . \tag{3.2}$$

The adiabatic exponent $\gamma = \frac{c_p}{c_V} = \frac{f+2}{f}$ is the ratio of the specific heats (c_p, c_V). It is a constant that depends on the degrees of freedom (f) in the chosen type of gas. In a monoatomic ideal gas the energy per degree of freedom is $k_{\text{B}}T/2$. Thus, for increasing the temperature we can write $c_V \Delta T = fk/2\Delta T$, which has been normalized with the number of particles N. If the volume has to be adjusted to keep the pressure constant, we find from the EOS that V is proportional to T and thus $p\Delta V = \frac{pV}{T}\Delta T = \frac{k_{\text{B}}T}{T}\Delta T$, which leads to $c_p = c_v + k$ and $\frac{c_p}{c_V} = \frac{f+2}{f}$. For monoatomic gases (e.g. atomic hydrogen, H I) the adiabatic exponent attains the value $\gamma = \frac{5}{3}$. Twoatomic gases and linear molecules (e.g. gases like air or H_2) have an adiabatic exponent of $\gamma = \frac{7}{5} = 1.4$. In the isothermal case (i.e. constant temperature) the pressure is a function of density only $P = c_s^2 \rho$.
$e_{\text{in}} = E/\rho - 0.5|\vec{v}^2|$ is the specific internal energy density. In Equation 3.2 e_{in} has to be multiplied with the density, since e_{in} is the internal energy per unit mass and not an internal energy volume density $E_{\text{therm}} = \rho e_{\text{in}}$.
The hydrodynamic equations without gravity and viscosity are:

$$\text{mass} \quad : \quad \frac{\partial \rho}{\partial t} + \frac{\partial \rho v_k}{\partial x_k} = 0 \tag{3.3}$$

$$\text{momentum} \quad : \quad \frac{\partial \rho v_i}{\partial t} + \frac{\partial \rho v_i v_k}{\partial x_k} = -(\gamma - 1)\frac{\partial \rho e_{\text{in}}}{\partial x_i} \tag{3.4}$$

$$\text{energy} \quad : \quad \frac{\partial E_{\text{tot}}}{\partial t} + \frac{\partial E_{\text{tot}} v_k}{\partial x_k} = -(\gamma - 1)\frac{\partial \rho e_{\text{in}} v_k}{\partial x_k} \tag{3.5}$$

$$\text{internal energy} \quad : \quad \frac{\partial \rho e_{\text{in}}}{\partial t} + \frac{\partial \rho e_{\text{in}} v_k}{\partial x_k} = -(\gamma - 1)\rho e_{\text{in}}\frac{\partial v_k}{\partial x_k} \quad . \tag{3.6}$$

[3] Strictly speaking, only Equation 3.8 is the Euler equation, but many authors call the whole system of partial differential equations (PDEs) Euler equations.

[4] A PDE for a function u(x,t) the form $Au_{tt} + 2Bu_{tx} + Cuxx + ... = 0$ is called hyperbolic, if $AC - B^2 < 0$. This kind of PDEs behaves like a wave equation and has real Eigenvalues. It describes a phenomenon with finite propagation speed.

[5] "adiabatic" means "no heat exchange with the environment".

In vector notation and with the EOS (Eq. 3.2), the system of these three conservation laws can be written as:

$$\begin{aligned} \partial_t \rho + \nabla \cdot (\rho \vec{v}) &= 0 && \text{[conservation of mass]} && (3.7)\\ \partial_t (\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v}) + \nabla p &= 0 && \text{[conservation of momentum]} && (3.8)\\ \partial_t E_{\text{tot}} + \nabla \cdot [\vec{v} (E_{\text{tot}} + p)] &= 0 && \text{[conservation of energy]} \quad . && (3.9) \end{aligned}$$

In this system of coupled nonlinear partial differential equations E_{tot} denotes the total energy density, p is the pressure, $\vec{v}$ is the velocity vector, ρ is the density, $\partial_a b = \frac{\partial b}{\partial a}$ are partial derivatives and $\otimes$ is the tensor product.
The technical terms "diffusive" and "convective" terms, often used in context of hydrodynamical simulations, refer to parts of such equations: A transport equation for a general flow quantity Φ and a diffusion coefficient Γ typically consists of four terms:

$$\underbrace{\partial_t \rho \Phi}_{\text{time}} + \underbrace{\nabla \cdot (\vec{v} \rho \Phi)}_{\text{convection}} = \underbrace{\nabla \cdot (\Gamma \rho \nabla \Phi)}_{\text{diffusion}} + \underbrace{S_\Phi}_{\text{source}} \quad .$$

The first term on the left side describes the net gain or net loss per unit volume and unit time. The convective term covers the downstream transport with velocity $\vec{v}$. A nonuniform spatial distribution of Φ leads to a diffusive term on the right hand side. All sources and sinks are collected in S_Φ.
Basically, the flow can be described either with the vector of primitive variables $\vec{W}^{\text{T}} = (\rho, \vec{v}, P)$ or with the vector of conservative variables $\vec{U}^{\text{T}} = (\rho, \rho\vec{v}, E)$. The latter is favorable for computations (see Sect. 3.2 where conservative methods like finite volumes are discussed), as it directly uses the conservation of mass, momentum and total energy.
Generally, a system of conservation laws can be written in a compact form using the flux vectors $\vec{F}_i\left(\vec{U}\right)$, the vector of conservative variables $\vec{U}$ and Einstein's summation convention:

$$\begin{aligned} \vec{F}_1^{\text{T}}\left(\vec{U}\right) &= \left(\rho v_1, \rho v_1^2 + p, \rho v_1 v_2, \rho v_1 v_3, v_1(E+p)\right)\\ \vec{F}_2^{\text{T}}\left(\vec{U}\right) &= \left(\rho v_2, \rho v_1 v_2, \rho v_2^2 + p, \rho v_2 v_3, v_2(E+p)\right)\\ \vec{F}_3^{\text{T}}\left(\vec{U}\right) &= \left(\rho v_3, \rho v_1 v_3, \rho v_2 v_3, \rho v_3^2 + p, v_3(E+p)\right) \end{aligned}$$

$$\partial_t \vec{U} + \partial_{x_i} \vec{F}_i\left(\vec{U}\right) = 0 \quad . \tag{3.10}$$

With the Jacobians of the flux functions $\vec{J}_i\left(\vec{U}\right) = \frac{\partial \vec{F}_i}{\partial \vec{U}}$ it can be rewritten as:

$$\partial_t \vec{U} + \vec{J}_i \partial_{x_i} \vec{U} = 0 \tag{3.11}$$

This system is hyperbolic if the matrix $\vec{J}$ has real and distinct Eigenvalues λ_i. Physically, the Eigenvalues represent velocities of propagation of information. The same type of system can be written down for the primitive variables.
An important concept for the numerical solution are the characteristic curves. These curves are possible trajectories of a signal in the space-time diagram. Fig. 3.5 shows this diagram for a hyperbolic PDE. The real Eigenvalues of this PDE correspond to wave families with finite speeds. These signal propagation velocities lead to a domain of dependence and a domain of influence.

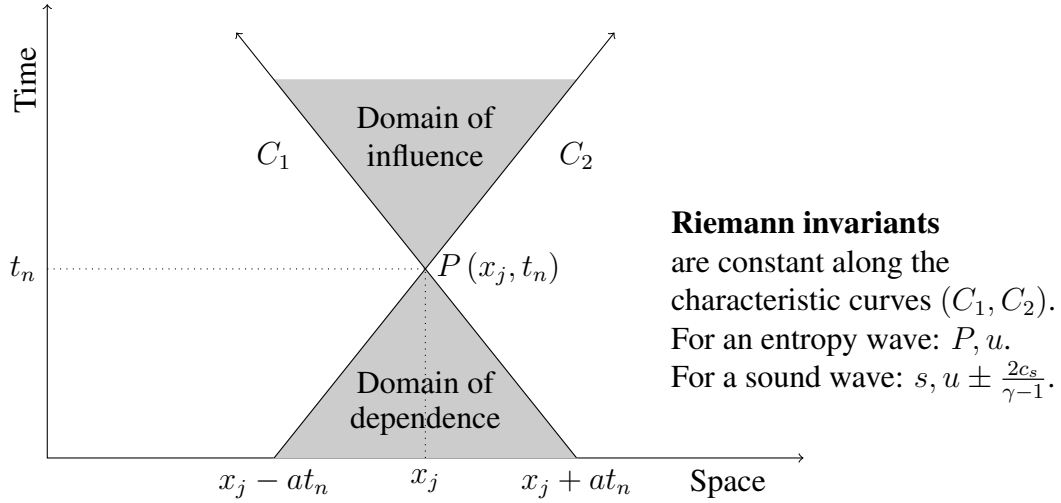


Figure 3.5: Hyperbolic partial differential equations have real Eigenvalues (λ) with the physical interpretation of finite wave speeds of different wave families. The characteristic curves (C_1, C_2) – which are possible wave trajectories (i.e. $x(t) = x_j + \lambda(t - t_n)$) – limit the domain of dependence and the domain of influence of the point $P\,(x_j, t_n)$ in the space-time diagram. The sketch also lists the Riemann Invariants for two wave families.

Since Eq. 3.11 will lead to a velocity of the wave $\lambda(\vec{U})$, which will be rather a function of the solution than constant, compression and expansion of the wave can be observed. For example, if the velocity increases for larger $\vec{U}$, the characteristics in the space-time plane are steeper in a region with smaller $\vec{U}$ than in a region with larger $\vec{U}$ (in this diagram the slope is indirectly proportional to the velocity). This is shown in Fig. 3.6. Diverging characteristics indicate a rarefaction fan – shown in Fig. 3.6 in the higher $\vec{U}$ region. The solution for such waves is discussed in Sect. 4.1.2. Converging characteristics lead to a shock – shown in Fig. 3.6 in the lower $\vec{U}$ region. The shock is a discontinuity and the integral forms of the conservation equations lead to the Rankine Hugoniot shock jump conditions, as shown in Sect. 4.1.3. If characteristics on both sides of an interface are parallel, a contact discontinuity can develop (see Sect. 4.1.1).

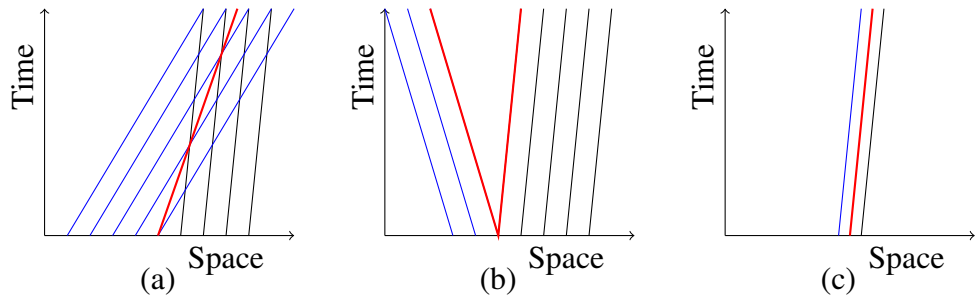


Figure 3.6: The dependence of the Eigenvalues on the vector of conservative variables leads to non-constant slopes of the characteristics. We assume two constant states. This leads to two sets of parallel characteristics shown in blue and black. They can interact in three ways: (a) a shock forms if characteristics of the same wave-type intersect; (b) diverging characteristics produce a rarefaction fan; (c) finally, regions with parallel characteristics can harbor a contact discontinuity.

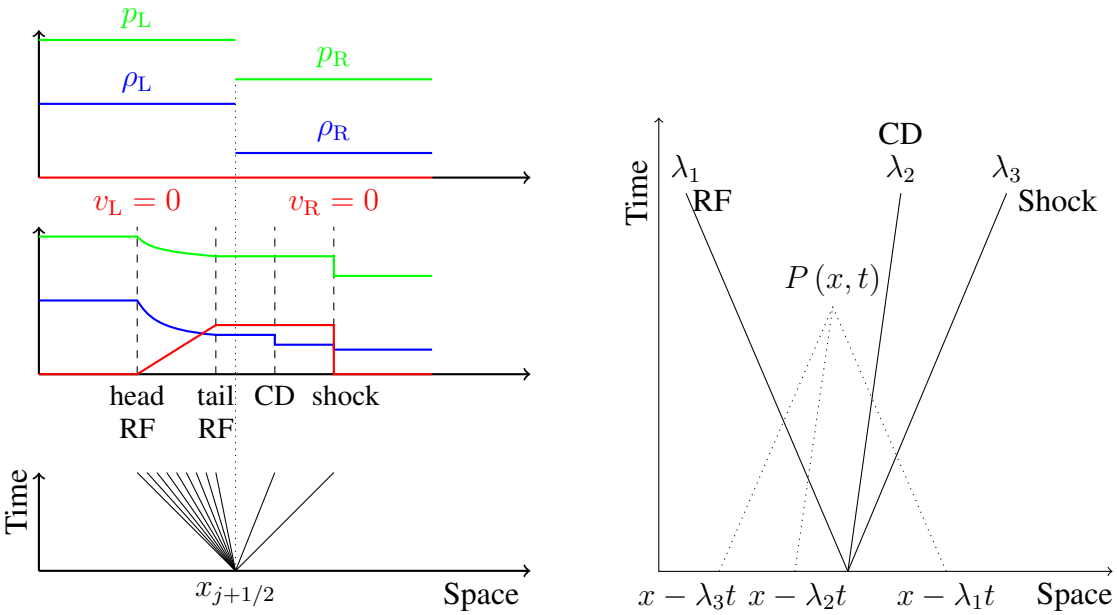


Figure 3.7: The Riemann problem. The top left panel shows the initial conditions of the Sod shock tube (see Sect. 4.2), which is a Riemann problem: a discontinuity separates a left and a right state. Pressures are shown in green, densities in blue and velocities in red. Details on the initial conditions can be found in Fig. 4.3. In the middle left panel the time evolved solution of the Riemann problem is shown: we see the propagation of a shock and a contact discontinuity to the right and a rarefaction wave propagating to the left. The dashed lines indicate the location of the head and the tail of the rarefaction fan (RF), the contact discontinuity (CD) and the shock. The lowest left panel shows the characteristics for the different waves. The right panel shows the domain of influence (solid lines) for the point P(x,0) and the domain of dependence (dashed lines) for the point P(x,t), which is similar to Fig. 3.5.

3.5 Riemann problem

Technically speaking, the Riemann problem (shown in Fig. 3.7) is a Cauchy initial value problem with piecewise constant initial conditions. This is the typical problem arising at each cell interface at each time step in a hydrodynamic simulation carried out with a grid code: Basically, a grid code discretizes the density-, pressure- and velocity distribution in the ISM and stores cell averaged quantities. As a consequence, all cell interfaces separate a constant "left" state from a constant "right" state. At each time step the gas in the cells has the chance to flow into adjacent cells. Therefore, the Riemann problem – waves created by the interaction of zones with piecewise constant values of the before mentioned quantities – has to be solved at every interface between cells at every time step. Hence, the solution of the Riemann problem is of fundamental importance for understanding hydrodynamical simulations with grid codes.

3.5.1 Solution of the Riemann problem

Since there is no closed analytic form of the general Riemann problem (not even in 1D), typically a Newton Raphson method and a specified accuracy are used if a hydrodynamical code claims to use an "exact Riemann solver". In this work, however, we will not use exact Riemann solvers,

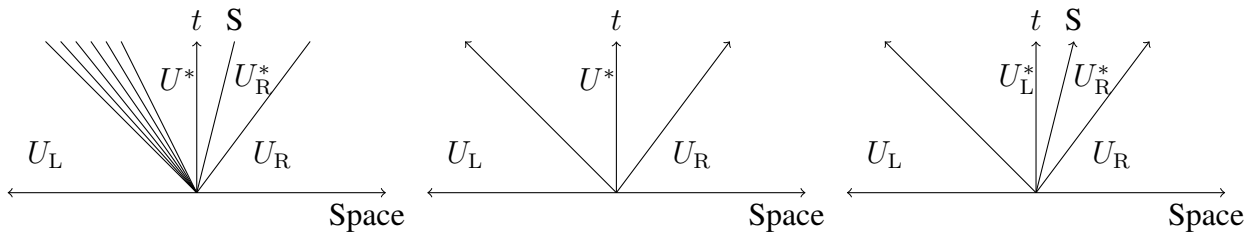


Figure 3.8: Characteristics in (approximate) Riemann solvers. It can be seen that the linearization of the Jacobian in the Roe solver replaces the rarefaction fan by a simple wave: The characteristics of an exact Riemann solver (left) show a rarefaction wave, which is missing in the characteristics of the HLL solver (center) and the HLLC solver (right). The HLL solver replaces all waves by simple waves and just keeps the fastest right moving and the fastest left moving wave of the Roe solver. The HLLC solver re-inserts the contact discontinuity.

but apply approximate Riemann solvers to reduce the computational cost of the simulations. Approximately solving the Riemann problem is a valid approach, since the Godunov scheme (see Sect. 3.6) uses averaged values for the initial conditions. Moreover, in the Godunov algorithm a full solution of the Riemann problem is not necessary: Since the Godunov algorithm aims to find fluxes, approximating the Riemann fluxes directly already solves the problem and calculating the states in the Riemann problem is not necessary. The approximate Riemann solver used for most of our work is the HLLC solver (Toro et al., 1994). To motivate this choice, we will have a brief look at the most common Riemann solvers. We already mentioned that Eq. 3.11 is non-linear and that it tells us that the discontinuity in the Riemann problem will create waves, which will travel at constant speed.

A well-known approximation, the Roe approximate Riemann solver Roe (1981, 1986), is a linearization of the Jacobian in Eq. 3.11:

$$\partial_t \vec{U} + \hat{J}(U_L, U_R)\partial_{x_i}\vec{U} = 0 \quad . \tag{3.12}$$

This constant coefficient linear system is then solved exactly instead of the original nonlinear system. Technically, the Roe algorithm would start by constructing this constant coefficient matrix. This includes finding Roe's average states, sound speeds and enthalpies at cell interfaces. Then it would proceed to create Eigenvectors and Eigenvalues. Next, it would compute wave strengths and fluxes for all Eigencomponents and apply the flux limiter for all Eigencomponents. Finally, the interface flux is found by using symmetric fluxes and adding a diffusive flux term again using the flux limiter. The interface flux is then used to update the state vector.

The difference between the Roe solution and an exact solution is that the Roe solver assumes simple waves. Thus, the solution will lead to constant states instead of the rarefaction fan shown e.g. in Fig. 3.8. The HLL (Harten et al., 1983) and the HLLE (Einfeldt, 1988) solver further simplify this solution by only following the two fastest waves. The HLLC (Harten-Lax-van Leer-Contact) solver (Toro et al., 1994) restores the missing rarefaction wave. Tests (see Sect. 4.2) showed that in the RAMSES code (Teyssier, 2002) this solver achieved the best results in the Sod shock tube test (see Sect. 4.2).

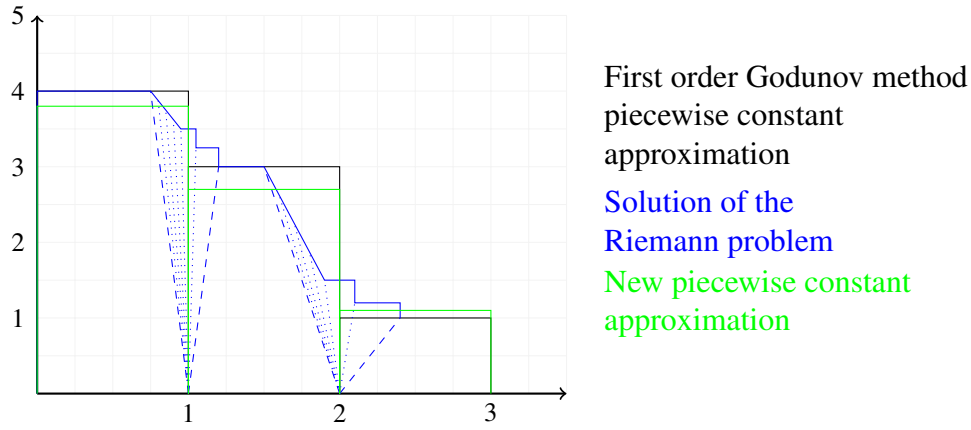


Figure 3.9: Sketch of the first order Godunov method. The averaged quantities in the cells are piecewise constant functions (black). At each time step the Riemann problem is solved at every cell interface (blue). The propagation of the different waves at this interface is evaluated and used to change the averaged quantities. At the end of the time step the averaged quantities are again represented by (possibly different) piecewise constant functions (green).

3.6 Godunov's method

Having settled that we are equipped with methods to (approximately) solve the Riemann problem, we can now discuss the algorithm initially proposed by Godunov[6].

Godunov's algorithm assumes that each computational cell represents a fluid volume with cell averages for density, velocity and energy. These cell averages are used to reconstruct a piecewise polynomial function. In the simplest case, the first order Godunov method sketched in Figure 3.9, the reconstructed function is piecewise constant. In the next step, the Riemann problem is solved at every cell interface. The solution leads to wave families traveling at constant speed. The time step in this algorithm is limited by the CFL for the wave family with the highest velocity, since the similarity solution of the Riemann problem for one interface gets messed up by waves from neighboring interfaces when the fastest traveling wave has time to cross a grid cell from one face to another. The propagation of the different wave families (e.g. entropy wave, sound waves and – in MHD – Alfvén waves) leads to Riemann fluxes, which are used to calculate new cell averages.

Unlike finite differencing algorithms, the first order Godunov algorithm is directly based on the conservation laws and leads to an exact solution by combining the solutions of the Riemann problems. Due to the averaging at the end of the step, this method is of first order accuracy. In other words, the diffusivity of the method depends on the method for calculating the cell averages, which is defined by the finite volume scheme. High order methods try to overcome this problem by using higher order reconstruction methods, for example a piecewise linear function that may have slopes. The first order Godunov method is very stable, but at the price of being very diffusive.

[6]This is usually cited as "Godunov, S. K. (1959), "A Difference Scheme for Numerical Solution of Discontinuous Solution of Hydrodynamic Equations", Math. Sbornik, 47, 271-306, translated US Joint Publ. Res. Service, JPRS 7226, 1969" but this reference is not easily accessible via ADS.

3.7 2nd order Godunov schemes: Total Variation Diminishing (TVD) Advection

Second order Godunov schemes allow for piecewise linear approximations of density, pressure and velocity instead of piecewise constant functions (see Fig. 3.10). This way, it is possible to better resolve discontinuities e.g. in the Sod shock tube test. The Godunov theorem states that only first order linear schemes are monotonicity preserving. Since the reconstruction of the averages should not create new local extrema, total variation diminishing (TVD) schemes are used. There are several different choices of flux limiting functions (sometimes also called "slope limiters", e.g. in the RAMSES documentation). All these methods are based on monotonicity criteria and will degenerate to first order at extreme points. Examples of such methods are shown in Fig. 3.10: The minimum modulus (MinMod) flux limiter (Roe, 1986, Eq. 56) produces a monotonous reconstructed function by setting the slopes to $\frac{\Delta q_\mathrm{i}}{\Delta x} = \min\left(\frac{q_{\mathrm{i}+1}-q_\mathrm{i}}{\Delta x}, \frac{q_\mathrm{i}-q_{\mathrm{i}-1}}{\Delta x}\right)$ if the signs of both slopes are the same. Otherwise the MinMod limiter degenerates to first order. The Sweby diagram (Sweby, 1984) in Fig. 3.10 shows that the MinMod limiter is the most diffusive limiting function, since it applies the maximal possible limiting in 2nd order TVD region. The counterpart is the superbee limiter (Roe, 1986, Eq. 58), which applies the minimal possible limiting and has the drawback to get unstable for most astrophysical problems. The monotonized central-difference limiter (MC limiter or MonCen limiter) limiter (van Leer, 1977, Eq. 66) lies between these two extrema. Here the right and left values are bounded by the initial average values: $\frac{\Delta q_\mathrm{i}}{\Delta x} = \min\left(\frac{q_{\mathrm{i}+1}-q_{\mathrm{i}-1}}{2\Delta x}, 2\frac{q_{\mathrm{i}+1}-q_\mathrm{i}}{\Delta x}, 2\frac{q_\mathrm{i}-q_{\mathrm{i}-1}}{\Delta x}\right)$, if the signs of all three slopes are the same. Otherwise it also falls back to first order.
The drawback of the second order methods is that they are less stable than the first order method and can lead to negative densities and negative pressures. Many codes start with the sharpest limiter (from the three discussed limiters superbee would be the "sharpest" but it is known to fail in most astrophysical applications) and go to more diffusive limiters if negative densities or negative pressures occur. The RAMSES code (Teyssier, 2002), however, cannot switch between limiters on the fly – here the limiting function chosen at the start of the simulation is used for all solutions of the Riemann problem. For our setup (stellar winds and supernova explosions in molecular clouds) the MonCen limiter or – if the simulation crashed – the MinMod limiter were used. As already mentioned, sharp limiters like superbee lead to code crashes in most astrophysical applications. Restarting any of our simulation with this limiter shows that our problem of stellar winds and supernovae is no exception.

3.8 Side note: alternatives to Godunov's method

In this work we will use Godunov's method for our simulations, but basically a grid code could also use an artificial viscosity approach. The advantage of this approach is a more accurate internal energy evolution in regions where $\frac{1}{2}\rho v^2 \gg U$. But this comes at the price of smearing out shocks and applying incorrect Rankine Hugoniot jump conditions. As already mentioned, another drawback of the artificial viscosity approach is that finite difference schemes do not directly use the conservation laws. However, they are easier to code than conservative shock-capturing scheme, which can be an advantage if one wants to add additional physical processes.
For our project we decided to use a Riemann solver, since it is less diffusive and resolving shocks and contact discontinuities well is important for the questions we are trying to tackle with our work (e.g. feedback energy efficiency of massive stars in the ISM). Moreover, with a Godunov

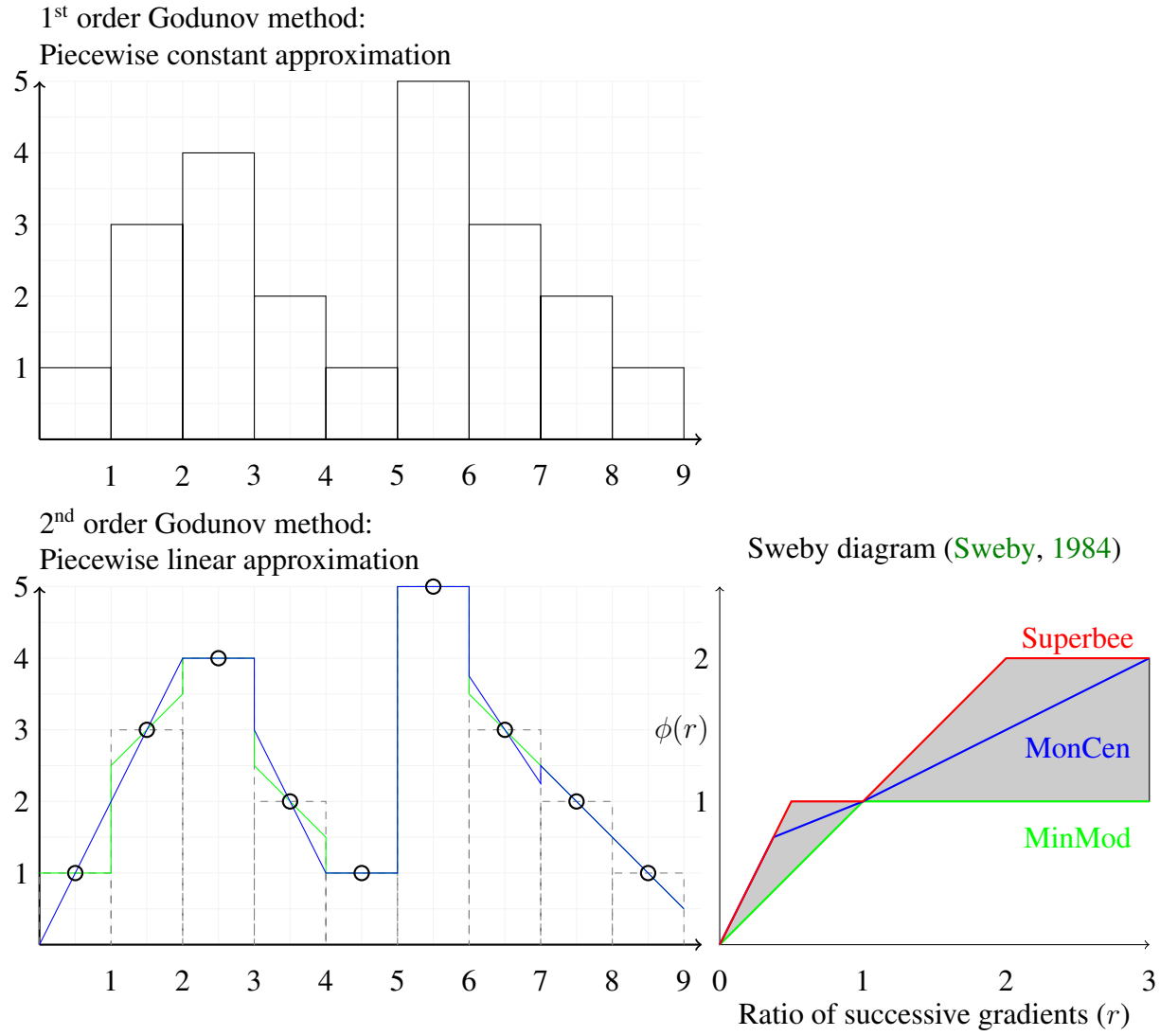


Figure 3.10: Second order Godunov methods. In contrast to the first order Godunov method, second order methods allow for non-zero slopes of the piecewise linear reconstruction. The left lower panel shows two different slope limiting functions: monotonized central differences (blue) and minimum modulus (green). The lower diffusivity of the 2nd order methods comes at the price that the method is no longer monotonicity preserving. The creation of new extrema and the increase of existing extrema can be avoided with Total Variation Diminishing (TVD) schemes. The Sweby diagram in the lower right panel shows the order of different flux limiting functions. The shaded area is the part of the TVD region, where second order accuracy is guaranteed and excessive compression of solutions is avoided. On the horizontal axis, we find the ratio of successive gradients (r). We stop at $r = 3$, since at this point the corresponding value of the flux limiter function ($\phi(r)$), shown on the vertical axis, reaches its maximum for all depicted limiters and stays at this values for higher r (i.e. for MonCen and Superbee we find $\phi(r > 3) = 2$ and for MinMod $\phi(r > 3) = 1$).

method shocks can be treated directly and sound waves and moving matter can be treated with the same precision. As it is a finite volume scheme, it strictly conserves mass, momentum and energy. However, it might run into problems with the internal energy evolution.
In conclusion we decided that a second order Godunov method is the best tool for our study.

3.9 Adaptive mesh refinement (AMR) and parallelization

In this section we will discuss methods to speed up the simulations. We already mentioned that the number of dimensions has a large effect on the computational cost of a simulation. In addition to exploiting the symmetries of the problem (i.e. lowering the number of dimensions) and limiting the included physical processes (e.g. gravity or radiative cooling are computationally intensive) run times can be shortened by using several CPUs in parallel. Technically, this is implemented via MPI in the RAMSES and PLUTO code, which are used for this work. Basically the simplest way of domain decomposition is sub-dividing the computational volume into parallel slabs. Cells on the boundary of these slabs are passed to all adjacent CPUs, whereas cells inside the slabs are passed to a single CPU. In practice in simulations with nonuniform resolution, cells are most commonly distributed between CPUs using a Peano-Hilbert curve which minimizes the number of cells which have to be passed to more than a single CPU and distributes the cells evenly among the CPUs. If some regions of space contain smaller cells, passing parallel slabs of the same volume to all CPUs could lead to sub-optimal load leveling.
Another way of speeding up mesh simulations is to use lower resolution in areas which are less interesting. Whereas resolution naturally follows the distribution of mass in an SPH simulation, this functionality has to be added to grid codes. If it is known beforehand, in which part of the computational volume high resolution will be desired, it is advantageous to define a fixed non-uniform grid. In most of our simulations, however, the areas with steep gradients move inside the box. In such situations adaptive mesh refinement (AMR) is the method of choice. AMR optimizes the computational cost of a simulation by increasing the resolution in crucial areas and lowering it in smooth regions. Technically, the RAMSES code's mesh has several levels: think of the cube containing the computational domain as level zero: In level one this cube is divided into 2^ν cells, where ν is the number of dimensions. For level two a cell of level one is again subdivided into 2^n cells. The advantage of AMR is that not all cells of a level have to be subdivided but that the AMR region can follow the interesting regions of the simulation. For example the AMR region in RAMSES can be computed by checking gradients of the primitive variables or via a geometrical criterion. E.g. this can be helpful, if one checks for density gradients and the initial conditions in a part of the volume contain a density jump in pressure equilibrium. In this situation it can help to exclude a part of the volume from refinement.
In the RAMSES code the interpolation at the borders of regions with different grid levels can be controlled with the `interpol_type` variable in the namelist file. Repeating the same simulation ($60\ M_\odot$, infinite cloud, spherical cavity) with `interpol_type=0` (no interpolation), 1 (MinMod[7]) and 2 (MonCen) showed that the feedback energy efficiency is similar in all three simulations. This is expected, since the algorithm optimizing the extent and location of highly resolved areas tries to put the grid borders in areas with small gradients.
In our production runs we used MonCen slope limiting for the Riemann solver (unless it became unstable and crashed) and MinMod interpolation between grid levels. Simulations with this mix-

[7] See Sect. 3.7 for a definition of MinMod and MonCen.

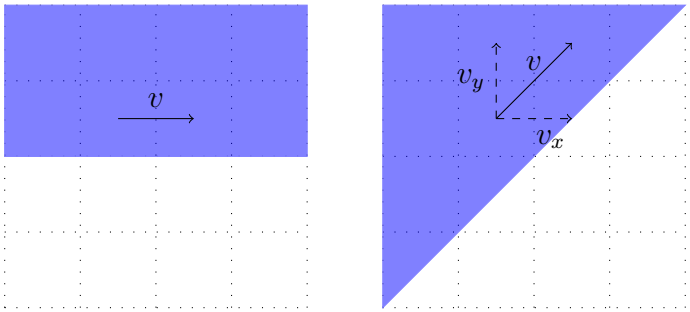


Figure 3.11: Sketch of numerical diffusion. The amount of numerical diffusion in a simulation is not only constrained by the choice of Riemann solver but also by the angle between the grid axes and the direction of the flow. Transversal flows, as shown in the right sketch are prone to higher numerical diffusivity.

ture of slope calculations did not differ significantly form a simulation with the same interpolation scheme for the Riemann solver and the grid levels, since the grid boundary should be in a smooth region of the flow anyway.

3.9.1 Pitfalls of AMR

If AMR is uses in a simulation, one has to be aware of its side effects. For example in simulations with moving AMR regions one can observe shocks seeded by the grid interfaces.
In our simulations we also observed cooling introduced by grid interfaces. The simplest test to reproduce this behavior is a stellar wind in a homogeneous ambient medium with a geometrical refinement criterion which limits the highest resolution grid to a cube centered around the feedback region's center. In this test also a gradient based refinement criterion is used. If the geometrical criterion starts to limit the refinement in areas which the gradient based refinement criterion would refine, the finest grid starts loosing track of the dense shell. Ultimately this results in enhanced cooling near boundaries of the finest grid. Since the region's boundaries now can happen be placed near strong gradients `interpol_type=0` makes the simulation stable enough to survive this phase. This is related to the code crashes with negative temperatures behind the shock! Obviously this problem is an over oscillation of the interpolation scheme near the edge of the grid levels. Interpolating with scheme 0 made the simulations very stable but diffusive.[8]

3.9.2 Numerical diffusion

Numerical diffusion[9] is a problem arising in grid codes, because the discretized hydrodynamical equations have truncation errors which tend to make them more diffusive than the differential equations. This error is smaller for higher order schemes. Also incorrect evaluation of field gradients (which can be caused by too coarse meshes or bad flux limiting functions) and an angle between the flow direction and the grid axes increases this problem.

[8] A re-run on the server OPTIMAL actually showed that in a 2D cut along $z = 0$ cooling in cells inside the bubble occurs. I.e. radiative losses in cells with high temperatures and low densities were found. However, such cells should not cool. This is e.g. seen at the positive x axis. Affected cells are close to the grid boundary and the problem is caused by the geometric refinement criterion. The grids loose track of the dense shell. Another pitfall of the geometric refinement criterion is seen in the velocity. Here, the geometric refinement criterion of the finest grid leads to a cross like feature in velocity, since the grid boundary is reached first along the grid axes. Type 1 and 2 crash after 177 code time units (output 12). This is the time when the geometrical refinement strategy already took over. In this phase the finest grid cannot follow the dense shock any more.

[9] Numerical diffusion is also known as "artificial" or "false" viscosity, damping or dissipation.

In the two computational domains shown in Fig. 3.11 density, pressure and velocity are constant. The conservative passive scalar shown in blue has a step which is parallel to the flow direction (small black arrow). If the flow is not parallel to the grid, numerical diffusion will smear out the step in the conservative passive scalar. Numerical diffusion is zero if there is either no gradient of the passive scalar (x, left sketch) or no velocity component (y, left sketch). Transversal flows (right sketch) show gradients of the passive scalar and have velocity components in all directions. Hence the discretization errors will lead to numerical diffusion.

Chapter 4

Basic building blocks of simulations

The aim of this work is to combine stellar feedback with the physics of the ISM. If such simulations are carried out with a grid code, this problem can be subdivided into a number of sub-problems which have to be solved accurately (enough). Many of these simplified problems have an analytic solution. Thus before putting it all together and looking at the simulations as a whole, we check how well the code recovers the analytic solutions of these sub-problems and we will discuss how we technically implement the stellar feedback. Readers not interested in these technical details can skip this section.

As mentioned in Sect. 3.5, the typical sub-problem arising at every cell interface at every time step is the Riemann problem. Basically the code subdivides the volume into cells. Each cell has an average density, an average pressure and average velocities. At each cell face that lies inside the simulated volume[1] the cell touches another cell. In 1D one can sketch this problem with a step function e.g. in a density vs. spatial coordinate diagram. If the averaged velocities in both cells are zero, this is the initial state of the Sod shock tube test (Sect. 4.2): Two media with possibly differing gas properties are separated by a diaphragm. As soon as this barrier is removed, gradients in the gas will lead to waves. In the hydrodynamic case, we will find an entropy wave, a rarefaction wave and a density wave. The Sod shock tube test is sometimes also called "dam-break-problem".

As soon as we are convinced that our numerical method can treat individual cell interfaces with sufficient accuracy, we can proceed to blast waves. The Sedov-Taylor problem, which is a blast wave without radiative cooling losses, should be recovered (Sect. 4.3).

Another agent in our models are stellar winds. Sect. 5.2 checks the conservation of mass and energy in our feedback prescription. It also compares the solution to analytic solutions of a constant wind without radiative cooling.

4.1 Waves, discontinuities and shocks

In the tests described in this chapter, we will come across different kinds of propagating waves. Thus, we will first introduce the terminology and mention the most important relations (e.g. shock jump conditions).

[1] Boundary cells are special cases, discussed in Sect. 3.2.1

4.1.1 Contact discontinuity (CD)

If two gasses with different temperatures are in pressure equilibrium, one can observe a contact discontinuity (CD) between them. Contact discontinuities propagate with the characteristic speed of the media on both sides. Hence, there is no pressure- or velocity gradient across a contact discontinuity. However, a sudden change in density is observed.
Thus, the EOS (Eq. 3.2), which connects energy and pressure, has to allow for the same pressure at two different densities. Consequently the aforementioned situation cannot be found in isothermal simulations, where the pressure is a function of density only. From the adiabatic EOS (Eq. 3.2) it is obvious that this contact discontinuity is also an entropy wave.
Contact discontinuities can also arise for passive tracers. This kind of contact discontinuities can also be found in isothermal simulations. For example, think of a jump in metallicity, which might be caused by stellar yields.
The antagonists of contact discontinuities are diffusive processes, which lead to mixing of the two media. Such mixing processes are described in Sect. 6. Whether the smearing of the discontinuity is relevant for the time scales of interest, can be derived from the diffusion coefficients.

4.1.2 Rarefaction wave

Fig. 3.6 shows possible slopes of characteristic curves. Converging characteristics lead to a compression of the wave whereas diverging characteristics form a rarefaction wave[2]. The wedge formed by diverging characteristics of the Euler equations is filled with a fan of characteristics.[3]
In the following, we will briefly derive the shape of the rarefaction wave's profiles in the Sod problem (details on this problem can be found in Sect. 4.2) to illustrate, how solutions can be found using Riemann invariants. In the Sod shock tube the rarefaction wave is located between a static medium with velocity $u_\mathrm{L} = 0$, pressure p_L, density ρ_L and sound speed $c_\mathrm{s,L}$ and a moving medium with $u_\mathrm{R}, p_\mathrm{R}, \rho_\mathrm{R}$ and $c_\mathrm{s,R}$. From the location x_o of the origin of the rarefaction fan, the location of the head of the expansion fan can be found from the leftmost characteristic as $x_\mathrm{hRF}(t) = x_0 - c_\mathrm{s,L}t$. The tail of the rarefaction wave is found at $x_\mathrm{tRF}(t) = x_0 + (u_\mathrm{R} - c_\mathrm{s,R})t$. Such rarefaction waves accelerate the fluid smoothly: The continuous function of the velocity of a rarefaction wave rises linearly between the left and right value:

$$u_\mathrm{RF}(x,t) = \begin{cases} 0 & \text{if } x \leq x_\mathrm{hRF} \\ \frac{(x-x_0)/t + c_\mathrm{s,L}}{u_\mathrm{R} + c_\mathrm{s,L} - c_\mathrm{s,R}} u_\mathrm{R} = \frac{2}{\gamma+1}\left(\frac{x-x_0}{t} + c_\mathrm{s,L}\right) & \text{if } x_\mathrm{hRF} \leq x \leq x_\mathrm{tRF} \\ u_\mathrm{R} & \text{if } x \geq x_\mathrm{tRF} \quad . \end{cases} \tag{4.1}$$

Where we used the Riemann invariant $u + \frac{2c_s}{\gamma-1}$ to replace the left sound speed in the denominator with $c_\mathrm{s,L} = \frac{\gamma-1}{2}u_\mathrm{R} + c_\mathrm{s,R}$. In addition to this linear velocity profile the rarefaction wave has an exponential density profile. To express the density we use that entropy is a Riemann invariant of the acoustic wave, which means that $p\rho^{-\gamma}$ is constant. This is plugged into the ratio of the sound speed to the left sound speed (this will be shown in Eq. 4.55) and leads to

$$\rho(x,t) = \rho_\mathrm{L}\left(\frac{c_s(x,t)}{c_\mathrm{s,L}}\right)^{2/(\gamma-1)} \tag{4.2}$$

[2] rarefaction waves are also called expansion waves
[3] This solution follows from entropy conservation.

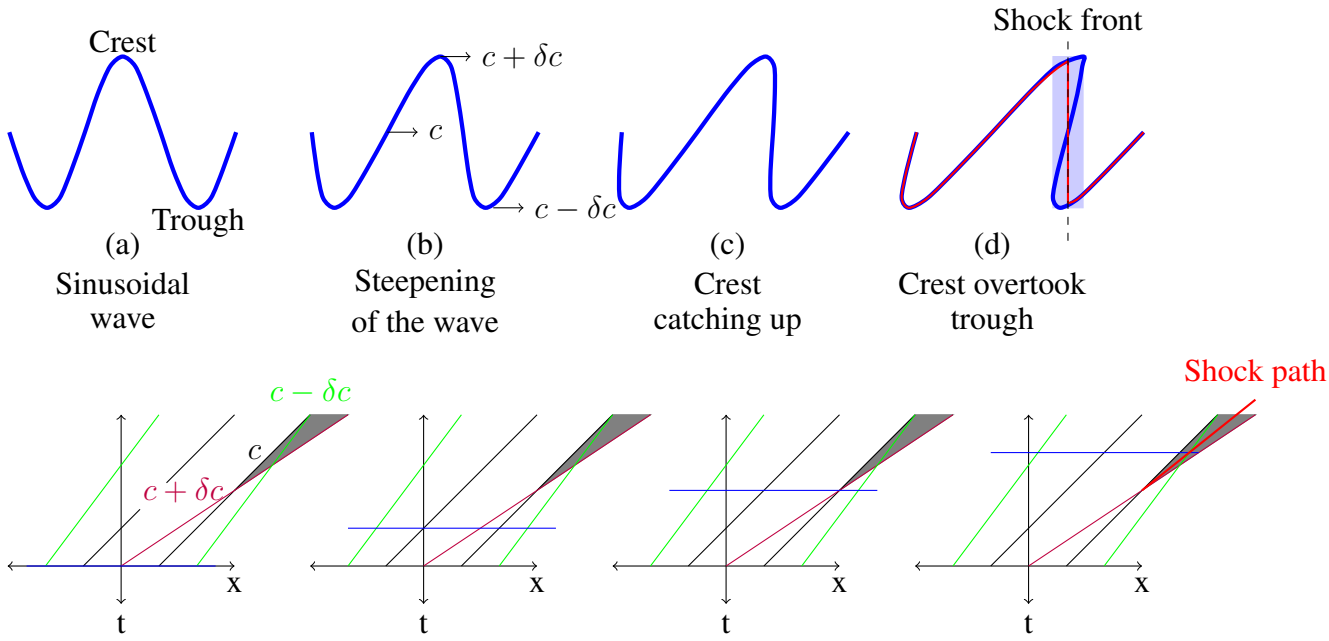


Figure 4.1: In the upper panel of this sketch of shock formation we see an initially sinusoidal pressure wave (a) which is distorted by pressure dependent propagation speeds (b). It steepens (c) and ultimately the crest would overtake the trough (d). But when the state variable would become multivalued (blue shaded area), a shock (red) forms. The location of shock fronts can be found via shock fitting. This problem is also sometimes called "slope catastrophe" and can also be visualized with the crossing of characteristics. The lower panel sketches such characteristics in the spacetime plot. Higher velocities (e.g. purple line) are depicted with characteristics with lower slopes (a). The gray areas in the lower panel indicate regions with crossing characteristics. The blue line in the lower plots indicates the position of the corresponding upper plots in spacetime. In plot (d) a shock path (red) is sketched in the gray area, where the characteristics cross. This path is also found via shock fitting.

The Riemann invariant $u + \frac{2c_s}{\gamma-1}$ allows us to use $c_\mathrm{s}(x,t) = c_{\mathrm{s,L}} - \frac{\gamma-1}{2}u(x,t)$ which we can combine with Eq. 4.1 and Eq. 4.2 to find

$$\rho(x,t) = \rho_\mathrm{L}\left[\frac{2}{\gamma+1} - \frac{\gamma-1}{\gamma+1}\frac{x-x_0}{c_{\mathrm{s,L}}t}\right]^{2/(\gamma-1)} \quad . \tag{4.3}$$

To find the pressure for a given density, we can again use that entropy is a Riemann invariant of the acoustic wave (i.a. $p\rho^{-\gamma}$ is constant) and find

$$p(x,t) = p_\mathrm{L}\left(\frac{\rho(x,t)}{\rho_\mathrm{L}}\right)^{\gamma} \quad . \tag{4.4}$$

If the velocity of the left state is not zero, a derivation of the rarefaction wave's shape from the Riemann invariants, can for example be found in LeVeque (2002, Sect. 14.12).

4.1.3 Shock wave and shock jump conditions

If characteristics of the same family cross, shocks form. An example is shown in Fig. 4.1. We start with a sinusoidal wave and check the effect of pressure dependent propagation speeds. As

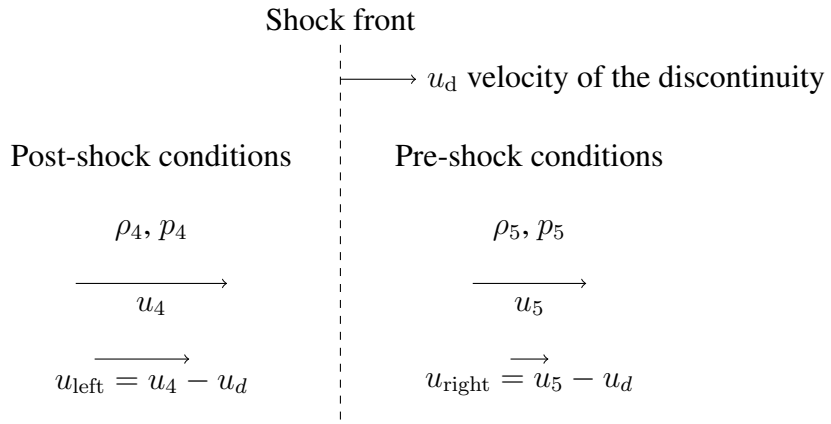


Figure 4.2: This sketch shows a discontinuity (dashed line) moving with a velocity u_d into unperturbed gas. The notation uses the same indices we will also use for the shock in the Sod shock tube test: the right unperturbed state (the pre-shock conditions) get the subscript "5". Since it is advantageous to use the rest-frame of the shock to find the Rankine Hugoniot jump conditions, the velocities in this coordinate system u_left and u_right are also given.

characteristics get closer, the wave profile gets distorted. Ultimately the characteristics cross, the crest tries to overtake the trough and a shock forms. The curve, which connects the intersection points of the characteristics in spacetime is called "shock path".
In contrast to the smooth acceleration we observed in rarefaction waves, the fluid is accelerated abruptly if it is hit by a shock.
In the section on rarefaction waves we derived the solution using conserved quantities. We will use a similar procedure to derive the **Rankine Hugoniot relations** at the discontinuity from the conservation laws $u_d[U] + [F(U)] = 0$ for the vector of the conserved quantities U (mass, momentum, energy).
For this purpose, it is advantageous to express the velocities in terms of velocities in the coordinate system comoving with the discontinuity: $u_{right} = u_5 - u_d = -u_d$ and $u_{left} = u_4 - u_d$, where u_d is the speed of the discontinuity (see also Fig. 4.2). This shock rest frame allows us to get rid of the time derivatives. We will show it here with the conservation of mass: The first term of the volume integrated equation of continuity $\frac{\mathrm{d}}{\mathrm{d}t}\int_V \rho \mathrm{d}V + \int_V \rho \nabla \vec{u} \mathrm{d}V = 0$ is zero in the rest frame of the shock[4]. Using the Gauß theorem (also known as Green's formula) the second term can be transformed into a surface integral: $\int_V \rho \nabla \vec{u} \mathrm{d}V = \int_O \rho \vec{u} \mathrm{d}A$. For shocks with $\vec{u} = (u, 0, 0)$ only the surface integrals at the surfaces parallel to the density step are nonzero: $\int_{A_4} \rho_4 u_\mathrm{left} \mathrm{d}A_4 + \int_{A_5} \rho_5 u_\mathrm{right} \mathrm{d}A_5$. The orientation of the surface with respect to the speed gives the sign for the integrals and leads to $\rho_4 u_\mathrm{left} = \rho_5 u_\mathrm{right}$ where u_left and u_right are one-dimensional rest frame speeds.
In the next step we convert the velocities back to the system with a moving discontinuity. We start from a coordinate system comoving with the discontinuity mass conservation (Eq. 3.7) and find:

$$\rho_4 u_\mathrm{left} = \rho_5 u_\mathrm{right} \tag{4.5}$$

$$\rho_4 u_4 - \rho_4 u_d = \rho_5 u_5 - \rho_5 u_d$$

$$\frac{\rho_4 u_4 - \rho_5 u_5}{\rho_4 - \rho_5} = u_d \quad . \tag{4.6}$$

[4]The flow of plasma can be treated like it was constant in time, because the time the shock needs to cross the step is too small for significant energy loss through processes like radiation

The relation from momentum conservation (Eq. 3.8) in a system comoving with the discontinuity is:

$$\begin{aligned} p_4 + \rho_4 u_{\text{left}}^2 &= p_5 + \rho_5 u_{\text{right}}^2 \qquad (4.7) \\ p_4 + \rho_4 u_4^2 - 2\rho_4 u_4 u_d + \rho_4 u_d^2 &= p_5 + \rho_5 u_5^2 - 2\rho_5 u_5 u_d + \rho_5 u_d^2 \\ \rho_4 u_4^2 - \rho_5 u_5^2 - 2(\rho_4 u_4 - \rho_5 u_5) u_d + (\rho_4 - \rho_5) u_d^2 &= p_5 - p_4 \quad . \qquad (4.8) \end{aligned}$$

Here p is the gas pressure and ρu^2 is the ram pressure. We can eliminate the speed of the discontinuity u_d from the Rankine Hugoniot relations from mass (Eq. 4.6) and momentum (Eq. 4.8) conservation and get:

$$\rho_4 u_4^2 - \rho_5 u_5^2 - \frac{(\rho_4 u_4 - \rho_5 u_5)^2}{\rho_4 - \rho_5} = p_5 - p_4 \quad . \qquad (4.9)$$

Combining mass conservation (Eq. 4.5) and momentum conservation (Eq. 4.7) in the rest frame of the shock leads to

$$\rho_4^2 u_{\text{left}}^2 (\frac{1}{\rho_4} - \frac{1}{\rho_5}) = p_5 - p_4 \quad . \qquad (4.10)$$

For energy conservation (Eq. 3.9) we can immediately divide the equation by $\rho_4 u_{\text{left}} = \rho_5 u_{\text{right}}$ (Eq. 4.5) from mass conservation and get the specific[5] internal energy e and the specific kinetic energy $0.5u^2$ of the fluid. We find:

$$\begin{aligned} e_4 + \frac{p_4}{\rho_4} + \frac{u_{\text{left}}^2}{2} &= e_5 + \frac{p_5}{\rho_5} + \frac{u_{\text{right}}^2}{2} \\ e_4 - e_5 &= \frac{2p_5 + \rho_5 u_{\text{right}}^2}{2\rho_5} - \frac{2p_4 + \rho_4 u_{\text{left}}^2}{2\rho_4} \qquad \text{with Eq. 4.7} \\ e_4 - e_5 &= \frac{p_5}{2\rho_5} - \frac{p_4}{2\rho_4} + \frac{p_4 + \rho_4 u_{\text{left}}^2}{2} (\frac{1}{\rho_5} - \frac{1}{\rho_4}) \qquad \text{with Eq. 4.10} \\ e_4 - e_5 &= \frac{p_4 + p_5}{2} (\frac{1}{\rho_5} - \frac{1}{\rho_4}) \qquad \text{(Rankine-Hugoniot equation)} \qquad (4.11) \end{aligned}$$

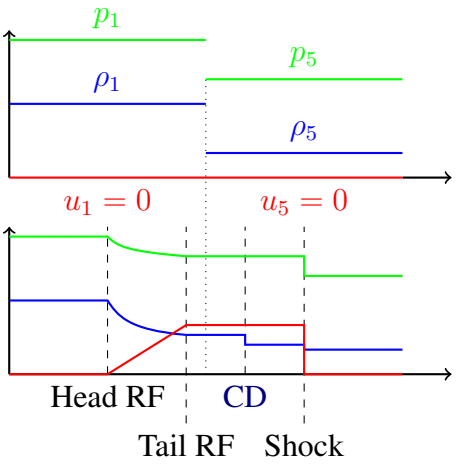
Combining the Rankine Hugoniot equation (Eq. 4.11) with the EOS $p = (\gamma - 1)\rho e$ (Eq. 3.2) leads to:

$$\frac{p_4}{\rho_4} - \frac{p_5}{\rho_5} = (\gamma - 1)\frac{p_4 + p_5}{2} (\frac{1}{\rho_5} - \frac{1}{\rho_4}) \quad . \qquad (4.12)$$

A very common way of expressing the Rankine-Hugoniot jump conditions is as pressure-, density- or temperature ratios for both sides of the discontinuity. For this purpose it can be combined with the EOS and expressed with Mach numbers. Since we will not directly use these formulations, we refer the reader to the text book of Shu (1992, Eq. 15.35 to Eq. 15.37) for ratios with Mach numbers and just show the density ratio, which we will need for the shock tube test. For this purpose the Rankine Hugoniot equation (Eq. 4.12) can be rearranged to:

$$\frac{\rho_4}{\rho_5} = \frac{\gamma \frac{p_4}{p_5} + \gamma + \frac{p_4}{p_5} - 1}{\gamma \frac{p_4}{p_5} + \gamma - \frac{p_4}{p_5} + 1} \quad . \qquad (4.13)$$

[5] Specific means “per mass unit”.



Length:	$x \leq 0.5x_{max}$	$x > 0.5x_{max}$
Velocity:	$u_1 = 0$	$u_5 = 0$
Density:	$\rho_1 = 1$	$\rho_5 = 0.125$
Pressure:	$p_1 = 1$	$p_5 = 0.1$
Thermal energy:	$E_{\mathrm{therm},1} = 1.5$	$E_{\mathrm{therm},5} = 0.15$

Figure 4.3: Sod shock tube test. This sketch is a variant of Fig. 3.7. The top panel and the table show the initial conditions of the Sod shock tube. Pressures are shown in green, densities in blue and velocities in red. The dashed lines in the lower panel separate the five zones of the Sod similarity solution: (1) the unperturbed state of the denser medium, (2) the rarefaction fan, RF (3) the contact discontinuity, CD (4) the fast shock wave and (5) the unperturbed state of the tenuous medium.

4.2 Sod shock tube test

The Sod shock tube is a widely used test for the accuracy of hydrodynamics codes. Sod (1978) proposed this test case to investigate the typical problems of finite difference schemes such as oscillations behind shocks and smearing of contact discontinuities. Sod shock tubes are a class of Riemann problems (see Sect. 3.5) with zero initial velocities. The initial conditions have a discontinuity in pressure and density placed across the grid. On one side is a cold, low density gas and on the other side is a hotter, denser gas. At time $t = 0$ a diaphragm that separates those two media is removed and waves start propagating. This setup is well suited to test numerical schemes, since it produces steep gradients and strong shocks. The importance of this test is obvious, since in a grid code discontinuities can arise at all cell boundaries.
Basically this test problem is one-dimensional, but if the simulation can deal with more dimensions, putting the shock front not perpendicular to the cell faces can help testing how well the code can deal with flows along cell diagonals (which is prone to numerical diffusion, as we saw in Sect. 3.9.2).

4.2.1 Analytic solution of the Sod shock tube problem

The similarity solution of this special Riemann problem consists of five distinct zones (sketched in the lower panel in Fig. 4.3):

1. unperturbed state of denser, high pressure medium ($E_{\mathrm{therm},1}, p_1, \rho_1, u_1$)
2. rarefaction wave propagating into denser medium ($E_{\mathrm{therm},2}, p_2, \rho_2, u_2$)
3. slowly moving contact discontinuity towards the less dense medium ($E_{\mathrm{therm},3}, p_3, \rho_3, u_3$)
4. fast shock wave moving into tenuous medium ($E_{\mathrm{therm},4}, p_4, \rho_4, u_4$)

5. unperturbed state of tenuous, low pressure medium ($E_{\text{therm},5}, p_5, \rho_5, u_5$)

The states of the gas in zone 1 and zone 5 are known from the initial conditions . Also all thermal energies can be found from the EOS (Eq. 3.2) which relates the thermal energies to the adiabatic exponents, densities and pressures. Hence there are nine unknowns: p_2, ρ_2, u_2, p_3, ρ_3, u_3, p_4, ρ_4 and u_4. Sect. 4.1.2 tells us that pressure, velocity and density in the rarefaction wave (p_2, ρ_2 and u_2) are definite if the states 1 and 3 are known. Basically the rarefaction is a reversible, adiabatic process and the Riemann invariants lead to the solution. As we saw in Sect. 4.1.1, another unknown speed and pressure can be removed, since there is no mass flow through the contact discontinuity and the pressure is continuous at the contact discontinuity. Therefore we define new quantities at the contact discontinuity: a velocity $u_3 = u_4 =: u_c$ and a pressure $p_3 = p_4 =: p_c$. Moreover the constant entropy in the rarefaction wave allows us to connect the state 3 to the state 1 via

$$p_c = p_1 \left(\frac{\rho_3}{\rho_1}\right)^{\gamma} \quad . \tag{4.14}$$

From the other Riemann invariant, the sound speed, we find:

$$u_c + \frac{2}{\gamma - 1}\sqrt{\frac{\gamma p_c}{\rho_3}} = \frac{2}{\gamma - 1}\sqrt{\frac{\gamma p_1}{\rho_1}} \quad . \tag{4.15}$$

Thus, we are left with three unknowns: p_c, ρ_4 and u_c. The post-shock medium (state 4) is separated from the pre-shock medium (state 5) by a discontinuity. We can connect these two states with the Rankine Hugoniot shock jump conditions (Sect. 4.1.3). For pressure and density we can use the density ratio from the Rankine Hugoniot equation (Eq. 4.13):

$$\frac{\rho_4}{\rho_5} = \frac{\gamma\frac{p_c}{p_5} + \gamma + \frac{p_c}{p_5} - 1}{\gamma\frac{p_c}{p_5} + \gamma - \frac{p_c}{p_5} + 1} \quad . \tag{4.16}$$

For the post-shock velocity we use Eq. 4.9. Since the pre-shock medium is at rest ($u_5 = 0$), we can drop all terms containing u_5 and find:

$$(p_c - p_5)\left(\frac{1}{\rho_5} - \frac{1}{\rho_4}\right) = u_c^2 \quad . \tag{4.17}$$

combining Eq. 4.14, 4.15, 4.16 and 4.17 leads to

$$\frac{p_5\rho_1}{p_1\rho_5}\frac{\left(1 - \frac{p_c}{p_5}\right)^2}{\gamma\left(1 + \frac{p_c}{p_5}\right) - 1 + \frac{p_c}{p_5}} = \frac{2\gamma}{(\gamma - 1)^2}\left[1 - \left(\frac{p_c}{p_1}\right)^{\frac{\gamma-1}{2\gamma}}\right]^2 \quad . \tag{4.18}$$

The solution of Eq. 4.18 can be computed with an iteration scheme. It provides one with p_c. To get ρ_3, this result for p_c has to be inserted into Eq. 4.14; p_c and Eq. 4.16 lead to ρ_4. Finally the results for p_c and ρ_4 are inserted into Eq. 4.17 to get u_c. For the commonly used parameters in the Sod shock tube test ($\gamma = \frac{5}{3}$, $\frac{\rho_1}{\rho_5} = 8$, $\frac{p_5}{p_1} = 0.1$) the solution is $p_c = 2.93945 p_5$. It can for example be found via www.wolframalpha.com by typing

```
solve R*P*(x-1)^2/(x*(G+1)-1+G)=2*G/(G-1)^2(1-(P*x)^((G-1)/(2G)))^2
 for G=5/3,R=8,P=0.1     .
```

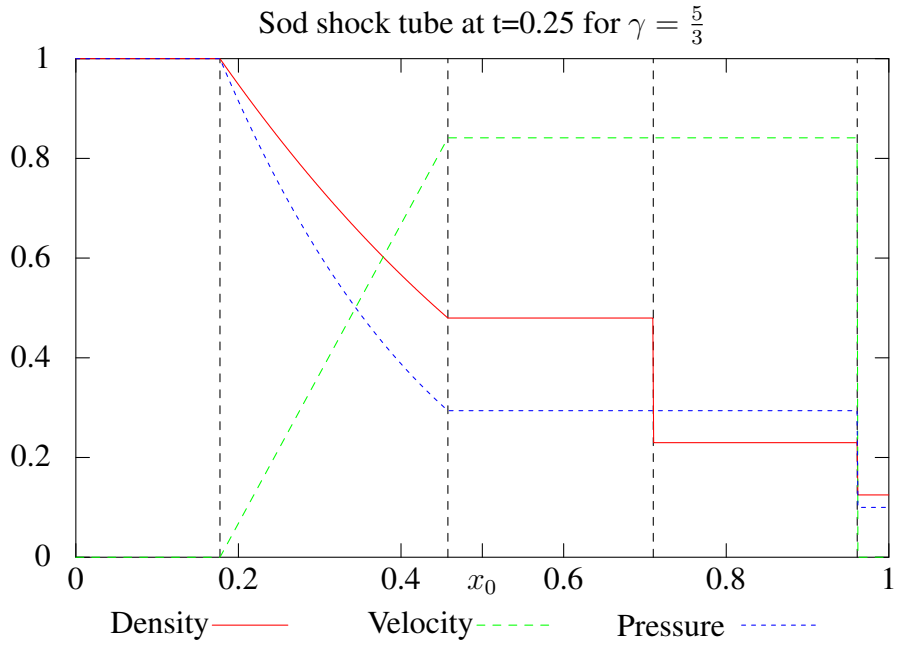


Figure 4.4: This figure shows the analytic solution of the Sod shock tube as discussed in Sect. 4.2.1. The vertical dotted lines show the zone boundaries.

The locations of the zone boundaries are found from the characteristics through the point x_0, which is the location of interface between the media at $t = 0$. The head of the rarefaction wave travels with the sound speed of the unperturbed high pressure medium. Thus, it is found at $x_{h,RF} = x_0 - c_{s,1}t$. The velocity of the tail of this wave is found by subtracting the sound speed from the bulk velocity of the adjacent right region. The tail is thus found at $x_{t,RF} = x_0 + (u_c - c_{s,3})t$. The location of the contact discontinuity is set by the bulk velocity in this adjacent region, which leads to $x_{CD} = x_0 + u_c t$ and for the shock the velocity can be found from Eq. 4.6): $x_s = x_0 + \frac{r_4 u_c}{r4 - r5}t$. The solution for this setup at t=0.25 is shown in Fig. 4.4.

4.2.2 Initial conditions of the Sod shock tube test

The typical setup of the Sod shock tube test is summarized in Fig. 4.3. A density (ρ) and pressure (p) jump in the middle of the computational volume separates two gas phases with $\rho_1 = 1$, $p_1 = 1$, $\rho_5 = 0.125$ and $p_5 = 0.1$. In this notation subscript 1 denotes the initial state of the higher pressure gas and subscript 5 the initial state of the lower pressure gas. Subscripts 2 to 4 are reserved for the intermediate zones which will emerge later on. These zones will be separated by characteristics (characteristics are discussed in Sect. 3.4). Both gasses are initially at rest (i.e. the velocities are $u_1 = u_5 = 0$). With the adiabatic exponent of a monoatomic gas $\gamma = 5/3$ and the adiabatic EOS (Eq. 3.2) this leads to the thermal energies $E_{\mathrm{therm},1} = e_{i,1}\rho_1 = p_1/(\gamma - 1) = 1.5$ and $E_{\mathrm{therm},5} = e_{i,5}\rho_5 = 0.15$.

4.2.3 Results of the RAMSES Sod shock tube test

In our simulations we will focus on the feedback energy efficiency of massive stars in molecular clouds. Cooling losses of the gas near the contact discontinuity (CD) play an important role for

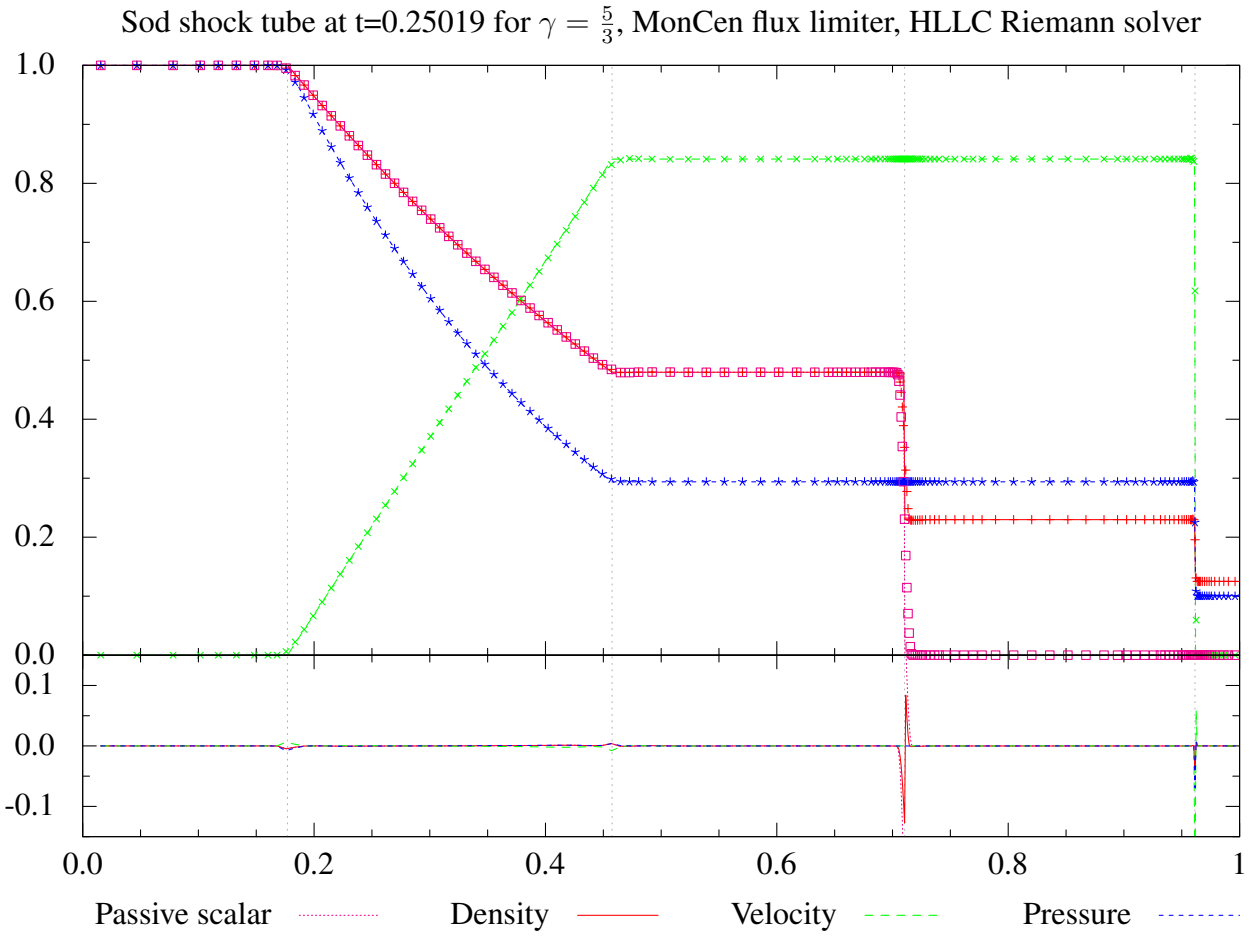


Figure 4.5: This figure shows the results of the HLLC Riemann solver with MonCen flux limiting, which is the preferred choice for our simulations. Lines in the upper panel show the analytic solutions as presented in Sect. 4.2.1 and Fig. 4.4, the superplotted points are results of a simulation carried out with the RAMSES code. The lower panel shows the differences between the result of the simulation and the analytic solution. The residuals for all choices of Riemann solvers and flux limiters in the RAMSES code are compared in Fig. 4.6.

this study. Additionally we want to trace the products of nucleosynthesis (i.e. ^{26}Al and ^{60}Fe). Thus, selecting a Riemann solver and a flux limiter that perform well near contact discontinuities is crucial. Unsurprisingly[6] it turned out that the acoustic and HLLC Riemann solver achieved the best results for the contact discontinuity. The results of the HLLC Riemann solver with MonCen limiting are shown in Fig. 4.5.

This test was carried out with all Riemann solvers implemented in the RAMSES code (for the concepts behind these solvers see Sect. 3.5.1) and the MinMod and MonCen slope limiters (for details on TVD slope limiting see Sect. 3.7). The results are shown in Fig. 4.6. For these simulations the hydrodynamic module of the RAMSES code was used and AMR was switched on. The minimal resolution was set to 2^3 cells in the computational domain. The maximal resolution was 2^{10} cells per unit length. The grid was refined whenever the relative variation of density, velocity or pressure

[6] As discussed in Sect. 3.5.1 the HLLC is a variant of the HLL solver, designed to perform well at contact discontinuities.

across a cell boundary was larger than 5%. In this case the data for newly refined cells was found using a MonCen interpolation scheme for the conservative variables. Moreover reflexive boundary conditions, a CFL of 0.8 (see Sect. 3.3) and the MUSCL scheme were used. The intended time of the data dump was $t = 0.25$. To compare simulations with different AMR grid levels, the parameter `nsubcycle` has to be set accordingly to enforce the time step of the highest resolution grids on coarser grid to get an output at roughly the same times.

The `nsubcycle` parameter controls how many sub cycles will be used for the next finer level. The default value is 2 (in agreement to the dependence of the CFL condition on the grid size: if the wave can travel half the length it may travel on the coarser grid, the time step size has to be halved too). However, it is possible to set this parameter to 1. In this case the time step size from the CFL of the coarsest grid with `nsubcycle=2` will be used. For example for "`nsubcycle=1,1,2,2`" the coarsest level 1, as well as the finer levels 2 and 3 would all use the CFL of level 3, whereas the finest grid at level 4 would use its own CFL. Setting `nsubcycle=1` slows down the code, which is not a problem for small scale tests like the Sod shock tube problem, but permits outputs at desired times.

As a consequence different choices of `nsubcycle` for the same grid levels in different AMR setups will change the output times. Data is only dumped at time steps of the coarsest grid. E.g. if the coarse grid has 2^3 cells in each direction and refinement up to 2^{10} is possible and `nsubcycle=3*1,5*2` are used, the output times will be more sparse and probably differ more strongly from the desired output times than if the coarsest allowed grid has 2^5 cells, the finest possible grid has again 2^{10} cells and `nsubcycle=3*1,3*2` is used. In this simulation the whole computational box is always refined beyond 2^4 cells in each direction.

With the default setting for `nsubcycle` the actual times of the data dumps vary between the simulations with different slope limiters and Riemann solvers as shown in Fig. 4.7. In comparison in Fig. 4.6 the time-step of the 2^{10} grid was also used for all coarser grids.

To avoid problems arising from the difference between the actual and the desired output times, the analytic solution (Sect. 4.2.1) was calculated for the specific end times of the individual simulations. Obviously the time dependence affects the locations of the zone boundaries. The small time differences between the data dumps also have a slight effect on the slopes in the rarefaction wave.

A zoom in on the residuals near the contact discontinuity is shown in Fig. 4.6. The MonCen flux limiter produces a less smeared out contact discontinuity than the MinMod flux limiter but at the price of over-oscillations. In this test, this can be seen in the residuals for the LLF solver, displayed in the right upper panel in Fig. 4.6. Under "messier" conditions, like near the aforementioned CD in stellar winds and supernova bubbles, also the HLLC solver sometimes happens to run into negative densities and crash the simulations. Hence we used the HLLC solver with MonCen limiting unless we ran into problems. In this case, we restarted with HLLC and MinMod.

Figure 4.5 shows the result of HLLC and MonCen, which is the most accurate setup in the set displayed in Fig. 4.6. The purple line depicts the solution for a conservative passive scalar. For our purposes, it is interesting to check how diffusive the contact discontinuity is in different numerical schemes, since this diffusivity affects the spatial distribution of our trace elements. Furthermore mixing across the CD enhances cooling losses, since dense, but cold gas and hot dilute gas will mix and lead to a dense, warm, efficiently cooling gas phase. In this setup mixing is found in about 15 cells near the contact discontinuity at $t = 0.25$, as shown in Fig. 4.8.

Since RAMSES can also treat 2D and 3D cases, we have also tested the dependence of the shock on the orientation of the grid. Therefore in 2D the shock tube test was once set up with the discontinuity parallel to a grid axis and once with the discontinuity along the grid diagonal. In

3D these two orientations of the shock and also a discontinuity parallel to the space diagonal were tested.
The 2D and 3D results for diagonal and parallel shocks (Fig. 4.9 and 4.10) were in good agreement. However, it is interesting that the diagonal shocks have a steeper contact discontinuity than the parallel shocks – even though the distance between cell centers along the diagonal is larger than along the axis (and lower resolution enhances numerical diffusion) and also despite the fact that diagonal flows also have higher numerical diffusion than parallel flows. This looks like an effect of the flux limiter.

4.3 Sedov-Taylor blast wave test

The Sedov-Taylor test follows the expansion of a blast wave. The blast wave is created by depositing a huge amount of energy in a very small volume in a very short time. In the context of this thesis the obvious astrophysical application of strong shock waves created in this way are early phases of supernova (SN) explosions. To be concise, the Sedov-Taylor blast wave describes the adiabatic expansion phase of the SN remnant in which cooling losses are still irrelevant. This phase follows the initial free expansion (with a duration of the order of few tens of years, which ends when the swept up mass equals the ejected mass) and is expected to last of the order of 10^4 years.

4.3.1 Analytic solution of the Sedov-Taylor blast wave

The analytical blast wave solution was independently discovered by several authors (Taylor, 1950; Von Neumann, 1963; Sedov, 1993). The Sedov-Taylor blast wave is a self-similar problem and can be tackled via dimensional analysis. For this purpose one assumes that the pressure of the ambient medium is negligible ($p_{\mathrm{right}} = 0$) and that the ambient medium is at rest ($v_{\mathrm{right}} = 0$). Under these assumptions the only remaining parameters for an estimate of the time dependent shock radius are the ν-dimensional mass density of the ambient medium (ρ_{right}, with the unit [mass/length$^\nu$]), the deposited amount of energy (E_0 with the unit [mass length^2/time^2]) and of course time (t). The dimensions of these quantities are:

$$\begin{aligned} [\rho_{\mathrm{right}}] &= ML^{-\nu} \\ [E_0] &= ML^2T^{-2} \\ [t] &= T \end{aligned}$$

where ν indicates the number of dimensions. Under the aforementioned assumptions, it is thus possible to convert distance, density, energy and time into a dimensionless variable λ:

$$\lambda = r \left(\frac{E_0}{\rho_0} \right)^{-\frac{1}{2+\nu}} t^{-\frac{2}{2+\nu}} \quad . \tag{4.19}$$

This dimensionless parameter can now be used to calculate how a change in one of these quantities influences the others. The equations for gas-dynamic parameters in a shock-front in a gas with the

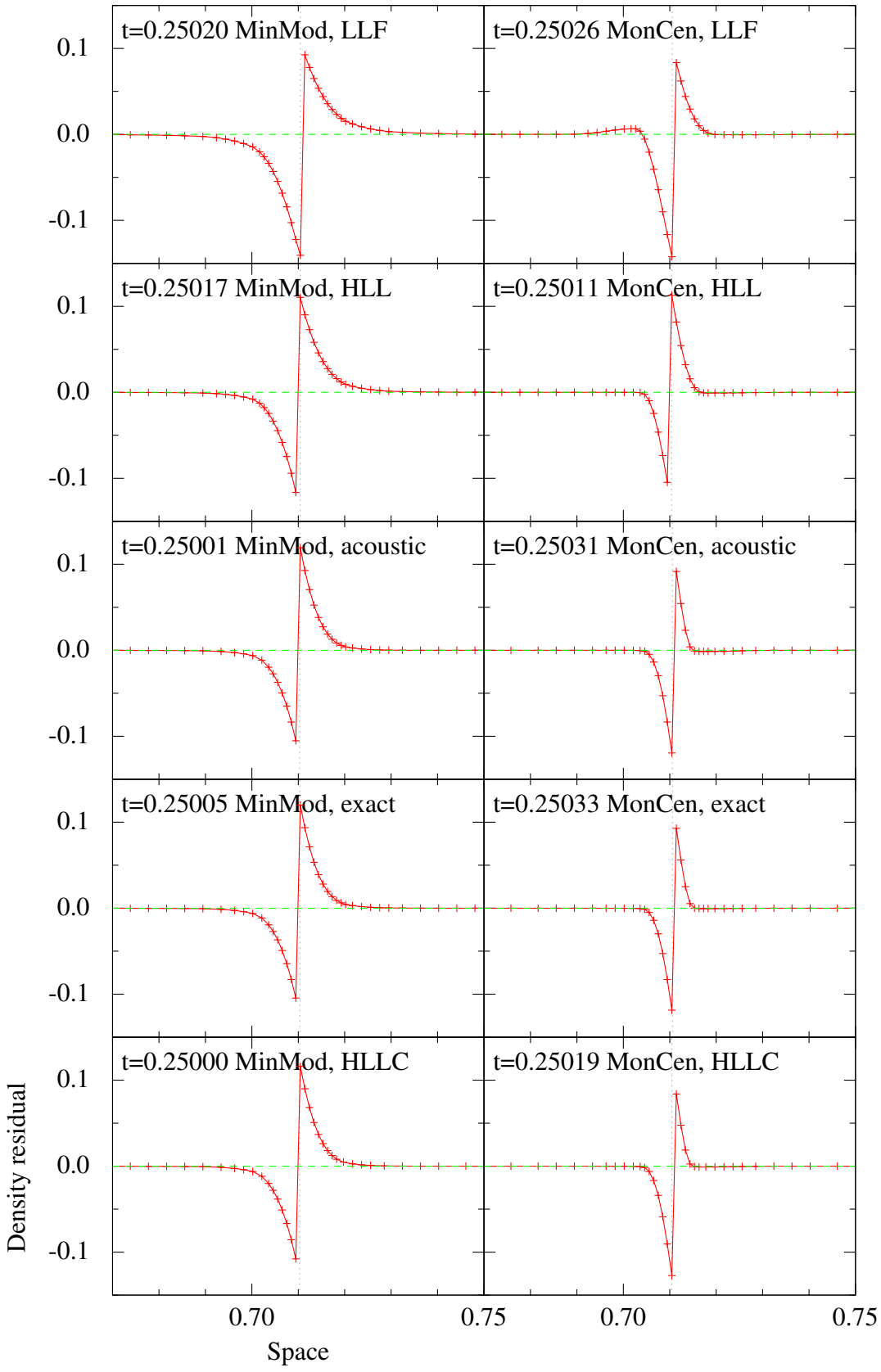


Figure 4.6: Density at the contact discontinuity in a Sod shock tube test. The analytic solution at the time of the data dump was subtracted from the numerical results obtained with the RAMSES code. Rows show different Riemann solvers and columns show different flux limiters (MinMod and MonCen). HLLC + MonCen (lower right corner) is the preferred choice for our simulations.

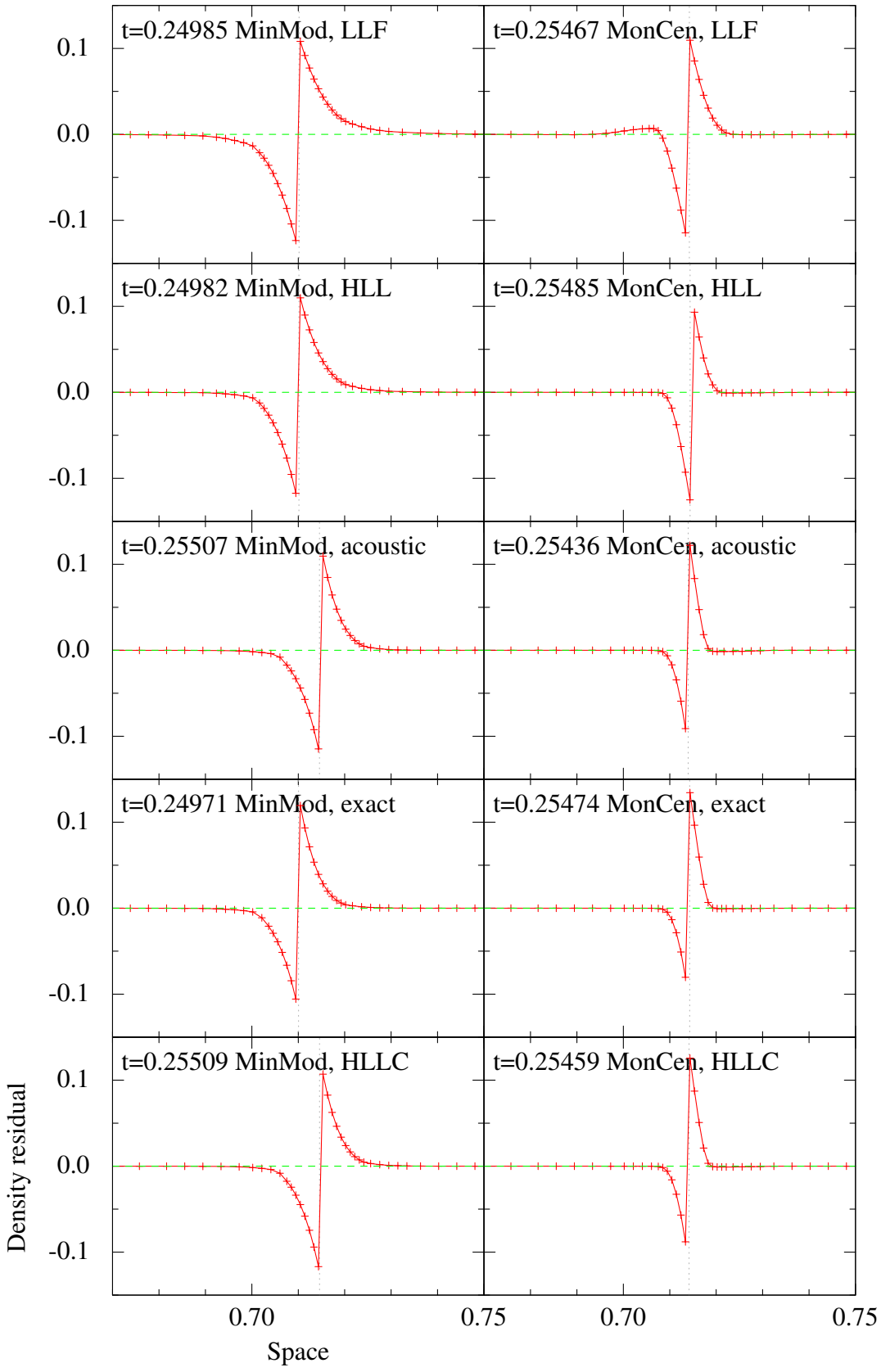


Figure 4.7: Same as Fig. 4.6 but with the default setting of `nsubcycle` and thus differing output times. The differences in the locations of CD (dotted lines) are best seen in the left lower three plots. This plot motivates, why we went through the analytic Sod solution before this test with the aim to identify the Riemann solver, which is best suited for our task.

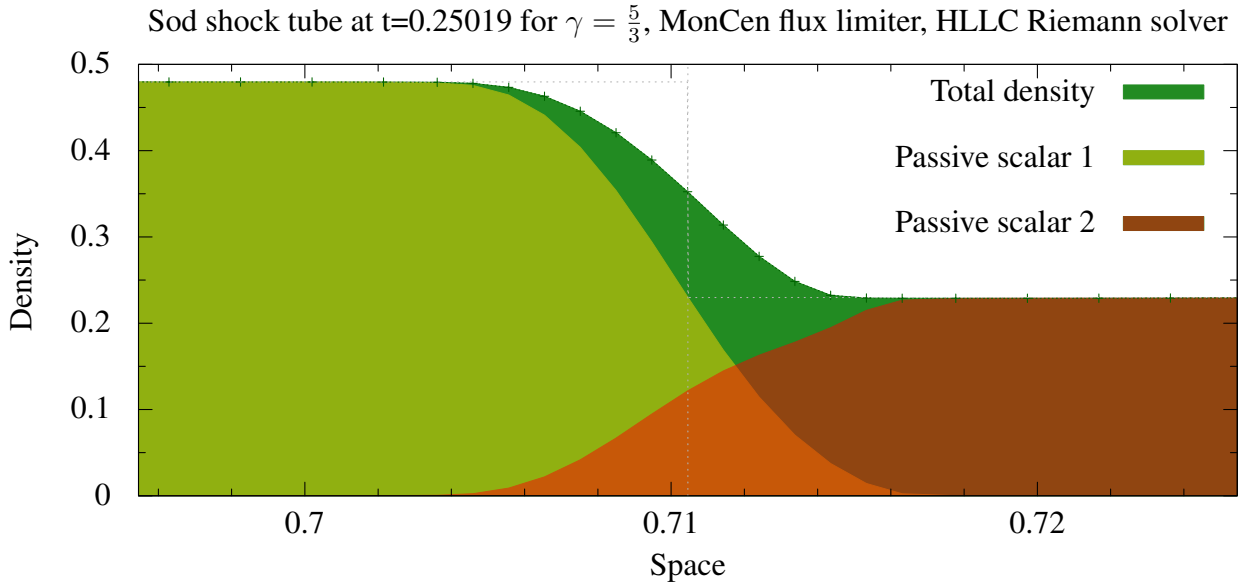


Figure 4.8: Zoom in on the density near the contact discontinuity of the Sod shock tube test with the HLLC Riemann solver with MonCen flux limiting. Dotted lines show the analytic solution. At $t = 0.25$ the media mix in ~ 15 cells in the vicinity of the CD.

EOS shown in Eq. 3.2 are:

$$r_{\rm shock} = \left(\frac{E_0}{\alpha\rho_0}\right)^{\frac{1}{2+\nu}} t^{\frac{2}{2+\nu}} \tag{4.20}$$

$$v_{\rm shock} = \frac{2}{2+\nu}\left(\frac{E_0}{\alpha\rho_0}\right)^{\frac{1}{2+\nu}} t^{-\frac{\nu}{2+\nu}} \tag{4.21}$$

$$p_{\rm shock} = \frac{2\rho_0}{\gamma+1}\frac{4}{(2+\nu)^2}\left(\frac{E_0}{\alpha\rho_0}\right)^{\frac{2}{2+\nu}} t^{-\frac{2\nu}{2+\nu}} \quad . \tag{4.22}$$

To obtain the numerical value of the constant α (of order unity) we need to find the structure of the solution inside the bubble, since α is found iteratively by integrating the energy in the bubble. $alpha$ is then adjusted until the desired input energy is reached.

In the scope of this thesis, the internal structure of the solution of the Sedov-Taylor problem is interesting, since it yields the kinetic to thermal energy ratio. With a few changes a similar procedure can be used to find the thermal to kinetic energy ratio in wind blown bubbles (Sect. 4.4.1).

We will only show the procedure used for the code tests in this thesis. The reader interested in analytic functions for the structure of the solution inside the bubble is referred to Sedov (1993, chapter 4 and pages 261 to 276). However, also they need to solve a part of the problem numerically.

For the internal structure, we exploit the symmetry of the problem and use the spherically sym-

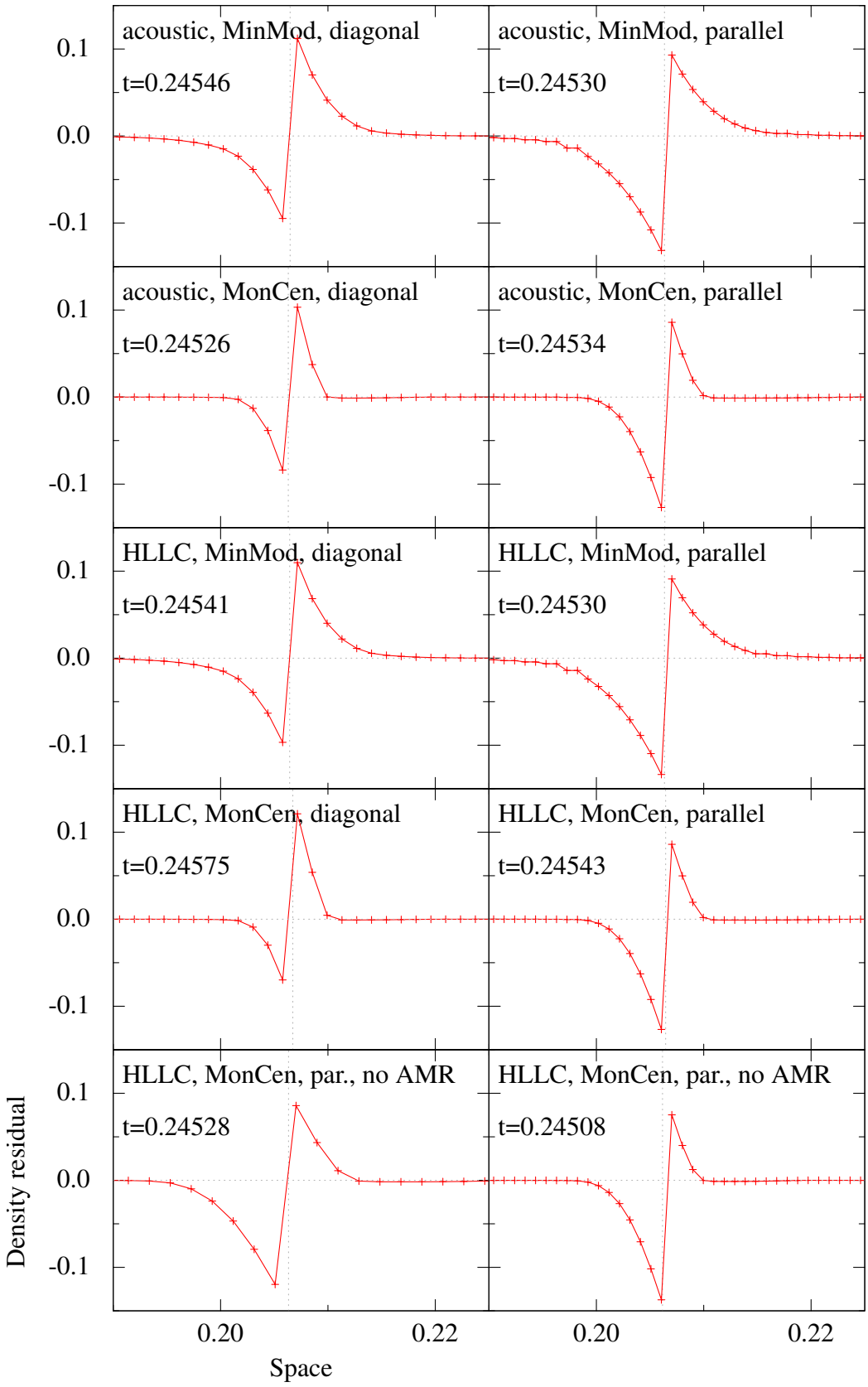


Figure 4.9: CD in a 2D Sod shock tube test. See also Fig. 4.6. On uniform grids (bottom panels), lower resolution (left) enhances numerical diffusion in the parallel shock. However, diagonal shocks exhibit a steeper CD (i.e. narrower region with nonzero residuals) than parallel shocks – even though the distance between cell centers along the diagonal is larger than along the axis and also despite the fact that diagonal flows lead to more numerical diffusion than parallel flows.

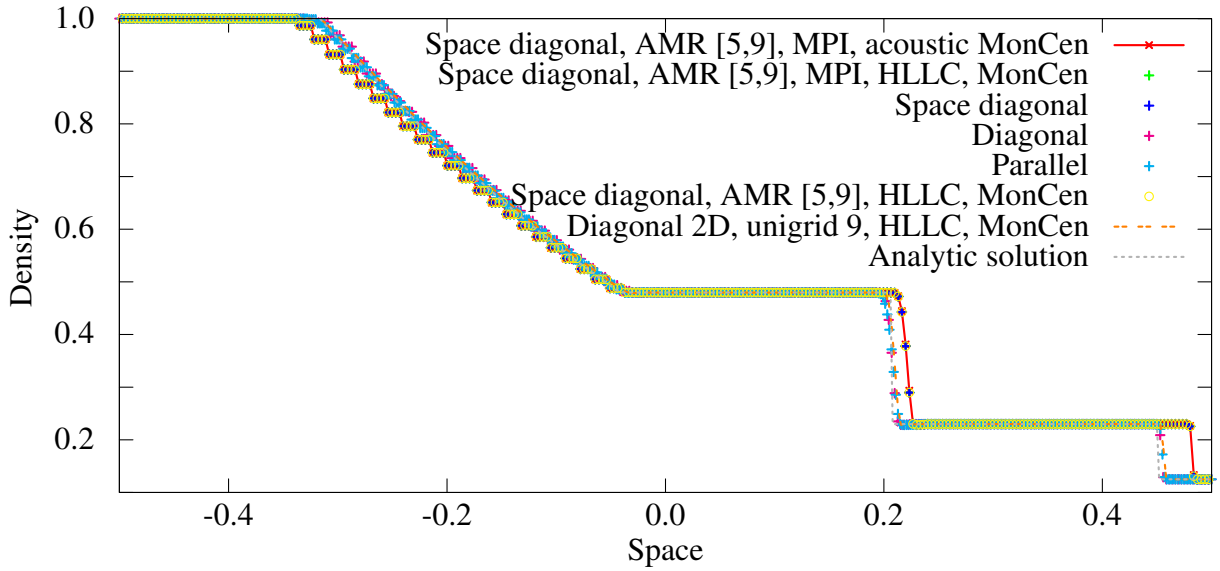


Figure 4.10: 3D Sod shock tube test.

metric Euler equations with an ideal EOS

$$\frac{\partial \rho}{\partial t} + v\frac{\partial \rho}{\partial r} + \rho\frac{\partial v}{\partial r} + 2\frac{\rho v}{r} = 0 \tag{4.23}$$

$$\frac{\partial v}{\partial t} + v\frac{\partial v}{\partial r} + \frac{1}{\rho}\frac{\partial p}{\partial r} = 0 \tag{4.24}$$

$$\frac{\partial p\rho^{-\gamma}}{\partial t} + v\frac{\partial p\rho^{-\gamma}}{\partial r} = 0 \tag{4.25}$$

Next we change the variables from (r, t) to λ (Eq. 4.19). Scale-similarity, the ideal EOS (Eq. 3.2) and the conservation laws in the rest frame of the shock (c.f. Eq. 3.3 to 3.5) permit us to write the quantities at the location of the shock as:

$$\rho_{\text{right}} v_{\text{shock}} = \rho_{\text{left}} \left(v_{\text{left}} - v_{\text{shock}}\right) \tag{4.26}$$

$$\rho_{\text{right}} v_{\text{shock}}^2 = \rho_{\text{left}} \left(v_{\text{left}} - v_{\text{shock}}\right)^2 + p_{\text{left}} \tag{4.27}$$

$$v_{\text{shock}}^2 = \left(v_{\text{left}} - v_{\text{shock}}\right)^2 + \frac{2\gamma}{\gamma - 1}\frac{p_{\text{left}}}{\rho_{\text{left}}} \quad . \tag{4.28}$$

Which can be rewritten to:

$$\rho_{\text{left}} = \frac{\gamma + 1}{\gamma - 1}\rho_{\text{right}} \tag{4.29}$$

$$v_{\text{left}} = \frac{2}{\gamma + 1} v_{\text{shock}} \tag{4.30}$$

$$p_{\text{left}} = \frac{2}{\gamma + 1}\rho_{\text{right}} v_{\text{shock}}^2 \quad . \tag{4.31}$$

These values are now used as boundary conditions at the shock. The structure inside the bubble is

self-similar and can be described with the functions $G(\lambda), U(\lambda), P(\lambda)$:

$$\rho(r,t) = G(\lambda)\frac{\gamma+1}{\gamma-1}\rho_{\rm right} \quad (4.32)$$

$$v(r,t) = U(\lambda)\frac{2}{\gamma+1}v_{\rm shock} \quad (4.33)$$

$$p(r,t) = P(\lambda)\frac{2}{\gamma+1}\rho_{\rm right}v_{\rm shock}^2 \quad . \quad (4.34)$$

These substitutions are then inserted into Eq. 4.23 to 4.25. We use $\lambda = r/r_{\rm shock}$, $\frac{\partial}{\partial t} = \frac{\partial\lambda}{\partial t}\frac{\partial}{\partial\lambda} = -\lambda\frac{v_{\rm shock}}{r_{\rm shock}}\frac{\partial}{\partial\lambda}$ and $\frac{\partial v_{\rm shock}}{\partial t} = \frac{-\nu}{2}\frac{v_{\rm shock}^2}{r_{\rm shock}}$ and find:

$$\begin{aligned}
\left(U(\lambda)-\lambda\frac{\gamma+1}{2}\right)\frac{1}{G(\lambda)}\frac{\partial G(\lambda)}{\partial\lambda}+\frac{\partial U(\lambda)}{\partial\lambda}+\frac{2U(\lambda)}{\lambda} &= 0\\
\left(\frac{2}{\gamma+1}U(\lambda)-\lambda\right)\frac{\partial U(\lambda)}{\partial\lambda}-\frac{\nu}{2}U(\lambda)+\frac{\gamma-1}{\gamma+1}\frac{1}{G(\lambda)}\frac{\partial P(\lambda)}{\partial\lambda} &= 0\\
-\nu P(\lambda)+\left(\frac{2}{\gamma+1}U(\lambda)-\lambda\right)\frac{-\gamma P(\lambda)}{G(\lambda)}\frac{\partial G(\lambda)}{\partial\lambda}-\left(\frac{2}{\gamma+1}U(\lambda)-\lambda\right)\frac{\partial P(\lambda)}{\partial\lambda} &= 0
\end{aligned}$$

For our Sedov-Taylor tests we used Mathematica to solve this set of three 1st order, coupled linear differential equations (Code Listing A.1). We iteratively solved for the value of the constant α until the numerical integration of the energy from $\lambda_{\rm shock}$ where $G(\lambda_{\rm shock}) = U(\lambda_{\rm shock}) = P(\lambda_{\rm shock}) = 1$ equaled the released energy:

$$E_0 = (\nu-1)2\pi\int_0^{r_{\rm shock}}\rho(r)\left(\varepsilon+\frac{v^2(r)}{2}\right)r^{\nu-1}{\rm d}r \quad . \quad (4.35)$$

This is shown in Code Listing A.2. As soon as α is found, also the location of the shock front ($\lambda_{\rm shock}$) and the pressure, density and velocity near the shock are known. The dimensionfree solution for the pressure, density and velocity inside the bubble is shown in Fig. 4.11. Code Listing A.2 led to $\alpha = 0.507566, r_{\rm shock}(0.688) = 0.986136$ for $t = 0.688$. We checked the Mathematica results for α for different evolution times and different numbers of dimensions with the code of Haque (2006) and found no problems.

Thermal energy fraction

The self-similar solution of the Sedov-Taylor expansion in a uniform medium without mass loading lead to a thermal energy fraction of 71.7% E_0 which is in accordance with Chevalier (1974); McKee and Ostriker (1977); Ostriker and McKee (1988). This fraction will be used in our setup of the blast waves.

4.3.2 Initial conditions of the Sedov-Taylor blast wave test

The initial conditions for the Sedov-Taylor explosion consist of a sphere with an internal energy of 10^{51} erg placed in a homogeneous medium.
As shown in Sect. 4.3.1, the equations for the gas-dynamic parameters in the shock front depend on the number of dimensions taken into account. Thus, for our 2D RAMSES models, the performance of the solvers at the Sedov-Taylor problem was tested with axisymmetric explosions with

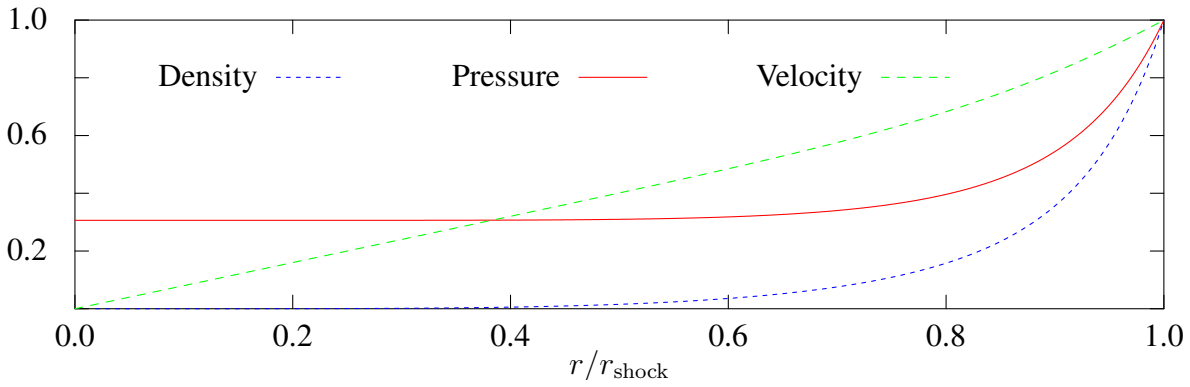


Figure 4.11: Internal pressure, density and velocity structure of the Sedov-Taylor blast's bubble as obtained with Code Listing A.1

only one layer of cells plus two layers of ghost cells for the boundary conditions in z-direction. The simulations start at time $t = 3.46 \times 10^{-4}$ [code-time-units]. At the start of the simulation the internal energy $E_{\rm tot} = 1$ [code-mass-unit code-length-unit^2/code-time-unit^2] is assumed to be stored inside a circular region with radius $r = 0.02$ [code-length-units] and constant energy density. The start time $t = 3.46 \times 10^{-4}$ [code-time-units] was chosen, because the iterative solution (see Sect. 4.3.1) carried out with the code of Haque (2006) with dimensionless coordinates and automatically chosen initial α (started from command line using: `./sedov sedov.param.start -v -auto`) yielded $\alpha = 0.56$ for a two dimensional blast with energy $E_{\rm tot} = 1$ [code-mass-unit code-length-unit^2/code-time-unit^2] in an initially homogeneous ambient medium with $\rho = 1$ [code-mass-unit/code-length-unit$^\nu$] and adiabatic exponent $\gamma = \frac{5}{3}$. Hence according to Eq. 4.20 the two dimensional shock front is at $r = \sqrt[4]{\frac{t^2}{\alpha}} = 0.02$ [code-length-units] at this time. The shape, size and resolution of the region into which the blast energy is inserted have a strong influence on the results (see also Sect. 5.2.1 and Sect. 5.2.3 on the feedback region). To insert the given total energy, the volume weighted sum of the energy density inside this region has to be the desired total energy divided by the volume of the region. Let us assume that all the blast energy would be stored in just four cells with cell-lengths of 0.02 [code-length-units]. In this case these four cells get internal energies of $E_{\rm therm,i} = 625$ [code-mass-unit code-length-unit/code-time-unit^2], because $E_{\rm tot} = 1 = \sum E_{\rm therm,i} \left(\Delta x\right)^2 = 4 \times 625 \times 0.02^2$. Of course this cannot be done in a real simulation, because the grid would create an x-shaped outflow rather than an axisymmetric outflow. In the simulations the feedback region radius is at least 8 cell-lengths and the internal energy of the cells inside the spherical feedback region is $E_{\rm therm,i} = w_i \frac{E_{\rm tot}}{(\Delta x)^2}$ with weights w_i that account for the fact that near the border of the region only a part of the cell i might be inside the feedback region. The weights are normalized: $\sum w_i = 1$.

4.3.3 Results of the Sedov-Taylor blast wave test

The Sedov-Taylor test is already relatively close to our production runs without winds. The simplifications (compared to the production runs) are: (1) the Sedov-Taylor test ignores radiative cooling losses and (2) the ambient pressure is unimportant for Sedov-Taylor blasts. The Sedov-Taylor test shows us, (1) that our prescription of energy deposits actually manages to insert the desired amount of energy into the computational box and (2) that energy-, mass- and momentum conser-

vation works under this conditions (which are already close to the production runs) and (3) the diffusivity of the numerical schemes. As for the Sod shock tube test, we ran this test with all solvers and flux limiters. Since initial conditions for this test are distributed both with the PLUTO and RAMSES code, we do not include the plots here. Basically, the test results reach the same conclusion (HLLC + MonCen) as the Sod shock tube test.

4.4 Theories of stellar winds

The feedback of a group of stars is dominated by massive stars[7]. This is, e.g., shown in Fig. 2.18, where the energy input from the most massive still existing star dominates. However, it is also obvious if one considers the high luminosity of massive stars, the fast evolution (i.e. rather early SN) the energy input from SN events (10^{51} erg) and the kinetic energy and mass loss rates of WR winds.

Massive stars shape the medium surrounding them via ionizing radiation, stellar winds and SN explosions. This work will not treat the effects of the ionizing radiation. Although the energy in the stellar winds is about two orders of magnitude smaller than the radiative energy (e.g. Voss et al., 2009; Ekström et al., 2012), stellar winds are very efficient in heating the surrounding ISM: whereas the temperature in the Strömgren sphere is of the order of 10 000 K, temperatures in the shocked wind gas and the shocked ISM can be $> 10^7$ K (see also Lamers and Cassinelli, 1999, chapter 12.1, page 357).

Stellar winds are a flow of particles escaping from a star. They are characterized by their mass loss rate $\dot{M}$ and the terminal velocity v_∞ which is the velocity of the wind particles at a large distance from the star. Winds of massive stars exhibit terminal velocities of the order of 10^4 km/s (Lamers et al., 1995; Leitherer et al., 1999; Niedzielski and Skorzynski, 2002, see also Tab. 2.4 and Fig. 2.12). The expected mass loss rates for a 40 $M_\odot$ star are of the order of 10^{-6} $M_\odot$ per year before the WR phases and up to $> 10^{-4}$ $M_\odot$ per year during the WR phases (e.g. Meynet et al., 1994; Meynet and Maeder, 2003; Ekström et al., 2012).

Voss et al. (2009) and also Abbott (1982) showed that the total energy input of winds of massive stars is of the same order as the energy released in the SN event (10^{51} erg).

4.4.1 Wind theory of Castor et al. (1975)

The wind theory of Castor et al. (1975) describes an idealized spherically symmetric stellar wind, which starts at $t = 0$. It is characterized by its constant terminal velocity $v_{\rm wind}$ and constant mass loss rate $\dot{M}_{\rm wind}$. When it flows into an ISM with not-negligible uniform density n_0, the interaction of the wind with the ISM creates a two-shock structure.

After the initial free streaming phase, which lasts for about < 100 yr and ends when swept up mass equals the wind mass, the wind spends of the order of 1 000 yr in an adiabatic phase. This phase ends, when the cooling time equals the evolutionary time. Consequently the stellar wind transits to the snowplow phase, which lasts longer than the aforementioned phases. Finally dissipation destroys the wind bubble. For our study we are interested in the snowplow phase, since our aim is to find out, how much of the feedback energy is lost via radiative cooling.

During the snowplow phase the structure of the wind can be subdivided into 4 zones (see also Fig. 4.12):

[7] massive stars are defined as stars with high enough masses to undergo a SN explosion (not type Ia).

(1) **Supersonic free streaming wind**
The stellar wind drives a wave into the ISM. In this zone the radius-dependent density is $\rho_{\rm wind}(r) = \frac{\dot{M}_{\rm wind}}{4\pi r^2 v_{\rm wind}}$, since the mass per shell is constant. Also the temperature T in this zone is constant and lower than in zone (2). A pressure less gas would move with constant velocity $v_{\rm wind}$. As shown e.g. in Fig. 4.15, the velocity profile in this region is $v(r) \propto 1 - \frac{1}{r^2}$ and the motion is supersonic.

The observational data of Gruendl et al. (2000) shows a clear offset between the $H\alpha$ emission in the free streaming wind in and the O[III] emission in the shocked wind in RCW 58. This suggests that this region in WR bubbles can indeed span a few parsecs in low density environments.

(2) **Hot layer with shocked wind**
The shocked wind layer is separated from the free wind region by the "inward facing shock"[8]. During the transition through this reverse shock at the interface of zone (1) and (2) the gas is heated and compressed. The density ρ increases by a factor ~ 4, as follows from the shock-jump-conditions (see Sect. 4.1.3) for high Mach numbers and the velocity decreases as can be seen from mass conservation: $\rho_1 : \rho_2 = u_2 : u_1$. The hot shocked wind zone is almost isobaric and contains also a small fraction of swept up ISM gas besides the heated wind.

The wind adds energy at rate $\dot{E} = \frac{\dot{M}_{\rm wind} v_{\rm wind}^2}{2}$ via a collision-less shock at small radii or via Coulomb stopping of wind ions inside this zone.

With the mass loss $\dot{M}_6$ in $10^{-6} M_\odot$/ year the v_{2000} wind velocity in 2000 km/s and t_6 time in Myr Castor et al. (1975) find:

- $n = 0.01 n_0^{19/35} \left(\dot{M}_6 v_{2000}^2\right)^{6/35} t_6^{-22/35} {\rm cm}^{-3}$
- $T = 1.6 \times 10^6 n_0^{2/35} \left(\dot{M}_6 v_{2000}^2\right)^{8/35} t_6^{-6/35} K$
- $L_{\rm x-ray} = 3.8 \times 10^{33} n_0^{18/35} \left(\dot{M}_6 v_{2000}^2\right)^{37/35} t_6^{-16/35} {\rm erg/s}$

Since the sound speed rises with T the Mach number in this zone is lower than in the free wind region. Thus, the flow is subsonic in the hot region (2) and supersonic in zone (1).

(2-3) **Contact discontinuity (CD)**
The CD at radius $R_{\rm CD} = 0.86 R_{\rm s}$ separates wind material from swept up ISM. The expansion velocity $\dot{R_{\rm CD}}$ is the same on both sides of the CD but a density jump is observed. This zone is numerically challenging, since the resolution is problematic in this thin shell. However, this zone is very important for our simulations: on the one hand it contains a large amount of compressed swept up medium, since ρ is highest at the CD and on the other hand, energy losses peak there and T at the CD is lowest, since radiative cooling becomes very efficient at high ρ.

The hot gas (separated by the CD) expands into region (4) and back into region (1) causing a two-shock structure.

[8] The direction of a shock is: hot medium $\rightarrow$ cold medium. At the interface (1) to (2) this is thus "inward facing".

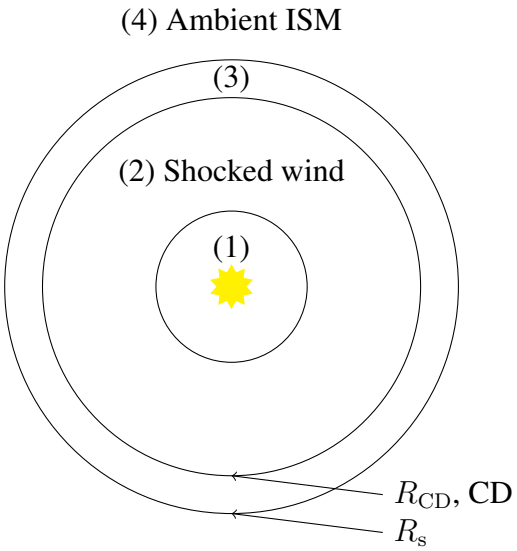


Figure 4.12: Structure of the wind bubble during the snowplow phase: ring structure with (1) free wind, (2) shocked wind, (3) swept up ISM and (4) ambient medium. The location of the CD ($R_{\rm CD}$) is also indicated.

(3) **Hot shocked ISM**
The hot layer containing swept up and heated ISM is separated from region (2) through a contact discontinuity and through a shock, where the ISM is compressed and heated, from region (4). The gas moves at the same velocity as in zone (2). Analogous to the inward facing shock, we also find a ρ jump by a factor 1/4 at the interface (3) to (4). This interface is located at $R_{\rm s} = 0.76 \left(\frac{\dot{E}_0 t^3}{\rho_0}\right)^{1/5} = 28 \left(\frac{\dot{M}_6 v_{2000}^2}{n_0}\right)^{1/5} t_6^{3/5}$ pc.

(4) **Undisturbed ISM**
The ambient, low temperature medium is assumed to be at rest.

4.4.2 Thin shell approximation

During the snowplow phase the width of the zone containing the shocked ISM is much smaller than the radius of the bubble. Therefore the thin shell approximation can be used to describe the evolution of the pressure driven shell.
Since we will need winds in 1D, 2D and 3D, we will use n-spheres for the thin shell approximation. Basically a ν-dimensional sphere with radius r has a surface

$$S_{\nu-1} r^{\nu-1} = \frac{2\pi^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)} r^{\nu-1} \tag{4.36}$$

and a volume

$$V_\nu r^\nu = \frac{2\pi^{\frac{\nu}{2}}}{\nu\Gamma\left(\frac{\nu}{2}\right)} r^\nu \tag{4.37}$$

where Γ is the gamma function with $\Gamma(\frac{1}{2}) = \sqrt{\pi}$,$\Gamma(1) = 1$ and $\Gamma(x+1) = x\Gamma(x)$. We consider stellar wind bubbles that are placed in a homogeneous ambient medium with density ρ_0. We now

assume that all mass inside the shell radius $R_\mathrm{s}(t)$ has been swept up into a thin, dense, high pressure shell. We further assume that the pressure inside the shell is so much larger than the pressure in the shocked wind region that it can be ignored in the momentum conservation of the compressed shell (Eq. 4.38). With n-spheres equating the rate change of momentum with the pressure force in ν dimensions can be written as:

$$\begin{aligned} \frac{\mathrm{d}Mv}{\mathrm{d}t} = & \quad S_{\nu-1} R_\mathrm{s}^{\nu-1} p && \text{(momentum conservation)} \\ & M = \rho_0 V_\nu R_\mathrm{s}^\nu && \text{(mass of swept up medium in shell)} \\ & p = (\gamma-1)\,\frac{E_\mathrm{th}}{V_\nu R_\mathrm{s}^\nu} && \text{(EOS, Eq. 3.2)} \\ & v = \tfrac{\mathrm{d}R_\mathrm{s}}{\mathrm{d}t} && \text{(rate of bubble expansion)} \end{aligned} \tag{4.38}$$

$$\frac{\mathrm{d}\rho_0 V_\nu R_\mathrm{s}^\nu \frac{\mathrm{d}R_\mathrm{s}}{\mathrm{d}t}}{\mathrm{d}t} = (\gamma-1)\, S_{\nu-1} R_\mathrm{s}^{\nu-1} \frac{E_\mathrm{th}}{V_\nu R_\mathrm{s}^\nu} \quad . \tag{4.39}$$

If one assumes that the shell's radius R_s and the total thermal energy E_th follow the power laws $R_\mathrm{s}(t) \propto t^a$ and $E_\mathrm{th}(t) \propto t^b$, one can compare the exponents of t in Equation 4.39. This leads to

$$\begin{aligned} (\nu a + (a-1)) - 1 &= (\nu-1)\,a - \nu a + b \\ a &= \frac{b+2}{\nu+2} \end{aligned} \tag{4.40}$$

and thus $R_\mathrm{s}(t) \propto t^{(b+2)/(\nu+2)}$. The cumulative thermal feedback energy $E_\mathrm{th}(t)$ turns out to be a fixed fraction of the cumulative total feedback energy $E(t) = L_\mathrm{wind} t^b$. Where the exponent b discriminates several energy input modes: The energy inserted in a blast of a SN explosion would be described with $b = 0$ and hence $E(t) = E_0 =$ constant. The cumulative feedback energy of a constant wind with luminosity L_wind as described by Castor et al. (1975) is $E(t) = L_\mathrm{wind} t$ with $\frac{\mathrm{d}L_\mathrm{wind}}{\mathrm{d}t} = 0$. Sequential star formation can be described with $E(t) = L_\mathrm{wind} t^2$.

Under the assumption that $p\mathrm{d}V$ work is the dominant thermal energy loss, the relation between the kinetic and the thermal energy can be found from energy conservation. I.e. the increase of the total energy $\dot{E}$ equals the change of the kinetic and thermal energy:

$$\begin{aligned} \frac{\mathrm{d}E_\mathrm{total}}{\mathrm{d}t} &= \frac{\mathrm{d}E_\mathrm{th}}{\mathrm{d}t} + \frac{\mathrm{d}E_\mathrm{kin}}{\mathrm{d}t} \\ b L_\mathrm{wind} t^{b-1} &= b L_\mathrm{wind} t^{b-1} - E_\mathrm{th}\,(\gamma-1)\,\frac{\nu}{R_\mathrm{s}}\frac{\mathrm{d}R_\mathrm{s}}{\mathrm{d}t} + E_\mathrm{kin}\left(\frac{\nu}{R_\mathrm{s}}\frac{\mathrm{d}R_\mathrm{s}}{\mathrm{d}t} + \frac{2}{v_\mathrm{s}}\frac{\mathrm{d}v_\mathrm{s}}{\mathrm{d}t}\right) \\ 0 &= -E_\mathrm{th}\frac{(\gamma-1)\,\nu\,(b+2)}{\nu+2} + E_\mathrm{kin}\frac{\nu\,(b+2) + 2\,(b-\nu)}{\nu+2} \\ E_\mathrm{kin} &= E_\mathrm{th}\frac{(\gamma-1)\,\nu\,(b+2)}{b\,(\nu+2)} \\ L_\mathrm{wind} t^b &= E_\mathrm{th}\frac{b\,(\nu+2) + (\gamma-1)\,\nu\,(b+2)}{b\,(\nu+2)} \\ E_\mathrm{th} &= \frac{b\,(\nu+2)}{b\gamma\nu + 2b + 2\gamma\nu - 2\nu} L_\mathrm{wind} t^b \quad . \end{aligned} \tag{4.41}$$

For a constant energy input ($b = 1$, $E \propto t$) in 3D with $\gamma = \frac{5}{3}$ this yields $E_\mathrm{th} = \frac{5}{11} L_\mathrm{wind} t$ and Eq. 4.40 leads to $R_\mathrm{s}(t) \propto t^{3/5}$. In 2D the power law is $R_\mathrm{s}(t) \propto t^{3/4}$ and the thermal fraction

is $E_{\rm th} = \frac{1}{2} L_{\rm wind} t$. Sequential star formation ($b = 2$, $E_{\rm th} \propto t^2$) leads to $R_{\rm s}(t) \propto t^{4/(\nu+2)}$ i.e. in 2D $R_{\rm s}(t) \propto t$ and in 3D $R_{\rm s}(t) \propto t^{4/5}$. For a blast ($b = 0$, constant $E = E_0$) the Sedov-Taylor solution $R_{\rm s}(t) \propto t^{2/\nu+2}$ is recovered but the constant thermal energy content of $0.72 E_0$ cannot be found from Eq. 4.41. However, as shown in Sect. 4.3.1, it can be found via an integration of the density/pressure/velocity structure.

We will now solve for the proportionality constant α using $R_{\rm s}(t) = \alpha t^{\frac{b+2}{\nu+2}}$ in the equation for momentum conservation in ν dimensions (Eq. 4.39).

$$\begin{aligned} \alpha^{\nu+1} \quad & \rho_0 V_\nu \frac{{\rm d}t^{\nu(b+2)/(\nu+2)} \frac{{\rm d}t^{(b+2)/(\nu+2)}}{{\rm d}t}}{{\rm d}t} = (\gamma-1) \frac{t^{-(b+2)/(\nu+2)}}{\alpha} \frac{S_{\nu-1}}{V_\nu} E_{\rm th} \\ \alpha^{\nu+2} \quad & = \frac{b\,(\nu+2)^3\,(\gamma-1)}{(b^2\nu + b^2 + 3b\nu + 2\nu + 2b)\,(b\gamma\nu + 2b + 2\gamma\nu - 2\nu)} \frac{S_{\nu-1}}{V_\nu^2} \frac{L_{\rm wind}}{\rho_0} \end{aligned}$$

For 3D, $\gamma = \frac{5}{3}$ and constant energy input ($b = 1$) we get $S_{\nu-1} V_\nu^{-2} = \frac{9}{4\pi}$ and $\alpha = \sqrt[5]{\frac{2\times5\times25}{4\times7\times11\pi} \frac{L_{\rm wind}}{\rho_0}}$ and thus $R_{\rm s}(t) = 0.76 \sqrt[5]{\frac{L_{\rm wind}}{\rho_0}} t^{3/5}$. The solution for the swept up shell is:

$$R_{\rm s}(t) = \left(\frac{b\,(\nu+2)^3\,(\gamma-1)\,\frac{S_{\nu-1}}{V_\nu^2}\frac{L_{\rm wind}}{\rho_0} t^{(b+2)}}{(b^2\nu + b^2 + 3b\nu + 2\nu + 2b)\,(b\gamma\nu + 2b + 2\gamma\nu - 2\nu)} \right)^{1/(\nu+2)} \tag{4.42}$$

$$v_{\rm s}(t) = \left(\frac{b\,(\nu+2)^3\,(\gamma-1)\,\frac{S_{\nu-1}}{V_\nu^2}\frac{L_{\rm wind}}{\rho_0} t^{(b-\nu)}}{(b^2\nu + b^2 + 3b\nu + 2\nu + 2b)\,(b\gamma\nu + 2b + 2\gamma\nu - 2\nu)} \right)^{1/(\nu+2)} \frac{b+2}{\nu+2} \tag{4.43}$$

Internal structure of the stellar wind bubble

In Sect. 4.3.1 the internal structure of the Sedov-Taylor blast wave has been discussed. For a point explosion we had a constant total energy $E(t) = E_0 =$ constant and a constant mass $M = M_0$ in the computational volume. If we consider a stellar wind, we have an energy source that constantly increases the ISM mass $M = M_0 + \dot{M} t$ and the total energy $E = E_0 + \frac{\dot{M} v^2}{2} t$.

We can now again use the adiabatic, ideal EOS

$$\varepsilon = \frac{p_2}{\rho_{\rm ISM}(\gamma-1)} \tag{3.2}$$

and the Rankine Hugoniot relations (Eq. 4.6 to 4.11; Lagrangian system of the shock)

$$\begin{aligned} v_2 &= \frac{2}{\gamma+1} v_{\rm SWB} \\ \rho_2 &= \frac{\gamma+1}{\gamma-1} \rho_{\rm SWB} \\ p_2 &= \frac{2}{\gamma+1} \rho_{\rm ISM} v_{\rm SWB}^2 \end{aligned}$$

to define dimensionfree functions

$$\begin{aligned} \rho(r,t) &= G(\lambda)\rho_0 \\ v(r,t) &= U(\lambda) v_2 \\ p(r,t) &= P(\lambda)\rho_0 v_2^2 \qquad . \end{aligned}$$

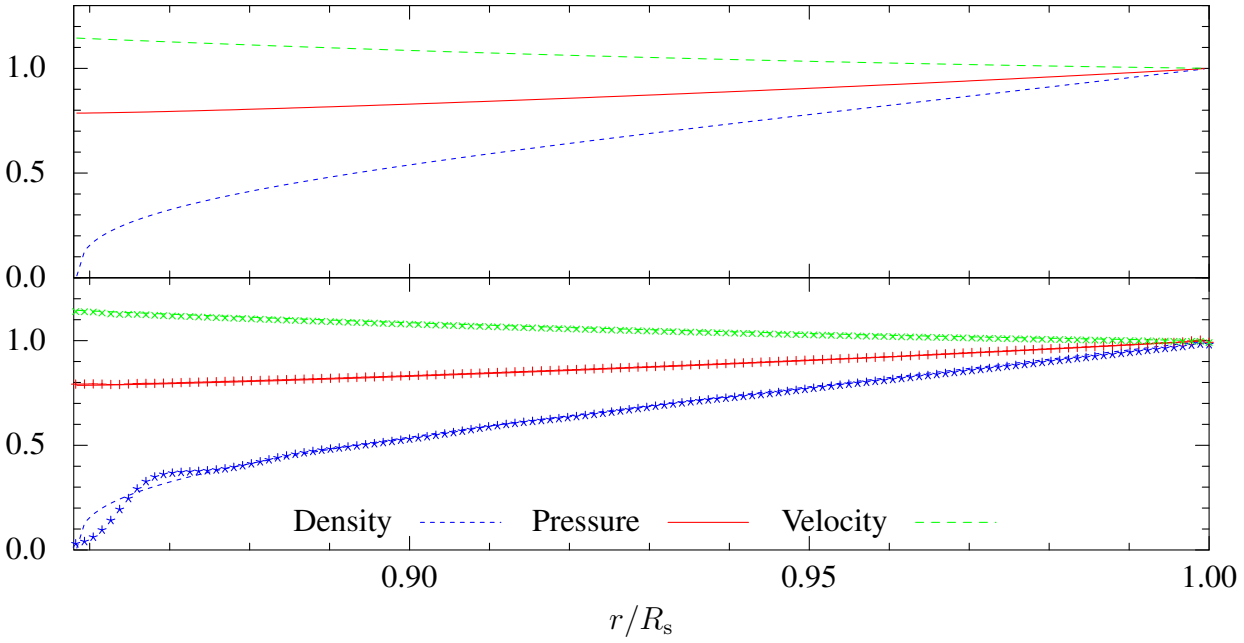


Figure 4.13: Internal pressure, density and velocity structure between the CD and the unperturbed medium as obtained with Code Listing A.3. The superplotted points on the lower panel are a PLUTO model for a wind of a 60 $M_\odot$ star without radiative cooling.

and insert them into the spherically symmetric Euler equations. We use $\lambda = r/R_s$, $\frac{\partial}{\partial t} = \frac{\partial \lambda}{\partial t}\frac{\partial}{\partial \lambda} = -\lambda \frac{v_2}{R_s}\frac{\partial}{\partial \lambda}$ and $\frac{\partial v_2}{\partial t} = \frac{-2}{\nu}\frac{v_2^2}{R_{\rm shell}}$ and find:

$$\begin{aligned}
(U(\lambda)-\lambda)\,\frac{1}{G(\lambda)}\frac{\partial G(\lambda)}{\partial \lambda} + \frac{\partial U(\lambda)}{\partial \lambda} + \frac{2U(\lambda)}{\lambda} &= 0 \\
(U(\lambda)-\lambda)\,\frac{\partial U(\lambda)}{\partial \lambda} - \frac{2}{\nu}U(\lambda) + \frac{1}{G(\lambda)}\frac{\partial P(\lambda)}{\partial \lambda} &= 0 \\
-\frac{4}{\nu}P(\lambda) + (U(\lambda)-\lambda)\,\frac{-\gamma P(\lambda)}{G(\lambda)}\frac{\partial G(\lambda)}{\partial \lambda} - (U(\lambda)-\lambda)\,\frac{\partial P(\lambda)}{\partial \lambda} &= 0
\end{aligned}$$

We solve again with Mathematica and find the structure of the bubble between the CD and the shell (Code Listing A.3 and Fig. 4.13). The thermal energy fraction in the 3D constant wind model with $\gamma = \frac{5}{3}$ is $\frac{5}{11}E_{\rm total}$ as expected from Eq. 4.41.

4.4.3 Steady-state wind of Chevalier and Clegg (1985)

The Chevalier and Clegg (1985) steady-state model basically treats the feedback region like our code: the source term in the energy conservation equation $Q = \dot{E}/V$, is the energy loss rate divided by the volume of the spherical feedback region (with radius R) $V = \frac{4\pi}{3}R^3$ and the source term in the continuity equation $q = \dot{M}/V$ is the mass loss divided by the feedback region's volume. The difference to our simulation is that this model neglects the surrounding ISM and thus no driven wave develops. The Chevalier and Clegg (1985) solution (Chevalier and Clegg, 1985, Fig. 1) is similar to the behavior of the free wind zone near the feedback region in our simulations. Comparing this zone in our simulations to the Chevalier and Clegg (1985) solution is thus a good

test for our implementation of the stellar feedback. However, the Chevalier and Clegg (1985) solution is not a good model for feedback in regions with not negligible ISM density, since it cannot describe regions with shocked wind or swept up medium.

The basic equations of the Chevalier and Clegg (1985) model are:

Continuity equation:
$$\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left(\rho u r^2\right) = q \tag{4.44}$$
with $q = \dot{M}/V$ in the feedback region and $q = 0$ elsewhere.

Momentum conservation:
$$\rho u \frac{\mathrm{d}u}{\mathrm{d}r} = -\frac{\mathrm{d}P}{\mathrm{d}r} - qu \tag{4.45}$$

Energy conservation:
$$\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left[\rho u r^2 \left(\frac{u^2}{2} + \frac{\gamma}{\gamma-1}\frac{P}{\rho}\right)\right] = Q \tag{4.46}$$
with $Q = \dot{E}/V$ in the feedback region and $Q = 0$ elsewhere.

Chevalier and Clegg (1985) assume that the energy is thermalized and hence the flow is subsonic in the feedback region. The wind speed u reaches the sound speed $c = \sqrt{\gamma\frac{P}{\rho}}$ at the radius of the feedback region (R). Outside the feedback region the flow becomes supersonic. The relation between the Mach number $M = \frac{u}{c}$ and the scaled radius r/R, is derived by integrating the above mentioned conservation laws (Eq. 4.44, 4.45 and 4.46).

These relations are:

$$\left(\frac{3\gamma + 1/M^2}{1+3\gamma}\right)^{-(3\gamma+1)/(5\gamma+1)} \left(\frac{\gamma - 1 + 2/M^2}{1+\gamma}\right)^{(\gamma+1)/[2(5\gamma+1]} = \frac{r}{R} \qquad (r < R) \tag{4.47}$$

$$M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{1+\gamma}\right)^{(\gamma+1)/[2(\gamma-1]} = \left(\frac{r}{R}\right)^2 \qquad (r > R) \quad . \tag{4.48}$$

Here we will only briefly show how the relation outside the feedback region (Eq. 4.48) can be obtained. The relation for the feedback region (Eq. 4.47) can be derived in a similar way but the algebra is more cumbersome than for the equations without source terms.

First the conservation equations without source terms are used to find the relation between pressure and density:

$$\frac{\mathrm{d}}{\mathrm{d}r}\left(\rho u r^2\right) = 0 \tag{4.49}$$

$$\rho u \frac{\mathrm{d}u}{\mathrm{d}r} = -\frac{\mathrm{d}P}{\mathrm{d}r} \tag{4.50}$$

$$u\frac{\mathrm{d}u}{\mathrm{d}r} + \frac{\gamma}{\gamma-1}\frac{\mathrm{d}\frac{P}{\rho}}{\mathrm{d}r} = 0 \tag{4.51}$$

Eq. 4.51 was simplified by using Eq. 4.49 to remove $\left[\left(\frac{u^2}{2} + \frac{\gamma}{\gamma-1}\frac{P}{\rho}\right)\right]\frac{\mathrm{d}}{\mathrm{d}r}\left(\rho u r^2\right) = 0$

Combining the momentum conservation (Eq. 4.50) with a the energy equation (Eq. 4.51) leads to

$$\begin{aligned} -\frac{1}{\rho}\frac{\mathrm{d}P}{\mathrm{d}r} + \frac{\gamma}{\gamma-1}\frac{\mathrm{d}\frac{P}{\rho}}{\mathrm{d}r} &= 0 \\ -\frac{1}{\rho}\frac{\mathrm{d}P}{\mathrm{d}r} + \frac{\gamma}{\gamma-1}\frac{1}{\rho}\frac{\mathrm{d}P}{\mathrm{d}r} + \frac{\gamma}{\gamma-1}P\frac{\mathrm{d}\frac{1}{\rho}}{\mathrm{d}r} &= 0 \\ \frac{1}{\rho}\frac{\mathrm{d}P}{\mathrm{d}r} + \gamma P\frac{\mathrm{d}\frac{1}{\rho}}{\mathrm{d}r} &= 0 \\ \text{The chain rule leads to} \quad \frac{\mathrm{d}\left(\frac{1}{\rho}\right)^{\gamma}}{\mathrm{d}r} &= \frac{\gamma}{\rho^{\gamma-1}}\frac{\mathrm{d}\frac{1}{\rho}}{\mathrm{d}r} \\ \text{the pressure density relation is } \frac{\mathrm{d}\frac{P}{\rho^{\gamma}}}{\mathrm{d}r} &= 0 \text{ or } \frac{P}{\rho^{\gamma}} = \frac{P_0}{\rho_0^{\gamma}} \end{aligned} \tag{4.52}$$

Where the subscript 0 indicates quantities at the center of the feedback region. Integrating the continuity equation (Eq. 4.49) results in

$$r^2 u\rho = R^2 u_0 \rho_0 \quad . \tag{4.53}$$

R is the radius of the feedback region. An integral over the momentum conservation (Eq. 4.50) combined with Eq. 4.52 leads to:

$$\begin{aligned} \int \mathrm{d}\frac{u^2}{2} &= -\int \frac{1}{\rho}\mathrm{d}P && \text{with } \frac{1}{\rho} = \frac{P_0^{1/\gamma}}{\rho_0}P^{-1/\gamma} \\ \frac{1}{2}\left(u^2 - u_0^2\right) &= \frac{P_0^{1/\gamma}}{\rho_0}\frac{1}{1-1/\gamma}\left(P_0^{1-1/\gamma} - P^{1-1/\gamma}\right) \\ \frac{1}{2}\left(u^2 - u_0^2\right) &= \frac{\gamma}{\gamma-1}\left(\frac{P_0}{\rho_0} - \frac{P}{\rho}\right) && \text{with } c^2 = \gamma\frac{P}{\rho} \text{ and } u_0 = c_0 \\ \frac{u^2}{u_0^2} - 1 &= \frac{2}{\gamma-1}\left(1 - \frac{c^2}{c_0^2}\right) \\ \frac{u}{u_0} &= \sqrt{\frac{\gamma+1}{\gamma-1} - \frac{2}{\gamma-1}\frac{c^2}{c_0^2}} \quad . \end{aligned} \tag{4.54}$$

The relation between pressure and density (Eq. 4.52) can now be used to rewrite the ratio between the adiabatic sound speeds $\left(c = \sqrt{\gamma\frac{P}{\rho}}\right)$:

$$\begin{aligned} \frac{c_0^2}{c^2} &= \frac{\gamma}{\gamma}\frac{P_0/\rho_0}{P/\rho} && \text{with Eq. 4.52} \\ \frac{c_0^2}{c^2} &= \left(\frac{\rho_0}{\rho}\right)^{\gamma-1} \end{aligned} \tag{4.55}$$

For the integral over the energy equation it is convenient to use the adiabatic sound speed c and the

Mach number $M = u/c$:

$$\begin{aligned}
\int \left(\frac{u^2}{2} + \frac{\gamma}{\gamma - 1} \frac{P}{\rho} \right) \mathrm{d}r = 0 \qquad & \text{with } M = \frac{u}{c} \\
\int M^2 c^2 \frac{\gamma - 1 + 2/M^2}{\gamma - 1} \mathrm{d}r = 0 \qquad & \text{with } M_0 = 1 \\
M^2 c^2 \frac{\gamma - 1 + 2/M^2}{\gamma - 1} &= c_0^2 \frac{\gamma + 1}{\gamma - 1} \\
M^2 \frac{\gamma - 1 + 2/M^2}{\gamma + 1} &= \frac{c_0^2}{c^2} \quad . \qquad (4.56)
\end{aligned}$$

We now combine the relations found from the integrals over the conservation equations (Eq. 4.54 to 4.56) to recover the Chevalier and Clegg (1985) solution at large radii (Eq. 4.48). We start by combining Eq. 4.55 and Eq. 4.56:

$$\begin{aligned}
M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{\gamma + 1} \right)^{1/(\gamma-1)} &= \frac{\rho_0}{\rho} && \text{with Eq. 4.53} \\
M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{\gamma + 1} \right)^{1/(\gamma-1)} &= \left(\frac{r}{R} \right)^2 \frac{u}{u_0} && \text{with Eq. 4.54} \\
M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{\gamma + 1} \right)^{1/(\gamma-1)} &= \left(\frac{r}{R} \right)^2 \sqrt{\frac{\gamma + 1}{\gamma - 1} - \frac{2}{\gamma - 1} \frac{c^2}{c_0^2}} && \text{with Eq. 4.56} \\
M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{\gamma + 1} \right)^{1/(\gamma-1)} &= \left(\frac{r}{R} \right)^2 \sqrt{\frac{\gamma + 1}{\gamma - 1} - \frac{2}{\gamma - 1} \frac{\gamma + 1}{M^2 \left(\gamma - 1 + 2/M^2 \right)}} \\
M^{2/(\gamma-1)} \left(\frac{\gamma - 1 + 2/M^2}{\gamma + 1} \right)^{(1+\gamma)/2(\gamma-1)} &= \left(\frac{r}{R} \right)^2 \quad . \qquad (4.48)
\end{aligned}$$

The solution for the radius dependent Mach number outside the feedback region (Eq. 4.48) is shown in Fig. 4.14 together with a subset of our spherically symmetric simulations of stellar winds. In our simulations the sonic point is outside the feedback region and the density of the surrounding medium plays an important role. Combining Eq. 4.53, 4.54 and 4.55 leads to the relation between radius and density:

$$\frac{r}{R} = \frac{1}{\sqrt[4]{\left(\frac{\rho}{\rho_0} \right)^2 \frac{\gamma+1}{\gamma-1} - \frac{2}{\gamma-1} \left(\frac{\rho}{\rho_0} \right)^{\gamma+1}}} \quad . \qquad (4.57)$$

In Fig. 4.16 this expected density distribution for the free wind zone is compared to the inner zones of our simulations with the rotating 60 $M_\odot$ and 40 $M_\odot$ stellar models (details on the implementation of the time dependent wind can be found in Sect. 2.7). The free wind zones of the models seem to follow this trend, but further away from the feedback region, the non-negligible density of the ambient medium leads to a solution which is better described by the Castor et al. (1975) models. With Eq. 4.52 the radius-density relation Eq. 4.57 can be converted to a radius-pressure relation:

$$\frac{r}{R} = \frac{1}{\sqrt[4]{\left(\frac{p}{p_0} \right)^{2/\gamma} \frac{\gamma+1}{\gamma-1} - \frac{2}{\gamma-1} \left(\frac{p}{p_0} \right)^{1+1/\gamma}}} \quad . \qquad (4.58)$$

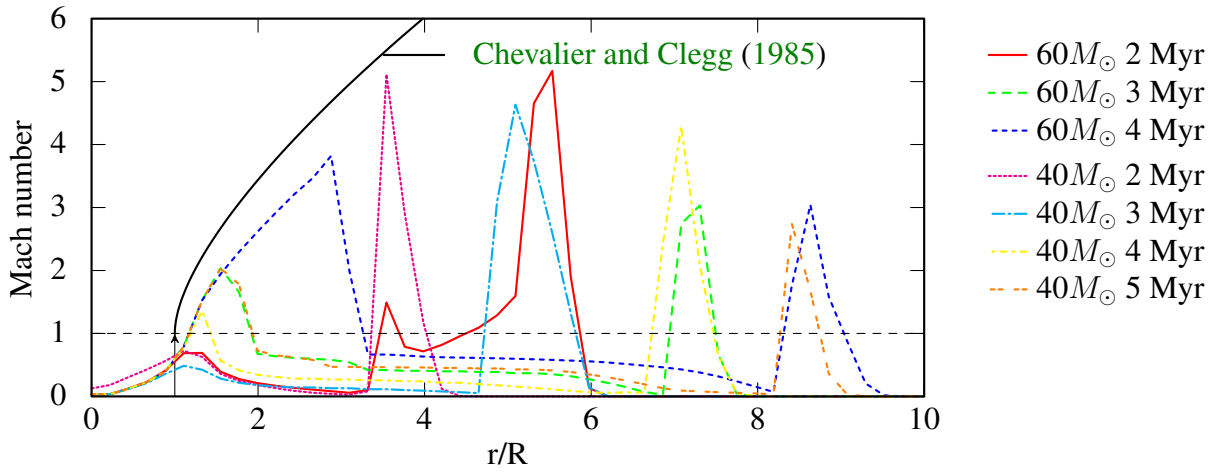


Figure 4.14: Mach number of the Chevalier and Clegg (1985) steady state wind model (black solid line) compared to our simulations. Our models reach Mach number 1 (dashed line to guide the eye) outside the feedback region (radius 0 to R). Generally the Mach number in our models is lower than predicted by the Chevalier and Clegg (1985) models. This is caused by (1) the driven wave into the ISM, (2) cooling, (3) varying wind speeds and mass loss rates.

This relation is shown in Fig. 4.17. Again the free wind regions of the simulations can be described with this model, but beyond the reverse shock Chevalier and Clegg (1985) has to fail. Finally Eq. 4.53 and 4.55 can be combined to $\frac{c^2}{c_0^2} = \left(\frac{u_0}{u}\frac{R^2}{r^2}\right)^{\gamma-1}$ which in turn combined with Eq. 4.54 leads to the velocity-radius relation:

$$\begin{aligned} \frac{u^2}{u_0^2} &= \frac{\gamma+1}{\gamma-1} - \frac{2}{\gamma-1}\left(\frac{u_0}{u}\frac{R^2}{r^2}\right)^{\gamma-1} \\ \frac{r}{R} &= \left[\frac{\gamma+1}{2}\left(\frac{u}{u_0}\right)^{\gamma-1} - \frac{\gamma-1}{2}\left(\frac{u}{u_0}\right)^{\gamma+1}\right]^{-1/(2(\gamma-1))} \end{aligned} \tag{4.59}$$

Fig. 4.18 shows that the velocity structure found near the feedback region in our simulations does not follow this model. Since the velocities found in the simulation were normalized by the velocity found at $R = 1$, which is lower than expected (i.e. the sonic point is found further outside than expected), the simulated velocities seem to be higher than the model. Actually, this is only due to the deviations at the point used for the normalization. The summary plot Fig. 4.15 compares Eq. 4.57, 4.58 and 4.59 to a simulations with a rotating 60 $M_\odot$ stellar model in a dense medium. Again the velocities show the problem that the velocity at the border of the feedback region is lower than expected.

4.5 Snowplow phases

The Sedov-Taylor phase ($r \propto t^{2/5}$, $v \propto t^{-3/5}$, Eq. 4.20 and 4.21) ends when the cooling time becomes comparable to the dynamical time. In the subsequent radiative phase a dense shell forms and the expansion is driven by pdV work in this so-called pressure-driven snowplow phase ($r \propto t^{2/7}$, $v \propto t^{-5/7}$, Eq. 4.61 and 4.62). In this phase, the pressure in the dense shell is the same as

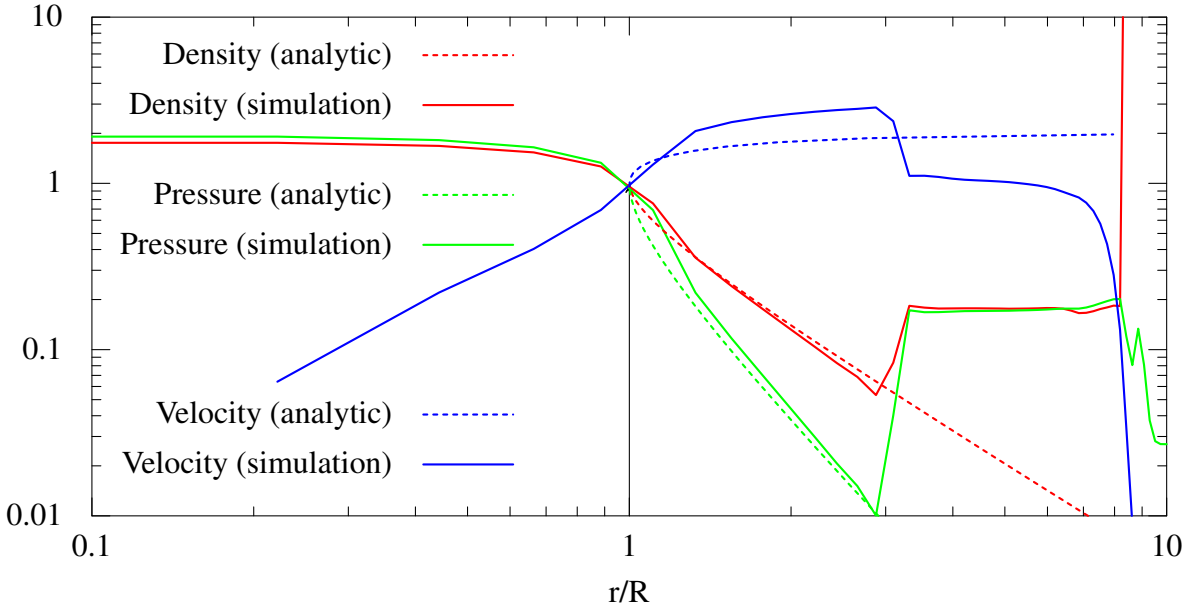


Figure 4.15: This plot shows normalized density, velocity and pressure profiles for a 60 $M_\odot$ star immersed in an ISM with a density of 100 particles cm^{-3} after 4 Myr. For the normalization the values of the outermost cell in the feedback region were used. The solution close to the feedback region is indeed following the trends in Chevalier and Clegg (1985, fig 1) but the free streaming region is driving a wave into the ISM and thus further out the solution shows the behavior of the 4 zone model of Castor et al. (1975).

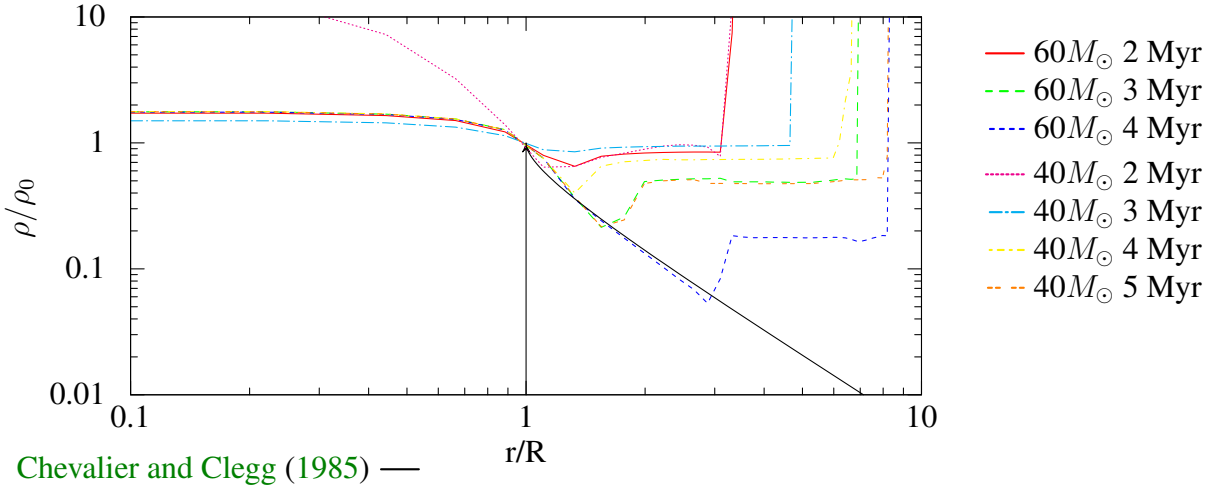


Figure 4.16: The Chevalier and Clegg (1985) density function ($\frac{\rho}{\rho_0} \sim c_2 \left(\frac{r}{R}\right)^{-2}$) compared to simulations. The simulations were normalized by the density value at the edge of the feedback region.

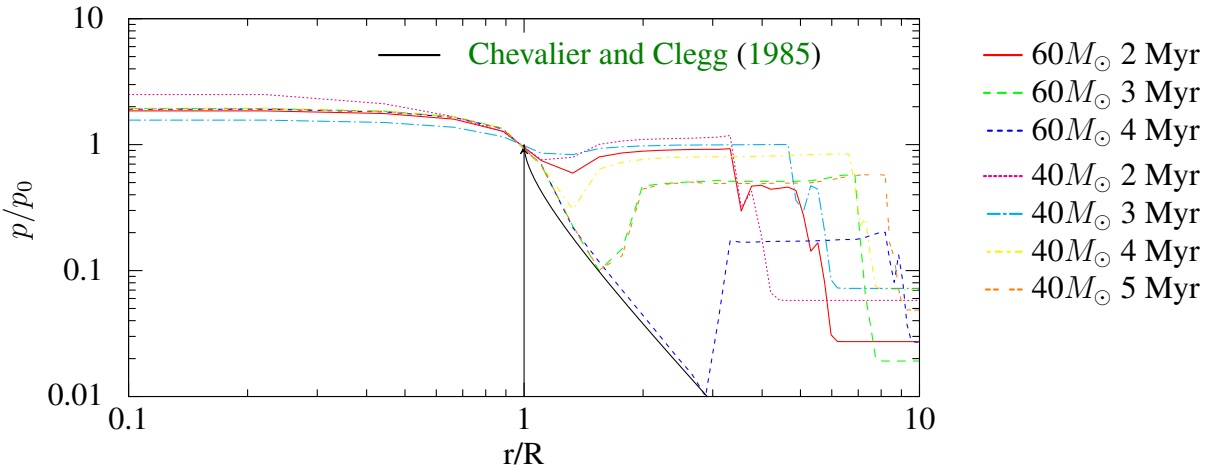


Figure 4.17: The Chevalier and Clegg (1985) pressure function ($\frac{p}{p_0} \sim c_1 \left(\frac{r}{R}\right)^{-10/3}$) compared to simulations. The pressure at $R = 1$ was used for the normalization.

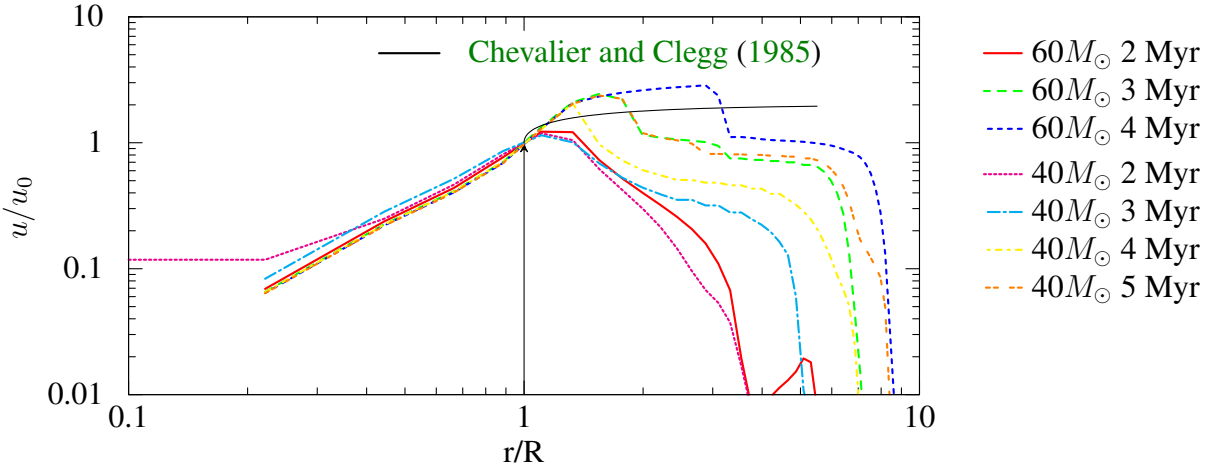


Figure 4.18: The Chevalier and Clegg (1985) velocity function ($\frac{u}{u_0} \rightarrow 2$) compared to simulations. Fig. 4.14 shows that the simulation is less hypersonic than expected. The velocities in the simulations seem to be too high, since they were normalized by the (too low) velocity at $R = 1$. However, the velocities in all simulations seem to approach the same asymptotic limit in the free wind zone.

in the shocked zone. When the pressure in the cavity has decreased enough, the remnant enters the momentum conserving phase ($r \propto t^{1/4}$, $v \propto t^{-3/4}$, Eq. 4.65 and 4.66) in which the shell's momentum leads to further expansion of the bubble. We will briefly show, how these power laws can be derived.

4.5.1 Adiabatic pressure driven snowplow

In this phase the pressure inside the bubble pushes the shell into the ambient medium. Near the contact discontinuity a density peak forms. Behind the shock, at the outer side of the bubble's shell, a layer of heated, swept up medium at 4-times the ambient density develops. (The maximal compression of an adiabatic mono-atomic gas leads to a factor 4 in density.) Despite radiative cooling losses the pressure in the shell gets much larger than the bubble pressure. Material starts to flow into the cavity and the bubble shell's density profile becomes symmetric. The largest cooling losses arise at the CD on the interface between the dilute bubble material and the swept up ambient medium.
During phases in which the pressure of the adiabatic expansion of the hot dilute (and therefore not cooling) interior of the bubble pushes the shell (c.f. Ostriker and McKee, 1988; McKee and Ostriker, 1977), the change of momentum (here written with the ν-dimensional sphere from Eq. 4.36 to 4.37)

$$\rho V_\nu \frac{\mathrm{d}\left(r\left(t\right)\right)^\nu \frac{\mathrm{d}r(t)}{\mathrm{d}t}}{\mathrm{d}t} = \underbrace{S_{\nu-1}\left(r\left(t\right)\right)^{\nu-1}}_{\text{bubble surface}} p_{\text{bubble}} \tag{4.38}$$

can be combined with the law of adiabatic expansion

$$\frac{p_{\text{bubble}}\left(t\right)}{p_{\text{bubble}}\left(0\right)} = \left(\frac{r\left(t\right)}{r\left(0\right)}\right)^{-\nu\gamma} \quad . \tag{4.60}$$

This way the exponents of r become

$$\begin{aligned} \nu a + (a-1) - 1 &= (\nu - 1 - \nu\gamma)a \\ a &= 2/(2+\nu\gamma) \quad . \end{aligned}$$

For 3D ($\nu = 3$) and an adiabatic exponent of $\gamma = \frac{5}{3}$ we find $a = \frac{2}{7}$. Thus dimensional analysis leads to

$$r\left(t\right) = ct^{2/(2+\nu\gamma)} \tag{4.61}$$

(c.f. Eq. 12 of McKee and Ostriker (1977) for the pressure-driven phase: $r\left(t\right) = 10^{-0.32}\sqrt[7]{\frac{R_c^2 E_{\text{SN}}}{n_0}}t^{2/7}$), which in turn leads to a velocity of

$$\frac{\mathrm{d}r\left(t\right)}{\mathrm{d}t} = \frac{2c}{2+\nu\gamma}t^{-\nu\gamma/(2+\nu\gamma)} \tag{4.62}$$

and a kinetic energy of

$$E_{\text{kin}} = \frac{mv^2}{2} = 0.5\rho V_\nu r^\nu v^2 \propto t^{2\nu(1-\gamma)/(2+\nu\gamma)} \tag{4.63}$$

As explained e.g. in Bandiera and Petruk (2004), Eq. 4.42 describes the fully radiative case whereas Eq. 4.61 can be used in the adiabatic case where no kinetic energy of the incoming flow is radiated in the outer shock.

4.5.2 Momentum conserving snowplow

When the pressure inside the bubble has decreased to the ambient pressure, momentum conservation governs the further expansion of the bubble. Assuming that all ambient medium is swept up in a thin, dense shell (thin shell approximation), this shell is at radius $r\left(t\right)$ moving with a velocity of $\frac{\mathrm{d}r(t)}{\mathrm{d}t}$ at time t. Momentum conservation

$$\rho V_\nu \frac{\mathrm{d}\left(r\left(t\right)\right)^\nu \frac{\mathrm{d}r(t)}{\mathrm{d}t}}{\mathrm{d}t} = 0 \tag{4.64}$$

leads to a radius of

$$r\left(t\right) = b\sqrt[\nu+1]{a + \left(\nu+1\right)t} \tag{4.65}$$

and a velocity of

$$\frac{\mathrm{d}r\left(t\right)}{\mathrm{d}t} = b\left(a + \left(\nu+1\right)t\right)^{-\nu/(\nu+1)} \tag{4.66}$$

which leads to a kinetic energy of

$$\begin{aligned} E_{\mathrm{kin}} &= \frac{mv^2}{2} = c\left(a + \left(\nu+1\right)t\right)^{-\nu/(\nu+1)} \\ &\text{with} \qquad c = \frac{\rho V_\nu b^{\nu+2}}{2} \end{aligned} \tag{4.67}$$

where a, b and c are constants.

Chapter 5

Method: codes and code modifications

Our numerical simulations were carried out with well tested, publicly available astrophysical Eulerian hydrodynamics codes. Namely PLUTO (Mignone et al., 2007), RAMSES (Teyssier, 2002) and ATHENA (Stone et al., 2008, 2010).

Our main modifications of the codes are time dependent stellar feedback, a minimal density to numerically stabilize the very dilute hot zones inside the bubbles, a cooling-heating prescription as described in Ntormousi et al. (2011) which allows for a multi-phase ISM and a threshold density below which radiative cooling is not taken into account. The latter can be used to stabilize cells near the CD and will be discussed in Sect. 6. Moreover we added a passive scalar to follow the spread of the radioactive trace element ^{26}Al in our simulations.

We will start by introducing the codes (Sect. 5.1). After this, in Sect. 5.2, we will then focus on the implementation of the feedback, we discussed in Sect. 2.7.

5.1 Hydrodynamic codes

Important considerations for the code choice were (in this order) the available Riemann solvers (Sect. 3.5.1), the implemented grids and physics modules and the available knowledge in the CAST group[1]. We decided to use different codes for different aspects of the problem. E.g. the spherical mesh in PLUTO made this code the best choice for 1D simulation, whereas following the trace element ^{26}Al was easier to implement with RAMSES. Finally the impact of radiation transfer was tested with ATHENA in the scope of the ISIMA summer school, since the GPU radiation transfer module of RAMSES was not yet publicly available at this time.

5.1.1 The PLUTO code: spherical symmetry

PLUTO (Mignone et al., 2007, 2012) is a modularized mesh code for astrophysical magnetohydrodynamics, developed at the Dipartimento di Fisica, Torino University in a joint collaboration with INAF, Osservatorio Astronomico di Torino and the SCAI Department of CINECA. The code web page is http://plutocode.ph.unito.it/. Although PLUTO is a freely-distributed software there is no publicly accessible code repository.

[1] http://www.usm.uni-muenchen.de/CAST/

For this work we used version 4.0 of PLUTO. It is MPI parallel and includes Cartesian, cylindrical or spherical meshes in 1, 2 or 3 dimensions. While the static grid version if entirely written in C, adaptive mesh refinement (AMR) requires the Chombo library and needs C++ and FORTRAN in addition to C. We used it for classical hydrodynamics (HD) with thermal condition and optically thin cooling. Our standard choices were RK3 explicit time-marching algorithm, MinMod piecewise interpolation scheme and the HLLC Riemann solver. PLUTO also includes the Two-Shocks, Roe, HLLD, HLL and Lax-Friedrichs Riemann solvers.

We decided to use this code for our 1D spherically symmetric models. The most important reasons for this choice were that PLUTO provides this desired mesh, thermal conduction and the HLLC Riemann solver. The latter is important, since our models require an accurate treatment of the contact discontinuity in the stellar wind bubbles. The PLUTO expertise in the CAST group (members of the CAST group published Ballone et al., 2013; Schartmann et al., 2012; Burkert et al., 2012; Schartmann et al., 2011; Junk et al., 2010; Schartmann et al., 2010, 2009, using this code) is also one of the pros for using the PLUTO code. To adapt PLUTO to our scientific problem, we had to modify the cooling-heating routine to allow for a multi-phase ISM and to add a source term for our time dependent stellar feedback.

5.1.2 The RAMSES code: radioactive tracers

RAMSES (Teyssier, 2002) is an astrophysical magnetohydrodynamics mesh code that was originally developed in Saclay to study large scale structure and galaxy formation. It is free software for non-commercial use only and can be downloaded from its bitbucket web-page: https://bitbucket.org/rteyssie/ramses. RAMSES is written in Fortran90, uses MPI and provides tree-based adaptive mesh refinement. The hydrodynamics module comes with five choices for the Riemann solver: exact, acoustic, LLF, HLL and HLLC. The TVD slope limiters MinMod and MonCen are implemented. This work uses version 3.10[2], which includes a Cartesian grid in 1, 2 or 3 dimensions. The physics modules include gravity, a cooling-heating module (discussed also in Sect. 2.2.6), star formation and supernova blasts.

In this work RAMSES is used for all simulations using a Cartesian grid. The most important reasons for this are the large choice of Riemann solvers, the user-friendly implementation of AMR, the simple implementation of additional passive scalars (which we need to follow our radioactive trace elements) and source terms (i.e. our stellar winds) and last but not least the RAMSES cooling module patch of Eva Ntormousi (Ntormousi et al., 2011) to allow for a multi-phase ISM.
Disadvantages – and thus reasons to resort to PLUTO or ATHENA – were that spherical symmetry was not implemented in version 3.10 and that ionization (on GPUs) was still in development in this version.
Expertise of the CAST group with this code is documented by papers and theses (Moeckel and Burkert, 2014; Ntormousi et al., 2011; Behrendt, 2011). Hints on passive scalars by R. Teyssier during his lectures at the Evora Supercomputing school[3] are gratefully acknowledged.

[2] More specifically we used the ramses.tar.gz from July 12th 2011 for our patches – also the git version at bitbucket still calls itself 3.10 although it has major differences e.g. the aton package. Ionization tests were carried out with the ramses.tar.gz version from December 11th 2011 which is close to the GPU branch of the bitbucket site, which identifies (still) as version 3.07

[3] http://www.lca.uevora.pt/supercomputing2009/

5.1.3 The ATHENA code: the effect of ionization

ATHENA (Gardiner and Stone, 2005, 2008; Stone et al., 2008) is a mesh code for astrophysical magnetohydrodynamics. It is parallelized with MPI. In contrast to PLUTO and RAMSES, ATHENA only comes with static (fixed) mesh refinement. The available mesh geometries are Cartesian or cylindrical.
One advantage of ATHENA is that it was developed for studies of the interstellar medium. Thus, many groups in the community use it and develop customized versions with additional physics included. For example, our work on elephant trunks was carried out with the code version of Mark Krumholz, which treats ionizing radiation, in the scope of ISIMA 2010[4]. However, in this thesis we do not include our work on ionizing radiation. Nevertheless, our future work on massive stars might use ATHENA simulations. The standard version of ATHENA (v4.2) implements compressible hydrodynamics (and MHD) in 1D, 2D, and 3D, thermal conduction and optically-thin radiative cooling. As in RAMSES an arbitrary number of passive scalars can be advected with the flow. ATHENA can treat gravity.
Further advantages are the comprehensive documentation and test suite available on the code web page https://trac.princeton.edu/Athena/ as well as the large choice of Riemann solvers (force, two-shock, exact, HLLE, HLLC, Roe) for hydrodynamics. Up to 3rd order reconstruction (piecewise parabolic) is implemented.
An example for the expertise of the CAST group with this code is Moeckel and Burkert (2014).

5.2 Implementation of mass, momentum and energy feedback

The insertion of the time-dependent stellar feedback[5] in our simulations can be considered as a generalization of the Chevalier and Clegg (1985) steady-state wind model (Sect. 4.4.3): in a spherical region in the simulated volume time-dependent source terms are added to the energy conservation equation and the continuity equation. We call this zone the feedback region or "the driver region" since it is driving the bubble expansion. At each time-step the feedback model yields a mass loss rate and a kinetic energy loss rate. These values are multiplied with the time step length and divided by the volume of the spherical feedback region. The resulting densities are added homogeneously to the mass density and the internal energy density in the feedback region.
If the gas in a cell inside the feedback region has a nonzero velocity, the increase of the mass in this cell due to stellar mass loss will lead to an increase of the kinetic energy. We take this into account when we add the feedback energy. In most models, we added the remaining feedback energy as thermal energy. Basically, adding all feedback energy as kinetic energy or using the energy fractions of a Sedov-Taylor blast (Sect. 4.3) leads to the same result, however, on a Cartesian mesh, adding kinetic energy leads to more asymmetries than adding thermal energy.
In simulations with spherical symmetry (i.e. in our PLUTO models), the feedback region is placed in the center of the grid. If a Cartesian grid is used (i.e. in all RAMSES models), the radius of the feedback region is always resolved with at least three grid cells since smaller feedback regions produce spikes along the diagonals of the grid. This problem was also discussed in Brighenti and D'Ercole (1994). On the other hand too large feedback regions lead to oscillations inside

[4] http://isima.ucsc.edu

[5] The feedback is also called "the wind" since it is injecting energy and mass into the simulation over a longer time period than a supernova (SN) burst.

the feedback region resulting in spikes[6] along the grid axes. The kinetic energy increase is not influenced by the feedback region's size. If the feedback region is small enough to resolve a free streaming region, the temperature in this zone is lower than the temperature in the wind bubble. This does not change the bubble evolution but it leads to a higher kinetic energy fraction[7] and a slightly lower feedback energy efficiency.
On a Cartesian grid, spherical feedback regions are produced by weighting cells which are only partially inside the feedback region by the amount of the overlap of the cell with the feedback region. To achieve this consistently for all simulations independently of the number of CPUs used, a mask with weights is calculated for all AMR levels at the start of the simulation. This mask is only recalculated if the feedback region moves with respect to the grid. We use a Monte-Carlo method to find the weights: the code randomly generates positions in the cell and checks which fraction of them is situated inside the feedback region. The typical number of random points per cell was 100 corresponding to a 10% error in the volume fraction in these cells. This error leads to slight asymmetries of the feedback region, but does not introduce errors in the total amount of inserted mass or energy, since the energy and mass input are converted to densities using the actual volume of the feedback region, which differs from $\frac{4\pi}{3}r^3$ due to the Monte-Carlo errors. Since these volume fractions are only calculated at the start of the simulation and stored in a mask for stars not moving inside the computational box, the slightly asymmetric shape of the feedback region stays constant during the simulation.
Sect. 5.2.1 and 5.2.3 give details on our implementation of the stellar feedback in the different codes. In all of them we find the current mass loss rate and energy injection rate via a table look-up in the feedback models at the end of each time step of the code. The stellar mass and energy feedback during the last time step is then added homogeneously as a source term in a designated feedback region.

5.2.1 PLUTO code modifications

Code Listing B.15 shows a minimal implementation of a constant stellar wind. In this code snippet, we see two different methods to insert a constant wind with a terminal velocity of 10^8 cm s^{-1} and a mass loss rate of 3×10^{-5} $M_\odot$ per year. The preprocessor directives (#ifdef-directives) in lines 22-24 switch between no wind (neither THERMALWIND nor INFLOWING_WIND defined), kinetic or thermal energy input. According to these choices, lines 140-181 either insert the thermal energy of the wind inside the domain or use the wind's kinetic energy in an inflow boundary condition. In the rest of Code Listing B.15 we see the definition of the units (lines 53-55), the specific heat ratio (line 57) and the initial conditions (a homogeneous cloud, lines 59-63). In the latter the velocities are not shown, since they are also set to zero in the default template for `init.c`.

Time dependent stellar feedback

Code Listing B.16 shows the implementation of a time dependent stellar wind with a subsequent SN explosion in `init.c`. Again, we define feedback modes (lines 15-22), set the code units and global parameters (lines 54-62) as well as the initial conditions (lines 78-83) where we added

[6] Some authors call this phenomenon "artificial jets".

[7] The free streaming region is not removed from the efficiency plots. After 1 Myr the free streaming region of a 60 $M_\odot$ star in a $n \sim 100$ cm^{-3} contains $\sim 2\%$ of the kinetic energy. Its share of thermal energy is larger than 2% [no percentage calculated yet].

a tracer to monitor energy losses via radiative cooling. Lines 85-97 contain code for tests with viscosity and SNe with linear velocity profiles. These lines are not used in our "standard models". Lines 113-138 take care of reading in old models. The new computational volume is typically larger than the volume in the read-in simulation. Therefore all cells are initialized with our desired initial conditions and only the cells present in the old simulation are overwritten with the read-in data. In the boundary conditions routine (lines 155-390) we use the tabulated wind data. This routine calculates the feedback region volume (lines 194-214). The kinetic energy feedback (lines 215-236) is similar to the aforementioned constant wind. Lines 241-276 can merge cells, if the mean free path becomes larger than a grid cell length. This part of the code was only used for tests, not for production runs. In lines 278-372 we finally find the time dependent stellar feedback. The routine first checks, if a SN explosion is due (lines 278-280). If this is the case, it either adds the SN ejecta derived from the Ekström et al. (2012) final masses or a canonical value of 3 $M_\odot$ (lines 292-304). Which one of these two SN models is used, depends on the preprocessor directive GENEVA. Lines 311-315 reduce the time step length shortly before the SN. If the SN is not due yet, and the Geneva stellar evolution models (Sect. 2.7.1, data from Ekström et al., 2012) are used, the code interpolates in the table (line 319). In lines 324-329 the code evaluates the cavity size and adds a SN explosion if a pre-defined bubble size is reached before the time when the SN would be due. This was used for consistency checks with Tenorio-Tagle et al. (1990) who use a constant wind and add the SN explosion when a given cavity size is reached. Lines 346-355 add the time dependent wind and lines 355-367 add a constant wind.
The interpolation routine is shown in lines 391-472. Basically, we read in a table when this interpolation routine is called for the first time (lines 409-430). Then we do a binary search in the table (lines 439-469) and use the time step length, the code units and the desired feedback region volume to get mass and energy densities (lines 469-470).

Calls to the feedback routines

The code in `boundary.c` (Code Listing B.1) checks if there is still a massive star that has not exploded and calls `UserDefBoundary` from Code Listing B.16 if it has not been done yet for this time step. This check is necessary since the predictor corrector scheme would add the stellar feedback several times (twice for RK2 and three times for RK3) otherwise.
`pluto.h` (Code Listing B.8) now also contains a global variable for the time of the SN explosion and the SN routine and the wind table look-up are listed in `prototypes.h` (Code Listing B.9).

Radiative cooling in a multi-phase ISM

The RAMSES cooling-heating module of Ntormousi et al. (2011), which is discussed in Sect. 5.2.3, has been ported and merged with the PLUTO cooling table. This is shown in Code Listing B.10 of `radiat.c`. However, from Code Listing B.3 it can be seen that we do not use radiative cooling inside the feedback region and that we have moved the minimal temperature check to Code Listing B.10. Moreover we created an artificial equilibrium for the 1 000 K models in the cooling table Code Listing B.2.

Other patches

The patches in Code Listing B.6 (`input_data.c`) help us restarting the simulation: Our strategy is to start with a small box, but before the shock can reach the boundary, we restart the simulation

and add more cells of unperturbed medium. This increase of the simulation volume is carried out when a minimal number of unperturbed cells (i.e. cells with zero velocity, initial density and initial pressure) is reached. This is not visible in the source code since this process is controlled with a shell script (Code Listing B.20). The minor patch in Code Listing B.11 takes care that after a restart with a larger volume the output files are still numbered consecutively. Basically, this only makes post-processing of the data easier. Code Listing B.12 only got some additional debugging output. Our modifications in Code Listing B.13 avoids outflows from already empty cells. Code Listing B.7 shows our modification of the minimal pressure and minimal density. This patch was needed due to the very strong gradients in our models. Code Listing B.14 shows, at which place the thermal conduction coefficient can be modified. It also shows that we do not use thermal conduction in the feedback region. Code Listing B.4 indicates at which place the viscosity can be changed. In Code Listing B.5 we added new units and limits.
Typical code settings for our models are shown in Code Listing B.18. An example of pluto.ini containing run-time parameters can be found in Code Listing B.17. Code Listing B.19 shows an example for a post processing routine.

5.2.2 Code tests

We have tested our implementation of the stellar feedback by comparing to the analytic models (Sect. 4.3, 4.4.1 and 4.4.3), by checking the total energy content in simulations without radiative cooling and finally by comparing to published simulations (e.g. Thornton et al., 1998; Tenorio-Tagle et al., 1990). None of these tests showed problems in our implementation [TO DO: add plots or describe test results].

5.2.3 RAMSES code modifications

Code snippets of our RAMSES patches are found in the appendix. We will briefly discuss the new modules and the modifications of existing modules.

Stellar model database

The module `geneva_models` (Code Listing C.2) contains the tabulated mass loss and ^{26}Al data of Ekström et al. (2012) as well as feedback energies computed as explained in Sect. 2.7.3 and SN data as described in Sect. 2.7.4. Moreover, it provides a routine to convert all feedback data to the units used in the simulation, an output routine and a routine for linear interpolation in the feedback tables which can also add up feedback for several stars.

Feedback region and mask

However, the subroutine `read_driver` (lines 56-213) in the module `driver` (Code Listing C.1) can also read in stellar feedback from an ASCII table and convert it to code units. This is used for example for the Voss et al. (2009) population synthesis models. The subroutine `read_sn` (lines 214-291) in this module reads tabulated SN data. The allocated driver arrays can be deallocated with the subroutines `remove_driver` (lines 292-309) and `remove_sn` (lines 310-324). This module also comes with a routine for linear interpolation of the read-in tables (`interpolate_driver`, lines 325-376). The subroutine `add_SN` (lines 377-403) searches for SN explosions occurring during the present time-step and returns the mass and energy feedback.

We now need to find the region into which the stellar feedback will be injected. This will be done with a mask: We define an array that tells us, how much of the cell's volume lies inside the feedback region: 0.0 for cells fully outside the feedback region, 1.0 for cells fully contained in the region and a number between 0.0 and 1.0 for cells partly inside. For the latter case we use the fraction of randomly generated positions in the cell that lie inside the feedback region. For 1D or 2D we also have a subroutine which calculates the volume fractions analytically.
To flag the feedback region, the subroutine `allocate_driver_mask` (Code Listing C.1, lines 404-588) uses the FORTRAN derived data type `driver_mask` (lines 38-51). This object contains the number of cells along the feedback region radius (plus one), the cell size, the actual volume of the feedback region found via Monte-Carlo which slightly differs from $\frac{4\pi r^3}{3}$ and an n-dimensional mask for a volume containing the feedback region plus approximately one cell in each direction. The size of the boundary layer is not exactly one cell in each direction, since the center of the feedback region does not have to be aligned with the grid. Of course, placing the feedback region center asymmetrically on the grid does not sound like a wise choice for a star in a homogeneous cloud. However, the idea behind this implementation is that at some point our simulations will contain multiple stars, represented by the feedback regions, which will have proper motions.
The routine now allocates the array `driver1` of such objects with as many entries as grid levels and fills it with data. The module contains two functions that read data from this object: `get_driver_volume` (lines 589-598) can read the actual volume and `get_driver_mask` (lines 599-655) can look up how much a given cell overlaps with the feedback region. Since RAMSES always loops over the grids by vector sweeps, the subroutine `driver_weights_fixed` (lines 671-726) does this look up for a whole array of `dimension(1:ngrid)`. Before RAMSES exits, `driver1` has to be deallocated. This is done by the subroutine `deallocate_driver_mask` (lines 656-670). For moving feedback regions, it might be advantageous to find cells belonging to the feedback region on the fly. This can be done with subroutine `driver_weights` (lines 727-857).
If the stellar feedback is inserted as kinetic energy, the subroutine `driver_vector` (lines 858-981) finds radial vectors. Finally subroutine `print_xyz` (lines 982-1061) helps to find out in which cell the code encounters a problem. The module also contains a routine with analytic weights for 2D simulations.

Calls to the feedback routines

These two new modules now have to be called by RAMSES. The feedback parameters are stored in `amr_parameters.f90` (Code Listing C.3). These parameters can be set in the namelist and are read-in by `read_params.f90` and `read_hydro_params.f90` (Code Listing C.4 and C.5). The mass and energy injection of the star(s) are taken into account if the run time parameter `nstars` in the namelist (an example is shown in Code Listing C.27) is larger than zero. In this case, the code will add the newly emitted mass (total mass and radioactive tracers) and the internal energy (unresolved kinetic wind energy, radiation pressure) to the density resp. energy in the feedback region. The size and location of this feedback region are set using the run time parameters `r_driver`, `x_driver`, `y_driver` and `z_driver` in the namelist. The feedback data is loaded in `init_time.f90` (Code Listing C.6). If the run-time parameter `ifgeneva` is set to `.true.` in the namelist, the model grid, the stellar masses and the star formation times in the run-time parameters `genevayear, mstars` and `tstars` are used. Otherwise the code searches for tabulated feedback data. The data file names are stored in `file_driver` (default: wind.dat) and `file_sn` (default: sn.dat). After every time step, `courant_fine.f90` (Code Listing C.7) calls the feedback inter-

polation routine. It also uses the weights of the mask to identify feedback regions. Moreover, it takes care of the decay of ^{26}Al and ^{60}Fe. If the preprocessor directive CARINA is used, the feedback subroutine `wind` uses sequential star formation. The preprocessor directive EKIN switches between compiling the code for kinetic energy feedback or code where the feedback is inserted mainly as thermal energy. The preprocessor directive TMIN ensures that the total energy is always larger than the kinetic energy. Further preprocessor directives are TMAX, which sets a maximal temperature (used for tests only), DECAYINTERVAL and KAHANBABUSKA that avoid problems with number precision in the tracer decay and in large sums, and THII which sets $T = 10.000$ Kelvin in the feedback region. Finally the feedback arrays are deallocated by `clean_stop` in `update_time.f90` (Code Listing C.8).
To control the adaptive mesh refinement in the feedback region we patched `flag_utils.f90` and `hydro_flag.f90` (Code Listing C.9 and C.10). This is advantageous since too low refinement leads to x-shaped outflows whereas very high resolution leads to bouncing waves inside the feedback region which can be computationally costly.

Radiative cooling in a multi-phase ISM

The standard treatment of cooling and heating processes in RAMSES is discussed in Sect. 2.2.6. For our simulations we used two modified versions of `cooling_module.f90`. One version is shown in Code Listing C.19, the other one has been described by Ntormousi et al. (2011). The latter contains cooling tables generated with the `CLOUDY` code. In the version shown in Code Listing C.19, the preprocessor directive `artificial_ISM` (lines 416-447) establishes a warm phase by using a density dependent temperature floor. We need this, since we want two stable thermal phases in our simulations: a cold cloud and a warm, dilute ISM. Cells, which are undisturbed by stellar feedback should neither cool nor heat. For our production runs, we used the version of Ntormousi et al. (2011), but tests with the artificially stable ISM showed that the dense bubble shell will always loose almost all its thermal energy and cool to the equilibrium temperature (i.e. the minimum temperature in the artificially stable ISM). Hence the feedback energy efficiency of both cooling modules was of similar order. It turned out that mixing across the contact discontinuity has a very strong influence on the feedback energy efficiency.
We patched `cooling_fine.f90` (Code Listing C.18) to switch off cooling in the feedback region. Technically, this is implemented with a mask. All cells inside the feedback region plus a layer with a width set by `coolplus` in the driver parameters in the namelist do not suffer radiative cooling losses.
Since our simulations focus on the feedback energy efficiency, it is important for us to be able to monitor the energy losses via radiative cooling. We thus store the loss during the last time step for every cell. Since we do not want this array to be advected like a passive scalar, we store it at `nvar+1`. To do this, we increase the array size in `init_hydro.f90` (Code Listing C.14). To avoid loosing the data during memory defragmentation, we also patched `load_balance.f90` (Code Listing C.15). The losses per time step are analyzed in `output_hydro.f90` (Code Listing C.16) and re-set in `amr_step.f90` (Code Listing C.17).

Initial conditions

Patches to `hydro_parameters.f90` (Code Listing C.11) and `init_flow_fine.f90` (Code Listing C.12) enabled us to set the initial distribution of radioactive tracers and to read in SPH data. For this purpose we also wrote the module `sph` (Code Listing C.13). The preprocessor directive

JIM uses the settings for the SPH data provided by Jim Dale (ISM structures in the simulations presented in Dale and Bonnell, 2011). If it is not defined, the data format of the SPH data provided by Clare Dobbs (molecular clouds in the simulations presented in Dobbs et al., 2011) is used.

Other patches

Additionally we have set default units in `amr_commons.f90` (Code Listing C.22). These defaults can be overwritten by different choices in the namelist. Since some of our simulations (namely the simulations with stellar feedback into the clouds of Clare Dobbs (Dobbs et al., 2011)) observe energy flowing out of the computational box, we monitored such energy losses with the new module in `outflow.f90` (Code Listing C.23). Finally we often re-started our simulations and thus we patched `init_amr.f90` (Code Listing C.20) since RAMSES does not allow to change the initially chosen output times, which is quite inconvenient for re-simulations when the initial simulation showed an interesting phase in the model's evolution that should be analyzed in more detail. Our version of `godunov_utils.f90` (Code Listing C.21) removes outflows from almost empty cells. It can also ignore almost empty cells when the CFL (sect 3.3) is evaluated. For the HLLC solver, the preprocessor directive ALUSTOP only allows the radioactive tracers to flow into cells with temperatures above a temperature threshold.
To stabilize our simulations we also avoid negative internal energies in `set_uold` (Code Listing C.24, `godunov_fine.f90`). We remove outflows from almost empty cells and reset the pressures in these cells in subroutine `godfine1` (Code Listing C.24, `godunov_fine.f90`) and subroutine `ctoprim` (Code Listing C.25, `umuscl.f90`). `godunov_fine.f90` can also be used to reset the cooling losses.
The Makefile in Code Listing C.26 summarizes the newly defined preprocessor directives.

5.2.4 Code tests: ^{26}Al feedback

The ^{26}Al feedback was implemented as passive scalar with a decay law (`courant_fine.f90`, Code Listing C.7). Since time steps can be a small fraction of the half life time of ^{26}Al, the decay law can also be invoked after a given amount of time instead of being used at every time step. This helps to ensure that the decay is not lost due to limited numerical precision. However, during most of the simulations the time steps were large enough that the decay could be calculated at every time-step for all cells. Figure 5.1 shows the convergence of the approaches in the simulation of a 60 $M_\odot$ star in a homogeneous medium of 100 particles cm^{-3} after 60 kyr. For all five runs, 8 processors and AMR with grid levels 5 to 7 (i.e. at least 2^5 cells but up to 2^7 cells along each axis) were used.

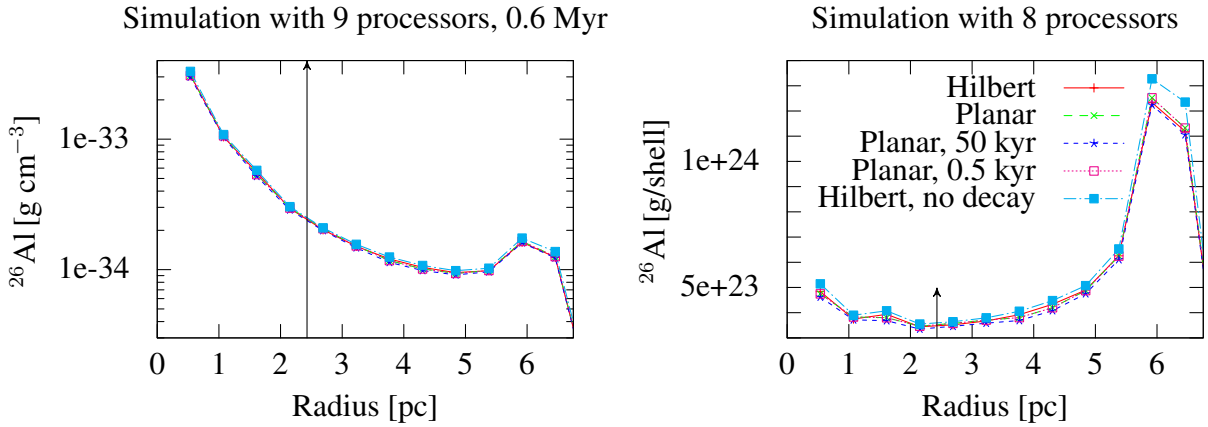


Figure 5.1: This figure shows the convergence of different decay routines. The back arrow indicates the location of the border of the feedback region. The turquoise line shows a simulation without decay. A simulation in which the decay of ^{26}Al is calculated every 50 kyr is shown in blue. Obviously this line has to overestimate the decay since it assumes that all the ^{26}Al in the cell has been there since the last calculation of the decay. The simulation with decay at every time step (green, typical time step $\sim$ 50 yr) coincides with the simulation with 500 yr between decay calculations. Number precision problems are expected if the time steps become of the order of years or smaller. It is interesting to note that different parallelization methods or a different number of processors influenced the result (green line and red line). It has to be tested if this is just a problem of the boundary cells if domain decomposition happens inside the driver region or if this problem always occurs if AMR is combined with MPI.

Simulations

The main questions addressed in the simulations described in this work are, how long massive star feedback takes to disrupt a Giant Molecular Cloud (GMC)(GMC) ($\rightarrow$ molecular cloud lifetimes), how much of the feedback energy can be converted to kinetic and thermal energy of the GMC gas ($\rightarrow$ feedback energy efficiency, energy reservoir for driving of turbulence) and which fraction of the cold GMC gas is heated ($\rightarrow$ mass distribution of ISM phases). Another important aspect of these simulations was to check whether a scaled Voss et al. (2009) feedback model, which is based on the mean of 100 coeval stars between 8 and 120 $M_{\odot}$, is a suitable model for the feedback of an OB association with ~ 10 massive stars. This is of importance for modeling the Orion-Eridanus Superbubble (OES), because the feedback of Voss et al. (2009) destroys homogeneous GMCs very efficiently and produces bubbles significantly larger than observed. To assess if the Voss et al. (2009) prescription is a realistic model for the feedback of a typical[8] OB association, we compared its influence onto GMCs (1) to the influence of the feedback of individual Monte-Carlo realizations of an OB association with 10 stars between 8 and 120 $M_{\odot}$ and star formation with a dispersion (σ) of 1 Myr (as described in Voss et al., 2010) and (2) to feedback of individual massive stars.
For all simulations in this work we use a cooling-heating prescription describing a cold neutral medium (CNM) and not molecular clouds ($T \sim$ 10–30 K and $n \sim 1000$ particles cm^{-3}). This prescription does not include ionization, cosmic ray heating, C^+, CO, C or H_2O cooling.

[8]We call an OB association "typical" if its stellar mass distribution has a high probability to be drawn from the assumed IMF.

Chapter 6

1D: Feedback efficiency in spherical symmetry

We start to investigate the amount of energy massive stars can convert to kinetic energy of the surrounding ISM with one-dimensional spherically symmetric models. The big advantage of 1D simulations is that they make it feasible to search a large region of parameter space in a short time. The obvious drawback is that non-radial motions (e.g. hydrodynamic instabilities in the bubble's shell) cannot be taken into account. We will thus assume that the retained energy in the 1D models is just an upper limit and re-simulate the most interesting models in more dimensions. The 1D simulations were carried out with the patched PLUTO code (Sect. 5.2.1) and contain a single, massive star with $60\ M_{\odot}$. As we have shown in Sect. 2.7.5, this is a valid first approximation for feedback in GMCs. The stellar feedback is calculated from the mass loss rate of the rotating models of Ekström et al. (2012) as described in Sect. 2.7.3. The implementation in the code is discussed in Sect. 5.2.1.
With the 1D models it is possible to study the feedback energy efficiency's dependence on resolution. Our simulations use a static mesh with up to 250 cells per parsec. We assume that the star is placed in an infinite, homogeneous cloud and start with a computational box of 5 pc. During the simulation, we monitor how many undisturbed cells of ambient medium are left and add another 5 pc of undisturbed medium to the computational box if the number of such cells drops below 100. The standard assumptions for the cloud material in this study are solar metallicity, a density of $\rho_0 = 2.2 \times 10^{-22}$ g cm^{-3} and a pressure of $p_0 = 1.48 \times 10^{-12}$ erg cm^{-3} corresponding to a temperature of approximately 37 K. This phase of the ISM is in cooling-heating equilibrium if we use the same cooling model as Ntormousi et al. (2011) (see Fig. 2.1 for the cooling-heating equilibrium). The cooling-heating equilibrium temperature $T_{\rm eq}(n)$ depends on the cooling model. We added an artificially stable gas phase for the initial conditions (n_0, T_0) if this ISM phase was not in cooling-heating equilibrium in the chosen cooling model. The number density (n) of ~ 100 cm^{-3} resembles the average density of molecular cloud complexes as shown in Sect. 2.5. It is known that molecular clouds exhibit a fractal structure, which will be addressed in our future work with models taking more dimensions into account.
Our work extends the published stellar feedback energy efficiency models in two important aspects:

1. In all simulations shown in this section, we follow the energy content of the simulations from star formation until several million years after the SN, when peak velocity in the bubble shell becomes smaller than the sound speed of the ambient medium. At this time the shell is not

infinitely thin and the highest velocity is found near the highest density. We argue that at latest at this point turbulent motions will lead to break-up of the shell and very efficient mixing (and energy deposition) in the ambient medium. Therefore, we follow the evolution of the models substantially longer than it was done in the work of Tenorio-Tagle et al. (Tenorio-Tagle et al., 1990, 1991; Tenorio-Tagle, 1996). Thornton et al. (1998) also stop the simulations after 13 time of maximal luminosity (t_0)s (defined in Sect. 6.1.1), which is in most models shortly after the transition to the momentum conserving phase (Sect. 4.5.2).

2. We test how stellar winds and variations of the wind strength affect the feedback energy efficiency.
   Observationally, the impact of the wind of the SN's progenitor star is illustrated for example by the shell of the progenitor star around SN 1987A reported by Wampler et al. (1990), the wind shell of a 25 $M_\odot$ star seen in the SN remnant G296.1–0.5 (Castro et al., 2011) or the stellar-wind envelope seen in SN 2006aj (Sonbas et al., 2008).
   However, in the literature on feedback energy efficiencies stellar winds are either ignored (e.g. Thornton et al., 1998) or assumed to be constant (e.g. Tenorio-Tagle et al., 1990, 1991; Tenorio-Tagle, 1996). In our simulations, it turned out that ignoring winds is problematic: Table 6.2 shows that the amount of mechanical luminosity[1] that can be converted to shell motions differs between models, which insert all energy in a blast (a SN) and models where stellar winds are energy sources over long periods of time. Similar effects were observed by: Tenorio-Tagle et al. (1990, 1991); Oey and Massey (1994); Oey (1996); Tenorio-Tagle (1996). The reason for the higher feedback energy efficiencies of continuous energy injection processes is that WR winds of the progenitor star create a bubble in the ISM. Blast waves of SN explosions in such cavities undergo an almost loss-less expansion until they hit the cavity walls. As a consequence, wind-blown bubbles delay the time of maximal luminosity (defined in Sect. 6.1.1) and increase the amount of retained energy, since such cavities can act as pressure reservoirs. When the blast hits the cavity walls, so-called catastrophic cooling in the dense shell of swept-up ambient medium sets in (Tenorio-Tagle et al., 1990; Smith and Rosen, 2003). This process (strong radiative cooling losses caused by a SN blast wave hitting a pre-existing shell) is a likely explanation for the X-ray emission in excess of an adiabatic model in X-ray bright superbubbles (Chu and Mac Low, 1990; Arthur and Henney, 1996; Oey, 1996).

In the first part of this chapter (Sect. 6.1) we will discuss wind-less reference models and proceed to time dependent winds in Sect. 6.2.

6.1 SNe without progenitor winds

The models discussed in this sub-section do not take the stellar winds of the SN's progenitor star into account. Hence at the time of the SN explosion the ambient ISM in these models is homogeneous without pre-existing stellar wind bubbles. One of the goals of this section is a consistency check of our setup with the published feedback energy efficiencies of Thornton et al. (1998) and Tenorio-Tagle et al. (1990).

[1] "mechanical luminosity" is the energy input inferred from the mass loss rate and estimated wind velocity.

6.1.1 Previous work

A very well studied case of a SN explosion in the literature is the deposition of $E_{\rm SN} = 10^{51}$ erg (also called 1 FOE) into a homogeneous ambient medium with a number density of $n_0 = 1$ cm^{-3}. The mass of the SN ejecta differs between the studies (e.g. Tenorio-Tagle et al. (1990) use 4 $M_\odot$ and Thornton et al. (1998) use 3 $M_\odot$) but has – as shown in Tab. 6.1 and Sect. 6.1.3 – only a minor influence on the feedback energy efficiency.
The study of Thornton et al. (1998) also covers ambient densities better matching to GMCs. To compare models with different ambient densities, they normalized the simulation times with the time, when the largest energy losses due to radiative cooling occur in the simulation. This time is called "time of maximal luminosity" (t_0). Please note that despite this name it does not correspond to the maximum in the SN light curve, which is caused by radioactive decays. Thornton et al. (1998) found a feedback energy efficiency of $\sim 10\%$ after 13 t_0 for a wide range of ISM number densities ($n = 0.001$ to 1 000 cm^{-3}) and metallicities ($\log(Z/Z_\odot) = -3.0$ to 0).

6.1.2 Grid of models

As a parameter study, a large number of simulations was run with ambient densities from 2.2×10^{-25} to 2.2×10^{-22} g cm^{-3} (see Tab. 6.1) in order to check the influence of the ambient pressure on the resulting feedback energy efficiency. The temperature of the ambient medium in the study of Thornton et al. (1998) is 1 000 K. Tab. 6.1 also contains models with $T_{\rm eq}(n_0) = 37$ K, since this is the cooling-heating equilibrium temperature for a density of 2.2×10^{-22} g cm^{-3} if the cooling prescription described in Ntormousi et al. (2011) (see Fig. 2.1) is used. A subset of these models (no stellar wind, ambient density 2.2×10^{-22} g cm^{-3}) is also shown in the uppermost part of Tab. 6.2.
Our reference model for this section is: $T_{\rm eq}(n_0) = 37$ K ambient medium temperature, $n_0 = 2.2 \times 10^{-22}$ g cm^{-3} ambient number density, cooling function as described in Ntormousi et al. (2011) (Fig. 2.1), 0.32 pc feedback region radius, Sedov-Taylor like energy ratios in the initial conditions (Sect. 4.3.1) and 11 $M_\odot$ mass loss during the SN explosion (Sect. 2.7.4). The influence of the parameters in the simulation was checked by varying just one of them at a time. Since this leads to a large grid of models, we only show a selection in Tab. 6.1. The models in this table differ in more than one parameter from each other.

Low order interpolation functions (i.e. linear interpolation) and the two-shock solver were used to avoid numerical issues at the sharp discontinuity between the hot bubble and the cold shell. Otherwise over-oscillations near the contact discontinuity would build up and cause negative pressures and spurious energy gains.

6.1.3 Findings and discussion

For SN explosions without prior stellar wind bubbles in a homogeneous ambient medium with 2.2×10^{-22} g cm^{-3}, solar metallicity and a temperature of 1 000 K Thornton et al. (1998) find a feedback energy efficiency of about 8% after 13 t_0 (times of maximal luminosity, defined in Sect. 6.1.1). At this time we find similar feedback energy efficiencies for this model and also for our reference model (Tab. 6.1). However, when the shell velocity has decreased to the sound speed of the ambient medium just 0.11% of the SN feedback energy are still retained in a model,

	ρ [g cm^{-3}]	t [kyr]	t/t_0	$E_{\rm kin}$(shell) [10^{50} erg]	$E_{\rm kin}$(total) [10^{50} erg]	r [pc]
Thornton et al. (1998),	2.2×10^{-25}	122	1	2.14	2.73	55.8
$T_{\rm eq}(n_0) = 1\,000$ K,		1 590	13	0.77	0.78	114.3
$\Delta x = 0.056$ pc,	2.2×10^{-24}	34.4	1	2.17	2.74	21.4
$r_{\rm f} = 1.5$ pc,		447	13	0.75	0.84	43.0
3 $M_\odot$	2.2×10^{-23}	9.73	1	2.33	2.67	8.2
		126	13	0.84	0.76	16.4
	2.2×10^{-22}	3.06	1	2.35	2.61	3.3
		39.8	13	0.76	0.80	6.6
$T_{\rm eq}(n_0) = 1\,000$ K,	2.2×10^{-25}	96.5	1	2.41	2.84	47.5
$\Delta x = 0.004$ pc,		1 245.5	13	0.82	0.82	106.2
$r_{\rm f} = 1.5$ pc,	2.2×10^{-24}	28.0	1	2.27	2.77	18.6
3 $M_\odot$		364.0	13	0.77	0.78	39.4
	2.2×10^{-23}	8.0	1	2.18	2.69	7.3
		104.0	13	0.72	0.74	15.1
	2.2×10^{-22}	2.5	1	2.84	3.23	3.1
		32.5	13	0.66	0.66	6.0
$T_{\rm eq}(n_0) = 1\,000$ K,	2.2×10^{-25}	100.5	1	2.13	2.68	49.4
$\Delta x = 0.004$ pc,		1 306.5	13	0.79	0.80	103.5
$r_{\rm f} = 0.3$ pc,	2.2×10^{-24}	30.0	1	2.19	2.68	19.1
11 $M_\odot$		390.0	13	0.71	0.73	38.3
	2.2×10^{-23}	9.0	1	2.32	2.81	7.5
		104.0		0.72	0.72	15.0
		117.0	13	0.66	0.66	15.6
	2.2×10^{-22}	3.0	1	3.00	3.03	3.0
		39.0	13	0.58	0.59	6.2
$T_{\rm eq}(n_0) = 37$ K	2.2×10^{-22}	3.0	1	2.61	2.96	3.0
$\Delta x = 0.004$ pc,		32.5		0.68	0.68	5.9
$r_{\rm f} = 0.3$ pc, 0 $M_\odot$		39.0	13	0.59	0.59	6.2
$T_{\rm eq}(n_0) = 37$ K	2.2×10^{-22}	3.0	1	2.60	2.97	3.0
$\Delta x = 0.008$ pc,		32.5		0.64	0.66	5.9
$r_{\rm f} = 0.3$ pc, 0 $M_\odot$		39.0	13	0.57	0.58	6.2
$T_{\rm eq}(n_0) = 37$ K	2.2×10^{-22}	3.0	1	2.52	2.95	3.0
$\Delta x = 0.016$ pc,		32.5		0.62	0.63	5.9
$r_{\rm f} = 0.3$ pc, 0 $M_\odot$		39.0	13	0.53	0.55	6.2
$T_{\rm eq}(n_0) = 37$ K	2.2×10^{-22}	2.5	1	2.59	2.89	2.8
$\Delta x = 0.032$ pc,		32.5	13	0.61	0.61	5.9
$r_{\rm f} = 0.3$ pc, 0 $M_\odot$		39.0		0.52	0.53	6.2

Table 6.1: Retained kinetic energy ($E_{\rm kin}$) of SNe in homogeneous media. For all models 10^{51} erg were inserted at $t = 0$. $E_{\rm kin}$ and the bubble radius (r) were evaluated at the time of maximal luminosity (t_0, defined in Sect. 6.1.1) and after 13 t_0, which is the end of the simulations in Thornton et al. (1998). The resolution (Δx) and the state of the ambient medium (T, ρ) are varied. Since the bubble pressure at t_0 is much higher than the ambient pressure, the efficiency of the 1 000 K model is comparable to the 37 K model. 37 K is the equilibrium temperature for a density of 2.2×10^{-22} g cm^{-3} in the cooling function described in Ntormousi et al. (2011). For the ambient medium in the 1 000 K model an artificially stable gas phase had to be created in the cooling table (Code Listing B.2). t_0 also depends on the size of the feedback region ($r_{\rm f}$) and on the kinetic to thermal energy ratio. Therefore three SN models with different mass loading ($M_\odot$) are shown.

Δx [pc]	SN [10^{51} erg]	wind [2.34×10^{51} erg]	thermal conduction	a	ϵ ($v_{\rm sh} = c_{\rm s}$) [10^{51} erg]	$\epsilon_{\rm k}$ (wind) [10^{51} erg]	$\epsilon_{\rm t}$ (wind) [10^{51} erg]
0.032	yes	no	no	0	0.0011	-	-
0.016	yes	no	no	0	0.0011	-	-
0.008	yes	no	no	0	0.0011	-	-
0.032	no	yes	no	0	0.0213	0.0884	0.4981
0.016	no	RW	no	0	0.0231	0.0896	0.4981
0.064	yes	yes	no	0	0.0265	0.1027	0.5422
0.032	yes	yes	no	0	0.0271	0.0884	0.4981
0.016	yes	RW	no	0	0.0304	0.0896	0.4981
0.016	yes	yes	no	0	0.0365	0.1136	0.6019
0.008	yes	yes	no	0	0.0475	0.1340	0.6859
0.004	yes	yes	no	0	0.0620	0.1598	0.7756
0.032	yes	yes	no	1	0.0710	0.1841	0.8286
0.016	yes	yes	no	1	0.0791	0.1947	0.8696
0.008	yes	yes	no	1	0.0904	0.2076	0.9113
0.032	yes	yes	yes	0	0.0244	0.0827	0.4549
0.016	yes	yes	yes	0	0.0302	0.1014	0.5570
0.032	yes	yes	extreme	0	0.0094	0.0329	0.1915
0.016	yes	yes	extreme	0	0.0098	0.0353	0.2211
0.032	yes	CW	no	0	0.0293	0.0932	0.2070

Table 6.2: Stellar feedback in an ambient medium with a density of 2.2×10^{-22} g cm^{-3} and a pressure of 1.47684×10^{-12} erg cm^{-3}. This ISM phase is in cooling-heating equilibrium at ~ 37 K. Δx is the cell size in the simulation. Despite the lower ambient temperatures the three uppermost models without winds are comparable to Thornton et al. (1998) (1000 K). For models with a SN explosion ("yes" in column 3), 10^{51} erg and 11 $M_\odot$ of ejecta were inserted after 4.859 Myr. For simulations with stellar winds ("yes" in column 4) the Ekström et al. (2012) model for a rotating 60 $M_\odot$ star and the wind velocities summarized in Voss et al. (2009) were used (Sect. 2.7.3). In total this stellar wind inserts 2.34×10^{51} erg. The constant wind model ("CW" in column 4) inserts the same total wind energy at a constant rate. To check the influence of the resolution on the energy-efficiency of the SN explosion, simulations with lower resolution were re-sampled directly before the SN (indicated as "RW" in column 4), since the efficiency during the wind phase also depends on the resolution. The slightly higher kinetic energy in the rescaled model at the end of the wind phase is due to smooth interpolation. ϵ lists the kinetic energy in 10^{51} erg when the densest cell is decelerated to the ambient sound speed. $\epsilon_{\rm k}$ and $\epsilon_{\rm t}$ list the retained kinetic and thermal energy at the end of the wind phase (in units of 10^{51} erg). "Extreme" thermal conduction mimics a very efficient diffusion process by increasing κ by 15 orders of magnitude. The parameter a describes a density threshold, below which radiative cooling is no longer taken into account. This decreases the energy losses due to mixing of gas across the CD. The density threshold a is normalized to the density of the ambient medium. The table shows that **higher** efficiencies are reached for higher resolutions, thus the higher maximal densities are outweighed by the smaller amount of mixing across the CD in the higher resolved simulations. Whereas in lower resolved simulations a decrease of the efficiency with increasing resolution is found, since the cell near the CD is too large to reach high enough densities or temperatures due to the mixing across the CD to suffer substantial energy losses at every time-step.

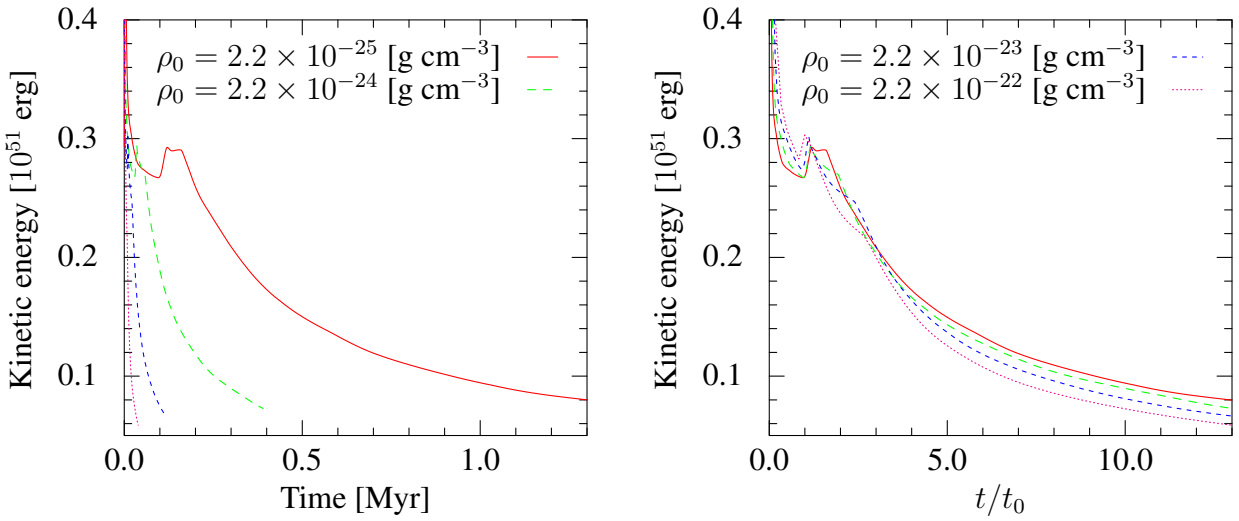


Figure 6.1: Retained kinetic energy in units of canonical SN energies ($E_{\rm SN} = 10^{51}$ erg) of a supernova in a homogeneous medium with a temperature of 1 000 K. For this simulation, an artificially stable ISM phase at the temperature and the density of the ambient medium had to be created (Code Listing B.2). t_0 is the time of maximal luminosity (defined in Sect. 6.1.1). In our simulations, a lower feedback energy efficiency in denser media is observed. The thermal energy fraction was 0.7 $E_{\rm SN}$, the SN mass loss 11 $M_\odot$, and the feedback region radius 0.3 pc.

which only differs in the initial energy ratios (purely thermal) from our reference model (Fig. 6.2 and Tab. 6.2). Moreover, our models show a slightly stronger density dependence of the feedback energy efficiency: Fig. 6.1 plots the evolution of the retained kinetic energy as a function of time in Myr in the left panel and in the right panel normalized to t_0, which is larger for lower ambient densities. Tab. 6.2 also shows that wind-less models with different spatial resolutions converge nicely.

Impact of the feedback model

The SN implementation of Thornton et al. (1998) assumes a mass loss of 3 $M_\odot$ and an energy input ($E_{\rm SN}$) of 10^{51} erg. They insert 6.9% of the SN energy via thermal energy and the rest via a linear velocity profile in a region of 1.5 pc radius.
In our preferred SN implementation (Sect. 2.7.4 and 6.1.2), 11 $M_\odot$ of ejecta are initially homogeneously distributed over a small sphere with a radius of $r_{\rm f} = 0.32$ pc. We will refer to this zone as "feedback region". Our test simulations show that the size of this feedback region does not influence the results if it is small enough to be fully contained in the wind bubble, which is the case for the presented set-ups with stellar winds. If there is no prior stellar wind, the feedback region size can influence the kinetic to thermal energy ratio after 13 t_0 (called $t_{\rm f}$ in Thornton et al., 1998). For our reference model the size of the feedback region was reduced until the kinetic to thermal energy ratio in the SN blast changed the retained kinetic energy (ϵ_k) at $t_{\rm f}$ by less than one percent (of $\epsilon_{\rm k}\,(t_{\rm f})$) in the model with the highest ambient density (Tab. 6.1). Since the bubble size of a Sedov-Taylor blast is proportional to $\rho^{-1/5}$, models with higher ambient medium density are more sensitive to the too large feedback region problem.
Increasing the feedback region radius to 1.5 pc in our reference model (Sect. 6.1.2) decreases the kinetic energy by $\sim$ 3% and increases the bubble size by $\sim$ 0.5% at 13 t_0. The variation of the

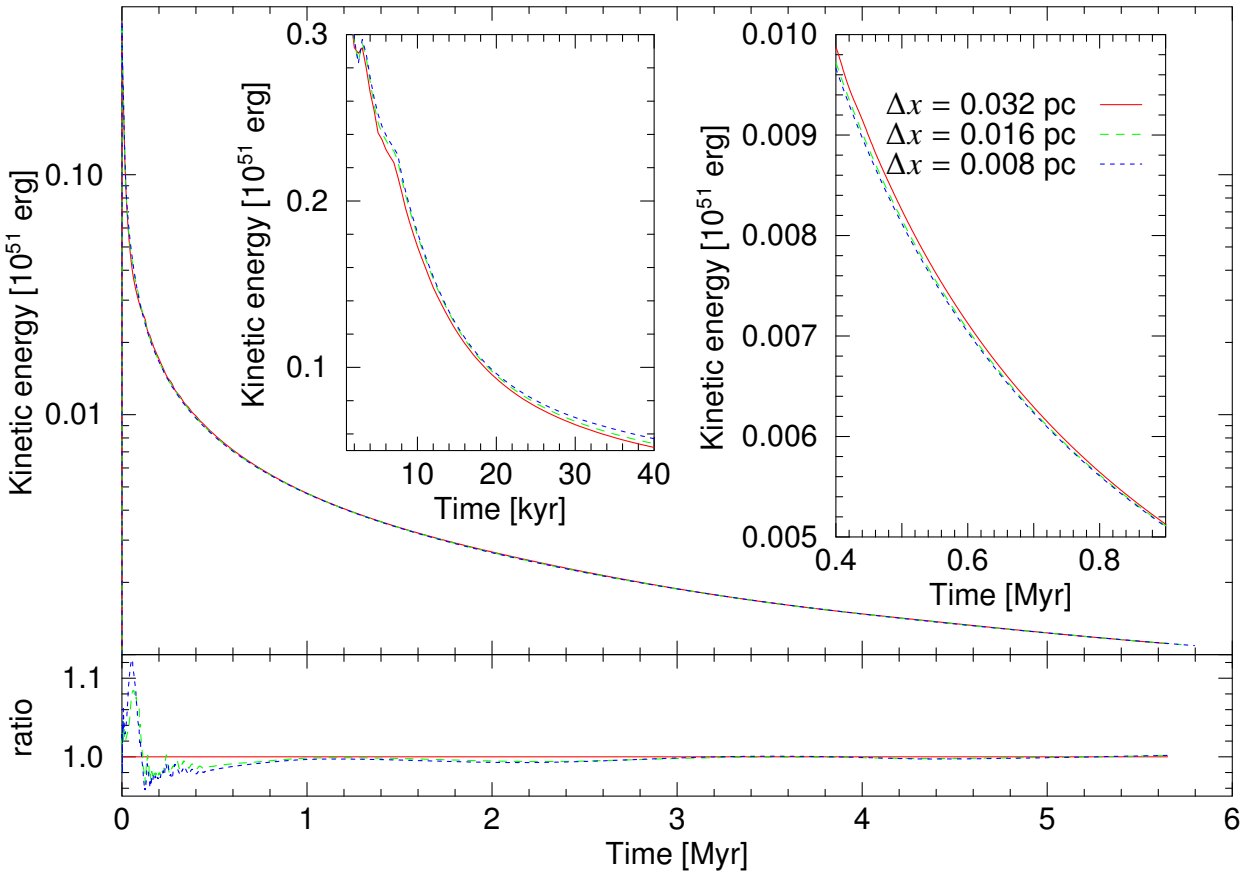


Figure 6.2: Retained kinetic energy in units of canonical SN energies (10^{51} erg) of a SN in a homogeneous medium with $T_0 = 37$ K and $\rho_0 = 2.2 \times 10^{-22}$ g cm^{-3}. The SN mass loss, leading to a kinetic energy increase, is 11 $M_\odot$. The rest of the 10^{51} erg was added as thermal energy. The energy is lost quickly via radiative cooling, but the shell needs more than 5.6 Myr to decelerate to the ambient sound speed. The lines end when the shell is decelerated to the ambient sound speed. The lower panel shows the retained kinetic energy of the models divided by the retained kinetic energy of the model with the lowest resolution at the same time. In these kinetic energy ratios it can be seen that higher resolution models lose less energy in the pressure driven phase due to the smaller cooling region at the sides of the shell (in this phase the dashed lines are above the solid line in the lower panel) but make up in the momentum conserving phase (dashed line below solid line). The left insert shows a zoom on the pressure driven phase. After a Myr the results for different resolutions are very well converged. The convergence of the retained energies at different resolutions can be seen in the right insert and in the lower panel. The model with $\Delta x = 0.004$ pc is not shown, since it was stopped after 37 t_0.

feedback region radius is also the leading effect causing the differences between the two 1000 K models in Tab. 6.1.

The thermal energy fraction of the SN energy in our 1 000 K models in Tab. 6.1 is 72% (which is Sedov-Taylor-like, see Sect. 4.3.1). In the 37 K model shown in Tab. 6.1, all SN energy was inserted via thermal energy. Therefore no mass loss was used. This leads to a slightly different kinetic to thermal energy ratio before t_0 than the ratio found in models in which the energy fractions at the SN blast are chosen according to the Sedov-Taylor solution. After 200 kyr, a model that

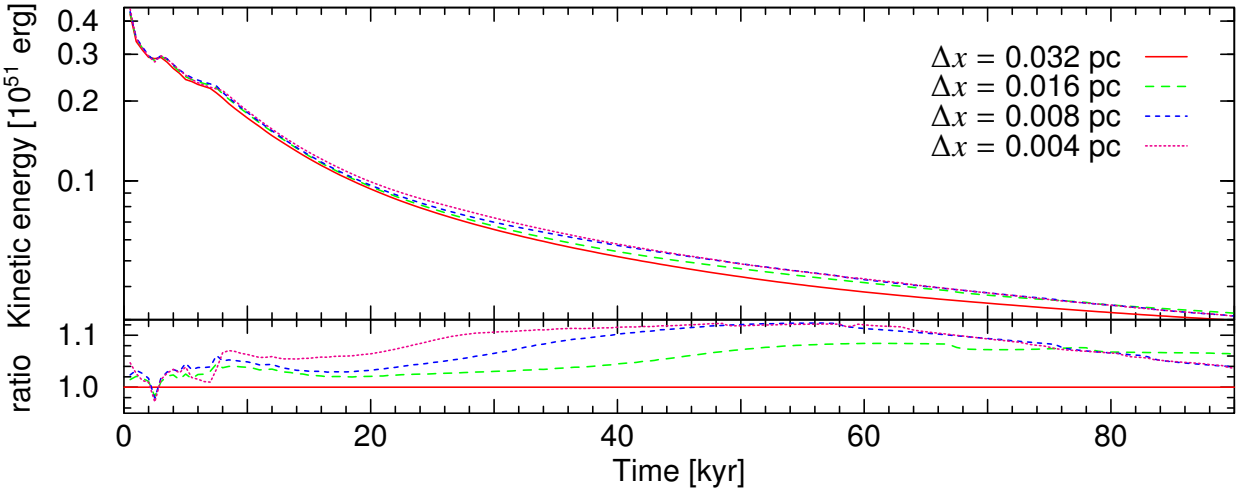


Figure 6.3: Zoom of Fig. 6.2. In this plot the highest resolution model is added, which was stopped after 40 t_0.

differs only in the mass loss (3 $M_\odot$) from our reference model (Sect. 6.1.2) still retains a kinetic energy of 0.01678×10^{51} erg. In contrast, replacing the Sedov-Taylor like energy ratios by purely thermal energy input in this model results in 0.01684×10^{51} erg at this time. We conclude that for small enough feedback region radii the energy fractions in the SN blast do not have a significant impact on the feedback energy efficiency.
Tab. 6.1 shows that t_0 occurs later, if the mass of the SN ejecta is increased from 3 $M_\odot$ to 11 $M_\odot$ (as in our preferred SN model, which is discussed in Sect. 2.7.4). However, this increase only slightly lowers the feedback energy efficiency: After 200 kyr our reference model (Sect. 6.1.2) still retains 0.01637×10^{51} erg kinetic energy, whereas, as already mentioned, the same model in which only the SN mass loss was changed to 3 $M_\odot$ finds 0.01678×10^{51} erg at this time. The unimportance of the mass of the ejecta is not surprising, since in an ambient medium with $n = 2.2 \times 10^{-22}$ g cm^{-3} the swept-up shell's mass exceeds 11 $M_\odot$ as soon as the bubble's radius is larger then 2 pc.

Impact of the ambient pressure

Comparing models with $T_{\rm eq}(n_0) = 37$ K and $T_{\rm eq}(n_0) = 1\,000$ K, with $n_0 = 2.2 \times 10^{-22}$ g cm^{-3}, 3 $M_\odot$ mass injection and Sedov-Taylor like energy ratio (Sect. 4.3.1) shows that the ambient pressure only has a minor effect on the feedback energy efficiency: The changes in bubble size (5.93 pc for both models) and kinetic energy (0.06878×10^{51} erg vs. 0.06836×10^{51} erg) after 13 t_0 (32.5 kyr) are less than a percent and would thus be invisible in Tab. 6.1. As expected, higher ambient pressure leads to a slightly smaller bubble, if the model is followed for a longer time: e.g. after 200 kyr we find a shell radius of 9.60 pc and a kinetic energy of 0.01678×10^{51} erg in the 37 K model and 9.54 pc and 0.01505×10^{51} erg in the 1000 K model. However, this is a very small effect and is less important compared to the spatial resolution and the size of the feedback region.

Convergence

The retained kinetic energies at 13 t_0 in the $T_{\rm eq}(n_0) = 37$ K models in Tab. 6.1 indicate a dependence of the feedback energy efficiency on spatial resolution. However, Fig. 6.2 to 6.3 show that this problem is only found in the first Myr and the retained kinetic energies of the 37 K models

without wind converge for all resolutions (0.004 to 0.032 pc) as soon as the shell has cooled to the equilibrium temperature and cooling losses only occur in the newly swept-up compressed and heated gas at the outside of the shell. The zone, which is suffering cooling losses, is resolved with several cells. At this time the pressure in the swept-up shell is already larger than the pressure inside the bubble. For all spatial resolutions a kinetic feedback energy efficiency of 0.11 % is recovered when the shell speed reaches the ambient sound speed.

Phases of SN bubble evolution

In Sect. 4.3 and 4.5 we explained, which power laws we would expect after a SN explosion. We will now check, if our simulations behave accordingly.

Simulated pressure driven expansion

During the pressure driven expansion, the largest cooling losses arise near the CD, where a strong density gradient at the interface between the dilute bubble material and the swept-up ambient medium is found. The maximum luminosity is reached earlier for simulations with larger cells, since lower resolution will mix more of the hot gas in the bubble with the swept-up medium and thus enhance the cooling losses.
Sect. 4.5.1 finds $r \propto t^{2/7}$, $v \propto t^{-5/7}$ and $E_{\rm kin} \propto t^{-4/7}$ (Eq. 4.61 to 4.63) for the adiabatic pressure driven expansion and Sect. 4.4.1 finds $r \propto t^{2/5}$ and $v \propto t^{-3/5}$ (Eq. 4.42 and 4.42) leading to constant kinetic energy for the fully radiative case. The best fits to the 37 K models for times between the time of maximal luminosity t_0 (defined in Sect. 6.1.1) and the time when the pressure inside the bubble has decreased to the ambient pressure (Tab. 6.3, column 5) are $r \propto t^{0.272}$, $v \propto t^{-0.75}$ and $E_{\rm kin} \propto t^{-0.7}$. These fits rather resemble the behavior of the momentum-conserving phase ($r \propto t^{1/4}$, $v \propto t^{-3/4}$ and $E_{\rm kin} \propto t^{-3/4}$, Eq. 4.65 to 4.67). And indeed, our models show that the pressure inside the bubble is much lower than the pressure in the shell. In contrast to the analytic model, the simulated shell is not infinitely thin and resolved with several cells. Column 3-4 in Tab. 6.3 list the times, when the shell pressure becomes larger than the bubble pressure. These times mark the end of the purely bubble pressure driven phase and very close to these times (near 8 kyr) a "knee" can be seen in Fig. 6.2 and 6.3. Moreover the best fits for the radius and the velocity in this short period of time are in agreement with fits of a pressure driven phase. The total kinetic energy decreases more slowly than a pressure driven fit would predict, since not all the kinetic energy is stored in the shell.
Tenorio-Tagle et al. (1990) and Tenorio-Tagle (1996) report hot swept-up matter separating the CD several parsecs from the outer shock for their SN explosion in a homogeneous medium. This is also seen in our simulation with $n_0 = 1$ cm^{-3}, $T_{\rm eq}(n_0) = 100$ K. The CD and the outward shock are at the same radius as reported by Tenorio-Tagle et al. (1990). In our simulations the hot material between the CD and the thin dense shell (with a sub-parsec shell width, created by a sound wave from the reverse shock) is hot shocked swept-up ISM.

Simulated momentum conservation

Comparing the pressure inside the bubble to the pressure of the ambient medium shows that at 13 t_0 ($\sim$ 40 kyr) the $T_{\rm eq}(n_0) = 1\,000$ K model is already in the momentum conserving phase, whereas the bubble pressure in the 37 K model is still higher than the ambient pressure (but lower than the shell pressure). The times when the pressure inside the bubble has decreased to the

p_0 [erg cm^{-3}]	Δx [pc]	t [kyr] peak	t [kyr] average	t [kyr] bubble	$E_{\rm kin}$ [10^{49} erg]
3.99×10^{-11}	0.032	6.5	7.5	34.5	6.32
1.83×10^{-12}	0.032	9.5	9.5	118.5	2.58
1.83×10^{-12}	0.016	8.0	8.0	147.0	2.12
1.83×10^{-12}	0.008	6.5	6.5	174.0	1.85

Table 6.3: Ends of pressure driven phases. This table lists the times, when pressures in the shell or the ambient medium (p_0) become larger than the pressure inside the bubble. In all four models, the SN without prior winds is placed in a homogeneous ambient medium with a density of 2.2×10^{-22} g cm^{-3}. The ambient medium is in cooling-heating equilibrium: at 1 000 K in the first model and at 37 K in the other models. Column 1 (p_0) lists the ambient pressure, column 2 (Δx) the cell size in the simulation. Column 3-5 contain the times when the pressure inside the bubble becomes smaller than the peak pressure in the shell (column 3), the average pressure in the shell (column 4) or p_0 (column 5). Column 6 lists the retained kinetic energy at the times in column 5.

ambient pressure are listed in Tab. 6.3. Eq. 4.67 was used to fit the kinetic energy evolution of the simulations after the times listed in column 5 of Tab. 6.3. The fits of the bubble radius, the shell velocity and the kinetic energy show that the kinetic energy decreases more slowly than Eq. 4.67 predicts (resp. the shell moves faster). The best fit to the bubble radius after the end of the pressure driven phase is $r \propto t^{0.28}$ (Eq. 4.65 predicts $r \propto t^{0.25}$). The best fits for velocity and kinetic energy are $v \propto t^{-0.77}$ and $E_{\rm kin} \propto t^{-0.78}$ (Eq. 4.66 and 4.67 predict $v \propto t^{-0.75}$ and $E_{\rm kin} \propto t^{-0.75}$).
The ratio between the shell's kinetic energy and the bubble's kinetic energy as well as the deviations of the fit from the kinetic energy found in the simulations in Fig. 6.4 indicate that the overpressure in the cavity wall leads to an expansion of the shell into the cavity. As a consequence, a high pressure wave starts to run back and forth in the cavity (Fig. 6.5). The impacts onto the shell increase the shell velocity.
The time when the shell velocity reaches the sound speed can be estimated from the fits by setting Eq. 4.66 equal to the sound speed. The mass of the swept-up medium can be estimated from the expected radius and the ambient density and leads to a kinetic energy, when combined with the sound speed. Since this approximation assumes that all swept-up medium is compressed into an infinitely thin pressure-less slab, all fits predicted a shorter time and a higher final kinetic energy than the simulation data. In the simulation, the highest velocity is found near the densest cell. This cell is only a few cells away from the undisturbed ambient medium. However, the overpressure in the shell leads to a flow of swept-up medium into the shell. It is observed that the peak density decreases during the simulation. Since not all swept-up medium is accelerated to the peak velocity, the shell can travel longer before the peak velocity falls below the ambient sound speed. The lower than expected kinetic energy is also due to the fact that much of the gas at the inner side of the shell was already significantly decelerated.

6.2 SN blast in a cavity

Since the progenitor stars of SNe have strong stellar winds, SN explosions always happen inside wind-blown bubbles. In this section we show that this is not a detail but a very important feature

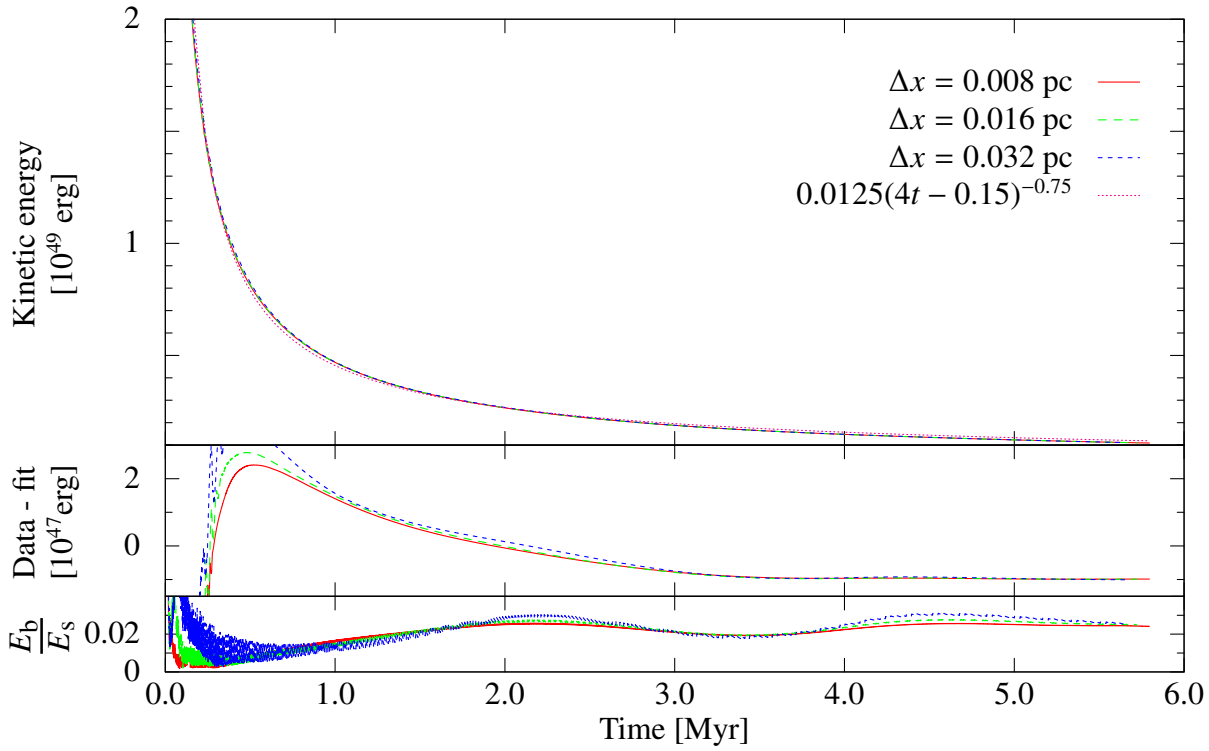


Figure 6.4: Fit of a momentum conserving shell to the data. The middle panel shows the deviations from the fit. It can be seen that the kinetic energy decays more slowly than a momentum conserving model predicts. This indicates that the widening of the over-pressured shell contributes to the growth of the cavity. In the lowest panel the kinetic energy of bubble-gas is compared to the kinetic energy of the dense shell. The oscillations are caused by a wave traveling inside the cavity (see text).

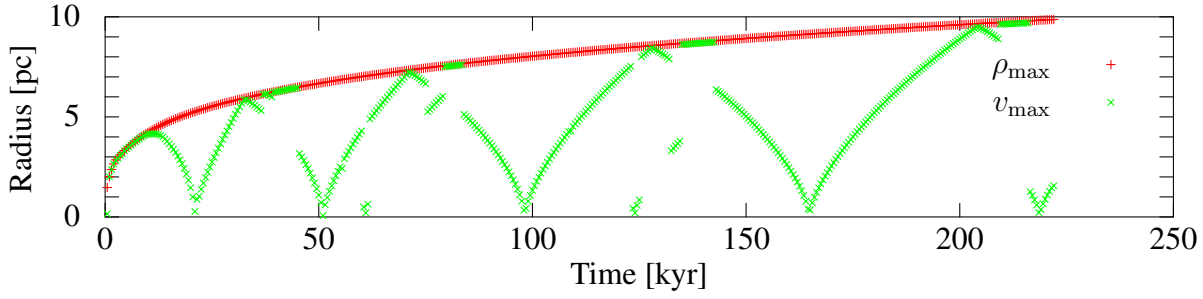


Figure 6.5: The position of the densest cell (red) is an indicator of the shock position. The over-pressure in the swept-up shell causes a wave inside the cavity, which can be tracked by the position of the maximal absolute value of the velocity (green). The jumps in the green dots are caused by a second wave, starting in the compressed material at the second reflection of the aforementioned wave in the center of the cavity.

of the model, since it strongly influences the feedback energy efficiency.
The stellar winds in the 60 $M_\odot$ model based on rotating models of Ekström et al. (2012) as described in Sect. 2.7.3 insert 2.34 times the SN energy into the ambient ISM. This wind-to-SN ratio is larger than in Voss et al. (2009), since we consider individual massive stars, whereas Voss et al. (2009) are interested in OB associations with in the order of 100 members. In groups of stars, less massive stars lower the ratio of wind energy to SN energy if a canonical SN energy of 10^{51} erg is assumed.
The massive star first produces a stellar wind bubble and subsequently undergoes a SN explosion. The bubble structure contains a contact discontinuity (CD) separating two distinct phases of the ISM: a hot dilute[2] phase, of stellar wind gas which cannot cool due to its low density and a cold, denser[3] phase, which also does not cool strongly, because its thermal energy is too low to cool efficiently (the cooling curves e.g. in Sutherland and Dopita (1993) show a strong increase of $\Lambda(n, T)$ above 10 000 K).

6.2.1 Comparison to previous work on SNe in pre-existing bubbles

Tenorio-Tagle et al. (1990) study SNe exploding in bubbles blown by a constant WR wind mimicking the feedback of a 40 $M_\odot$ star with a mass loss rate of $\dot{M} = 3 \times 10^{-5}\ M_\odot\ \mathrm{yr}^{-1}$ and a terminal velocity of 1 000 km s^{-1} into a homogeneous medium with a number density $n_0 = 1\ \mathrm{cm}^{-3}$ and a temperature of 100 K. In their study the wind phase ends as soon as the wind bubble has reached a predefined radius (r_bubble). They found feedback energy efficiencies of 50% ($r_\mathrm{bubble} = 4.5$ pc, ~ 60 kyr after the SN: $E_\mathrm{kin} \sim 3 \times 10^{50}$ erg, $E_\mathrm{th} \sim 2 \times 10^{50}$ erg) to 70% ($r_\mathrm{bubble} = 15$ pc, ~ 40 kyr after the SN: $E_\mathrm{kin} \sim 3 \times 10^{50}$ erg, $E_\mathrm{th} \sim 4 \times 10^{50}$ erg). We re-simulated these models with different cooling prescriptions and found that the feedback energy efficiencies were relatively robust against these changes (Fig. 6.6).
In the case of the 4.5 pc bubble we find feedback energy efficiencies similar to Tenorio-Tagle et al. (1990). For this model the plots in Tenorio-Tagle et al. (1990) show that the temperature in the dense shell is below 10^3 K (interestingly, this minimal temperature in the dense region is more than a factor 10 lower than the minimal temperature in the referenced cooling table). For this comparison, we also used the cooling curve in Fig. 1 of Raymond et al. (1976), which provides a cooling function for temperatures above 10^4 K. Consequently, this is the minimum temperature reachable via radiative cooling in our simulations. Fig. 6.7 shows that this temperature floor is reached in the bubble walls of our 15 pc model. By contrast, Fig. 7 of Tenorio-Tagle et al. (1990) shows a shell considerably hotter than this minimal temperature. Also, our models show a steep density decline between the supersonic shock and the ambient medium and a smoother decline of the density towards the hot bubble. Fig. 7 of Tenorio-Tagle et al. (1990) indicates that they seem to find a not step-like density increase between the supersonic shock and the ambient medium. This difference arises, since the PLUTO code treats the shock with a Riemann solver in contrast to the artificial viscosity treatment in Tenorio-Tagle et al. (1990).

Stellar wind bubble sizes

The bubbles considered in Tenorio-Tagle et al. (1990, 1991) and Rozyczka et al. (1993) have radii of up to 16 pc at the SN. These bubble sizes seem rather small if the wind model of Voss

[2] several orders of magnitude below the ambient density, 10^6K or hotter
[3] more than a factor 4 denser than the ambient medium, 10 K

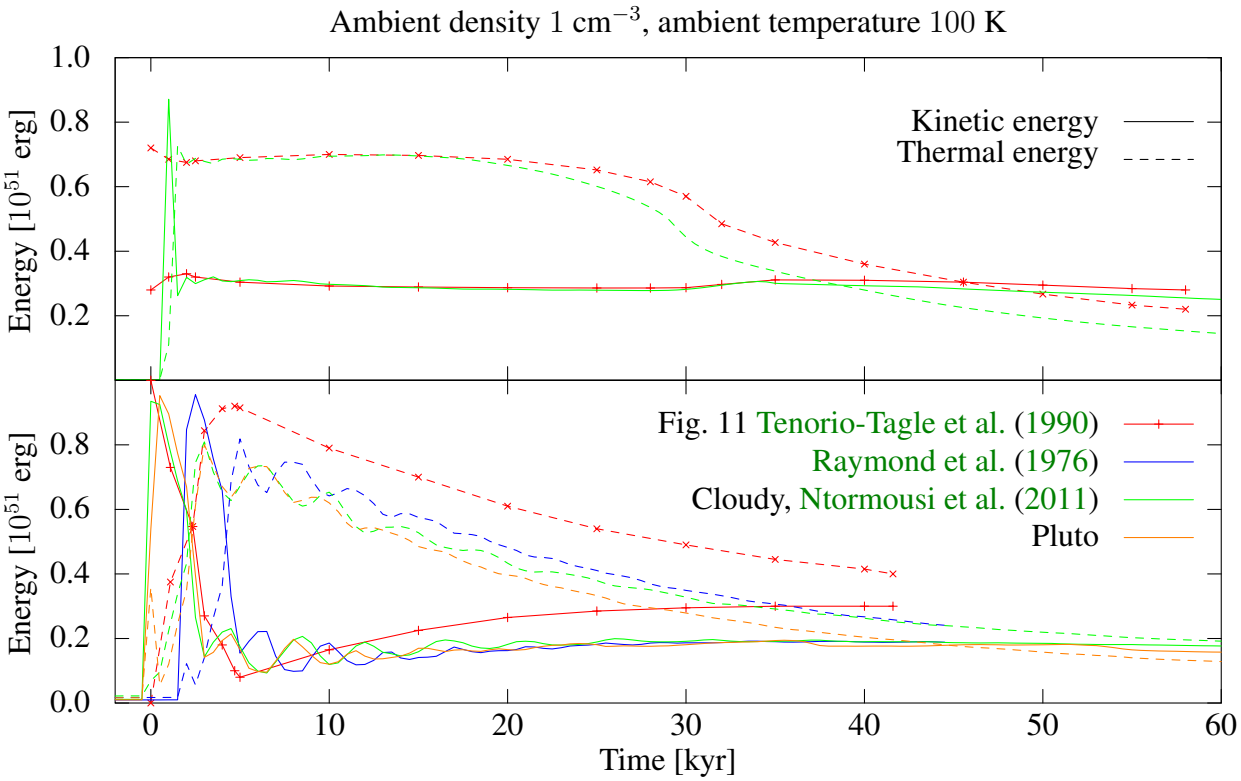


Figure 6.6: The feedback energy efficiencies of our test simulations of SN explosions inside wind bubbles in a $n_0 = 1$ cm^{-3}, $T_{\rm eq}(n_0) = 100$ K ambient medium show results similar to Fig. 11 of Tenorio-Tagle et al. (1990) for the 4.5 pc bubble (top panel) and much lower feedback energy efficiencies for the 15 pc bubble (lower panel). The retained kinetic energies in our models are relatively insensitive to the cooling table: the lower panel compares simulations with Pluto cooling, Cloudy cooling (as implemented by Ntormousi et al., 2011) and with the cooling according to Fig. 1 of Raymond et al. (1976). The disagreement between Tenorio-Tagle et al. (1990) and our work is caused by differences in the structure of the shock/ambient medium interface (see text).

et al. (2009) is applied to the rotating stellar models of Ekström et al. (2012) (Sect. 2.7.3): The lowest mass star still ending in a SN in this grid of models already produces a bubble with a radius of 13.6 pc in a 100 times denser medium ($n_0 = 100$ cm^{-3} and $T_0 = 100$ K). Since the bubble sizes considered in Tenorio-Tagle et al. (1990, 1991) and Rozyczka et al. (1993) were based on observations, this indicates that higher densities of the ambient medium should be taken into account. Thus, we put our emphasis on $n_0 = 100$ cm^{-3} models instead of $n_0 = 1$ cm^{-3} models. The ambient density plays an important role for the feedback energy efficiency: Models with higher ambient densities have lower feedback energy efficiencies (Fig. 6.1, Tab. 6.1).

Minimal energy bubbles: Is there a dichotomy of SNe in stellar wind bubbles?

Tenorio-Tagle (1996) report a dichotomy of wind-blown bubbles: (1) light bubbles, which are overrun by the SN-shock and (2) stable bubbles that switch to the radiative phase as soon as they are hit by the blast. For reference we produced a set of stellar wind bubbles with a constant wind, consistent with the feedback used by Tenorio-Tagle et al. (1990) and ignited the SN as soon as the

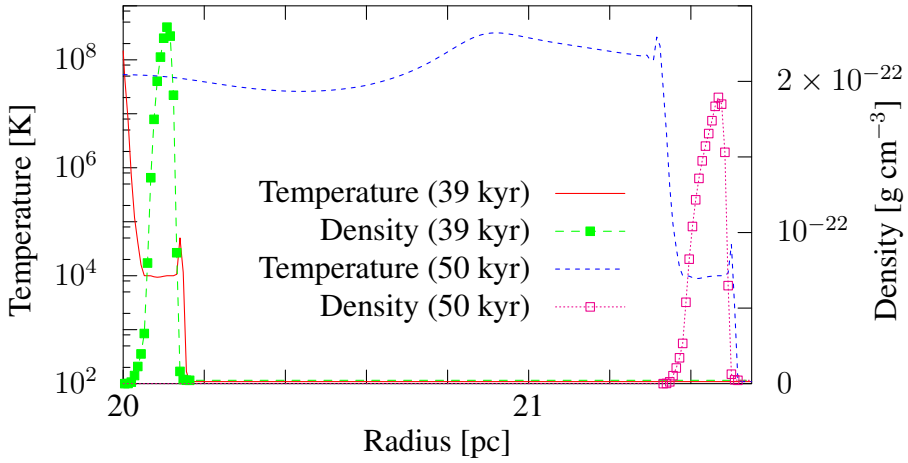


Figure 6.7: The shell temperature 39 resp. 50 kyr after a SN blast in a 15 pc wind cavity in a $n_0 = 1$ cm^{-3}, $T_{\rm eq}(n_0) = 100$ K ambient medium with the cooling table in Fig. 1 of Raymond et al. (1976) shows that the dense shell has cooled to the minimum temperature reachable via radiative losses. It can also be seen that the width of the density step between the ambient medium and the supersonic shock is in the order of 5 grid cells. Moreover, due to the different shock structure, the 15 pc cavity of Tenorio-Tagle et al. (1990) has the densest cell near 14 pc, whereas our model has the densest cell near 15 pc, when the SN explodes.

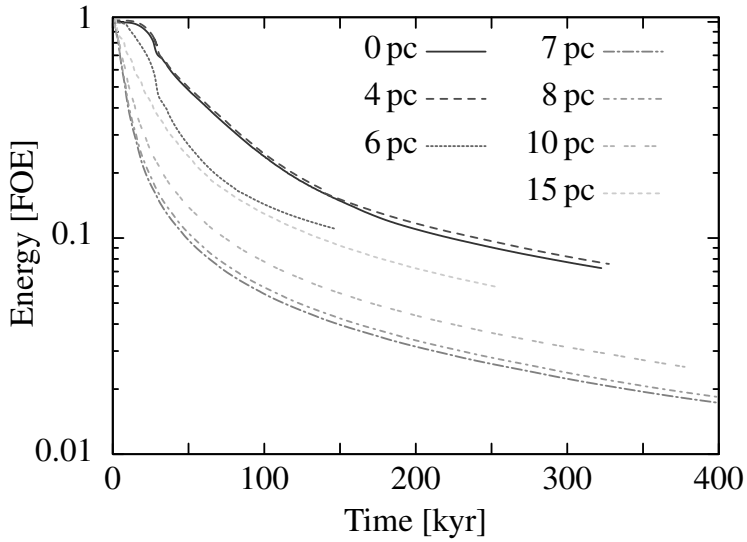


Figure 6.8: Minimal energy bubbles. In this study SNe exploded at $t = 0$ in a cavity of given radius. These cavities were created by a constant wind and the ambient medium has $n_0 = 1$ cm^{-3}, $T_{\rm eq}(n_0) = 1\,000$ K. The feedback energy efficiency of a SN in a pre-existing bubble depends on the bubble size, since on the one hand, the bubble can act as a pressure reservoir due to the very small cooling losses inside the bubble and on the other hand, the dense cavity walls lead to large radiative losses. It can be seen that bubbles of ~ 7 pc radius have the smallest feedback energy efficiency. Such bubbles are, however, even too small for the winds of the least massive star ending in a SN. For larger radii the feedback energy efficiency rises with increasing radius.

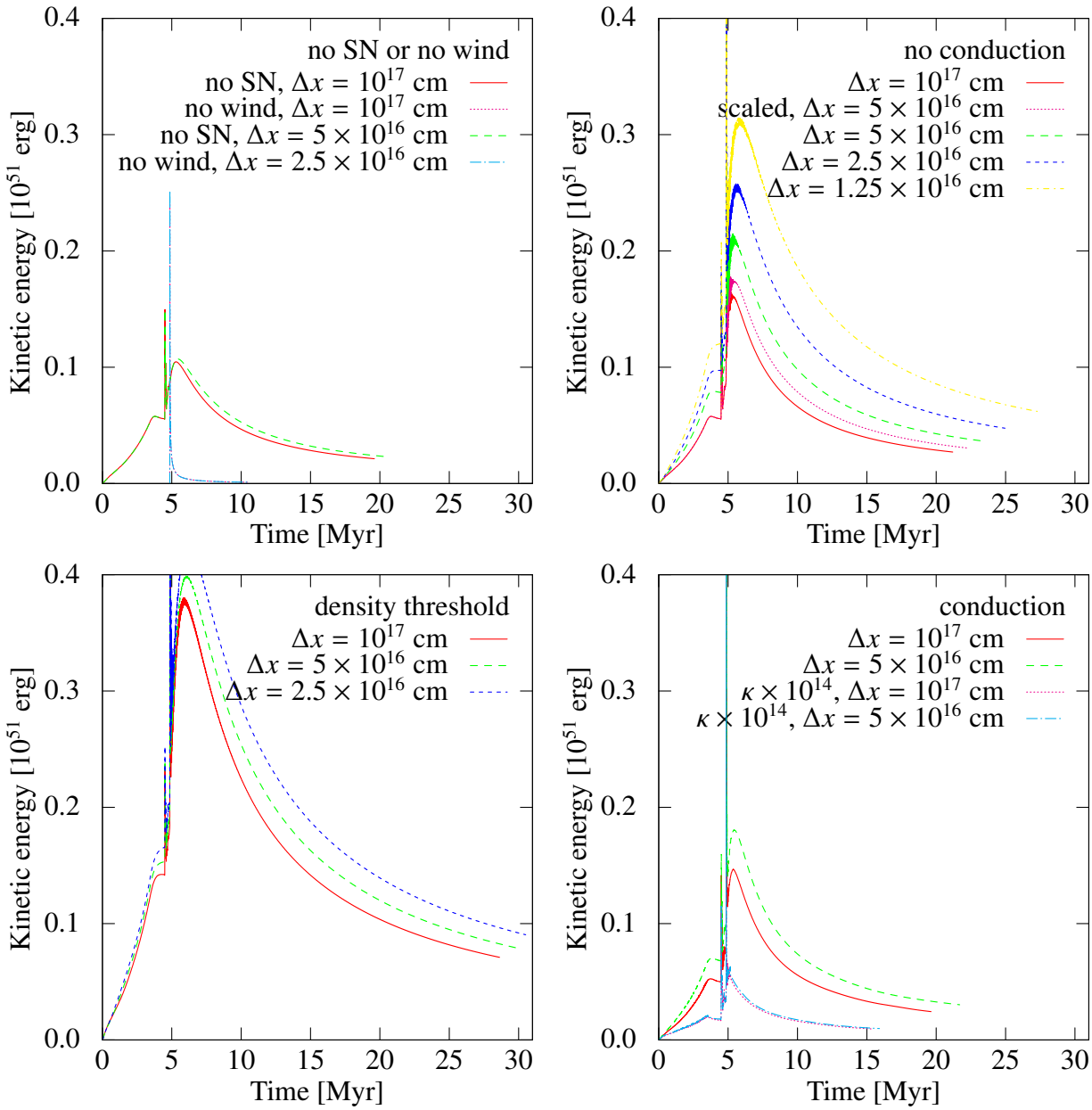


Figure 6.9: Time evolution of the retained kinetic energy. The wind phase ends after 4.859 Myr. All lines end when the densest cell is decelerated to the ambient sound speed. Simulations with a supernova without pre-existing wind bubble have a six times lower feedback energy efficiency than supernovae in pre-existing bubbles [Tab. 6.2: for $\Delta x = 0.032$ pc the simulation with a supernova without wind leads to 0.11×10^{49} erg of kinetic energy compared to the difference between simulations with wind and with/without supernova: 0.58×10^{49} erg]. $t_\mathrm{f} = 13\ t_0$ [t_0 is the time of max. loss, at t_f the efficiencies are evaluated.] as defined by Thornton et al. (1998) is 4.8915 Myr for the model without wind (kinetic shell energy: 0.61×10^{50} erg) and ranges from 4.9955 Myr to 5.0605 Myr for all other simulations.

desired bubble radius was reached. Fig. 6.8 shows that we observed minimal energy bubbles in between these two cases: The minimal efficiency occurred at "intermediate" cavity sizes of 7 pc

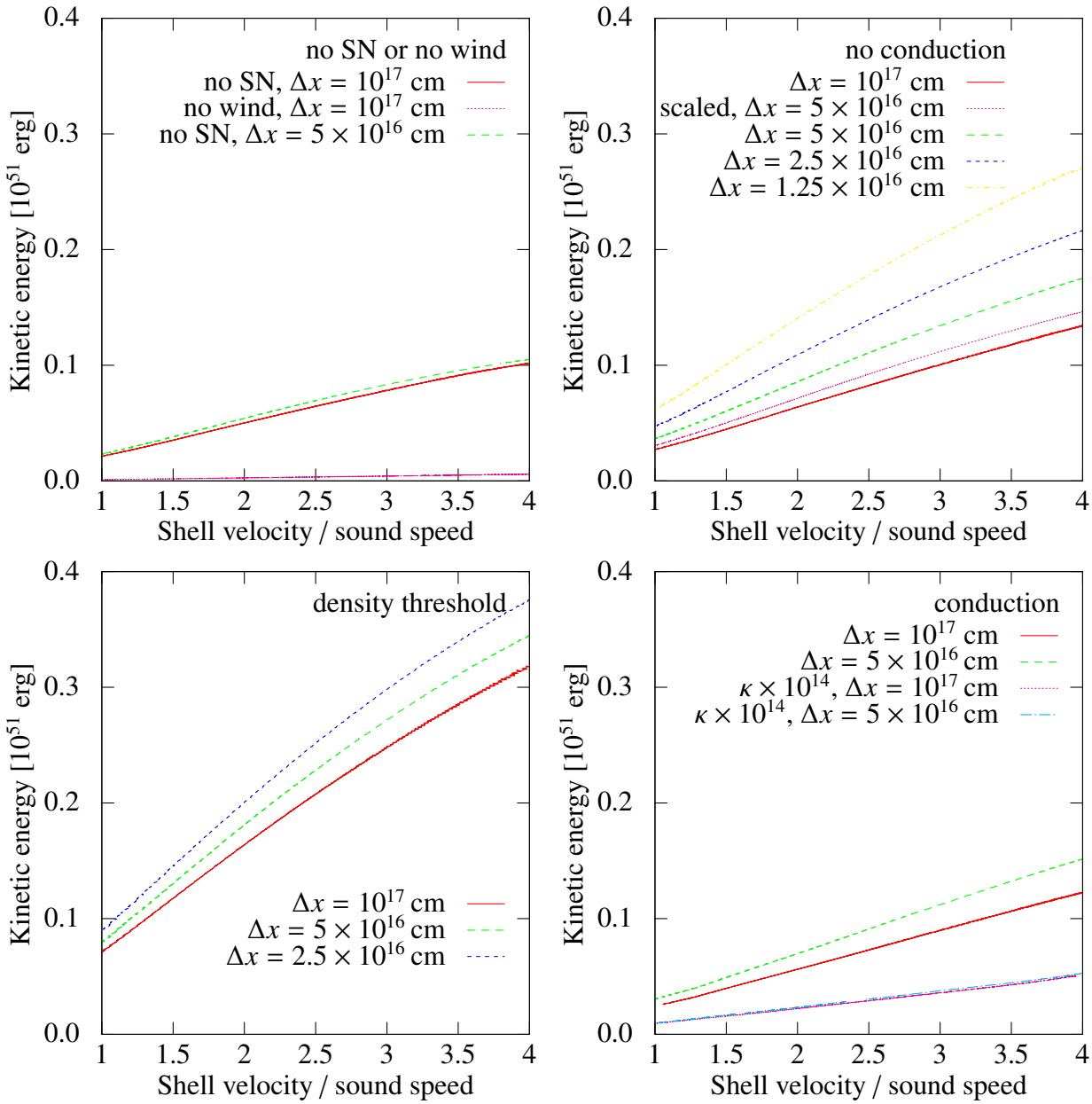


Figure 6.10: Variant of Fig. 6.9: The retained kinetic energy for the same models is displayed as a function of the densest cell's velocity instead of time. The velocities are normalized to the ambient sound speed (~ 1 km s^{-1}).

in a $n_0 = 1$ cm^{-3}, $T_{\rm eq}(n_0) = 1000$ K medium. This minimum is created by the counteracting effects of (1) efficient cooling in the denser shells of larger bubbles and (2) the larger cavities with inefficient cooling serving as pressure reservoirs. However, this minimum is of academic interest only. Modeling the wind of the lightest star, which still ends in a SN shows that even the stellar winds of this star can produce a cavity larger than 7 pc before the SN. Thus, our models indicate that nature does not produce such minimal energy bubbles.

Wind phase

During the stellar wind phase the models show the structure expected from stellar wind bubble theory (Pikel'Ner, 1968; Avedisova, 1972; Castor et al., 1975; Weaver et al., 1977; Dyson, 1977, and Sect. 4.4.1). Obviously, the width of the zones in models affected by radiative cooling losses have to differ from these simple adiabatic models (e.g. the swept-up shell is thinner, denser and moves more slowly into the ambient medium). A region of freely expanding wind (mostly kinetic energy; cold gas, discussed in Sect. 4.4.3) is separated from the thermalized ejecta (mostly thermal energy; hot dilute plasma) by a reverse shock. The presence of this free expansion zone in our simulations shows that our feedback region radius is not too large. The $p\mathrm{d}V$ work of the thermalized ejecta sweeps up the ambient medium. This medium forms a thin, efficiently cooling shell, which is separated from the thermalized ejecta by a contact discontinuity (CD). Due to the absence of pressure and velocity gradients across this surface, no mixing (except for diffusion) between the medium inside and outside the CD is expected (see also Tenorio-Tagle, 1996).

Post SN phase

If a SN explodes in the wind bubble of its progenitor, the blast wave expands freely and also adiabatically in the dilute medium inside the wind bubble. In a pre-existing cavity the Sedov-Taylor expansion phase (Sect. 4.3) is skipped (see e.g. Tenorio-Tagle et al., 1990). After the free expansion phase, when the blast wave hits the bubble wall, the evolution continues snowplow-phase-like (Sect. 4.5).
Since radiative losses start when the wind shell is hit, the time maximal luminosity (t_0) occurs later than it would have in absence of the wind bubble. In fact, the SN ejecta do not reach the dense shell of swept-up ambient medium. They rather compress the wind gas and get reflected. Thus – according to our models – the velocity of the SN-ejecta is expected to be higher than the velocity of the gas in the bubble wall.
After reflection from the bubble wall the SN blast wave continues to travel back and forth inside the cavity. These sound waves can be seen as oscillations in the kinetic and thermal energy evolution (e.g. in Fig. 6.6 and near the kinetic energy peak in Fig. 6.9) as well as in the cooling losses. The bouncing SN blast wave inside the wind cavity causes double peaks in the loss rate: The first maximum in the loss is reached when the cavity wall (resp. the wind gas in front of it) is compressed and kinetic energy is converted to thermal energy and the second (smaller) peak is found when the wall expands and thermal energy is converted back to kinetic energy. Due to the reflection of this wave inside the cavity the interaction of the wave and the cavity wall causes periodic conversions between thermal and kinetic energy with decreasing peak loss values until the SN wave is damped away.
As in the models without progenitor winds the cold outer shell is accelerated by $p\mathrm{d}V$ work from the hot (SN-) gas inside the bubble. In later stages, when the pressure in the bubble becomes ineffective, momentum conservation pushes the shell into the ambient medium.

6.2.2 Feedback energy efficiency: winds or SNe?

Fig. 6.9 and 6.10 show the kinetic energy evolution of our models summarized in Tab. 6.2. For these models time resolved stellar winds of a 60 $M_\odot$ star (Sect. 2.7.3) were blown into a homogeneous medium with a density of $\rho_0 = 2.2 \times 10^{-22}$ g cm^{-3}. The ambient medium is in cooling-heating equilibrium as described by Ntormousi et al. (2011). In cells with densities (ρ) above $a\rho_0$

radiative cooling is taken into account (see also section 6.2.3). Less dense cells do not suffer cooling losses. The grid of models spans $a = 0$ to 1.3 (only the model with $a = 1$ and $a = 0$ are shown in Fig. 6.9 and 6.10) and the resolutions of 1, 2, 4 or 8 cells per 0.032 pc ($= 10^{17}$ cm). The time of the SN explosion is set by the stellar model, thus the wind bubble size can only be influenced indirectly via the density of the ambient medium and the chosen stellar model. [In contrast to the constant wind test shown in Fig. 6.8, where the SN explosions occur at a pre-defined bubble size]. For reference some models in our grid lack the SN explosion or the wind phase. The model without SN explosion demonstrates the importance of stellar winds: The total energy input into the wind-only model is 2.34×10^{51} erg, which is $\sim 70\%$ of the total energy input of a model with wind and SN. The kinetic energy of the shell in the wind-only model at the time when it is decelerated to the ambient sound speed is 79% of the final energy of the model with a SN blast after the wind phase.
Another indication that continuous energy input is more efficient than blasts is the comparison between the model with a constant wind (CW) and the model with time dependent wind strengths (Tab. 6.2). For reference the same total wind energy and wind ejecta mass are inserted at a constant rate in the CW model. This steady wind has more power at early times, since the energy input of the WR phase is distributed over time: before the SN (occuring after 4.86 Myr) the mass loss rate is $8.65 \times 10^{-6}\ M_\odot\ \mathrm{yr}^{-1}$ (compare to Fig. 2.11) and the energy injection rate is 4.8×10^{44} erg yr^{-1} or $1.53 \times 10^{-14}\ E_{\mathrm{SN}}\ \mathrm{s}^{-1}$ (compare to Fig. 2.9). In total both, the constant and the time resolved model for the $60\ M_\odot$ star's wind inject 2.34×10^{51} erg into the surroundings during the wind phase. We find that the steady winds produce larger bubbles than time-resolved winds with the same total energy input. Since wind-blown bubbles serve as pressure reservoirs after the SN, higher efficiencies are found for larger bubbles (see also Sect. 6.2.1).
Overall it can be seen that wind-bubbles enhance the feedback energy efficiency. For example, the models with a resolution of 0.032 pc without progenitor wind retain 1.1×10^{48} erg of 10^{51} erg (0.11%), whereas models with preexisting bubbles retain more than 4.7×10^{49} erg of 3.34×10^{51} erg (1.5%).

6.2.3 Zones with enhanced radiative losses

The largest cooling losses of the models are

- at the CD during pressure driven phases.
- in the dense shell during momentum conserving phases.

High resolution simulations are more efficient in the wind phase (and other pressure driven phases), because

- in all simulations in this grid Δx is small enough that a strongly mixing, cooling cell exists at every time-step.
- the volume of the strongly cooling layer gets smaller at higher resolutions.
- smaller cells lead to a better separation of the media and mimic a gas with less efficient mixing processes.

The deviations from the heating-cooling equilibrium and the cooling losses are shown in Fig. 6.11. In this figure the evolution of the gas phases in the $a = 0, \Delta x = 0.016$ pc model are visualized. The

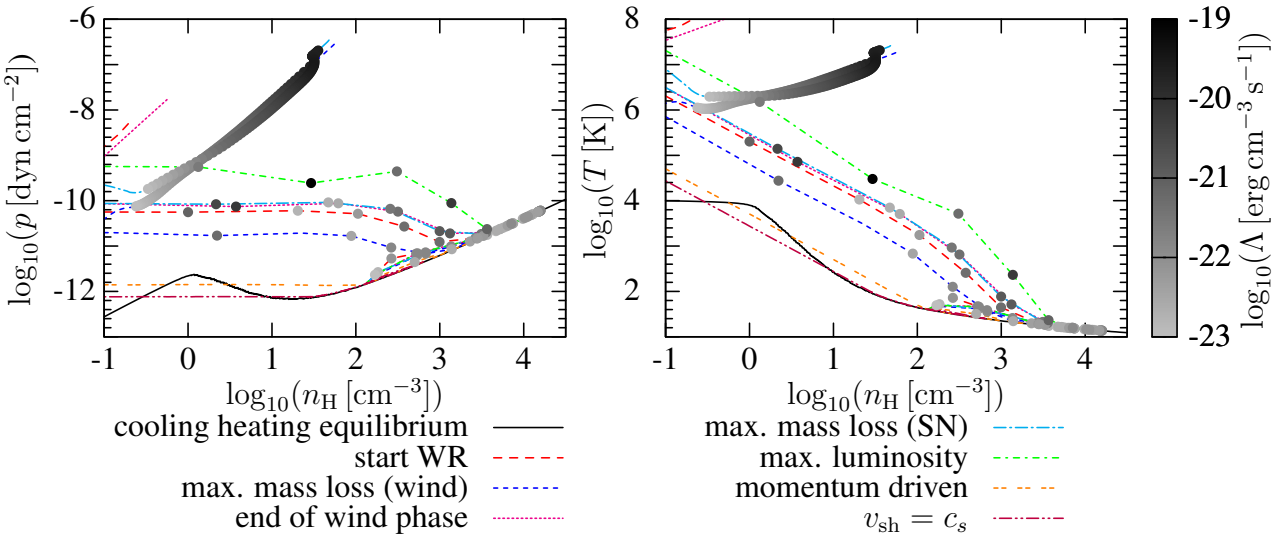


Figure 6.11: Gas phases in the $a = 0$, $\Delta x = 0.016$ pc model: The densest cells approach the heating-cooling equilibrium (solid line). The fill color of the dots carries information on the radiative losses. The dark colors of the rightmost points on the curves show cooling losses in the dense, swept-up shell. Bright points on the equilibrium curve depict the ambient medium. The lines connecting dots should guide the eye and link gas from adjacent cells. The dark dots in the center show the cooling losses near the CD. The plot compares the location of the cooling losses at different stages of the evolution. When the mass loss rate peaks, cooling losses of dense gas are found near the feedback region. Typical pressure driven phases (start of the WR phase at 3.457 Myr, end of the wind phase at 4.4975 Myr, time of maximal luminosity at 4.8695 Myr) show cooling losses near the CD, whereas losses in the dense shell dominate during momentum driven phases (in the plot the start of the momentum driven phase at 9.7595 Myr and the end of the simulation ($v_\mathrm{sh} = c_\mathrm{s}$) at 23.3975 Myr are shown).

solid line shows the cooling-heating equilibrium curve. The ambient medium would be represented by a very bright dot (no losses) on the equilibrium curve. The gas properties in the swept-up shell and inside the bubble are shown by dots linked with lines connecting adjacent cells. The color of the dots contains information on the radiative losses. It can be seen that there are two regions with enhanced cooling losses: the CD (center) and the dense part of the shell (bottom right). The cooling-heating phase space plot (Fig. 6.11) shows seven distinct snapshots of the model:

1. 3.457 Myr at the start of the WR phase the shell is pressure driven and we find cooling losses near the CD and in the shell.

2. The mass loss rate of the winds peaks at 4.4975 Myr and leads to dense, cooling gas near the feedback region.

3. At the end of the wind phase at 4.859 Myr radiative cooling is effective in the shell and near the CD.

4. As soon as the SN explosion has taken place (4.8695 Myr), again dense material is found near the feedback region.

5. At the time of maximal luminosity (4.8695 Myr), when the SN blast wave hits the cavity wall, cooling near the CD is very important.

6. When the model has transited to the momentum driven phase (9.7595 Myr) cooling in the dense, swept-up shell dominates. At this stage models of different spatial resolution converge.

7. Also at the end of the simulation (23.3975 Myr), when the shell has decelerated to the ambient sound speed, cooling is only effective in the dense shell.

Comparing cooling losses in these snapshots shows that the energy losses in or near the CD cell get unimportant after the end of the pressure driven phase. At this point models with different spatial resolutions start to converge.

Artificial mixing across the contact discontinuity

Numerical simulations find large radiative cooling losses near the contact discontinuity (CD) separating the dilute, extremely hot shocked wind gas and the dense swept-up medium (see also Sect. 6.2.3). In the literature this is sometimes called "catastrophic cooling" (Tenorio-Tagle et al., 1990; Smith and Rosen, 2003). These losses arise, because the code mixes two media, which should be separated by a CD and the cell with the mixture of the two gas phases efficiently cools, acting like a valve, considerably reducing the feedback energy efficiency. If the mixing scale of the gas phases (see Sect. 2.2) is not resolved numerically, this process could lead to artificially high radiative losses.

In this work we also test the importance of this effect by regulating the radiative energy loss of the critical cell near the CD, which acts as the dominant energy sink. Numerically there are basically two strategies to prevent extreme cooling losses in cells at a CD where the two media mix:

(1) Strictly enforcing the separation of these two gas phases: The simplest way to avoid cooling losses in the hot, dilute cells in which shell material and wind material can be found, is to increase the density threshold of the cooling function. Our cooling function is tabulated for number densities $n_{\rm H} > 0.01\,{\rm cm}^{-3}$. To avoid cooling losses at the CD, in the models with "density thresholds" radiative cooling is switched off if the cell's density is below a times the ambient density ρ_0. For example, in runs with $a = 1$ radiative cooling is switched off at all densities below the ambient density. By doing this, we mimic a sub-grid model with two nicely separated ISM components in the cell: the gas is either to cold or not dense enough to cool and no strongly cooling intermediate gas phase is produced. Or in other words, at densities below $a\rho_0$ the simulation becomes adiabatic.

(2) Postulating a strong mixing process, which smears out the temperature and density slope near the CD: This leads to low temperatures in regions, which are dense enough to cool. Efficiently mixing gas across the CD can be achieved e.g. via heat conduction (which we show to be too inefficient in Fig. 6.9), hydrodynamic instabilities or other mixing processes (e.g. molecular diffusion through the shell walls or ablation of clouds and clumps). The radiative cooling losses are a function of temperature and density. Lowering the density and the temperature by enhancing the mixing at the discontinuity can limit the energy losses via radiative cooling by producing cells, which are already too cold to cool efficiently.

6.2.4 Convergence of the retained kinetic energy

In our simulations cooling losses occur in two distinct regions of the models: in the dense, swept-up shell and near the CD (see Sect. 6.2.3). Our models converge if cooling losses in the newly swept-up medium dominate. This is the case in momentum driven bubbles (i.e. in all our models for SNe without progenitor-winds and at late phases of the other models), whereas our models can not converge when the cooling losses caused by mixing across the CD dominate in the pressure driven bubbles (e.g. during the wind phases and in the early post-SN phase). This convergence issue can, however, be solved by working out, on which scales the ISM mixes (Sect. 2.2.2 to 2.2.4). The spatial resolution of the numerical simulation governs the mixing of gas phases across the CD (the PLUTO code allows for one gas phase per cell) and thus implies a length scale, on which diffusive processes occur. Thus, the feedback energy efficiencies of our simulations with different resolutions can be interpreted as solutions for different efficiencies and scale lengths of turbulent diffusion. If the assumed length scale of the mixing processes is below our resolution, the efficiencies in Tab. 6.2 are lower limits.

Spatial resolution

Tab. 6.2 and the top right panel of Fig. 6.9 show that resampling the wind bubble to twice the resolution at the SN leads to an increase of the retained kinetic energy. If the model is resampled as soon as the oscillations (back and forth conversion of energy) due to the evanescent SN wave are damped away (at 6 Myr) to twice the resolution, also a higher efficiency is found in the rescaled model. Restarting at the end of pressure driven phase (9 Myr) with twice the resolution does not change the efficiency. This is consistent with the SN model without wind, which retained 0.11% of the inserted energy when the shell speed reached the ambient sound speed independently of the resolution.
In simulations with lower spatial resolutions than the models shown in Tab. 6.2, the swept-up shell becomes unresolved. Thus, increasing the resolution reduces the feedback energy efficiency, since it causes higher peak densities in the swept-up shells and the cooling losses rise with number density squared. At higher spatial resolutions mixing across the CD starts to produce strongly cooling cells: such strongly cooling cells arise if enough energy from the hot phase is mixed with enough density from the cold phase. At low resolution this occurs only every n-th time step. In the models shown in Tab. 6.2 strongly cooling cells are created at every time-step. If this is the case, the feedback energy efficiency starts to rise again with increasing resolution.

The cooling losses are proportional to the volume of the cooling region, time and density squared. We did not find a dependence of the feedback energy efficiencies on the CFL factors used (see "temporal resolution" paragraph below). The density in the mixing cell also does not depend strongly on the resolution, since the flux of hot gas into the CD cell is set by the shell velocity. Due to the CFL using smaller cell sizes also reduces the time step. In other words, $\frac{\Delta t}{\Delta x}$ is set by the peak velocity and the CFL. The number density of the gas mixture in the strongly cooling cell is given by: $n_{\mathrm{average}} = n_{\mathrm{hot}} v_{shell} \frac{\Delta t}{\Delta x} + \left(1 - v_{shell} \frac{\Delta t}{\Delta x}\right) n_{\mathrm{cold}}$ or $n_{\mathrm{average}} = \left(n_{\mathrm{hot}} - n_{\mathrm{cold}}\right) v_{shell} \frac{\Delta t}{\Delta x} + n_{\mathrm{cold}}$. The shell velocity to peak velocity ratio ($v_{shell} \frac{\Delta t}{\Delta x}$) differs less than 10% between the models with different resolutions. Our simulations showed lower densities in the strongly cooling cell for higher resolutions.

The reason for the increase of the feedback energy efficiency with spatial resolution is the reduction of the strongly cooling zone's volume. The volume of this one-cell-wide shell located close to the CD is affected by two counteracting effects: (1) changing the (cell-) width of a shell reduces the volume by a factor $\frac{\Delta x_1}{\Delta x_2}$ (i.e. 0.5 for doubling the cell number) but (2) at the same simulation-time, simulations with higher resolution and thus higher efficiency have already produced larger bubbles. This makes the volume ratio at the same simulation-time larger than $\frac{\Delta x_1}{\Delta x_2}$ i.e. > 0.5 for doubling the cell number.

From Tab. 6.2 we find that the retained kinetic energy of the shell when the shell has been decelerated to the sound speed seems to rise like $E_0 \times (1.3)^n$ for $a = 0$ and like $E_0 \times (1.1)^n$ for $a = 1$, where n is the number of cells per unit length and E_0 is a proportionality constant. The lower factor for $a = 1$ strengthens the assumption that this treatment of the CD reduces the importance of radiative losses near the CD in this model.
The comparison of these factors and the fact that resampling the model when the losses in the newly swept-up medium start to dominate to higher resolution does not influence the feedback energy efficiency show that the treatment of the CD and the assumed mixing processes are most important during the wind phases and the pressure driven post-SN phase.
To avoid energy losses at the reverse shock, the spatial interpolation scheme should be as sharp as possible in this region. The scheme "WEN03", which is suited for smooth data, led to a lower efficiency and stronger oscillations in the shocked wind region than the "LINEAR" scheme. Also "WEN03" produces acell with a sharp local density minimum on the inside of the shell, which leads to code crashes.

Temporal resolution

In our simulations the time-step is limited by the CFL condition (Sect. 3.3), which ensures that gas cannot travel more than a cell length per time-step. Thus, we can reduce the time-step via reducing the cell size $\left(\frac{\Delta x}{2}\right)$ or via reducing the factor in the CFL condition $\left(\frac{\mathrm{CFL}}{2}\right)$. I.e. the time-step for a simulation with CFL=0.3 is similar to the time-step in a simulation with CFL=0.6 and twice the number of cells per parsec. The time-steps of these two simulations differ a little, since variations in the velocities caused by the spatial resolution are a second order effect on the time-step size. The maximal velocities at a given time in the different simulations vary by less than 10%. The location of the cell, which limits the time-step depends on the evolution of the model: after 1 Myr the gas velocity in the outermost cell of the free streaming wind region limits the time-step, whereas after 4 Myr the sound speed in the shocked wind region near the bubble wall limits the time-step size.
The two-shock Riemann solver's efficiency is independent of the time-step size (varied via the CFL and by changing the time-marching algorithm from Runga-Kutta II to Runga-Kutta III), whereas the Roe solver gets more efficient for larger time-steps, since the energy loss at the reverse shock occurs less often.

Riemann solver

In the simulations[4] with initial densities of $\rho_0 = 2.2 \times 10^{-22}$ g cm^{-3}, pressures of $p_0 = 1.47683 \times 10^{-12}$ erg cm^{-3}, resolutions of $\Delta x = 0.032$ pc and extreme mass loss (500 $M_\odot$, which is much too high but was used for tests of the kinetic energy fraction) in the SN, the two-shock solver

[4]This is a different set from the simulations in Tab. 6.2

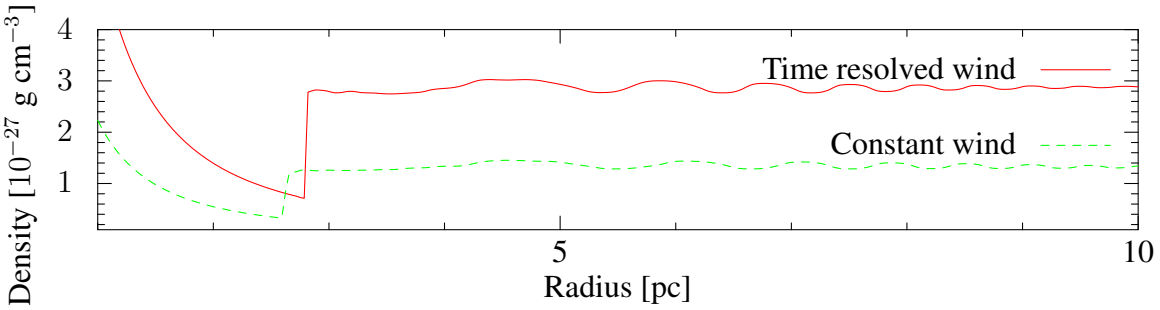


Figure 6.12: Oscillations near the reverse shock after 2 Myr in simulations using the two-shock solver.

(1.8×10^{49} erg when the shell speed reaches the sound speed) is more efficient than the Roe solver (1.5×10^{49} erg) and less efficient than the HLLC solver (2.2×10^{49} erg). This is the expected behavior, since the HLLC solver is the most diffusive of the three solvers and hence the density and temperature slopes at the contact discontinuity are shallower and thus the temperature in the first cell, which is dense enough to cool is smaller than in simulations with the two-shock solver. On the other hand, the Roe solver has problems with energy losses at the slowly moving reverse shock. This can be seen as damped oscillations in the shocked wind.
Actually all solvers produce oscillations inside the shocked wind region. A test with a constant wind showed that these oscillations are not caused by changes of the wind power, since they are also observed in a simulation with a constant wind (Fig. 6.12).

Influence of the feedback region size

The standard radius of the feedback regions in our 1D simulations is $r_\mathrm{f} = 0.32$ pc (Sect. 6.1.3).
To test the influence of the number of cells in the feedback region onto the energy content of the simulation, models with different resolutions (Δx from 0.008 pc to 0.032 pc) and diameters of the feedback region (r_f from 0.32 pc to 0.64 pc) were compared.
Also these models follow the general trend that simulations with higher spatial resolution find higher feedback energy efficiencies. Comparing the free streaming region to the solution of Chevalier and Clegg (1985) (see Sect. 4.4.3) showed good agreement for all models: The density profile was $\sim \frac{1}{30x^2}$ for all Δx and all r_f. Also the kinetic energy profiles for all Δx and all r_f were similar to those in Chevalier and Clegg (1985). Since the pressure in the top hat distribution in the feedback region is proportional to r_f^{-2}, the pressure is larger for larger r_f. All models showed a decay like $p \propto x^{-10/3}$, as expected.
The kinetic and thermal energy increase starts later for $\Delta x = 0.016$ pc and $r_\mathrm{f} = 0.64$ pc than for $r_\mathrm{f} = 0.32$ pc at the same resolution, since the initial top hat structure has to evolve into a wind structure, which takes longer for larger regions. The energy uptake rate is the same. As a result increasing r_f leads to slightly smaller bubbles. However, if the spatial resolution is decreased to $\Delta x = 0.032$ pc, the energy increase also starts later for larger r_f but after 0.1 Myr the energy uptake rate becomes higher for larger r_f, leading to larger bubbles for larger r_f. Doubling the feedback region radius thus led to an increased feedback energy efficiency for the lowest resolution. For $\Delta x = 0.016$ pc, however, the region diameter did not change the efficiency any more. Strangely for $\Delta x = 0.032$ pc the radiative losses (Λ) for smaller feedback regions ($r_\mathrm{f} = 0.32$ pc) are smaller than for larger feedback regions ($r_\mathrm{f} = 0.64$ pc), but less energy is stored in the simulation. The

time-step size in the early phases is smaller for smaller feedback regions, since outermost free streaming cell limits the time-step size.

6.2.5 Retained kinetic energy

The kinetic feedback energy efficiencies listed in Tab. 6.2 were evaluated at the moment, when the densest cell was decelerated to the ambient sound speed. This is also the time when the lines in Fig. 6.9, showing the time evolution of the retained kinetic energy, end. In the left upper panel of Fig. 6.9 the reference models without SN explosions or without winds are shown. The $60\ M_\odot$ model explodes in a SN after 4.8915 Myr. Simulations of models without wind phases are started at this time. The dependence of the models on the resolution during the pressure driven phase (right upper panel, see discussion in Sect. 6.2.4) leads to more efficient stellar feedback in higher resolved simulations. Also rescaling directly after the wind phase leads to an increased efficiency, whereas rescaling during the momentum driven phase (~ 9 Myr) does not change the efficiency (not shown in the plot, since the lines would be on top of each other). In the lower left panel the CD is artificially enforced via a. The dependence on the resolution in these models is less pronounced than in the standard case (right upper panel) but still exists, since the treatment with a reduces the losses near the CD but cannot prevent mixing of the two gas phases. The right lower panel shows the second approach to limit the losses near the CD: A mixing process smears out the CD and thus prevents that high temperature gas mixes with dense gas at the CD (by producing intermediate temperature gas and intermediate density gas). Thermal conduction leads to a 10% ($\Delta x = 0.032$ pc) or 18% ($\Delta x = 0.016$ pc) lower efficiencies. In this panel of Fig. 6.9 and 6.10 we also show a 14 orders of magnitude higher diffusion coefficient to mimic a very efficient mixing process. This model is converged for all resolutions. In Fig. 6.10 the retained kinetic energy of all these models is depicted as a function of the shell velocity. The phase, when the shell velocity has decreased to the ambient sound speed occurs later, at larger radii and at higher kinetic energies for higher a and higher resolutions.

The influence of a

If radiative cooling is applied for all densities in the cooling table ($a = 0$, Tab. 6.2, Fig. 6.9 right upper panel), the kinetic energy at the end of the wind phase is a factor 1.3 higher in simulations with a cell size of $\Delta x = 0.008$ pc than in simulations with $\Delta x = 0.016$ pc. The latter simulation's kinetic energy during the wind phase is a factor 1.2 higher than in a simulation with $\Delta x = 0.032$ pc. The feedback energy efficiency when the bubble shell has decelerated to the ambient sound speed rises by a factor 1.3 if the number of cells is doubled.
If there is no density threshold for radiative cooling ($a = 0$), also the SN shell can cool. More than 70% of the energy is lost via radiative cooling when the SN blast hits the bubble wall. All the kinetic energy in the reflected wave is lost at the origin, since the reflected wave sweeps up the gas and creates an efficiently cooling density peak at the origin. Again losses are higher in simulations with larger cells.
Limiting the mixing processes across the CD by applying radiative cooling only to cells with densities above the ambient density, leads to a feedback energy efficiency of approximately 7% for a cell size of $\Delta x = 0.032$ pc. If all cells with densities below the ambient density are considered to contain not radiatively cooling hot gas ($a = 1.0$, Tab. 6.2, Fig. 6.9 left lower panel), halving the cell size increases the kinetic energy when the bubble shell has decelerated to the ambient sound speed

or the kinetic energy at the end of the wind phase by a factor of 1.1. If the cellsize is reduced, the oscillations between kinetic and thermal energy caused by the SN are less damped. The radiative energy losses are largest when thermal energy is converted to kinetic energy (every second time, when the wave enhances the pressure near the bubble wall, strong radiative cooling losses arise in cells, which are dense and hot enough to cool. Since no density peak (as high as the ambient medium) is found at the origin no additional losses occur when the SN wave is reflected at the origin. The losses are larger if the cells are larger).

Wind-only models

Comparing the kinetic energies at the end of the wind phase of models that differ only by spatial resolution (Tab. 6.2) show that we find an increase of the retained kinetic energy by a factor ~ 1.1 for models with $a = 1$ and a factor ~ 1.3 for models with $a = 0$. For the model without SN and $a = 0$, which was resampled at the end of the wind phase, we find an increase of the retained kinetic energy by a factor ~ 1.1 against the not resampled model. No energy is added to this model after resampling, but the higher resolved model can retain more kinetic energy, since it loses less energy at the CD.

The influence of mixing processes

The dependence of the feedback energy efficiency on the spatial resolution decreases, if thermal conduction is taken into account. For extreme conduction the differences between the simulations with different resolutions essentially vanish. As mentioned in Sect. 6.2.4, our spatial resolution defines a scale length on which gases are mixed with 100% efficiency. Since our resolution has reached or even gone below the proposed length scale of turbulent mixing (Sect. 2.2.4) we conclude that the dependence of the feedback energy efficiency on the spatial resolution depicts the dependence of the radiative losses on the efficiency and the length scale of turbulent mixing across the CD.

6.3 Conclusions

We investigated the efficiency of stellar energy deposition in the ISM. For this study we compared the feedback energy efficiency of SNe in different environments. Our main results are:

- If a simulation with 100 particles per cm^3 refers to Thornton et al. (1998) and uses a feedback energy efficiency of 10% as a sub-grid model, a time-step of 33 kyr has to be resolved. A short time later the efficiency drops far below 10% (Fig. 6.2 and 6.3).

- Without the stellar wind of the progenitor star, the feedback energy efficiency of a massive star, which is placed in a dense medium, is much (here a factor 6) smaller than if the wind is taken into account (Tab. 6.2).

- The cumulative feedback energy of the stellar wind of a 60 $M_\odot$ star is $2.34\,E_{\rm SN}$. The impact of the stellar wind can be seen from a comparison between a model with no SN blast at the end of the wind phase and a model with both progenitor wind and SN blast. The energy difference when the shell reaches the sound speed (Tab. 6.2) is $2.13 \times 10^{48}\,\mathrm{erg}$ in a model without SN compared to $2.71 \times 10^{48}\,\mathrm{erg}$ in a model with SN and wind. This differs from

the ratio of the total energy inputs (2.34×10^{51} erg and 3.34×10^{51} erg). Thus, steady stellar feedback is more efficient than a blast.

- The feedback energy efficiency of a constant wind with the same net energy input is slightly higher than for the time-resolved wind (Tab. 6.2). Averaging the WR phase over the whole stellar lifetime makes the constant wind stronger than the time resolved wind in early phases and allows it to create a larger bubble at early times, which serves as a pressure reservoir for the bubble expansion later on. At the time when SN explosion happens, the bubble size and the retained kinetic energy of the constant wind model are larger than in the time resolved model, whereas the thermal energy is smaller, since the time resolved models boost the thermal energy during the WR phase directly before the SN.

- The time of maximal luminosity (t_0, defined in Sect. 6.1.1) occurs later, if stellar wind bubbles are taken into account. In this case, the blast expands adiabatically until it impacts onto the cavity wall. Subsequently, the SN blast wave bounces inside the bubble and as a result the luminosity peaks are periodic events and occur whenever the SN shock-wave hits the cavity wall (more precisely, it does not directly hit the cavity wall but compress the wind gas in front of the cavity wall) and kinetic energy is converted to thermal energy (and vice versa). The losses show a double peak at times when the conversion rates are largest.

- Mixing processes across the CD are important during pressure driven phases. In these phase the resolution mimics the scale of mixing and thus has an effect on the feedback energy efficiency. In the subsequent momentum driven phase radiative cooling in the swept-up, compressed and thus heated medium is the dominant energy sink.

- Comparing the constant wind models at different resolutions (which mimic the length scale of the mixing processes in the ISM) shows that the 0.032 pc model has a higher efficiency than expected. Low resolution models can find a higher efficiency, if they underestimate the density in the shell. In this case the efficiently cooling temperature-density combination is not found at every time-step in the 0.032 pc model, whereas later on this gas phase is always present. In higher resolved models the efficiently cooling layer near the CD has a smaller volume: At the same time of the simulation it is found at larger radii in higher resolved simulations but it is only a single cell wide. Simulations with a resolution of 0.001 pc showed cooling losses of the same order of magnitude in the compressed swept-up medium and near the CD. At even higher resolutions the cooling layer will at some point become irrelevant.

- The feedback energy efficiency in 1D simulation is expected to be an upper limit for multi-dimensional simulations, since (non-radial) instabilities, which arise in more dimensions, increase the surface of the CD and enhance mixing between the hot and cold gas phase. In our work these mixing processes are treated indirectly via the mixing length-scale (i.e. by the resolution or via diffusion coefficients).

- During the wind phase the density threshold in the cooling function (e.g. $a = 1$) reduces the dependence of the feedback energy efficiency on the resolution (Tab. 6.2). However, the differences between the feedback energy efficiencies for different resolutions at the end of the simulations are not significantly reduced if the threshold $a = 1$ is used instead of $a = 0$.

- If the coefficient κ of heat conduction is strongly increased, the models converge, since the gradients at the CD, which were sensitive to spatial resolution, get smeared out. However, the total feedback energy efficiency is drastically lowered by this treatment.

Chapter 7

3D: Porosity and depth of embedding

With our 3D simulations, we study the influences of the position of the massive stars inside the GMCs and the "porosity" of the cloud onto the feedback energy efficiency. Porosity in this context describes the sum of the cross-section areas of all holes in the GMC allowing stellar feedback material to escape from the GMC into the warm phase of the ISM (phases of the ISM are discussed in Sect. 2.1). We now move on from infinite clouds to semi infinite clouds.

7.1 Setup of the 3D models

The hydrodynamic simulations discussed in this section were carried out with the Eulerian grid code RAMSES (Teyssier, 2002, discussed in Sect. 5.1.2) on a Cartesian mesh. The simulations take advantage of AMR and reach a resolution of 0.13 pc. The grid is refined near large pressure and density gradients. If nothing other is mentioned, a HLLC solver and MinMod flux limiting are used. Our modifications of the code (cooling-heating equilibrium and stellar feedback) are described in Sect. 5.2.3. Tests showing that our numerical solution for stellar feedback in an infinite homogeneous cloud without radiative cooling approaches the wind theory of Castor et al. (1975), which describes an idealized spherically symmetric stellar wind, are presented in Sect. 4.4.1.

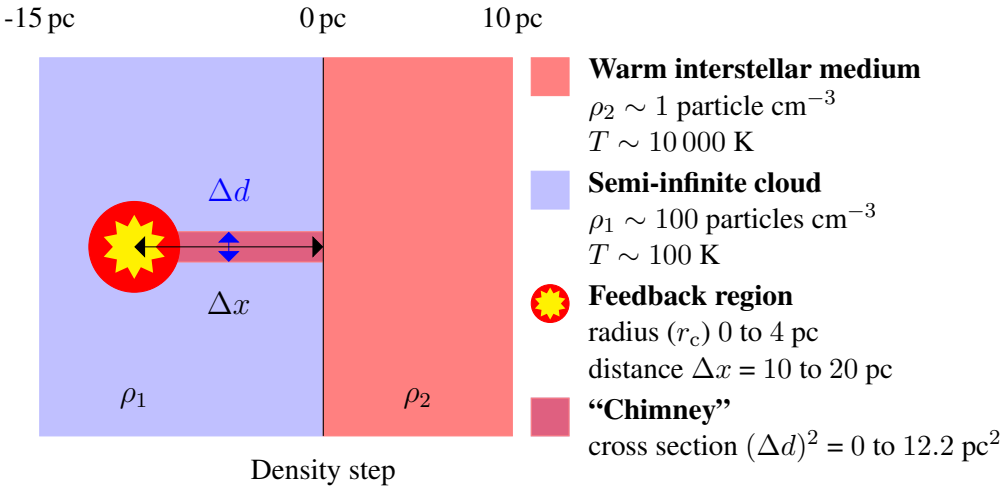


Figure 7.1: Components of the toy model: The feedback region is immersed in a semi-infinite cloud. It is connected to the ambient ISM by a "chimney" in the cloud.

The basic setup for the models discussed in this chapter is sketched in Fig. 7.1. The feedback region can be placed in a pre-existing spherical cavity mimicking a Strömgren sphere[1]. This helps to avoid losing the newly inserted energy in this zone immediately. Alternatively we can also turn off energy losses via radiative cooling in the feedback region. But, since the ionizing radiation of the massive star will create a Strömgren sphere, the assumption of a small pre-existing cavity is less artificial than a non-cooling dense part of the cloud. As a reference we also present models without pre-existing cavities. In these models, cooling losses can occur inside the feedback region. The smallest initial cavity in our set of models is approximately one cell larger than the feedback region ($r_\mathrm{c} = 0.64$ pc, $r_\mathrm{fb} = 0.49$ pc). That it cannot be exactly one cell larger is an implication of the implementation of this spherical region on a Cartesian grid (see Sect. 5.2.3). The largest pre-existing cavities we tested have a radius of $r_\mathrm{c} = 4$ pc. For the chosen cloud density of $100\,m_\mathrm{H}$ cm^{-3}, this bubble radius would be well inside the Strömgren radius of a 60 $M_\odot$ star. For example Sternberg et al. (2003, Fig. 5 and Tab. 1) find 3×10^{49} ionizing photons per second for an O5 main sequence star with similar mass, luminosity and effective temperature as found in the initial phases of the stellar evolution model we use for this study (Ekström et al., 2012, rotating 60 $M_\odot$ star). For a Hydrogen number density of $n_\mathrm{H} \sim 71$ cm^{-3}, this leads to a Strömgren radius of ~ 5.44 pc.
The pre-existing cavity contains gas with the same density and temperature as the ambient medium and can be connected to the ambient medium via a cuboidal "chimney". Since no friction or viscosity are taken into account, the shape of the "chimney" is irrelevant and even if there is a single hole or if there are several holes is a second order effect. The leading order term is the total cross-sectional area of all holes. Therefore, our setup uses a single "chimney" with a quadratic cross-section, since this shape of the "chimney" is more convenient than a cylindrical hole on a Cartesian grid.
The GMC gas is assumed to have a proto-solar chemical composition according to Lodders (2003) to calculate cooling and heating. This leads to a hydrogen mass fraction of $X = 0.711$. The assumed density of $100\,m_\mathrm{H}$ cm^{-3} $= 1.66 \times 10^{-22}$ g cm^{-3} thus corresponds to $n_\mathrm{H} \sim 71$ cm^{-3}, which is slightly below the densities in the 1D models[2]. However, also this density leads to a cooling-heating model dependent equilibrium temperature in the order of 100 K (see Fig. 2.1). Our models now also contain a warm dilute phase of the ISM which is in pressure equilibrium with the dense phase. We use a density of 1.66×10^{-24} g cm^{-3} and a temperature in the order of 10 000 K. Both phases are in cooling-heating equilibrium.

7.2 Grid of models

In the sensitivity analysis the impact of (1) the distance of the feedback region from the cloud edge, mimicking the position of the OB associations inside a GMC and (2) the "porosity" via the cross section of the "chimney", parametrizing the density structure of the cold ISM, onto the feedback energy efficiency are studied.
A typical setup of our grid of models is sketched in Fig. 7.1: The distance Δx of a 60 $M_\odot$ star from the edge of a semi-infinite cloud is either 10 pc or 20 pc. As described in Sect. 5.2, the best

[1] The gas in a Strömgren sphere would also have a temperature of $\sim 10\,000$ K, however, the gas density would be higher than in the 10 000 K gas phase we use for the cavity. We fill the pre-existing cavity with ambient medium, since only in this way we can set up static IC.

[2] in 1D: 2.2×10^{-22} g cm^{-3}, or 133 m_H cm^{-3} with a molar mass of 0.5 or 1.33 g mol^{-1}.
in 3D: 1.66×10^{-22} g cm^{-3}, Hydrogen mass fraction $X = 1$ or $n < 100$ Lodders (2003): molar mass $X = 0.7110$, $Y = 0.2741$, and $Z = 0.0149$.

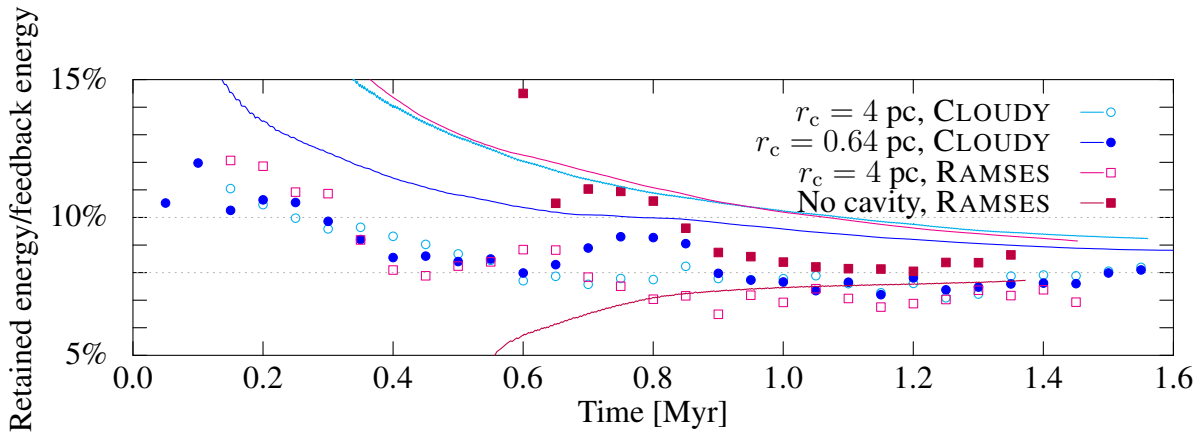
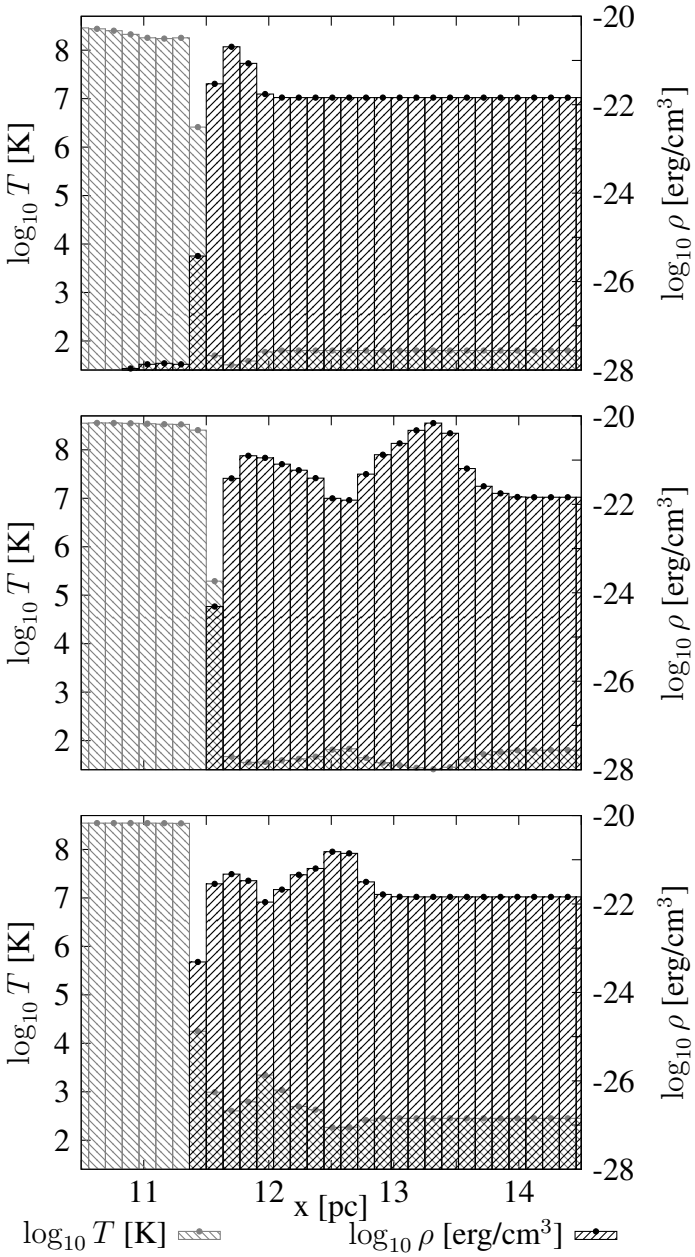


Figure 7.2: Cooling-heating model dependence of the feedback energy efficiency of the wind of a $60\ M_\odot$ star inside a homogeneous cloud with a density of 1.66×10^{-22} g cm^{-3}. The graph shows the fraction of retained wind energy (in thermal and kinetic energy) during 0.05 Myr (symbols) and the total energy injection efficiency computed at each coarse grid time step (lines). The models have a resolution of 0.13 pc and differ in the cooling-heating model and in the size of a pre-existing cavity at the onset of the wind. The model without cavity can also be found in Fig. 7.7(a).

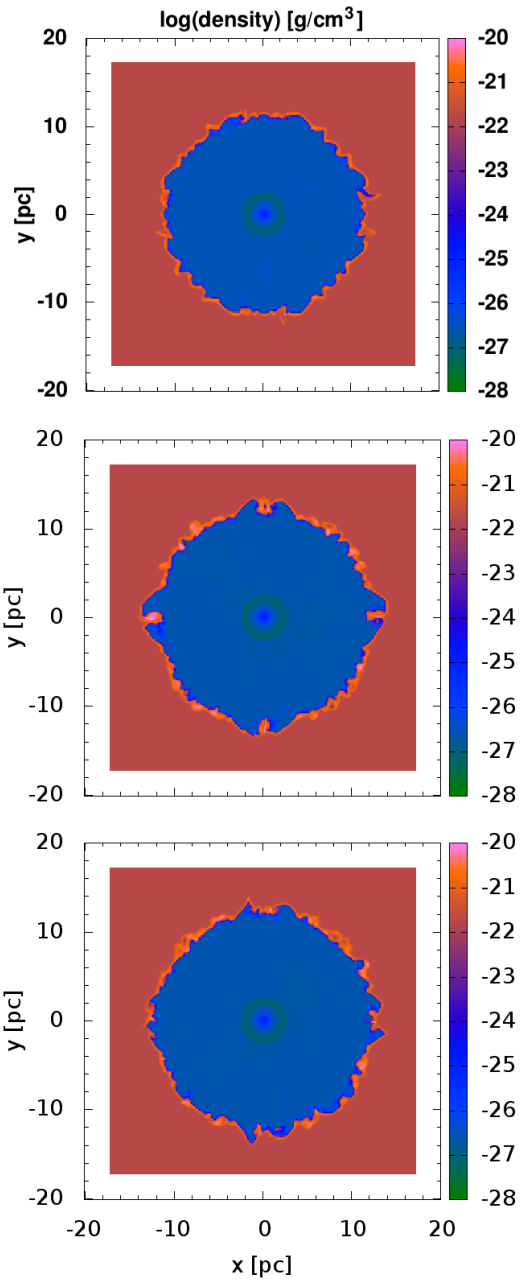
radius of the feedback region (0.5 pc) follows from the chosen resolution.
The tested cross sections of the "chimneys" $(\Delta d)^2$ are 12.2 pc^2, 3.5 pc^2 and 1.2 pc^2. The initial cavity in the cold phase is either absent or has a radius of 0.64 pc, which is slightly larger than the feedback region, or 4 pc, which is of the order of the initial Strömgren radius (see Sect. 7.1).
Presently, only the early wind phases have been modeled with a resolution much below the resolution in the 1D work. However, a comparison between models that differ only in the "chimney" size already led to a few interesting results, which will be discussed in the rest of this chapter. In our future work we plan to follow up with higher resolution 3D models that will be monitored until the shell has reached the ambient sound speed, as in the 1D work.

7.3 Impact of the cooling-heating model

As in our 1D models, our simulations use detailed cooling tables. We compare the results of two cooling models: (1) cooling tables extracted from CLOUDY and implemented in RAMSES by Eva Ntormousi as described in Ntormousi et al. (2011); (2) an artificial two-phase medium (see Sect. 2.2.6 and 5.2.3) based on the RAMSES cooling table. For the latter, the equilibrium temperature of the dense phase is higher than in the CLOUDY tables, as can be seen in Fig. 2.1. As the RAMSES cooling table does not create a two-phase medium, an artificially stable hot phase, which is in pressure equilibrium with the cold phase, is added. For both models artificially stable phases for the two gas phases in the initial conditions (IC) are implemented and can be switched on or off at compile-time. However, tests showed that the gas phases in the IC are close enough to stable phases in the CLOUDY cooling model that the presence or absence of the artificial equilibria for the IC did not have a huge impact on the simulation. Moreover, this artificial equilibrium is unstable: Tiny density changes, which can be created e.g. by sonic waves caused by stellar feedback, will make the gas temperature evolve towards the cooling-heating equilibrium curve value.



(a) Temperature and density distribution of the cells along the x-axis



(b) Cut through the density distribution

Figure 7.3: The simulations with an initial cavity (radius r_c) in a homogeneous cloud are shown after 1 Myr. Top: $r_c = 0.64$ pc, CLOUDY cooling (implemented in the same way as in Ntormousi et al., 2011), center: $r_c = 4$ pc, CLOUDY cooling, bottom: $r_c = 4$ pc, modified RAMSES cooling. The lower two plots exhibits a two peak shell structure in the swept-up shell, since the wind shell ran into the walls of dense gas surrounding the pre-existing cavity.

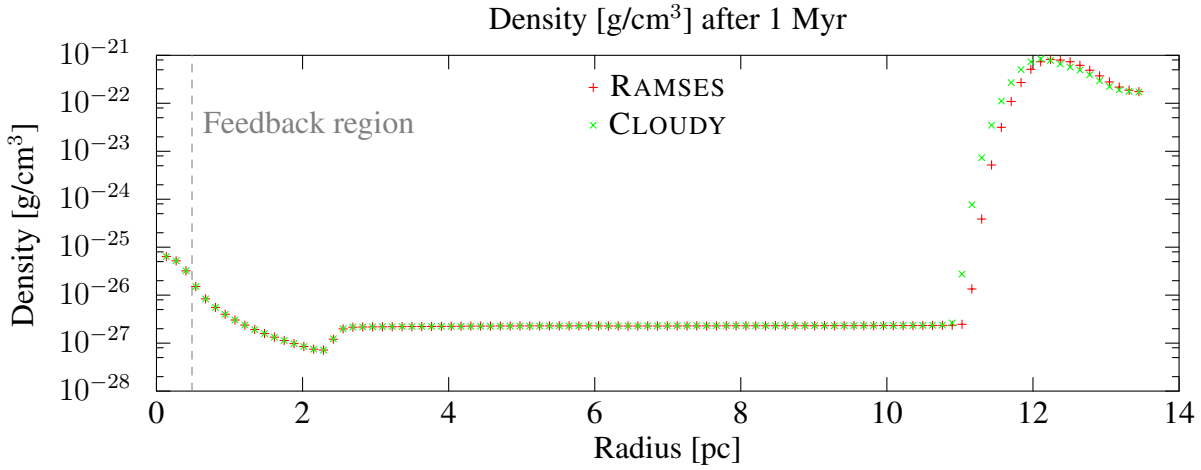


Figure 7.4: Averaged radial bins of the 3D simulations on a Cartesian grid. The plot compares simulations with an initial cavity radius of 4 pc and different cooling models. These simulations are also shown in Fig. 7.2 and 7.3. Despite the larger ambient pressure, the model with RAMSES cooling seems to produce slightly larger bubbles. Also the average pressure in the shell of the model with CLOUDY cooling is lower than in the model with RAMSES cooling (this can be inferred from Fig. 7.3).

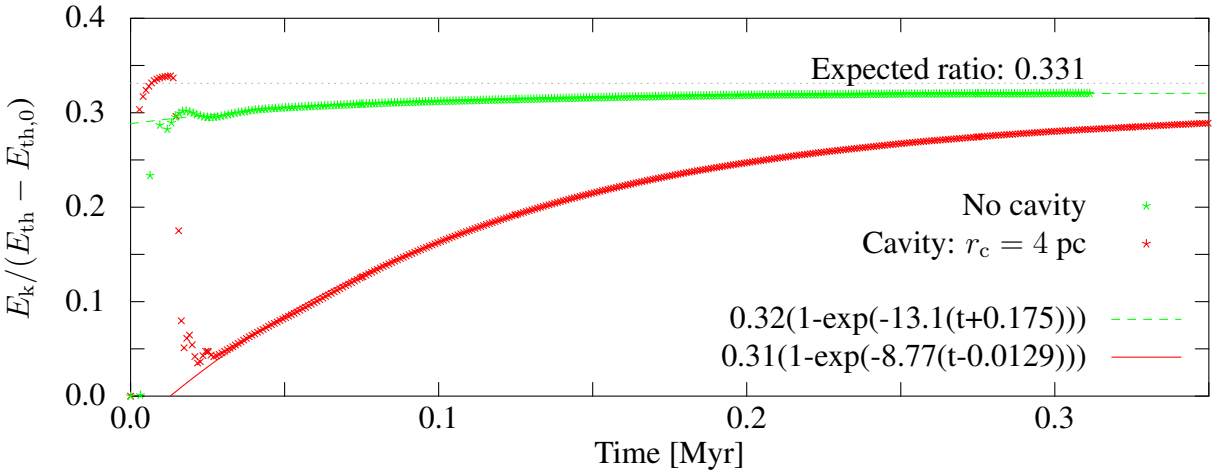


Figure 7.5: Numerical tests agree well with a kinetic energy fraction of 45.6% in the shell and a 5:6 energy ratio between bubble and shell. This leads to a total kinetic to thermal energy ratio of 0.331 as found in the simulations without cooling losses. Since the initial density in the feedback region is treated like wind gas, the kinetic energy fraction is initially overestimated. The red data shows the effect of the presence of a pre-existing cavity: the swept-up shell contains less mass and hence, less kinetic energy. This effect is also seen in the simulation without cavity (shown in green). Here it is caused by the mass of the feedback region ($r_{\rm fb} = 1.5 \times 10^{18}$ cm) ending up "on the wrong side of the contact discontinuity". The evolution of the energy ratios towards an equilibrium value was fitted with an increasing exponential decay form $(1-e^{-t})$. The presence of an initial cavity leads to an initially lower kinetic energy fraction, since (1) the bubble is larger due to the faster expansion in the initial cavity than a bubble forming in a homogeneous dense medium and (2) the swept-up mass at a swept-up shell radius $r_{\rm shell}$ is $(4\pi/3) r_{\rm shell}^3 \rho_{\rm average} = (4\pi/3)\left(r_{\rm shell}^3 \rho_{\rm cold} - (\rho_{\rm cold} - \rho_{\rm warm})\, r_{\rm c}^3\right)$.

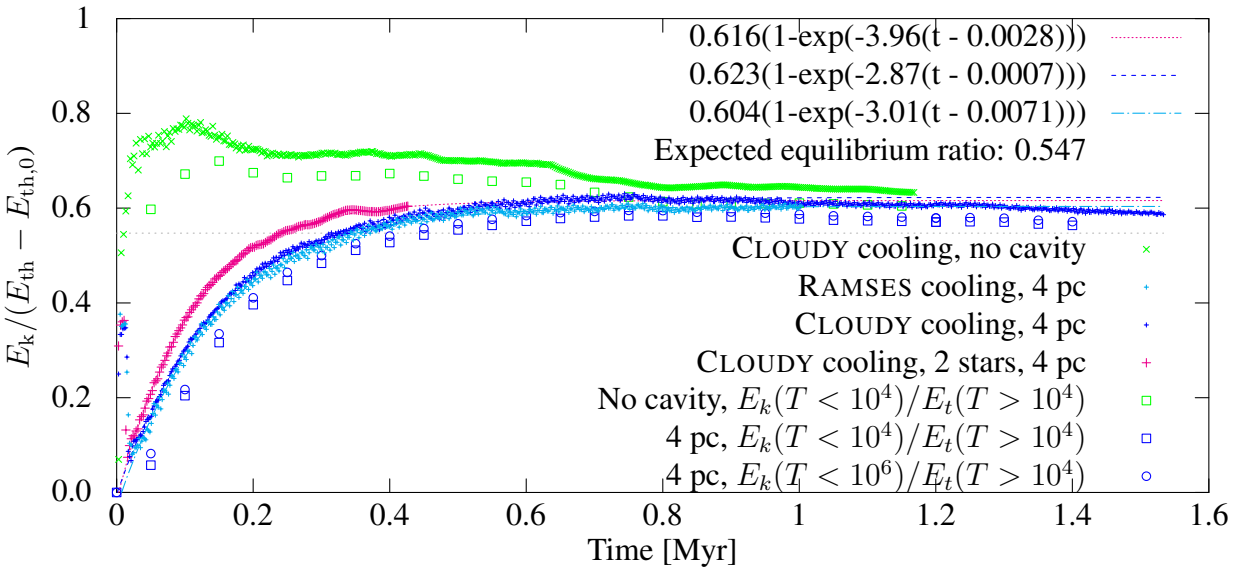


Figure 7.6: Without cooling, we expect a ratio of 0.547 between the kinetic energy in the shell and the thermal energy in the cavity from the Weaver et al. (1977) wind theory. The open symbols indicate the ratio of the kinetic energy of the cold medium ($T < 10^4$ K) to the thermal energy in the bubble ($T > 10^4$ K). The crosses use the difference between the thermal energy of the initial conditions and the current thermal energy as a proxy for the thermal energy of the bubble and the shell. In the plot the energy content after every 3rd coarse time step is shown. In our simulations the ratio of the total kinetic energy and the thermal energy increase is 0.6. The kinetic energy of the shell over the thermal energy of the hot medium is closer to the aforementioned expected ratio.

A comparison of simulations with the CLOUDY and RAMSES cooling-heating treatments (Fig. 7.2) shows that the feedback energy efficiency is of the same order for both cooling models. This is consistent with the findings in 1D (Fig. 6.6). Fig. 7.2 contains four models that differ in the cooling-heating model and/or in the initial cavity size. The simulation with modified RAMSES cooling and no pre-existing cavity shows very low feedback energy efficiencies until a wind bubble has been established (~ 0.6 Myr). Adding a small cavity (that might represent an initial Strömgren sphere) helps to reach a roughly constant energy uptake per time step (of $\sim 8\%$, shown with symbols) and the expected wind structure inside the bubble earlier. Comparing this plot to the 1D models in the last section (first 1.5 Myr in Fig. 6.9), one has to take the dependence of the retained energy on the resolution and the lower ambient density of the 3D models into account.
The dense swept-up shell and the ambient medium (if not artificially stabilized at the IC pressure and density) reach the temperature of the cooling-heating equilibrium (see Fig. 2.1 for the equilibrium), which is lower in the CLOUDY cooling prescription. Consequently, the cooling-heating implementation of Ntormousi et al. (2011) leads to a lower ambient pressure (without artificial equilibrium for the IC the pressure is 7.2×10^{-13} erg cm^{-3} instead of 3.2×10^{-12} erg cm^{-3}). After 1 Myr slightly larger bubble radii (Fig. 7.3) in the cut along the x-axis are observed in the simulation with CLOUDY cooling than in the run with the modified RAMSES cooling with an artificially stable second phase. The latter, however, leads to slightly larger averaged bubbles (Fig. 7.4).
The total feedback energy efficiency is set by the pressure in the hot wind-blown bubble. Without cooling the ratio between the energy in the cavity and the energy in the shell can be found from the pressure driven expansion (see Weaver et al., 1977, Sect. III and Sect. 4.4.2 in this thesis). Eq. 4.41

predicts for a constant wind (in 3D):

$$E_{\rm cavity+shell} = \epsilon L_{\rm w} t, \qquad E_{\rm cavity} = \epsilon \frac{5}{11} L_{\rm w} t, \qquad E_{\rm shell} = \epsilon \frac{6}{11} L_{\rm w} t$$

With the kinetic wind luminosity $L_{\rm w} = 0.5 \dot{M} v_\infty^2$ where v_∞ is the terminal wind velocity and $\dot{M}$ is the mass loss rate. ϵ is the feedback energy efficiency.
Weaver et al. (1977) predict that 40% of the energy in the swept-up shell ($E_{\rm shell}$) are kinetic energy. Actually checking the numerical integration described Weaver et al. (1977) with MATHEMATICA showed that 45.6% of the shell's energy is kinetic. The kinetic energy fraction of $E_{\rm cavity}$ is negligible. This leads to a total kinetic to thermal energy ratio of 0.33, which is in excellent agreement with our numerical tests (Fig. 7.5).

7.4 Impact of pre-existing cavities

Fig. 7.5 also illustrates the effect of an initial cavity onto the energy ratios. The size of the pre-existing cavity influences the early phase of the bubble expansion (~ 0.4 Myr): The wind bubble expansion slows down with increasing ambient density. In our setup, the wind bubble first sweeps up the lower density medium in the cavity. We find a fast expansion with almost no cooling losses before the cavity wall is reached (visible e.g. via the similarity of the leftmost red crosses in Fig. 7.5 and the leftmost blue crosses in Fig. 7.6). When the shell starts to sweep up the dense cloud material, the expansion slows down and the cooling losses rise. The mass in Eq. 4.38 can no longer use a constant ambient density and becomes $\rho_{\rm cold} V_\nu r_{\rm shell}^\nu - (\rho_{\rm cold} - \rho_{\rm warm}) V_\nu r_{\rm c}^\nu$. Eq. 4.40 holds only for $r_{\rm shell} \gg r_{\rm c} \sqrt[\nu]{1 - \frac{\rho_{\rm warm}}{\rho_{\rm cold}}}$. A larger exponent a in Eq. 4.40 would lead to a lower kinetic energy fraction in Eq. 4.41. This is also observed in our simulations (Fig. 7.5 and 7.6). The bubble evolution can be approximated with two asymptotic expansion laws. One of them describes the expansion before the shock impinges on the dense cloud material and the other one is recovered when the second term in the aforementioned swept-up mass becomes negligible. These laws can be obtained from the Weaver et al. (1977) wind theory (see Sect. 4.4.1). The transition between the bubble expansion in the cavity and the expansion into the homogeneous surrounding medium leads to a decay-law-like evolution of the energy ratios[3].
We will now assume that all thermal energy of the shell is lost via radiative cooling. Without cooling, the expected ratio between the kinetic energy of the cold gas and the thermal energy of the hot bubble is 0.547[4]. In Fig. 7.6 we see that the total kinetic energy in simulations where radiative cooling is taken into account contains a contribution from gas above 10^4 K (compare blue open squares to blue open circles). The crosses and the fits in Fig. 7.6 use the total kinetic energy – which is dominated by the cold phase (as the tests each 50 kyr show). The thermal energy is found from the difference between the total thermal energy and the total thermal energy of the initial conditions. It is thus lower than the thermal energy of the bubble and the shell, since the initial thermal energy in this zone is subtracted. Every 50 kyr we evaluated the energy of the bubble in detail. The open symbols in Fig. 7.6 show the ratio of kinetic energy of gas with temperatures

[3]The convergence of the energy ratios looks a bit like the temporal evolution until an equilibrium concentration of reactants in a second order chemical reaction is established. See e.g. "A Second-Order Chemical Reaction" from the Wolfram Demonstrations Project http://demonstrations.wolfram.com/ASecondOrderChemicalReaction/

[4]$6/5 \times 0.456 = 0.547$, where $6/5$ are taken from Eq. 4.41 and 45.6% were found via numerical integration of Fig. 4.13.

below 10^4 K and thermal energies of cells with temperatures above 10^4 K (i.e. we do not subtract the energy of the initial conditions here). 10^4 K is used as limiting temperature, since it is the initial temperature of the cavity.

If all kinetic energy is used, the ratio seems to converge to $E_{\mathrm{kin,shell}} : E_{\mathrm{therm}} \sim 0.6$. If we only take the kinetic energy of cold gas into account, the ratio comes closer to the expected ratio of 0.547. The initially higher ratio in simulations with pre-existing cavities is also influenced by the kinetic energy of the free streaming wind.

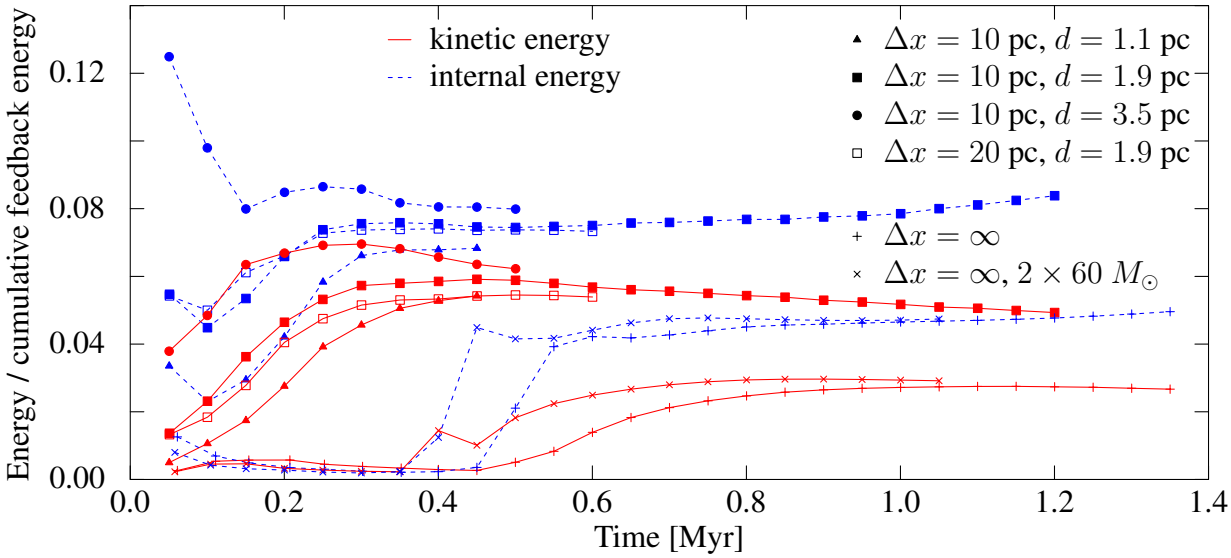
7.5 Homogeneous infinite cloud

As a limiting case of a very narrow and very long "chimney" a homogeneous infinite cloud is used. The total feedback energy efficiency in this setup is the lowest in the whole sample, since the gas cannot escape the cloud and the dense swept-up shell leads to large cooling losses (Fig. 7.7(a)). However, for all models with a resolution of 0.13 pc (purple lines in Fig. 7.7(b)) the kinetic feedback energy efficiency in the gas below the initial temperature in the warm medium (i.e. $< 10^4$ K) seems to converge to $\sim 3\%$ during the wind-phase. However, we will need more simulations to find out, whether this is a coincidence. Fig. 7.2 indicates that the radiative losses lead to a feedback energy efficiency factor $\epsilon \sim 8\%$ for the total retained energy $E(t) = \epsilon L_{\mathrm{wind}} t$. The largest radiative losses occur at the interface between the hot bubble and the shell (Fig. 7.8). When this constant energy uptake rate is observed, the dense shell cools to the temperature of the cooling-heating equilibrium for this density, as can be seen in Fig. 2.1, 7.3(a) and 7.9. Fig. 7.9 also shows a small indication of a temperature rise in the shock – however, our simulations do not resolve this feature. Increasing the significance by averaging over concentric shells does not help here, since we would average over the Vishniac instability and get a smeared out shell. As already discussed in Sect. 7.4 – a kinetic to internal energy ratio close to 0.547 (Fig. 7.6) is found. In accordance with the predictions of Weaver et al. (1977), most of the kinetic energy is found in the swept-up shell.

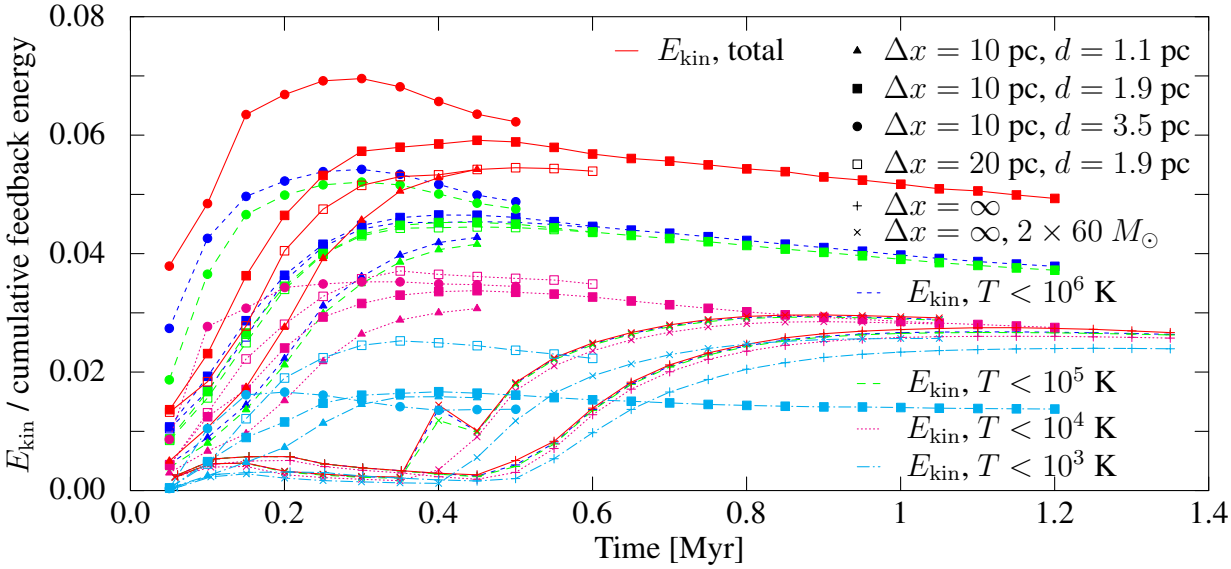
The time dependent cavity volume[5] (Fig. 7.10) exhibits the $V \propto t^{\frac{9}{5}}$ behavior expected for an almost constant wind from Eq. 4.42 and Castor et al. (1975, Eq. 6).

Since the IC of the models presented in Fig. 7.7 do not use an analytical sub-grid model for a wind shell in the feedback region or a pre-existing cavity to mimic a Strömgren sphere, these simulations show an artificially extended free expansion phase, caused by the homogeneous density in the feedback region: The simulation treats mass inside the feedback region like wind gas and the end of the free expansion phase is reached when the swept-up mass exceeds the wind mass. This artifact can be minimized by keeping the feedback region on the highest AMR level (which makes it smaller, since the optimal size of this region follows from an optimal number of grid cells therein) or by the assumption of a pre-existing cavity filled with ionized gas around the star. Placing the feedback region directly in the cold dense medium results in efficient cooling inside the bubble. If the feedback region is large enough to lead to oscillations inside it (i.e. in this case it contains more than the optimal number of cells), also the formation of tiny strongly cooling clumplets near the boundary of the feedback region is observed.

[5] We compare volumes instead of radii, since the simulations with "chimneys" are not spherically symmetric.

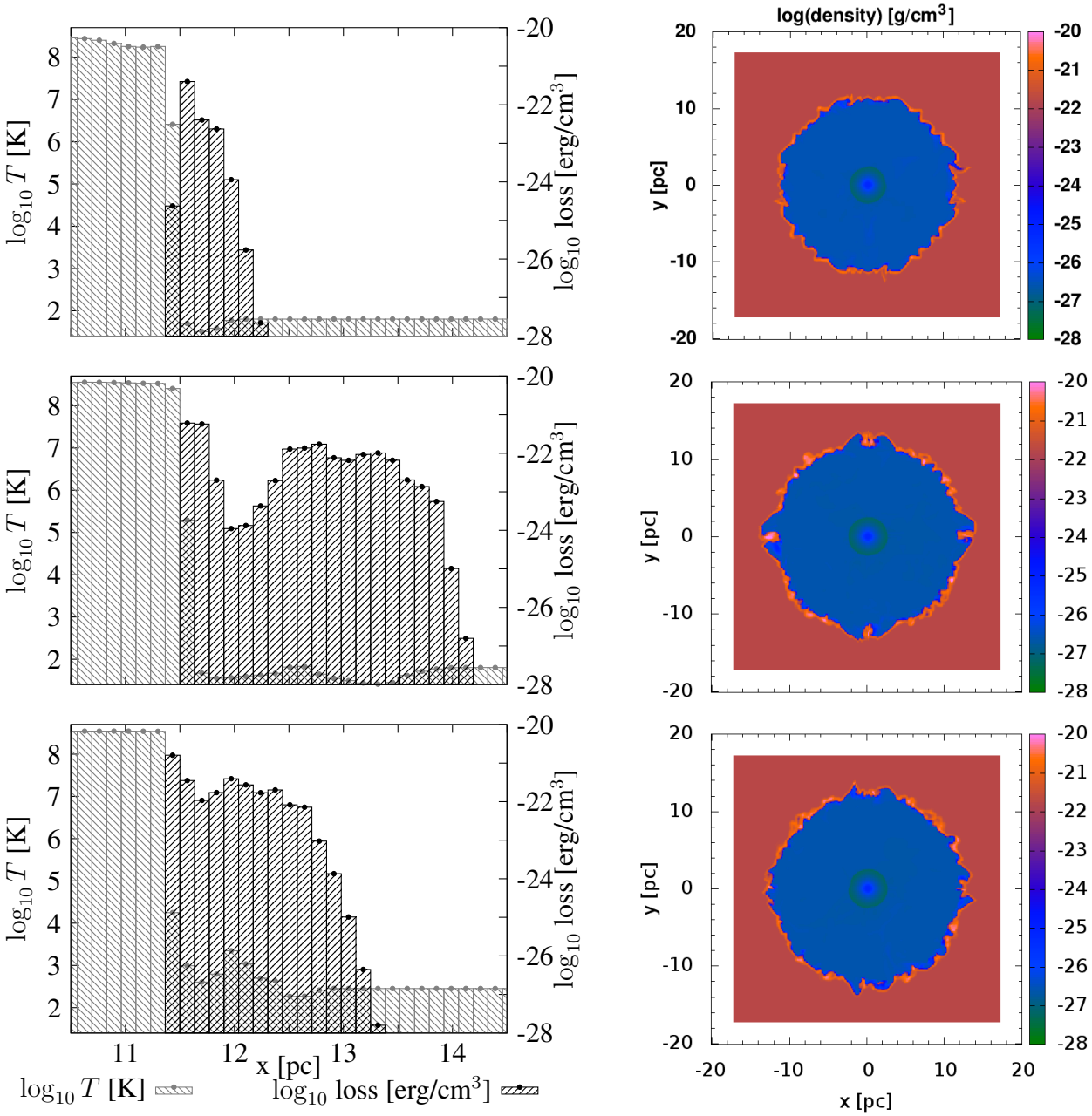


(a) Total feedback energy efficiency



(b) Kinetic feedback energy efficiency for different temperature ranges

Figure 7.7: Panel (a) shows the total feedback energy efficiency in six different setups. There is no pre-existing cavity in these simulations and all of them use the modified RAMSES cooling implementation. The efficiency of two 60 $M_\odot$ stars in an infinite cloud is similar to the efficiency of a single 60 $M_\odot$ star (crosses and pluses). A higher "porosity" increases the efficiency of the energy input: the retained energy rises with rising "chimney" diameter (filled symbols [triangle, box, circle]). This is expected, since a stellar wind bubble in the tenuous medium can grow faster and suffers less cooling losses than a stellar wind bubble in the dense medium. The cooling losses occur in the dense shell surrounding the not-cooling hot pressure reservoir. The length of the "chimney" only influences the kinetic energy via the size of the superbubble (filled and open boxes). In all models about 90% of the stellar feedback are immediately lost via radiative cooling. In contrast to panel (a) panel (b) shows only the kinetic feedback energy efficiency. The colors indicate the temperature range of the moving gas: total retained kinetic energy (red), kinetic energy in cells with temperatures below 10^3 K (light blue), 10^4 K (purple), 10^5 K (green) or 10^6 K (dark blue). All simulations seem to converge to $\sim 3\%$ retained kinetic energy in gas below 10^4 K.



(a) Temperature and cooling losses in cells along the x-axis

(b) Cut through the density distribution

Figure 7.8: Similar to Fig. 7.3(a). In (a) the cooling loss distribution is shown instead of the density. (b) shows the 2D density cut. Cooling was switched off at densities below the ambient medium density. The temperature in the bubble ($\sim 10^8$ K) is set by the energy injection rate [see also Fig. 7.9]. The highest cooling losses are found near the interface of the wind blown bubble and the shell.

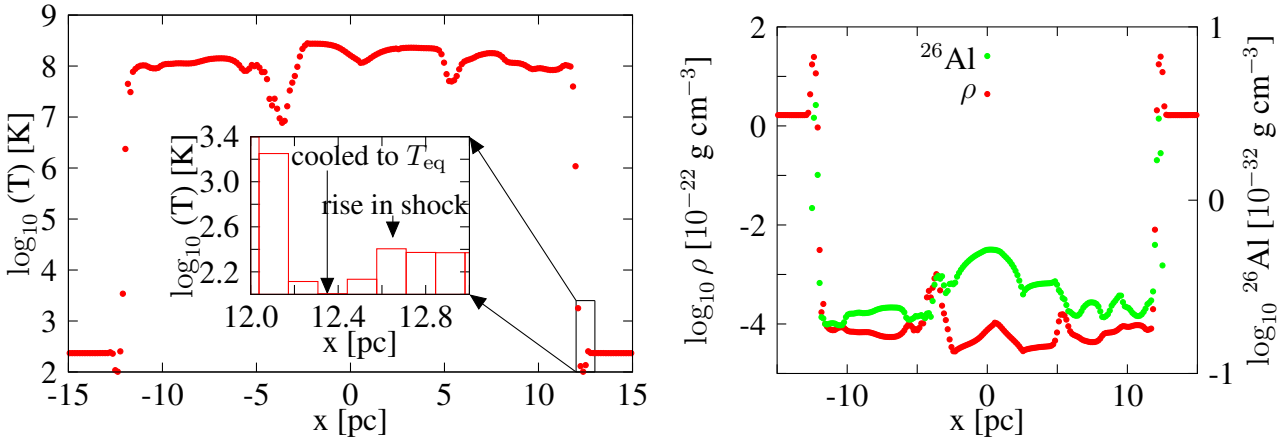
7.5.1 Doubling the feedback

In our simulations, inserting two stars at the same place in the infinite cloud is roughly as efficient (total feedback energy efficiency $\sim 8\%$, Fig. 7.7) as inserting two stars at the same time at infinite distance. A small difference in the total feedback energy efficiency – which is seen e.g. in the thermal energy in Fig. 7.7(a) and in the total energy in Fig. 7.11 – is a relic of the early phase of the bubble evolution. In simulations without pre-existing bubbles (e.g. shown in Fig. 7.7(a)) the different amounts of retained energy result from the phase before the wind bubble manages to excavate a dilute (almost) not cooling region, which is shorter for stronger feedback. Consequently, feedback from isolated stars is slightly less efficient under these conditions. In contrast, the energy differences in simulations with pre-exisiting cavities (Fig. 7.11) reflect the time it takes the wind shell to reach the edge of a pre-existing initial cavity. In this phase (almost) no cooling losses occur. The end of this phase is best seen in the total energy in Fig. 7.11, which starts to deviate from the feedback energy when radiative losses set in. This phase ends earlier if the stars are placed in the same feedback region: Castor et al. (1975) wind theory predicts that the wind bubble's radius increases with $r(t) \propto \rho_c^{-1/5} E^{1/5} t^{3/5}$. Hence, a simulation with isolated stars reaches the cavity edges a factor $\sqrt[3]{2}$ later than a simulation with two stars in the same feedback region. Which is exacly the factor we find when we compare the simulations with different densities inside the cavity or different numbers of stars in Fig. 7.11: In the model with two stars at the same location, a RAMSES cooling function and a density of $\rho_c = 0.92 \times 10^{-24}$ g cm^{-3} in the cavity, we find that the total amount of retained energy is larger after 11.6 kyr than after the next time-step (which is at 12.5 kyr). Therefore, we predict that the same evolution stage is reached a factor $\sqrt[3]{\frac{166}{92}}$ later (at $\sim$ 14.1 kyr), if the density in the pre-ecisting cavity is increased to 1.66×10^{-24} g cm^{-3}. For infinitely separated stars, we expect a factor $\sqrt[3]{2}$ (leading to $\sim$ 14.6 kyr). Finally, a factor $\sqrt[3]{\frac{332}{92}}$ ($\sim$ 17.5 kyr) is expected if both, the density and the number of stars are changed. In our simulations we find a snapshot with these properties at 13.5 kyr for two stars and $\rho_c = 1.66 \times 10^{-24}$ g cm^{-3}, 14.5 kyr for one star and $\rho_c = 0.99 \times 10^{-24}$ g cm^{-3} and 16.6 kyr for one star and $\rho_c = 1.66 \times 10^{-24}$ g cm^{-3}. This is in good agreement with the expectations, since the time between snapshots is $\sim$ 1 kyr.

In simulations with a pre-existing cavity, the longer duration of the almost lossless early phase makes feedback from isolated stars slightly more efficient. However, in both setups (with or without cavity), correcting for this initial phase leads to similar efficiencies for these extreme cases (infinite separation or same position) mimicking concentrated and loose star groups. Shell interactions are not taken into account, but also Krause et al. (2012), who study the effect of stellar wind bubble shell interactions, do not find significant differences in the feedback energy efficiency of isolated stars or star groups during the wind phase.

7.6 Homogeneous semi-infinite cloud with "chimney"

An other means – besides bubble expansion and radiative cooling – to release pressure from the star forming region inside the dense gas cloud is connecting this region with a "chimney" to the ambient medium. In such channels we will first observe a shock wave, clearing the path. After the shock wave has passed, an isentropic flow sets in. This flow will try to establish a pressure balance of both parts of the wind blown bubble – the one inside the GMC and the one outside. However, the sonic flow can have a too small flux to accomplish this, since the sound speed limits



(a) Temperature (b) Density

Figure 7.9: Cut along the x-axis of a 60 $M_\odot$ model placed in a homogeneous cloud without initial cavity after 1.25 Myr. The modified RAMSES cooling model was used. The zoomed region shows an indication of a temperature rise in the shock. The dense swept-up gas cools to the cooling heating equilibrium temperature (100 K). ^{26}Al peaks near the cavity wall.

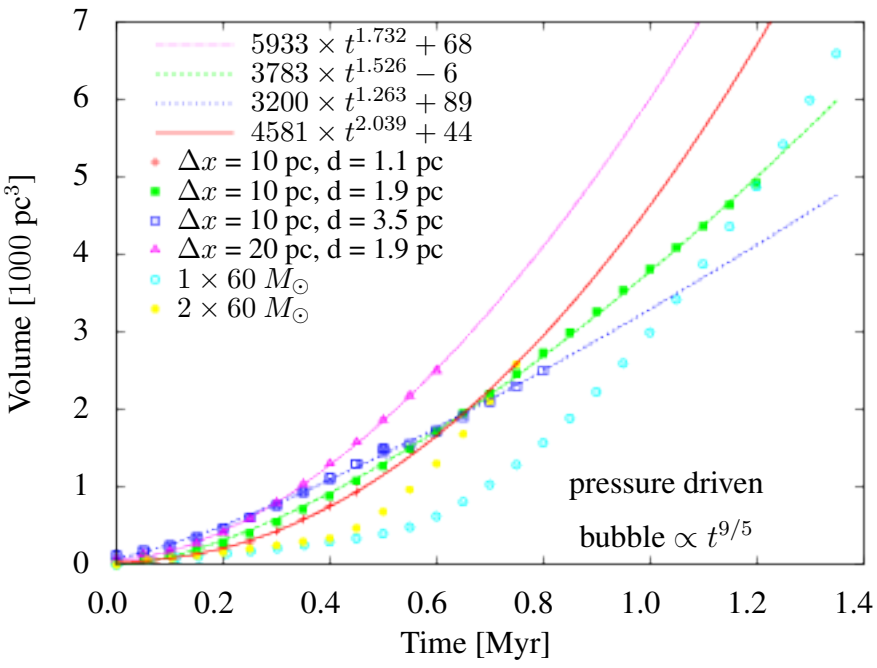


Figure 7.10: Evolution of the cavity's size in different setups. The cavities in the infinite cloud initially grow more slowly then the cavities in the simulation with "chimneys", since there is no pre-existing cavity and thus the mass inside the feedback region is treated like wind-gas and thus there is an artificially extended free expansion phase.

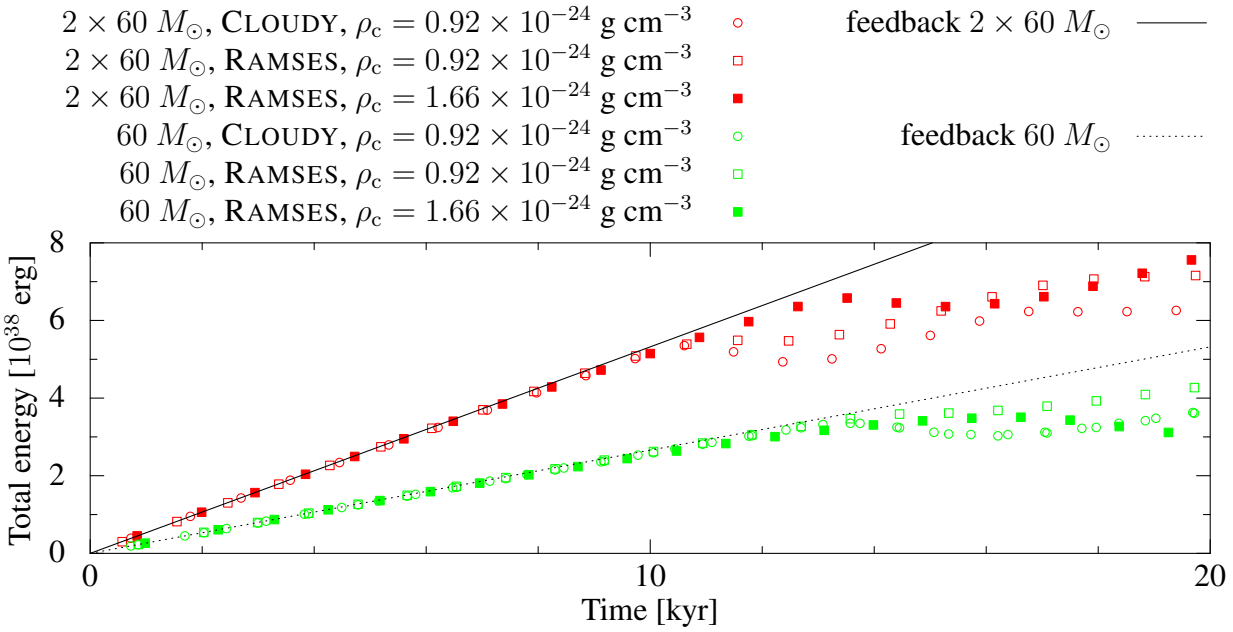
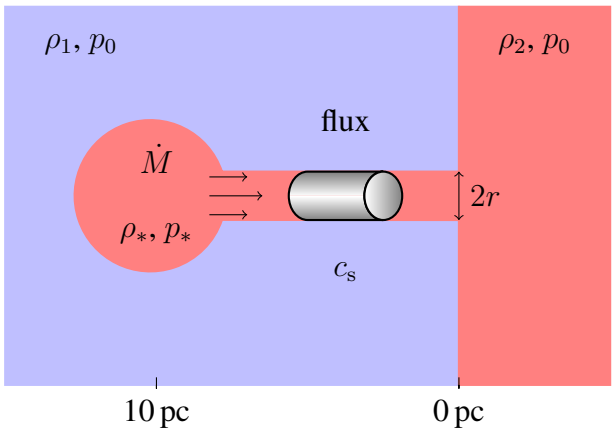
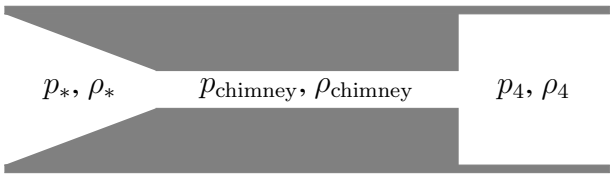


Figure 7.11: Initial almost loss-less expansion before the edge of the initial cavity is reached. Lines show the cumulative feedback energy, symbols indicate the energy increase in the simulation. The time at which the feedback energy efficiency drops rapidly, shows that the edge of the initial cavity is reached. Comparing the turn-off times of models that differ in the amount of feedback or in the density of the dilute medium, confirms that the wind-blown bubble expands with $r(t) \propto \rho_c^{-1/5} E^{1/5} t^{3/5}$. The duration of this phase impacts the feedback energy efficiency and can be seen as a (small) offset at later times.

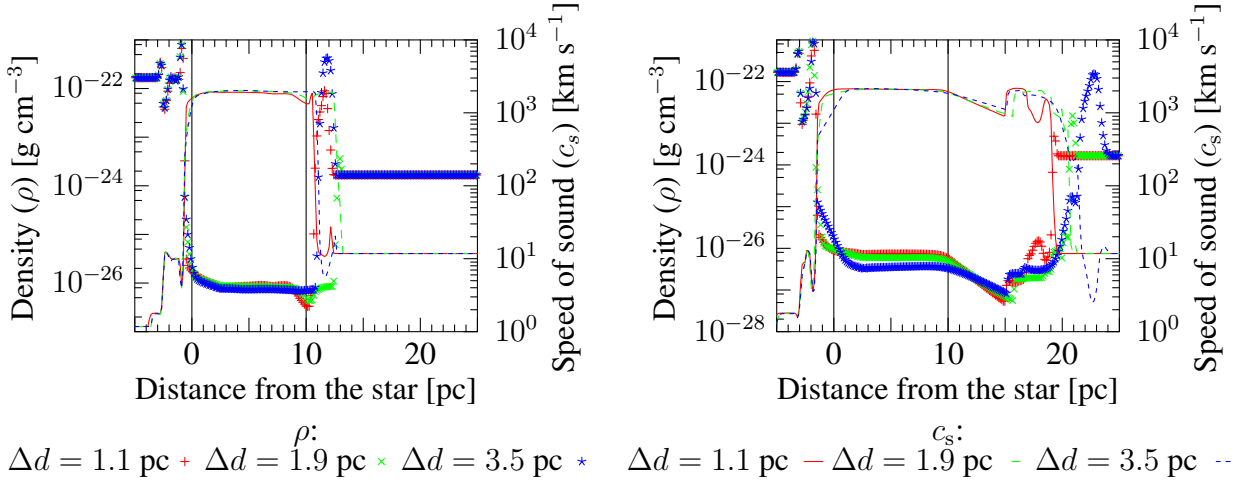


(a) Maximal outflow.



(b) Isentropic flow through a “chimney”.

Figure 7.12: Sketch for the assessment of the critical “chimney” radius. The maximal flux in the “chimney” is set by the sound speed, the density and the cross section. For technical reasons (less artifacts) it can be helpful to include a pre-existing cavity which is slightly larger than the feedback region to ensure that also cells partially inside the feedback region lie fully inside the dilute medium.



(a) 0.1 Myr (b) 0.2 Myr

Figure 7.13: Speed of sound (lines) and density (symbols) in the "chimney". This is a cut along the x-axis. The dashes indicate the position of the feedback region center (0 pc) and the cloud edge (10 pc). All "chimneys" have a length of 10 pc, but they differ in diameter. It can be seen that the flows in the two narrower "chimneys" (1.1 and 1.9 pc initial width) show a density maximum lagging behind the density maximum in the "chimney" with 3.5 pc initial width.

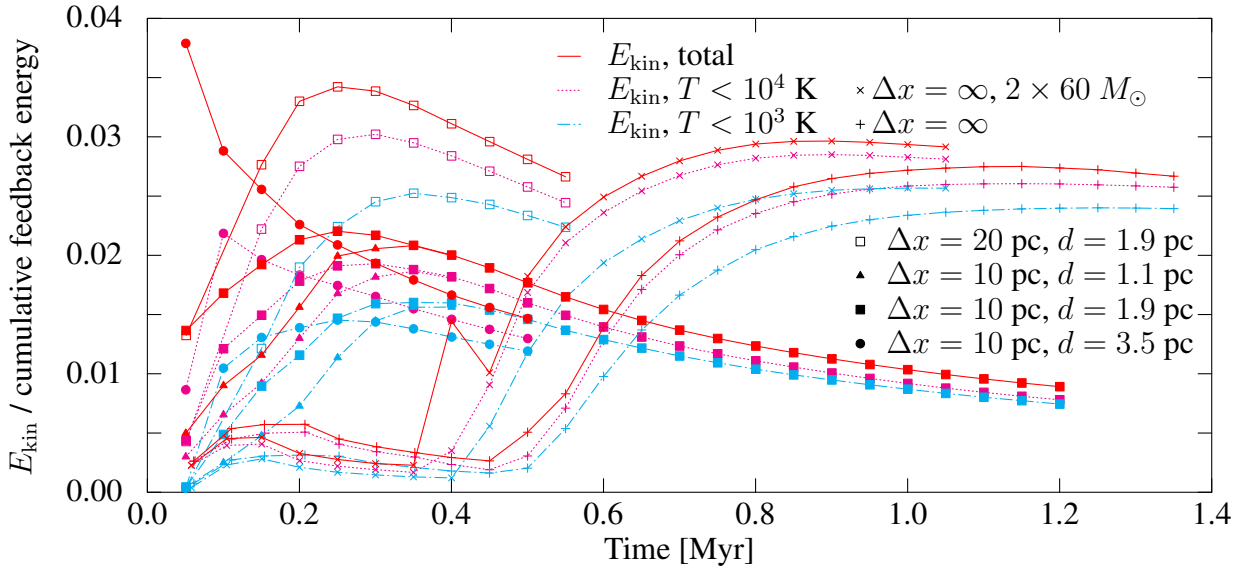
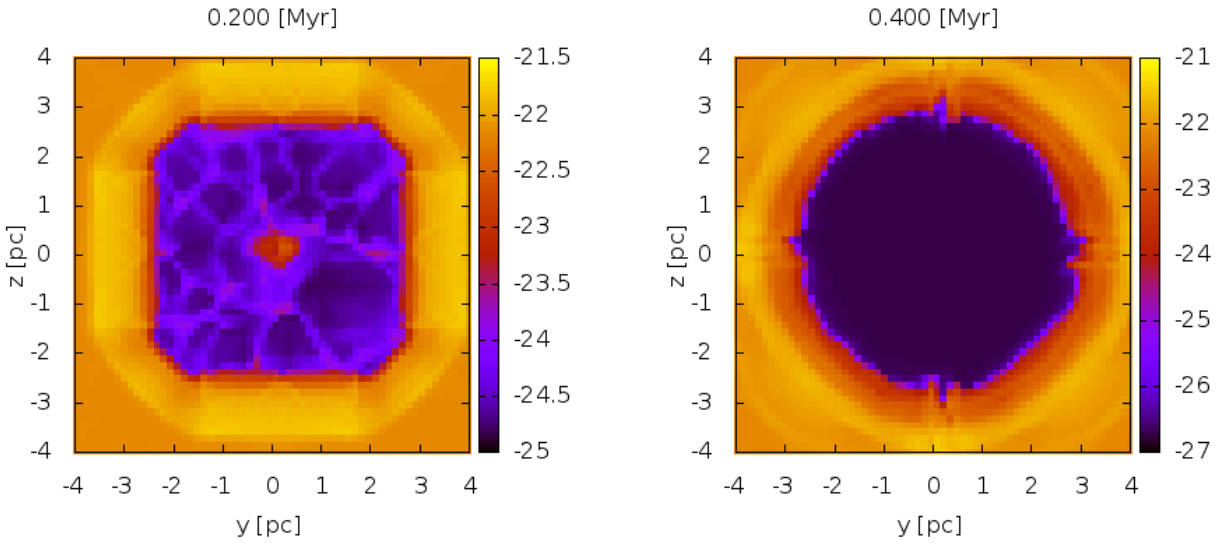


Figure 7.14: Retained kinetic energy in the cloud region compared to the stellar feedback. In contrast to Fig. 7.7(a) this plot shows the energy uptake in the region where the dense cloud was located in the initial conditions.



(a) Initial diameter: 3.5 pc

(b) Initial diameter: 1.1 pc

Figure 7.15: The plots show the minimal cross section along the "chimney": they display averaged density (in g cm^{-3}) as seen from the star looking along the "chimney" axis. We stop integrating $\sim$ 15 pc outside the cloud's surface to exclude the receding shell. The minimal cross section areas are $\sim$ 25 pc^2 in both setups. The pixel size in the plots corresponds to the pixel size in the simulations.

the propagation speed in the "chimney". The flow will work towards lowering the density in the overpressured cavity in the cloud. As a lower limit, we can calculate the minimal "chimney" diameter that is necessary to remove all the newly inserted stellar ejecta $\dot{M}$. As can be seen from Fig. 7.12(a), the minimal "chimney" diameter to remove all newly injected stellar yields is $A_{\rm crit} = \dot{M}/c_{\rm s,chimney}/\rho_{\rm chimney}$. In Fig. 7.12(a) the maximal mass flow rate out of the cavity is visualized with a cylinder. The length of this cylinder is set by the sound speed in the "chimney". The colors show the initial density distribution. At the start, the whole computational box is in pressure equilibrium at pressure p_0. The cold cloud has a density (ρ_1) of 100 particles per cubic centimeter corresponding to a temperature of about 100 K and the surroundings have a density (ρ_2) of one particle per cubic centimeter. Stellar feedback will enhance the pressure (p_*) and the gas flow resulting from this will lower the density (ρ_*) in the feedback region. To find the critical cross section $A_{\rm crit}$ of the "chimney", we sketch the pressure and density distribution in the problem at a later time of the evolution in Fig. 7.12(b). For a setup like this, we expect an isentropic flow[6] from the feedback region (p_*, ρ_*) through a "chimney" ($p_{\rm chimney}$, $\rho_{\rm chimney}$) with a cross section A into a region that sweeps up ambient medium:

$$\frac{\rho_*}{\rho_{\rm chimney}} = \left(1 + \frac{\gamma-1}{2} M^2\right)^{-\frac{1}{\gamma-1}} \qquad \text{isentropic flow.} \tag{7.1}$$

The simulations indeed find an isentropic flow with constant $p\rho^{-\gamma}$ in the "chimney" and in the free flowing zone downstream. The sonic point is reached near the downstream end of the "chimney",

[6]The equations for an isentropic flow can be found e.g. at http://www.grc.nasa.gov/WWW/k-12/airplane/isentrop.html

which is also where we find the smallest cross section. The "chimney" cross section decreases downstream, since the shock wave reaches this region later and in the mean time the pressure already had time to act on other parts of the "chimney" surface. If the cross section of the "chimney" is below the critical value ($A < A_{\rm crit}$) for free flow, we observe a choked flow. In this case, the flux through the "chimney" is no longer influenced by the downstream pressure (p_4) if it is lower than $(\frac{2}{\gamma+1})^{\gamma/(\gamma-1)} p_{\rm chimney}$. For an adiabatic exponent of $\gamma = \frac{5}{3}$, this is $0.487\, p_{\rm chimney}$. In this case – and if the bubble would not change its volume by expanding into the cold cloud – the pressure in the bubble would rise until the "chimney" cross section gets large enough that no choked flow occurs any more. In reality, we see a superposition of these two effects, leading to delayed pressure loss.
Fig. 7.13 shows the profiles of the density in the "chimney" ($\rho_{\rm chimney} \sim 8 \times 10^{-27}$ g cm^{-3}) and the speed of sound ($c_{\rm s,chimney} \sim 2 \times 10^8$ cm s^{-1}) 0.1 Myr and 0.2 Myr after the onset of the wind. The mass loss rate is $\sim 2 \times 10^{-6}\ M_\odot$ per year or 1.26×10^{20} g s^{-1}. This leads to a lower limit of $A_{\rm crit} \sim \frac{1.26}{2\times 8} \times 10^{20+27-8} \sim 8.3$ pc^2 or $\Delta d_{\rm crit} \sim 2.9$ pc. A comparison of the time evolution of models with different "chimney" cross sections shows a faster pressure drop in the part of the wind-blown bubble that is inside the cloud as soon as this critical cross section is passed.
The lower limit for the critical "chimney" cross section $A_{\rm crit}$ can also be found from the conditions inside the feedback region via the isentropic flow:

$$\begin{aligned}
\frac{c_{\rm s,chimney}}{c_{\rm s,*}} &= \left(\frac{\rho_*}{\rho_{\rm chimney}}\right)^{\frac{1-\gamma}{2}} && \text{(Eq. 4.55 or 4.2)} \\
\frac{c_{\rm s,chimney}}{c_{\rm s,*}} &\overset{M=1}{=} \left(\frac{\gamma+1}{2}\right)^{-\frac{1}{2}} && \text{(from Eq. 4.55 and 7.1)} \qquad (7.2)
\end{aligned}$$

$$\begin{aligned}
\dot{M} &< A c_{\rm s,chimney} \rho_{\rm chimney} \\
\dot{M} &< A c_{\rm s,*} \rho_* \left(\frac{\gamma+1}{2}\right)^{-\frac{1}{2}+\frac{1}{\gamma-1}} && \text{with } c_s = \sqrt{\gamma \frac{p}{\rho}} \\
\dot{M} &< A\sqrt{\gamma p_* \rho_*} \left(\frac{\gamma+1}{2}\right)^{-\frac{1}{2}\frac{3-\gamma}{\gamma-1}} \overset{\gamma=5/3}{=} 0.97 A \sqrt{p_* \rho_*} && (7.3)
\end{aligned}$$

The initial values are $p_* = 1.38 \times 10^{-12}$ g cm^{-1} s^2, $\rho_* = 1.66 \times 10^{-24}$ g cm^{-3} and $\dot{M} = 1.26 \times 10^{20}$ g s^{-1}. This leads to $A_{\rm crit} = \frac{\dot{M}}{0.97\sqrt{p_* \rho_*}} \sim 8.6 \times 10^{37}$ cm^2 $= 9$ pc^2 or $\Delta d_{\rm crit} \sim 3$ pc. This is larger than two of the three tested cross sections. Our test runs used[7] $\Delta d = \{3.5, 1.9, 1.1\}$ pc. Due to the overpressure of the hot wind gas, the diameter of "chimney" grows with time. After 0.4 Myr e.g. the "chimney" with an initial diameter of 1.1 pc already grew by a factor 5 in diameter.

7.6.1 The "chimney" width

Since cold gas is found in the swept-up cloud gas as well as in the shell of swept-up ambient medium, we also evaluated the kinetic energy at the initial location of the cold cloud (Fig. 7.14). This way we can monitor how much energy is found in the (remainders) of the cloud. This estimate is useful, since we need the energy inside the cloud to drive turbulence there.
We see that the kinetic feedback energy efficiency in the cloud material below 10^3 K stops rising at 0.2 Myr if the initial "chimney" width was 3.5 pc whereas the model with an initial width of

[7] The plan was to use $\Delta d = \{1, 0.5, 0.25\} \times 10^{19}$ cm, resp. $\{3.2, 1.6, 0.8\}$ pc, but the IC routine added one cell at each side of the region.

1.1 pc needs twice as long to reach this phase. Fig. 7.15 shows that this evolution phase in the two setups with different initial "chimney" width are reached when comparable cross section areas are observed: In both setups the cross section at this time is roughly 25 pc^2.
The models in this sensitivity analysis do not have pre-existing cavities. This leads to a gas phase with $10^3 \leq T < 10^4$ created by the part of the feedback region that does not overlap with the initial "chimney" zone. The peak of the kinetic energy in such gas with $10^3 \leq T < 10^4$ inside the cloud zone, is reached at break-out at ~ 0.1 Myr for the model with $\Delta d = 3.5$ pc. The model with $\Delta d = 1.9$ pc reaches this phase at 0.2 Myr, when the cross section has increased to 11.14 pc^2 from 7.3 pc^2 at 0.15 Myr. And finally, the model with $\Delta d = 1.1$ pc reaches this phase after 0.25 Myr at a cross section of 9.7 pc^2. Again we find a similar evolution phase at compareable cross sections.
To summarize, Fig. 7.7 shows a higher total feedback energy efficiency for wider "chimneys". We also find a higher total kinetic feedback energy efficiency in the simulations with wider "chimneys". This can be understood, since – due to the aforementioned choked flow problem – larger "chimneys" can transport more energy out of the cloud and build up larger not cooling pressure reservoirs. However, for the same reason, the kinetic feedback energy efficiency in the cold cloud material is lower for wider "chimneys", as can be seen in Fig. 7.14.

7.6.2 The "chimney" length

The "chimney" length influences the amount of energy, which is deposited in the cloud, via the break-out time. If we double the length of the $10 \times 1.9 \times 1.9$ pc "chimney", the first snap-shot showing break-out is found at 0.165 Myr instead of 0.1 Myr. I.e. in Fig. 7.7 at 0.1 Myr and 0.15 Myr the shock front of the $10 \times 1.9 \times 1.9$ pc model has already passed the end of the "chimney", whereas it is still stuck therein in the $20 \times 1.9 \times 1.9$ pc model. We see in Fig. 7.7 and 7.14, that during this time the longer "chimney" (open squares) leads to more retained energy inside the cloud and less (i.e. none) outside than the shorter "chimney" (filled squares), since with the longer "chimney" the stellar yields have not yet found their way out of the cloud.

7.7 Convergence

In the same way as discussed in the chapter on our 1D work, we also checked the influence of free parameters in our 3D models by varying them one by one. Among the parameters tested were the feedback model (e.g. stars of different masses, groups of stars), the implementation of the feedback (kinetic or thermal energy input, feedback region size, treatment of cooling in the feedback region, pre-existing cavities, treatment of cells partly inside the feedback region), the cooling model, the spatial and temporal resolution and the numerical method (Riemann solver type, flux limiting scheme, number precision).
The conclusions from these tests are similar to what we see in 1D: Whereas the results are quite robust against changes in temporal resolution and the choice of the cooling function, they show a dependence on spatial resolution and the diffusivity of the Riemann solver. Interestingly, the number precision only has a minor effect on the feedback energy efficiency (open symbols in Fig. 7.16).
Our interpretation of these results is that the treatment of the CD is very important for the feedback energy efficiency. This can be seen e.g. in Fig. 7.16 (crosses), where the acoustic Riemann solver, which ignores the CD, finds a lower feedback energy efficiency than the HLLC Godunov scheme.

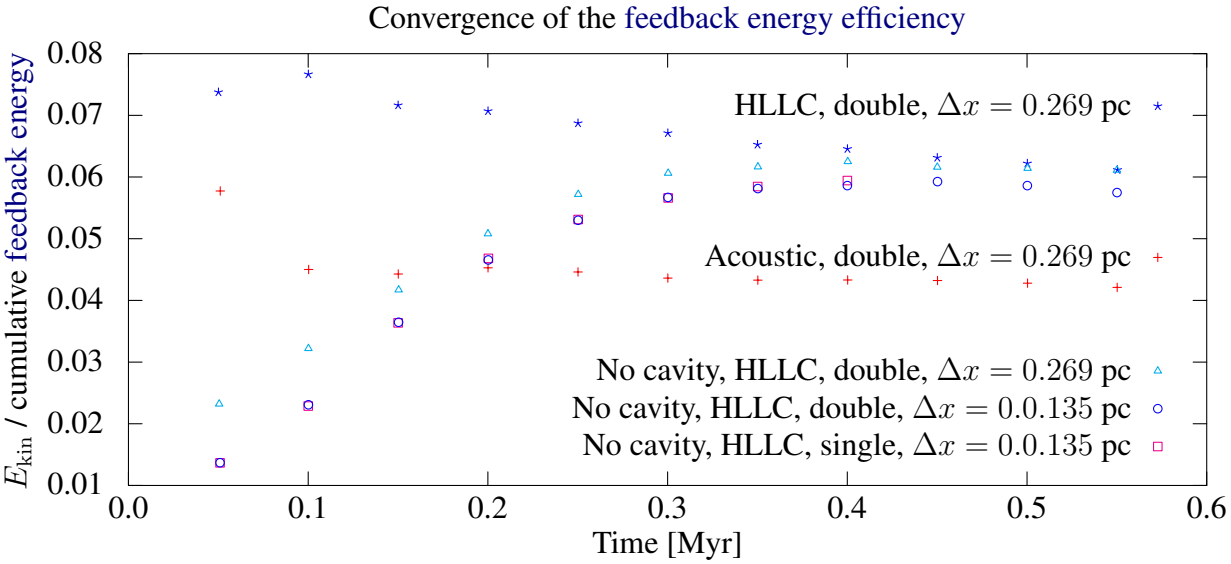


Figure 7.16: Variants of the $10 \times 1.9 \times 1.9$ pc "chimney" model. A comparison of models with pre-existing cavities (crosses) and models without, shows that the influence of the pre-existing cavity is overcome at ~ 0.55 Myr. In this regime, increasing the spatial resolution lowers the feedback energy efficiency. Changing the number precision did not have a significant effect on the feedback energy efficiency. A more diffusive Riemann solver lowered the feedback energy efficiency.

The explanation for this behavior is that this change does not alter the width of the swept-up shell or the maximum density significantly. However, not treating the CD accurately results in more mixing across the CD. This mixing of "too-cold-to-cool" swept up material and "too-dilute-to-cool" wind material in turn leads to enhanced radiative losses near the CD.
Since we do not reach the resolutions of our 1D work with the 3D models yet, the convergence behavior of the feedback energy efficiency between the studies differs. Basically, in 1D the feedback energy efficiency rises with increasing resolution since the mixing across the CD, which is the most important energy loss channel, decreases with increasing resolution. At low resolution, the CD is smeared out and energy losses due to mixing across the CD become less important than the cooling at the peak density, which rises with density squared. In this regime higher resolution leads to higher peak densities and thus lower feedback energy efficiency (Fig. 7.16).
As a consequence, simulations find a minimal efficiency, if the resolution starts to be high enough to produce a strongly cooling cell at the CD at every time step. If the resolution is lower than this, strongly cooling cells at the CD are created less frequently. At higher resolutions (than in the simulation with minimal feedback energy efficiency), the zone with the high energy losses is found near the CD and gets smaller with increasing resolution. Thus, the feedback energy efficiency rises.

7.8 Conclusions from the 3D "chimney" models

Our 3D simulations show that $\geq 90\%$ of the stellar feedback energy leave the cloud immediately via radiative losses. Convergence tests (Sect. 7.7) at resolutions near 0.13 pc indicate that simulations with this resolution are still in a regime where the feedback energy efficiency decreases with increasing resolution. As discussed in the last chapter, we argue that at higher resolution the

efficiency of mixing across the CD strongly influences the feedback energy efficiency. In our 1D work we further argued that one needs to identify the most efficient mechanism for mixing across the CD. With this information, one can extract the feedback energy efficiency from a simulation with mixing of similar strength caused by our numerical methods.

Consequently – as the 3D simulations do not reach the resolutions necessary for this – the predictive power of our present 3D data lies in a comparison of models which differ in only one aspect, i.e. the existence, width and length of a "chimney", which can be understood as a proxy for the cloud's density structure.

If the star forming region is connected to the ambient medium via a "chimney", obviously the depth of embedding (in this case parametrized via the "chimney" length) will change the break-out-time. The stellar feedback creates a shock traveling through the "chimney". When this shock wave reaches the ambient medium, it depends on the cross section of the "chimney", how fast the pressure can escape from the cloud. If the speed of sound at the smallest cross section of the "chimney" limits the flow, we call this a choked flow. In this case, the downstream pressure in the part of the bubble that is outside the cloud cannot influence the pressure in the cavity and the "chimney". Therefore, during a choked flow more energy can be deposited in the cloud, than in the presence of a wide enough "chimney" to establish a homogeneous pressure in all parts of the bubble filled with stellar feedback. In our simulations we find indeed an isentropic flow in the "chimney" with a sonic point near the smallest cross section.

To summarize, we expect higher kinetic feedback energy efficiencies in the cloud material and lower total feedback energy efficiencies for deeper embedded stars. The deeper the stars are embedded, the longer the pressure is confined in the cloud, which leads to more acceleration of the cold swept-up cloud gas. Less embedded stars manage to channel more energy to the ambient medium. Radiative losses peak in the bubble shell. Thus, less embedded stars can build up pressure reservoirs outside the cloud. Also a higher porosity leads to a faster loss of pressure to the bubble parts outside the cloud. Fig. 7.7 and Fig. 7.14 agree with the expected trends, how the length and the width of the "chimney" are expected to influence the amount of kinetic energy that is deposited in the cloud material.

For our re-simulations we will start with pre-existing cavities one cell larger than the feedback region, since otherwise the "chimney" will lower the density in a part of the feedback region (up to 50%). Starting with the same initial density in all feedback regions makes interpreting the results in the early phase easier. The lower initial density in this region also reduces the artificial prolongation of the free expansion phase, since the initial mass in the feedback region will end up "on the wrong side of the CD". We will use the cooling-heating function of (Ntormousi et al., 2011), since it provides us with two thermal phases in pressure equilibrium and contains more physics than setting up artificial equilibria. For the dense gas in the IC we will use the same density as in the 1D models and the temperature corresponding to the cooling-heating equilibrium for this density. For the warm component, we will use the same pressure. Therefore, the density follows from the cooling-heating equilibrium. Finally, we will extend our grid to test smaller "chimney" cross sections and take care that the IC routine does not add a layer of cells around the "chimney". In our set of models, we had the fully embedded stars as limiting cases. The other extreme case, a feedback region at the cloud's surface, will be part of our future work. Acutally a model of this type will be shown as reference model in the next chapter. It has, however, a lower resolution (~ 0.5 pc) than the models presented in this chapter. Anyway, it is interesting to note that choked flows also occur in the models presented in the next chapter. They are visible via the slight overpressure in the region inside the dense cloud and the sonic point near the cloud's surface.

Chapter 8

3D: Feedback in non-homogeneous clouds

The aim of this chapter is to motivate future work on the ^{26}Al distribution. The model presented here is a test run[1] for future, better resolved models of this kind. It places stellar feedback in the densest part of a GMC with non-homogeneous density. The IC for this cloud are taken from large scale simulations of Dobbs et al. (2011). We will use this model to explain, how we create artificial ^{26}Al observations from our simulations, since it illustrates the spread of ^{26}Al in less artificial environments than presented in Sect. 6 and 7. In the reference models, which use a cloud with a homogeneous density of 1.66×10^{-22} g cm^{-3}, a radius of 25.1 pc and a temperature of 100 K, the feedback region is placed in the cloud center or 1 pc below the surface. The ambient medium is in pressure equilibrium with the cloud and has a temperature of $\sim 10\,000$ K. In contrast to the infinite clouds in Sect. 6 and the semi-infinite clouds in Sect. 7, the clouds discussed here have a finite size. The properties of both cloud models can be found in Tab. 2.1. Their size and mass are comparable to the properties of the Orion A and B molecular clouds. The initial column densities for both models are shown in Fig. 8.1.

In contrast to Sect. 6 and 7, we will now use the stellar feedback from the population synthesis of Voss et al. (2009) instead of a single star. The motivation for this is that we had observed that this kind of feedback has a quite disruptive effect on homogeneous clouds. We were thus interested, if non-homogeneous clouds were able to dump the feedback energy in their surroundings.

8.1 Simulation Setup

For the simulations shown in this chapter, we used the RAMSES code (Teyssier, 2002) with the modifications discussed in Sect. 5.2.3 and 7. The standard code settings for RAMSES include an adiabatic exponent of $\gamma = \frac{5}{3}$, a Courant factor of 0.5 and outflow (zero gradient) boundary conditions. In contrast to Sect. 7, we use the the MinMod slope limiter and the acoustic Riemann solver for these simulations, since this is the most robust method. The cubic computational box has a length of 68.9 pc. The resolution of the 3D simulations was $128 \times 128 \times 128$ cells or 0.54 pc. The radius of the feedback region is 2.43 pc (or ~ 4.5 cells).

[1] The main benefit of these tests was an optimization of the output of the simulation. Since the snapshots of our simulations are quite memory intensive, we had to test which quantities we want to analyze on the fly and how often we need to store a snapshot. Moreover we tested different energy injection techniques or feedback region radii.

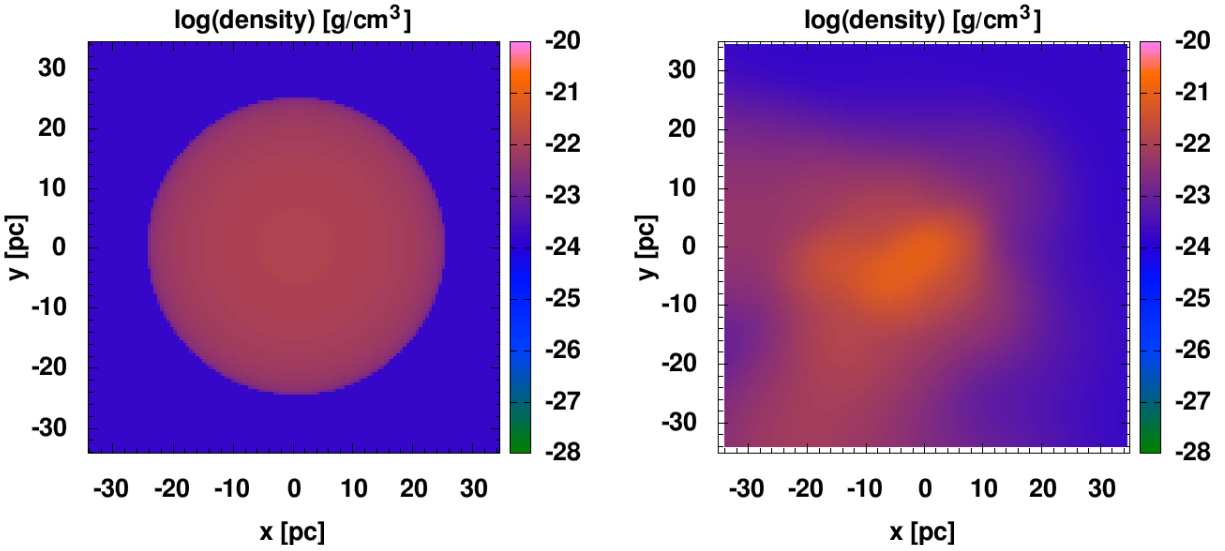


Figure 8.1: Initial conditions. The figures show logarithmic average densities in the computational box in g cm^{-3} . The properties of the homogeneous spherical cloud (left) and the SPH cloud taken from (Dobbs et al., 2011) can also be found in Tab. 2.1.

8.2 Results

The feedback energy efficiency of the population synthesis feedback based on Voss et al. (2009) in homogeneous and realistic clouds (Fig. 8.2) is in agreement with our findings in Sect. 6 and 7, where we used feedback of a single, massive star. As in Fig. 7.7(a), also in Fig. 8.2 $> 90\%$ of the stellar feedback was immediately radiated away. After break-out of the bubble from the GMC the radiative losses decreased, since radiative losses peak in the compressed shell. Interestingly, as in the break-out at the end of a "chimney", also the model in which the feedback region was placed only 1 pc below the cloud's surface seems to show a flow inside the superbubble that is limited by the speed of sound. Since the resulting superbubble shape has similarities with the shape observed in Sect. 7, we will now also denote the point at which the superbubble diameter suddenly changes as "end of the "chimney"". In Fig. 8.3, which shows the simulation after 2 Myr, we see the sonic point near the end of the "chimney" and observe a slight overpressure in the part of the superbubble that is bounded by the GMC material. As in Sect. 7 also here the kinetic energy rises after break-out from the GMC.

8.3 Artificial observations of ^{26}Al

The simulations also follow the radioactive isotope ^{26}Al (see Sect. 2.4.2, 2.7.1 and 5.2.4) to trace mixing processes of stellar ejecta with the ISM. Considering ^{26}Al in the numerical simulations should explore interpretational views for the measurements of ^{26}Al emission from the Orion-Eridanus region, since the simulations predict whether ^{26}Al should be detected predominantly in the narrow shell or in the inside of the superbubble.
In our present set of models the ^{26}Al distribution peaks near the cavity walls (Fig. 7.9(b), 8.3 and 8.6. We now briefly present the tools we developed to produce artificial observations of the ^{26}Al velocity in our simulations.

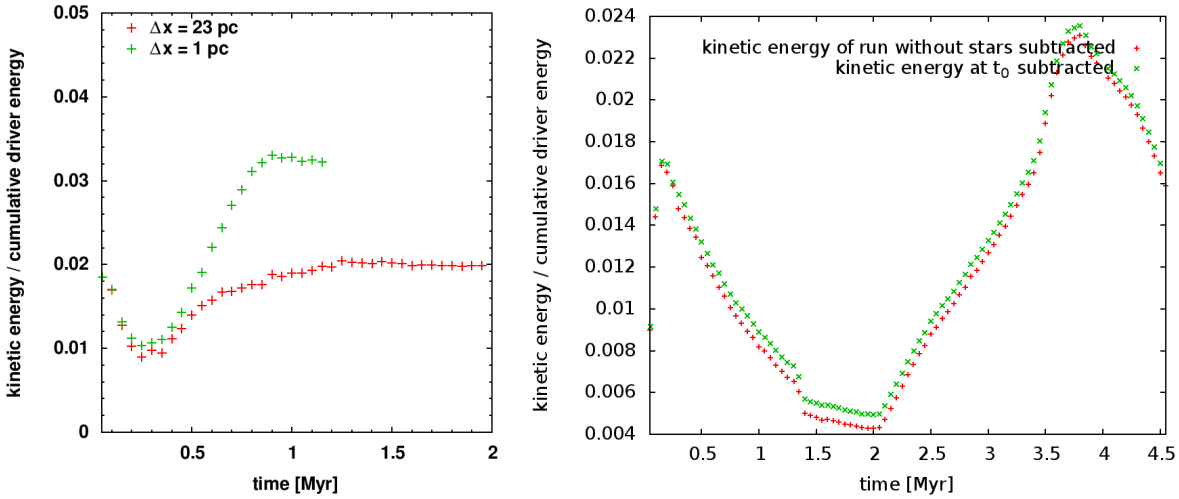


Figure 8.2: Feedback energy efficiency. These graphs show the effect of the density structure of the surrounding medium onto the fraction of the feedback energy from the Voss et al. (2009) model that can be converted into kinetic energy of the ISM. Left: homogeneous cloud with different distances Δx between the surface of the feedback region and cloud surface. The feedback energy efficiency is only followed until the bubble breaks out of the computational box. Right: structured cloud. The OB association is assumed to move with the same velocity as the GMC. Since the cooling–heating function in the SPH simulation differs from the RAMSES cooling–heating function, the behavior of a cloud without stellar feedback is subtracted (red points). As a comparison the green points show the same data with only the initial kinetic energy of the cloud subtracted.

In Fig. 8.3 we see a 3D simulation of a homogeneous cloud with an off-center OB association. This snapshot will be used as an example to discuss the method. The observer is placed at (0,0,+400 pc) with respect to the center of the feedback region and we place a "target point" in the center of the feedback region. Vedrenne et al. (2003) report an angular resolution of 2.5^{deg} for SPI (Spectrometer on INTEGRAL). They mention that sources can be localised better, depending on the source intensity. R. Diehl (private comm.) estimates an angular resolution of 2^{deg} for the Orion-Eridanus region. Thus, to take the resolution of the instrument into account, we select all cells in our simulation that are within the viewing angle of one degree (i.e. angle target – observer – cell center). This way, we get all cells within a cone with an opening angle of 2 degrees. If we decide that a certain column density leads to optically thick gas, we can further limit the number of cells taken into account. However, extinction is not a problem for the ^{26}Al observations, since the absorption depth (decrease of the signal to 1/e) for ^{26}Al is reached at a column density of the order of a few grams per cm^2 (page 12 Schönfelder, 2001; Diehl, 2014, report an estimate of the order of 3 g cm^{-2} found from balloon missions). For material of solar metallicity and an average density of the order of 100 particles cm^{-3} a column density of 1 g cm^{-2} is reached after $\sim$ 2 kpc, which is much larger than the assumed distance to the OES and the spatial extent of our whole simulation. Thus, even if our whole computational box would be filled with GMC material, γ-radiation from ^{26}Al could still penetrate it.

For the selected cells, we store the velocity and – as a quantity mimicking the intensity – ρ/d^2, where d is the distance of the cell from the observer. We then subdivide the range of 0 to 100 km s^{-1} into 1 km s^{-1} bins and sum ρ/d^2 in these bins (Fig. 8.4).

The natural line width of the γ-line is negligible (^{26}Mg^{2+} has a half life of 476 fs leading to a line

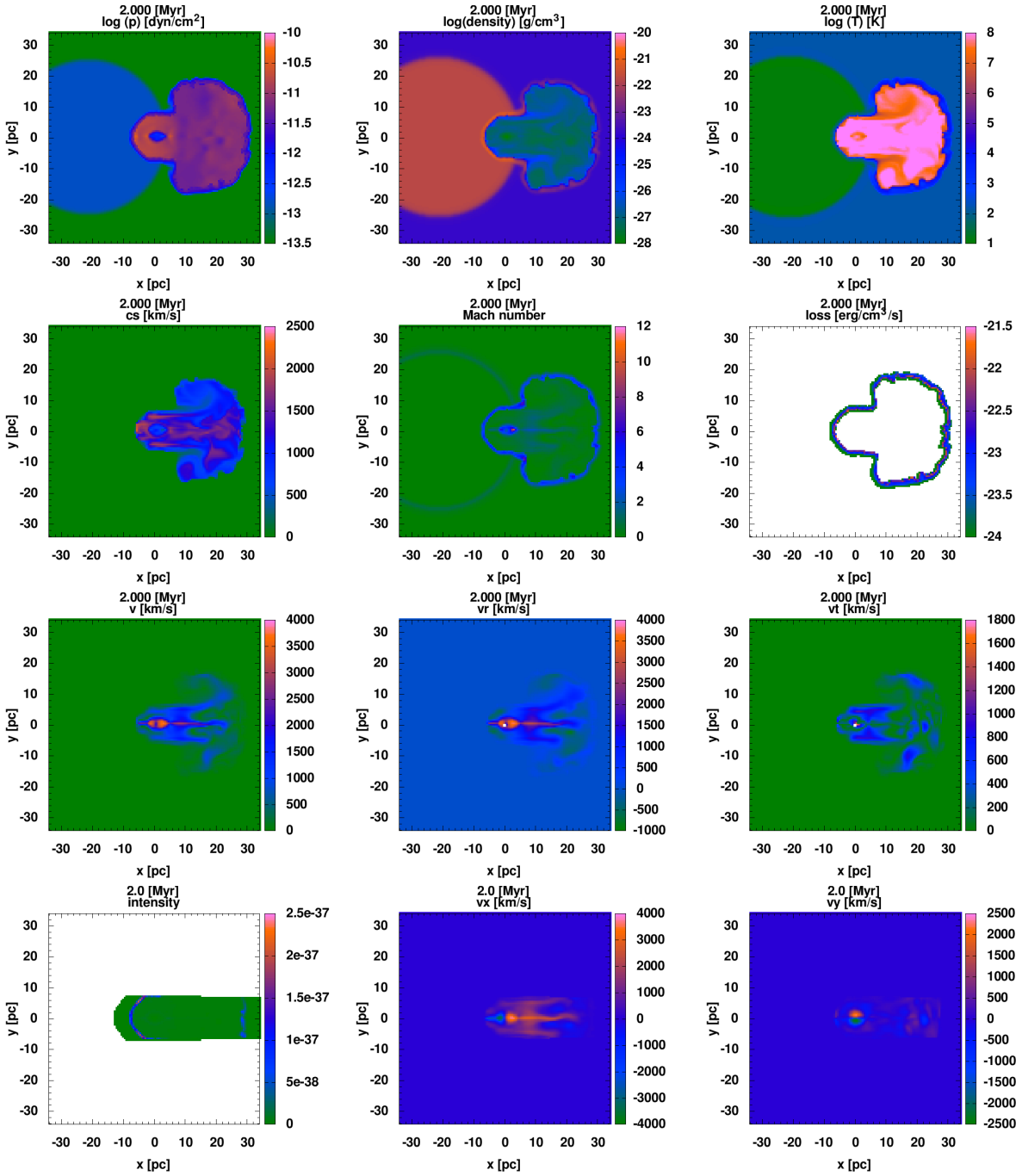


Figure 8.3: This figure shows xy cuts through the simulation data cube used for the artificial observations of ^{26}Al in a simulation with a homogeneous cloud. In the bottom row the intensity of the ^{26}Al in the sight angle of the observer and the velocities in this viewing angle are shown.

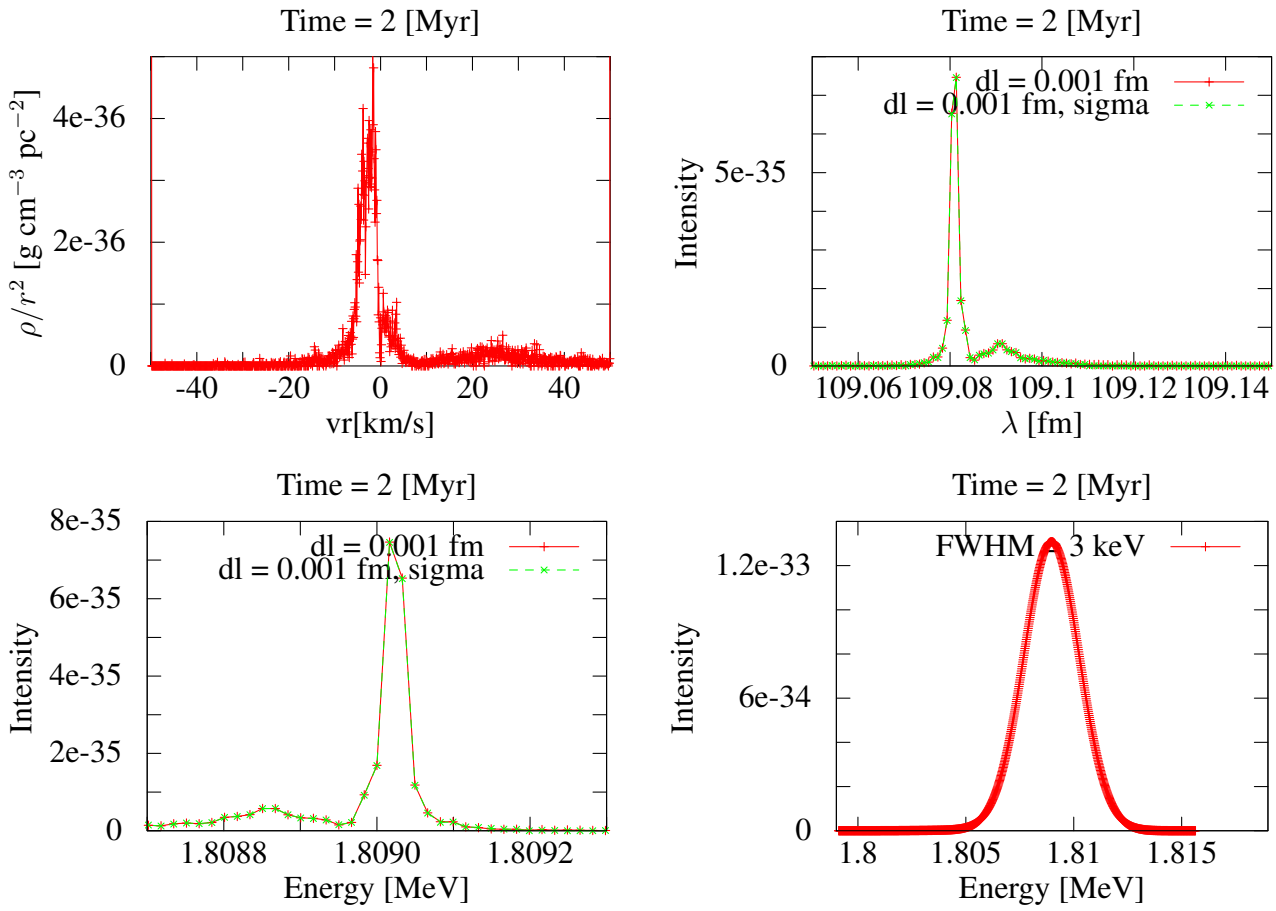


Figure 8.4: Work flow of an artificial observation (1) integrated intensity in radial velocity bins, (2) line without instrument profile (fm), (3) line without instrument profile (MeV), (4) line with instrument profile (MeV)

width of 0.7 meV).Thus, we can start with a single energy. We then calculate the Doppler shift $\Delta\lambda$ of the 1809.63 keV line ($\frac{v_{\rm gas}}{c} = \frac{\Delta\lambda}{\lambda_0}$ with $\lambda_0 = \frac{\hbar c}{1.80963}$ MeV $= 109$ fm and $\hbar c = 197.33$ MeV fm) and take the instrumental profile (R. Diehl (private comm.) assumes a Gaussian with 3 keV FWHM at 1.80963 MeV, Vedrenne et al. (2003); Roques et al. (2003) report an energy resolution of 2.5 keV at 1.3 MeV, which degrades with time and which gets largere for higher energies. Roques et al. (2003) find a mean energy resolution of 2.9 keV at 1764 keV.) into account. For the latter we use a discretized Gaussian of given FWHM and center it in the energy bin. We then multiply our proxy for the intensity with the Gaussian and sum over the Gaussians for all bins.
As a result the initial skewness of the profile in Fig. 8.4 is no longer seen, since it is smeared out. To conclude, there are several reasons why this result should not be interpreted as a negative prognosis for the observability of velocities in ^{26}Al: First of all, we used a very badly resolved simulation of a quite artificial setup for these tests. Also we did not optimize the time of the snapshot or the viewing angle to get a maximal effect. Fig. 8.5 shows that after 5 Myr a redshifted component becomes visible in ^{26}Al.

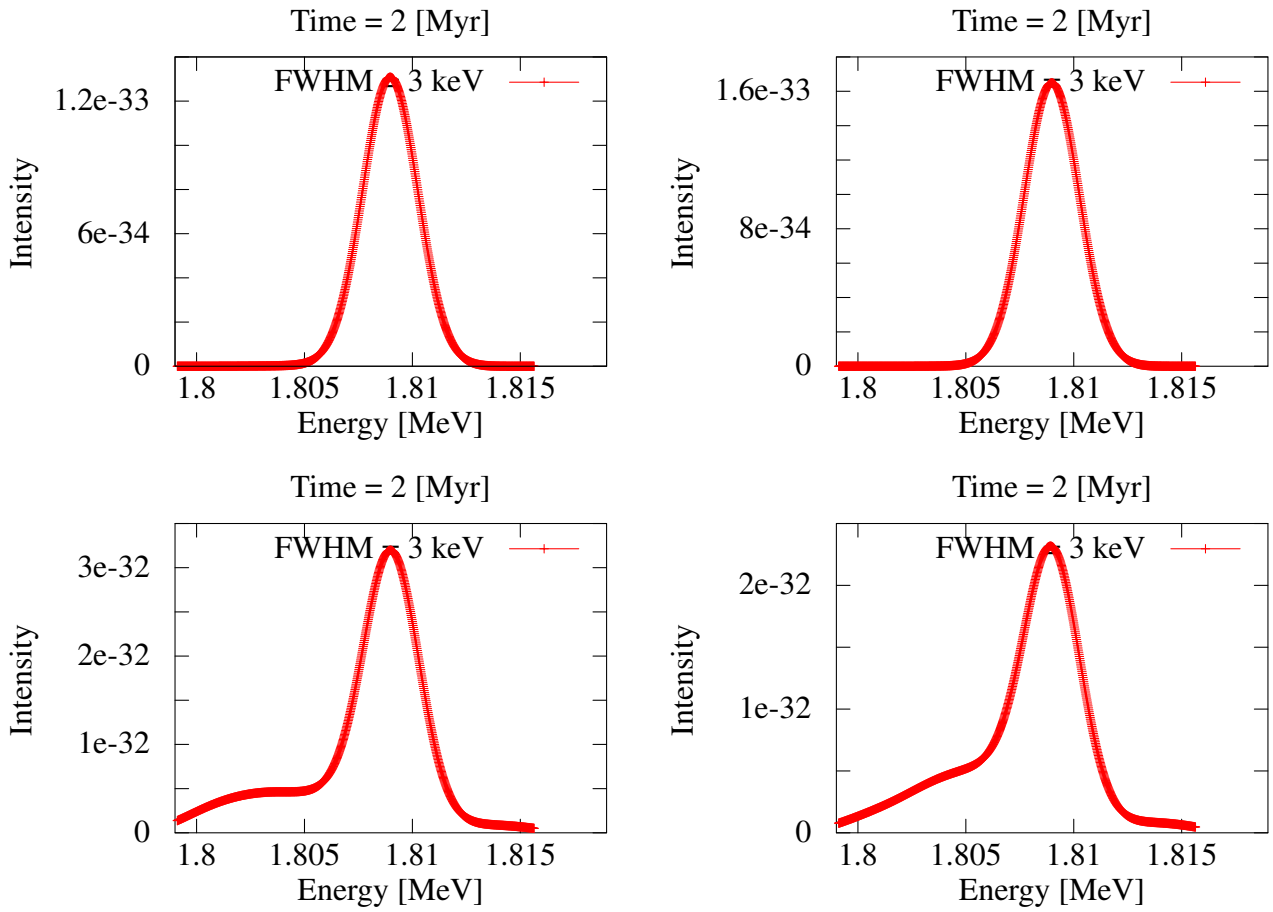


Figure 8.5: Line with instrument profile (MeV). The viewing angle is 0 degrees in the left plots and 45 degrees in the right plots. The snapshots in the top row were taken after 2 Myr. In the bottom row snapshots after 5 Myr are displayed.

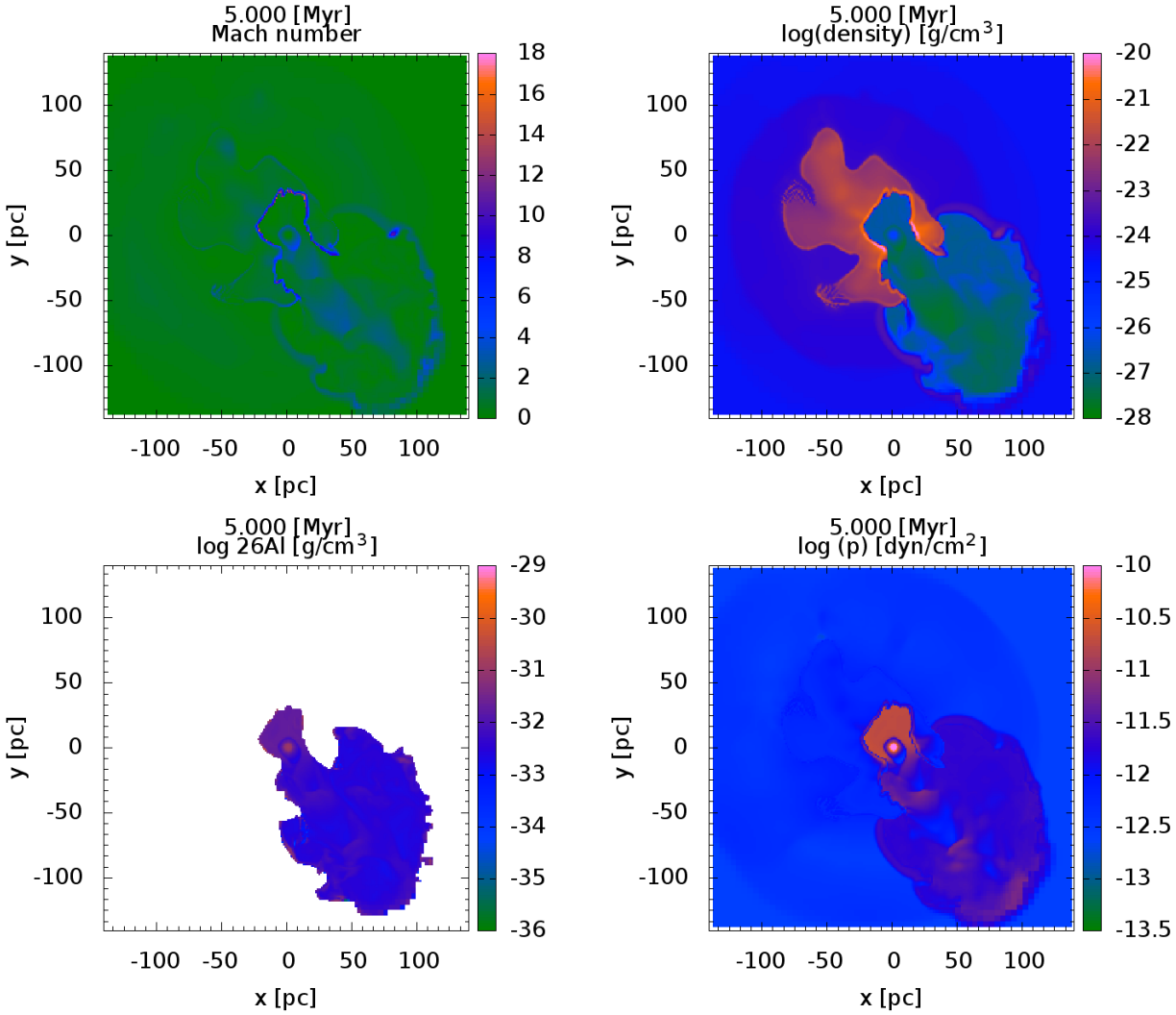


Figure 8.6: These plots show a cut through the SPH cloud, 5 Myr after the stellar feedback started. We see the sonic point at the smallest "chimney" cross section. This leads to an overpressure in the cavity. Also some the flux of ^{26}Al out of the cavity is limited by the speed of sound. The online material contains a movie of artificial observations of this model.

Chapter 9

Discussion and Conclusions

The motivation for this work were the puzzling ^{26}Al data from the Orion-Eridanus region. The favored de-projection of the observational evidence back in 2008 was based on the model suggested by Burrows et al. (1993) for the Orion-Eridanus Superbubble. A version, which was slightly adapted to new observational evidence, is shown in Fig. 2.5. It was unclear, why a banana-shaped superbubble like the one suggested for the OES would form and why ^{26}Al is only observed in a part of the region with X-ray emission (Fig. 2.6 and 2.7). Actually, numerical studies like the simulations of de Avillez and Breitschwerdt (2005) show, that superbubbles can come in a number of peculiar shapes. In non-quiescent surroundings the bubble shape follows the density and pressure gradients the superbubble shell encounters. However, the spread of ^{26}Al was a real puzzle since the gas velocities inside the superbubbles should be high enough to spread ^{26}Al allover the superbubble. The question was, whether the shape of the OES can be a real quirk of nature. In the mean time, the region has been successfully modeled by Pon et al. (2014a), using models based on Kompaneets (1960) assuming a stratified, but quiescent ambient medium. However, it is still debated, if a single bubble model or a two bubble model is to be preferred for the OES.

It turned out, that there is no simple explanation, how the assumed peculiar shape of the OES follows naturally from the stellar feedback of the Orion OB I associations in a quiescent ambient medium. Our simulations used stellar feedback based on population synthesis models, which Voss et al. (2010) tuned to the Orion OB I associations. These OB associations are expected to have formed one after the other with a few million years delay and are expected to have participated in forming the OES. The Voss et al. (2010) feedback model, based on observed stars plus an estimate of the exploded stars via the IMF, turned out to be so disruptive that molecular clouds of sizes as they are found in GMC surveys (see Sect. 2.5) were quite efficiently destroyed by the first OB association already. We were thus faced with the problem, that we either need extremely massive GMCs or an efficient energy sink for the stellar feedback. Otherwise the problem can only be solved with four generations of GMCs: Individual, newly formed GMCs for each of the four OB associations.

We thus decided to take a step back and to start from simple, homogeneous toy models and gradually add complexity. Since our GMCs had a hard time to survive the stellar feedback, we decided that we had to understand the feedback energy efficiency first. Since we need cold, dense GMC gas for the later episodes of star formation, we also checked, how the stellar feedback affects the mass fractions in the ISM. This is interesting, since GMC lifetimes are debated. Whereas the detection of inter-arm GMCs (e.g. Scoville et al., 1979; Koda et al., 2009, inter-arm crossing times $\sim$ 100 Myr) and observations of extragalactic GMCs seem to point to GMC lifetimes of

$20 - 30$ Myr (Kawamura et al., 2009) of which 7 Myr are after the onset of stellar evolution, the lifetimes of local GMCs is expected to be much shorter (e.g. Elmegreen, 2000b; Hartmann et al., 2001, expect immediate star formation and GMC lifetimes < 10 Myr). In comparison, stellar feedback from the Orion OB I associations is assumed to be ongoing since $8 - 12$ Myr (Voss et al., 2010). Our simulations favor the scenario of transient GMCs that are reshuffled by stellar feedback and form again in zones of colliding flows (for recent work on the formation of molecular gas via converging flows see Micic et al., 2013; Ntormousi et al., 2011, and references therein).

One of the conclusions from our spherically symmetric models is, that stellar feedback indeed reshuffles the cold gas. In our models the total thermal energy when the shock velocity has decelerated to the ambient sound speed towards the end of the simulations is lower than in the initial conditions. The net-effect of the stellar feedback is acceleration and compression of the surrounding cloud material. The latter leads to radiative losses.

The other conclusions from the spherically symmetric models are shown in Fig. 6.9 and 6.10: We identify mixing processes across the contact discontinuity as an efficient energy sink. In numerical simulations, we can choose a Riemann solver, which treats the contact discontinuity accurately. But, in the end, the spatial resolution will always lead to mixing of the ambient medium and the stellar ejecta. If the simulation does not take any physical process that leads to stronger mixing than the mixing due to the grid cell size into account, the spatial resolution governs the energy loss at the contact discontinuity. Or to put it the other way around, since we only have a single gas phase per cell, the resolution of our simulations can be interpreted as a proxy for the length scale of the most efficient mixing process. Assuming a mixing length now enables us to find a feedback energy efficiency from Tab. 6.2, Fig. 6.9 or Fig. 6.10. The latter shows the evolution of the feedback energy efficiency as a function of the peak velocity in the swept-up shell. The simulations end when it falls below the ambient sound speed. Thus, if one assumes that the dissipation of the kinetic energy of the shell already happens at higher shell velocities than the ambient sound speed, Fig. 6.10 can be used to find the feedback energy efficiency. If we assume that turbulent mixing acts on scales of 0.004 pc, (which is smaller than the assumed eddy sizes in Gounelle et al., 2009) we find a feedback energy efficiency of roughly two percent. This is less than the often-used value of 10% reported by Thornton et al. (1998, i.e. 10^{50} erg). However, due to the stellar wind, in our case the total energy input is 3.34×10^{51} erg instead of 10^{51} erg, which brings the net amount of retained kinetic energy again closer the often-used value of 10^{50} erg (Thornton et al., 1998).

In our 3D models we explore a different possibility to make the GMCs exist longer: Since the turbulent structure of the ISM produces GMCs that have a sponge like self-similar density structure, we connect the feedback region inside the GMC with a "chimney" to the ambient medium. We show, that this lowers the energy deposition in the GMC (Fig. 7.7 and 7.14). But, since the sound speed limits the flow out of the GMC, the parts of the superbubble inside the cloud can have a higher pressure than the rest of the bubble. In our simulations we see an isentropic flow through the "chimney" that reaches the sonic point at smallest cross section of the "chimney", outside the dense cloud the flow of ejecta continues like an over-expanded flow until it hits the bubble wall and is turned around, leading to a mushroom like bubble shape.

We also placed the stellar feedback in a GMC created in the large scale SPH simulation of Dobbs et al. (2011). As expected, the asymmetries in the initial conditions also produced a peculiar shaped bubble. The first simulations tell us, that ^{26}Al is found near the superbubble's shell in all our models. We did not yet observe bubbles partly filled with ^{26}Al in our grid of simulations. One could thus interpret our results as an indication that there might be some kind of shell between the parts of the OES containing ^{26}Al and the parts which do not. However, we will need a larger set

of models to find fully conclusive evidence for this. Our main reservation in this respect is that averaging in the population synthesis feedback according to Voss et al. (2010), which was used for the models with inhomogeneous SPH clouds, smears out distinct SN events. In our future work we will thus also test models for individual OB associations instead of "averaged OB association" from population synthesis.

To conclude, we found a way to relate the feedback energy efficiency of our spherically symmetric models to a length scale of mixing across the contact discontinuity. To tackle asymmetries in the GMCs, we need to add more dimensions. Our present 3D models are on the edge of reaching realistic estimates of mixing scales (e.g. Stasińska et al., 2007, estimate 1-0.1 pc, which would be resolved in our models) and to be used to estimate feedback energy efficiencies. Moreover, they are not yet customized for the OES. In our future work we plan improve on this and to test turbulent clouds.

Index

Glossary

AMR (adaptive mesh refinement) is a strategy to optimize the resolution and the computational cost during a numerical simulation. If the refinement criteria are fulfilled (e.g. strong density gradient), a cell is subdivided into 2^ν cells, where ν is the number of dimensions in the simulation. 39, 41, 51, 52, 63, 64, 88, 90, 94–96, 127, 134

CFL (Courant-Friedrichs-Lewy condition) maximal stable time-step-size in a hydrodynamical simulation (Sect. 3.3) which ensures that gas cannot travel more than a cell length per time-step. 42, 48, 64, 95, 119, 120

"chimney" toy model for dilute areas connecting a stellar wind bubble or SN remnant, located inside a structured GMC, to the ambient medium. 127–129, 134, 135, 137–145, 148, 153, 156, 167

choked flow situation, in which the sound speed limits the flux through a bottleneck. 142, 143, 145, 167

contact discontinuity (CD) interface between two media with different density but no pressure and velocity gradients across this surface. 5, 6, 8, 9, 17, 45, 46, 49, 56, 60–69, 74, 75, 78, 85, 87, 88, 94, 101, 103, 107, 110, 115–125, 131, 143–145, 156, 157

de-projection converts 2D observational data into a 3D model. 13, 155

downstream direction with respect to the flow. The other direction is called upstream. If we sit on a fluid particle, we have already passed points upstream and will move on to points located downstream. 44, 141, 142, 145

feedback energy energy input into the ISM via stellar winds and SN explosions. 2, 5, 10, 25, 26, 73, 76, 89, 92, 97, 101, 123, 137, 139, 144, 147, 149

feedback energy efficiency (ϵ) describes how much of the energy input via stellar winds and SN explosions can be retained by the ISM (as kinetic energy of a shell). Without cooling: $\epsilon = 1$. 2, 3, 5, 8, 10, 11, 21, 23, 49, 51, 62, 90, 94, 97, 99–101, 104, 106, 107, 110–113, 116, 118–125, 127–129, 132–135, 137, 139, 142–145, 148, 149, 155–157, 167

feedback region (also driver region) part of the computational box in which source terms for stellar mass loss and stellar energy feedback are evaluated. 52, 72, 78–83, 89–94, 96, 101, 102, 104–106, 115, 117, 121, 122, 127–129, 131, 134, 137–143, 145, 147–149, 156, 168, 310–312, 316

fluid element (also fluid particle) volume small compared to the box size but large with respect to intermolecular distances. Macroscopic fluid properties like local density, local velocity are defined over a fluid particle. 38, 39

FORTRAN **derived data type** an object that can group data of different types. It can be handled like any other variable. Elements of derived data types can be accessed with the % operator. 93

GMC (Giant Molecular Cloud) dense phase of the ISM (described in Sect. 2.1). 3–5, 10, 12, 14, 16, 20–23, 97, 99, 101, 127, 128, 137, 147–149, 155–157, 163, 164

IC (initial conditions) setup at the start of a numerical simulation. 39–41, 46, 47, 60, 61, 71, 73, 90, 91, 99, 101, 128, 129, 132–134, 140, 142, 145, 147, 148, 156

IMF (Initial Mass Function) empirical function describing the initial distribution of stellar masses. 22, 23, 26, 32–36, 97, 155

ISM (Interstellar Medium) gas and dust between stars (described in Sect. 2.1). 1, 3–6, 9–13, 16–18, 20, 23, 37, 38, 40, 46, 49, 55, 73–75, 77–79, 82, 83, 87–89, 94, 95, 97, 99–101, 103, 104, 107, 110, 118, 119, 123, 124, 127, 128, 148, 149, 155, 156, 163–165

mass cut the mass coordinate that separates ejected material from material forming the remnant. 26

mean free path (λ) average distance a particle travels before colliding with an other particle [see kinetic theory of gas, e.g. Kennard (1938)]. 6–9, 38, 91, 167

namelist file containing all run-time parameters for a RAMSES simulation. 51, 93–95

OES (Orion-Eridanus Superbubble) a well observed, relatively close by region, which is very well suited to study the interaction of massive stars and the ISM. 2, 3, 12–14, 16, 17, 19, 20, 23, 97, 149, 155–157

pluto.ini file containing all run-time parameters for a PLUTO simulation. 92

porosity in the context of Sect. 7 describes the sum of the cross-sectional areas of all holes in the GMC allowing stellar feedback material to escape from the GMC into the warm phase of the ISM. 21, 127, 128, 135, 145

preprocessor directive contains information on which parts of the code should be compiled. We use e.g. `#define EKIN 1` to compile source code parts inserting the feedback via kinetic energy instead of code parts using thermal feedback energy. Definitions can be removed with `#undef`. Source code parts can be enclosed between constructs like `#ifdef EKIN`, `#else` and `#endif`. 90, 91, 94, 95, 165

Strömgren sphere ionized hydrogen around a massive star. The Strömgren radius can be found from $R_\mathrm{S} \sim \sqrt[3]{\frac{3}{4\pi}\frac{N_{\mathrm{Ly}\alpha}}{3\times10^{-13}n^2}}$ cm, with the number density n in units of cm^{-3} and the number of Lyman continuum photons $N_{\mathrm{Ly}\alpha}$ per second. 12, 73, 128, 129, 132, 134

superbubble cavity in the ISM created by the combined feedback of several massive stars. 2, 3, 5, 12–14, 16, 17, 19, 100, 135, 148, 155, 156

supernova (SN) stellar explosion. In the context of this work we focus on core collapse SNe. These occur when nuclear fusion fails to balance gravity in the core of massive stars. We do not take SN Ia explosions into account in this work, since we do not follow the evolution of the stellar content of our cloud long enough to obtain white dwarfs, which in turn could undergo a SN Ia explosion. x, xiii, 1, 2, 4–6, 9, 11, 12, 22, 23, 26, 27, 32, 33, 35, 49, 64, 65, 73, 76, 88–92, 99–108, 110–120, 122–124, 157, 163, 165–167

time of maximal luminosity (t_0) time, when the largest energy losses due to radiative cooling occur in the simulation. Please note that despite this name it does not correspond to the maximum in the SN light curve, which is caused by radioactive decays. 100–102, 104–107, 113, 115, 124

vector sweep contains a part of the simulation data. RAMSES allows to control the maximal memory allocation within each MPI process. Since the simulation can be too large to fit into the memory at once, the user can specify a vector size with the preprocessor directive `NVECTOR` and the data will be subdivided into arrays of `dimension(1:nvector)`. The default setting is `NVECTOR=500`. Only one of these arrays is loaded into the memory at a time. 93

WR the Wolf-Rayet phase is the last phase in the evolution of a massive star. During this phase the star undergoes extreme mass losses due to very strong winds. 24, 73, 74, 100, 110, 116, 117, 124

Units

List of frequently used units.

distance	
AU	Astronomical Unit, 149597870700 m
cm	10^{-2} m
km	10^{3} m
m	meter
micron	10^{-6} m
pc	parsec, $3.08567758 \times 10^{16}$ m
energy	
FOE	10^{51} erg, canonical supernova energy
GeV	1.6021765710^{-10} Joule
MeV	1.6021765710^{-13} Joule
eV	1.6021765710^{-19} Joule
erg	10^{-7} Joule
keV	1.6021765710^{-16} Joule
meV	1.6021765710^{-22} Joule
flux	
Jy	Jansky, 10^{26} W m^{-2} Hz^{-1}
R	Rayleigh, 10^{10} photons m^{-2} s^{-1}
frequency	
Hz	Hertz, s^{-1}
GHz	10^{9} Hertz
THz	10^{12} Hertz
mass	
g	gram
$M_\odot$	solar mass, 1.9891×10^{33} g
(number) density	
cm^{-3}	particles cm^{-3} , number density
g cm^{-3}	mass density
temperature	
K	Kelvin
mK	10^{-3} K
time	
fs	10^{-15} seconds
s	second
yr	year
kyr	10^{3} years
Myr	10^{6} years
velocity	
cm s^{-1}	0.01 meter per second
km s^{-1}	kilometer per second
pc Myr^{-1}	parsec per million years (~ 0.978 km s^{-1})

Symbols

List of frequently used symbols.

ø	diameter
a	factor for the cooling floor (at $a\rho_0$) or exponent a in Eq. 4.40
A	cross section
$A_{\rm crit}$	critical cross section for choked flows
$\vec{B}$	magnetic field
b	Galactic latitude
c	wave speed
c_p, c_V	specific heat capacity (per particle)
$c_{\rm s}$	speed of sound
$c_{\rm s,iso}$	isothermal speed of sound
D	diffusion coefficient or total derivative
Δd	"chimney" diameter (used in Sect. 7)
Δt	time step size
Δx	cell size or "chimney" length
$\dot{E}$	energy loss/gain
E	energy
$E_{\rm SN}$	supernova energy input (10^{51} erg)
$E_{\rm kin}$	kinetic energy
$E_{\rm therm}$	thermal energy
e	electron charge
$e_{\rm in}$	internal energy
ϵ	feedback energy efficiency
$\epsilon_{\rm k}$	kinetic feedback energy efficiency
$\epsilon_{\rm t}$	thermal feedback energy efficiency
f	degree of freedom
F	flux
$F_{\rm c}$	heat flux
$F_{\rm sat}$	saturated heat flux
$\vec{F}(\vec{U})$	flux vector
γ	adiabatic exponent
Γ	Gamma function (in Sect. 4.4.1), diffusion coefficient (in Sect. 3.4) or heating rate (all other Sect.)
$\vec{J}$	Jacobian
k	wave number
k or $k_{\rm B}$	Boltzmann constant ($1.3806488(13) \times 10^{-16}$ erg K^{-1})
κ	heat conduction coefficient
Λ	Coulomb logarithm (in Sect. 2.2.1) or cooling rate (all other Sect.)
λ	mean free path
λ_i	i$^{\rm th}$ Eigenvalue
L	scale length
$l_{\rm T}$	scale length of the temperature gradient
$L_{\rm w}$	kinetic wind luminosity $L_{\rm w} = 0.5\dot{M}v_\infty^2$
l	Galactic longitude
$\dot{M}$	mass loss rate
M	Mach number or Mass
$M_\odot$	solar mass, 1.9891×10^{33} g
$m_{\rm H}$	hydrogen mass
$\mu_{\rm mol}$	molar mass

$\dot{n}$	
n	number density (unit: cm^{-3})
N	number of particles in the EOS
n_0	number density of the ambient medium
n_H	hydrogen number density
ν	number of dimensions
ω	angular frequency
p	pressure
ϕ	angle
Φ	general flow quantity
R	gas constant $8.314 \times 10^7\,\mathrm{erg\,K^{-1}mol^{-1}}$ or radius
r	radial coordinate or radius
r_c	cavity radius
r_f or r_fb	feedback region radius
r_shell	shell radius
$\dot{R}$	shell velocity (i.e. bubble radius change)
ρ	density
ρ_0	ambient density
Σ	surface density
σ	standard deviation, velocity dispersion or cross section
S_Φ	source terms
S_ν	coefficients for the surface of an ν dimensional sphere
T	temperature
T_eq	temperature of the cooling-heating equilibrium for a given number density
T_0	temperature of the ambient medium
t, τ	time
$\tau_{1/2}$	half life time
θ	angle
t_0	time of maximal luminosity (see page 101)
u	velocity or component of the vector of system properties
U	system properties (e.g. density, flow velocity and pressure)
$\vec{U}$	vector of conservative variables (ρ,$\rho\vec{v}$,E)
$\bar{v}$	average velocity
v_rms	rms-velocity
v or $\vec{v}$	velocity
v_∞	terminal wind velocity
V	volume
V_ν	coefficients for the volume of an ν dimensional sphere
$\mathrm{d}V$	volume change
$\vec{W}$	vector of primitive variables (ρ,$\vec{v}$,P)
$\xi(k)$ amplification factor	
x or $\vec{x}$	position
X	Hydrogen mass fraction
Y	Helium mass fraction
Z	metallicity, mass fraction of all elements except H and He ($Z = 1 - X - Y$)
$Z_\odot$	solar metallicity

Bibliography

Abbott, D. C.: 1982, *ApJ* **263**, 723

Arthur, S. J. and Henney, W. J.: 1996, *ApJ* **457**, 752

Atwood, W. B., Abdo, A. A., Ackermann, M., Althouse, W., Anderson, B., Axelsson, M., Baldini, L., Ballet, J., Band, D. L., Barbiellini, G., and et al.: 2009, *ApJ* **697**, 1071

Avedisova, V. S.: 1972, *Soviet Astronomy* **15**, 708

Balbus, S. A. and McKee, C. F.: 1982, *ApJ* **252**, 529

Ballone, A., Schartmann, M., Burkert, A., Gillessen, S., Genzel, R., Fritz, T. K., Eisenhauer, F., Pfuhl, O., and Ott, T.: 2013, *ApJ* **776**, 13

Bandiera, R. and Petruk, O.: 2004, *A&A* **419**, 419

Beaumont, C. N., Goodman, A. A., Kendrew, S., Williams, J. P., and Simpson, R.: 2014, *ApJS* **214**, 3

Behrendt, M.: 2011, Diplomarbeit, LMU, Department of Physics

Blitz, L.: 1993, in E. H. Levy & J. I. Lunine (ed.), *Protostars and Planets III*, pp 125–161

Blitz, L., Fukui, Y., Kawamura, A., Leroy, A., Mizuno, N., and Rosolowsky, E.: 2007, *Protostars and Planets V* pp 81–96

Bloemen, H., Morris, D., Knödlseder, J., Bennett, K., Diehl, R., Hermsen, W., Lichti, G., van der Meulen, R. D., Oberlack, U., Ryan, J., Schönfelder, V., Strong, A. W., de Vries, C., and Winkler, C.: 1999, *ApJ Lett.* **521**, L137

Bolatto, A. D., Wolfire, M., and Leroy, A. K.: 2013, *ARA&A* **51**, 207

Bonnarel, F., Fernique, P., Bienaymé, O., Egret, D., Genova, F., Louys, M., Ochsenbein, F., Wenger, M., and Bartlett, J. G.: 2000, *Astron. Astrophys. Suppl. Ser.* **143**, 33

Boss, A. P. and Keiser, S. A.: 2012, *ApJ Lett.* **756**, L9

Boumis, P., Dickinson, C., Meaburn, J., Goudis, C. D., Christopoulou, P. E., López, J. A., Bryce, M., and Redman, M. P.: 2001, *MNRAS* **320**, 61

Brighenti, F. and D'Ercole, A.: 1994, *MNRAS* **270**, 65

Brown, A. G. A., de Bruijne, J. H. J., Hoogerwerf, R., de Zeeuw, P. T., and Blaauw, A.: 2000, in F. Favata, A. Kaas, & A. Wilson (ed.), *Star Formation from the Small to the Large Scale*, Vol. 445 of *ESA Special Publication*, pp 239–+

Brown, A. G. A., de Geus, E. J., and de Zeeuw, P. T.: 1994, *A&A* **289**, 101

Brown, A. G. A., Hartmann, D., and Burton, W. B.: 1995, *A&A* **300**, 903

Burkert, A., Schartmann, M., Alig, C., Gillessen, S., Genzel, R., Fritz, T. K., and Eisenhauer, F.: 2012, *ApJ* **750**, 58

Burrows, D. N. and Guo, Z.: 1996, in H. U. Zimmermann, J. Trümper, and H. Yorke (eds.), *Roentgenstrahlung from the Universe*, p. 221

Burrows, D. N. and Mendenhall, J. A.: 1991, *Nature* **351**, 629

Burrows, D. N., Singh, K. P., Nousek, J. A., Garmire, G. P., and Good, J.: 1993, *ApJ* **406**, 97

Castor, J., McCray, R., and Weaver, R.: 1975, *Astrophysical Journal* **200**, L107

Castro, D., Slane, P. O., Gaensler, B. M., Hughes, J. P., and Patnaude, D. J.: 2011, *ApJ* **734**, 86

Cen, R.: 1992, *ApJS* **78**, 341

Chevalier, R. A.: 1974, *ApJ* **188**, 501

Chevalier, R. A. and Clegg, A. W.: 1985, *Nature* **317**, 44

Chevalier, R. A. and Imamura, J. N.: 1982, *ApJ* **261**, 543

Chu, Y.-H. and Mac Low, M.-M.: 1990, *ApJ* **365**, 510

Clark, B. G.: 1965, *ApJ* **142**, 1398

Courant, R., Friedrichs, K., and Lewy, H.

Cox, D. P.: 2005, *ARA&A* **43**, 337

Crutcher, R. M.: 2012, *ARA&A* **50**, 29

Dale, J. E. and Bonnell, I.: 2011, *MNRAS* **414**, 321

Dame, T. M., Hartmann, D., and Thaddeus, P.: 2001, *ApJ* **547**, 792

de Avillez, M. A. and Breitschwerdt, D.: 2005, *A&A* **436**, 585

de Avillez, M. A. and Breitschwerdt, D.: 2012, *The Astrophysical Journal Letters* **761(2)**, L19

de Zeeuw, P. T., Hoogerwerf, R., de Bruijne, J. H. J., Brown, A. G. A., and Blaauw, A.: 1999, *AJ* **117**, 354

Dickey, J. M. and Lockman, F. J.: 1990, *ARA&A* **28**, 215

Diehl, R.: 2002, *New Astronomy Review* **46**, 547

Diehl, R.: 2014, in S. Jeong, N. Imai, H. Miyatake, and T. Kajino (eds.), *American Institute of Physics Conference Series*, Vol. 1594 of *American Institute of Physics Conference Series*, pp 109–116

Diehl, R., Cerviño, M., Hartmann, D. H., and Kretschmer, K.: 2004, *New Astronomy Review* **48**, 81

Diehl, R., Kretschmer, K., Plüschke, S., Cerviño, M., and Hartmann, D. H.: 2003, in K. van der Hucht, A. Herrero, & C. Esteban (ed.), *A Massive Star Odyssey: From Main Sequence to Supernova*, Vol. 212 of *IAU Symposium*, pp 706–+

Dobbs, C. L., Burkert, A., and Pringle, J. E.: 2011, *MNRAS* **413**, 2935

Dyson, J. E.: 1977, *A&A* **59**, 161

Einfeldt, B.: 1988, *SIAM Journal on Numerical Analysis* **25**, 294

Ekström, S., Georgy, C., Eggenberger, P., Meynet, G., Mowlavi, N., Wyttenbach, A., Granada, A., Decressin, T., Hirschi, R., Frischknecht, U., Charbonnel, C., and Maeder, A.: 2012, *A&A* **537**, A146

Elmegreen, B. G.: 2000a, *ApJ* **539**, 342

Elmegreen, B. G.: 2000b, *ApJ* **530**, 277

Field, G. B.: 1965, *ApJ* **142**, 531

Field, G. B., Goldsmith, D. W., and Habing, H. J.: 1969, *ApJ Lett.* **155**, L149+

Finkbeiner, D. P.: 2003, *ApJS* **146**, 407

Gardiner, T. A. and Stone, J. M.: 2005, *Journal of Computational Physics* **205**, 509

Gardiner, T. A. and Stone, J. M.: 2008, *Journal of Computational Physics* **227**, 4123

Gent, F. A., Shukurov, A., Fletcher, A., Sarson, G. R., and Mantere, M. J.: 2013, *MNRAS* **432**, 1396

Genzel, R. and Stutzki, J.: 1989, *ARA&A* **27**, 41

Godard, B., Falgarone, E., and Pineau Des Forêts, G.: 2009, *A&A* **495**, 847

Gounelle, M., Meibom, A., Hennebelle, P., and Inutsuka, S.-i.: 2009, *ApJ Lett.* **694**, L1

Gounelle, M. and Meynet, G.: 2012, *A&A* **545**, A4

Gruendl, R. A., Chu, Y.-H., Dunne, B. C., and Points, S. D.: 2000, *AJ* **120**, 2670

Guo, Z. and Burrows, D. N.: 1996, in *Bulletin of the American Astronomical Society*, Vol. 28 of *Bulletin of the American Astronomical Society*, p. 834

Guo, Z., Burrows, D. N., Sanders, W. T., Snowden, S. L., and Penprase, B. E.: 1995, *Astrophysical Journal* **453**, 256

Haque, A.: 2006, *Sedov Analytical Solution Program*, http://flash.uchicago.edu/~ahaque/sedov.html, Online since 27.11.2006; accessed: 11.05.2009

Harten, A., Lax, P., and Leer, B.: 1983, *SIAM Review* **25(1)**, 35

Hartmann, L., Ballesteros-Paredes, J., and Bergin, E. A.: 2001, *ApJ* **562**, 852

Haslam, C. G. T., Salter, C. J., Stoffel, H., and Wilson, W. E.: 1982, *Astron. Astrophys. Suppl. Ser.* **47**, 1

Heiles, C., Haffner, L. M., and Reynolds, R. J.: 1999, in A. R. Taylor, T. L. Landecker, and G. Joncas (eds.), *New Perspectives on the Interstellar Medium*, Vol. 168 of *Astronomical Society of the Pacific Conference Series*, p. 211

Hennebelle, P. and Audit, E.: 2007, *A&A* **465**, 431

Heyer, M., Krawczyk, C., Duval, J., and Jackson, J. M.: 2009, *ApJ* **699**, 1092

Hill, A. S., Joung, M. R., Low, M.-M. M., Benjamin, R. A., Haffner, L. M., Klingenberg, C., and Waagan, K.: 2012, *The Astrophysical Journal* **761(2)**, 189

Hill, A. S., Joung, M. R., Mac Low, M.-M., Benjamin, R. A., Haffner, L. M., Klingenberg, C., and Waagan, K.: 2012, *ApJ* **750**, 104

Ianjamasimanana, R., de Blok, W. J. G., Walter, F., and Heald, G. H.: 2012, *AJ* **144**, 96

IceCube Collaboration, Aartsen, M. G., Abbasi, R., Abdou, Y., Ackermann, M., Adams, J., Aguilar, J. A., Ahlers, M., Altmann, D., Auffenberg, J., and et al.: 2013, *ArXiv e-prints*

Jijina, J., Myers, P. C., and Adams, F. C.: 1999, *ApJS* **125**, 161

Jo, Y.-S., Min, K.-W., Seon, K.-I., Edelstein, J., and Han, W.: 2011, *ApJ* **738**, 91

Joung, M. K. R. and Mac Low, M.-M.: 2006, *ApJ* **653**, 1266

Junk, V., Walch, S., Heitsch, F., Burkert, A., Wetzstein, M., Schartmann, M., and Price, D.: 2010, *MNRAS* **407**, 1933

Kalberla, P. M. W. and Kerp, J.: 2009, *ARA&A* **47**, 27

Kawamura, A., Mizuno, Y., Minamidani, T., Filipović, M. D., Staveley-Smith, L., Kim, S., Mizuno, N., Onishi, T., Mizuno, A., and Fukui, Y.: 2009, *ApJS* **184**, 1

Kendrew, S., Simpson, R., Bressert, E., Povich, M. S., Sherman, R., Lintott, C. J., Robitaille, T. P., Schawinski, K., and Wolf-Chase, G.: 2012, *ApJ* **755**, 71

Kennard, E.: 1938, *Kinetic theory of gases: with an introduction to statistical mechanics*, International series in pure and applied physics, McGraw-Hill

Koda, J., Scoville, N., Sawada, T., La Vigne, M. A., Vogel, S. N., Potts, A. E., Carpenter, J. M., Corder, S. A., Wright, M. C. H., White, S. M., Zauderer, B. A., Patience, J., Sargent, A. I., Bock, D. C. J., Hawkins, D., Hodges, M., Kemball, A., Lamb, J. W., Plambeck, R. L., Pound, M. W., Scott, S. L., Teuben, P., and Woody, D. P.: 2009, *ApJ Lett.* **700**, L132

Kompaneets, A. S.: 1960, *Soviet Physics Doklady* **5**, 46

Korpi, M. J., Brandenburg, A., Shukurov, A., Tuominen, I., and Nordlund, A&A.: 1999, *ApJ Lett.* **514**, L99

Koyama, H. and Ostriker, E. C.: 2009, *ApJ* **693**, 1316

Krause, M., Diehl, R., Böhringer, H., Freyberg, M., and Lubos, D.: 2014, *A&A* **566**, A94

Krause, M., Fierlinger, K., Diehl, R., Burkert, A., Voss, R., and Ziegler, U.: 2012, *ArXiv e-prints*

Krause, M. G. H. and Diehl, R.: 2014, *ArXiv e-prints*

Kregenow, J., Edelstein, J., Korpela, E. J., Welsh, B. Y., Heiles, C., Ryu, K., Min, K.-W., Lim, Y., Yuk, I.-S., Jin, H., and Seon, K.-I.: 2006, *ApJ Lett.* **644**, L167

Kretschmer, K., Diehl, R., Krause, M., Burkert, A., Fierlinger, K., Gerhard, O., Greiner, J., and Wang, W.: 2013, *A&A* **559**, A99

Kritsuk, A. G., Lee, C. T., and Norman, M. L.: 2013, *MNRAS* **436**, 3247

Kroupa, P.: 2001, *MNRAS* **322**, 231

Lamers, H. J. G. L. M. and Cassinelli, J. P.: 1999, *Introduction to Stellar Winds*

Lamers, H. J. G. L. M., Snow, T. P., and Lindholm, D. M.: 1995, *ApJ* **455**, 269

Larson, R. B.: 1981, *MNRAS* **194**, 809

Leitherer, C., Schaerer, D., Goldader, J. D., González Delgado, R. M., Robert, C., Kune, D. F., de Mello, D. F., Devost, D., and Heckman, T. M.: 1999, *ApJS* **123**, 3

LeVeque, R.: 2002, *Finite Volume Methods for Hyperbolic Problems*, Cambridge Texts in Applied Mathematics, Cambridge University Press

Limongi, M. and Chieffi, A.: 2006, *ApJ* **647**, 483

Lodders, K.: 2003, *ApJ* **591**, 1220

McKee, C. F.: 1990, in L. Blitz (ed.), *The Evolution of the Interstellar Medium*, Vol. 12 of *Astronomical Society of the Pacific Conference Series*, pp 3–29

McKee, C. F. and Ostriker, J. P.: 1977, *ApJ* **218**, 148

Mel'Nik, A. M. and Efremov, Y. N.: 1995, *Astronomy Letters* **21**, 10

Meynet, G. and Maeder, A.: 2003, *A&A* **404**, 975

Meynet, G. and Maeder, A.: 2005, *A&A* **429**, 581

Meynet, G., Maeder, A., Schaller, G., Schaerer, D., and Charbonnel, C.: 1994, *Astron. Astrophys. Suppl. Ser.* **103**, 97

Micic, M., Glover, S. C. O., Banerjee, R., and Klessen, R. S.: 2013, *MNRAS* **432**, 626

Mignone, A., Bodo, G., Massaglia, S., Matsakos, T., Tesileanu, O., Zanni, C., and Ferrari, A.: 2007, *ApJS* **170**, 228

Mignone, A., Zanni, C., Tzeferacos, P., van Straalen, B., Colella, P., and Bodo, G.: 2012, *ApJS* **198**, 7

Miville-Deschênes, M.-A. and Lagache, G.: 2005, *ApJS* **157**, 302

Miville-Deschenes, M.-A. and Lagache, G.: 2008, *IRIS - Improved Reprocessing of the IRAS Survey*, http://www.cita.utoronto.ca/~mamd/IRIS/, last modified 9.12.2004, accessed: 10.09.2014

Moeckel, N. and Burkert, A.: 2014, *ArXiv e-prints*

Mouschovias, T. C.: 1987, in G. E. Morfill and M. Scholer (eds.), *NATO ASIC Proc. 210: Physical Processes in Interstellar Clouds*, pp 491–552

Murray, C. E., Lindner, R. R., Stanimirović, S., Goss, W. M., Heiles, C., Dickey, J., Pingel, N. M., Lawrence, A., Jencson, J., Babler, B. L., and Hennebelle, P.: 2014, *ApJ Lett.* **781**, L41

Murray, N.: 2011, *ApJ* **729**, 133

Netz, H.: 1986, *Netz Formeln der Mathematik, neu bearb. von J. Rast*, Hanser, 6 edition

Niedzielski, A. and Skorzynski, W.: 2002, *Acta Astron.* **52**, 81

Ntormousi, E., Burkert, A., Fierlinger, K., and Heitsch, F.: 2011, *ApJ* **731**, 13

Oey, M. S.: 1996, *ApJ* **467**, 666

Oey, M. S.: 2005, in *American Astronomical Society Meeting Abstracts*, Vol. 37 of *Bulletin of the American Astronomical Society*, p. 1243

Oey, M. S. and Massey, P.: 1994, *ApJ* **425**, 635

Okumura, A., Kamae, T., and for the Fermi LAT Collaboration: 2009, *ArXiv e-prints*

Ostriker, J. P. and McKee, C. F.: 1988, *Reviews of Modern Physics* **60**, 1

Palacios, A., Meynet, G., Vuissoz, C., Knödlseder, J., Schaerer, D., Cerviño, M., and Mowlavi, N.: 2005, *A&A* **429**, 613

Pan, L., Desch, S. J., Scannapieco, E., and Timmes, F. X.: 2012, *ApJ* **756**, 102

Parizot, E. M. G.: 1998, *A&A* **331**, 726

Pikel'Ner, S. B.: 1968, *Astrophys. Lett.* **2**, 97

Planck Team: 2013, *Planck*, http://planck.caltech.edu/publications2013Results.html, last modified 21.3.2013 , accessed: 10.09.2014

Pon, A., Johnstone, D., Bally, J., and Heiles, C.: 2014a, *ArXiv e-prints*

Pon, A., Johnstone, D., Bally, J., and Heiles, C.: 2014b, *MNRAS* **441**, 1095

Pon, A., Johnstone, D., and Kaufman, M. J.: 2012, *ApJ* **748**, 25

Pon, A. R.: 2013, *Ph.D. thesis*, University of Victoria (Canada

Project, L. I.: 2004, *WWW Table of Radioactive Isotopes*, http://ie.lbl.gov/toi, last accessed Oct., 11th 2014

Raymond, J. C., Cox, D. P., and Smith, B. W.: 1976, *ApJ* **204**, 290

Reich, P. and Reich, W.: 1986, *Astron. Astrophys. Suppl. Ser.* **63**, 205

Reich, W.: 1982, *Astron. Astrophys. Suppl. Ser.* **48**, 219

Reynolds, R. J. and Ogden, P. M.: 1979, *ApJ* **229**, 942

Roe, P. L.: 1981, *Journal of Computational Physics* **43**, 357

Roe, P. L.: 1986, *Annual Review of Fluid Mechanics* **18**, 337

Roman-Duval, J., Jackson, J. M., Heyer, M., Rathborne, J., and Simon, R.: 2010, *ApJ* **723**, 492

Roques, J. P., Schanne, S., von Kienlin, A., Knödlseder, J., Briet, R., Bouchet, L., Paul, P., Boggs, S., Caraveo, P., Cassé, M., Cordier, B., Diehl, R., Durouchoux, P., Jean, P., Leleux, P., Lichti, G., Mandrou, P., Matteson, J., Sanchez, F., Schönfelder, V., Skinner, G., Strong, A., Teegarden, B., Vedrenne, G., von Ballmoos, P., and Wunderer, C.: 2003, *A&A* **411**, L91

Rozyczka, M., Tenorio-Tagle, G., Franco, J., and Bodenheimer, P.: 1993, *MNRAS* **261**, 674

Ryu, K., Min, K.-W., Park, J.-W., Lee, D.-H., Han, W., Nam, U.-W., Park, J.-H., Edelstein, J., Korpela, E. J., Nishikida, K., and van Dishoeck, E. F.: 2006, *ApJ Lett.* **644**, L185

Ryu, K., Min, K. W., Seon, K.-I., Nonesa, J., Edelstein, J., Korpela, E., Sankrit, R., Han, W., Park, J.-H., and Park, Y. S.: 2008, *ApJ Lett.* **678**, L29

Salpeter, E. E.: 1955, *ApJ* **121**, 161

Schartmann, M., Burkert, A., Alig, C., Gillessen, S., Genzel, R., Eisenhauer, F., and Fritz, T. K.: 2012, *ApJ* **755**, 155

Schartmann, M., Burkert, A., Krause, M., Camenzind, M., Meisenheimer, K., and Davies, R. I.: 2010, *MNRAS* **403**, 1801

Schartmann, M., Krause, M., and Burkert, A.: 2011, *MNRAS* **415**, 741

Schartmann, M., Meisenheimer, K., Klahr, H., Camenzind, M., Wolf, S., and Henning, T.: 2009, *MNRAS* **393**, 759

Schönfelder, V.: 2001, *The Universe in Gamma Rays*, Astronomy and Astrophysics Library, Springer

Scoville, N. Z., Solomon, P. M., and Sanders, D. B.: 1979, in W. B. Burton (ed.), *The Large-Scale Characteristics of the Galaxy*, Vol. 84 of *IAU Symposium*, pp 277–282

Sedov, L. I.: 1993, *Similarity and Dimensional Methods in Mechanics, 10th edition*, CRC Press

Shu, F. H.: 1992, *The Physics of Astrophysics, Volume II, Gas dynamics*, Vol. II of *The Physics of Astrophysics*, University Science Books

Simpson, R. J., Povich, M. S., Kendrew, S., Lintott, C. J., Bressert, E., Arvidsson, K., Cyganowski, C., Maddison, S., Schawinski, K., Sherman, R., Smith, A. M., and Wolf-Chase, G.: 2012, *VizieR Online Data Catalog* **742**, 42442

Smith, L. F. and Maeder, A.: 1991, *A&A* **241**, 77

Smith, M. D. and Rosen, A.: 2003, *MNRAS* **339**, 133

Snowden, S. L., Burrows, D. N., Sanders, W. T., Aschenbach, B., and Pfeffermann, E.: 1995, *Astrophysical Journal* **439**, 399

Snowden, S. L., Egger, R., Freyberg, M. J., McCammon, D., Plucinsky, P. P., Sanders, W. T., Schmitt, J. H. M. M., Truemper, J., and Voges, W.: 1997, *ApJ* **485**, 125

Sod, G. A.: 1978, *Journal of Computational Physics* **27**, 1

Sonbas, E., Moskvitin, A. S., Fatkhullin, T. A., Sokolov, V. V., Castro-Tirado, A., de Ugarte Postigo, A., Gorosabel, G., Guzij, S., Jelinec, M., Sokolova, T. N., and Chernenkov, V. N.: 2008, *Astrophysical Bulletin* **63**, 228

Spitzer, L.: 1956, *Physics of Fully Ionized Gases*

Spitzer, L.: 1962, *Physics of Fully Ionized Gases*

Springel, V.: 2010, *MNRAS* **401**, 791

Stasińska, G., Tenorio-Tagle, G., Rodríguez, M., and Henney, W. J.: 2007, *A&A* **471**, 193

Sternberg, A., Hoffmann, T. L., and Pauldrach, A. W. A.: 2003, *ApJ* **599**, 1333

Stone, J. M., Gardiner, T. A., Teuben, P., Hawley, J. F., and Simon, J. B.: 2008, *ApJS* **178**, 137

Stone, J. M., Gardiner, T. A., Teuben, P., Hawley, J. F., and Simon, J. B.: 2010, *Athena: Grid-based code for astrophysical magnetohydrodynamics (MHD)*, Astrophysics Source Code Library

Strong, A.: 1994, *CGRO Compton Telescope: 3 channel data*, http://skyview.gsfc.nasa.gov, May 1991 to October 1994, accessed: 10.09.2014

Sutherland, R. S. and Dopita, M. A.: 1993, *ApJS* **88**, 253

Sweby, P. K.: 1984, *SIAM Journal on Numerical Analysis* **21(5)**, pp995

Tan, J. C., Shaske, S. N., and Van Loo, S.: 2013, in T. Wong and J. Ott (eds.), *IAU Symposium*, Vol. 292 of *IAU Symposium*, pp 19–28

Taylor, G.: 1950, *Royal Society of London Proceedings Series A* **201**, 159

Tenorio-Tagle, G.: 1996, *AJ* **111**, 1641

Tenorio-Tagle, G., Bodenheimer, P., Franco, J., and Rozyczka, M.: 1990, *MNRAS* **244**, 563

Tenorio-Tagle, G., Rozyczka, M., Franco, J., and Bodenheimer, P.: 1991, *MNRAS* **251**, 318

Teyssier, R.: 2002, *A&A* **385**, 337

Theuns, T., Leonard, A., Efstathiou, G., Pearce, F. R., and Thomas, P. A.: 1998, *MNRAS* **301**, 478

Thornton, K., Gaudlitz, M., Janka, H.-T., and Steinmetz, M.: 1998, *ApJ* **500**, 95

Toro, E. F., Spruce, M., and Speares, W.: 1994, *Shock Waves* **4**, 25

van Leer, B.: 1977, *Journal of Computational Physics* **23**, 276

Vedrenne, G., Roques, J.-P., Schönfelder, V., Mandrou, P., Lichti, G. G., von Kienlin, A., Cordier, B., Schanne, S., Knödlseder, J., Skinner, G., Jean, P., Sanchez, F., Caraveo, P., Teegarden, B., von Ballmoos, P., Bouchet, L., Paul, P., Matteson, J., Boggs, S., Wunderer, C., Leleux, P., Weidenspointner, G., Durouchoux, P., Diehl, R., Strong, A., Cassé, M., Clair, M. A., and André, Y.: 2003, *A&A* **411**, L63

Vishniac, E. T.: 1994, *ApJ* **428**, 186

Von Neumann, J.: 1963, *Theory of games, astrophysics, hydrodynamics and meteorology*, Vol. 6 of *Collected Works*, Pergamon Press

Voss, R., Diehl, R., Hartmann, D. H., Cerviño, M., Vink, J. S., Meynet, G., Limongi, M., and Chieffi, A.: 2009, *A&A* **504**, 531

Voss, R., Diehl, R., Vink, J. S., and Hartmann, D. H.: 2010, *A&A* **520**, A51+

Wampler, E. J., Wang, L., Baade, D., Banse, K., D'Odorico, S., Gouiffes, C., and Tarenghi, M.: 1990, *ApJ Lett.* **362**, L13

Weaver, R., McCray, R., Castor, J., Shapiro, P., and Moore, R.: 1977, *ApJ* **218**, 377

Weidner, C., Kroupa, P., and Bonnell, I. A. D.: 2010, *MNRAS* **401**, 275

Weidner, C., Kroupa, P., and Pflamm-Altenburg, J.: 2013, *MNRAS* **434**, 84

Williams, J. P., Blitz, L., and McKee, C. F.: 2000, *Protostars and Planets IV* p. 97

Williams, J. P. and McKee, C. F.: 1997, *ApJ* **476**, 166

Wilson, B. A., Dame, T. M., Masheder, M. R. W., and Thaddeus, P.: 2005, *A&A* **430**, 523

Danksagung

Ich möchte mich bei allen Personen bedanken, die bei der Entstehung dieser Arbeit direkt und indirekt mitgeholfen haben. Besonders bedanken möchte ich mich bei meinen Betreuern Andreas Burkert und Roland Diehl, für das Vertrauen, mir dieses spannende Projekt zu übertragen, die Geduld, die vielen Tipps, die Unterstützung dabei, meine Arbeit bei lokalen sowie internationalen Treffen zu präsentieren und die gute Betreuung. Weiters hätte es diese Arbeit ohne Stephan Paul, der mich mit Roland und Andi in Kontakt gebracht hat, nicht gegeben. Und sie wäre nun wohl auch nicht fertig, wenn er mir nicht nahegelegt hätte, die Arbeit baldigst einzureichen. Für die selbe Mischung an Vertrauen und sanften Hinweisen zur Effizienzsteigerung (wie bei dem ineffizienten Energieinput der massereichen Sterne ins ISM, der in dieser Arbeit diskutiert wird, sind auch bei mir wohl $> 90\%$ des Aufwandes ohne nachhaltigen Effekt verpufft) sowie für die kontinuierliche Versicherung, dass sie daran glauben, dass ich das Projekt Doktorarbeit irgendwann abschließen kann, danke ich meinem Mann Peter, meinen Eltern und Schwiegereltern, meinen Schwestern und all meinen Verwandten und Freunden. Ich bedanke mich bei ihnen auch für den sanften Druck, mich nicht in ein beschauliches Hausfrauenleben zu verabschieden.

Inhaltlich und methodisch hat diese Arbeit stark davon profitiert, dass ich in die CAST Gruppe an der USM und in den Universe Cluster eingebunden war und das Kursangebot für IMPRS Student/inn/en nutzen durfte. Neben Kolleg/inn/en, die mich direkt in meiner Forschung unterstützt haben, wie Andreas Burkert, Roland Diehl, Rasmus Voss, Eva Ntormousi, Marc Schartmann, Clare Dobbs, Martin Krause, Alessandro Ballone, Judith Ngoumou, Manuel Behrendt, Nickolas Moeckel, Matthias Gritschneder, James Dale, Tadziu Hoffmann und Klaus Dolag möchte ich auch allen anderen derzeitigen und ehemaligen Mitgliedern der CAST Gruppe – stellvertretend seien Rhea-Silvia Remus und Martin Zintl genannt – sowie des Universe Clusters danken. Auch falls wir vielleicht bisher nicht an gemeinsamen Projekten gearbeitet haben, haben mich ihre spannenden Ergebnisse dennoch immer wieder daran erinnert, wie viele tolle Dinge wir erforschen können.

Diese Arbeit hat sehr davon profitiert, dass Marianne Busch und Peter als Testleser/in mich dazu bewogen haben, ein paar Monstersätze doch zu teilen. Ebenfalls herzlicher Dank für das Testlesen geht an Judith, Manuel, meine Schwester Johanna und meine Eltern.

Für die Rechenzeit bedanke ich mich bei Andreas Weiss, Peter Fierlinger und seiner Gruppe, Roland Diehl (RZG Zugang), der CAST Gruppe (Burkert/Dolag/Schartmann, LRZ Rechenzeit) und Tadziu Hoffmann (USM Server für GPU Tests). Für die Weitergabe der verwendeten Codes möchte ich mich bei A. Mignone, M. Krumholz und R. Theyssier bedanken. Letzterem bin ich auch für Tipps im Rahmen der Sommerschule in Evora und im Rahmen seines Münchenbesuches dankbar. Ich möchte mich auch bei M.-M. Mac Low stellvertretend für alle Personen, die Interesse an meiner Arbeit gezeigt haben, dafür bedanken, dass er mich dazu gebracht hat, die Stärken

meiner Arbeit, die hoffentlich zumindest in kleinen Bereichen ein wenig über das bereits publizierte Wissen anderer Autoren hinausgehen, zu erkennen.

Mein herzlicher Dank geht auch an Birgit Schaffhauser, Heidi Jonas, Tina Jacobs, Petra Riedel, Sonja Lutz, die MIAPP Direktoren und alle anderen Teammitglieder am Universe Cluster und am MIAPP für die Hilfe, mir die Zeit zum Niederschreiben der Arbeit freizuschaufeln. Besonderer Dank geht hier an Thomas Würstl, der mich immer wieder an Effizienz und Prioritätensetzung erinnert hat.

Einen wichtigen Beitrag haben auch jene Mentoren, Kollegen und Freunde (stellvertretend seien Mike Breger, Gerd Hensler, Dieter Breitschwerdt, Marianne Wenzel, Kurt Krachler, Bernhard Aringer, Marianne Busch, Eva Ntormousi und mein Mann Peter genannt) geliefert, die mich immer daran erinnert haben, nochmal nachzudenken, bevor ich die Astronomie verlasse, ob ich das wirklich will und mir Wiedereinstiege ermöglicht haben. Ebenfalls wichtig war der Beitrag des Universe Cluster Teams, das mir nebenbei immer Aufgaben mit schnell sichtbarem Erfolg übertragen hat und mir dadurch geholfen hat, meinen Horizont und meine Fertigkeiten zu erweitern.

Diese Arbeit trägt absichtlich keine Widmung, denn ich denke, sie soll für alle sein, die sich für dieses Thema interessieren. Ich lade also jede/n Leser/in der Arbeit ein, eine Widmung für sich selbst auf diese Arbeit zu schreiben.

Appendix A

Mathematica source code listings

Listing A.1: Solve for the internal structure of the Sedov-Taylor bubble with Mathematica

```
r0 = 0.01;(*inner boundary, zero leads to crashes*)
rsh = 1;(*outer boundary, location of the shock*)
g = 5/3;(*heat capacity ratio*)
n = 3;(*number of dimensions*)
Sedov = NDSolve[{(2/(g + 1)*u[x] - x)*u'[x] - n/2*u[x] + (g - 1)/(g + 1)*p'[x]/d[x
    ] == 0, (u[x] - x*(g + 1)/2)*d'[x]/d[x] + u'[x] + 2*u[x]/x == 0, (2/(g + 1)*u[
    x] - x)*p'[x] - 5/3*p[x] (2/(g + 1)*u[x] - x)*d'[x]/d[x] - n*p[x] == 0, p[rsh]
     == 1, d[rsh] == 1, u[rsh] == 1}, {p, d, u}, {x, r0, rsh}]
Export["sedov.csv", Table[Flatten[{t, p[x], u[x], d[x]} /. Sedov], {x, r0, rsh,
    0.001}]];
```

Listing A.2: Iterative solution for α with Mathematica

```
alpha0 = 0.507565;(*proportionality constant, initial value*)
t = 0.688;(*evolution time for which the solution is computed*)
g = 5/3;(*heat capacity ratio*)
n = 3;(*number of dimensions*)
rs[alpha_, t_, n_] := alpha^{-0.2}*t^{2/(2 + n)}; (*shock front radius*)
ps[alpha_, t_, n_, g_] := alpha^{-0.4}*2/(g + 1)*4/(2 + n)^2*t^{-2*n/(2 + n)}; (*
    shock front pressure*)
vs[alpha_, t_, n_, g_] := (2/(g + 1))*alpha^{-0.2}*2/(2 + n)*t^{-n/(2 + n)}; (*
    shock front expansion velocity*)
Etotal[alpha_, t_, n_, g_] := 4*Pi*(NIntegrate[{(x/rs[alpha, t, n][[1]])^2* p[x/rs
    [alpha, t, n][[1]]]*ps[alpha, t, n, g][[1]]/(g - 1) /. Sedov}, {x, 0, rs[alpha
    , t, n][[1]]}] + NIntegrate[{(x/rs[alpha, t, n][[1]])^2*0.5*d[x/rs[alpha, t,
    n][[1]]]*(g + 1)/(g - 1)*u[x/rs[alpha, t, n][[1]]]*vs[alpha, t, n, g][[1]]*u[x
    /rs[alpha, t, n][[1]]]*vs[alpha, t, n, g][[1]] /. Sedov }, {x, 0, rs[alpha, t,
     n][[1]]}]); (*total energy*)
Eth[alpha_, t_, n_, g_] := NIntegrate[{(x/rs[alpha, t, n][[1]])^2*p[x/rs[alpha, t,
     n][[1]]]* ps[alpha, t, n, g][[1]]/(g - 1) /. Sedov}, {x, 0, rs[alpha, t,n
    ][[1]]}]/(NIntegrate[{(x/rs[alpha, t, n][[1]])^2*p[x/rs[alpha, t, n][[1]]]*ps[
    alpha, t, n, g][[1]]/(g - 1) /. Sedov}, {x, 0, rs[alpha, t, n][[1]]}] +
    NIntegrate[{(x/rs[alpha, t, n][[1]])^2*0.5*d[x/rs[alpha, t, n][[1]]]*(g + 1)/(
    g - 1)*u[x/rs[alpha, t, n][[1]]]*vs[alpha, t, n, g][[1]]*u[x/rs[alpha, t, n
    ][[1]]]*vs[alpha, t, n, g][[1]] /. Sedov }, {x, 0, rs[alpha, t, n][[1]]}]); (*
    thermal energy fraction*)
newAlpha = FindRoot[Etotal[alpha, t, n, g] == 1, {alpha, alpha0}, Evaluated ->
    False][[1, 2]] (*iterative solution for the proportionality constant*)
```

```
Flatten[Eth[newAlpha, t, n, g]][[1]] (*output of the thermal energy fraction*)
```

Listing A.3: Solve for the structure between CD and shell with Mathematica

```
rcd = 0.85839; (*location of the CD, inner boundary*)
Weaver77 = NDSolve[{3*(u[x] - x)*u'[x] - 2*u[x] + 3*p'[x]/g[x] == 0, (u[x] - x)*g
    '[x]/g[x] + u'[x] + 2*u[x]/x == 0, 3*(u[x] - x)*p'[x] - 3*5/3*p[x] (u[x] - x)*
    g'[x]/g[x] - 4*p[x] == 0, p[1] == 0.75, g[1] == 4, u[1] == 0.75}, {p, g, u},
    {x, rcd, 1}]
Export["weaver77.csv", Table[Flatten[{t, p[x], u[x], d[x]} /. Weaver77], {x, rcd,
    1, 0.001}]];
```

Appendix B

Pluto source code listings

The listings in this section show the differences between the code used for this work and the standard version of PLUTO 4.0 (Mignone et al., 2007, 2012). The latter version can be obtained at http://plutocode.ph.unito.it/.

Listing B.1: Modifications in boundary.c

```
32 /* ////////////////////////////////////////////////////////////////// */
33 /* Modifications K. Fierlinger:
34    call UserDefBoundary for SN explosions (defined in init.c)
35    and store time of last SN event                    lines 65 and 93-98 */
36 /* ////////////////////////////////////////////////////////////////// */
65   static double last_time = -1.0; //time of the last SN event
93    if ( g_time-g_dt <= g_supernova && g_time > last_time){
94     //insert all feedback at once - predictor/corrector scheme would otherwise
           make energy input unpredictable (to 10%)
95     UserDefBoundary (d, NULL, 0, grid);
96     //printf("time %g %g\n",g_time,last_time);
97    }
98    last_time=g_time;
```

Listing B.2: Modifications in cooltable.dat to create an artificial equilibrium

```
226 1.057800e+03    9.560100e-25
227 1.079900e+03    0.0
228 1.102600e+03    0.0
229 1.125600e+03    9.736300e-25
```

Listing B.3: Modifications in cooling_source.c

```
1   #define RHOMIN 0.09
2   #undef RHOMIN
35  /* ////////////////////////////////////////////////////////////////// */
36  /* Modifications K. Fierlinger:
37     RHOMIN can be used to switch off cooling below this density: line 1-2, 106-109,
           222-227
38     no cooling losses in feedback region:                    line 111-112, 158
39     store cooling losses for analysis:                       line 113
40     g_minCoolingTemp is ignored:                             line 196
41  */
42  /* ////////////////////////////////////////////////////////////////// */
106 #ifdef RHOMIN
```

```
107   //print ("! CoolingSource %f %f \n", g_inputParam[RHO_MIN] , g_inputParam[RHO_IN
          ]);
108   if(v0[RHO] > g_inputParam[RHO_MIN] * g_inputParam[RHO_IN]){
109 #endif
111    if (GXYZ[IDIR].x[i] < g_inputParam[R_DRIVER]) {v0[TRC]=0.0;} //no cooling losses
           inside feedback region
112    else { Radiat(v0, k1);}
113    d->Vc[TRC][k][j][i]=v0[TRC]; //save cooling loss in tracer
196 //   if (T1 < g_minCoolingTemp && T0 > g_minCoolingTemp)
222 #ifdef RHOMIN
223   }else
224   {v0[TRC]=0.0;
225    d->Vc[TRC][k][j][i]=v0[TRC]; //save cooling loss in tracer
226   }
227 #endif
```

Listing B.4: Modifications in eta_visc.c

```
4  /* ////////////////////////////////////////////////////////////////// */
5  /* Modifications K. Fierlinger:
6     custom value of eta2_visc in line 27 */
7  /* ////////////////////////////////////////////////////////////////// */
27  *eta2_visc = 5.5e-8;
```

Listing B.5: Modifications in globals.h

```
100 double g_unitDensity  = 1.e-22; /**< Unit density in gr/cm^3. */
101 double g_unitLength   = 1.e19;  /**< Unit Legnth in cm. */
102 double g_unitVelocity = 1.e8;   /**< Unit velocity in cm/sec. */
109 double g_smallDensity   = 1.e-7; /**< Small value for density fix. */
110 double g_smallPressure  = 1.e-7; /**< Small value for pressure fix. */
117 double g_supernova = 500.; /**< The latest time when the SN goes off. */
```

Listing B.6: Modifications in input_data.c

```
34 /* ////////////////////////////////////////////////////////////////// */
35 /* Modifications K. Fierlinger:
36    InputDataRead loop over nv removed. nv is now input parameter lines 195, 206,
          242, 268
37    New interpolation function: void InputDataExpand1d (double *vs, double x1) */
38 /* ////////////////////////////////////////////////////////////////// */
195 void InputDataRead (char *data_fname, int nv)
206   int    i, j, k;
242   //for (nv = 0; nv < id_nvar; nv++){
268   //}
469 void InputDataExpand1d (double *vs, double x1)
470 /*!
471  * Perform bi- or tri-linear interpolation on external
472  * dataset to compute vs[] at the given point {x1,x2,x3}.
473  *
474  * \param [in] vs interpolated value
475  * \param [in] x1 coordinate point at which at interpolates are desired
476  * \param [in] x2 coordinate point at which at interpolates are desired
477  * \param [in] x3 coordinate point at which at interpolates are desired
478  * \return This function has no return value.
479  *
```

```
480  * The function performs the following tasks.
481  ****************************************************************** */
482 {
483   int nv, inv, ii=0 ;
484   double ***V;
485 /* ---------------------------------------------------------------- */
486 /*! - Make sure point (x1,x2,x3) does not fall outside input grid range.
487       Limit to input grid edge otherwise.                           */
488 /* ---------------------------------------------------------------- */
489  if (x1 >= id_x1[0] && x1 <= id_x1[id_nx1-1])  {
490   ii = 0;
491   while (x1 > id_x1[ii] && ii<id_nx1-1){
492     ii++;
493   }
494   //if(ii>2){ii-=2;};
495   for (nv = 0; nv < id_nvar; nv++) {
496     inv = id_var_indx[nv];
497     V = Vin[nv];
498     vs[inv] = V[0][0][ii];
499   }
500  }
501  else{
502   vs[RHO] = g_inputParam[RHO_IN]; /* 1e-22 g/cm3 */
503   vs[VX1] = 0.0; /*initial Vx array*/
504   vs[VX2] = 0.0; /*initial Vy array*/
505   vs[VX3] = 0.0; /*initial Vz array*/
506   vs[PRS] = g_inputParam[PRS_IN]; /* 1e-6 erg/cm3 */
507   vs[TRC] = 0.0;
508  }
509 }
```

Listing B.7: Modifications in mappers.c

```
21 /* ////////////////////////////////////////////////////////////////// */
22 /* Pluto 4.0 only sets a minimal density, if negative densities are found.
23    Modifications K. Fierlinger:
24    (1) don't let density drop below minimal density: ... lines 58-59 and 116-132
25    (2) use maximum of g_smallPressure or mean value of left+ right cell (to avoid
          new pressure mimina)
26    if due to the new pressure the thermal energy is now larger than the total
          energy, remove kinetic energy.
27    otherwise reduce kinetic energy by scaling velocities. ... lines 178-182 and
          191-209 */
28 /* ////////////////////////////////////////////////////////////////// */
58     rho=MAX(g_smallDensity, rho);
59 //     if(rho < 0.0){print("rho<0");}
116    if (u[RHO] < g_smallDensity) {
117 #ifdef WARNMINRHO
118       print("! ConsToPrim: too low density (%8.2e), ", u[RHO]);
119       print(" old pressure: (%8.2e), ", u[PRS]);
120 #endif
121       // constant temperature (p/nV)=kT
122       //u[PRS] = MAX(g_smallPressure,g_smallDensity*u[PRS]/u[RHO]);
123       // constant internal energy E(intermal) = p/(gamma-1)
124        u[PRS] = MAX(g_smallPressure,u[PRS]);
125       // constant total energy:  E = p/(gamma-1)+ 0.5 rho v^2
```

```
126         //u[PRS] = MAX(g_smallPressure,u[PRS]+0.5*(g_gamma-1.0)*m2*(1./u[RHO]-1./
                g_smallDensity));
127         u[RHO] = g_smallDensity;
128 #ifdef WARNMINRHO
129         print("␣new␣density␣(%8.2e),␣", u[RHO]);
130         print("␣new␣pressure␣(%8.2e),␣", u[PRS]);
131         Where (i, NULL);
132 #endif
178             EXPAND(v[VX1] = 0.0;   ,
179                    v[VX2] = 0.0;   ,
180                    v[VX3] = 0.0;)
182
182           u[ENG]    = g_smallPressure/gmm1;
191           // use maximum of g_smallPressure or mean value of left+ right cell (to
                avoid new pressure mimina)
192           v[PRS]    = MAX(g_smallPressure,0.5*(uprim[i-1][PRS]+uprim[i+1][PRS]));
                // warning: uprim[i+1][PRS] not yet updated
193           print("!␣␣negative␣p(left)␣(%8.2e)␣p␣(%8.2e)␣p(right)␣(%8.2e),␣\n",
                uprim[i-1][PRS],v[PRS], uprim[i+1][PRS]);
194           if(u[ENG] <= v[PRS]/gmm1){
195           // if due to the new pressure the thermal energy is now larger than the
                total energy, remove kinetic energy.
196            v[PRS]    = u[ENG]*gmm1;
197            EXPAND(v[VX1] = 0.0;   ,
198                   v[VX2] = 0.0;   ,
199                   v[VX3] = 0.0;)
200           }else{
201           // otherwise reduce kinetic energy by scaling velocities.
202            //corr_e = sqrt(E_kin(new) / E_kin(old)) = v(new)/v(old)
203            corr_e = sqrt((u[ENG]-v[PRS]/gmm1)/kin) ;
204            print("!␣␣negative␣p(E)␣v(new)/v(old)␣(%8.2e),␣\n", corr_e);
205            //kin = 0.5*m2/u[RHO];
206            EXPAND(v[VX1] = v[VX1]*corr_e;   ,
207                   v[VX2] = v[VX2]*corr_e;   ,
208                   v[VX3] = v[VX3]*corr_e;)
209           }
```

Listing B.8: Modifications in pluto.h

```
1076 extern double g_time, g_dt,g_supernova;
```

Listing B.9: Modifications in prototypes.h

```
50  void InputDataRead (char *, int );
158 void Wind (double,double *e, double *m);
```

Listing B.10: Modifications in radiat.c

```
1 #include "pluto.h"
2 #define mass_X   0.7519   /*   = hydrogen mass fraction, mean molar mass 1.33 like
      Thornton */
3 //#define mass_X   0.732   /*   = hydrogen mass fraction,  frac_H =  0.917,
     (1.008+4.004*0.082+0.03)=1.366 */
4 //#define frac_Z   1.e-3   /*   = N(Z) / N(H), fractional number density of metals
      (Z)
5 //                              with respect to hydrogen (H) */
```

```
6  //#define frac_He  0.082    /*    = N(Z) / N(H), fractional number density of helium
       (He)
7  //                              with respect to hydrogen (H) */
8  //#define A_Z      30.0     /*   mean atomic weight of heavy elements  */
9  //#define A_He     4.004    /*   atomic weight of Helium  */
10 //#define A_H      1.008    /*   atomic weight of Hydrogen  */
12
12 #undef OUTPUT_EQUILIBRIUM
13 #undef COOLING_SUBSTEP
14 #define artificial_ISM
15 #undef artificial_ISM
16 #undef EVA_table
17 #define EVA_table 1 // use the table of Eva Ntormousi et al. 2011 ApJ 731, 13 ...
       the CLOUDY part (>25000 K) includes grains, the lower part takes ionization
       into account.
18 #undef TAB_table
19 #define TAB_table 1 // use Pluto's CLOUDY table too [NOT good below 25000 K]
20 /* ************************************************************** */
21 /* Modifications K. Fierlinger:
22    Modes: write equilibrium data to file                    line 10
23    Modes: make several cooling time steps per hydro time step line 11
24    Modes: add cooling as in Ntormousi et al. 2011 ApJ 731, 13 lines 12-13
25    Modes: use Pluto's table above 25000 K                   line 14
26 */
27 /* ////////////////////////////////////////////////////////////////// */
28 void Radiat (real *v, real *rhs)
29 /*
30  * NAME          Radiat
31  * PURPOSE       Provide r.h.s. for tabulated cooling.
32  *
33  ************************************************************** */
34 {
35 #ifdef EVA_table
36 /* ------------------------------------------
37   Read tabulated cooling function:
38   This code is based on the f90 version of E. Ntormousi. See Ntormousi et al
39   2011 ApJ 731, 13. It reads cooling and heating rates from a file created
40   by the program cooleq.pro (F. Heitsch) for the Interstellar Medium
41   This fixed table includes heating and cooling rates, as well as their
42   derivatives with respect to temperature
43    ------------------------------------------ */
44   static double *n1_tab, *n2_tab, *T1_tab, *T2_tab; // 1d Arrays for hydrogen
       number density nH [1/cm3] or temperature [K]
45   static double **cool1_tab, **heat1_tab;           // 2d Arrays for Lambda(n,T) -
       heating and cooling from table1
46   static double **cool1prime_tab, **heat1prime_tab; // 2d Arrays for Lambda'(n,T)
       - heating and cooling derivatives just present in table1
47   static double **cool2_tab, **heat2_tab;           // 2d Arrays for Lambda(n,T) -
       heating and cooling from table2
48   FILE *fcool1;                                     //filepointer to the binary
       files
49   size_t testresult;
50   //real smallnum_cooling= 1e-13;                   // avoid numerical problems
52
52   real  nH_min_fix   = 0.01  ; //minimum density [nH in 1/cm3]
```

```
53    real  nH_max_fix   = 1.e5  ; //maximum density [nH in 1/cm3]
54    real  T_min_fix_1 = 10.    ; //minimum temperature [K]  for cooling table 1
55    real  T_max_fix_1 = 2.5e4  ; //maximum temperature [K]  for cooling table 1
56    real  T_min_fix_2 = 2.5e4  ; //minimum temperature [K]  for cooling table 2
57    real  T_max_fix_2 = 9.999e6; //maximum temperature [K]  for cooling table 2
59
59    int nbin_T_1=500;   //table 1 resolution in temperature
60    int nbin_T_2=9975;  //table 2 resolution in temperature
61    int nbin_n_1=500;   //table 1 resolution in density
62    int nbin_n_2=8;     //table 2 resolution in density
64
64    double nH;              // Hydrogen number density nH [1/cm3]
65    double log10n, log10T; // log10 of the hydrogen number density nH [1/cm3] or
          temperature [K]
66    double dlog_nH_1,dlog_T_1,h_1, h2_1, h3_1; // first table: equidistant step size
           in log10(nH) and log10(T)
67    double dlog_nH_2,d_T_2; // second table: equidistant step size in log10(nH) and
          T [NOT log10(T)]
68    int i_nH_1,i_nH_2,i_T;  // position of the lower interval boundary
69    double w1H_1,w1H_2,w2H_1,w2H_2; // weights for lower/upper part of the interval
          in density ... the density is between two values in the logarithmically
          equidistant density table
70  #endif
71  #if !defined(EVA_table) || defined(TAB_table)
72    int     klo, khi, kmid;       //table element number for binary search
73    static int ntab;              //table size
74    static real *L_tab, *T_tab; //1d Arrays for the cooling function and temperature
           [K]
75    real    scrh;                 //interpolate L_Tab [erg  cm3 / s]
76    real    Tmid, dT;             //T at center of interval, temperature difference
77    FILE *fcool;                  //filepointer to the cooling table
78  #endif
79    real   mu, T, lambda;
80    static real  E_cost;          //converts erg / cm3 / s to code units
81  #ifdef artificial_ISM
82    real    T0;
83  #endif
84  //  if (x1 < g_inputParam[R_DRIVER]) {rhs[PRS] = 0.0; v[TRC]=0.0; return; }
85  /* ------------------------------------------------
86  /* ------------------------------------------------
87          Load Table
88     ------------------------------------------------ */
89  //  nH = v[RHO]*g_unitDensity/CONST_mp;
90  #ifdef EVA_table
91    nH = v[RHO]*g_unitDensity*mass_X/CONST_mp;
92    log10n=log10(nH);
93  /* ------------------------------------------------
94          Read tabulated cooling function
95     ------------------------------------------------ */
96    if (T1_tab == NULL){
97      print1 ("␣>␣Reading␣table␣1␣from␣disk...\n");
98      fcool1 = fopen("cooling_table_n_-2_5_T_10_25000.dat","rb");
99      if (fcool1 == NULL){
100       print1 ("!␣cooling_table_n_-2_5_T_10_25000.dat␣does␣not␣exist.\n");
101       QUIT_PLUTO(1);
```

```
102     }
103     n1_tab = ARRAY_1D(nbin_n_1, double);
104     T1_tab = ARRAY_1D(nbin_T_1, double);
105     heat1_tab       = ARRAY_2D(nbin_n_1,nbin_T_1, double);
106     cool1_tab       = ARRAY_2D(nbin_n_1,nbin_T_1, double);
107     heat1prime_tab = ARRAY_2D(nbin_n_1,nbin_T_1, double);
108     cool1prime_tab = ARRAY_2D(nbin_n_1,nbin_T_1, double);
110
110     //file1 contains binary reak kind=8 (8 byte): heating rates,
111     //cooling rates, d(heating)/dT, d(cooling)/dT in this order.
113
113     double a;
114     char tmp[8];
115     testresult=fread(tmp,1,4,fcool1);
116     int i,j;
117     for(i=0; i<nbin_n_1; i++){//populate n1_tab
118      testresult=fread(&n1_tab[i],sizeof(a),1,fcool1);
119      //printf (" ! n[%d]= %12.6e \n",i,n1_tab[i]);
120     }
121     testresult=fread(&a,sizeof(a),1,fcool1);
122     for(i=0; i<nbin_T_1; i++){//populate T1_tab
123      testresult=fread(&T1_tab[i],sizeof(a),1,fcool1);
124      //printf (" ! T[%d]= %12.6e \n",i,T1_tab[i]);
125     }
127
127     testresult=fread(&a,sizeof(a),1,fcool1);
128     for(i=0; i<nbin_T_1; i++){
129      for(j=0;j<nbin_n_1; j++){//populate heat1_tab
130       testresult=fread(&heat1_tab[j][i],sizeof(a),1,fcool1);
131      }
132     }
133     testresult=fread(&a,sizeof(a),1,fcool1);
134     for(i=0; i<nbin_T_1; i++){
135      for(j=0;j<nbin_n_1; j++){//populate cool1_tab
136       testresult=fread(&cool1_tab[j][i],sizeof(a),1,fcool1);
137      }
138     }
139     testresult=fread(&a,sizeof(a),1,fcool1);
140     for(i=0; i<nbin_T_1; i++){
141      for(j=0;j<nbin_n_1; j++){//populate heat1prime_tab
142       testresult=fread(&heat1prime_tab[j][i],sizeof(a),1,fcool1);
143      }
144     }
145     for(i=0; i<nbin_T_1; i++){
146      for(j=0;j<nbin_n_1; j++){//populate cool1prime_tab
147       testresult=fread(&cool1prime_tab[j][i],sizeof(a),1,fcool1);
148      }
149     }
151
151     fclose(fcool1);
153
153     print1 ("␣>␣Reading␣table␣2␣from␣disk...\n");
154     fcool1 = fopen("cooling_table_n_-2_5_T_25000_107.dat","rb");
155     if (fcool1 == NULL){
156       print1 ("!␣cooling_table_n_-2_5_T_25000_107.dat␣does␣not␣exist.\n");
```

```
157         QUIT_PLUTO(1);
158       }
159       n2_tab = ARRAY_1D(nbin_n_2, double);
160       T2_tab = ARRAY_1D(nbin_T_2, double);
161       heat2_tab       = ARRAY_2D(nbin_n_2,nbin_T_2, double);
162       cool2_tab       = ARRAY_2D(nbin_n_2,nbin_T_2, double);
164
164       testresult=fread(tmp,1,4,fcool1);
165       for(i=0; i<nbin_n_2; i++){//populate n2_tab
166        testresult=fread(&n2_tab[i],sizeof(a),1,fcool1);
167       }
168       testresult=fread(&a,sizeof(a),1,fcool1);
169       for(i=0; i<nbin_T_2; i++){//populate T2_tab
170        testresult=fread(&T2_tab[i],sizeof(a),1,fcool1);
171       }
173
173       testresult=fread(&a,sizeof(a),1,fcool1);
174       for(i=0; i<nbin_T_2; i++){
175        for(j=0;j<nbin_n_2; j++){//populate cool2_tab
176         testresult=fread(&cool2_tab[j][i],sizeof(a),1,fcool1);
177         cool2_tab[j][i] = log10(cool2_tab[j][i]);
178         //cool2_tab[j][i] = log10(cool2_tab[j][i])-(2.*n2_tab[j]);
179        }
180       }
181       // !heat/cool swapped. Heating always >> cooling
182       testresult=fread(&a,sizeof(a),1,fcool1);
183       for(i=0; i<nbin_T_2; i++){
184        for(j=0;j<nbin_n_2; j++){//populate heat2_tab
185         testresult=fread(&heat2_tab[j][i],sizeof(a),1,fcool1);
186         heat2_tab[j][i] = log10(heat2_tab[j][i]);
187        }
188       }
189       fclose(fcool1);
190   #ifdef OUTPUT_EQUILIBRIUM
191     /*-------------------------------------------------*/
192       //output cooling heating equilibrium
193       double xT=0.0;
194       double dL=1e11;
195       double xdL,Lsign;
196       for(i=1; i<nbin_T_1; i++){
197        dL=1e11;
198        for(j=0;j<nbin_n_1; j++){
199         Lsign=(cool1_tab[j][i]-heat1_tab[j][i])*(cool1_tab[j][i-1]-heat1_tab[j][i
               -1]);
200         //printf (" %12.6e %12.6e %12.6e  %12.6e %12.6e \n",cool1_tab[j][i],
               heat1_tab[j][i],pow(10.,cool1_tab[j][i]),pow(10.,heat1_tab[j][i]),xdL);
201         if(Lsign < 0.0)
202          {
203           xT=n1_tab[j-1]+(n1_tab[j-1]-n1_tab[j])*(cool1_tab[j][i-1]-heat1_tab[j][i
                 -1])/(cool1_tab[j][i-1]-heat1_tab[j][i-1]-cool1_tab[j][i]+heat1_tab[j
                 ][i]);
204          };
205        }
206        printf (" %12.6e %12.6e \n",xT,T1_tab[i]);
207        //printf (" %12.6e %12.6e %12.6e \n",xT,T1_tab[i],Lsign);
```

```
208       }
209       QUIT_PLUTO(1);
210     /*-------------------------------------------------*/
211 #endif
212     /*-------------------------------------------------*
213       //output cooling function
214       dlog_nH_1 = (double)(nbin_n_1-1)/(n1_tab[nbin_n_1-1]-n1_tab[0]); // 1 / delta
              (log10 nH)
215       for(j=0;j<nbin_n_2; j++){
216        i_nH_1  = MIN(MAX((int)floor(((double)(j-2)-n1_tab[0])*dlog_nH_1),0),nbin_n_1
              -2); // left index in nH
217        printf (" \n \n # log10(nH[%d])= %12.6e [nH/cm3], log10(nH[%d])= %12.6e [nH/
              cm3] , log10(nH[%d])= %12.6e [nH/cm3]\n",j,n2_tab[j],i_nH_1,n1_tab[i_nH_1
              ],i_nH_1+1,n1_tab[i_nH_1+1]);
218        for(i=0; i<nbin_T_1; i++){
219         printf (" %12.6e %12.6e %12.6e  %12.6e %12.6e \n",pow(10.,T1_tab[i]),
              cool1_tab[i_nH_1][i], heat1_tab[i_nH_1][i], cool1_tab[i_nH_1+1][i],
              heat1_tab[i_nH_1+1][i]);
220        }
221        for(i=0; i<nbin_T_2; i++){
222         printf (" %12.6e %12.6e %12.6e \n",T2_tab[i], cool2_tab[j][i], heat2_tab[j][
              i]);
223        }
224       }
225       QUIT_PLUTO(1);
226     *-------------------------------------------------*/
227 #ifdef TAB_table
228     } //add this "}" if you want to use both tables
229 #endif
230 #endif
231 #if !defined(EVA_table) || defined(TAB_table)
232     /*-------------------------------------------------*/
233     if (T_tab == NULL){
234       print1 ("␣>␣Reading␣table␣from␣disk...\n");
235       fcool = fopen("cooltable.dat","r");
236       if (fcool == NULL){
237         print1 ("!␣cooltable.dat␣does␣not␣exist.\n");
238         QUIT_PLUTO(1);
239       }
240       L_tab = ARRAY_1D(20000, double);
241       T_tab = ARRAY_1D(20000, double);
243
243       ntab = 0;
244       while (fscanf(fcool, "%lf␣␣%lf\n", T_tab + ntab,
245                                          L_tab + ntab)!=EOF) {
246         ntab++;
247       }
248     /* ------------------------------------------------ */
249 #endif
250       E_cost     = g_unitLength/g_unitDensity/pow(g_unitVelocity, 3.0); //converts
              erg / cm3 / s to code units
251     }
253
253 /* ------------------------------------------------
254               Get temperature
```

```
255     ------------------------------------------------------- */
257
257     if (v[PRS] < 0.0) v[PRS] = g_smallPressure;
258     /* mean molecular weight */
259     if (v[PRS]/v[RHO]*KELVIN < 1e4){mu=1.33;} //mu=1.33 like in Thornton
260     else {mu  = MeanMolecularWeight(v);}; //fully ionized
261     T    = v[PRS]/v[RHO]*KELVIN*mu;
262     if (T != T){
263       printf ("␣!␣Nan␣found␣in␣radiat␣\n");
264 #ifdef EVA_table
265       printf ("␣!␣rho␣=␣%12.6e␣[1/cm3],␣pr␣=␣%12.6e␣[1e-6␣erg/cm3]\n",nH, v[PRS]);
266 #else
267       printf ("␣!␣rho␣=␣%12.6e␣[1e-22␣g/cm3],␣pr␣=␣%12.6e␣[1e-6␣erg/cm3]\n",v[RHO],
            v[PRS]);
268 #endif
269       QUIT_PLUTO(1);
270     }
272
272 #ifdef artificial_ISM
273     T0=g_inputParam[PRS_IN]/g_inputParam[RHO_IN]*KELVIN*mu;
274     if (T > T0-0.02 && T < T0+0.02) { //artificial equilibrium at initial conditions
275       rhs[PRS] = 0.0;
276       v[TRC]=0.0;
277       return;
278     }
279 #endif
281
281     if (T < g_minCoolingTemp) {
282       rhs[PRS] = 0.0;
283       v[TRC]=0.0;
284       return;
285     }
286 /* -------------------------------------------------------
287           Table lookup
288     ------------------------------------------------------- */
289 #ifdef EVA_table
290 #ifdef TAB_table
291     if (T > MAX(T_max_fix_2,T_tab[ntab-1]) || T < T_min_fix_1 || nH < nH_min_fix ||
          nH > nH_max_fix){
292       // use both cooling functions - useful if high temperature end of the cooling
            table should be included
293       //-- if you want this, the "usual table" has to be included ("ifdef TAB_table"
            must be true) */
294 #else
295     if (T > T_max_fix_2 || T < T_min_fix_1 || nH < nH_min_fix || nH > nH_max_fix){
          //avoid extrapolation
296 #endif
297       rhs[PRS] = 0.0;
298       v[TRC]=0.0;
299       return;
300     }
302
302     //first table
303     dlog_nH_1 = (double)(nbin_n_1-1)/(n1_tab[nbin_n_1-1]-n1_tab[0]); // 1 / delta (
          log10 nH)
```

```
304   dlog_T_1  = (double)(nbin_T_1-1)/(T1_tab[nbin_T_1-1]-T1_tab[0]); // 1 / delta (
          log10 T)
305   h_1       = 1.0/dlog_T_1; //  delta (log10 T)
306   h2_1      = h_1*h_1;      // (delta (log10 T))^2
307   h3_1      = h2_1*h_1;     // (delta (log10 T))^3
309
309   //second table
310   dlog_nH_2 = (double)(nbin_n_2-1)/(n2_tab[nbin_n_2-1]-n2_tab[0]); // 1 / delta (
          log10 nH)
311   //WARNING this table is equidistant in T NOT in log10 T
312   d_T_2   = (double)(nbin_T_2-1)/(T2_tab[nbin_T_2-1]-T2_tab[0]);    // 1 / delta (T
          )
315
315
315   //both tables are equidistant in log10(rho)
316   //first table
317   i_nH_1    = MIN(MAX((int)floor((log10n-n1_tab[0])*dlog_nH_1),0),nbin_n_1-2); //
          left index in nH
318   w1H_1     = (n1_tab[i_nH_1+1]-log10n)*dlog_nH_1; // left weight in nH (smaller
          distance -> higher weight)
319   w2H_1     = (log10n-n1_tab[i_nH_1])*dlog_nH_1;   // right weight in nH
321
321   //second table
322   i_nH_2    = MIN(MAX((int)floor((log10n-n2_tab[0])*dlog_nH_2),0),nbin_n_2-2);//
          left index in nH
323   w1H_2     = (n2_tab[i_nH_2+1]-log10n)*dlog_nH_2; // left weight in nH (smaller
          distance -> higher weight)
324   w2H_2     = (log10n-n2_tab[i_nH_2])*dlog_nH_2;   // right weight in nH
326
326   if(T<T_min_fix_2){
327    double yy,yy2,yy3,fa,fb,fprima,fprimb,fbfa,beta,gamma1,cool,cool_prime,heat,
           heat_prime;
329
329    log10T=log10(T);
331
331    //printf (" ! T1[0]=%12.6e, T1[%d]=%12.6e \n",T1_tab[0], nbin_T_1-1, T1_tab[
           nbin_T_1-1]);
332    //printf (" ! log10(T)=%12.6e, dlog_T_1=%12.6e \n",log10T, dlog_T_1);
333    i_T   = MIN(MAX((int)floor((log10T-T1_tab[0])*dlog_T_1),0),nbin_T_1-2); // left
          index in T
334    yy    = log10T-T1_tab[i_T]; // (log10 T - log10 T_grid)
335    yy2   = yy*yy;              // (log10 T - log10 T_grid)^2
336    yy3   = yy2*yy;             // (log10 T - log10 T_grid)^3
338
338    fa      = cool1_tab[i_nH_1][i_T]*w1H_1   + cool1_tab[i_nH_1+1][i_T]*w2H_1;   //
          interpolate left T in nH
339    fb      = cool1_tab[i_nH_1][i_T+1]*w1H_1 + cool1_tab[i_nH_1+1][i_T+1]*w2H_1; //
          interpolate right T in nH
340    fprima = cool1prime_tab[i_nH_1][i_T]*w1H_1   + cool1prime_tab[i_nH_1+1][i_T]*
          w2H_1;   // interpolate left dT in nH
341    fprimb = cool1prime_tab[i_nH_1][i_T+1]*w1H_1 + cool1prime_tab[i_nH_1+1][i_T+1]*
          w2H_1; // interpolate right dT in nH
342    fbfa    = (fb-fa);
344
344    real smallnum_cooling= 1e-13; // avoid numerical problems
```

```
345     if(abs(fbfa/fb)<smallnum_cooling){fbfa=0.0;}
347
347     beta        = 3.0*(fbfa)/h2_1-(2.0*fprima+fprimb)/h_1;
348     gamma1      = (fprima+fprimb)/h2_1-2.0*(fbfa)/h3_1;
349     cool        = pow(10.0,fa+fprima*yy+beta*yy2+gamma1*yy3) ;
351
351     fa      = heat1_tab[i_nH_1][i_T]*w1H_1   + heat1_tab[i_nH_1+1][i_T]*w2H_1;   //
            interpolate left T in nH
352     fb      = heat1_tab[i_nH_1][i_T+1]*w1H_1 + heat1_tab[i_nH_1+1][i_T+1]*w2H_1; //
            interpolate right T in nH
353     fprima = heat1prime_tab[i_nH_1][i_T]*w1H_1   + heat1prime_tab[i_nH_1+1][i_T]*
            w2H_1;   // interpolate left dT in nH
354     fprimb = heat1prime_tab[i_nH_1][i_T+1]*w1H_1 + heat1prime_tab[i_nH_1+1][i_T+1]*
            w2H_1; // interpolate right dT in nH
355     fbfa    = (fb-fa);
357
357     if(abs(fbfa/fb)<smallnum_cooling){fbfa=0.0;}
359
359     beta        = 3.0*(fbfa)/h2_1-(2.0*fprima+fprimb)/h_1;
360     gamma1      = (fprima+fprimb)/h2_1-2.0*(fbfa)/h3_1;
361     heat        = pow(10.0,fa+fprima*yy+beta*yy2+gamma1*yy3) ;
362     lambda      = cool-heat;//in [erg/ cm3 / s]
364
364 #ifdef COOLING_SUBSTEP
365   // time until next lower tabulated temperature is reached (if cooling - not
        heating - dominates)
366     if(lambda>0.0){
367      // cooling time: thermal energy above nect T_i divided by loss
368      double dt1 = (v[PRS]-pow(10.0,T1_tab[i_T])*v[RHO]/mu/KELVIN)/((g_gamma - 1.0)*
           lambda*E_cost);
369      //recompute lambda if next lower tabulated temperature is reached:
370      if(dt1>g_dt){
371       double dt=dt1;
372       double avg_lam=0.0;
373       avg_lam+=dt1*lambda;
374       while(dt<g_dt){
375         i_T--;
376         fa      = cool1_tab[i_nH_1][i_T]*w1H_1   + cool1_tab[i_nH_1+1][i_T]*w2H_1;
                  // interpolate left T in nH
377         cool    = pow(10.0,fa) ;
378         fa      = heat1_tab[i_nH_1][i_T]*w1H_1   + heat1_tab[i_nH_1+1][i_T]*w2H_1;
                  // interpolate left T in nH
379         heat    = pow(10.0,fa) ;
380         lambda = (cool-heat);
381         dt1=MIN(((pow(10.0,T1_tab[i_T+1])-pow(10.0,T1_tab[i_T]))*v[RHO]/mu/KELVIN)
                /((g_gamma - 1.0)*lambda*E_cost),g_dt-dt);
382         dt+=dt1;
383         avg_lam+=dt1*lambda;
384       }
385       lambda=avg_lam/dt;
386      }
387     }
388 #endif
389   }
390 #ifndef TAB_table
```

```
391   else if(T<T_max_fix_2)
392   {
393     double yy,fa,fb,cool,heat,w1T,w2T;
394     i_T = MIN(MAX((int)floor((T-T2_tab[0])*d_T_2),0),nbin_T_2-2); // left index in
          T
395     yy = T-T2_tab[i_T];// (T - T_grid)
396     w2T = yy*d_T_2;     // right weight (smaller distance -> higher weight)
397     w1T = 1.0-w1T;      // left weight
399
399     fa   = cool2_tab[i_nH_2][i_T]*w1H_2   + cool2_tab[i_nH_2+1][i_T]*w2H_2;   //
          interpolate left T in nH
400     fb   = cool2_tab[i_nH_2][i_T+1]*w1H_2 + cool2_tab[i_nH_2+1][i_T+1]*w2H_2; //
          interpolate right T in nH
401     cool = pow(10.0,fa)*w1T+pow(10.0,fb)*w2T;
403
403     fa   = heat2_tab[i_nH_2][i_T]*w1H_2   + heat2_tab[i_nH_2+1][i_T]*w2H_2;   //
          interpolate left T in nH
404     fb   = heat2_tab[i_nH_2][i_T+1]*w1H_2 + heat2_tab[i_nH_2+1][i_T+1]*w2H_2; //
          interpolate right T in nH
405     heat = pow(10.0,fa)*w1T+pow(10.0,fb)*w2T;
406     lambda = cool-heat; //in [erg/ cm3 / s]
407   }
408 #endif
409   else{
410 #endif
411     // use both cooling functions - useful if high temperature end of the cooling
          table should be included
412     //-- if you want this, the "usual table" has to be included (remove all "ifdef
          Eva_table" lines)
413 #if !defined(EVA_table) || defined(TAB_table)
414 /* ------------------------------------------------
415         Table lookup by binary search
416    ------------------------------------------------*/
417     klo = 0;
418     khi = ntab - 1;
419 #if !defined(EVA_table)
420     if (T > T_tab[khi] || T < T_tab[klo] ){
421       rhs[PRS] = 0.0;
422       v[TRC]=0.0;
423       return;
424     }
425     else{
426 #endif
427      while (klo != (khi - 1)){
428       kmid = (klo + khi)/2;
429       Tmid = T_tab[kmid];
430       if (T <= Tmid){
431         khi = kmid;
432       }else if (T > Tmid){
433         klo = kmid;
434       }
435      }
436 #if !defined(EVA_table)
437     }
438 #endif
```

```
439     dT        = T_tab[khi] - T_tab[klo];
440     scrh      = L_tab[klo]*(T_tab[khi] - T)/dT + L_tab[khi]*(T - T_tab[klo])/dT; //
            in [erg  cm3 / s]
441 #ifdef EVA_table
442     lambda=scrh*nH*nH; //in [erg  cm3 / s]
443 #else
444   //use PLUTO coolingtable between  T_tab[khi] and T_min_fix_1
445     lambda    = scrh*v[RHO]*v[RHO]*g_unitDensity*g_unitDensity/(CONST_mp*CONST_mp);
446 #endif
447 #ifdef COOLING_SUBSTEP
448   // time until next lower tabulated temperature is reached (if cooling - not
        heating - dominates)
449     if(lambda>0.0){
450      // cooling time: thermal energy above nect T_i divided by loss
451      double dt1 = (v[PRS]-T_tab[klo]*v[RHO]/mu/KELVIN)/((g_gamma - 1.0)*lambda*
            E_cost);
452      //recompute lambda if next lower tabulated temperature is reached:
453      if(dt1>g_dt){
454       double dt=dt1;
455       double avg_lam=0.0;
456       avg_lam+=dt1*lambda;
457       while(dt<g_dt){
458        klo--;
459        khi--;
460        lambda=L_tab[klo]*v[RHO]*v[RHO]*g_unitDensity*g_unitDensity/(CONST_mp*
               CONST_mp);
461        dt1=MIN(((T_tab[khi]-T_tab[klo])*v[RHO]/mu/KELVIN)/((g_gamma - 1.0)*lambda*
               E_cost),g_dt-dt);
462        dt+=dt1;
463        avg_lam+=dt1*lambda;
464       }
465       lambda=avg_lam/dt;
466      }
467     }
468 #endif
469 #ifdef EVA_table
470   }
471 #endif
472 #endif
473   v[TRC]=lambda;
474   rhs[PRS] = -(g_gamma - 1.0)*lambda;//already in [erg / cm3 / s]  //  *v[RHO]*v[
        RHO]*g_unitDensity*g_unitDensity/(CONST_mp*CONST_mp);
475   rhs[PRS] *= E_cost;
476  /* ------------------------------------------------------------*/
477 }
478 #undef T_MIN
479 /* ********************************************************************* */
480 double MeanMolecularWeight (real *V)
481 /* ********************************************************************* */
482 {
483   return (0.5); //fully ionized
484 /*
485   return  ( (A_H + frac_He*A_He + frac_Z*A_Z) /
486             (2.0 + frac_He + 2.0*frac_Z - 0.0));
487 // (1+0.082*4.004+30e-3)/(2+0.082+2e-3)=0.65
```

```
488 // (1+0.082*4.004+30e-3)=1.35
489 */
490 }
```

Listing B.11: Modifications in set_output.c

```
22 /* ///////////////////////////////////////////////////////////////////// */
23 /* Modifications K. Fierlinger:
24    start numbering outputfiles after g_inputParam[READIN] (lines 59)     */
25 /* ///////////////////////////////////////////////////////////////////// */
59     output->nfile     = -1+(int)g_inputParam[READIN];   //output->nfile     = -1;
```

Listing B.12: Modifications in startup.c

```
13 /* ///////////////////////////////////////////////////////////////////// */
14 /* Modifications K. Fierlinger: added value of negative density/pressure to output
15    lines  209 and 214 */
16 /* ///////////////////////////////////////////////////////////////////// */
209       print ("! Startup: density is negative, zone [%f, %f, %f] %f\n", x1,x2,x3,
              us[RHO]);
214       print ("! Startup: pressure is negative, zone [%f, %f, %f] %f\n",x1,x2,x3,
              us[PRS]) ;
```

Listing B.13: Modifications in sweep.c

```
17 /* ///////////////////////////////////////////////////////////////////// */
18 /* Modifications K. Fierlinger: no outflow from empty cells (lines 170, 221, 223,
      276) */
19 /* ///////////////////////////////////////////////////////////////////// */
170         if(u[in][RHO]+state.rhs[in][RHO]<0.0){state.rhs[in][nv]=0.0;}
221       if(u[in][RHO]+state.rhs[in][RHO]<0.0){state.rhs[in][nv]=0.0;}
223       if(u[in][RHO]<0.0){print("rho u[in][RHO]=%g \n state.rhs[in][RHO]=%g  \n
              3.*UU[*k][*j][*i][RHO]=%g \n",u[in][RHO],state.rhs[in][RHO],u[in][RHO]-
              state.rhs[in][RHO],3.*UU[*k][*j][*i][RHO]);
276       if(u[in][RHO]+state.rhs[in][RHO]<0.0){state.rhs[in][nv]=0.0;}
```

Listing B.14: Modifications in tc_kappa.c

```
16 /* ///////////////////////////////////////////////////////////////////// */
17 /* Modifications K. Fierlinger: no thermal conduction in the feedback region
18    lines  44-47 and 61 */
19 /* ///////////////////////////////////////////////////////////////////// */
44   if (x1<g_inputParam[R_DRIVER]){
45     *kpar = 0.0;
46     *knor = 0.0;
47   }else{
61   }
```

Listing B.15: init.c for a constant wind

```
13 /* ///////////////////////////////////////////////////////////////////// */
14 /* Modifications:
15    re-define code units                              lines 53-55
16    gamma, density and pressure from pluto.ini        lines 57, 59 and 63
17    feedback (mass+pressure) into sphere inside domain  lines 22,23,142-158
18    feedback (mass+velocity) into sphere near inner BC  lines 24,166-181
```

```
19 */
20 /* ////////////////////////////////////////////////////////////////// */
22 #define THERMALWIND 1
23 #undef THERMALWIND
24 #define INFLOWING_WIND 1
53   g_unitDensity  = 1.e-22; /* reference density (\rho_0) in units of gr/cmËĘ3 */
54   g_unitVelocity = 1.e8;   /* reference velocity (v_0) in units of cm/sec */
55   g_unitLength   = 1.e19;  /* reference length (L_0) in cm */
57   g_gamma = g_inputParam[GAMMA]; /* calls the auxillary parameter GAMMA*/
59   v[RHO] = g_inputParam[RHO_IN]; /* 1e-22 g/cm3 */
63   v[PRS] = g_inputParam[PRS_IN]; /* 1e-6 erg/cm3 */
140  if (side == 0) {    /* -- check solution inside domain -- */
141    DOM_LOOP(k,j,i){
142 #ifdef THERMALWIND
143       // Add mass and pressure
144       //
145       //  wind: v = 1e8 cm/s
146       //  wind: 3e-5 Msun/yr = 5.9673e28 g/yr = 18.9e20 g/s= 18.9e-4 1e35g/1e11s
147       //  injected mass in 1e35g per time step (g_dt in 1e11s) -> 18.9e-4*g_dt
148       // Feedback into sphere -> region volume =  math.pi/0.75e-57 * r_driver**3
149       //  wind density: 18.9e-4*0.75/pi/r_driver**3 =0.0004512/r_driver**3
150    //if(pow(x1[i],2) <= pow(g_inputParam[R_DRIVER],2)){ // driver radius, if x1[i
          ] < 0
151       if(x1[i]        <= g_inputParam[R_DRIVER]){        // driver radius
152       d->Vc[RHO][k][j][i] += g_dt*0.0004512042636655326/pow(g_inputParam[R_DRIVER
          ],3); /* 1e-22 g/cm3 for a wind with 3e-5 Msun/yr */
153        //
154        //kinetic energy density = 0.5 density v^2 = 0.5 e-22 1e16 -> 0.5e-6
155        //pressure = (gamma - 1) * energydensity
156       d->Vc[PRS][k][j][i] += (g_gamma-1.0)*g_dt*0.5*0.0004512042636655326/pow(
          g_inputParam[R_DRIVER],3); /* 1e-6 erg/cm3 for a wind with 3e-5 Msun/yr
          */
157       }
158 #endif
159    };
160  }
162  if (side == X1_BEG){  /* -- X1_BEG boundary -- */
163     BOX_LOOP(box,k,j,i){
164 #ifdef INFLOWING_WIND
165       // Add mass and velocity
166       //
167       //  wind: v = 1e8 cm/s
168       //  wind: 3e-5 Msun/yr = 5.9673e28 g/yr = 18.9e20 g/s= 18.9e-4 1e35g/1e11s
169       //  injected mass in 1e35g per time step (g_dt in 1e11s) -> 18.9e-4*g_dt
170       // ALL feedback into a sphere:
171       // boundary cell size 0.0 to g_inputParam[R_DRIVER]
172       // -> region volume =  math.pi/0.75e-57 *pow(g_inputParam[R_DRIVER],3)
173       //  wind density: 18.9e-4*0.75/pi/r_driver**3 =0.0004512/r_driver**3  for a
          wind with 3e-5 Msun/yr
174       d->Vc[RHO][k][j][i] = g_dt*0.00045/pow(g_inputParam[R_DRIVER],3); /* 1e-22 g
          /cm3, 3e-5 Msun/yr */
175       EXPAND(d->Vc[VX1][k][j][i] = 1.0; , /* [1e8 cm/s] = [1e3 km/s] */
176              d->Vc[VX2][k][j][i] = 0.0; ,
177              d->Vc[VX3][k][j][i] = 0.0;)
```

```
178         d->Vc[PRS][k][j][i] = g_inputParam[PRS_IN];   /* 1e-6 erg/cm3 */
179 #endif
180       }
181   }
```

Listing B.16: init.c as used for our 1D simulations

```
14  #include "pluto.h"
15  #define READINTRUE 88000
16  #define VISCOSITYRUN 1
17  #undef VISCOSITYRUN
18  #undef GENEVA
19  #define GENEVA 60 //use 60 Msun model from the Geneva grid
20  #undef SN_linear_vel // rather use internal boundary than linear velocity profile.
21  #define MeanFreePath 1e-4 // merge cells below mean free path
22  #undef MeanFreePath
23  /* ********************************************************************* */
24  void Init (double *v, double x1, double x2, double x3)
50  {
51  #if defined(SN_linear_vel) || defined(VISCOSITYRUN)
52    double dr, vol, r, dx;
53  #endif
54    g_unitDensity  = 1.e-22; /* reference density (\rho_0) in units of gr/cmËË3 */
55    g_unitVelocity = 1.e8;   /* reference velocity (v_0) in units of cm/sec */
56    g_unitLength   = 1.e19;  /* reference length (L_0) in cm */
57    g_minCoolingTemp =  g_inputParam[TMIN]; /* lowest temperature in Kelvin */
59
59    g_gamma = g_inputParam[GAMMA]; /* calls the auxillary parameter GAMMA*/
60    g_supernova = g_inputParam[SN]; /* read latest SN time*/
62
62    g_smallPressure = MIN(g_smallPressure, g_inputParam[PRS_IN]*0.01);
63  #if defined(SN_linear_vel) || defined(VISCOSITYRUN)
64    dx = x1[2] - x1[1]; //mesh spacing
65    //xr[i] = x1[i] + 0.5 * dx; //cell centers
66    /* ---------------------------------------------------
67       dr is the size of the initial energy deposition
68       region: 2 ghost zones.
69       --------------------------------------------------- */
70    //dr = 2.0*(g_domEnd[IDIR]-g_domBeg[IDIR])/(double)NX1;
71    // convert to full cells
72    dr = (round((g_inputParam[R_DRIVER] - g_domBeg[IDIR])/dx) ) * dx + g_domBeg[IDIR
          ];
73    /* ---------------------------------------------
74       compute region volume
75       --------------------------------------------- */
76    vol = 4.0/3.0*CONST_PI*(pow(dr,3)-pow(g_domBeg[IDIR],3));
77  #endif
78    v[RHO] = g_inputParam[RHO_IN]; /* 1e-22 g/cm3 */
79    v[VX1] = 0.0; /*initial Vx array*/
80    v[VX2] = 0.0; /*initial Vy array*/
81    v[VX3] = 0.0; /*initial Vz array*/
82    v[PRS] = g_inputParam[PRS_IN]; /* 1e-6 erg/cm3 */
83    v[TRC] = 0.0;
84    //  v[FNEUT] = 1.0; // for SNEq cooling
85  #ifdef VISCOSITYRUN
86    if ( x1 <= dr ) {
```

```
87      v[RHO] =  g_inputParam[ETH]*g_inputParam[RHO_IN]; //lower density by a factor
88    };
89  #endif
90  #ifdef SN_linear_vel
91    if ( x1 <= dr ) {
92      printf ("%g %g \n", x1,x1/dr*150.);
93      v[PRS] =  (g_gamma - 1.0)*g_inputParam[ETH]/vol;
94      v[RHO] =  0.059673/vol+g_inputParam[RHO_IN]; //3 solar masses are 0.059673e35
          g
95      v[VX1] =  sqrt(g_inputParam[EK]/0.3/(0.059673+vol*g_inputParam[RHO_IN]))*x1/dr
          ; // linear velocity profile
96    };
97  #endif
113 #ifdef READINTRUE
114   if(g_inputParam[READIN]>0.0){
115   static int first_call = 1;
116   char    fname[512];
117   if (first_call){
118     g_time=g_inputParam[READIN]*0.1578;
119     int k, input_var[6];
120     for (k = 0; k < 5; k++) input_var[k] = -1;
121     input_var[0] = RHO;
122     input_var[1] = VX1;
123     input_var[2]= PRS;
124     sprintf(fname, "grid.%04d.out", (int)g_inputParam[READIN]);
125     InputDataSet (fname, input_var);
127
127     sprintf(fname, "rho.%04d.dbl", (int)g_inputParam[READIN]);
128     InputDataRead (fname,0);
129     sprintf(fname, "vx1.%04d.dbl", (int)g_inputParam[READIN]);
130     InputDataRead (fname,1);
131     sprintf(fname, "prs.%04d.dbl", (int)g_inputParam[READIN]);
132     InputDataRead (fname,2);
134
134     first_call = 0;
135   }
136   //InputDataExpand1d(v ,x1);
137   InputDataInterpolate(v ,x1,x2,x3);
138   }
139 #endif
140 }
141 /* ********************************************************************* */
142 void Analysis (const Data *d, Grid *grid)
143 /*!
144  *  Perform runtime data analysis.
145  *
146  * \param [in] d the PLUTO Data structure
155 void UserDefBoundary (const Data *d, RBox *box, int side, Grid *grid)
175  ********************************************************************* */
176 {
177   int   i, j, k, nv;
178   double *x1, *x2, *x3;
179   double dr, dx, vol, r, de, dm, ekin_help;
189   x1 = grid[IDIR].x;
190   x2 = grid[JDIR].x;
```

```
191   x3 = grid[KDIR].x;
193
193 #ifndef VISCOSITYRUN
194   /* ------------------------------------------------------
195      cell size ... dx
196      ------------------------------------------------------ */
197   //without MPI dx = (g_domEnd[IDIR]-g_domBeg[IDIR])/(double)NX1;
198   dx=x1[2]-x1[1];
199   /* ------------------------------------------------------
200      feedback region ... dr
201      ------------------------------------------------------ */
202   // convert to full cells
203   //xr[i] = x1[i] + 0.5 * dx;
204   //dr = (floor((g_inputParam[R_DRIVER] - x1[1])/dx) + 0.5) * dx + x1[1];
205   dr = (round((g_inputParam[R_DRIVER] - g_domBeg[IDIR])/dx) ) * dx + g_domBeg[IDIR
        ];
206   if(g_time<2e-4)printf("time: %g radius of the feedback region: %g x 1e19 cm.
        Actual radius %g , %g , %g \n",g_time, g_inputParam[R_DRIVER], dr, dx,
        g_domBeg[IDIR] );
207   /* ------------------------------------------------------
208      SN region ... dr
209      ------------------------------------------------------ */
210   if (g_time >= g_supernova) {dr = g_inputParam[R_SN];};
211   /* ------------------------------------------------
212      compute region volume
213      ------------------------------------------------ */
214   vol = 4.0/3.0*CONST_PI*(pow(dr,3)-pow(g_domBeg[IDIR],3));
215 #ifdef INFLOWING_WIND
216   if (side == X1_BEG){  /* -- X1_BEG boundary -- */
217     if (box->vpos == CENTER) {
218       BOX_LOOP(box,k,j,i){
219        //  wind: v = 1e8 cm/s
220        //  wind: 3e-5 Msun/yr = 5.9673e28 g/yr = 18.9e20 g/s
221        //  wind density: 18.9e-4*0.75/pi/r_driver**3 =0.0004512/r_driver**3
222          // inflow BC
223          d->Vc[RHO][k][j][i] = g_dt*0.00045/pow(dr,3); /* 1e-22 g/cm3, 3e-5 Msun/yr
                */
224          EXPAND(d->Vc[VX1][k][j][i] = 1.0; , /* [1e8 cm/s] = [1e3 km/s] */
225                 d->Vc[VX2][k][j][i] = 0.0; ,
226                 d->Vc[VX3][k][j][i] = 0.0;)
227            d->Vc[PRS][k][j][i] = g_inputParam[PRS_IN];  /* 1e-6 erg/cm3 */
228       }
229     }else if (box->vpos == X1FACE){
230       BOX_LOOP(box,k,j,i){  }
231     }else if (box->vpos == X2FACE){
232       BOX_LOOP(box,k,j,i){  }
233     }else if (box->vpos == X3FACE){
234       BOX_LOOP(box,k,j,i){  }
235     }
236   }
237 #endif
239
239   // THIS ROUTINE WOULD BE CALLED MORE THAN ONCE: twice for RK2 three times for
        RK3 - prevent this in boundary.c
240   if (side == 0) {    /* -- check solution inside domain -- */
```

```
241 #ifdef MeanFreePath
242     // merge cells with inner cell if the hydrogen mean free path is larger than
            the cellsize
243     double gmm1 = g_gamma - 1.0;
244     DOM_LOOP(k,j,i){
245       if (x1[i] > dr+dx && d->Vc[RHO][k][j][i] < 1.e-4*dx*200){ //don't merge
              feedback cells.
246         double etherm1,etherm2,ekin1,ekin2; // energy
247         //printf ("%g dx < mean free path, i=%d , rho = %g [g/cm3]",dx, i,d->Vc[
              RHO][k][j][i]*1.e-22);
248         //energy conservation
249         etherm1=d->Vc[PRS][k][j][i]/gmm1;
250         ekin1=0.5*d->Vc[RHO][k][j][i]*pow(d->Vc[VX1][k][j][i],2);
251         etherm2=d->Vc[PRS][k][j][i-5]/gmm1;
252         ekin2=0.5*d->Vc[RHO][k][j][i-5]*pow(d->Vc[VX1][k][j][i-1],2);
253         if (ekin1<etherm1 && ekin2<etherm2) { //don't mix in free streaming zone
              or 5 point interface region
254           double V1, V2, V3; // Volumes of spheres for weighting the densities
255           double w1, w2; // weights
256           double etotal; // energy
257           etherm2=d->Vc[PRS][k][j][i-1]/gmm1;
258           ekin2=0.5*d->Vc[RHO][k][j][i-1]*pow(d->Vc[VX1][k][j][i-1],2);
259           // x[i] ... cell center coordinate
260           V1=pow(x1[i]+0.5*dx,3);
261           V2=pow(x1[i]-0.5*dx,3);
262           V3=pow(x1[i]-1.5*dx,3);
263           w1 = (V1-V2)/(V1-V3);
264           w2 = 1.0 - w1;
265           etotal = (w1*(etherm1+ekin1)+w2*(etherm2+ekin2));
266           d->Vc[RHO][k][j][i]   = w1*d->Vc[RHO][k][j][i] + w2*d->Vc[RHO][k][j][i
                -1]; /* 1e-22 g/cm3 */
267           d->Vc[PRS][k][j][i]   = w1*d->Vc[PRS][k][j][i] + w2*d->Vc[PRS][k][j][i
                -1]; /* 1e-6 erg/cm3 */
268           d->Vc[RHO][k][j][i-1] = d->Vc[RHO][k][j][i];
269           d->Vc[PRS][k][j][i-1] = d->Vc[PRS][k][j][i];
270           etotal -= d->Vc[PRS][k][j][i]/gmm1;//kinetic energy
271           d->Vc[VX1][k][j][i]  = sqrt(2.0*etotal/d->Vc[RHO][k][j][i]); //x1
                velocity
272           d->Vc[VX1][k][j][i-1] = d->Vc[VX1][k][j][i];
273         }
274       }
275     }
276 #endif
277     //printf ("time %g dt %g x0 %g \n",g_time, g_dt, g_domBeg[IDIR]);
278     if (g_time - g_dt < g_supernova)
279       {
280         if (g_time >= g_supernova) {
281           // SN
282           dr = g_inputParam[R_SN];
283           //printf("radius of the SN %g x 1e19 cm. supernova went off at %g x 1e11
                s. \n",g_inputParam[R_SN],g_time );
284           DOM_LOOP(k,j,i){
285             if ( x1[i] <= dr){// radius of the feedback zone
286               if(i==5){printf("radius␣of␣the␣SN␣%g␣x␣1e19␣cm.␣supernova␣went␣off␣
                    at␣%10.4e␣x␣1e11␣s.␣%10.4e␣\n",g_inputParam[R_SN],g_time,g_dt )
```

```
                ;}
287             /* ------------------------------------------
288                add SN energy and mass
289                ------------------------------------------ */
290             //printf("p: %g + %g e: %g x: %g\n",d->Vc[PRS][k][j][i], (g_gamma
                    -1.0)*g_inputParam[ETH]/vol,g_inputParam[ETH],x1[i]);
291             // THIS ROUTINE COULD BE CALLED MORE THAN ONCE (Predictor, Corrector
                    ..) - prevent this in boundary.c
292 #ifdef GENEVA
293             if(g_inputParam[ETH]<0.5*g_inputParam[M_SN]*pow(d->Vc[VX1][k][j][i
                    ],2)) //added energy smaller than increase of kinetic energy
294               {
295                 printf("%d SN energy: %g [FOE], kinetic energy %g  \n",i,
                        g_inputParam[ETH],0.5*g_inputParam[M_SN]*pow(d->Vc[VX1][k][j
                        ][i],2));
296                 //ek=0.5(m v^2)+de=0.5((m+dm)u^2 -> 2de/(m+dm) + m/(m+dm) v^2 =
                        u^2
297                 d->Vc[VX1][k][j][i]=MAX(0.0,pow(2.*(g_inputParam[ETH]/vol)/(
                        g_inputParam[M_SN]/vol+d->Vc[RHO][k][j][i])+pow(d->Vc[VX1][k
                        ][j][i],2))*d->Vc[RHO][k][j][i]/(g_inputParam[M_SN]/vol+d->
                        Vc[RHO][k][j][i]),0.5));//reduce velocity
298               }
299             d->Vc[PRS][k][j][i] +=  ((g_gamma - 1.0)*(g_inputParam[ETH]-0.5*
                    g_inputParam[M_SN]*pow(d->Vc[VX1][k][j][i],2)))/vol; /* 1e-6 erg
                    /cm3 */
300             d->Vc[RHO][k][j][i] +=  g_inputParam[M_SN]/vol; //10.9807 solar
                    masses in 1e35 g (10.9807 solar masses = 2.18417104e34 g )
301 #else         //Can lead to negative pressures if added energy is smaller than
    increase of kinetic energy (since energy difference added/subtracted to/from
    pressure)
302             d->Vc[PRS][k][j][i] +=  ((g_gamma - 1.0)*(g_inputParam[ETH
                    ]-0.5*0.059673*pow(d->Vc[VX1][k][j][i],2)))/vol; /* 1e-6 erg/cm3
                     */
303             d->Vc[RHO][k][j][i] +=  0.059673/vol; //3 solar masses in 1e35 g (3
                    solar masses = 0.059673e35 g )
304 #endif
305             //d->Vc[VX1][k][j][i] += sqrt(2.*g_inputParam[EK]/vol/d->Vc[RHO][k][
                    j][i]); // for density variations ... constant energy density
306             //d->Vc[VX1][k][j][i] += sqrt(g_inputParam[EK]/0.3/(vol*d->Vc[RHO][k
                    ][j][i]))*x1[i]/dr; // linear velocity profile ... for constant
                    density!
307           }
308           g_supernova=0.0;
309         }
310       }else{
311         if (g_time + g_dt >= g_supernova) {
312           g_dt = 0.0001*g_dt;
313           g_supernova=g_time+0.9*g_dt;
314           printf("reached given SN time  of %12.6e x 1e11 s. supernova will go
                    off at %12.6e x 1e11 s. \n",g_inputParam[SN],g_supernova );
315         }
316         //check bubble size and start SN
317         if (g_supernova >= g_inputParam[SN]){
318 #ifdef GENEVA
319           Wind(vol,&de,&dm);
```

```
320             //if(g_time<0.3 && g_dt > 2.e-4){g_dt = 0.1*g_dt;printf("g_dt=%g,
                    g_time=%g \n",g_dt,g_time);}
321 #endif
322             //volumecheck=0.0;
323             DOM_LOOP(k,j,i){
324               if (x1[i] >= g_inputParam[R_BUBBLE] && d->Vc[RHO][k][j][i]  <= 0.5*
                      g_inputParam[RHO_IN] )
325                 {
326                   g_dt = 0.0001*g_dt; //reduce time step size shortly before SN
327                   g_supernova=g_time+0.9*g_dt;
328                   printf("reached␣given␣cavity␣size␣of␣%g␣x␣1e19␣cm.␣supernova␣
                          will␣go␣off␣at␣%10.4e␣x␣1e11␣s.␣\n",g_inputParam[R_BUBBLE],
                          g_supernova );
329                 }
330               /* ---------------------------------------------
331                  add wind energy and mass
332                  --------------------------------------------- */
333               // Geneva 60 Msun: T_SN = 4.859663989740756 [years]
334               //                 massloss_SN =  Geneva2011V4(1:n_models)%
                      masslossSN = 10.9807 ! [solar masses]
335               //
336               // mass loss:  3e-5 Msun/yr = 5.9673e28 g/yr = 1.89e21 g/s = 0.00189
                      e35 g / 1e11 s
337               //->g_dt in 1e11s
338               // rho in 1e35 g / 1e57cm^3
339               // Feedback into sphere with radius = g_inputParam[R_DRIVER] in 1e19
                      cm
340               //-> region volume =  math.pi/0.75e-57 * pow(g_inputParam[R_DRIVER
                      ],3); in cm3
341               //0.00189/(math.pi/0.75)=0.0004512042636655233
342               //
343               //if(x1[i] <= dr && x1[i]>g_domBeg[IDIR]){  // radius of the
                      feedback zone
344               if(x1[i] <= dr){  // radius of the feedback zone
345                 //   printf("d0 %g dm %g p0 %g de %g \n", d->Vc[RHO][k][j][i] ,
                        0.5*dm, d->Vc[PRS][k][j][i] , 0.5*(g_gamma-1.0)*de);
346 #ifdef GENEVA
347                 //if(i<4){ printf("i %d t %g d %g \n",i,g_time,d->Vc[RHO][k][j][i
                        ]);}
348                 d->Vc[RHO][k][j][i] += dm;
349                 ekin_help=dm*0.5*pow(d->Vc[VX1][k][j][i],2);//kinetic energy
                        increase due to added mass
350                 if( de >= ekin_help){//added energy larger than increase of
                        kinetic energy
351                   d->Vc[PRS][k][j][i] += (g_gamma-1.0)*(de-ekin_help);//add
                          remaining energy to thermal energy
352                 }else{ // ek=0.5(m v^2)+de=0.5((m+dm)u^2 -> m/(m+dm) v^2 + 2de/(m+
                        dm) = u^2
353                   d->Vc[VX1][k][j][i]=sqrt((d->Vc[RHO][k][j][i]-dm)/d->Vc[RHO][k][
                          j][i].0*pow(d->Vc[VX1][k][j][i],2)+2.*de/d->Vc[RHO][k][j][i
                          ].0);//reduce velocity
354                 }
355 #else
356                 // THIS ROUTINE COULD BE CALLED MORE THAN ONCE if it is not
                        prevented in boundary.c
```

```
357                 d->Vc[RHO][k][j][i] += g_dt*0.00189/vol; /* 1e-22 g/cm3 ... added
                        3e-5 Msun/year */
358                 //EXPAND(d->Vc[VX1][k][j][i] = 1.0; , /* v= 1 [1e8 cm/s] = [1e3 km
                        /s] */
359                 //       d->Vc[VX2][k][j][i] = 0.0; ,
360                 //       d->Vc[VX3][k][j][i] = 0.0;)
361                 ////0.5 m v^2 (gamma-1) = 0.5 e-22 1e16 -> 0.5e-6
362                 ////energy input: m v^2 *0.5 = (3e-5 2e33) 10e16 * 0.5 erg / year
                        = 3e44 erg/year
363                 ////pressure = (gamma - 1) * internal energy density
364                 // Can lead to negative pressures for high d->Vc[VX1][k][j][i]
                        since difference
365                 // between kinetic energy increase and added wind energy taken
                        from pressure
366                 d->Vc[PRS][k][j][i] += (g_gamma-1.0)*g_dt*0.5*0.00189*(1.0-pow(d->
                        Vc[VX1][k][j][i],2))/vol; /* 1e-6 erg/cm3 ... added 3e-5 Msun/
                        year  */
367 #endif
368                 //volumecheck += pow(x1[i]+0.5*dx,3)-pow(x1[i]-0.5*dx,3);
369                 //printf("i %d x1 %g v %g sum v %g \n", i,x1[i],(pow(x1[i]+0.5*dx
                        ,3)-pow(x1[i]-0.5*dx,3))*4.*CONST_PI/3., 4.*CONST_PI/3.*
                        volumecheck);
370               }
371             }
372           }
373           //printf("vol: %g, volumcheck: %g",vol,vol-4.*CONST_PI/3.*volumecheck);
374           //QUIT_PLUTO(1);
375 #ifdef PARALLEL
376           double g_supernova_local = g_supernova;
377           MPI_Allreduce (&g_supernova_local, &g_supernova, 1, MPI_DOUBLE, MPI_MIN,
                  MPI_COMM_WORLD);
378           g_supernova_local = g_dt;
379           MPI_Allreduce (&g_supernova_local, &g_dt, 1, MPI_DOUBLE, MPI_MIN,
                  MPI_COMM_WORLD);
380 #endif
381         }
382       }
383   }
384 #endif
385 }
387 
387 #define year_to_seconds  31556926.0  /* seconds per year */
388 #define msun_to_g        1.9891e33   /* g per solar mass */
389 #define msunYear_to_gs   6.30321217e25  /* converts Msun/yr to g/s */
390 /* ************************************************************* */
391 void Wind (double vol, double *e, double *m)
392 /*
393  * NAME      Wind
394  * PURPOSE  Provide wind mass and energy input (tabulated 60 Msun model).
395  *
396  ************************************************************** */
397 {
398   int     klo, khi, kmid; //table element number for binary search
399   static int ntab=400; //table size
400   static real *t_tab, *m_tab, *e_tab; // years, solar masses per year, 1e-30 erg/s
```

```
401   static double convertE,convertM;
402   double     t,Tmid, dT;
403   FILE *fcool;
405
405   //printf("vol %g dm %g de %g \n", vol , *m, *e);
406   /* ---------------------------------------------------
407      Load Table
408      --------------------------------------------------- */
409   if (t_tab == NULL){
410     print1 ("␣>␣Reading␣Geneva␣table␣from␣disk...\n");
411     fcool = fopen("Geneva2011V4.dat","r");
412     if (fcool == NULL){
413       print1 ("!␣wind␣table␣does␣not␣exist\n");
414       QUIT_PLUTO(1);
415     }
416     t_tab = ARRAY_1D(400, double);
417     e_tab = ARRAY_1D(400, double);
418     m_tab = ARRAY_1D(400, double);
420
420     ntab = 0;
421     while (fscanf(fcool, "%lf␣␣%lf␣%lf␣\n", t_tab + ntab,
422                   m_tab + ntab, e_tab + ntab)!=EOF) {
423       ntab++;
424     }
425     printf("ntab:␣%d,␣tmax:␣%g␣years␣\n", ntab,t_tab[ntab-1]);
426     /* ---------------------------------------------------- */
427     //erg to code unit: 1.0/pow(g_unitLength, 3.0)/g_unitDensity/pow(
          g_unitVelocity, 2.0);
428     convertE=1.e30/pow(g_unitLength,2.0)/g_unitDensity/pow(g_unitVelocity, 3.0);
          //  [1e30 erg / s] * code time step to code energy units
429     convertM=msunYear_to_gs/g_unitVelocity/pow(g_unitLength, 2.0)/g_unitDensity;
          //  [Msun / yr]    * code time step to code mass units
430   }
431   /* ---------------------------------------------------
432      Get time
433      --------------------------------------------------- */
434      t= g_time*g_unitLength/g_unitVelocity/year_to_seconds; //converts code units
          to years
435   /* ---------------------------------------------------
436      Table lookup by binary search
437      ---------------------------------------------------*/
438   klo = 0;
439   khi = ntab - 1;
440   if (t < t_tab[klo] ){
441     *e = e_tab[klo];
442     *m = m_tab[klo];
443   }
444   else if (t > t_tab[khi]){
445     *e = 0.0;
446     *m = 0.0;
447     return;
448   }
449   else{
450     while (klo != (khi - 1)){
451       kmid = (klo + khi)/2;
```

```
452         Tmid = t_tab[kmid];
453         if (t <= Tmid){
454           khi = kmid;
455         }else if (t > Tmid){
456           klo = kmid;
457         }
458       }
459       dT  = t_tab[khi] - t_tab[klo];
460   // interpolation: find rate at given time and use rectangles to integrate.
461   // to get trapezoidal numerical integration better use klo and khi value and
      compute rise
462       *m  = 0.5*(m_tab[klo]+ m_tab[khi]); //in [Msun / s]
463       *e  = 0.5*(e_tab[klo]+ e_tab[khi]); //in [Msun / s]
464     }
465     //printf ("T:%g [years] M:%g [Msun/year] E:%g [1e30erg/s] \n", t,*m,*e);
466     //printf ("convertM %g 1./vol %g g_dt %g  \n", convertM, 1./vol, g_dt);
467     //printf ("convertE %g 1./vol %g g_dt %g  \n", convertE, 1./vol, g_dt);
468     //printf ("T:%g [code] M:%g [code] E:%g [code] \n", g_time,m*convertM,e*convertE
        );
469     *m *= g_dt*convertM/vol;
470     *e *= g_dt*convertE/vol;
471     return;
472   }
```

Listing B.17: example of pluto.ini

```
1   [Grid]
3
3   X1-grid     1     0.01    2400    u     6.01
4   X2-grid     1     0.0     1       u     1.0
5   X3-grid     1     0.0     1       u     1.0
7
7   [Chombo Refinement]
9
9   Levels           4
10  Ref_ratio        2 2 2 2 2
11  Regrid_interval  2 2 2 2
12  Refine_thresh    0.3
13  Tag_buffer_size  3
14  Block_factor     8
15  Max_grid_size    64
16  Fill_ratio       0.75
18
18  [Time]
20
20  CFL              0.4
21  CFL_max_var      1.1
22  tstop            15.79
23  first_dt         1.e-9
25
25  [Solver]
27
27  Solver           roe
29
29  [Boundary]
31
```

```
X1-beg          reflective
X1-end          outflow
X2-beg          reflective
X2-end          reflective
X3-beg          reflective
X3-end          reflective

[Static Grid Output]

uservar     0
dbl         0.1578 -1 multiple_files
flt         -1.0  -1   single_file
vtk         -1.0  -1   single_file
tab         -1.0  -1
ppm         -1.0  -1
png         -1.0  -1
log          100
analysis    -1.0  -1

[Chombo HDF5 output]

Checkpoint_interval  -1.0  0
Plot_interval         1.0  0

[Parameters]

RHO_IN      0.022
PRS_IN      3.9889e-7
GAMMA      1.66666666667
R_DRIVER      0.1
ETH      1.0
EK      0.0
R_BUBBLE      1.22
SN      7.0
R_SN      0.15
TMIN      50.0
M_SN      0
RHO_MIN      0
READIN      0
```

Listing B.18: customized definitions.h

```
#define  PHYSICS                HD
#define  DIMENSIONS             1
#define  COMPONENTS             1
#define  GEOMETRY               SPHERICAL
#define  BODY_FORCE             NO
#define  COOLING                TABULATED
#define  INTERPOLATION          LINEAR
#define  TIME_STEPPING          RK2
#define  DIMENSIONAL_SPLITTING  YES
#define  NTRACER                1
#define  USER_DEF_PARAMETERS    13

/* -- physics dependent declarations -- */

```

```
#define     EOS                     IDEAL
#define     ENTROPY_SWITCH          NO
#define     THERMAL_CONDUCTION      NO
#define     VISCOSITY               NO
#define     ROTATING_FRAME          NO

/* --- pointers to user-def parameters --- */

#define   RHO_IN                0
#define   PRS_IN                1
#define   GAMMA                 2
#define   R_DRIVER              3
#define   ETH                   4
#define   EK                    5
#define   R_BUBBLE              6
#define   SN                    7
#define   R_SN                  8
#define   TMIN                  9
#define   M_SN                  10
#define   RHO_MIN               11
#define   READIN                12

/* --- supplementary constants (user editable) --- */

#define   INITIAL_SMOOTHING     NO
#define   WARNING_MESSAGES      YES
#define   PRINT_TO_FILE         NO
#define   INTERNAL_BOUNDARY     YES
#define   SHOCK_FLATTENING      ONED
#define   ARTIFICIAL_VISCOSITY  NO
#define   CHAR_LIMITING         YES
#define   LIMITER               MINMOD_LIM
```

Listing B.19: post processing routine

```
#include <stdio.h>   /* required for file operations */
#include <math.h>   /* required for pow(n,3) */

FILE *ft,*fr,*fp,*fv;            /* declare the file pointer */

main( int argc, char *argv[] )

{
   int n;
   char bytes[8];
   char filenamet[128],filenamer[128],filenamep[128],filenamev[128];
   double m,t,ek,et,ekintot,ethermtot,ezero;
   double rhomax=0.0, trhomax=0.0, vshell=0.0, rho0=0.23, shellmass=0.0, shellv
       =0.0;
   int nrhomax=0, shellwidth=0,supersonic=0;
   //60e19 cm 24000 cells
   double dV = 3.14159/400./400./400./0.75, dV1;
   // debug: printf("%s\n",argv[1]);

   sprintf(filenamet, "tr1.%s.dbl", argv[1]);
   sprintf(filenamer, "rho.%s.dbl", argv[1]);
```

```
21      sprintf(filenamep, "prs.%s.dbl", argv[1]);
22      sprintf(filenamev, "vx1.%s.dbl", argv[1]);
24
24      //debug: printf("%s %s %s",filenamer,filenamep,filenamev);
26
26      ft = fopen (filenamet, "r");  /* open the file for reading */
27      fr = fopen (filenamer, "r");  /* open the file for reading */
28      fp = fopen (filenamep, "r");  /* open the file for reading */
29      fv = fopen (filenamev, "r");  /* open the file for reading */
31
31      ezero=0.0; // initial thermal energy
32      ekintot=0.0;
33      ethermtot=0.0;
34      n=0;
35      m=0.0;
36      printf("#(1) cell number \n");
37      printf("#(2) density [1e-22 g/cm3]\n");
38      printf("#(3) pressure [1e-6 erg/cm3]\n");
39      printf("#(4) velocity [1e8 cm/s]\n");
40      printf("#(5) temperature [K]\n");
41      printf("#(6) thermal energy [1e-6 erg/cm3]\n");
42      printf("#(7) kinetic energy [1e-6 erg/cm3]\n");
43      printf("#(8) cumulative mass [1e35 g] \n");
44      printf("#(9) cooling loss [erg/cm3/s] \n");
45      while(n<10000)
46       {
47       fread(&bytes, 8, 1, ft);
48       double tr = *((double*)bytes);//erg cm3/s
49       fread(&bytes, 8, 1, fr);
50       double d = *((double*)bytes); //1e-22 g/cm3
51       fread(&bytes, 8, 1, fp);
52       double p = *((double*)bytes); //1e-6 erg/cm3
53       fread(&bytes, 8, 1, fv);
54       double v = *((double*)bytes); //1e3 km/s
55       // inner boundary: 0.01 -> 400 * 0.01 = 4
56       dV1=dV*(double)(pow((n+5),3)-pow((n+4),3));
57       //cell centered radii
58       //dV1=dV*(pow(((double)(n)+4.5),3)-pow(((double)(n)+3.5),3));
59       n++;
60  // CONST_amu      1.66053886e-24      /**<  Atomic mass unit.          */
61  // CONST_kB       1.3806505e-16       /**<  Boltzmann constant.        */
62  // KELVIN (g_unitVelocity*g_unitVelocity*CONST_amu/CONST_kB)
63  // KELVIN (1e16*1.66053886e-24/1.3806505e-16)
64  // KELVIN (1e16*1.66053886e-8/1.3806505)
65  // KELVIN (1.66053886e8/1.3806505)=120272209.
66       //mu=0.5
67       //X=1-0.082-1e-3=0.917
68       t=120272209.*p/d*0.5*0.917;
69       if(d>rhomax){trhomax=t;nrhomax=n; rhomax = d; vshell=v;}
70       if(d>rho0){shellwidth+=1;shellmass+=dV1*d;shellv+=dV1*d*v;if(v>pow(1.666667*p/
             d,0.5)){supersonic+=1;}}
71       ek=0.5*d*v*v;
72       //one over gamma-1:  1.5 = 1./(5./3.-1.) = 1 / (gamma - 1)
73       et=1.5*p;
74       //thermal energy of initial conditions: 1.5*p0 = 1.5*7.30974e-7
```

```
75      if(p < 7.30973e-7 ||  p > 7.30975e-7){
76      ezero +=1.096461e-06*dV1;
77      ethermtot+=et*dV1;
78      }
79      ekintot+=ek*dV1;
80      m+=d*dV1;
81      printf("%d %g %g %g %g %g %g %g %g %g\n",n,d,p,v,t,et,ek,m,tr,d*dV1);
82      }
84
84    fclose(fr);  /* close the file prior to exiting the routine */
85    fclose(fp);  /* close the file prior to exiting the routine */
86    fclose(fv);  /* close the file prior to exiting the routine */
87    //printf("# %s+13111 %g %g %g t[0.5kyr] Ekin Etherm Etot [FOE] rhomax %d %g %g
          %g ezero %g shell %d %g %g %d \n",argv[1],ekintot, ethermtot, ekintot+
          ethermtot, nrhomax, rhomax, trhomax, vshell, ezero, shellwidth, shellv/
          shellmass,shellmass,supersonic) ;
88    printf("# %d %g %g %g t[0.5kyr] Ekin Etherm Etot [FOE] rhomax %d %g %g %g ezero
          %g shell %d %g %g %d \n",atoi(argv[1])+13111,ekintot, ethermtot, ekintot+
          ethermtot, nrhomax, rhomax, trhomax, vshell, ezero, shellwidth, shellv/
          shellmass,shellmass,supersonic) ;
89  } /*of main*/
```

Listing B.20: shell script with automatic expansion of the volume

```
1   #!/bin/bash
2   ZAHL=0 #factor for cooling threshold: ${ZAHL}.${COUNTER}
3   COUNTER=0
4   NMAX=10
5   PLUSMYR=2 #duration of individual simulations
6   OLDAMBIENT=0 # grid point where the undisturbed medium starts in the last
        simulation + buffer
7   #while [  $ZAHL -lt 2 ]; do
8   # let COUNTER=2-2*ZAHL
9   # let COUNTER=1-ZAHL
10  # while [  $COUNTER -lt $NMAX ]; do
11      echo The ratio is ${ZAHL}.${COUNTER}
12      # Control will enter here if $DIRECTORY exists.
13      if [  -d "${ZAHL}p${COUNTER}/nh100" ]; then
14       #find output with highest number
15       STARTFILE=$(ls ${ZAHL}p${COUNTER}/nh100/rho.?????.dbl | tail -n 1 | sed -e 's
             /^\([0-9]p[0-9]\/nh100\/rho\)\(\.\)\([0-9]*\).*/\3/')
16       #write zero if no file is found
17       let STARTFILE=STARTFILE
18       grep ghosts ${ZAHL}p${COUNTER}/nh100/job* | awk 'BEGIN{max=800}{if($6>max){max
             =$6};print $0}END{print max}'
19       AMBIENT=$(grep ghosts ${ZAHL}p${COUNTER}/nh100/job* | awk 'BEGIN{max=800}{if(
             $6>max){max=$6}}END{print max}')
20       gcc -cpp -DXMAX=$(expr ${AMBIENT} / 100 ) -o writeascii asciiTe4.c -lm
21      else
22       if [ ! -d "${ZAHL}p${COUNTER}" ]; then
23         mkdir ${ZAHL}p${COUNTER}
24       fi
25       mkdir ${ZAHL}p${COUNTER}/nh100
26       STARTFILE=0
27       OLDAMBIENT=0
28       AMBIENT=800
```

```
29    fi
30    echo Startfile $STARTFILE
31    REWND=0
32    while [ $STARTFILE -lt 60000 ]; do
33     if [ $STARTFILE -gt 0 ]; then
34      cd ${ZAHL}p${COUNTER}/nh100
35      #copy output with highest number and ini file
36      cp pluto.ini pluto${STARTFILE}.ini
37      cp *.${STARTFILE}.dbl ../..
38      cp dbl.out dbl.${STARTFILE}.out
39      cp dbl.${STARTFILE}.out ../../dbl.out
40      cp grid.out grid.${STARTFILE}.out
41      cp grid.${STARTFILE}.out ../..
42      cp restart.out restart.${STARTFILE}.out
43      MYR=$(bc <<<"scale=2;${STARTFILE}␣/␣2000␣")
44      ../../writeascii ${STARTFILE} > ${MYR}Myr.txt
45      echo ${MYR}Myr.txt
46      if [ $REWND -eq 0 ]; then
47       let OLDAMBIENT=AMBIENT
48       #OLDAMBIENT=$(awk 'BEGIN{n=0}{n++;if( n>12 && $1 != "#" && $5 != "-nan" ){
             ambient2=$1}}END{print ambient2}' ${MYR}Myr.txt)
49      else
50       REWND=0
51      fi
52      if [ $STARTFILE -gt 16000 ]; then
53       # ensure 200 to 300 points of ambient medium @ right box side @ restart
54       AMBIENT=$(awk 'BEGIN{rho0=2.2;ambient=0;ambient2=0}{if($2>rho0 && $1 != "#")
             {ambient=$1};if($2==rho0 && ( $4 != 0 ) && ( $5 != "-nan") ){ambient2=$1
             }}END{print ambient2-ambient2%100+300}' ${MYR}Myr.txt)
55      elif [ $STARTFILE -gt 9999 ]; then
56       # ensure 400 to 500 points of ambient medium @ right box side @ restart
57       AMBIENT=$(awk 'BEGIN{rho0=2.2;ambient=0;ambient2=0}{if($2>rho0 && $1 != "#")
             {ambient=$1};if($2==rho0 && ( $4 != 0 ) && ( $5 != "-nan") ){ambient2=$1
             }}END{print ambient2-ambient2%100+500}' ${MYR}Myr.txt)
58      else
59       # ensure 200 to 300 points of ambient medium @ right box side @ restart
60       AMBIENT=$(awk 'BEGIN{rho0=2.2;ambient=0;ambient2=0}{if($2>rho0 && $1 != "#")
             {ambient=$1};if($2==rho0 && ( $4 != 0 ) && ( $5 != "-nan") ){ambient2=$1
             }}END{print ambient2-ambient2%100+300}' ${MYR}Myr.txt)
61      fi
62      echo $OLDAMBIENT $AMBIENT $MYR $STARTFILE
63      if [ $OLDAMBIENT -gt $AMBIENT ]; then
64       cd ../..
65       echo "shell␣left␣box"
66       #Rewind until a snapshot where stell is still in the box is reached
67       REWND=1;
68       #exit 2
69      else
70       if [ -f pluto.ini ]; then
71        cp pluto.ini pluto${STARTFILE}.ini
72       fi
73       cd ../..
74      fi
75     fi
76     if [ $REWND -eq 0 ]; then
```

```
77      awk -v restart="${STARTFILE}" -v points="${AMBIENT}" -v n="${ZAHL}.${COUNTER}
            " -v myr="${PLUSMYR}"  '{
78       if($1=="X1-grid"){$4=points;$6=points*0.01+0.01;};
79       if($1=="tstop"){$2=(myr+restart*0.0005)*315.61;};
80       if($1=="RHO_MIN"){$2=n};
81       if($1=="READIN"){$2=restart};
82       print $0}' pluto2.init > pluto.ini
83      nohup nice -n 19 mpirun -np 4 ../MyCode/pluto_RHO_COOL_MIN > job${STARTFILE}.
            ${ZAHL}p${COUNTER}.out
84      gcc -cpp -DXMAX=$(expr ${AMBIENT} / 100 ) -o writeascii asciiTe4.c -lm
85      ./ascii.shell $STARTFILE $(expr 2000 \* ${PLUSMYR} + $STARTFILE )
86      sed 's/\# //g' energy.txt >> ${ZAHL}p${COUNTER}/nh100/energy.txt
87      rm energy.txt
88      ./writeascii $(expr 2000 \* ${PLUSMYR} + $STARTFILE ) > $(bc <<<"scale=2;␣${
            STARTFILE}␣/␣2000␣+␣${PLUSMYR}␣")Myr.txt
89      mv *txt *ini *dbl *out ${ZAHL}p${COUNTER}/nh100
90      let STARTFILE=2000*PLUSMYR+STARTFILE
91     else
92      let STARTFILE=STARTFILE-REWND
93     fi
94    done
95 #   let COUNTER=COUNTER+1
96 # done
97 # let ZAHL=ZAHL+1
98 # NMAX=4
99 #done
```

Appendix C

Ramses source code listings

The listings in this section contain patches for RAMSES version 3.10 git commit 792ce06 (first seven digits)
from August, 27th 2014 at https://bitbucket.org/rteyssie/ramses

Listing C.1: New module with a feedback routine for Ramses: driver.f90

```
!> \short reads and interpolates driver data; calculates weights for a homogeneous, circular driver region
!-------------------------------------------------------------------------
!> \version 1.5
!> \author Katharina M. Fierlinger
!> \date last modification 27.01.2012
!-------------------------------------------------------------------------
!> \details  PURPOSE:
!>  \n  read and interpolate:
!>  \n  * driver mass loss (per time unit)
!>  \n  * driver energy production (per time unit)
!>  \n  * driver wind speed
!>  \n file_driver ... name of driver file
!>  \n assume that the driver data are stored in a file called "file_driver"
!>  that is stored in the local directory
!> \n dp ... precision
!> \n file_driver ... name of driver file
!-------------------------------------------------------------------------
!> ifdef SMOOTH_DRIVER_EDGE ... calculate weights for cells partly inside the driver region
!> This definition should be made at compile time. You can also hard-code it here.
!> #define SMOOTH_DRIVER_EDGE 1
!> #undef SMOOTH_DRIVER_EDGE
module driver
 use amr_parameters, only: dp, file_driver, file_sn
 implicit none
 save ! retain the value of the variables from one call to the next
 integer, parameter :: i9 = selected_int_kind(r=9) !< integer type definition
 real(dp), private :: endtimedriver = 0.0_dp !< for times later than this no driver data exists (time in code units)
 real(dp), dimension(:), allocatable, private :: timedriver !< array containing times (code-time-units) at which driver data is available
 real(dp), dimension(:), allocatable, private :: eidriver   !< array containing energy output per unit time (in code-energy-units per code-time-unit) at times stored in timedriver
```

```
30  real(dp), dimension(:), allocatable, private :: dMdriver   !< array containing
       mass output per unit time (in code-mass-units per code-time-units) at times
       stored in timedriver
31  real(dp), dimension(:), allocatable, private :: veldriver  !< array containing
       wind speeds (in code-length-units per code-time-units) at times stored in
       timedriver
32  real(dp), dimension(:), allocatable, private :: al26driver !< array containing \
       f$^{26}{\rm Al}\f$ (in code-mass-units per code-time-units) at times stored
       in timedriver
33  real(dp), dimension(:), allocatable, private :: fe60driver !< array containing \
       f$^{60}{\rm Fe}\f$ (in code-mass-units per code-time-units) at times stored
       in timedriver
34  real(dp), dimension(:), allocatable, private :: timeSN !< array containing times
       (code-time-units) when SN explode
35  real(dp), dimension(:), allocatable, private :: eSN !< array containing SN
       energies (in code-energy-units) at times stored in timeSN
36  real(dp), dimension(:), allocatable, private :: mSN !< array containing SN mass
       loss (in code-mass-units) at times stored in timeSN
38
38  type driver_mask
39    integer :: halfsize
40    real(dp) :: dx
41    real(dp) :: volume
42 #if NDIM==1
43    real(dp),dimension(:)      ,allocatable::weights
44 #endif
45 #if NDIM==2
46    real(dp),dimension(:,:)    ,allocatable::weights
47 #endif
48 #if NDIM==3
49    real(dp),dimension(:,:,:)  ,allocatable::weights
50 #endif
51  end type driver_mask
53
53  type(driver_mask), dimension(:), allocatable :: driver1
55
55 contains
56 ! subroutine read_driver(ntime)
57 !> \short reads driver data
58 !> \param ntime ... read first ntime lines of driver data from a file called "
       file_driver" in the local dir
59 !-------------------------------------------------------------------------
60 !> \version 1.6
61 !> \author Katharina M. Fierlinger
62 !> \date last modification 10.08.2011
63 !-------------------------------------------------------------------------
64 !> \details  PURPOSE:
65 !> \n read first "ntime" lines of driver data from a file called "file_driver" in
       the local dir
66 !> \n file_driver ... name of driver file
67 !> \n change on 03-03-2009: driver file has an additional column for 26-Al yields
68 !> \n change on 10-12-2009: driver file has an additional column for 60-Fe yields
69 !> \n change on 10-08-2011: "zero energy" now possible
70 !-------------------------------------------------------------------------
71 !> \n driver file contents:
```

```
!> \n column 1: time from starformation (in years)
!> \n column 2: cumulative output of 26Al (in Msol)
!> \n column 3: cumulative output of 60Fe (in Msol)
!> \n column 4: UV radiation (photons/s)
!> \n column 5: energy emitted in winds (log(erg/s))
!> \n column 6: energy emitted in supernovae (log(erg/s))
!> \n column 7: mass ejected by supernova (Msol/year)
!> \n column 8: mass ejected in winds (Msol/year)
!-----------------------------------------------------------------------
subroutine read_driver(ntime)
 implicit none
 integer(i9), intent(in), optional :: ntime !< ntime ... read first ntime lines of
     driver data from a file called "file_driver" in the local dir
 integer(i9) :: nlines = 0_i9 !< number of lines read from driver file
 integer(i9) :: i      = 1_i9 !< for do loop
 integer(i9) :: ifEOF  = 0_i9 !< checks when the end of the file is reached
 integer(i9) :: error_alloc !< checks if memory allocation works
 real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
    factors between cgs and user units (subroutine units in units.f90)
 real(dp) :: scale_energy, scale_m, scale_dm !< conversion factors between cgs and
     user units
 real(dp) :: timestep !< time interval between the current line and the last line
    in in years
 real(dp) :: old26Al,old60Fe,oldtime=0.0_dp !< stores values from the last line (
    to convert cumulative data to fluxes)
 real(dp) :: sumenergy,summass,sum26Al,sum60Fe,sum !< mean yields (averaged over
    tsum)
 real(dp) :: col1,col2,col3,col4,col5,col6,col7,col8 !< reads data from the 8
    columns in the input file
 real(dp) :: tsum=1.d7 !< timeinterval for mean yields in years
 real(dp), parameter :: YearToSeconds = 31556926._dp !< convert years to seconds;
    1 year = 31556926 seconds
 real(dp), parameter :: SolarMass = 1.98892e33_dp !< solar mass in [g]
 open (1,file=TRIM(file_driver),form='formatted')
 print*,"Reading␣driver␣data␣from␣>>", TRIM(file_driver), "<<␣."
 if (present(ntime)) then
   nlines = ntime
   print*,"searching␣for␣",nlines, "␣lines␣in␣driver␣file"
 else
   nlines = 0_i9
   ifEOF  = 0_i9
   do
    read(1,*,IOSTAT=ifEOF) endtimedriver, endtimedriver, endtimedriver, &
&                          endtimedriver, endtimedriver, endtimedriver, &
&                          endtimedriver, endtimedriver
    if (ifEOF.lt.0_i9) then
      exit ! eof is reached, jump out of the do-loop
    end if
    nlines=nlines+1
   end do
   rewind(1)
   print*,"found␣",nlines, "␣lines␣in␣driver␣file"
 end if
 allocate(timedriver(1:nlines+2),stat=error_alloc) ! in code-time-units
 if (error_alloc /= 0) then
```

```
119     stop 'exiting: allocation of memory for driver data did not work'
120   end if
121   allocate(eidriver(1:nlines+2),stat=error_alloc)    ! in code-energy-units per code
        -time-unit
122   if (error_alloc /= 0) then
123     stop 'exiting: allocation of memory for driver data did not work'
124   end if
125   allocate(dMdriver(1:nlines+2),stat=error_alloc)    ! in code-mass-units per code-
        time-unit
126   if (error_alloc /= 0) then
127     stop 'exiting: allocation of memory for driver data did not work'
128   end if
129   allocate(veldriver(1:nlines+2),stat=error_alloc)   ! in code-length-units per code
        -time-unit
130   if (error_alloc /= 0) then
131     stop 'exiting: allocation of memory for driver data did not work'
132   end if
133   allocate(al26driver(1:nlines+2),stat=error_alloc) ! in code-mass-units per code-
        time-unit
134   if (error_alloc /= 0) then
135     stop 'exiting: allocation of memory for driver data did not work'
136   end if
137   allocate(fe60driver(1:nlines+2),stat=error_alloc) ! in code-mass-units per code-
        time-unit
138   if (error_alloc /= 0) then
139     stop 'exiting: allocation of memory for driver data did not work'
140   end if
141   ifEOF = 0_i9
142   old26Al=0.0_dp
143   old60Fe=0.0_dp
144   oldtime=0.0_dp
145   sumenergy=0.0_dp
146   summass=0.0_dp
147   sum26Al=0.0_dp
148   sum60Fe=0.0_dp
149   sum=0.0_dp
150   timestep=0.0_dp
152
152   call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
153   scale_m=scale_d*scale_l**3 !10^(+35)
154   !1 year = 31556926 seconds
155   !mass loss = 1 solar mass / year
156   !mass loss = \f$ \frac{1.98892 \times 10^{33}}{31556926}  \frac{\rm [g]}{\rm [s]}
        \f$
157   !mass loss = 63.e+24 g/s
158   scale_dm=SolarMass/YearToSeconds*scale_t/scale_m
159   !kinetic luminosity: erg/s = g cm^2 / s^3
160   !1 erg/s = 10^(-35-2*19+3*11) code-mass units code length units^2/code-time-units
        ^3
161   !1 erg/s = 10^(-40) code-mass units code length units^2/code-time-units^3
162   scale_energy=scale_t/scale_m/scale_v**2 !10^(-40)
163   do i=1,nlines
164    read(1,*,IOSTAT=ifEOF) col1,col2,col3,col4,col5,col6,col7,col8
165    timedriver(i)=col1*YearToSeconds/scale_t
166    dMdriver(i)=(col7+col8)*scale_dm
```

```
167     if(col5.gt.0.0)then
168        eidriver(i)=(10.0_dp**(col5))*scale_energy
169     else
170        eidriver(i)=0.0_dp
171     end if
172     if(col6.gt.0.0)then
173        eidriver(i)=eidriver(i)+(10.0_dp**(col6))*scale_energy
174     end if
175     timestep=col1-oldtime
176     if(timestep.le.0.0_dp)then
177       al26driver(i)=0.0_dp
178       fe60driver(i)=0.0_dp
179     else
180       al26driver(i)=(col2-old26Al)/timestep*scale_dm
181       fe60driver(i)=(col3-old60Fe)/timestep*scale_dm
182     end if
183     old26Al=col2
184     old60Fe=col3
185     oldtime=col1
186     if(col1.le.tsum)then
187       sumenergy=sumenergy+eidriver(i)
188       summass=summass+dMdriver(i)
189       sum26Al=sum26Al+al26driver(i)
190       sum60Fe=sum60Fe+fe60driver(i)
191       sum=sum+1._dp
192     end if
193     if (ifEOF.gt.0_i9) then
194      print*, 'Something␣went␣wrong␣during␣read␣in␣of␣the␣driver␣data.'
195     else if(ifEOF.lt.0_i9) then
196      print*, 'End␣of␣file␣reached␣at␣line␣', i
197     end if
198   end do
199   close(1)
200   timedriver(nlines+1)=timedriver(nlines)*10.0_dp
201   dMdriver(nlines+1)  =dMdriver(nlines)
202   eidriver(nlines+1)  =eidriver(nlines)
203   al26driver(nlines+1)=al26driver(nlines)
204   fe60driver(nlines+1)=fe60driver(nlines)
205   timedriver(nlines+2)=timedriver(nlines)*100.0_dp
206   dMdriver(nlines+2)  =summass/sum
207   eidriver(nlines+2)  =sumenergy/sum
208   al26driver(nlines+2)=sum26Al/sum
209   fe60driver(nlines+2)=sum60Fe/sum
210 !no velocities in analyt.dat
211   veldriver(:)=0.0_dp
212   endtimedriver = timedriver(nlines)
213 end subroutine read_driver
214 ! subroutine read_sn(ntime)
215 !> \short reads driver SN data
216 !> \param ntime ... read first ntime lines of driver data from a file called "
      file_sn" in the local dir
217 !-------------------------------------------------------------------------
218 !> \version 1.0
219 !> \author Katharina M. Fierlinger
220 !> \date last modification 10.08.2011
```

```
221 !------------------------------------------------------------------------
222 !> \details  PURPOSE:
223 !> \n read first "ntime" lines of driver data from a file called "file_sn" in the
        local dir
224 !> \n file_sn ... name of driver supernova file
225 !------------------------------------------------------------------------
226 !> \n driver file contents:
227 !> \n column 1: time from starformation (in years)
228 !> \n column 6: energy emitted in supernovae (in 1e51 erg)
229 !> \n column 7: mass ejected by supernova (in Msol)
230 !------------------------------------------------------------------------
231 subroutine read_sn(ntime)
232  implicit none
233  integer(i9), intent(in), optional :: ntime !< ntime ... read first ntime lines of
         driver data from a file called "file_driver" in the local dir
234  integer(i9) :: nlines = 0_i9 !< number of lines read from driver file
235  integer(i9) :: i      = 1_i9 !< for do loop
236  integer(i9) :: ifEOF  = 0_i9 !< checks when the end of the file is reached
237  integer(i9) :: error_alloc   !< checks if memory allocation works
238  real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
        factors between cgs and user units (subroutine units in units.f90)
239  real(dp) :: scale_energy, scale_m, scale_dm !< conversion factors between cgs and
         user units
240  real(dp) :: col1,col2,col3 !< reads data from the 3 columns in the input file
241  real(dp), parameter :: YearToSeconds = 31556926._dp !< convert years to seconds;
        1 year = 31556926 seconds
242  real(dp), parameter :: SolarMass = 1.98892e33_dp !< solar mass in [g]
243  open (1,file=TRIM(file_sn),form='formatted')
244  print*,"Reading␣driver␣data␣from␣>>", TRIM(file_sn), "<<␣."
245  if (present(ntime)) then
246    nlines = ntime
247    print*,"searching␣for␣",nlines, "␣lines␣in␣driver␣file"
248  else
249    nlines = 0_i9
250    ifEOF  = 0_i9
251    do
252     read(1,*,IOSTAT=ifEOF) col1, col2, col3
253     if (ifEOF.lt.0_i9) then
254       exit ! eof is reached, jump out of the do-loop
255     end if
256     nlines=nlines+1
257    end do
258    rewind(1)
259    print*,"found␣",nlines, "␣lines␣in␣SN␣file"
260  end if
261  allocate(timeSN(1:nlines+2),stat=error_alloc) ! in code-time-units
262  if (error_alloc /= 0) then
263    stop 'exiting:␣allocation␣of␣memory␣for␣driver␣sn␣data␣did␣not␣work'
264  end if
265  allocate(eSN(1:nlines+2),stat=error_alloc)   ! in code-energy-units per code-time
        -unit
266  if (error_alloc /= 0) then
267    stop 'exiting:␣allocation␣of␣memory␣for␣driver␣sn␣data␣did␣not␣work'
268  end if
269  allocate(mSN(1:nlines+2),stat=error_alloc)   ! in code-mass-units per code-time-
```

```
      unit
270   if (error_alloc /= 0) then
271     stop 'exiting: allocation of memory for driver sn data did not work'
272   end if
274
274   call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
275   scale_m=scale_d*scale_l**3 !convert mass in g to code-mass-units
276   !mass loss = 1 solar mass
277   !mass loss = \f$ 1.98892 \times 10^{33} \rm [g] \f$
278   scale_dm=SolarMass/scale_m ! solar masses to code-mass-units
279   !SN energy: erg = g cm^2 / s^2
280   !1 erg = 10^(-35-2*19+2*11) code-mass-units code-length-units^2/code-time-units^2
281   scale_energy=10._dp**(51.0_dp-log10(scale_m)-2*log10(scale_v)) !FOE to code-
        energy-units
282   do i=1,nlines
283     read(1,*,IOSTAT=ifEOF) col1,col2,col3
284     timeSN(i)=col1*YearToSeconds/scale_t
285     eSN(i)=col2*scale_energy ! convert energy in FOE into code-energy-units
286     mSN(i)=col3*scale_dm     ! convert mass in solar masses to code-mass-units
287     print*, "SN (",i,"): at",timeSN(i),"code-time-units, mass",mSN(i),"code-mass-
          units, energy:",eSN(i),"code-energy-units"
288     print*, "SN (",i,"): at",col1,"years, mass",col3,"solar masses, energy:",col2,"
          FOE"
289   end do
290   close(1)
291   end subroutine read_sn
292 ! subroutine remove_driver
293 !> \short deallocates driver data vectors
294 !-----------------------------------------------------------------------
295 !> \version 1.2
296 !> \author Katharina M. Fierlinger
297 !> \date last modification 10.12.2009
298 !-----------------------------------------------------------------------
299 !> \details PURPOSE: deallocate driver data vectors
300 !-----------------------------------------------------------------------
301 subroutine remove_driver
302   implicit none
303   deallocate(timedriver)
304   deallocate(eidriver)
305   deallocate(dMdriver)
306   deallocate(veldriver)
307   deallocate(al26driver)
308   deallocate(fe60driver)
309 end subroutine remove_driver
310 ! subroutine remove_sn
311 !> \short deallocates driver sn data vectors
312 !-----------------------------------------------------------------------
313 !> \version 1.0
314 !> \author Katharina M. Fierlinger
315 !> \date last modification 10.08.2011
316 !-----------------------------------------------------------------------
317 !> \details PURPOSE: deallocate driver sn data vectors
318 !-----------------------------------------------------------------------
319 subroutine remove_sn
320   implicit none
```

```
321  deallocate(timeSN)
322  deallocate(eSN)
323  deallocate(mSN)
324 end subroutine remove_sn
325 ! subroutine interpolate_driver (age,edriver,rhodriver,aldriver,fedriver)
326 !> \short interpolates driver data
327 !> \param age ... age of the OB association (code units)
328 !> \param edriver ... energy output of the OB association (code units) at time "
    age"
329 !> \param rhodriver ... mass output of the OB association (code units) at time "
    age"
330 !> \param aldriver ... 26Al fraction of the mass output of the OB association at
    time "age"
331 !> \param fedriver ... 60Fe fraction of the mass output of the OB association at
    time "age"
332 !------------------------------------------------------------------------
333 !> \version 1.4
334 !> \author Katharina M. Fierlinger
335 !> \date last modification 04.02.2010
336 !------------------------------------------------------------------------
337 !> \details PURPOSE: linear interpolation of driver data
338 !------------------------------------------------------------------------
339 subroutine interpolate_driver (age,edriver,rhodriver,aldriver,fedriver)
340  implicit none
341  real(dp), intent(in)  :: age
342  real(dp), intent(out) :: edriver, rhodriver, aldriver, fedriver
343  integer(i9) :: i
344  if (age.lt.endtimedriver) then
345    i=1
346    do while (timedriver(i).lt.age)
347      i=i+1
348    end do
349    if (i<2) then
350     rhodriver=dMdriver(1)
351     edriver = eidriver(1)
352     aldriver=al26driver(1)
353     fedriver=fe60driver(1)
354    else
355     edriver = eidriver(i-1)+                                          &
356 &   (age-timedriver(i-1))/(timedriver(i)-timedriver(i-1))*            &
357 &   (eidriver(i)-eidriver(i-1))
358     rhodriver=dMdriver(i-1)+                                          &
359 &   (age-timedriver(i-1))/(timedriver(i)-timedriver(i-1))*            &
360 &   (dMdriver(i)-dMdriver(i-1))
361     aldriver=al26driver(i-1)+                                         &
362 &   (age-timedriver(i-1))/(timedriver(i)-timedriver(i-1))*            &
363 &   (al26driver(i)-al26driver(i-1))
364     fedriver=fe60driver(i-1)+                                         &
365 &   (age-timedriver(i-1))/(timedriver(i)-timedriver(i-1))*            &
366 &   (fe60driver(i)-fe60driver(i-1))
367    end if
368  else
369    print*, 'time: ', age
370    print*, "no driver data for time > ", endtimedriver
371   edriver  = 0.0_dp
```

```
372   rhodriver= 0.0_dp
373   aldriver = 0.0_dp
374   fedriver = 0.0_dp
375  end if
376 end subroutine interpolate_driver
377 !subroutine add_SN
378 !> \short checks is SN explosions occurred during this timestep
379 !------------------------------------------------------------------------------
380 !> \version 1.0
381 !> \author Katharina M. Fierlinger
382 !> \date last modification 10.08.2011
383 !------------------------------------------------------------------------------
384 !> \details PURPOSE: Add SN mass+energy loss if any SN exploded.
385 !------------------------------------------------------------------------------
386 subroutine add_SN(age, deltat, m_sn, e_sn)
387  implicit none
388  real(dp), intent(in)  :: age, deltat
389  real(dp), intent(out)  :: m_sn, e_sn
390  integer(i9) :: i
391 m_sn=0.0_dp
392 e_sn=0.0_dp
393 do i=1,SIZE(timeSN)
394  if ((timeSN(i).gt.age).and.(timeSN(i).le.age+deltat))then
395    m_sn=m_sn+mSN(i)
396    e_sn=e_sn+eSN(i)
397    print *, "SN", timeSN(i),mSN(i),eSN(i),m_sn,e_sn
398  end if
399 end do
400 !if (m_sn.le.0.0_dp) then
401 ! print*, "no SN"
402 !end if
403 end subroutine add_SN
404 !------------------------------------------------------------------------------
405 !> \short calculates Monte Carlo weights for a spherical driver region
406 !------------------------------------------------------------------------------
407 !> \version 1.0
408 !> \author Katharina M. Fierlinger
409 !> \date last modification 27.01.2012
410 !------------------------------------------------------------------------------
411 !> \details PURPOSE: Monte Carlo weights for a spherical driver region.
412 !> If a cell is partly inside the driver region, randgridsize random
413 !> positions in this cells are computed. The percentage of random points
414 !> that lie inside the driver region is asumed to be equal to the
415 !> percentage of cell volume that is inside the driver region.
416 !------------------------------------------------------------------------------
417 subroutine allocate_driver_mask
418   use amr_parameters, only:  r_driver,x_driver,y_driver,z_driver,      &
419 &  boxlen,levelmin,nlevelmax
420 #ifndef WITHOUTMPI
421   use amr_commons, only: myid
422 #endif
423   use random
424   implicit none
426
426   integer::ii,iix,iiy,iiz,ix,iy,iz !< loop variables
```

```
427   integer::halfsize !<half size of driver region
428   integer:: randgridsize = 100000 !< number of points in random subgrid
429   integer:: nn = 0 !< counter for Monte Carlo Volume
430   integer,dimension( IRandNumSize ) ::                                  &
431  &  localseed = (/ 3281, 4041, 595, 2376 /)
432   integer(i9) :: error_alloc !< checks if memory allocation works
433   real(dp),parameter :: pi = DACOS(-1.D0)
434   real(dp):: minx=0.0_dp,maxx=0.0_dp
435   real(dp):: help_low,r_driver_scaled,help1,help2,help3,dx
436   real(kind=8)::help_k8
437  #if NDIM>1
438   real(dp):: miny=0.0_dp,maxy=0.0_dp
439  #endif
440  #if NDIM>2
441   real(dp):: minz=0.0_dp,maxz=0.0_dp
442  #endif
444
444  !allocate array of pointers (1..nlevelmax-levelmin+1)
445   allocate(driver1(1:nlevelmax-levelmin+1),stat=error_alloc)
446   if (error_alloc /= 0) then
447     stop 'exiting: allocation of memory for driver data (driver1) did not work'
448   end if
450
450  !driver1 contains for each level
451  ! %weights(ndim x ndim array) allocatable
452  ! %volume
453  ! %dx
454   do ii=levelmin,nlevelmax
455    dx=boxlen*0.5_dp**(ii)        !< cell size [boxlen units]
456    driver1(ii-levelmin+1)%dx=dx !< cell size [boxlen units]
457  ! check how many grid cells are along the driver diagonal on each grid
458  ! level add 2 cells since the driver center may not be on a cell face
459    r_driver_scaled=r_driver/dx
460    halfsize=CEILING(r_driver_scaled)+1
461    driver1(ii-levelmin+1)%halfsize=halfsize
462  !allocate an array for the weights
463  !%weights points to a ndim array of 2*ceiling(r_driver/dx_level(1))
464    allocate(driver1(ii-levelmin+1)%weights(1:2*halfsize                &
465  #if NDIM>1
466  &                                        ,1:2*halfsize                &
467  #endif
468  #if NDIM>2
469  &                                        ,1:2*halfsize                &
470  #endif
471  & ),stat=error_alloc)
472   if (error_alloc /= 0) then
473     stop 'exiting: allocation of memory for driver data (driver1%weights) did not
          work'
474   end if
475  ! cell number = FLOOR((x-xc)/dx) + halfsize
476  !-------------------
477  !calculate the location of the driver center with respect to the grid
478  !-------------------
479    help_low=(x_driver/boxlen+0.5_dp)/(0.5_dp**(ii)) ! driver location in cells
480    minx=help_low-dble(FLOOR(help_low))!non integer value ... space between driver
```

```
      grid and next (lower) grid point
481   minx=(-minx-dble(halfsize))
482   maxx=minx
483 !  print*,"cells along radius:",r_driver_scaled
484 !-------------------
485 !get the weights
486 !-------------------
487   do iix=1,2*halfsize
488 #if NDIM>1
489    help_low=(y_driver/boxlen+0.5_dp)/(0.5_dp**(ii)) ! driver location in cells
490    miny=help_low-dble(FLOOR(help_low))!non integer value ... space between driver
         grid and next (lower) grid point
491    miny=(-miny-dble(halfsize))
492    maxy=miny
493    do iiy=1,2*halfsize
494 #endif
495 #if NDIM>2
496     help_low=(z_driver/boxlen+0.5_dp)/(0.5_dp**(ii)) ! driver location in cells
497     minz=help_low-dble(FLOOR(help_low))
498     minz=(-minz-dble(halfsize))
499     maxz=minz
500     do iiz=1,2*halfsize
501 #endif
502      if((maxx**2+maxy**2+maxz**2).lt.(r_driver_scaled)**2) then !fully inside
503 #if NDIM==1
504       driver1(ii-levelmin+1)%weights(iix) = 1.0_dp
505 #endif
506 #if NDIM==2
507       driver1(ii-levelmin+1)%weights(iix,iiy) = 1.0_dp
508 #endif
509 #if NDIM>2
510       driver1(ii-levelmin+1)%weights(iix,iiy,iiz) = 1.0_dp
511 #endif
512      else if (((abs(maxx)-1._dp)**2+(abs(maxy)-1._dp)**2+(abs(maxz)-1._dp)**2).gt
         .(r_driver_scaled)**2)then !fully outside
513 #if NDIM==1
514       driver1(ii-levelmin+1)%weights(iix) = 0.0_dp
515 #endif
516 #if NDIM==2
517       driver1(ii-levelmin+1)%weights(iix,iiy) = 0.0_dp
518 #endif
519 #if NDIM>2
520       driver1(ii-levelmin+1)%weights(iix,iiy,iiz) = 0.0_dp
521 #endif
522      else !partly inside
523       nn=0
524 #ifdef SMOOTH_DRIVER_EDGE
525       help1=0._dp
526       help2=0._dp
527       help3=0._dp
528       do ix=1,randgridsize
529         call ranf(localseed,help_k8)
530         help1=minx+dble(help_k8)
531 #if NDIM>1
532         call ranf(localseed,help_k8)
```

```
533           help2=miny+dble(help_k8)
534 #endif
535 #if NDIM>2
536           call ranf(localseed,help_k8)
537           help3=minz+dble(help_k8)
538 #endif
539           if((help1**2+help2**2+help3**2).lt.(r_driver_scaled)**2)then
540             nn=nn+1
541           end if
542         end do
543 #endif
544 #if NDIM==1
545         driver1(ii-levelmin+1)%weights(iix) = dble(nn)/dble(randgridsize)
546 #endif
547 #if NDIM==2
548         driver1(ii-levelmin+1)%weights(iix,iiy) = dble(nn)/dble(randgridsize)
549 #endif
550 #if NDIM>2
551         driver1(ii-levelmin+1)%weights(iix,iiy,iiz) = dble(nn)/dble(randgridsize)
552 #endif
553       end if
554 #if NDIM>2
555       minz=minz+1._dp
556       maxz=max(abs(minz),abs(minz+1.0))
557     end do
558 #endif
559 #if NDIM>1
560       miny=miny+1._dp
561       maxy=max(abs(miny),abs(miny+1.0))
562    end do
563 #endif
564    minx=minx+1._dp
565    maxx=max(abs(minx),abs(minx+1.0))
566  end do
567
568 !sum the weights to check if the weighting with a sphere is okay for each grid
       llevel
569 #if NDIM==1
570   driver1(ii-levelmin+1)%volume=sum(driver1(ii-levelmin+1)%weights(:))*dx
571 #endif
572 #if NDIM==2
573   driver1(ii-levelmin+1)%volume=sum(driver1(ii-levelmin+1)%weights(:,:))*dx*dx
574 #endif
575 #if NDIM>2
576   driver1(ii-levelmin+1)%volume=sum(driver1(ii-levelmin+1)%weights(:,:,:))*dx*dx*
       dx
577 #endif
578 #ifndef WITHOUTMPI
579   if(myid==1)then
580 #endif
581    print*,"level",ii,"size",2*halfsize,"driver␣radius␣[cells]",r_driver_scaled
582    print*,"Volume",driver1(ii-levelmin+1)%volume,"expected␣Volume", 4.*pi/3.*
       r_driver**3
583    print*,"Volume␣ratio",driver1(ii-levelmin+1)%volume/4./pi*3./r_driver**3
584 #ifndef WITHOUTMPI
```

```
585   end if
586 #endif
587  end do
588 end subroutine allocate_driver_mask
589 REAL(dp) function get_driver_volume(currentlevel)
590 !-------------------------------------------------------------------------
591 !> \details PURPOSE: look up Monte Carlo Volume that slightly differs
592 !> from 4 pi/3 r^3
593 !-------------------------------------------------------------------------
594   use amr_parameters, only: levelmin,nlevelmax
595   implicit none
596   integer, intent(in) :: currentlevel
597   get_driver_volume = driver1(currentlevel-levelmin+1)%volume
598 end function get_driver_volume
599 !function get_driver_mask
600 !-------------------------------------------------------------------------
601 !> \details PURPOSE: look up the volume fraction of ``feedback region''
602 !> in a cell with given coordinates
603 !-------------------------------------------------------------------------
604 #if NDIM==1
605 REAL(dp) function get_driver_mask(currentlevel,x)
606 #endif
607 #if NDIM==2
608 REAL(dp) function get_driver_mask(currentlevel,x,y)
609 #endif
610 #if NDIM==3
611 REAL(dp) function get_driver_mask(currentlevel,x,y,z)
612 #endif
613   use amr_parameters, only:  x_driver,y_driver,z_driver,                &
614 &  levelmin,nlevelmax
615   implicit none
616   integer, intent(in) :: currentlevel
617   real(dp),intent(in) :: x
618 #if NDIM>1
619   real(dp),intent(in) :: y
620 #endif
621 #if NDIM>2
622   real(dp),intent(in) :: z
623 #endif
624   integer :: ix=1
625   integer :: iy=1
626   integer :: iz=1
627   integer :: level_integer
628   level_integer=currentlevel-levelmin+1
629 ! lowest point: -minx-dble(halfsize) with minx e [0:1[ should get grid index 1
630 ! cell number = CEILING(([xyz]-[xyz]_driver)/dx) + halfsize
631   ix=CEILING((x-x_driver)/driver1(level_integer)%dx                     &
632 &    +dble(driver1(level_integer)%halfsize))
633 #if NDIM>1
634   iy=CEILING((y-y_driver)/driver1(level_integer)%dx                     &
635 &    +dble(driver1(level_integer)%halfsize))
636 #endif
637 #if NDIM>2
638   iz=CEILING((z-z_driver)/driver1(level_integer)%dx                     &
639 &    +dble(driver1(level_integer)%halfsize))
```

```
640 #endif
641   if((min(ix,iy,iz).lt.1)                                                &
642 &     .or.(max(ix,iy,iz).gt.2*driver1(level_integer)%halfsize))then
643    get_driver_mask = 0.0_dp
644   else
645 #if NDIM==1
646    get_driver_mask = driver1(level_integer)%weights(ix)
647 #endif
648 #if NDIM==2
649    get_driver_mask = driver1(level_integer)%weights(ix,iy)
650 #endif
651 #if NDIM>2
652    get_driver_mask = driver1(level_integer)%weights(ix,iy,iz)
653 #endif
654   end if
655 end function get_driver_mask
656 subroutine deallocate_driver_mask
657 !------------------------------------------------------------------------
658 !> \details PURPOSE: free memory allocated for driver1
659 !------------------------------------------------------------------------
660   use amr_parameters, only:  levelmin,nlevelmax
661 !  integer, intent(in) :: levelmin,nlevelmax
662   integer::ii !< loop variable
664
664  do ii=levelmin,nlevelmax
665   deallocate(driver1(ii-levelmin+1)%weights)
666  end do
668
668  deallocate(driver1)
670
670 end subroutine deallocate_driver_mask
671 subroutine driver_weights_fixed(ind,ilevel,igrid,ngrid,dx,driverweight)
672   use amr_commons, only : active, xg  !< index array, coordinates (values in
        interval [0.5,2.5]
673   use amr_parameters, only : boxlen, dp, icoarse_max, icoarse_min, jcoarse_min,
        kcoarse_min !< floating point type, lower [xyz] coarse grid boundaries
674   implicit none
675   integer, intent(in) :: ind    !< position of new grids
676   integer, intent(in) :: ilevel !< AMR level
677   integer, intent(in) :: igrid  !< grid index
678   integer, intent(in) :: ngrid  !< grid size
679   real(dp),intent(in) :: dx !< cell size
680   real(dp), dimension(1:ngrid),intent(out) :: driverweight !< fraction of the cell
       volume that is inside the driver area
682
682   integer:: i,ix,iy,iz,ind_grid !< loop variable, position in coordinate array
683   real(dp), dimension(1:3) :: skip_loc !< grid boundaries
684   real(dp), dimension(1:3) :: xc !< center of new grid
685   real(dp):: x,y,z , boxscale!< lower boundary of grid cell coordinates
687
687   driverweight(:)=0.0_dp
689
689   !ind=1,2**ndim
690   !2d: ind=1,4
691   !3d: ind=1,8
```

```
692     ! Set new grids position
693     iz=(ind-1)/4          ! integer division -> 0 or 1
694     iy=(ind-1-4*iz)/2     ! integer division -> 0 or 1
695     ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
696     skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
697     xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5D0 or +0.5D0
698     skip_loc(1)=dble(icoarse_min)
699 #if NDIM>1
700     xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5D0 or +0.5D0
701     skip_loc(2)=dble(jcoarse_min)
702 #endif
703 #if NDIM>2
704     xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5D0 or +0.5D0
705     skip_loc(3)=dble(kcoarse_min)
706 #endif
707     boxscale=boxlen/dble(icoarse_max-icoarse_min+1)
708     !xg(:,1)-1.5 ... values in interval [-1,1]
709     do i=1,ngrid
710       ind_grid=active(ilevel)%igrid(igrid+i-1)
711       !xg(ind_grid(i),1) .. x coordinate of the center of the subgrid
712       !x ... lower boundary
713       x=(xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp+0.5_dp*dx)*boxscale
714 #if NDIM==1
715       driverweight(i)=get_driver_mask(ilevel,x)
716 #else
717       y=(xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp+0.5_dp*dx)*boxscale
718 #if NDIM==2
719       driverweight(i)=get_driver_mask(ilevel,x,y)
720 #else
721       z=(xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp+0.5_dp*dx)*boxscale
722       driverweight(i)=get_driver_mask(ilevel,x,y,z)
723 #endif
724 #endif
725     end do
726 end subroutine driver_weights_fixed
727 ! subroutine driver_weights(ind,ilevel,igrid,ngrid,dx,rdriver_scaled, xdriver,
      ydriver,zdriver,driverweight)
728 !> \short calculates Monte Carlo weights for a spherical driver region
729 !-------------------------------------------------------------------------
730 !> \version 1.6
731 !> \author Katharina M. Fierlinger
732 !> \date last modification 07.06.2010
733 !-------------------------------------------------------------------------
734 !> \details PURPOSE: Monte Carlo weights for a spherical driver region.
735 !> This routine is better suited for moving feedback regions than
736 !> driver_weights_fixed if they do not move by integer numbers of grid cells.
737 !> If a cell is partly inside the driver region, randgridsize random
738 !> positions in this cells are computed. The percentage of random points
739 !> that lie inside the driver region is asumed to be equal to the percentage
740 !> of cell volume that is inside the driver region.
741 !-------------------------------------------------------------------------
742 subroutine driver_weights(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,    &
743 &                         xdriver,                                     &
744 #if NDIM>1
745 &                         ydriver,                                     &
```

```
746 #endif
747 #if NDIM>2
748 &                         zdriver,                                    &
749 #endif
750 &                         driverweight)
752
752   use amr_commons, only : active, xg  !< index array, coordinates (values in
          interval [0.5,2.5]
753   use amr_parameters, only : dp, icoarse_min, jcoarse_min, kcoarse_min !< floating
          point type, lower [xyz] coarse grid boundaries
754   use random
755   implicit none
756   integer, intent(in) :: ind     !< position of new grids
757   integer, intent(in) :: ilevel  !< AMR level
758   integer, intent(in) :: igrid   !< grid index
759   integer, intent(in) :: ngrid   !< grid size
760   real(dp),intent(in) :: dx      !< cell size
761   real(dp),intent(in) :: rdriver_scaled !< driver radius in coarse grid cells
762   real(dp),intent(in) :: xdriver !< driver [xyz] coordinate
763 #if NDIM>1
764   real(dp),intent(in) :: ydriver !< driver [xyz] coordinate
765 #endif
766 #if NDIM>2
767   real(dp),intent(in) :: zdriver !< driver [xyz] coordinate
768 #endif
769   real(dp), dimension(1:ngrid),intent(out) :: driverweight !< fraction of the cell
          volume that is inside the driver area
771
771   integer:: i,ind_grid,ix,iy,iz,nn !< loop variable, position in coordinate array,
          new grid [xyz] index, random numbers inside driver
772   integer:: randgridsize = 100     !< number of points in random subgrid
773   real(dp):: r2                    !< squared driver radius (in coarse grid cells)
774   real(dp):: xmin,ymin,zmin,xmax,ymax,zmax !< boundaries of new grid cells
775   real(kind=8) :: help_k8              !< random coordinates [0:1]
776   real(dp):: help1, help2, help3       !< random coordinates [0:1]
777   real(dp), dimension(1:3) :: skip_loc !< grid boundaries
778   real(dp), dimension(1:3) :: xc       !< center of new grid
779   integer,dimension( IRandNumSize ) ::                                    &
780        & localseed = (/ 3281, 4041, 595, 2376 /)
782
782   driverweight(:)=0.0_dp
783   r2=rdriver_scaled**2
785
785   !ind=1,2**ndim
786   !2d: ind=1,4
787   !3d: ind=1,8
788   ! Set new grids position
789   iz=(ind-1)/4         ! integer division -> 0 or 1
790   iy=(ind-1-4*iz)/2    ! integer division -> 0 or 1
791   ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
792   skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
793   xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5D0 or +0.5D0
794   skip_loc(1)=dble(icoarse_min)
795 #if NDIM>1
796   xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5D0 or +0.5D0
```

```
797     skip_loc(2)=dble(jcoarse_min)
798   #endif
799   #if NDIM>2
800     xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5D0 or +0.5D0
801     skip_loc(3)=dble(kcoarse_min)
802   #endif
803
804     !xg(:,1)-1.5 ... values in interval [-1,1]
805     do i=1,ngrid
806       ind_grid=active(ilevel)%igrid(igrid+i-1)
807       !xg(ind_grid(i),1) .. x coordinate of the center of the subgrid
808       xmax=abs(xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp-xdriver)+0.5_dp*dx ! minimum:
              0.5_dp*dx
809       xmin=xmax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(xmin)<abs(xmax)
810   #if NDIM>1
811       ymax=abs(xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp-ydriver)+0.5_dp*dx ! minimum:
              0.5_dp*dx
812       ymin=ymax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(ymin)<abs(ymax)
813   #else
814       ymax=0.0_dp
815       ymin=0.0_dp
816   #endif
817   #if NDIM>2
818       zmax=abs(xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp-zdriver)+0.5_dp*dx ! minimum:
              0.5_dp*dx
819       zmin=zmax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(zmin)<abs(zmax)
820   #else
821       zmax=0.0_dp
822       zmin=0.0_dp
823   #endif
824       if((xmin**2+ymin**2+zmin**2).lt.r2)then
825       !part of cell inside driver region
826         if((xmax**2+ymax**2+zmax**2).lt.r2)then
827           !cell fully inside driver region
828           driverweight(i)=1.0_dp
829   #ifdef SMOOTH_DRIVER_EDGE
830         else
831           nn=0
832           help1=0._dp
833           help2=0._dp
834           help3=0._dp
835           do ix=1,randgridsize
836             call ranf(localseed,help_k8)
837             help1=xmin+dble(help_k8)*dx
838   #if NDIM>1
839             call ranf(localseed,help_k8)
840             help2=ymin+dble(help_k8)*dx
841   #endif
842   #if NDIM>2
843             call ranf(localseed,help_k8)
844             help3=zmin+dble(help_k8)*dx
845   #endif
846             if((help1**2+help2**2+help3**2).lt.r2)then
847               nn=nn+1
848             end if
```

```
849           end do
850           driverweight(i) = dble(nn)/dble(randgridsize)
851 #endif
852         end if
853       else
854         driverweight(i)=0.0_dp
855       end if
856     end do
857 end subroutine driver_weights
858 ! subroutine driver_vector(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,   &
859 !&                         xdriver,ydriver,zdriver,                   &
860 !&                         driverweightvx,driverweightvy,driverweightvz)
861 !> \short calculates a radial vector for a spherical driver region
862 !-------------------------------------------------------------------------
863 !> \version 1.1
864 !> based on driver_vector version 1.6
865 !> \author Katharina M. Fierlinger
866 !> \date last modification 20.01.2011
867 !-------------------------------------------------------------------------
868 !> \details PURPOSE: For a spherical driver region radial, normalized
869 !> x and y, z velocity vectors are computed if a cell is partly inside
870 !> the driver region. Nothing to be done in 1d.
871 !-------------------------------------------------------------------------
872 subroutine driver_vector(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,     &
873 &                        xdriver                                       &
874 #if NDIM>1
875 &                        ,ydriver                                      &
876 #endif
877 #if NDIM>2
878 &                        ,zdriver                                      &
879 #endif
880 #if NDIM>1
881 &                        ,drivervectorx,drivervectory                  &
882 #endif
883 #if NDIM>2
884 &                        ,drivervectorz                                &
885 #endif
886 &                        )
888
888   use amr_commons, only : active, xg  !< index array, coordinates (values in
        interval [0.5,2.5]
889   use amr_parameters, only : dp, icoarse_min, jcoarse_min, kcoarse_min !< floating
        point type, lower [xyz] coarse grid boundaries
890   use random
891   implicit none
892   integer, intent(in) :: ind            !< position of new grids
893   integer, intent(in) :: ilevel         !< AMR level
894   integer, intent(in) :: igrid          !< grid index
895   integer, intent(in) :: ngrid          !< grid size
896   real(dp),intent(in) :: dx             !< cell size
897   real(dp),intent(in) :: rdriver_scaled !< driver radius in coarse grid cells
898   real(dp),intent(in) :: xdriver        !< driver [xyz] coordinate
899 #if NDIM>1
900   real(dp),intent(in) :: ydriver        !< driver [xyz] coordinate
901   real(dp), dimension(1:ngrid),intent(out) :: drivervectorx, drivervectory !<
```

```
      radial vector
902 #endif
903 #if NDIM>2
904   real(dp),intent(in) :: zdriver        !< driver [xyz] coordinate
905   real(dp), dimension(1:ngrid),intent(out) :: drivervectorz                !<
      radial vector
906 #endif
907   integer:: i,ind_grid,ix,iy,iz,nn !< loop variable, position in coordinate array,
       new grid [xyz] index, random numbers inside driver
908   real(dp):: r2           !< squared driver radius (in coarse grid cells)
909   real(dp):: xmin         !< boundaries of new grid cells
910 #if NDIM>1
911   real(dp):: xx,yy,rr     !< auxiliary variables (radial vector)
912   real(dp):: ymin         !< boundaries of new grid cells
913 #endif
914 #if NDIM>2
915   real(dp):: zz           !< auxiliary variables (radial vector)
916   real(dp):: zmin         !< boundaries of new grid cells
917 #endif
918   real(dp), dimension(1:3) :: skip_loc !< grid boundaries
919   real(dp), dimension(1:3) :: xc !< center of new grid
921
921 #if NDIM>1
922   drivervectorx(:)=0.0_dp
923   drivervectory(:)=0.0_dp
924 #if NDIM>2
925   drivervectorz(:)=0.0_dp
926 #endif
927   r2=rdriver_scaled**2
929
929   !ind=1,2**ndim
930   !2d: ind=1,4
931   !3d: ind=1,8
932   ! Set new grids position
933   iz=(ind-1)/4          ! integer division -> 0 or 1
934   iy=(ind-1-4*iz)/2     ! integer division -> 0 or 1
935   ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
936   skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
937   xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
938   skip_loc(1)=dble(icoarse_min)
939   xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
940   skip_loc(2)=dble(jcoarse_min)
941 #if NDIM>2
942   xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
943   skip_loc(3)=dble(kcoarse_min)
944 #endif
946
946   !xg(:,1)-1.5 ... values in interval [-1,1]
947   do i=1,ngrid
948     ind_grid=active(ilevel)%igrid(igrid+i-1)
949     !xg(ind_grid(i),1) .. x coordinate of the center of the subgrid
950     xmin=abs(xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp-xdriver)-0.5_dp*dx ! minimum:
          0.5_dp*dx
951     ymin=abs(xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp-ydriver)-0.5_dp*dx ! minimum:
          0.5_dp*dx
```

```
952 #if NDIM>2
953     zmin=abs(xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp-zdriver)-0.5_dp*dx ! minimum:
        0.5_dp*dx
954 #endif
955     if((xmin**2 +ymin**2                                                &
956 #if NDIM>2
957 &       +zmin**2                                                        &
958 #endif
959 &               ).lt.r2)then
960     !part of cell inside driver region
961       xx=xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp-xdriver
962       yy=xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp-ydriver
963 #if NDIM>2
964       zz=xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp-zdriver
965 #endif
966 #if NDIM==2
967       rr=sqrt(xx*xx+yy*yy)
968 #endif
969 #if NDIM==3
970       rr=sqrt(xx*xx+yy*yy+zz*zz)
971 #endif
972       drivervectorx(i)=xx/rr
973       drivervectory(i)=yy/rr
974 #if NDIM>2
975       drivervectorz(i)=zz/rr
976 #endif
977     end if
978   end do
979 #endif
980  rr=1
981 end subroutine driver_vector
982 !subroutine print_xyz(ind,ilevel,igrid,ngrid,dx,i)
983 !> \short output of the xyz coordinates of a given cell
984 !-------------------------------------------------------------------------
985 !> \version 1.0
986 !> \author Katharina M. Fierlinger
987 !> \date last modification 14.03.2011
988 !-------------------------------------------------------------------------
989 !> \details PURPOSE:
990 !> \n for debugging ...
991 !> \n helps to find out in which cell the code encounters a problem.
992 !-------------------------------------------------------------------------
993 subroutine print_xyz(ind,ilevel,igrid,ngrid,dx,i)
995
995   use amr_commons, only : active, xg  !< index array, coordinates (values in
        interval [0.5,2.5]
996   use amr_parameters, only : dp, icoarse_min, jcoarse_min, kcoarse_min !< floating
        point type, lower [xyz] coarse grid boundaries
997   use random
998   implicit none
999   integer, intent(in) :: ind    !< position of new grids
1000  integer, intent(in) :: ilevel !< AMR level
1001  integer, intent(in) :: igrid  !< grid index
1002  integer, intent(in) :: ngrid  !< grid size
1003  real(dp),intent(in) :: dx     !< cell size
```

```
1004   integer, intent(in) :: i       !< 1..ngrid
1006
1006   integer:: ind_grid,ix,iy,iz,nn !< loop variable, position in coordinate array,
           new grid [xyz] index, random numbers inside driver
1007   real(dp):: xmin,ymin,zmin,xmax,ymax,zmax !< boundaries of new grid cells
1008   real(dp), dimension(1:3) :: skip_loc     !< grid boundaries
1009   real(dp), dimension(1:3) :: xc           !< center of new grid
1012
1012
1012   !ind=1,2**ndim
1013   !2d: ind=1,4
1014   !3d: ind=1,8
1015   ! Set new grids position
1016   iz=(ind-1)/4         ! integer division -> 0 or 1
1017   iy=(ind-1-4*iz)/2    ! integer division -> 0 or 1
1018   ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
1019   skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
1020   xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1021   skip_loc(1)=dble(icoarse_min)
1022 #if NDIM>1
1023   xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1024   skip_loc(2)=dble(jcoarse_min)
1025 #endif
1026 #if NDIM>2
1027   xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1028   skip_loc(3)=dble(kcoarse_min)
1029 #endif
1031
1031   !xg(:,1)-1.5 ... values in interval [-1,1]
1032     ind_grid=active(ilevel)%igrid(igrid+i-1)
1033     !xg(ind_grid(i),1) .. x coordinate of the center of the subgrid
1034     xmax=(xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp)+0.5_dp*dx ! minimum: 0.5_dp*dx
1035     xmin=xmax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(xmin)<abs(xmax)
1036 #if NDIM>1
1037       ymax=(xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp)+0.5_dp*dx ! minimum: 0.5_dp*
              dx
1038       ymin=ymax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(ymin)<abs(ymax)
1039 #else
1040       ymax=0.0_dp
1041       ymin=0.0_dp
1042 #endif
1043 #if NDIM>2
1044       zmax=(xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp)+0.5_dp*dx ! minimum: 0.5_dp*
              dx
1045       zmin=zmax-dx ! can get < 0, minimum: -0.5_dp*dx, but abs(zmin)<abs(zmax)
1046 #else
1047       zmax=0.0_dp
1048       zmin=0.0_dp
1049 #endif
1050     print*,"dx=",dx
1051     print*,"xmin=",xmin,xmin/dx
1052     print*,"xmax=",xmax,xmax/dx
1053 #if NDIM>1
1054     print*,"ymin=",ymin,ymin/dx
1055     print*,"ymax=",ymax,ymax/dx
```

```
1056 #endif
1057 #if NDIM>2
1058      print*,"zmin=",zmin
1059      print*,"zmax=",zmax
1060 #endif
1061 end subroutine print_xyz
1062 !-------------------------------------------------------------------------
1063 ! subroutine driver_weights_analyt(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,
     xdriver,ydriver,zdriver,driverweight)
1064 !> \short calculates weights for a spherical driver region
1065 !-------------------------------------------------------------------------
1066 !> \version 1.5
1067 !> \author Katharina M. Fierlinger
1068 !> \date last modification 04.06.2010
1069 !-------------------------------------------------------------------------
1070 !> \details PURPOSE:
1071 !> \n  2d:
1072 !> \n   produce weights for a cylindrical driver region and store them in
1073 !>   (allocated) array driver_geom.
1074 !>   these weights are for pseudo-2d simulations with nz=1
1075 !>   use 1st quadrant's weights - number representation causes 4-symmetry
1076 !>  \n  3d:
1077 !>  \n  Monte Carlo weights for a spherical driver regionMonte Carlo weights for a
      spherical driver region
1078 !-------------------------------------------------------------------------
1079 !>\latexonly
1080 !> \section{Driver region}
1081 !> The size and location of the driver region are set in the namelist:
1082 !> \begin{verbatim}
1083 !> &DRIVER_PARAMS
1084 !> file_driver='analyt.dat' ! driver file name (relative to working directory)
1085 !> r_driver=0.83_dp          ! driver radius in code units
1086 !> x_driver=(-5.64453125_dp)  ! driver x coordinate in code units
1087 !> y_driver=0.0_dp           ! driver y coordinate in code units
1088 !> z_driver=0.0_dp           ! driver z coordinate in code units
1089 !> coolplus=0.5_dp           ! the driver is not cooled, coolplus is the
     difference
1090 !> ! between the driver radius and the radius of the not cooled region in code
     units
1091 !> n_stars=20._dp            ! the data in "file_driver" contains yields per star.
1092 !> !n_stars is the number of stars in the driver
1093 !> /
1094 !> \end{verbatim}
1095 !> To include this additional namelist in the code the files {\tt read\_params.f90
     } and {\tt amr\_parameters.f90} have tp be patched too. After {\tt ramses.f90}
      has read in the parameters, they can be accessed via \\\
1096 !> {\tt use amr\_parameters, only: r\_driver,x\_driver,y\_driver,z\_driver,n\
     _stars,file\_driver,coolplus}
1097 !>
1098 !> A driver module {\tt driver.f90} provides arrays to store the data from the
     driver file, reads and interpolates driver data and calculates weights for a
     homogeneous, circular driver region. It uses {\tt units.f90} to convert from
     driver file units to code units. The expected units in the driver file are:
1099 !>
1100 !> \begin{verbatim}
```

```
1101 !> column 1: time from starformation (in years)
1102 !> column 2: cumulative output of 26Al (in Msol)
1103 !> column 3: cumulative output of 60Fe (in Msol)
1104 !> column 4: UV radiation (photons/s)
1105 !> column 5: energy emitted in winds (log(erg/s))
1106 !> column 6: energy emitted in supernovae (log(erg/s))
1107 !> column 7: mass ejected by supernova (Msol/year)
1108 !> column 8: mass ejected in winds (Msol/year)
1109 !> \end{verbatim}
1110 !>
1111 !> The file containing the driver data is read in the subroutine {\tt init\_time.
     f90} which is called (once) by the subroutine {\tt adaptive\_loop.f90}.
1112 !>
1113 !> The file {\tt courant\_fine.f90} was patched to include mass and energy
     injection of the driver.
1114 !>
1115 !> Before starting the loop over active grids by vector sweeps, the stellar winds
     and SN yields are insertef using the new subroutine{\tt wind\_fine} in the
     same file. This new subroutine loops over all cells of the given gridlevel and
      checks if a part of the cell is inside the driver region.
1116 !>
1117 !> If this is the case, the code will add the newly emitted mass (total mass and
     radioactive tracers (nvar in {\tt hydro\_parameters.f90} is changed to get
     larger {\tt uold} and {\tt unew} arrays and thus {\tt output\_hydro.f90} had
     to be adapted)) and the internal energy (unresolved kinetic wind energy,
     radiation pressure) to the density resp. energy in the driver region.
1118 !> %
1119 !> The driver energy and mass are homogeneously distributed over a sphere of given
      radius ({\tt r\_driver}). Then the corresponding energy density and number
     density are computed. For each cell in the computational box, this value is
     scaled with the percentage of the cell volume that is inside the driver region
      (e.g. weight = 0.0 : cell lies fully outside,  weight = 1.0 : cell fully
     inside).
1120 !> %
1121 !> In 2d the percentage of the cell volume that is inside the driver region can be
      calculated analytically. To set the integration limits, the driver routine
     checks how many of the corners of the cell are inside the driver region. The
     routine uses the absolute values of the $x$, $y$ and $z$ distances of the cell
      corners to reduce the number of different cases.\\\
1122 !> \begin{tikzpicture}[scale=2.0]
1123 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50] (-0.5,-0.5) -- (
     0.5,-0.5) -- (0.5,0.5) -- (-0.5,0.5) -- cycle ;
1124 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50] (1,-0.5) -- (2,-0.5)
      -- (2,0.0) arc (30:48:2cm) -- (1.5,0.5) -- (1,0.5) -- cycle ;
1125 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50]  (2.5,0) -- (3.5,0)
      -- (3.5,0.2) arc (70:89.5:3cm) -- cycle ;
1126 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50]  ( 2.5,-1.1) -- (
     3.3,-1.1) arc (15:36:3cm) -- (2.8,-0.1) -- (2.5,-0.1) -- cycle ;
1127 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50] ( 4.0,-0.5) -- (
     4.5,-0.5) arc (46:55:4cm) -- (4.0,-0.1) -- cycle ;
1128 !>  \draw[-] (-0.5,-0.5) -- (-0.5,0.5) -- (0.5,0.5) -- (0.5,-0.5) -- (0.0,-0.5)
     node[below] {all corners} -- cycle ;
1129 !>  \draw[-] ( 1.0,-0.5) -- ( 1.0,0.5) -- (2.0,0.5) -- (2.0,-0.5) -- (1.5,-0.5)
     node[below] {3 corners} -- cycle ;
1130 !>  \draw[-] ( 2.5,-0) -- ( 2.5,1) -- (3.5,1) -- (3.5,0) -- cycle ;
```

```
1131 !>  \draw[-] ( 2.5,-1.1) -- ( 2.5,-0.1) -- (3.5,-0.1) -- (3.5,-1.1) -- (3.0,-1.1)
         node[below] {2 corners} -- cycle ;
1132 !>  \draw[-] ( 4.0,-0.5) -- ( 4.0,0.5) -- (5.0,0.5) -- (5.0,-0.5) -- (4.5,-0.5)
         node[below] {1 corner} -- cycle ;
1133 !>  \draw[-] ( 5.5,-0.5) -- ( 5.5,0.5) -- (6.5,0.5) -- (6.5,-0.5) -- (6.0,-0.5)
         node[below] {no corner} -- cycle ;
1134 !> \end{tikzpicture}\\\
1135 !> The cases ``no corner'' and ``all corners'' are trivial (0\% or 100\% inside)
         .\\\
1136 !> The code looks up the $x$ and $y$ coordinates of the cell center with respect
         to the driver center. Negative coordinates are changed to positive ones. If
         the driver's $x$ or $y$ axis lies inside the cell (one coordinate is smaller
         than half a grid cell length) the integrals should use the area between the
         curve and the other axis. If both axis lie in a cell that is not fully inside
         the driver, the code stops and asks for a larger driver radius - the driver
         should use more than 4 cells anyway.
1137 !>
1138 !> % and the other three cases will be discussed in the following subsubsections.
1139 !> \parbox{120mm}{In the 2d case with only the corner $\left(x_{\rm min}|y_{\rm
         min} \right)$ inside the driver region, the fraction the cell volume inside
         the driver region ($p_{\rm driver}$) can be calculated with:
1140 !> \begin{eqnarray}
1141 !> p_{\rm driver}&=&\frac{\int_{x_{\rm min}}^{x_{1}}{\rm d}x\int_{y_{\rm min}}^{\
         sqrt{r^2-x^2}}{\rm d}y}{V_{\rm cell}} \nonumber \\\
1142 !> &=&\frac{\int_{x_{\rm min}}^{x_{1}} \sqrt{r^2-x^2} {\rm d}x - y_{\rm min}\left(
         x_{1}-x_{\rm min}\right) }{(\Delta x)^2} \nonumber \\\
1143 !> &=&\frac{\frac{1}{2}\left(x\sqrt{r^2-x^2}+r^2 \arcsin \frac{x}{r} \right)_{x_{\
         rm min}}^{x_{1}} - y_{\rm min}\left(x_{1}-x_{\rm min}\right) }{(\Delta x)^2}\
         nonumber \\\
1144 !> &=&\frac{x_{\rm min}y_{\rm min}- \frac{x_{\rm min}y_{1}}{2}-\frac{x_{1}y_{\rm
         min}}{2}
1145 !> +\frac{r^2}{2}\left( \arcsin \frac{x_{1}}{r} - \arcsin \frac{x_{\rm min}}{r} \
         right)
1146 !> }{(\Delta x)^2}\nonumber \label{2d:1corner}
1147 !> \end{eqnarray} }
1148 !> \parbox[t]{40mm}{
1149 !> \begin{tikzpicture}[scale=2.0]
1150 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50] ( 0.0,-0.5) -- (
         0.5,-0.5) arc (46:55:4cm) -- (0.0,-0.1) -- cycle ;
1151 !> \draw[-] ( 0.0,-0.5) -- ( 0.0,0.1) -- (0.6,0.1) -- (0.6,-0.5) -- cycle ;
1152 !> \filldraw (0.0,-0.5) circle (0.3mm) node[left] {$\left(x_{\rm min}|y_{\rm min}
          \right)$};
1153 !> \filldraw (0.0,-0.1) circle (0.3mm) node[left] {$\left(x_{\rm min}|y_{1}
         \right)$};
1154 !> \filldraw (0.5,-0.5) circle (0.3mm) node[right] {$\,\,\left(x_{1}      |y_{\rm
         min}\right)$};
1155 !> \end{tikzpicture}}
1156 !> %\subsubsection{2 corners}
1157 !> \parbox{120mm}{If there are two corners of the 2d cell inside the driver region
         , these corners are $\left(x_{\rm min}|y_{\rm min} \right)$ and $\left(x_{\rm
          max}|y_{\rm min} \right)$ or $\left(x_{\rm min}|y_{\rm max} \right)$. In the
         case $x_{\rm min}>y_{\rm min}$ the x and y coordinates are swapped to get an
         $x$-integral. The fraction the cell volume inside the driver region ($p_{\rm
         driver}$) can be calculated with:
1158 !> \begin{eqnarray}
```

```
1159 !> p_{\rm driver}&=&\frac{\int_{x_{\rm min}}^{x_{\rm max}}{\rm d}x\int_{y_{\rm min
     }}^{\sqrt{r^2-x^2}}{\rm d}y}{V_{\rm cell}} \nonumber \\\
1160 !> &=&\frac{\int_{x_{\rm min}}^{x_{\rm max}} \sqrt{r^2-x^2} {\rm d}x - y_{\rm min}
      \Delta x }{(\Delta x)^2} \nonumber \\\
1161 !> &=&\frac{\frac{1}{2}\left(x\sqrt{r^2-x^2}+r^2 \arcsin \frac{x}{r} \right)_{x_{\
     rm min}}^{x_{\rm max}} - y_{\rm min}\Delta x }{(\Delta x)^2}\nonumber \\\
1162 !> &=&\frac{\frac{x_{\rm max}y_{2}}{2}-\frac{x_{\rm min}y_{1}}{2}
1163 !> +\frac{r^2}{2}\left( \arcsin \frac{x_{\rm max}}{r} -  \arcsin \frac{x_{\rm min
     }}{r} \right) - y_{\rm min}\Delta x
1164 !> }{(\Delta x)^2}\nonumber \label{2d:2corners}
1165 !> \end{eqnarray} }
1166 !> \parbox[t]{40mm}{
1167 !> \begin{tikzpicture}[scale=2.0]
1168 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50](0,0.2) -- (1,0.2)
     arc (70:89.5:3cm) -- cycle ;
1169 !>  \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.25](0,0) -- ( 1,0) --
     (1,0.2) -- (0,0.2) -- cycle ;
1170 !> \draw[-] (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle ;
1171 !> \draw[-] (0,0.2) -- (1,0.2);
1172 !> \filldraw (0,0)   circle (0.3mm) node[left]  {$\left(x_{\rm min}|y_{\rm min} \
     right)$};
1173 !> \filldraw (0,0.4) circle (0.3mm) node[left]  {$\left(x_{\rm min}|y_{1}      \
     right)$};
1174 !> \filldraw (1,0.2) circle (0.3mm) node[right] {$\left(x_{\rm max}|y_{2}      \
     right)$};
1175 !> \end{tikzpicture}}
1176 !> %\subsubsection{3 corners}
1177 !> \parbox{140mm}{If only the corner $\left(x_{\rm max}|y_{\rm max} \right)$ lies
     outside the driver region, $p_{\rm driver}$ can be calculated with:
1178 !> \begin{eqnarray}
1179 !> p_{\rm driver}&=&\frac{\int_{x_{1}}^{x_{\rm max}}{\rm d}x\int_{y_{\rm min}}^{\
     sqrt{r^2-x^2}}{\rm d}y+(x_{1}-x_{\rm min})\Delta x}{V_{\rm cell}} \nonumber
     \\\
1180 !> &=&\frac{\int_{x_{1}}^{x_{\rm max}} \sqrt{r^2-x^2} {\rm d}x - y_{\rm min}\left(
     x_{\rm max}-x_{1}\right) +(x_{1}-x_{\rm min})\Delta x}{(\Delta x)^2} \nonumber
      \\\
1181 !> &=&\frac{\frac{1}{2}\left(x\sqrt{r^2-x^2}+r^2 \arcsin \frac{x}{r} \right)_{x_
     {1}}^{x_{\rm max}} - y_{\rm min}\left(x_{\rm max}-x_{1}\right)  +(x_{1}-x_{\rm
      min})\Delta x}{(\Delta x)^2}\nonumber \\\
1182 !> &=&\frac{\frac{x_{\rm max}y_{1}}{2}-\frac{x_{1}y_{\rm max}}{2}
1183 !> +\frac{r^2}{2}\left( \arcsin \frac{x_{\rm max}}{r} -  \arcsin \frac{x_{1}}{r} \
     right) - y_{\rm min}\left(x_{\rm max}-x_{1}\right)+(x_{1}-x_{\rm min})\Delta x
1184 !> }{(\Delta x)^2}\nonumber \label{2d:3corners}
1185 !> \end{eqnarray}}
1186 !> \parbox[t]{20mm}{
1187 !> \begin{tikzpicture}[scale=2.0]
1188 !> \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.75] (0,0) -- (0.6,0) --
     (0.6,1) -- (0,1) -- cycle ;
1189 !> \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.50] (0.6,0.5) -- (1,0.5)
     arc (30:48:2cm) -- (0.6,1) --  cycle ;
1190 !> \filldraw[color=blue!50,thin,fill=blue,fill opacity=0.25] (0.6,0) -- (1,0) --
     (1,0.5) -- (0.6,0.5) -- cycle ;
1191 !> \draw[-] (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle ;
1192 !> \filldraw (0.6,1) circle (0.3mm) node[above] {$\left(x_{1}|y_{\rm max} \right)$
     };
```

```
1193 !> \filldraw (1,0.5) circle (0.3mm) node[right] {$\left(x_{\rm max}|y_{1} \right)$
         };
1194 !> \filldraw (0.6,0) circle (0.3mm) node[below] {$\left(x_{1}|y_{\rm min} \right)$
         };
1195 !> \end{tikzpicture}\\\
1196 !> In 3d or in the subroutine {\tt driver\_weights} the percentage of the cell
         inside the driver area is calculated with Monte Carlo if it is not a trivial
         case (0\% or 100\%). For all three directions $n$ random variables are
         calculated. The fraction the cell volume inside the driver region $p_{\rm
         driver}$ is the number of random points inside the driver region ($|(x_i|y_i|
         z_i)|<r$) divided by the total number of random points $n$.
1197 !>\endlatexonly
1198 subroutine driver_weights_analyt(ind,      & !< position of new grids
1199 &                           ilevel,       & !< AMR level
1200 &                           igrid,        & !< grid index
1201 &                           ngrid,        & !< grid size
1202 &                           dx,           & !< cell size
1203 &                           rdriver_scaled, & !< driver radius in coarse grid cells
1204 &                           xdriver,      & !< driver [xyz] coordinate
1205 &                           ydriver,      & !< driver [xyz] coordinate
1206 &                           zdriver,      & !< driver [xyz] coordinate
1207 &                           driverweight)   !< fraction of the cell volume that is
         inside the driver area
1208   use amr_commons, only : active, xg
1209   use amr_parameters, only : dp, icoarse_min, jcoarse_min, kcoarse_min, &
1210 & verbose_patches
1211   use random
1212   implicit none
1213   integer, intent(in) :: ind            !< position of new grids
1214   integer, intent(in) :: ilevel         !< AMR level
1215   integer, intent(in) :: igrid          !< grid index
1216   integer, intent(in) :: ngrid          !< grid size
1217   real(dp),intent(in) :: dx             !< cell size
1218   real(dp),intent(in) :: rdriver_scaled !< driver radius in coarse grid cells
1219   real(dp),intent(in) :: xdriver        !< driver [xyz] coordinate
1220   real(dp),intent(in) :: ydriver        !< driver [xyz] coordinate
1221   real(dp),intent(in) :: zdriver        !< driver [xyz] coordinate
1222   real(dp), dimension(1:ngrid),intent(out) :: driverweight !< fraction of the cell
         volume that is inside the driver area
1224
1224   integer:: i,ind_grid,ix,iy,iz,nn
1225 ! integer:: subgridsize = 10 ! 3d subgrid
1226   integer:: randgridsize = 100 ! 3d random subgrid
1227   real(dp):: dx2,r2,rr!,rd
1228   real(dp):: xmin,ymin,zmin,xmax,ymax,zmax
1229   real(kind=8)::help_k8
1230   real(dp):: help1, help2, help3, help4, help5
1231   real(dp), dimension(1:3) :: skip_loc
1232   real(dp), dimension(1:3) :: xc
1233   integer,dimension( IRandNumSize ) :: &
1234        & localseed = (/ 3281, 4041, 595, 2376 /)
1236
1236   driverweight(:)=0.0_dp
1237   r2=rdriver_scaled**2
1238   rr=(rdriver_scaled-0.01_dp*dx)**2
```

```
1239    dx2=dx**2
1241
1241    !ind=1,2**ndim
1242    !2d: ind=1,4
1243    !3d: ind=1,8
1244    ! Set new grids position
1245    iz=(ind-1)/4          ! integer division -> 0 or 1
1246    iy=(ind-1-4*iz)/2     ! integer division -> 0 or 1
1247    ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
1248    xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1249 #if NDIM>1
1250    xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1251 #endif
1252 #if NDIM>2
1253    xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5_dp or +0.5_dp
1254 #endif
1256
1256    skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
1257    skip_loc(1)=dble(icoarse_min)
1258 #if NDIM>1
1259    skip_loc(2)=dble(jcoarse_min)
1260 #endif
1261 #if NDIM>2
1262    skip_loc(3)=dble(kcoarse_min)
1263 #endif
1265
1265    !xg(:,1)-1.5 ... values in interval [-1,1]
1266    do i=1,ngrid
1267      ind_grid=active(ilevel)%igrid(igrid+i-1)
1268      !xg(ind_grid(i),1) .. x coordinate of the center of the subgrid
1269      xmax=abs(xg(ind_grid,1)+xc(1)-skip_loc(1)-0.5_dp-xdriver)+0.5_dp*dx ! minimum:
           0.5_dp*dx
1270      xmin=xmax-dx !xmax-dx can get < 0, minimum: -0.5_dp*dx, but abs(zmin)<abs(zmax
           )
1271 #if NDIM>1
1272      ymax=abs(xg(ind_grid,2)+xc(2)-skip_loc(2)-0.5_dp-ydriver)+0.5_dp*dx ! minimum:
           0.5_dp*dx
1273      ymin=ymax-dx !ymax-dx can get < 0, minimum: -0.5_dp*dx, but abs(zmin)<abs(zmax
           )
1274 #else
1275      ymax=0.0_dp
1276      ymin=0.0_dp
1277 #endif
1278 #if NDIM>2
1279      zmax=abs(xg(ind_grid,3)+xc(3)-skip_loc(3)-0.5_dp-zdriver)+0.5_dp*dx ! minimum:
           0.5_dp*dx
1280      zmin=zmax-dx !zmax-dx can get < 0, minimum: -0.5_dp*dx, but abs(zmin)<abs(zmax
           )
1281 #else
1282      zmax=0.0_dp
1283      zmin=0.0_dp
1284 #endif
1285      if((xmin**2+ymin**2+zmin**2).lt.rr)then
1286      !part of cell inside driver region
1287        if((xmax**2+ymax**2+zmax**2).lt.rr)then
```

```
1288          !cell fully inside driver region
1289          driverweight(i)=1.0_dp
1290        else
1291 #if NDIM==1
1292 ! 1d
1293          !if(xmin.gt.0.0_dp)then: (r-|xmin|) ... part of r inside interval
1294          !if(xmin.lt.0.0_dp)then: (r+|xmin|) ... part of r inside positive part of
                 interval plus negative part of interval
1295          !in both cases: (r-xmin)
1296          driverweight(i)=(rdriver_scaled-xmin)/dx
1297 #endif
1298 #if NDIM==2
1299 ! 2d
1300          if (max(xmin**2,ymin**2)+min(xmax,ymax)**2.lt.rr) then !at least two
                 corners inside central region
1301           if (min(xmin**2,ymin**2)+max(xmax,ymax)**2.gt.rr) then !two corners
                  inside central region
1302            if((ymin.lt.0.0_dp).and.(xmin.lt.0.0_dp))then
1303              print*,"xmin AND ymin negative: use a larger driver radius!"
1304              print*,"xmin=",xmin, "xmax=",xmax
1305              print*,"ymin=",ymin, "ymax=",ymax
1306              stop
1307            end if
1308            help1=min(xmin,ymin)/rdriver_scaled ! lower boundary of the integral
1309            help2=sqrt(1.0_dp-help1**2)          ! surface of sphere @ lower
                   boundary of the integral
1310            help3=min(xmax,ymax)/rdriver_scaled ! upper boundary of the integral
1311            help4=sqrt(1.0_dp-help3**2)          ! surface of sphere @ upper
                   boundary of the integral
1312 !          help5: subtract rectangle between x-axis (if xmin < ymin, otherwise y-
     axis) and ymin (if xmin < ymin, otherwise xmin) and integral boundaries
1313            help5=(-max(xmin,ymin)*dx/r2)
1314           else !three corners inside central region, ymax/xmax corner outside
1315            !okay for (ymin.lt.0.0_dp.and.xmin.lt.0.0_dp)
1316            help2=ymax/rdriver_scaled  ! surface of sphere @ lower boundary of the
                   integral
1317            help1=sqrt(1.0_dp-help2**2) ! lower boundary of the integral
1318            help3=xmax/rdriver_scaled  ! upper boundary of the integral
1319            help4=sqrt(1.0_dp-help3**2) ! surface of sphere @ upper boundary of the
                    integral
1320 !          help5: subtract rectangle between (x-axis and ymin) and integral
     boundaries
1321 !          for ymin<0  add rectangle between (x-axis and ymin) and integral
     boundaries
1322 !          help5: add rectangle between (xmin and help1) and (ymin and ymax) (okay
      for xmin<0)
1323            help5=(-(help3-help1))*ymin/rdriver_scaled+                    &
1324 &                (help1-xmin/rdriver_scaled)*dx/rdriver_scaled
1325           end if
1326          else !one corner inside central region
1327            if((ymin.lt.0.0_dp).and.(xmin.lt.0.0_dp))then
1328              print*,"xmin AND ymin negative: use a larger driver radius!"
1329              print*,"xmin=",xmin, "xmax=",xmax
1330              print*,"ymin=",ymin, "ymax=",ymax
1331              stop
```

```
1332            end if
1333            help1=min(ymin,xmin)/rdriver_scaled  ! lower boundary of the integral
1334            help2=sqrt(1.0_dp-help1**2)           ! surface of sphere @ lower
                   boundary of the integral
1335            help4=max(xmin,ymin)/rdriver_scaled  ! surface of sphere @ upper
                   boundary of the integral
1336            help3=sqrt(1.0_dp-help4**2)           ! upper boundary of the integral
1337 !          if(ymin.lt.0.0_dp) help5: subtract rectangle between (y-axis and xmin)
       and integral boundaries
1338 !          else               help5: subtract rectangle between (x-axis and ymin)
       and integral boundaries
1339            help5=(-(help3-help1))*help4 ! also ok if help1 < 0
1340          end if
1341 !        weight:
1342 !       (r2* ...) : lower and upper boundary of (see e.g. Netz, 7th.edition
       integral 113) \int_{xmin}^{x @ ymin}\sqrt(1-(x/r)^2)dx = 0.5*(xy+arcsin(x))
1343 !        help5: subtract rectangle between x-axis and lowest y
1344 !        help5: add rectangle between xmin and the lower boundary of the integral
1345          driverweight(i) = (0.5_dp*(help3*help4+asin(help3)-          &
1346 &        help1*help2-asin(help1))+help5)*r2/dx2
1347          if(driverweight(i).gt.1._dp)then
1348            if(verbose_patches) print*,"driverweight(i)", driverweight(i)
1349            if(verbose_patches) print*,"xmin,ymin",xmin,ymin
1350            if(verbose_patches) print*,"xmax,ymax",xmax,ymax
1351            if(verbose_patches) print*,"rr",rr
1352            driverweight(i)=0._dp
1353            stop 'exiting: driverweight > 1 '
1354          else if(driverweight(i).lt.0._dp)then
1355            if(verbose_patches) print*,"driverweight(i)", driverweight(i)
1356            if(verbose_patches) print*,"xmin,ymin",xmin,ymin
1357            if(verbose_patches) print*,"xmax,ymax",xmax,ymax
1358            if(verbose_patches) print*,"rr",rr
1359            driverweight(i)=0._dp
1360            stop 'exiting: driverweight < 0'
1361          end if
1362 #endif
1363 #if NDIM==3
1364 ! 3d Monte Carlo
1365          nn=0
1366          do ix=1,randgridsize
1367            call ranf(localseed,help_k8)
1368            help1=xmin+dble(help_k8)*dx
1369            call ranf(localseed,help_k8)
1370            help2=ymin+dble(help_k8)*dx
1371            call ranf(localseed,help_k8)
1372            help3=zmin+dble(help_k8)*dx
1373            if((help1**2+help2**2+help3**2).lt.r2)then
1374              nn=nn+1
1375            end if
1376          end do
1377          driverweight(i) = dble(nn)/dble(randgridsize)
1378 #endif
1379        end if
1380      else
1381        driverweight(i)=0.0_dp
```

```
1382     end if
1383   end do
1384 end subroutine driver_weights_analyt
1385 end module driver
```

Listing C.2: New module with tabulated stellar models for Ramses: geneva_models.f90

```
1  module geneva_models
2   use amr_parameters, only: dp,ifgeneva,genevarotating,genevayear,mstars,&
3  & tstars,n_stars
4  ! Stellar feedback
5  !  integer,parameter::MAXSTARS=100
6  !  real(dp) :: n_stars = 10.0_dp  ! number of OB stars inside the driver region
7  !  logical::ifgeneva=.false.      ! use geneva models -> ignore/overwrite
       file_driver and file_sn
8  !  logical::genevarotating=.true. ! use rotating geneva models
9  !  integer::genevayear=2011       ! chose geneva grid
10 !  real(dp),dimension(1:MAXSTARS)::mstars=9.0_dp ! mass of the stars
11 !  real(dp),dimension(1:MAXSTARS)::tstars=0.0_dp ! formation of the stars at this
       time
12  implicit none
13  save ! retain the value of the variables from one call to the next
14  integer, parameter :: i9 = selected_int_kind(r=9) !< integer type definition
15 ! integer,parameter::dp=kind(1.0E0) ! real type definition
16  integer, parameter :: n_points = 400
17  integer, parameter :: n_models = 11
19
19  type sn_matrix
20 #if NPRE==4
21     real(kind=8)::timeSN                          !yr
22     real(kind=8)::masslossSN                      !msun
23     real(kind=8)::Al26SN                          !msun
24     real(kind=8)::energySN                        !erg
25 #else
26     real(dp)::timeSN                              !yr
27     real(dp)::masslossSN                          !msun
28     real(dp)::Al26SN                              !msun
29     real(dp)::energySN                            !erg
30 #endif
31  end type sn_matrix
33
33  type driver_matrix
34 #if NPRE==4
35     real(kind=8)::timeSN                          !yr
36     real(kind=8)::masslossSN                      !msun
37     real(kind=8)::energySN                        !erg
38     real(kind=8),dimension(1:n_points)::time      !yr
39     real(kind=8),dimension(1:n_points)::massloss  !msun/yr
40     real(kind=8),dimension(1:n_points)::velocity  !km/s
41     real(kind=8),dimension(1:n_points)::energy    !1e30 erg/s
42     real(kind=8),dimension(1:n_points)::Al26      !msun/yr
43 #else
44     real(dp)::timeSN                              !yr
45     real(dp)::masslossSN                          !msun
46     real(dp)::energySN                            !erg
47     real(dp),dimension(1:n_points)::time          !yr
```

```
48      real(dp),dimension(1:n_points)::massloss      !msun/yr
49      real(dp),dimension(1:n_points)::velocity      !km/s
50      real(dp),dimension(1:n_points)::energy        !1e30 erg/s
51      real(dp),dimension(1:n_points)::Al26          !msun/yr
52  #endif
53   !   real(dp),dimension(1:n_points)::z  !mass fraction (1-H-He)
54   end type driver_matrix
56
56   integer,parameter,dimension(1:n_models)::initialmass = &
57   & (/7,9,12,15,20,25,32,40,60,85,120/)
59
59   type(driver_matrix),dimension(1:n_models)::Geneva2011V4
61
61   type(SN_matrix),dimension(1:n_models)::VossGenevaAl
63
63  contains
65
65  subroutine create_VossGenevaAl
66  #if NPRE==4
67   VossGenevaAl(1:n_models)%timeSN = (/ 0e0_8, 3.685e+07_8, 2.195e+07_8, &
68  & 1.555e+07_8, 11.05e+06_8, 8.65e+06_8,       6.95e+06_8,  5.95e+06_8,  &
69  & 4.65e+06_8,   3.85e+06_8, 3.45e6_8 /) ! [years]
71
71   VossGenevaAl(1:n_models)%masslossSN = (/  0e0_8, 7.06781_8, 9.1462_8, &
72  & 9.0551_8, 10.4174_8, 10.6538_8, 3.9536_8,       3.4385_8,  5.5983_8, &
73  & 9.8102_8, 4.4746_8 /) ! [solar masses]
75
75   VossGenevaAl(1:n_models)%Al26SN  = (/    0e0_8, 0.556278e-05_8,       &
76  & 1.94186e-05_8, 12.7075e-05_8, 5.4382336e-05_8, 9.77063e-05_8,        &
77  & 9.41636e-05_8, 10.3988e-05_8, 18.3466e-05_8, 29.5453e-05_8,          &
78  & 0.063e-05_8 /) ! [solar masses]
80
80  #else
81   VossGenevaAl(1:n_models)%timeSN = (/ 0e0_dp, 3.685e+07_dp,             &
82  & 2.195e+07_dp, 1.555e+07_dp, 11.05e+06_dp, 8.65e+06_dp, 6.95e+06_dp,  &
83  & 5.95e+06_dp,  4.65e+06_dp,   3.85e+06_dp, 3.45e6_dp /) ! [years]
85
85   VossGenevaAl(1:n_models)%masslossSN = (/ 0e0_dp, 7.06781_dp,           &
86  & 9.1462_dp, 9.0551_dp, 10.4174_dp, 10.6538_dp, 3.9536_dp,             &
87  & 3.4385_dp, 5.5983_dp,  9.8102_dp,  4.4746_dp /) ! [solar masses]
89
89   VossGenevaAl(1:n_models)%Al26SN  = (/ 0e0_dp, 0.556278e-05_dp,        &
90  & 1.94186e-05_dp, 12.7075e-05_dp,  5.4382336e-05_dp, 9.77063e-05_dp,   &
91  & 9.41636e-05_dp, 10.3988e-05_dp, 18.3466e-05_dp,   29.5453e-05_dp,    &
92  & 0.063e-05_dp /) ! [solar masses]
93  #endif
95
95  #if NPRE==4
96   VossGenevaAl(1:n_models)%energySN = 1.0_8 ! [erg]
97  #else
98   VossGenevaAl(1:n_models)%energySN = 1.0_dp ! [erg]
99  #endif
100 end subroutine create_VossGenevaAl
101 !-------------------------------------------------------------------------
102 subroutine create_Geneva2011V4
```

```
103 !------------------------------------------------------------------------
104 ! SN data
105 !------------------------------------------------------------------------
106 !grep  "400 " M???Z14V4.dat | grep ":400 " > SN.txt
107 !#col.  1: line number and initial mass
108 !#col.  2: age [yr]
109 !#col.  3: mass [Msol]
110 !awk '{split($1,help1,"M");split(help1[2],help2,"Z");sub(/p/, ".", help2[1]);
        initialmass=help2[1]; mass=$3; age=$2; if(initialmass<25){remanentmass=1.4}
        else{remanentmass=7.0};if(initialmass>6){print initialmass, age, mass,
        remanentmass, mass-remanentmass}}' SN.txt
111 !------------------------------------------------------------------------
112 #if NPRE==4
113  Geneva2011V4(1:n_models)%timeSN = (/ 5.89825423207157e7_8,             &
114 & 3.54627289784629e7_8, 2.07324437587086e7_8, 1.50658661141411e7_8,    &
115 & 1.04749575317549e7_8, 8.60585058063150e6_8, 7.22426176827787e6_8,    &
116 & 6.17506476706730e6_8, 4.85966398974075e6_8, 4.06404560769163e6_8,    &
117 & 3.55717089269368e6_8 /) ! [years]
118  Geneva2011V4(1:n_models)%masslossSN = (/5.46839_8,7.11747_8,           &
119 & 8.82398_8,9.67124_8,5.77851_8,2.68962_8,3.12489_8,5.33227_8,          &
120 & 10.9807_8,19.3934_8,12.0444_8/) ! [solar masses]
121  Geneva2011V4(1:n_models)%energySN = 1.0_8 ! [erg]
122 #else
123  Geneva2011V4(1:n_models)%timeSN = (/ 5.89825423207157e7_dp,            &
124 & 3.54627289784629e7_dp, 2.07324437587086e7_dp, 1.50658661141411e7_dp, &
125 & 1.04749575317549e7_dp, 8.60585058063150e6_dp, 7.22426176827787e6_dp, &
126 & 6.17506476706730e6_dp, 4.85966398974075e6_dp, 4.06404560769163e6_dp, &
127 & 3.55717089269368e6_dp /) ! [years]
128  Geneva2011V4(1:n_models)%masslossSN = (/5.46839_dp,7.11747_dp,         &
129 & 8.82398_dp,9.67124_dp,5.77851_dp,2.68962_dp,3.12489_dp,5.33227_dp,   &
130 & 10.9807_dp,19.3934_dp,12.0444_dp/) ! [solar masses]
131  Geneva2011V4(1:n_models)%energySN = 1.0_dp ! [erg]
132 #endif
133 !------------------------------------------------------------------------
134 ! Wind 120 solar masses
135 !------------------------------------------------------------------------
136 ! awk '{print $2 }' M007Z14V4.dat
138
138 ! Geneva2011V4(1:n_models)%time      !yr
139 ! Geneva2011V4(1:n_models)%massloss  !msun/yr
140 ! Geneva2011V4(1:n_models)%velocity  !km/s
141 ! Geneva2011V4(1:n_models)%energy    !1e30 erg/s
142 ! Geneva2011V4(1:n_models)%Al26      !msun/yr
143 ! Geneva2011V4(1:n_models)%z         !mass fraction (1-H-He)
145
145 Geneva2011V4(11)%time = (/0.188397310560790E+05, 0.243588632933556E+05, &
4977 Geneva2011V4(1)%Al26 = 0.0
4978 Geneva2011V4(1)%velocity = (/2.43448e+08, 1.21021e+08, 1.20297e+08, 1.19599e+08,
        &
5059 end subroutine create_Geneva2011V4
5060 subroutine scale_Geneva2011V4
5061  implicit none
5062  integer::ii,jj
5063  real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
        factors between cgs and user units (subroutine units in units.f90)
```

```
5064 #if NPRE==4
5065  real(kind=8) :: scale_energy, scale_m, scale_dm,scale_e !< conversion factors
         between cgs and user units
5066  real(kind=8), parameter :: YearToSeconds = 31556926._8  !< convert years to
         seconds; 1 year = 31556926 seconds
5067  real(kind=8), parameter :: SolarMass = 1.98892e33_8     !< solar mass in [g]
5068 #else
5069  real(dp) :: scale_energy, scale_m, scale_dm,scale_e !< conversion factors between
          cgs and user units
5070  real(dp), parameter :: YearToSeconds = 31556926._dp !< convert years to seconds;
         1 year = 31556926 seconds
5071  real(dp), parameter :: SolarMass = 1.98892e33_dp    !< solar mass in [g]
5072 #endif
5073  call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
5074  scale_e=51.0-(log10(scale_d)+3.*log10(scale_l)+2.*log10(scale_v))  ! 0
5075  scale_m=scale_d*scale_l**3                                         !10^(+35)
5076  !1 year = 31556926 seconds
5077  !mass loss = 1 solar mass / year
5078  !mass loss = \f$ \frac{1.98892 \times 10^{33}}{31556926} \frac{\rm [g]}{\rm [s]}
         \f$
5079  !mass loss = 63.e+24 g/s
5080  scale_dm=SolarMass/YearToSeconds*scale_t/scale_m
5081  !kinetic luminosity: erg/s = g cm^2 / s^3
5082  !1 erg/s = 10^(-35-2*19+3*11) code-mass units code length units^2/code-time-units
         ^3
5083  !1 erg/s = 10^(-40) code-mass units code length units^2/code-time-units^3
5084  scale_energy=scale_t/scale_m/scale_v**2                !10^(-40)
5085  do ii=1,SIZE(Geneva2011V4)
5086   Geneva2011V4(ii)%timeSN      = Geneva2011V4(ii)%timeSN*YearToSeconds/ &
5087 &  scale_t                                             !yr to code units
5088   Geneva2011V4(ii)%masslossSN = Geneva2011V4(ii)%masslossSN*SolarMass/ &
5089 &  scale_m                                             !msun to code units
5090   Geneva2011V4(ii)%energySN    = Geneva2011V4(ii)%energySN*10.**scale_e !erg to
         code units
5091   do jj=1,SIZE(Geneva2011V4(1)%time)
5092     Geneva2011V4(ii)%time(jj)      = &
5093 &     Geneva2011V4(ii)%time(jj)*YearToSeconds/scale_t    !yr to code units
5094     Geneva2011V4(ii)%massloss(jj) = &
5095 &     Geneva2011V4(ii)%massloss(jj)*scale_dm             !msun/yr to code units
5096     Geneva2011V4(ii)%Al26(jj)      = &
5097 &     Geneva2011V4(ii)%Al26(jj)*scale_dm                 !msun/yr to code units
5098     Geneva2011V4(ii)%energy(jj)    = &
5099 #if NPRE==4
5100 &     Geneva2011V4(ii)%energy(jj)*1e30_8*scale_energy    !erg/s  to code units
5101 #else
5102 &     Geneva2011V4(ii)%energy(jj)*1e30_dp*scale_energy   !erg/s  to code units
5103 #endif
5104     Geneva2011V4(ii)%velocity(jj) = Geneva2011V4(ii)%velocity(jj)/scale_v !km/s to
          code units
5105   end do
5106  end do
5107 end subroutine scale_Geneva2011V4
5108 subroutine print_Geneva2011V4
5109  use amr_commons, only: myid,ncpu
5110  implicit none
```

```
5111  integer(i9) :: i,j,ii,jj,ilun
5112  ilun=ncpu+myid+10
5113  open(unit=ilun,file="Geneva.txt",form='formatted')
5114  write(ilun,'("#time␣[yr],␣energy␣loss␣[erg/s],␣mass␣loss␣[msun/yr],␣",&
5115 &␣"wind␣velocity␣[cm/s],␣26AL␣[Msun/yr],␣stars",I4)') int(n_stars)
5116  do j=1,int(n_stars)
5117   do jj=1,11
5118     print*,initialmass(jj),mstars(j)
5119     if(int(initialmass(jj)).eq.int(mstars(j)))then
5120       ii=jj ! compute number of the model of this mass
5121     end if
5122   end do
5123   write(*,'("#␣star␣",I4,1X,"formation␣time␣[yr]",G14.5E4␣,␣␣␣␣␣␣␣␣␣␣␣␣␣&
5124 &␣␣␣␣"␣initial␣mass␣[Msun]",G14.5E4)')j,tstars(j),mstars(j)
5125   write(ilun,'("#␣star␣",I4,1X,"formation␣time␣[yr]",G14.5E4␣,␣␣␣␣␣␣␣␣␣&
5126 &␣␣␣"␣initial␣mass␣[Msun]",G14.5E4)')j,tstars(j),mstars(j)
5127   do i=1,n_points
5128   !energy in tables is 1e30 erg/s
5129    write(ilun,'(5(2x,G14.5E4))') Geneva2011V4(ii)%time(i)+tstars(j),   &
5130 &   Geneva2011V4(ii)%energy(i)*1e30_dp, Geneva2011V4(ii)%massloss(i),  &
5131 &   Geneva2011V4(ii)%velocity(i),       Geneva2011V4(ii)%Al26(i)
5132   end do
5133   write(ilun,'("SN␣at␣t=␣",G14.5E4)')Geneva2011V4(ii)%timeSN+tstars(j)
5134   write(ilun,'("SN␣energy␣[erg]␣=␣",G14.5E4)') Geneva2011V4(ii)%energySN
5135   write(ilun,'("SN␣mass␣loss␣[Msun]␣=␣",G14.5E4)') Geneva2011V4(ii)%masslossSN
5136   write(ilun,'("26Al␣fraction=␣",G14.5E4)') &
5137 &   Geneva2011V4(ii)%AL26(n_points)/Geneva2011V4(ii)%massloss(n_points) ! too high
      - surface mass fraction not mass fraction in ejecta
5138  end do
5139  close(ilun)
5140 end subroutine print_Geneva2011V4
5141 subroutine interpolate_Geneva2011V4 (age,dt,edriver,rhodriver,aldriver)
5142  implicit none
5143  real(dp), intent(in)  :: age ! simulation time
5144  real(dp), intent(in)  :: dt  ! time step size (for SN)
5145  real(dp), intent(out) :: edriver, rhodriver, aldriver
5146  integer(i9) :: i,j,ii,jj
5147  real(dp)::scaled_age ! simulation time corrected for star formation time
5149
5149  edriver   = 0.0_dp
5150  rhodriver = 0.0_dp
5151  aldriver  = 0.0_dp
5152  do j=1,int(n_stars)
5153   do jj=1,11
5154     if(initialmass(jj).eq.mstars(j))then
5155       ii=jj ! search for the number of the model for this mass
5156     end if
5157   end do
5158   scaled_age = age - tstars(j)
5159   if ((scaled_age.lt.Geneva2011V4(ii)%timeSN).and.                    &
5160 &      (scaled_age.ge.Geneva2011V4(ii)%time(1))) then
5161    i=1
5162    do while (scaled_age.gt.Geneva2011V4(ii)%time(i))
5163      i=i+1
5164    end do
```

```
5165     edriver =  edriver + Geneva2011V4(ii)%energy(i-1)+                    &
5166 &   (scaled_age-Geneva2011V4(ii)%time(i-1))/                              &
5167 &   (Geneva2011V4(ii)%time(i)-Geneva2011V4(ii)%time(i-1))*                 &
5168 &   (Geneva2011V4(ii)%energy(i)-Geneva2011V4(ii)%energy(i-1))
5169     rhodriver=rhodriver+Geneva2011V4(ii)%massloss(i-1)+                   &
5170 &   (scaled_age-Geneva2011V4(ii)%time(i-1))/                              &
5171 &   (Geneva2011V4(ii)%time(i)-Geneva2011V4(ii)%time(i-1))*                 &
5172 &   (Geneva2011V4(ii)%massloss(i)-Geneva2011V4(ii)%massloss(i-1))
5173     aldriver=aldriver+Geneva2011V4(ii)%Al26(i-1)+                         &
5174 &   (scaled_age-Geneva2011V4(ii)%time(i-1))/                              &
5175 &   (Geneva2011V4(ii)%time(i)-Geneva2011V4(ii)%time(i-1))*                 &
5176 &   (Geneva2011V4(ii)%Al26(i)-Geneva2011V4(ii)%Al26(i-1))
5177   else if  ((scaled_age.gt.Geneva2011V4(ii)%timeSN).and.                  &
5178 &           (scaled_age-dt.le.Geneva2011V4(ii)%timeSN)) then
5179     print*, 'time: ', scaled_age
5180     print*, "SN at t= ", Geneva2011V4(ii)%timeSN
5181     print*, "SN energy [erg] = ", Geneva2011V4(ii)%energySN
5182     print*, "SN mass loss [Msun] = ", Geneva2011V4(ii)%masslossSN
5183     print*, "26Al fraction= ",                                            &
5184 &    Geneva2011V4(ii)%AL26(n_points)/Geneva2011V4(ii)%massloss(n_points)
5185     edriver  = edriver   + Geneva2011V4(ii)%energySN/dt
5186     rhodriver= rhodriver + Geneva2011V4(ii)%masslossSN/dt
5187     aldriver = aldriver  + Geneva2011V4(ii)%masslossSN/dt *               &
5188 &    Geneva2011V4(ii)%AL26(n_points)/Geneva2011V4(ii)%massloss(n_points) ! too high
        - surface mass fraction not mass fraction in ejecta
5189   end if
5190  end do
5191 end subroutine interpolate_Geneva2011V4
5192 end module geneva_models
```

Listing C.3: Stellar feedback control: amr_parameters.f90

```
68   ! Stellar feedback control
69   integer,parameter::MAXSTARS=100
70   character(LEN=128) :: file_driver = 'wind.dat' ! file with wind data
71   character(LEN=128) :: file_sn = 'sn.dat'       ! file with SN data
72   character(LEN=128) :: file_sph = 'cloud1_ka_new' ! SPH particles for initial
        conditions
73   real(dp) :: r_driver = 0.75_dp ! driver radius (in user length units)
74   real(dp) :: x_driver = 0.0_dp  ! x coordinate of the driver center (in user
        length units)
75   real(dp) :: y_driver = 0.0_dp  ! y coordinate of the driver center (in user
        length units)
76   real(dp) :: z_driver = 0.0_dp  ! z coordinate of the driver center (in user
        length units)
77   real(dp) :: coolplus = 0.0_dp  ! space between cooling region and driver region
        (in user length units)
78   real(dp) :: n_stars =  1.0_dp  ! number of OB stars inside the driver region
79   integer :: max_driver_grid = 7 ! amr: refinement of driver region
80   logical::ifgeneva=.false.        ! use geneva models -> ignore/overwrite
        file_driver and file_sn
81   logical::genevarotating=.true. ! use rotating geneva models
82   integer::genevayear=2011       ! chose geneva grid
83   real(dp),dimension(1:MAXSTARS)::mstars=9.0_dp ! mass of the stars
84   real(dp),dimension(1:MAXSTARS)::tstars=0.0_dp ! formation of the stars at this
        time
```

```
164   real(dp)::T_min_fix=1.e-2_dp !minimum temperature for cooling table
```

Listing C.4: Read feedback parameters: read_params.f90

```
25    namelist/driver_params/file_driver,file_sn,file_sph,r_driver,          &
26         & x_driver,y_driver,z_driver,coolplus,n_stars,max_driver_grid,    &
27         & ifgeneva, genevarotating, genevayear, mstars, tstars
66     write(*,*)'                    Version 3.10                    '
67     write(*,*)'    https://bitbucket.org/rteyssie/ramses August 5th 2014    '
68     write(*,*)'    git commit 5a2d93b83d13bb48f21077c61cfda275256d8aea    '
69     write(*,*)'       written by Romain Teyssier (CEA/DSM/IRFU/SAP)        '
70     write(*,*)'                    (c) CEA 1999-2007                    '
71     write(*,*)'       with stellar feedback patches of K.M. Fierlinger      '
126   max_driver_grid=levelmax
127   read(1,NML=driver_params)
128   !-------------------
129   ! Max star number checks
130   !-------------------
131   if((ifgeneva).and.(n_stars>MAXSTARS))then
132      write(*,*) 'Error: n_stars>MAXSTARS'
133      call clean_stop
134   end if
135   rewind(1)
```

Listing C.5: Read-in of feedback parameters: read_hydro_params.f90

```
19    namelist/init_params/filetype,initfile,multiple,nregion,region_type  &
20         & ,x_center,y_center,z_center,aexp_ini &
21         & ,length_x,length_y,length_z,exp_region &
22 #if NENER>0
23         & ,prad_region &
24 #endif
25         & ,d_region,u_region,v_region,w_region,p_region,al_region
26    !var_region, that initializes the passive scalars is always zero!
27    namelist/hydro_params/gamma,courant_factor,smallr,smallc,larget       &
39    namelist/physics_params/cooling,T_min_fix,haardt_madau,metal          &
```

Listing C.6: Allocate feedback data: init_time.f90

```
127   ! Initialize wind table
128   if(ifgeneva)then
129    call create_Geneva2011V4
130 #ifndef WITHOUTMPI
131    if(myid==1)then
132 #endif
133      print*, "Geneva 2011 V4 models z=0.014"
134      call print_Geneva2011V4
135 #ifndef WITHOUTMPI
136    end if
137 #endif
138    call scale_Geneva2011V4
139  else
140    call read_driver
141    call read_sn
142  end if
143  call allocate_driver_mask
```

```
144 end subroutine init_time
```

Listing C.7: Insert feedback: courant_fine.f90

```
82    ! Insert stellar wind
83    if(n_stars.gt.0.0_dp)then
84      call wind_fine(ilevel)
85    end if
251 subroutine wind_fine(ilevel)
252 !------------------------------------------------------------------------
253 !> \version "sn+wind": thermal energy and mass loss read in from sn.dat and wind.
      dat
254 !> \author Katharina M. Fierlinger
255 !> \date last modification 13.09.2011
256 !------------------------------------------------------------------------
257 !  use amr_commons, only: active, dtold, ncoarse, numbtot, xg
258   use driver
259   use geneva_models
260   use amr_commons, only: active, dtold, ncoarse, numbtot, t , T_min_fix
261   use amr_parameters, only: r_driver,x_driver,y_driver,z_driver,n_stars, &
262 &  ifgeneva,genevarotating,genevayear,mstars,tstars
263   use hydro_commons, only: uold, nvar, gamma, smallr, i26al, i60fe
264   use poisson_parameters, only : dp, icoarse_max, icoarse_min, boxlen,  &
265 &  verbose, verbose_patches, nvector, ndim, ngridmax, twotondim
266   !this subroutine uses make_virtual_fine_dp
267   implicit none
268   integer::ilevel
270
270   integer::igrid,ncache,i,ind,iskip,ngrid
271   integer::ivar,ind_grid,ind_cell
272 #ifdef DECAYINTERVAL
273   real(dp),parameter :: decay_interval = 0.1578_dp
274 #endif
276
276   real(dp), dimension(1:nvector):: weight
277 #ifdef EKIN
278 #if NDIM>1
279   real(dp), dimension(1:nvector):: weightx
280   real(dp), dimension(1:nvector):: weighty
281 #endif
282 #if NDIM>2
283   real(dp), dimension(1:nvector):: weightz
284 #endif
285 #endif
286   real(dp) :: mdriver = 0.0_dp ! 5.76015d-5
287   real(dp) :: edriver = 0.0_dp ! 1.57015d-4
288   real(dp) :: driver26Al = 0.0_dp
289   real(dp) :: driver60Fe = 0.0_dp
290 #if DEBUG==2
291   real(dp) :: driverTest = 0.0_dp
292 #endif
293 #if DEBUG==3
294 !-------------------------------
295 !       check if kinetic energy is still smaller than total energy
296 real(dp)::OLD_rho, OLD_Etot, OLD_mx, OLD_Ekin_help
297 real(dp)::OLD_my, OLD_mz
```

```
298 #endif
299 #ifdef CARINA
300   real(dp) :: edriver_help,mdriver_help,driver26Al_help,driver60Fe_help
301   !real(dp) :: star_formation_periode = 1893.41556_dp !code time unit: 1.d11s=
        3.16887646 kyr , 6 Myr = 1.89341556e14 s =  1893.41556 ctu
302   real(dp) :: star_formation_periode = 946.70778_dp !code time unit: 1.d11s=
        3.16887646 kyr , 3 Myr = 9.4670778e13 s = 946.70778 ctu
303   real(dp) :: star_formation_interval
304   integer::existing_stars,ii
305 #endif
306   real(dp), parameter :: pi = acos(-1.0_dp)
307 #ifdef THII
308   real(dp), parameter :: T_HII =  1.0e4_dp
309 #endif
310 #ifdef TMAX
311   real(dp), parameter :: T_max =  5.0e6_dp
312 #endif
313   !
314   ! logical:: debug = .true.
315   ! real(dp)::xx
316   ! real(dp)::r2
317   !
318   real(dp) :: xdriver = 0.0_dp
319   real(dp) :: ydriver = 0.0_dp
320   real(dp) :: zdriver = 0.0_dp
321   real(dp)::rdriver_scaled
322 !  real(dp)::r3
323   real(dp)::dx,one_over_boxscale
324 #ifdef EKIN
325   real(dp)::momentum_help
326 #endif
327 #if defined (EKIN) || defined (THII) || defined (TMAX) || defined (TMIN)
328   real(dp)::Ekin_help
329 #endif
330 #if defined (THII) || defined (TMAX) || defined (TMIN)
331   real(dp)::T_help
332 #endif
333   real(dp)::de,dm,d26Al,d60Fe
334   real(dp)::deltat,one_over_v_sphere
335 #ifdef CARINA
336   real(dp)::DtV
337   real(dp)::eSN_help, mSN_help
338 #else
339   real(dp)::starsDtV
340 #endif
341  !26 Al settings
342  !half life time of 26Al; (7.17e5 \pm 0.24e5) years
343  ! Dr. Jagdish K. Tuli Nuclear Wallet Cards 2005 7th Edition
344  ! http://www.nndc.bnl.gov/wallet/wc7.html
345  !half life time of 26Al; (7.16e5 \pm 0.32e5) years
346  !real(dp), parameter :: thalf26Al = 7.16d12 * 3.1556926_dp ! [seconds]
347  real(dp), parameter :: thalf26Al = 2.2594759e13_dp! [seconds]
348  !60 Fe settings
349  !half life time of 60Fe; (2.62e6 \pm 0.04e6) years
350  !Rugel et al., Phys. Rev. Lett. 103, 072502 (2009)
```

```
351   real(dp), parameter :: thalf60Fe = 2.62d13 * 3.1556926_dp ! [seconds]
352   real(dp)::scale_nH,scale_T2,scale_l,scale_d,scale_t,scale_v
353   real(dp)::eSN = 0.0_dp
354   real(dp)::mSN = 0.0_dp
356
356 #if DEBUG==2
357     driverTest = 0.0_dp
358     if(verbose_patches)write(*,116) sum(uold(:,ndim+2))
359 #endif
361
361     if(numbtot(1,ilevel)==0)return
362     if(verbose)write(*,111)ilevel
364
364     ! Mesh spacing at that level
365     one_over_boxscale=dble(icoarse_max-icoarse_min+1)/boxlen
366     !scaled box:
367     dx = 0.5_dp**(ilevel)
368     !box in code units
369     !dx=0.5_dp**ilevel*boxscale
370     !vol=dx**ndim
371     !r2=rdriver**2
372     !r3=r_driver**3
373     xdriver = one_over_boxscale*x_driver
374     ydriver = one_over_boxscale*y_driver
375     zdriver = one_over_boxscale*z_driver
376     rdriver_scaled = one_over_boxscale*r_driver
378
378     !print*, "Driver coordinates", xdriver, ydriver
379     !print*, "driver radius" , r_driver
380     !print*, "scaled driver radius (boxsize: [0:1])" , rdriver_scaled, r2
381     !print*, "scaled grid spacing (boxsize: [0:1])", dx
382     !print*, "box length", boxlen
383     !print*, "box scale", boxscale
385
385     ! size of last timestep
386     if(dtold(ilevel).gt.0.0_dp)then
387       deltat=dtold(ilevel)
388       !!check if you already called read_driver in init_time (amr/init_time.f90)
389       if(ifgeneva)then
390        call interpolate_Geneva2011V4(t,deltat,edriver,mdriver,driver26Al)
391        driver60Fe=0.0_dp
392        eSN=0.0_dp
393        mSN=0.0_dp
394       else
395        call interpolate_driver (t,edriver,mdriver,driver26Al,driver60Fe)
396        call add_SN (t,deltat,eSN,mSN)
397       end if
398 #ifdef CARINA
399       star_formation_interval = t/star_formation_periode
400       existing_stars=max(1,min(70,int(70.0_dp*star_formation_interval)))
401       if(existing_stars.gt.1)then
402         do ii=2,existing_stars
403           !if(verbose_patches)write(*,114)ii,t-real(ii)*star_formation_interval
404           write(*,114)ii,t-real(ii)*star_formation_interval
405           if(ifgeneva)then
```

```
406             call interpolate_Geneva2011V4(t-real(ii)*star_formation_interval, &
407 &           deltat,edriver_help,mdriver_help,driver26Al_help)
408             ! scale since the interpolation used all stars
409             driver26Al_help=driver26Al_help/real(n_stars)
410             edriver_help   =edriver_help/real(n_stars)
411             mdriver_help   =mdriver_help/real(n_stars)
412             driver60Fe_help=0.0_dp
413             eSN_help=0.0_dp
414             mSN_help=0.0_dp
415           else
416             call interpolate_driver (t-real(ii)*star_formation_interval,     &
417 &           edriver_help,mdriver_help,driver26Al_help,driver60Fe_help)
418             call add_SN (t-real(ii)*star_formation_interval,deltat,          &
419 &           eSN_help,mSN_help)
420           end if
421           edriver=   edriver   +edriver_help
422           mdriver=   mdriver   +mdriver_help
423           driver26Al=driver26Al+driver26Al_help
424           driver60Fe=driver60Fe+driver60Fe_help
425           eSN=   eSN+eSN_help
426           mSN=   mSN+mSN_help
427         end do
428       end if
429 #endif
430       !!dont forget to call remove_driver in clean_stop (amr/update_time.f90)
431     else
432       eSN=0.0_dp
433       mSN=0.0_dp
434       deltat=1.e-4_dp
435       mdriver = 0.0_dp ! 5.76015d-5*n_stars
436       edriver = 0.0_dp ! 1.57015d-4*n_stars
437       driver26Al = 0.0_dp
438       driver60Fe = 0.0_dp
439       if(ifgeneva)then
440         call interpolate_Geneva2011V4(t,deltat,edriver,mdriver,driver26Al)
441       else
442         call interpolate_driver (t,edriver,mdriver,driver26Al,driver60Fe)
443       end if
444       if (verbose_patches) then
445         write(*,112)int(rdriver_scaled/dx)
446 !         one_over_v_sphere=0.75_dp/r3/pi
447           one_over_v_sphere=1._dp/get_driver_volume(ilevel)
448         write(*,117)one_over_v_sphere
449       end if
450     end if
451     !=> energy per driver region is distributed over cells
452     ! V_sphere = pi*r_driver**3/0.75
453 ! one_over_v_sphere=0.75_dp/r3/pi
454     one_over_v_sphere=1._dp/get_driver_volume(ilevel)
455 !     !e=mv^2/2 -> 2e /m
456 !     one_over_v_shell=0.75_dp/pi/(r3-(r_driver-deltat*sqrt(edriver*2.0_dp/mdriver)
      )**3)
457     ! injection of the wind of a given number of stars into V_sphere during the last
        timestep
458 #ifdef CARINA
```

```
459     DtV=deltat*one_over_v_sphere
460     de    =edriver    *DtV + eSN*one_over_v_sphere
461     dm    =mdriver    *DtV + mSN*one_over_v_sphere
462     d26Al=driver26Al*DtV
463     d60Fe=driver60Fe*DtV
464   #else
465     if(ifgeneva)then
466       starsDtV=deltat*one_over_v_sphere
467       de    =edriver    *starsDtV  ! [code-energy/code-length^3]
468       dm    =mdriver    *starsDtV
469     else
470       starsDtV=n_stars*deltat*one_over_v_sphere
471       de    =edriver    *starsDtV + eSN*n_stars*one_over_v_sphere ! [code-energy/code-
              length^3]
472       dm    =mdriver    *starsDtV + mSN*n_stars*one_over_v_sphere
473     end if
474     d26Al=driver26Al*starsDtV
475     d60Fe=driver60Fe*starsDtV
476   #endif
477   #if DEBUG==2
478     print*, "dm", dm, "de", de
479     if(verbose_patches)write(*,114)de
480   !   if(verbose_patches)write(*,113)driver26Al*n_stars*deltat
481   #endif
482     call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
483     ! Loop over active grids by vector sweeps
484     ncache=active(ilevel)%ngrid
485   ! if(debug)xx=0.0_dp
486     do igrid=1,ncache,nvector
487       ngrid=MIN(nvector,ncache-igrid+1)
488       ! Loop over cells
489       do ind=1,twotondim
490         iskip=ncoarse+(ind-1)*ngridmax
491         weight(:)=0.0_dp
492   #ifdef EKIN
493   #if NDIM==2
494         weightx(:)=0.0_dp
495         weighty(:)=0.0_dp
496         call driver_vector(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,  &
497   &       xdriver,ydriver,weightx(1:ngrid),weighty(1:ngrid))
498   #endif
499   #if NDIM==3
500         weightx(:)=0.0_dp
501         weighty(:)=0.0_dp
502         weightz(:)=0.0_dp
503         call driver_vector(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,  &
504   &       xdriver,ydriver,zdriver,                                    &
505   &       weightx(1:ngrid),weighty(1:ngrid),weightz(1:ngrid))
506   #endif
507   #endif
508         call driver_weights_fixed(ind,ilevel,igrid,ngrid,dx,weight(1:ngrid))
509   !       call driver_weights(ind,ilevel,igrid,ngrid,dx,rdriver_scaled,    &
510   !&                          xdriver,                                      &
511   !#if NDIM>1
512   !&                          ydriver,                                      &
```

```
513 !#endif
514 !#if NDIM>2
515 !&                          zdriver,                                    &
516 !#endif
517 !&                          weight(1:ngrid))
519
519 !      if(debug)xx=xx+sum(weight(1:ngrid))
520       do i=1,ngrid
521         ind_grid=active(ilevel)%igrid(igrid+i-1)
522         ind_cell=iskip+ind_grid
523 !       decay of 26Al and 60Fe
524 #ifdef DECAYINTERVAL
525 !(use larger timesteps to avoid subtracting a tiny number from a huge number)
526         if(floor(t/decay_interval).ne.floor((t+deltat)/decay_interval))then !
              decay_interval ends during this time step
527 !       decay of 26Al
528           uold(ind_cell,i26al)=uold(ind_cell,i26al)                       &
529 &         *2.0_dp**(-decay_interval*scale_t/thalf26Al)
530 !       decay of 60Fe
531           uold(ind_cell,i60fe)=uold(ind_cell,i60fe)                       &
532 &         *2.0_dp**(-decay_interval*scale_t/thalf60Fe)
533         end if
534 #else
535 !       decay of 26Al
536         uold(ind_cell,i26al)=uold(ind_cell,i26al)                         &
537 &         *2.0_dp**(-deltat*(scale_t/thalf26Al))
538 !       decay of 60Fe
539         uold(ind_cell,i60fe)=uold(ind_cell,i60fe)                         &
540 &         *2.0_dp**(-deltat*(scale_t/thalf60Fe))
541 #endif
542 !------------------------------------
543 !Driver region only
544 !------------------------------------
545         if(weight(i).gt.0.0_dp)then
546 !------------------------------------
547 !       mass
548           uold(ind_cell,1)=max(uold(ind_cell,1)+dm*weight(i),smallr)
549 #ifdef EKIN
550 ! dE = (dM v)^2 / (2 dM)
551 ! (dM v) = sqrt (dE 2 dM)
552 !! pure kinetic energy input (higher momentum)
553 !!          momentum_help=sqrt((de*weight(i)+Ekin_help)*2.0_dp*uold(ind_cell,1))
554 !! use driver-momentum
555 !! -> gas mixture will put unresolved kinetic energy into E_therm
556           momentum_help=sqrt(de*2.0_dp*dm)*weight(i)
557 #if DEBUG==2
558           print*,"wind␣speed", momentum_help/dm/weight(i)*scale_v, "[cm/s]"
559 #endif
560 ! uold = rho * vx
561 ! (M+dm)v_new= M v_old + dM v_new
562 #if NDIM==1
563 !       vx
564           uold(ind_cell,2)=uold(ind_cell,2)+momentum_help
565 #endif
566 #if NDIM>1
```

```
!       vx
          uold(ind_cell,2)=uold(ind_cell,2)+weightx(i)*momentum_help
!       vy
          uold(ind_cell,3)=uold(ind_cell,3)+weighty(i)*momentum_help
#endif
#if NDIM==3
          uold(ind_cell,4)=uold(ind_cell,4)+weightz(i)*momentum_help
#endif
#endif
!       total energy density
          uold(ind_cell,ndim+2)=uold(ind_cell,ndim+2)+de*weight(i)
!         old energydensity + driver energy density
!         don't subtract E-kin
!         uold(,ndim+2) contains e_int+e_kin
!          uold(ind_cell,ndim+2)=uold(ind_cell,ndim+2)+                  &
!&         weight(i)*max(0.0_dp,de-dm/uold(ind_cell,1)**2*0.5_dp*        &
!&         (uold(ind_cell,2)**2+uold(ind_cell,3)**2))
!       26Al
          uold(ind_cell,i26al)=uold(ind_cell,i26al)+d26Al*weight(i)
!       60Fe
          uold(ind_cell,i60fe)=uold(ind_cell,i60fe)+d60Fe*weight(i)
#if  defined (THII) || defined (TMAX) || defined (TMIN)
!       Compute T/mu in Kelvin
          Ekin_help=(uold(ind_cell,2)**2                                &
#if NDIM>1
&         +uold(ind_cell,3)**2                                          &
#endif
#if NDIM>2
&         +uold(ind_cell,4)**2                                          &
#endif
&         )*0.5_dp/uold(ind_cell,1)
          T_help=scale_T2*(gamma-1.0_dp)/uold(ind_cell,1)
!-------------------------------
#if DEBUG==3
!-------------------------------
!       check if kinetic energy is still smaller than total energy
          if(Ekin_help.gt.uold(ind_cell,ndim+2))then
            OLD_rho=uold(ind_cell,1)-dm*weight(i)
            OLD_Etot=uold(ind_cell,ndim+2)-de*weight(i)
            OLD_Ekin_help=Ekin_help*uold(ind_cell,1)/OLD_rho
            print*,"with wind: ekin,etot:",Ekin_help, uold(ind_cell,ndim+2)
            print*,"without wind: ekin,etot:",OLD_Ekin_help, OLD_Etot
            if(OLD_Ekin_help.gt.OLD_Etot)then
              uold(ind_cell,ndim+2)=Ekin_help+T_min_fix/T_help
            else
              uold(ind_cell,ndim+2)=OLD_Etot-OLD_Ekin_help+Ekin_help
            end if
          end if
!-------------------------------
#endif
#ifdef TMIN
!       Set Tmin
          if((uold(ind_cell,ndim+2)-Ekin_help).lt.T_min_fix/T_help)then
            uold(ind_cell,ndim+2)=Ekin_help+T_min_fix/T_help
          end if
```

```
622 #endif
623 #ifdef TMAX
624 !       Set Tmax
625           if((uold(ind_cell,ndim+2)-Ekin_help).gt.T_max/T_help)then
626             uold(ind_cell,ndim+2)=Ekin_help+T_max/T_help
627           end if
628 #endif
629 !       Set T(driver) to 10.000 K
630 #ifdef THII
631           if((uold(ind_cell,ndim+2)-Ekin_help).lt.T_HII/T_help)then
632             uold(ind_cell,ndim+2)=Ekin_help+T_HII/T_help
633           end if
634 #endif
635 #endif
636 #if defined (TMAX) || defined (TMIN)
637         else
638 !           Compute T/mu in Kelvin
639           Ekin_help=(uold(ind_cell,2)**2                                &
640 #if NDIM>1
641 &          +uold(ind_cell,3)**2                                         &
642 #endif
643 #if NDIM>2
644 &          +uold(ind_cell,4)**2                                         &
645 #endif
646 &          )*0.5_dp/uold(ind_cell,1)
647           T_help=scale_T2*(gamma-1.0_dp)/uold(ind_cell,1)
649
649 #if DEBUG==3
650 !-------------------------------
651 !       check if kinetic energy is still smaller than total energy
652         if(Ekin_help.gt.uold(ind_cell,ndim+2))then
653           print*,"without wind: ekin, etot:",Ekin_help, uold(ind_cell,ndim+2)
654           uold(ind_cell,ndim+2)=Ekin_help+T_min_fix/T_help
655         end if
656 !-------------------------------
657 #endif
658 #ifdef TMIN
659 !       Set Tmin
660           if((uold(ind_cell,ndim+2)-Ekin_help).lt.T_min_fix/T_help)then
661             uold(ind_cell,ndim+2)=Ekin_help+T_min_fix/T_help
662           end if
663 #endif
664 #ifdef TMAX
665 !       Set Tmax
666           if((uold(ind_cell,ndim+2)-Ekin_help).gt.T_max/T_help)then
667             uold(ind_cell,ndim+2)=Ekin_help+T_max/T_help
668           end if
669 #endif
670 #endif
671 #ifdef EKIN
672 #if DEBUG==3
673           if(uold(ind_cell,ndim+2).lt.Ekin_help)then
674             print*, "WARNING: negative pressure!!",uold(ind_cell,ndim+2),
                  Ekin_help
675             call print_xyz(ind,ilevel,igrid,ngrid,dx,i)
```

```
676           end if
677 #endif
678 #endif
679 !------------------------------------
680         end if
681 !------------------------------------
682       end do
683 #if DEBUG==2
684   driverTest = driverTest+ sum(weight(1:ngrid))*de
685 #endif
686     end do
687   end do
688 #if DEBUG==2
689   if(verbose_patches)write(*,115) driverTest, deltat,driverTest/deltat*(0.083
        e19_dp**3)*1.e-17_dp
690   if(verbose_patches)write(*,116) sum(uold(:,ndim+2))
691 !if(debug)print*, "sum of all weights = ", xx
692 !if(debug)print*, "pi*(r/dx)*4/3 = ", pi*(rdriver_scaled/dx)**3/0.75
693 !if(debug)print*, "pi*(r/dx)**2 = ", pi*(rdriver_scaled/dx)**2
694 !if(debug)print*,rdriver_scaled, dx,rdriver_scaled/dx
695 #endif
696   ! Update boundaries
697   do ivar=1,nvar
698      call make_virtual_fine_dp(uold(1,ivar),ilevel)
699   end do
700
701 111 format('   Entering wind_fine for level ',I2)
702 112 format('   Number of cells along driver radius: ', I2)
703 113 format('   New26Al: ', G14.5E4)
704 114 format('   dE: ', G14.5E4)
705 115 format('   Edriver: ', G14.5E4, "dt", G14.5E4, "dE/dt [erg/s]", G14.5E4)
706 116 format('   sum E: ', G14.5E4)
707 117 format('   driver volume [1e57 cm3] : ', G14.5E4)
708 end subroutine wind_fine
```

Listing C.8: De-allocation of feedback arrays: update_time.f90

```
180 subroutine clean_stop
181   use amr_parameters, only: n_stars
182   use amr_commons
183   use driver
184   implicit none
185 #ifndef WITHOUTMPI
186   include 'mpif.h'
187 #endif
188   integer::info
189 #ifndef WITHOUTMPI
190   call MPI_FINALIZE(info)
191 #endif
192   if (n_stars.gt.0.0) then
193    if(.NOT.ifgeneva)then
194     call remove_driver
195     call remove_sn
196    end if
197    call deallocate_driver_mask
198   end if
```

```
199   stop
200 end subroutine clean_stop
```

Listing C.9: Control refinement in the feedback region: flag_utils.f90

```
416 #ifdef MAXDRIVERGRID
417               call geometry_refine(xx,ind_cell,ok,ngrid,ilevel,dx_loc)
418 #else
419               call geometry_refine(xx,ind_cell,ok,ngrid,ilevel)
420 #endif
515 #if defined POISSON && ( POISSON > 0 )
516   end if
517 #endif
518 end subroutine poisson_refine
519 !################################################################
520 !################################################################
521 !################################################################
522 !################################################################
523 #ifdef MAXDRIVERGRID
524 subroutine geometry_refine(xx,ind_cell,ok,ncell,ilevel,dxloc)
525   use amr_parameters, only: r_driver,max_driver_grid
526 #else
527 subroutine geometry_refine(xx,ind_cell,ok,ncell,ilevel)
528 #endif
537 #ifdef MAXDRIVERGRID
538   real(dp)::dxloc
539 #endif
568         if(er<10)then
569            r=(xn**er+yn**er+zn**er)**(1.0/er)
570         else
571 #ifdef IGNOREX
572            !ignore xn to refine only  close to x axis
573            r=max(yn,zn) !don't refine outside this region
574 #else
575            r=max(xn,yn,zn)
576 #endif
577         end if
578 #ifdef MAXDRIVERGRID
579         !!refine at the cloud surface to highest level
580         !ok(i)=ok(i).or.((r < 1.0 + dxloc/rr).and.(r > 1.0 - dxloc/rr)) !cloud
              surface
581         !driver surface
582         if(ilevel.le.max_driver_grid)then
583           ok(i)=ok(i).or.(r < (r_driver + dxloc)/rr)
584         !else
585         ! !don't refine driver region beyond max_driver grid
586         ! if (r < (r_driver + dxloc)/rr) ok(i)=.false.
587         end if
588 #endif
589         ok(i)=ok(i).and.(r < 1.0) !Don't refine outside the region. Only refine
              inside if also another refinement criterium is met.
```

Listing C.10: Control the refinement in the feedback region: hydro_flag.f90

```
140 #ifdef MAXDRIVERGRID
141            call geometry_refine(xx,ind_cell,ok,ngrid,ilevel,dx_loc)
```

```
142 #else
143             call geometry_refine(xx,ind_cell,ok,ngrid,ilevel)
144 #endif
```

Listing C.11: Passive scalars and initial conditions for ^{26}Al and ^{60}Fe: hydro_parameters.f90

```
10  #ifndef NVAR
11    integer,parameter::nvar=ndim+2+nener+2 ! add two passive scalars for 26Al and 60
        Fe
12  #else
13    integer,parameter::nvar=NVAR+2 ! add two passive scalars for 26Al and 60Fe
14  #endif
58    real(dp),dimension(1:MAXREGION)::al_region=0.
74    real(dp)::larget=0.1_dp ! largest time step in coarsest grid .. in each smaller
        grid factor 0.5 smaller
87    integer::i26al=ndim+3
88    integer::i60fe=ndim+4
89    integer::iloss=ndim+5
```

Listing C.12: Initial conditions: SPH data, ^{26}Al data, triangles: init_flow_fine.f90

```
43  #ifdef SPH
44    use sph, ONLY: read_sph, erase_sph
45  #endif
52
52  #ifdef SPH
53    integer::k
54  #endif
55    integer::i,icell,igrid,ncache,iskip,ngrid,ilun
79  #ifdef SPH
80    logical::ifsph=.false.
81  #endif
429 #ifdef SPH
430     do k=1,nregion
431       if(region_type(k).eq.'sph')then
432         ifsph=.true.
433         call read_sph(length_x(k),x_center(k),u_region(k)                &
434 #if NDIM>1
435 &                        ,length_y(k),y_center(k),v_region(k)            &
436 #endif
437 #if NDIM>2
438 &                        ,length_z(k),z_center(k),w_region(k)            &
439 #endif
440 &                        )
441       end if
442     end do
443 #endif
482 #ifdef SPH
483     call erase_sph
484 #endif
493 !################################################################
494 subroutine region_condinit(x,q,dx,nn)
495   use amr_parameters
496   use hydro_parameters
497 #ifdef SPH
498   use sph, ONLY: interpolate_sph
```

```
499 #endif
500   use random
501   implicit none
502   integer ::nn
503   real(dp)::dx
504   real(dp),dimension(1:nvector,1:nvar)::q
505   real(dp),dimension(1:nvector,1:ndim)::x
507
507   integer::i,ix,ivar,k,n_weight
508   real(dp)::vol,r,xn,yn,zn,en
509   real(dp)::ro,xno,yno,zno,weight,scale
510   real(kind=8)::help_k8
511   real(dp)::help1,help2,help3 !< random coordinates [0:1]
512   real(dp)::help4,help5
513   real(dp)::scale_nH,scale_T2,scale_l,scale_d,scale_t,scale_v
514   real(dp),parameter::T_minimum=10._dp
516
516   integer,dimension( IRandNumSize ) :: &
517 & localseed = (/ 3281, 4041, 595, 2376 /)
518   integer:: randgridsize = 10000 !< number of points in random subgrid
519 ! regions are overwritten by regions with higher "region index"
520 ! only "point" regions are an overlay
521 ! and (now) edges of "square" regions apply weights if the
522 ! cells are partly inside (parameters of regions with
523 ! lower "region index" are used)
524   ! Set some (tiny) default values in case n_region=0
525   q(1:nn,1)=smallr
526   q(1:nn,2)=0.0d0
527 #if NDIM>1
528   q(1:nn,3)=0.0d0
529 #endif
530 #if NDIM>2
531   q(1:nn,4)=0.0d0
532 #endif
533   q(1:nn,ndim+2)=smallr*smallc**2/gamma
534 #if NVAR > NDIM + 2
535   do ivar=ndim+3,nvar
536      q(1:nn,ivar)=0.0d0
537   end do
538 #endif
540
540   ! Loop over initial conditions regions
541   do k=1,nregion
542      ! For "alu" regions only:
543      if(region_type(k) .eq. 'alu')then ! region square,en=10
544         print*,"reading␣alu,region=",k
545         region_type(k) = "square"
546         exp_region(k)  = 10
547         u_region(k)    =0.0_dp
548 #if NDIM>1
549         v_region(k)    =0.0_dp
550 #endif
551 #if NDIM>2
552         w_region(k)    =0.0_dp
553 #endif
```

```
554           var_region(k,i26al-ndim-2) = al_region(k)
555        end if
556        ! For "driver" regions only:
557        if(region_type(k) .eq. 'driver')then ! region square,en=10
558           print*,"reading alu,region=",k
559           region_type(k) = "square"
560           exp_region(k)  = 2
561           u_region(k)    =0.0_dp
562 #if NDIM>1
563           v_region(k)    =0.0_dp
564 #endif
565 #if NDIM>2
566           w_region(k)    =0.0_dp
567 #endif
568           var_region(k,i26al-ndim-2) = al_region(k)
569        end if
570        ! For "square" regions only:
571        if (region_type(k) .eq. 'square') then
572           ! Overlap of regions is not checked.
573           ! the second region in the inputfile
574           ! will overwrite the first etc...
575           ! Exponent of choosen norm
576           en=exp_region(k)
577           if(en<10)then
578              ! Conversion factor from user units to cgs units
579              call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
580           end if
581           do i=1,nn
582              ! Compute position in normalized coordinates
583              xn=0.0d0; yn=0.0d0; zn=0.0d0
584 !new:
585 !ro ... distance between the center of the region and the innermost
586 !       corner of the cell if the ro computed this way is smaller than 1,
587 !       but r is larger than 1, the cell lies partly inside the region
588 !       for cells partly inside the region weights are computed
589 !r  ... distance between the center of the region and the outermost
590 !       corner if the r computed this way is larger than 1,
591 !       the cell lies fully outside the region
592 !
593 !old:
594 !r ... distance between the center of the region and the center of
595 !      the cell. cells are considdered as either completely
596 !      inside or completely outside the region
597 !
598              ! weights for cells partly inside circular region
599              ! normalize to r=1 using a factor 2
600              ! because length_[xyz] read in is the diameter, not the radius
601              ! => xno = (distance to region center + dx/2 )/ region radius
602              scale=2.0d0*dx/length_x(k)
603              xno=(2.0d0*abs(x(i,1)-x_center(k))+dx)/length_x(k)
604 !            xn=(2.0d0*abs(x(i,1)-x_center(k))-dx)/length_x(k)
605              xn=xno-scale
606              if(xn<0.d0.and.xno<0.d0)then
607                 help1=abs(xn)
608                 xn=abs(xno)
```

```
609                xno=help1
610             end if
611 #if NDIM>1
612             yno=(2.0d0*abs(x(i,2)-y_center(k))+dx)/length_y(k)
613 !           yn=(2.0d0*abs(x(i,2)-y_center(k))-dx)/length_y(k)
614             yn=yno-2.0d0*dx/length_y(k)
615             if(yn<0.d0.and.yno<0.d0)then
616                help1=abs(yn)
617                yn=abs(yno)
618                yno=help1
619             end if
620 #endif
621 #if NDIM>2
622             zno=(2.0d0*abs(x(i,3)-z_center(k))+dx)/length_z(k)
623 !           zn=(2.0d0*abs(x(i,3)-z_center(k))-dx)/length_z(k)
624             zn=zno-2.0d0*dx/length_z(k)
625             if(zn<0.d0.and.zno<0.d0)then
626                help1=abs(zn)
627                zn=abs(zno)
628                zno=help1
629             end if
630 #endif
631             ! Compute cell "radius" relative to region center
632             if(en<10)then
633 !new:
634 !ro ... distance between the center of the region
635 !       and the innermost corner of the cell
636 !       for cells partly inside the regiong weights are computed
637 !r  ... distance between the center of the region
638 !       and the outermost corner
639 !
640 !old:
641 !r ... distance between the center of the region
642 !      and the center of the cell
643 !      cells are considdered as either completely
644 !      inside or completely outside the region
645                ro=(xn**en+yn**en+zn**en)**(1.0/en)
646                r =(xno**en+yno**en+zno**en)**(1.0/en)
647             else
648                r=max(xn,yn,zn)
649                ro=1.1d0
650             end if
651             ! If cell lies within region,
652             ! REPLACE primitive variables by region values
653 #ifdef RANDZELLEN
654             if(r.lt.1.0)then
655 #else
656 !           if(ro.le.1.0)then
657             if(((en<10).and.(ro.le.1.0)).or.                                &
658 &              ((en.ge.10).and.(r.le.1.0)))then
659 #endif
660                if(en<10)then
661 ! r>ro; 1/r^2 undefined @ r=0
662                   q(i,1)=d_region(k)
663 ! p/rho = T / scale_T2
```

```
664                     if(p_region(k).lt.q(i,1)*T_minimum/scale_T2)then
665                        print*, "init flow fine: ",T_minimum
666                        print*, "init flow fine: ",p_region(k),              &
667 &                              q(i,1)*T_minimum/scale_T2
668                     end if
669                     q(i,ndim+2)=max(p_region(k),q(i,1)*T_minimum/scale_T2)
670                  else
671                     q(i,1)=d_region(k)
672                     q(i,ndim+2)=p_region(k)
673                  end if
674                  q(i,2)=u_region(k)
675 #if NDIM>1
676                  q(i,3)=v_region(k)
677 #endif
678 #if NDIM>2
679                  q(i,4)=w_region(k)
680 #endif
681 #if NENER>0
682                  do ivar=1,nener
683                     q(i,ndim+2+ivar)=prad_region(k,ivar)
684                  enddo
685 #endif
686 #if NVAR>NDIM+2+NENER
687 !                q(i,i26al)=al_region(k)
688                  do ivar=ndim+3+nener,nvar
689                     q(i,ivar)=var_region(k,ivar-ndim-2-nener)
690                  end do
691 #endif
692 #ifdef RANDZELLEN
693               else if((ro.le.1.0).and.(r.gt.1.0))then
694                  ! weights for cells partly inside circular region)
695                  n_weight=0
696                  help1=0._dp
697                  help2=0._dp
698                  help3=0._dp
699                  do ix=1,randgridsize
700                     call ranf(localseed,help_k8)
701 ! => xno = (distance to region center + dx/2 )/ region radius
702 ! radius_x = length_x(k) / 2._dp
703                     help1=xn+dble(help_k8)*2._dp*dx/length_x(k)
704 #if NDIM>1
705                     call ranf(localseed,help_k8)
706                     help2=yn+dble(help_k8)**2._dp*dx/length_y(k)
707 #endif
708 #if NDIM>2
709                     call ranf(localseed,help_k8)
710                     help3=zn+dble(help_k8)*2._dp*dx/length_z(k)
711 #endif
712                     if((help1**2+help2**2+help3**2).lt.1.0)then
713                        n_weight=n_weight+1
714                     end if
715                  end do
716                  weight = dble(n_weight)/dble(randgridsize)
717 #if DEBUG==2
718 !                print*,"weight",weight
```

```
719 !                 if(en<10)then
720                   if (weight.gt.1.0)then
721                      if(verbose_patches)print*,"weight=",weight
722                      weight=1.0_dp
723                   end if
724                   if (weight.le.0.0)then
725                      if(verbose_patches)print*,"weight=",weight
726                      weight=0.0_dp
727                   end if
728 #endif
729                   q(i,1)=q(i,1)*(1.d0-weight)+weight*d_region(k)
730                   if(p_region(k).lt.q(i,1)*T_minimum/scale_T2)then
731                      print*, "init flow fine: ",T_minimum
732                      print*, "init flow fine: ",p_region(k),                 &
733 &                           q(i,1)*T_minimum/scale_T2
734                   end if
735                   q(i,ndim+2)=q(i,ndim+2)*(1.d0-weight)+weight*              &
736 &                  max(p_region(k),q(i,1)*T_minimum/scale_T2/(gamma-1.0))
737 !else cannot enter this part of the loop ro=1.1>1
738 !  q(i,1)=q(i,1)*(1.d0-weight)+weight*d_region(k)
739 !  q(i,ndim+2)=q(i,ndim+2)*(1.d0-weight)+weight*p_region(k)
740 !end if
741                   q(i,2)=q(i,2)*(1.d0-weight)+weight*u_region(k)
742 #if NDIM>1
743                   q(i,3)=q(i,3)*(1.d0-weight)+weight*v_region(k)
744 #endif
745 #if NDIM>2
746                   q(i,4)=q(i,4)*(1.d0-weight)+weight*w_region(k)
747 #endif
748 #endif
749 #if NENER>0
750                   do ivar=1,nener
751                      q(i,ndim+2+ivar)=q(i,ndim+2+ivar)*(1.d0-weight)        &
752 &                                    +weight*prad_region(k,ivar)
753                   enddo
754 #endif
755 #if NVAR>NDIM+2+NENER
756                   do ivar=ndim+3+nener,nvar
757                      q(i,ivar)=q(i,ivar)*(1.d0-weight)                       &
758 &                              +weight*var_region(k,ivar-ndim-2-nener)
759                   end do
760 #endif
761                end if
762             end do
763          end if
764          ! For "point" regions only:
765          if(region_type(k) .eq. 'point')then
766             ! Volume elements
767             vol=dx**ndim
768             ! Compute CIC weights relative to region center
769             do i=1,nn
770                xn=1.0; yn=1.0; zn=1.0
771                xn=max(1.0-abs(x(i,1)-x_center(k))/dx,0.0_dp)
772 #if NDIM>1
773                yn=max(1.0-abs(x(i,2)-y_center(k))/dx,0.0_dp)
```

```
774 #endif
775 #if NDIM>2
776             zn=max(1.0-abs(x(i,3)-z_center(k))/dx,0.0_dp)
777 #endif
778             r=xn*yn*zn
779             ! If cell lies within CIC cloud,
780             ! ADD to primitive variables the region values
781             q(i,1)=q(i,1)+d_region(k)*r/vol
782             q(i,2)=q(i,2)+u_region(k)*r
783 #if NDIM>1
784             q(i,3)=q(i,3)+v_region(k)*r
785 #endif
786 #if NDIM>2
787             q(i,4)=q(i,4)+w_region(k)*r
788 #endif
789             q(i,ndim+2)=q(i,ndim+2)+p_region(k)*r/vol
790 #if NENER>0
791             do ivar=1,nener
792                q(i,ndim+2+ivar)=q(i,ndim+2+ivar)+prad_region(k,ivar)*r/vol
793             enddo
794 #endif
795 #if NVAR>NDIM+2+NENER
796             do ivar=ndim+3+nener,nvar
797                q(i,ivar)=var_region(k,ivar-ndim-2-nener)
798             end do
799 #endif
800          end do
801       end if
802 #ifdef SPH
803 ! For "sph" regions only:
804       if(region_type(k) .eq. 'sph')then
805          call interpolate_sph(x,q,dx,nn,d_region(k),p_region(k))
806       end if
807 #endif
808 ! For "triangle" regions only:
809       if(region_type(k) .eq. 'triangle')then
810          do i=1,nn
811             xn=0.0d0; yn=0.0d0; zn=0.0d0
812             xno=(2.0d0*abs(x(i,1)-x_center(k))+dx)/length_x(k)
813             xn=xno-2.0d0*dx/length_x(k)
814 #if NDIM>1
815             yno=(2.0d0*abs(x(i,2)-y_center(k))+dx)/length_y(k)
816             yn=yno-2.0d0*dx/length_y(k)
817 #endif
818 #if NDIM>2
819             zno=(2.0d0*abs(x(i,3)-z_center(k))+dx)/length_z(k)
820             zn=zno-2.0d0*dx/length_z(k)
821 #endif
822 #if NDIM==1
823             if(xn.le.0.0d0)then
824 #endif
825 #if NDIM==2
826             if(yn.le.xn)then
827 #endif
828 #if NDIM==3
```

```
829             if(2.0d0*zn.le.xn+yn)then
830 #endif
831                q(i,1)=d_region(k)
832                q(i,2)=u_region(k)
833 #if NDIM>1
834                q(i,3)=v_region(k)
835 #endif
836 #if NDIM>2
837                q(i,4)=w_region(k)
838 #endif
839                q(i,ndim+2)=p_region(k)
840 #if NENER>0
841                do ivar=1,nener
842                   q(i,ndim+2+ivar)=prad_region(k,ivar)
843                enddo
844 #endif
845                q(i,i26al)=al_region(k)
846 !#if NVAR>NDIM+2+NENER
847 !                do ivar=ndim+3+nener,nvar
848 !                   q(i,ivar)=var_region(k,ivar-ndim-2-nener)
849 !                end do
850 !#endif
851             end if
852          end do
853       end if
854    end do
855    return
856 end subroutine region_condinit
```

Listing C.13: New module to read-in SPH data: sph.f90

```
1  #undef CLARE
2  #define JIM 1
3  !> \short read + interpolate sph initial conditions
4  !-------------------------------------------------------------------------
5  #ifdef JIM
6  !> \version 1.2 Jim's initial conditions
7  #else
8  !> \version 1.2 Clare's initial conditions
9  #endif
10 !> \author Katharina M. Fierlinger
11 !> \date last modification 26.03.2011
12 !-------------------------------------------------------------------------
13 !> \details  PURPOSE:
14 !> \n file_sph ... name of sph initial conditions file
15 !> \n read the sph initial conditions from a file called "file_sph" in the local
       dir
16 !-------------------------------------------------------------------------
17 !> \n sph file contents:
18 #ifdef JIM
19 !> \n column 1: x (0.1 pc)
20 !> \n column 2: y (0.1 pc)
21 !> \n column 3: z (0.1 pc)
22 !> \n column 4: vx (2.0748E+04 cm/s)
23 !> \n column 5: vy (2.0748E+04 cm/s)
24 !> \n column 6: vz (2.0748E+04 cm/s)
```

```
25 !> \n column 7: density (6.7746E-20 g/cm^3)
26 !> \n column 8: temperature (K)
27 !> \n column 9: smoothing length (0.1 pc)
28 #else
29 !> \n column 1: x (kpc)
30 !> \n column 2: y (kpc)
31 !> \n column 3: z (kpc)
32 !> \n column 4: vx (km/s)
33 !> \n column 5: vy (km/s)
34 !> \n column 6: vz (km/s)
35 !> \n column 7: density (10^-24 cm-3)
36 !> \n column 8: temperature (K)
37 !> \n column 9: smoothing length (kpc)
38 #endif
39 !-----------------------------------------------------------------------
40 module sph
41   use amr_parameters, only : dp, file_sph
42   implicit none
43   save ! retain the value of the variables from one call to the next
44   integer, parameter :: i9 = selected_int_kind(r=9) !< integer type definition
45   integer(i9) :: nsph   = 0_i9 !< number of sph particles where part of the
       smoothing length is inside the box
46   real(dp), dimension(:), allocatable, private :: x_sph         !< array
       containing sph x coordinate [code units]
47   real(dp), dimension(:), allocatable, private :: vx_sph        !< array
       containing sph x velocity   [code units]
48 #if NDIM>1
49   real(dp), dimension(:), allocatable, private :: y_sph         !< array
       containing sph y coordinate [code units]
50   real(dp), dimension(:), allocatable, private :: vy_sph        !< array
       containing sph y velocity   [code units]
51 #endif
52 #if NDIM>2
53   real(dp), dimension(:), allocatable, private :: z_sph         !< array
       containing sph z coordinate [code units]
54   real(dp), dimension(:), allocatable, private :: vz_sph        !< array
       containing sph z velocity   [code units]
55 #endif
56   real(dp), dimension(:), allocatable, private :: T_sph         !< array
       containing sph temperature  [code units]
57   real(dp), dimension(:), allocatable, private :: rho_sph       !< array
       containing sph density      [code units]
58   real(dp), dimension(:), allocatable, private :: smoothing_sph !< array
       containing sph smoothing length [code units]
59 contains
60 ! subroutine read_sph
61 !> \short read sph initial conditions
62 !-----------------------------------------------------------------------
63 !> \version 1.0
64 !> \author Katharina M. Fierlinger
65 !> \date last modification 26.03.2011
66 !-----------------------------------------------------------------------
67 !> \details  PURPOSE:
68 !> \n file_sph ... name of sph initial conditions file
69 !> \n read the sph initial conditions from a file called "file_sph" in the local
```

```
    dir
!------------------------------------------------------------------------
!> \n sph file contents:
#ifdef JIM
!> \n column 1: x (0.1 pc)
!> \n column 2: y (0.1 pc)
!> \n column 3: z (0.1 pc)
!> \n column 4: vx (2.0748E+04 cm/s)
!> \n column 5: vy (2.0748E+04 cm/s)
!> \n column 6: vz (2.0748E+04 cm/s)
!> \n column 7: density (6.7746E-20 g/cm^3)
!> \n column 8: temperature (K)
!> \n column 9: smoothing length (0.1 pc)
#else
!> \n column 1: x (kpc)
!> \n column 2: y (kpc)
!> \n column 3: z (kpc)
!> \n column 4: vx (km/s)
!> \n column 5: vy (km/s)
!> \n column 6: vz (km/s)
!> \n column 7: density (10^-24 cm-3)
!> \n column 8: temperature (K)
!> \n column 9: smoothing length (kpc)
#endif
!------------------------------------------------------------------------
! subroutine read_sph_claresmooth(x_length,x_center,vx_center           &
  subroutine read_sph(x_length,x_center,vx_center                       &
#if NDIM>1
&                    ,y_length,y_center,vy_center                       &
#endif
#if NDIM>2
&                    ,z_length,z_center,vz_center                       &
#endif
&                   )
    implicit none
!> region size, location and bulk velocity
    real(dp), intent(in)::x_length  !< region width
    real(dp), intent(in)::x_center  !< region center
    real(dp), intent(in)::vx_center !< bulk speed
#if NDIM>1
    real(dp), intent(in)::y_length  !< region width
    real(dp), intent(in)::y_center  !< region center
    real(dp), intent(in)::vy_center !< bulk speed
#endif
#if NDIM>2
    real(dp), intent(in)::z_length  !< region width
    real(dp), intent(in)::z_center  !< region center
    real(dp), intent(in)::vz_center !< bulk speed
#endif
!> counters + error handling
    integer(i9) :: nlines = 0_i9 !< number of lines read from driver file
    integer(i9) :: ii     = 1_i9 !< for do loop
    integer(i9) :: ifEOF  = 0_i9 !< checks when the end of the file is reached
!> units + conversion factors
    real(dp) :: scale_dist, scale_dens, scale_vel
```

```
124     real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
            factors between cgs and user units (subroutine units in units.f90)
125     real(dp) :: col1,col2,col3,col4,col5,col6,col7,col8,col9 !< reads data from
            the 9 columns in the input file
126 #ifdef JIM
127     real(dp), parameter :: DeciPcToCm = 3.08568025e17_dp !< convert 0.1 pc to cm;
            1 pc =  3.08568025e18 cm
128     real(dp), parameter :: VelocityToCgs = 2.0748e4_dp   !< convert to cm/s
129     real(dp), parameter :: DensToCgs = 6.7746e-20_dp     !< convert to g/cm3
130 #else
131     real(dp), parameter :: KpcToCm = 3.08568025e21_dp !< convert kpc to cm; 1 kpc
            =  3.08568025e21 cm
132     real(dp), parameter :: KmToCm = 1e5_dp            !< convert km to cm
133     real(dp), parameter :: DensToCgs = 1e-24_dp       !< convert 10^-24 g/cm3 to g
            /cm3
134 #endif
135 !> local variables: test if sph particle is close enough to the region to be
        relevant
136     real(dp) :: x_help, x_max, smoothing_length
137 #if NDIM>1
138     real(dp) :: y_help, y_max
139 #endif
140 #if NDIM>2
141     real(dp) :: z_help, z_max
142 #endif
144
144     call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
145 #ifdef JIM
146     scale_dist=DeciPcToCm/scale_l
147     scale_vel=VelocityToCgs/scale_v
148 #else
149     scale_dist=KpcToCm/scale_l
150     scale_vel=KmToCm/scale_v
151 #endif
152     scale_dens=DensToCgs/scale_d
153     open (1,file=TRIM(file_sph),form='formatted')
154     print*,"Reading␣sph␣data␣from␣>>", TRIM(file_sph), "<<␣."
155     nsph    = 0_i9
156     nlines = 0_i9
157     ifEOF   = 0_i9
158     x_max = 0.5_dp*x_length
159 #if NDIM>1
160     y_max = 0.5_dp*y_length
161 #endif
162 #if NDIM>2
163     z_max = 0.5_dp*z_length
164 #endif
165     do
166        read(1,*,IOSTAT=ifEOF) col1,col2,col3,col4,col5,col6,col7,col8,col9
167        smoothing_length=col9*scale_dist
168        x_help=abs(col1*scale_dist+x_center-x_max)-smoothing_length
169 #if NDIM>1
170        y_help=abs(col2*scale_dist+y_center-y_max)-smoothing_length
171 #endif
172 #if NDIM>2
```

```
173         z_help=abs(col3*scale_dist+z_center-z_max)-smoothing_length
174 #endif
175         !help ... box contains values in [-xyz_max:xyz_max]
176         if (ifEOF.lt.0_i9) then
177            exit ! eof is reached, jump out of the do-loop
178         end if
179         nlines=nlines+1
180         if (x_help.le.x_max) then
181 !       if (abs(x_help).le.x_length) then
182 #if NDIM>1
183            if (y_help.le.y_max) then
184 !          if (abs(y_help).le.y_length) then
185 #endif
186 #if NDIM>2
187               if (z_help.le.z_max) then
188 !             if (abs(z_help).le.z_length) then
189 #endif
190                  nsph=nsph+1
191 #if NDIM>2
192 !        else
193 !          print*, z_help, z_length
194               end if
195 #endif
196 #if NDIM>1
197 !        else
198 !          print*, y_help, y_length
199            end if
200 #endif
201 !        else
202 !          print*, x_help, x_length
203         end if
204       end do
205       print*,"found ",nlines, " lines in sph file"
206       print*,"selected ",nsph, " particles"
207       rewind(1)
208       call allocate_sph
209       ifEOF = 0_i9
210       nsph  = 0_i9
211       do ii=1,nlines
212          read(1,*,IOSTAT=ifEOF) col1,col2,col3,col4,col5,col6,col7,col8,col9
213          smoothing_length=col9*scale_dist
214          x_help=abs(col1*scale_dist+x_center-x_max)-smoothing_length
215 #if NDIM>1
216          y_help=abs(col2*scale_dist+y_center-y_max)-smoothing_length
217 #endif
218 #if NDIM>2
219          z_help=abs(col3*scale_dist+z_center-z_max)-smoothing_length
220 #endif
221          if (ifEOF.lt.0_i9) then
222             exit ! eof is reached, jump out of the do-loop
223          end if
224          nlines=nlines+1
225          if (x_help.le.x_max) then
226 !        if (abs(x_help).le.x_length) then
227 #if NDIM>1
```

```
228           if (y_help.le.y_max) then
229 !         if (abs(y_help).le.y_length) then
230 #endif
231 #if NDIM>2
232              if (z_help.le.z_max) then
233 !            if (abs(z_help).le.z_length) then
234 #endif
235                 nsph=nsph+1
236                 smoothing_sph(nsph)=col9*scale_dist
237                 x_sph(nsph)=col1*scale_dist+x_center
238                 vx_sph(nsph)=(col4-vx_center)*scale_vel
239                 T_sph(nsph)=col8/scale_T2
240                 rho_sph(nsph)=col7*scale_dens
241 #if NDIM>1
242                 y_sph(nsph)=col2*scale_dist+y_center
243                 vy_sph(nsph)=(col5-vy_center)*scale_vel
244 #endif
245 #if NDIM>2
246                 z_sph(nsph)=col3*scale_dist+z_center
247                 vz_sph(nsph)=(col6-vz_center)*scale_vel
248 #endif
249 #if NDIM>2
250              end if
251 #endif
252 #if NDIM>1
253           end if
254 #endif
255        end if
256     end do
257     close(1)
258     print*,"selected ",nsph, " particles"
259   end subroutine read_sph
260 ! end subroutine read_sph_claresmooth
261 ! subroutine read_sph_anysmooth
262 !> \short read sph initial conditions; user defined smoothing length
263 !------------------------------------------------------------------------------
264 !> \version 1.0
265 !> \author Katharina M. Fierlinger
266 !> \date last modification 26.03.2011
267 !------------------------------------------------------------------------------
268 !> \details  PURPOSE:
269 !> \n file_sph ... name of sph initial conditions file
270 !> \n read the sph initial conditions from a file called "file_sph" in the local
        dir
271 !------------------------------------------------------------------------------
272 !> \n sph file contents:
273 #ifdef JIM
274 !> \n column 1: x (0.1 pc)
275 !> \n column 2: y (0.1 pc)
276 !> \n column 3: z (0.1 pc)
277 !> \n column 4: vx (2.0748E+04 cm/s)
278 !> \n column 5: vy (2.0748E+04 cm/s)
279 !> \n column 6: vz (2.0748E+04 cm/s)
280 !> \n column 7: density (6.7746E-20 g/cm^3)
281 !> \n column 8: temperature (K)
```

```
!> \n column 9: smoothing length (0.1 pc)
#else
!> \n column 1: x (kpc)
!> \n column 2: y (kpc)
!> \n column 3: z (kpc)
!> \n column 4: vx (km/s)
!> \n column 5: vy (km/s)
!> \n column 6: vz (km/s)
!> \n column 7: density (10^-24 cm-3)
!> \n column 8: temperature (K)
!> \n column 9: smoothing length (kpc)
#endif
!-------------------------------------------------------------------------
! subroutine read_sph(x_length,x_center,vx_center                          &
  subroutine read_sph_anysmooth(x_length,x_center,vx_center               &
#if NDIM>1
&                    ,y_length,y_center,vy_center                        &
#endif
#if NDIM>2
&                    ,z_length,z_center,vz_center                        &
#endif
&                    )
    implicit none
!> region size, location and bulk velocity
    real(dp), intent(in)::x_length  !< region width
    real(dp), intent(in)::x_center  !< region center
    real(dp), intent(in)::vx_center !< bulk speed
#if NDIM>1
    real(dp), intent(in)::y_length  !< region width
    real(dp), intent(in)::y_center  !< region center
    real(dp), intent(in)::vy_center !< bulk speed
#endif
#if NDIM>2
    real(dp), intent(in)::z_length  !< region width
    real(dp), intent(in)::z_center  !< region center
    real(dp), intent(in)::vz_center !< bulk speed
#endif
!> counters + error handling
    integer(i9) :: nlines = 0_i9 !< number of lines read from driver file
    integer(i9) :: ii     = 1_i9 !< for do loop
    integer(i9) :: ifEOF  = 0_i9 !< checks when the end of the file is reached
!> units + conversion factors
    real(dp) :: scale_dist, scale_dens, scale_vel
    real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
       factors between cgs and user units (subroutine units in units.f90)
    real(dp) :: col1,col2,col3,col4,col5,col6,col7,col8,col9 !< reads data from
       the 8 columns in the input file
#ifdef JIM
    real(dp), parameter :: DeciPcToCm = 3.08568025e17_dp !< convert 0.1 pc to cm;
       1 pc =  3.08568025e18 cm
    real(dp), parameter :: VelocityToCgs = 2.0748e4_dp   !< convert to cm/s
    real(dp), parameter :: DensToCgs = 6.7746e-20_dp     !< convert to g/cm3
#else
    real(dp), parameter :: KpcToCm = 3.08568025e21_dp !< convert kpc to cm; 1 kpc
       =  3.08568025e21 cm
```

```
333     real(dp), parameter :: KmToCm = 1e5_dp              !< convert km to cm
334     real(dp), parameter :: DensToCgs = 1e-24_dp         !< convert 10^-24 g/cm3 to g
            /cm3
335 #endif
336 !> local variables: test if sph particle is close enough to the region to be
        relevant
337     real(dp) :: x_help, smoothing_length,x_max
338 #if NDIM>1
339     real(dp) :: y_help,y_max
340 #endif
341 #if NDIM>2
342     real(dp) :: z_help,z_max
343 #endif
345
345     call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
346 #ifdef JIM
347     scale_dist=DeciPcToCm/scale_l
348     scale_vel=VelocityToCgs/scale_v
349 #else
350     scale_dist=KpcToCm/scale_l
351     scale_vel=KmToCm/scale_v
352 #endif
353     scale_dens=DensToCgs/scale_d
354     open (1,file=TRIM(file_sph),form='formatted')
355     print*,"Reading␣sph␣data␣from␣>>", TRIM(file_sph), "<<␣."
356     nsph   = 0_i9
357     nlines = 0_i9
358     ifEOF  = 0_i9
359     x_max = 0.5_dp*x_length
360 #if NDIM>1
361     y_max = 0.5_dp*y_length
362 #endif
363 #if NDIM>2
364     z_max = 0.5_dp*z_length
365 #endif
366     do ! count number of lines inside the file
367        read(1,*,IOSTAT=ifEOF) col1,col2,col3,col4,col5,col6,col7,col8,col9
368        if (ifEOF.lt.0_i9) then
369           exit ! eof is reached, jump out of the do-loop
370        end if
371        nlines=nlines+1
372     end do
373     nsph=nlines
374     print*,"found␣",nlines, "␣lines␣in␣sph␣file"
375     print*,"selected␣",nsph, "␣particles"
376     rewind(1)
377     call allocate_sph
378     ifEOF = 0_i9
379     nsph  = 0_i9
380     do ii=1,nlines
381        read(1,*,IOSTAT=ifEOF) col1,col2,col3,col4,col5,col6,col7,col8,col9
382        smoothing_length=col9*scale_dist
383        x_help=abs(col1*scale_dist+x_center-x_max)-smoothing_length
384 #if NDIM>1
385        y_help=abs(col2*scale_dist+y_center-y_max)-smoothing_length
```

```
386 #endif
387 #if NDIM>2
388         z_help=abs(col3*scale_dist+z_center-z_max)-smoothing_length
389 #endif
390         if (ifEOF.lt.0_i9) then
391            exit ! eof is reached, jump out of the do-loop
392         end if
393         nlines=nlines+1
394         nsph=nsph+1
395         smoothing_sph(nsph)=col9*scale_dist
396         x_sph(nsph)=col1*scale_dist+x_center
397         vx_sph(nsph)=(col4-vx_center)*scale_vel
398         T_sph(nsph)=col8/scale_T2
399         rho_sph(nsph)=col7*scale_dens
400 #if NDIM>1
401         y_sph(nsph)=col2*scale_dist+y_center
402         vy_sph(nsph)=(col5-vy_center)*scale_vel
403 #endif
404 #if NDIM>2
405         z_sph(nsph)=col3*scale_dist+z_center
406         vz_sph(nsph)=(col6-vz_center)*scale_vel
407 #endif
408     end do
409     close(1)
410     print*,"selected ",nsph, " particles"
411     print*,"calling calc_smoothing_length"
412 #if NDIM==1
413     call calc_smoothing_length(x_max,0.5_dp,0.5_dp)
414 #endif
415 #if NDIM==2
416     call calc_smoothing_length(x_max,y_max,0.5_dp)
417 #endif
418 #if NDIM==3
419     call calc_smoothing_length(x_max,y_max,z_max)
420 #endif
421  !end subroutine read_sph
422   end subroutine read_sph_anysmooth
423 ! subroutine allocate_sph
424 !> \short allocates sph arrays
425 !-------------------------------------------------------------------------
426 !> \version 1.0
427 !> \author Katharina M. Fierlinger
428 !> \date last modification 26.03.2011
429 !-------------------------------------------------------------------------
430 !> \details  PURPOSE:
431 !> \n allocate sph arrays
432 !-------------------------------------------------------------------------
433   subroutine allocate_sph
434     implicit none
435 !> error handling
436     integer(i9) :: error_alloc !< checks if memory allocation works
437     allocate(x_sph(1:nsph),stat=error_alloc) ! in code-time-units
438     if (error_alloc /= 0) then
439        stop 'exiting: allocation of memory for x_sph did not work'
440     end if
```

```
441     allocate(vx_sph(1:nsph),stat=error_alloc) ! in code-time-units
442     if (error_alloc /= 0) then
443        stop 'exiting: allocation of memory for vx_sph did not work'
444     end if
445 #if NDIM>1
446     allocate(y_sph(1:nsph),stat=error_alloc) ! in code-time-units
447     if (error_alloc /= 0) then
448        stop 'exiting: allocation of memory for y_sph did not work'
449     end if
450     allocate(vy_sph(1:nsph),stat=error_alloc) ! in code-time-units
451     if (error_alloc /= 0) then
452        stop 'exiting: allocation of memory for vy_sph did not work'
453     end if
454 #endif
455 #if NDIM>2
456     allocate(z_sph(1:nsph),stat=error_alloc) ! in code-time-units
457     if (error_alloc /= 0) then
458        stop 'exiting: allocation of memory for z_sph did not work'
459     end if
460     allocate(vz_sph(1:nsph),stat=error_alloc) ! in code-time-units
461     if (error_alloc /= 0) then
462        stop 'exiting: allocation of memory for vz_sph did not work'
463     end if
464 #endif
465     allocate(T_sph(1:nsph),stat=error_alloc) ! in code-time-units
466     if (error_alloc /= 0) then
467        stop 'exiting: allocation of memory for T_sph did not work'
468     end if
469     allocate(rho_sph(1:nsph),stat=error_alloc) ! in code-time-units
470     if (error_alloc /= 0) then
471        stop 'exiting: allocation of memory for rho_sph did not work'
472     end if
473     allocate(smoothing_sph(1:nsph),stat=error_alloc) ! in code-time-units
474     if (error_alloc /= 0) then
475        stop 'exiting: allocation of memory for smoothing_sph did not work'
476     end if
477   end subroutine allocate_sph
478 ! subroutine erase
479 !> \short deallocates sph arrays
480 !------------------------------------------------------------------------------
481 !> \version 1.0
482 !> \author Katharina M. Fierlinger
483 !> \date last modification 26.03.2011
484 !------------------------------------------------------------------------------
485 !> \details  PURPOSE:
486 !> \n deallocate sph arrays
487 !------------------------------------------------------------------------------
488   subroutine erase_sph
489     implicit none
490     deallocate(x_sph)         !< array containing sph x coordinate [code units]
491     deallocate(vx_sph)        !< array containing sph x velocity   [code units]
492 #if NDIM>1
493     deallocate(y_sph)         !< array containing sph y coordinate [code units]
494     deallocate(vy_sph)        !< array containing sph y velocity   [code units]
495 #endif
```

```
496 #if NDIM>2
497     deallocate(z_sph)          !< array containing sph z coordinate [code units]
498     deallocate(vz_sph)         !< array containing sph z velocity   [code units]
499 #endif
500     deallocate(T_sph)          !< array containing sph temperature  [code units]
501     deallocate(rho_sph)        !< array containing sph density      [code units]
502     deallocate(smoothing_sph) !< array containing sph smoothing length [code units
        ]
503   end subroutine erase_sph
505
505 ! subroutine interpolate_sph
506 !> \short interpolates sph initial conditions onto grid
507 !------------------------------------------------------------------------------
508 !> \version 1.0
509 !> \author Katharina M. Fierlinger
510 !> \date last modification 26.03.2011
511 !------------------------------------------------------------------------------
512 !> \details  PURPOSE:
513 !> \n interpolate sph initial conditions onto grid
514 !> e.g. velocities are interpolated like
515 !> \f$v_j=\sum_i v_i \frac{m_i w\left(\left|\vec{r}_{i}-\vec{r}_{j}\right|,h_i\
    right)}{\rho_i}\f$
516 !> with
517 !> \f$h\f$ ... smoothing length
518 !> \f$\vec{r}_i, \vec{r}_j\f$ ... location of SPH particles
519 !> \f$m_i\f$ ... mass of the SPH particle (here 2500 solar masses)
520 !> \f$\rho_i\f$ ... density of the SPH particle
521 !> \f$w\left(\left|\vec{r_i}-\vec{r_j}\right|,h_i\right)\f$ ... kernel function
522 !> \f$w\left(\left|\vec{r_i}-\vec{r_j}\right|,h_i\right)=\left\{\begin{array}{cl}
    \frac{4 - 6 v^2 + 3 v^3}{4 \pi h_i^3}, & \mbox{for }\left|\vec{r_i}-\vec{r_j}\
    right|<h_i\\ \frac{\left(2-v\right)^3}{4 \pi h_i^3}, & \mbox{for }h_i \le \
    left|\vec{r_i}-\vec{r_j}\right|<2 h_i \\ 0, & \mbox{else} \end{array}\right. \
    f$
523 !------------------------------------------------------------------------------
524   subroutine interpolate_sph(x,q,dx,nn,d_region,p_region)
525     use poisson_parameters, only : ndim
526     use hydro_parameters, only : nvar, nvector
527     implicit none
528     integer , intent(in)::nn !< size of vector sweep
529     real(dp), intent(in)::dx !< cell size
530     real(dp),dimension(1:nvector,1:nvar), intent(inout)::q !< primitive variables
531     real(dp),dimension(1:nvector,1:ndim), intent(in)::x    !< coordinates
532     real(dp), intent(in)::d_region !< background density (hot medium)
533     real(dp), intent(in)::p_region !< background pressure (hot medium)
534     integer(i9) :: ii = 1_i9 !< for do loop
535     integer(i9) :: jj = 1_i9 !< for do loop
536     real(dp) :: vx_help, x_help, rho_help, p_help, T_help, r, ufac
537     real(dp) :: r_smooth,kernel_weight,smoothing_length,smoothing_length3pi
538 #if NDIM>1
539     real(dp) :: vy_help, y_help
540 #endif
541 #if NDIM>2
542     real(dp) :: vz_help,z_help
543 #endif
544     real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
```

```
        factors between cgs and user units (subroutine units in units.f90)
545     real(dp), parameter :: pi = DACOS(-1.D0)
546 #ifdef JIM
547     real(dp),parameter  :: m_part_g = 2e-3_dp*1.98892e33_dp ! m_sph_particle in g
548 #else
549     real(dp),parameter  :: m_part_g = 2500._dp*1.98892e33_dp ! m_sph_particle in g
550 #endif
551     real(dp) :: m_part,rho_part ! m_sph_particle in g
553
553     call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
554     m_part=m_part_g/scale_d/scale_l**3 ! m_sph_particle in code units
555     rho_part=m_part/dx**3 ! density corresponding to a single m_sph_particle a
            cell in code units
556 !    q(:,1)=d_region
557     do ii=1,nsph
558       smoothing_length3pi=1._dp/(smoothing_sph(ii)**3)/pi
559       if(smoothing_sph(ii).lt.0.5*dx)then
560        qloop: do jj=1,nn
561           if(abs(x(jj,1)-x_sph(ii)).lt.0.5*dx)then
562 #if NDIM>1
563            if(abs(x(jj,2)-y_sph(ii)).lt.0.5*dx)then
564 #endif
565 #if NDIM>2
566             if(abs(x(jj,3)-z_sph(ii)).lt.0.5*dx)then
567 #endif
568                vx_help=vx_sph(ii)
569 #if NDIM>1
570                vy_help=vy_sph(ii)
571 #endif
572 #if NDIM>2
573                vz_help=vz_sph(ii)
574 #endif
575                !> \var scale_T2 ... converts (P/rho) in user unit into (T/mu) in
                       Kelvin (mu: molar mass)
576                p_help=T_sph(ii)*rho_part
577                ! ADD to primitive variables the region values
578                ! mass weighted velocity (momentum conservation)
579                q(jj,2)=q(jj,2)*q(jj,1)+vx_help*rho_part
580 #if NDIM>1
581                q(jj,3)=q(jj,3)*q(jj,1)+vy_help*rho_part
582 #endif
583 #if NDIM>2
584                q(jj,4)=q(jj,4)*q(jj,1)+vz_help*rho_part
585 #endif
586                 ! pressure
587                if((q(jj,ndim+2).lt.p_region*1.000001_dp).and.(q(jj,1).lt.d_region
                       *1.000001_dp))then
588                   q(jj,ndim+2)=max(p_help,p_region)
589                else
590                   q(jj,ndim+2)=q(jj,ndim+2)+p_help
591                end if
592                q(jj,1)=q(jj,1)+rho_part
593                q(jj,2)=q(jj,2)/q(jj,1)
594 #if NDIM>1
595                q(jj,3)=q(jj,3)/q(jj,1)
```

```
596 #endif
597 #if NDIM>2
598                 q(jj,4)=q(jj,4)/q(jj,1)
599 #endif
600 #if DEBUG==2
601                 print*, "density, pressure", rho_part, p_help
602                 print*, "x, vx", x_sph(ii), vx_help
603 #if NDIM>1
604                 print*, "y, vy", y_sph(ii), vy_help
605 #endif
606 #if NDIM>2
607                 print*, "z, vz", z_sph(ii), vz_help
608 #endif
609 #endif
610                 exit qloop
611 #if NDIM>1
612               end if
613 #endif
614 #if NDIM>2
615              end if
616 #endif
617             end if
618           end do qloop
619         else
620           do jj=1,nn
621             r=abs(x(jj,1)-x_sph(ii))
622 #ifdef JIM
623             if(r.lt.2.0*smoothing_sph(ii))then
624 #else
625             !if(r.lt.2.0*smoothing_sph(ii))then
626 #endif
627 #if NDIM>1
628               r=r*r+((x(jj,2)-y_sph(ii))**2)
629 #endif
630 #if NDIM>2
631               r=r+((x(jj,3)-z_sph(ii))**2)
632 #endif
633 #if NDIM>1
634               r=sqrt(r)
635 #endif
636               r_smooth=r/smoothing_sph(ii)
637               !if((r_smooth.gt.2).and.(r.le.0.5*dx))print*, r, smoothing_sph(ii),
                      r_smooth,0.5*dx
638               if (r_smooth.lt.2._dp)then
639               !SPH kernel
640                 if(r_smooth.lt.1._dp)then
641                   kernel_weight=(1._dp-1.5_dp*r_smooth*r_smooth+
                                                &
642                        &          0.75_dp*r_smooth*r_smooth*r_smooth)*
                            smoothing_length3pi
643                 else if (r_smooth.lt.2._dp)then
644                   kernel_weight=0.25_dp*((2._dp-r_smooth)**3)*smoothing_length3pi
645                 end if
646                 rho_help=kernel_weight*m_part
647                 ufac=rho_help/rho_sph(ii)
```

```
648                 !print*, kernel_weight,ufac,rho_help,rho_sph(ii)
649                 !stop
650                 vx_help=ufac*vx_sph(ii)
651 #if NDIM>1
652                 vy_help=ufac*vy_sph(ii)
653 #endif
654 #if NDIM>2
655                 vz_help=ufac*vz_sph(ii)
656 #endif
657                 !> \var scale_T2 ... converts (P/rho) in user unit into (T/mu) in
                        Kelvin (mu: molar mass)
658                 p_help=T_sph(ii)*rho_help
659                 ! ADD to primitive variables the region values
660                 ! mass weighted velocity (momentum conservation)
661                 q(jj,2)=q(jj,2)*q(jj,1)+vx_help*rho_help
662 #if NDIM>1
663                 q(jj,3)=q(jj,3)*q(jj,1)+vy_help*rho_help
664 #endif
665 #if NDIM>2
666                 q(jj,4)=q(jj,4)*q(jj,1)+vz_help*rho_help
667 #endif
668                 ! pressure
669                 if((q(jj,ndim+2).lt.p_region*1.000001_dp).and.(q(jj,1).lt.d_region
                        *1.000001_dp))then
670                    q(jj,ndim+2)=max(p_help,p_region)
671                 else
672                    q(jj,ndim+2)=q(jj,ndim+2)+p_help
673                 end if
674                 q(jj,1)=q(jj,1)+rho_help
675                 q(jj,2)=q(jj,2)/q(jj,1)
676 #if NDIM>1
677                 q(jj,3)=q(jj,3)/q(jj,1)
678 #endif
679 #if NDIM>2
680                 q(jj,4)=q(jj,4)/q(jj,1)
681 #endif
682 #if DEBUG==2
683                 print*, "density, pressure", rho_help, p_help
684                 print*, "x, vx", x_sph(ii), vx_help
685 #if NDIM>1
686                 print*, "y, vy", y_sph(ii), vy_help
687 #endif
688 #if NDIM>2
689                 print*, "z, vz", z_sph(ii), vz_help
690 #endif
691 #endif
692 #ifdef JIM
693              end if
694 #else
695              !end if
696 #endif
697           end if
698        end do
699       end if
700     end do
```

```
701   end subroutine interpolate_sph
703
703 ! subroutine sum_sph
704 !> \short sums sph particles per grid cell. no smoothing.
705 !------------------------------------------------------------------------
706 !> \version 1.0
707 !> \author Katharina M. Fierlinger
708 !> \date last modification 26.03.2011
709 !------------------------------------------------------------------------
710 !> \details PURPOSE:
711 !> \n interpolate sph initial conditions onto grid
712 !> e.g. velocities are interpolated like
713 !> \f$v_j=\sum_i v_i \frac{m_i w\left(\left|\vec{r}_{i}-\vec{r}_{j}\right|,h_i\
    right)}{\rho_i}\f$
714 !> with
715 !> \f$h\f$ ... smoothing length
716 !> \f$\vec{r}_i, \vec{r}_j\f$ ... location of SPH particles
717 !> \f$m_i\f$ ... mass of the SPH particle (here 2500 solar masses)
718 !> \f$\rho_i\f$ ... density of the SPH particle
719 !> \f$w\left(\left|\vec{r_i}-\vec{r_j}\right|,h_i\right)\f$ ... kernel function
720 !> \f$w\left(\left|\vec{r_i}-\vec{r_j}\right|,h_i\right)=\left\{\begin{array}{cl}
    \frac{4 - 6 v^2 + 3 v^3}{4 \pi h_i^3}, & \mbox{for }\left|\vec{r_i}-\vec{r_j}\
    right|<h_i\\ \frac{\left(2-v\right)^3}{4 \pi h_i^3}, & \mbox{for }h_i \le \
    left|\vec{r_i}-\vec{r_j}\right|<2 h_i \\ 0, & \mbox{else} \end{array}\right. \
    f$
721 !------------------------------------------------------------------------
722   subroutine sum_sph(x,q,dx,nn,d_region,p_region)
723     use poisson_parameters, only : ndim
724     use hydro_parameters, only : nvar, nvector
725     implicit none
726     integer , intent(in)::nn !< size of vector sweep
727     real(dp), intent(in)::dx !< cell size
728     real(dp),dimension(1:nvector,1:nvar), intent(inout)::q !< primitive variables
729     real(dp),dimension(1:nvector,1:ndim), intent(in)::x    !< coordinates
730     real(dp), intent(in)::d_region !< background density (hot medium)
731     real(dp), intent(in)::p_region !< background pressure (hot medium)
732     integer(i9) :: ii = 1_i9 !< for do loop
733     integer(i9) :: jj = 1_i9 !< for do loop
734     real(dp) :: vx_help, x_help, rho_part, p_help, T_help, r, ufac
735     real(dp) :: r_smooth,kernel_weight,smoothing_length,smoothing_length3pi
736 #if NDIM>1
737     real(dp) :: vy_help, y_help
738 #endif
739 #if NDIM>2
740     real(dp) :: vz_help,z_help
741 #endif
742     real(dp) :: scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2 !< conversion
        factors between cgs and user units (subroutine units in units.f90)
743     real(dp), parameter :: pi = DACOS(-1.D0)
744     real(dp), parameter :: m_part_g = 2e-3_dp*1.98892e33_dp ! m_sph_particle in g
745     real(dp) :: m_part ! m_sph_particle in g
747
747     call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
748 !    q(:,1)=d_region
749     rho_part=m_part_g/scale_d/scale_l**3/dx**3 ! m_sph_particle in code units
```

```
750     do ii=1,nsph
751        smoothing_length3pi=1._dp/(smoothing_sph(ii)**3)/pi
752        qloop: do jj=1,nn
753           if(abs(x(jj,1)-x_sph(ii)).lt.0.5*dx)then
754 #if NDIM>1
755            if(abs(x(jj,2)-y_sph(ii)).lt.0.5*dx)then
756 #endif
757 #if NDIM>2
758             if(abs(x(jj,3)-z_sph(ii)).lt.0.5*dx)then
759 #endif
760                vx_help=vx_sph(ii)
761 #if NDIM>1
762                vy_help=vy_sph(ii)
763 #endif
764 #if NDIM>2
765                vz_help=vz_sph(ii)
766 #endif
767                !> \var scale_T2 ... converts (P/rho) in user unit into (T/mu) in
                       Kelvin (mu: molar mass)
768                p_help=T_sph(ii)*rho_part
769                ! ADD to primitive variables the region values
770                ! mass weighted velocity (momentum conservation)
771                q(jj,2)=q(jj,2)*q(jj,1)+vx_help*rho_part
772 #if NDIM>1
773                q(jj,3)=q(jj,3)*q(jj,1)+vy_help*rho_part
774 #endif
775 #if NDIM>2
776                q(jj,4)=q(jj,4)*q(jj,1)+vz_help*rho_part
777 #endif
778                 ! pressure
779                if((q(jj,ndim+2).lt.p_region*1.000001_dp).and.(q(jj,1).lt.d_region
                       *1.000001_dp))then
780                   q(jj,ndim+2)=max(p_help,p_region)
781                else
782                   q(jj,ndim+2)=q(jj,ndim+2)+p_help
783                end if
784                q(jj,1)=q(jj,1)+rho_part
785                q(jj,2)=q(jj,2)/q(jj,1)
786 #if NDIM>1
787                q(jj,3)=q(jj,3)/q(jj,1)
788 #endif
789 #if NDIM>2
790                q(jj,4)=q(jj,4)/q(jj,1)
791 #endif
792 #if DEBUG==2
793                print*, "density, pressure", rho_part, p_help
794                print*, "x, vx", x_sph(ii), vx_help
795 #if NDIM>1
796                print*, "y, vy", y_sph(ii), vy_help
797 #endif
798 #if NDIM>2
799                print*, "z, vz", z_sph(ii), vz_help
800 #endif
801 #endif
802                exit qloop
```

```
803 #if NDIM>1
804               end if
805 #endif
806 #if NDIM>2
807             end if
808 #endif
809            end if
810         end do qloop
811       end do
812     end subroutine sum_sph
813
814 !> \short calculates how many neighbours can be found inside a given smoothing
        length
815 !------------------------------------------------------------------------------
816 !> \version 1.0
817 !> \author Katharina M. Fierlinger
818 !> \date last modification 27.03.2011
819 !------------------------------------------------------------------------------
820 !> \details  PURPOSE:
821 !> \n calculates how many neighbours can be found inside a given smoothing length
822 !------------------------------------------------------------------------------
823     subroutine count_neighbours(x_max,y_max,z_max)
824       implicit none
825       integer(i9) :: ii = 1_i9 !< for do loop: loop over particles
826       integer(i9) :: jj = 1_i9 !< for do loop: loop over neighbours
827       integer(i9) :: n_neighbour_ii = 1_i9 !< particles inside the smoothing length
            of particle ii
828       real(dp) :: r  !< distance between SPH particle ii and jj
829       real(dp) :: x_max,y_max,z_max
830 #if DEBUG==2
831       print*, "counting␣neighbours"
832 #endif
833       do ii=1,nsph ! loop over SPH particles
834          n_neighbour_ii = 0_i9
835          if(abs(x_sph(ii)).le.x_max)then
836             if(abs(y_sph(ii)).le.y_max)then
837 #if NDIM>2
838                if(abs(z_sph(ii)).le.z_max)then
839 #endif
840                   do jj=1,nsph ! loop over neighbours
841                      r=abs(x_sph(jj)-x_sph(ii))
842 #if NDIM>1
843                      r=r*r+((y_sph(jj)-y_sph(ii))**2)
844 #endif
845 #if NDIM>2
846                      r=r+((z_sph(jj)-z_sph(ii))**2)
847 #endif
848 #if NDIM>1
849                      r=sqrt(r)
850 #endif
851                      if(r.le.smoothing_sph(ii))then ! found a particle inside
                            current smoothing length
852                         n_neighbour_ii = n_neighbour_ii + 1_i9
853                      end if
854                   end do
```

```
855                 !> write neighbours for all particles inside the grid code box to
                        the screen
856                 if(rho_sph(ii).gt.10.)then
857                   print*, ii, n_neighbour_ii ,smoothing_sph(ii) ,rho_sph(ii)
858                 end if
859 #if NDIM>2
860               end if
861 #endif
862             end if
863           end if
864       end do
865     end subroutine count_neighbours
867
867 !> \short calculates radius inside which N neighbours can be found
868 !------------------------------------------------------------------------------
869 !> \version 1.0
870 !> \author Katharina M. Fierlinger
871 !> \date last modification 27.03.2011
872 !------------------------------------------------------------------------------
873 !> \details  PURPOSE:
874 !> \n calculates radius inside which N neighbours can be found
875 !------------------------------------------------------------------------------
876     subroutine calc_smoothing_length(x_max,y_max,z_max)
877       implicit none
878       integer(i9) :: ii = 1_i9 !< for do loop: loop over particles
879       integer(i9) :: jj = 1_i9 !< for do loop: loop over neighbours
880       integer(i9) :: n_neighbour_ii = 1_i9 !< particles inside the smoothing length
              of particle ii
881       integer(i9),parameter :: n_neighbour = 50_i9 !< number of neighbours inside
              smoothing length
882       real(dp) :: smoothing_length !< radius inside which you can find N particles
883       real(dp) :: r  !< distance between SPH particle ii and jj
884       real(dp) :: dr !< increment/decrement of the smoothing length (for iterative
              process)
885       real(dp) :: x_max,y_max,z_max
886 #if DEBUG==2
887       integer(i9) :: n_loop = 0_i9 !< loop counter
888       real(dp) :: sum !< deviations from Clare's smoothing length
889       sum=0._dp
890       print*, "calculating␣smoothing␣length.␣N=␣", n_neighbour
891 #endif
892       do ii=1,nsph ! loop over SPH particles
893          smoothing_length=1.0_dp
894          dr=0.01_dp
895 #if DEBUG==2
896          n_loop=0_i9
897 #endif
898          do
899             n_neighbour_ii = 0_i9
900             do jj=1,nsph ! loop over neighbours
901                r=abs(x_sph(jj)-x_sph(ii))
902 #if NDIM>1
903                r=r*r+((y_sph(jj)-y_sph(ii))**2)
904 #endif
905 #if NDIM>2
```

```
906                 r=r+((z_sph(jj)-z_sph(ii))**2)
907 #endif
908 #if NDIM>1
909                 r=sqrt(r)
910 #endif
911                 if(r.le.smoothing_length)then ! found a particle inside current
                        smoothing length
912                    n_neighbour_ii = n_neighbour_ii + 1_i9
913                 end if
914              end do
915              if (n_neighbour_ii.eq.n_neighbour) then
916                 exit ! right number of neighbours, jump out of the do-loop
917              else if  (n_neighbour_ii.lt.n_neighbour) then
918                 ! too low number of neighbours - increase smoothing length
919                 if(dr.lt.0_dp)then
920                    ! now changing from decrease to increase - change sign and step
                           size
921                    dr=(-0.2_dp)*dr
922                 end if
923                 smoothing_length=smoothing_length+dr
924              else
925                 ! too high number of neighbours - decrease smoothing length
926                 if(dr.gt.0_dp)then
927                    ! now changing from increase to decrease - change sign and step
                           size
928                    dr=(-0.2_dp)*dr
929                 end if
930                 smoothing_length=smoothing_length+dr
931              end if
932 #if DEBUG==2
933              n_loop=n_loop+1_i9
934 #endif
935           end do
936           ! write all particles inside the grid code box to the screen
937           if(abs(x_sph(ii)).le.x_max)then
938 #if NDIM>1
939              if(abs(y_sph(ii)).le.y_max)then
940 #endif
941 #if NDIM>2
942                 if(abs(z_sph(ii)).le.z_max)then
943 #endif
944 #if DEBUG==2
945                    !print*, ii,n_loop,(smoothing_length-smoothing_sph(ii))/dr,
                           smoothing_length,smoothing_sph(ii)
946                    print*, ii,smoothing_length,2.0_dp*smoothing_sph(ii)
947                    sum=sum+abs(smoothing_length-2.0_dp*smoothing_sph(ii))
948 #endif
949                    smoothing_sph(ii)=smoothing_length
950 #if NDIM>2
951                 end if
952 #endif
953 #if NDIM>1
954              end if
955 #endif
956           end if
```

```
957       end do
958 #if DEBUG==2
959       print*, sum
960       stop
961 #endif
962     end subroutine calc_smoothing_length
963 end module sph
964 !-----------------------------------------------------------------------
```

Listing C.14: Store energy losses via radiative cooling: init_hydro.f90

```
29    allocate(uold(1:ncell,1:nvar+1))
30    allocate(unew(1:ncell,1:nvar+1))
107                 do ivar=1,(nvar+1)
108                    read(ilun)xx
109                    if(ivar==1)then
114                    else if(ivar>=2.and.ivar<=ndim+1)then
120                    else if(ivar>=ndim+3.and.ivar<=ndim+2+nener)then
126                    else if(ivar==ndim+2)then
148                    else
149                    ! Read passive scalars
150                       do i=1,ncache
151                          uold(ind_grid(i)+iskip,ivar)=xx(i)*uold(ind_grid(i)+
                                iskip,1)
152                       end do
153                    end if
```

Listing C.15: Include the radiative cooling loss data, when defragmenting the main memory in subroutine “defrag”: load_balance.f90

```
1280     do ivar=1,nvar+1
```

Listing C.16: Output of energy losses via radiative cooling: output_hydro.f90

```
101               ivar=nvar+1 ! Cooling (energy loss), tracer density
102               do i=1,ncache
103                  xdp(i)=uold(ind_grid(i)+iskip,ivar)
104                  !if(xdp(i).gt.0.0)print*, xdp(i)
105               end do
106               write(ilun)xdp
```

Listing C.17: Reset energy losses via radiative cooling: amr_step.f90

```
152       ! reset energy loss
153       ! ... do it here if you want to sum over a main step
154       ! ... otherwise the set_uold routine can be used
155       uold(:,nvar+1)=0.0
156       unew(:,nvar+1)=0.0
```

Listing C.18: Add a mask for regions that may cool to cooling_fine.f90. I.e. exclude the feedback region. Therefore igrid in coolfine1 needed for driver_weights

```
43  #if COOLINGWEIGHTS > 0
44        call coolfine1(ind_grid,ngrid,igrid,ilevel) ! "igrid" used for driver_weights
45  #else
```

```
46      call coolfine1(ind_grid,ngrid,ilevel)
47  #endif
70  #if COOLINGWEIGHTS > 0
71  subroutine coolfine1(ind_grid,ngrid,igrid,ilevel)
72  #else
73   subroutine coolfine1(ind_grid,ngrid,ilevel)
74  #endif
75    use amr_commons, only: active, dtnew, ncoarse, son, t
76    use hydro_commons, only: uold, cooling, isothermal, dp, icoarse_max, &
77    &  icoarse_min, boxlen, nvector, ndim, ngridmax, twotondim, smallr,    &
78    &  gamma, nvar, imetal, ixion
79    use cooling_module
83  #if COOLINGWEIGHTS > 0
84  #if COOLINGWEIGHTS > 1
85    use amr_parameters, only: r_driver,x_driver,y_driver,z_driver,coolplus
86  #endif
87    use driver
88  #endif
95    integer::ilevel,ngrid
96  #if COOLINGWEIGHTS > 0
97    integer::igrid
98  #endif
99    integer,dimension(1:nvector)::ind_grid
100 !------------------------------------------------------------------
101 !------------------------------------------------------------------
102   integer::i,ind,iskip,idim,nleaf,nnleaf
103   real(dp)::scale_nH,scale_T2,scale_l,scale_d,scale_t,scale_v
104   real(kind=8)::dtcool,dE_help
105 ! real(kind=8)::nISM,nCOM,damp_factor,cooling_switch,t_blast
106 ! real(dp)::polytropic_constant
107   integer,dimension(1:nvector),save::ind_cell,ind_leaf
108   real(kind=8),dimension(1:nvector),save::nH,T2,delta_T2,ekk
109   real(kind=8),dimension(1:nvector),save::err                    !new
110   real(kind=8),dimension(1:nvector),save::T2min,Zsolar           !new
123 #if COOLINGWEIGHTS > 0
124   real(dp):: dx
125   real(dp), dimension(1:nvector):: weight
126 #if COOLINGWEIGHTS > 1
127   real(dp) :: rscaled,one_over_boxscale
128   real(dp) :: xdriver = 0.0_dp
129   real(dp) :: ydriver = 0.0_dp
130   real(dp) :: zdriver = 0.0_dp
131 #endif
132 #endif
133 #ifdef artificial_ISM
134   real(dp) :: density_crit ! critical density: 5 Hydrogen atoms per cm^3
135 #endif
137
137 ! Conversion factor from user units to cgs units
138   call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
139 ! scale_nH converts rho in user units into nH in H/cc
140 #ifdef artificial_ISM
141   density_crit=5._dp/scale_nH !!critical density in user units
142 #endif
147 #if COOLINGWEIGHTS > 0
```

```
148 !scaled box:
149   dx=0.5_dp**ilevel
150 #if COOLINGWEIGHTS > 1
151   one_over_boxscale=dble(icoarse_max-icoarse_min+1)/boxlen
152   xdriver = one_over_boxscale*x_driver
153   ydriver = one_over_boxscale*y_driver
154   zdriver = one_over_boxscale*z_driver
155   rscaled = one_over_boxscale*(r_driver+coolplus) !For driver_weights()
156 #endif
157 #endif
175   ! Loop over cells
176   do ind=1,twotondim
177 #if COOLINGWEIGHTS > 0
178 #if COOLINGWEIGHTS > 1
179      if(cooling)then
180         if (coolplus.lt.0.0_dp) then
181            weight(:)=1._dp
182         else if (coolplus.eq.0.0_dp) then
183            weight(:)=0._dp
184            call driver_weights_fixed(ind,ilevel,igrid,ngrid,dx,        &
185 &                                    weight(1:ngrid))
186            weight(1:ngrid)=1._dp-weight(1:ngrid)
187         else
188            weight(:)=0._dp
189            call driver_weights(ind,ilevel,igrid,ngrid,dx,rscaled,      &
190 &                              xdriver,                                &
191 #if NDIM>1
192 &                              ydriver,                                &
193 #endif
194 #if NDIM>2
195 &                              zdriver,                                &
196 #endif
197 &                              weight(1:ngrid))
198 !! weights: sum of all weights = pi*r_driver**2/(boxscale*dx)**2
199 !! 0.0_dp .le. weight(i) .le. 1._dp
200            weight(1:ngrid)=1._dp-weight(1:ngrid)
201         end if
202      end if
203 #else
204      weight(:)=0._dp
205      call driver_weights_fixed(ind,ilevel,igrid,ngrid,dx,weight(1:ngrid))
206      weight(1:ngrid)=1._dp-weight(1:ngrid)
207 #endif
208 #endif
209      iskip=ncoarse+(ind-1)*ngridmax
210      do i=1,ngrid
211         ind_cell(i)=iskip+ind_grid(i)
212      end do
214
214      ! Gather leaf cells
215      nleaf=0
216      do i=1,ngrid
217 #if COOLINGWEIGHTS > 0
218         if((son(ind_cell(i))==0).and.(weight(i).gt.0.0_dp))then
219 #else
```

```
220         if(son(ind_cell(i))==0)then
221 #endif
307     !========================================
308     ! You can put your own polytrope EOS here
309     !========================================
310      do i=1,nleaf
311        T2min(i) = T_min_fix
312      end do
313 !========================================
315
315 !     if(cooling)then
316         ! Compute thermal temperature by subtracting polytrope
317 !       do i=1,nleaf
318 !          T2(i) = max(T2(i)-T2min(i),T2_min_fix)
319 !       end do
320 !     endif
376 !     if(cooling.and..not.neq_chem)then
377       if(cooling)then
378         ! Compute "thermal" temperature by subtracting polytrope
379         do i=1,nleaf
380           T2(i)=MAX(T2(i),T_min_fix,T2_min_fix)
381 !         T2(i)=MAX(T2(i),T2min(i),T2_min_fix)
382         end do
383 !       call solve_cooling(nH,T2,Zsolar,boost,dtcool,delta_T2,nleaf)
384         call solve_cooling(nH,T2,Zsolar,dtcool,delta_T2,nleaf)
385 #if DEBUG >3
386         do i=1,nleaf
387           if ((T2(i).gt.1.e7).or.(nH(i).lt.0.01)) then
388             if(delta_T2(i).gt.0.0) then
389               print*,"COOLING in non-cooling regime:",T2(i),nH(i),delta_T2(i)
390               STOP
391             end if
392           end if
393         end do
394 #endif
395       end if
411       ! Compute net energy sink (user units)
412       ! delta_T := T_new - T_old
413 !     if(cooling.or.neq_chem)then
414       if(cooling)then
415         do i=1,nleaf
416 #ifdef artificial_ISM
417           !photoionization keeps the warm phase from cooling below 10.000 K
418           if(T2(i)+delta_T2(i).le.1.e4_dp)then
419             !test if there are more than 5 particles per cubic centimeter
420             if(uold(ind_leaf(i),1).lt.density_crit)then
421               !if the density is below density_crit
422               !call it the warm phase and keep it at 10.000K
423               if(T2(i).gt.1.e4_dp)then
424                 delta_T2(i) = 1.e4_dp-T2(i) !don't cool below to 10.000K
425                 T2min(i) = 1.e4_dp          !make sure cooling weights don't
                        interfere
426               else
427                 delta_T2(i)=max(0.0_dp,delta_T2(i)) !don't cool. Keep heeting
                        terms. But don't heat to 10.000K
```

```
!                   delta_T2(i)=0.0_dp                  !don't cool but also don't heat to 100K
!                   T2min(i) = T_min_fix  !keep low temperatures
                    T2min(i) = T2_min_fix !keep low temperatures
                end if
             else
                !if there are more than 5 particles per cubic centimeter
                !call it the cold phase and keep it at T_min_fix
                if(T2(i)+delta_T2(i).le.T_min_fix)then !cooling+heating would lead to a too small temperature
                   if(T2(i).gt.T_min_fix*1.01_dp)then !the initial state was "warm enough"
                      delta_T2(i)=T_min_fix-T2(i) !don't cool below 100K
                      T2min(i) = T_min_fix !make sure cooling weights don't interfere
                   else ! The initial temperature was too small too
                      delta_T2(i)=0.0_dp ! don't cool but also don't heat to 100K
!                     T2min(i) = T_min_fix  !don't keep low temperatures
                      T2min(i) = T2_min_fix !keep low temperatures ... T2_min fix is set in cooling_module
                   end if
                end if
             end if
          end if
#endif
          !delta_T2 ... Kelvin
          !scale_T2 ... g/erg Kelvin (cm/s code-time/code-length)^2 = Kelvin (code-time/code-length)^2
          !nH          (code-mass/code-length^3)
          delta_T2(i) = delta_T2(i)*nH(i)/scale_T2/(gamma-1.0) ![code-energy-unit/code-length-unit^3]
       end do
! Turn off cooling in blast wave regions
!       if(delayed_cooling)then
!          do i=1,nleaf
!             cooling_switch = uold(ind_leaf(i),idelay)/uold(ind_leaf(i),1)
!             if(cooling_switch > 1d-3)then
!                delta_T2(i) = MAX(delta_T2(i),real(0,kind=dp))
!             endif
!          end do
!       endif
    end if

    ! Compute minimal total energy from polytrope
    do i=1,nleaf
       T2min(i) = T2min(i)*nH(i)/scale_T2/(gamma-1.0) + ekk(i) + err(i)
    end do

    ! Update total fluid energy
    if(isothermal)then
       do i=1,nleaf
          uold(ind_leaf(i),ndim+2) = T2min(i)
       end do
    else
       do i=1,nleaf
```

```
476             T2(i) = uold(ind_leaf(i),ndim+2)
477           end do
478           if(cooling)then
479 #if COOLINGWEIGHTS > 0
480             nnleaf=0
481             do i=1,ngrid
482               if((son(ind_cell(i))==0).and.(weight(i).gt.0.0_dp))then
483                 nnleaf=nnleaf+1
484                 dE_help=delta_T2(nnleaf)*weight(i)
485                 T2(nnleaf) = T2(nnleaf)+dE_help
486                 uold(ind_leaf(nnleaf),nvar+1)=(-dE_help/dtcool) ! energy lost by
                        radiative cooling
487                 ![code-energy-unit/code-length-unit^3/code-time-unit]
488               end if
489             end do
490 #else
491             do i=1,nleaf
492               dE_help=delta_T2(i)
493               T2(i) = T2(i)+dE_help
494               uold(ind_leaf(i),nvar+1)=(-dE_help/dtcool) ! energy lost by
                      radiative cooling
495 ![code-energy-unit/code-length-unit^3/code-time-unit]
496             end do
497 #endif
498           end if
499           do i=1,nleaf
500             if(T2(i).lt.T2min(i))then
501               uold(ind_leaf(i),ndim+2) = T2min(i)
502               !subtract the re-added energy from the loss
503               uold(ind_leaf(nleaf),nvar+1)=uold(ind_leaf(nleaf),nvar+1)+(T2(i)-
                      T2min(i))/dtcool
504             else
505               uold(ind_leaf(i),ndim+2) = T2(i)
506             end if
507           end do
508       end if
```

Listing C.19: Local ISM values for XY, minimal temperature in the tables: cooling_module.f90

```
68 !------------------------------------------
69 !  real(kind=8),parameter ::X        = 0.76_dp
70 !  real(kind=8),parameter ::Y        = 0.24_dp
71 !  real(kind=8),parameter ::mu_mol   = 1.2195_dp
72 ! X = 0.76 , Y = 0.24 , Z = 0.0
73 ! 1/mu_mol = X/X_x + Y/A_y + Z/A
74 ! atomic H : X_x =1
75 ! A_y = 4
76 !>>> 1./(0.76+0.24*0.25)
77 !1.2195121951219512
78 !  real(kind=8),parameter ::mu_mol   = 1.2812524863014476_dp
79 !------------------------------------------
80 ! Lodders 2003
81   real(kind=8),parameter ::X        = 0.7110_dp
82   real(kind=8),parameter ::Y        = 0.2741_dp
83   real(kind=8),parameter ::mu_mol   = 1.2812524863014476_dp
84 ! X = 0.7110 , Y = 0.2741 , Z = 0.0149
```

```
85  ! 1/mu_mol = X/X_x + Y/A_y + Z/A
86  ! atomic H : X_x =1
87  ! A_y = 4
88  ! A = 15.5 (mean solar composition)
89  ! 1./(0.711+0.2741*0.25+0.0149/15.5)
90  ! 1.2812524863014476
102 ! integer,parameter      :: nbin_T_fix=91         !resolution in temperature
103   integer,parameter      :: nbin_T_fix=81         !resolution in temperature
104   integer,parameter      :: nbin_n_fix=141        !resolution in density
105   real(kind=8),parameter:: nH_min_fix=1.e-8_dp   !minimum density smallr=1e-7
106   real(kind=8),parameter:: nH_max_fix=1.e+6_dp   !maximum density
107 ! real(kind=8),parameter:: T2_min_fix=1.e+1_dp   !minimum temperature for cooling
      table
108   real(kind=8),parameter:: T2_min_fix=1.e+2_dp   !minimum temperature for cooling
      table
109   real(kind=8),parameter:: T2_max_fix=1.e+10_dp !maximum temperature for cooling
      table
239 subroutine set_model(Nmodel,J0in_in,J0min_in,alpha_in,normfacJ0_in,zreioniz_in, &
240  &                   correct_cooling,realistic_ne, &
241  &                   h,omegab,omega0,omegaL,astart_sim,T2_sim)
305 #ifndef CLOUDY
306   if (Nmodel /= -1) then
330   end if
331 #endif
380 subroutine set_table(aexp)
381 !================================================================
382   implicit none
383   real(kind=8) :: aexp
384   integer :: nbin_n,nbin_T
385   real(kind=8) :: nH_min,nH_max,T2_min,T2_max
386   nH_min=nH_min_fix
387   nH_max=nH_max_fix
388   T2_min=max(T_min_fix,T2_min_fix)
389   T2_max=T2_max_fix
390   nbin_n=nbin_n_fix
391   nbin_T=nbin_T_fix
392   call cmp_table(nH_min,nH_max,T2_min,T2_max,nbin_n,nbin_T,aexp)
393 end subroutine set_table
553 !subroutine solve_cooling(nH,T2,zsolar,boost,dt,deltaT2,ncell)
554 subroutine solve_cooling(nH,T2,zsolar,dt,deltaT2,ncell)
555 !================================================================
556   use hydro_commons, only : gamma
567   real(kind=8)::lambda,lambda_prime,logT2max,logT2min
568   real(kind=8)::fa,fb,fprimea,fprimeb,alpha1,beta1,gamma1
573   real(kind=8)::reduce_cooling=1.0_dp ! artificially increase cooling time
580   logT2max=log10(T2_max_fix)*1.01_dp
581   logT2min=log10(T2_min_fix)*0.99_dp
587   precoeff= X*(gamma-1.0_dp)/kB ! =2._dp*X/(3._dp*kB)
590 !    facH(i)=MIN(MAX(log10(nH(i)/boost(i)),table%nH(1)),table%nH(table%n1))
591       facH(i)=MIN(MAX(log10(nH(i)),table%nH(1)),table%nH(table%n1))
635         if((facT.le.logT2max).and.(facT.ge.logT2min))then
706           lambda=lambda*reduce_cooling              !artificially increase
                cooling time
707           lambda_prime=lambda_prime*reduce_cooling  !artificially increase
                cooling time
```

```
709
709             !1/wcool ... cooling_step size ... limits cooling step size
710             !varmax          decreases step size
711             !reduce_cooling increases step size
712             wcool=MAX(abs(lambda)/tau(ind(i))*varmax,wmax(ind(i)),-lambda_prime*
                    varmax)
714
714             tau_old(ind(i))=tau(ind(i))
715             !T = T0 (1 + lambda_prime dt - lambda/T0 dt)/( 1 + lambda_prime dt)
716             !varmax          ... decreases step size ... smaller change in T per
                    step
717             !reduce_cooling ... increases step size and decreases lambda(prime) ...
                     same change in T per step if the step size is not limited by wmax
                    ... less steps ... smaller change in T (overall)
718             tau(ind(i))=tau(ind(i))*(1._dp+lambda_prime/wcool-lambda/tau(ind(i))/
                    wcool)/(1._dp+lambda_prime/wcool)
719             time_old(ind(i))=time(ind(i))
720             time(ind(i))=time(ind(i))+1._dp/wcool
722
722             if (DEBUG.gt.0) then
723              if ((lambda.lt.0.0).and.(tau_old(ind(i)).gt.101.0)) then
724                write(*,'(8(1X,E12.5))') lambda,lambda_prime, wcool, tau_old(ind(i))
                       , tau(ind(i)), time_old(ind(i)),  time(ind(i)), time_max(ind(i))
725              end if
726             end if
728
728           else
730
730             time(ind(i))=time_max(ind(i))
732
732           end if
749     ! Compute exact time solution
750     do i=1,ncell
751     ! time_old < time_max < time ... density scaled cooling time
752       if(time(i).gt.time_max(i))then
753        tau(i)=tau(i)*(time_max(i)-time_old(i))/(time(i)-time_old(i)) & ! "right
              weight" * new temperature
754 &           +tau_old(i)*(time(i)-time_max(i))/(time(i)-time_old(i))    ! "left
      weight"  * old temperature
755       end if
756     end do
769     ! Compute delta T
770     do i=1,ncell
771       !avoid problems caused by number precision
772       if ((tau_ini(i).gt.T2_max_fix) &
773 &    .or.(tau_ini(i).le.T2_min_fix*1.1_dp) &
774 &    .or.(nH(i).lt.nH_min_fix)       ) then
775 !#if DEBUG==3
776 !      if (deltaT2(i).lt.0.0_dp) then
777 !        print*,"Tini:",tau_ini(i),"deltat",deltaT2(i),"fraction",deltaT2(i)/
      tau_ini(i),time(i),tau(i)
778 !        STOP
779 !      end if
780 !#endif
781       deltaT2(i)=0.0_dp
```

```
782       else
783        deltaT2(i)=tau(i)-tau_ini(i)
784       end if
785     end do
787
787 end subroutine solve_cooling
```

Listing C.20: Allow changes to the output times for restarted simulations: init_amr.f90

```
290            & (ngrid_current>ngridmax))then
291 !           & (ngrid_current>ngridmax).or.(noutput2>noutput) )then
303       do ii=1,noutput
304         iii=ii !ii ... index of the 1st output after restart
305         if (tout(ii).gt.t) exit
306       end do
307       !noutout is the index of the last output after restart
308       iout=iout2 ! number of previous outputs
309       tout2(1:iout)=t
310       tout2(iout+1:iout+noutput-iii+1)=tout(iii:noutput)
311       tout2(iout+noutput-iii+2:noutput)=0.0
312       noutput=iout+noutput-iii+1
313       print*,"output␣files␣will␣be␣generated␣at␣t="
314       print*, tout2(iout:noutput)
315       tout=tout2
316       !tout(1:noutput2)=tout2(1:noutput2)
317       !aout(1:noutput2)=aout2(1:noutput2)
318       ifout=ifout2
319       read(ilun)dtold(1:nlevelmax2)
```

Listing C.21: Ignore velocities in almost empty cells, remove outflows from empty cells, "Alustop": in HLLC tracer-flux only if accepting cell is warm enough: godunov_utils.f90

```
29
29    real(dp)::dtcell,smallp,help_EK
30    integer::k,idim
31 #if NENER>0
32    integer::irad
33 #endif
35
35   !smallc= 1.e-10
36   !smallr= 1.e-8
37    smallp = smallc**2/gamma ! 1.e-20/gamma
39
39    ! Convert to primitive variables
40    if((verbose_patches).and.(minval(uu(1:ncell,1)).le.smallr))then
41     print*, "godunov␣utils:␣lowest␣density:",minval(uu(1:ncell,1))      &
42 &   ,"␣hightest␣density:␣", maxval(uu(1:ncell,1))
43     print*,"godunov␣utils:␣lowest␣pressure:",minval(uu(1:ncell,ndim+2)) &
44 &   ,"␣hightest␣pressure:␣", maxval(uu(1:ncell,ndim+2))
45    end if
46    do k = 1,ncell
47       uu(k,1)=max(uu(k,1),smallr)
48    end do
49    ! Velocity
50    do idim = 1,ndim
51       do k = 1, ncell
```

```
52          uu(k,idim+1) = uu(k,idim+1)/uu(k,1)
53        end do
54      end do
55      ! Internal energy
56      do idim = 1,ndim
57        do k = 1, ncell
58          uu(k,ndim+2) = uu(k,ndim+2)-half*uu(k,1)*uu(k,idim+1)**2
59        end do
60      end do
61  #if NENER>0
62      do irad = 1,nener
63        do k = 1, ncell
64          uu(k,ndim+2) = uu(k,ndim+2)-uu(k,ndim+2+irad)
65        end do
66      end do
67  #endif
70
70
70      ! Debug
71      if(debug)then
72       do k = 1, ncell
73  #ifdef KMFCLEAN
74      ! KMF patch: cells with density = min. density are allowed
75        if(uu(k,ndim+2).le.smallp.or.uu(k,1).lt.smallr)then
76  #else
77        if(uu(k,ndim+2).le.smallp.or.uu(k,1).le.smallr)then
78  #endif
79          write(*,*)'stop in cmpdt'
80          !use driver call print_xyz(ind,ilevel,igrid,ngrid,dx,i)
81          write(*,*)'dx =',dx
82          write(*,*)'k=',k
83          write(*,*)'ncell=',ncell
84          write(*,*)'rho  =',uu(k,1)
85          write(*,*)'rho_min =',smallr
86          write(*,*)'P_min =',smallp
87          write(*,*)'P  =',uu(k,ndim+2)
88          write(*,*)'vel  =',uu(k,2:ndim+1)
89          help_EK=0.0_dp
90          do idim = 1, ndim
91           help_EK = help_EK + half*uu(k,1)*uu(k,idim+1)**2
92          end do
93          write(*,*)'Etot =',uu(k,ndim+2)+help_EK
94          write(*,*)'Ekin =',help_EK
95   !      write(*,*)'Eloss =',uu(k,nvar+1)!would be empty since it is reset after the
         output in amr_step
96          call dump_all
97          stop
98        end if
99       end do
100     end if
102
102     ! Compute maximum time step for each authorized cell
103     dt = courant_factor*dx/smallc
105
105     do k = 1, ncell
```

```
106   ! Compute pressure
107     uu(k,ndim+2) = max((gamma-one)*uu(k,ndim+2),uu(k,1)*smallp)
108 #if NENER>0
109     do irad = 1,nener
110       uu(k,ndim+2+irad) = (gamma_rad(irad)-one)*uu(k,ndim+2+irad)
111     end do
112 #endif
113   ! Compute sound speed
114     uu(k,ndim+2) = gamma*uu(k,ndim+2)
115 #if NENER>0
116     do irad = 1,nener
117       uu(k,ndim+2) = uu(k,ndim+2) + gamma_rad(irad)*uu(k,ndim+2+irad)
118     end do
119 #endif
120     uu(k,ndim+2)=sqrt(uu(k,ndim+2)/uu(k,1))
121   ! Compute wave speed
122     uu(k,ndim+2) = dble(ndim)*uu(k,ndim+2)
123     do idim = 1,ndim
124       uu(k,ndim+2)=uu(k,ndim+2)+abs(uu(k,idim+1))
125     end do
126 #ifdef KMFCLEAN
127 !> KMF patch: ignore time steps from almost empty cells
128     if(uu(k,1).gt.smallr)then ! density
129 #endif
130       uu(k,1)=zero
131       ! Compute gravity strength ratio
132       do idim = 1,ndim
133         uu(k,1)=uu(k,1)+abs(gg(k,idim))
134       end do
135       uu(k,1)=uu(k,1)*dx/uu(k,ndim+2)**2
136       uu(k,1)=MAX(uu(k,1),0.0001_dp)
137       dtcell=dx/uu(k,ndim+2)*(sqrt(one+two*courant_factor*uu(k,1))-one)/uu(k,1)
138 !dtcell=dx*courant_factor/uu(k,ndim+2)
139 !if(dtcell.lt.0.01)then
140 !  print*, dtcell, uu(k-1,1:ndim+2), k-1 , helpdt(k-1,1:ndim+2)
141 !  print*, dtcell, uu(k  ,1:ndim+2), k  , helpdt(k,1:ndim+2)
142 !  print*, dtcell, uu(k+1,1:ndim+2), k+1 , helpdt(k+1,1:ndim+2)
143 !  stop
144 !end if
145       dt = min(larget,dt,dtcell)
146 #ifdef KMFCLEAN
147 !> KMF patch: ignore time steps from almost empty cells
148     end if
149 #endif
150   end do
151 end subroutine cmpdt
237 #if defined(REFINE_BUBBLE) && ( REFINE_BUBBLE > 0 )
238         ok(k) = ok(k) .or. error > err_grad_d .or. dm < 0.01*d_region(1)  ! refine
              all cells inside the wind blown bubble
239 #else
240         ok(k) = ok(k) .or. error > err_grad_d
241 #endif
1111   INTEGER ::ivar, i
1112 #if defined ALUSTOP && ( ALUSTOP > 0 )
1113   REAL(dp)::Tcell,scale_nH,scale_T2,scale_l,scale_d,scale_t,scale_v
```

```
1114   REAL(dp), parameter :: Tmin26Al = 1.e6_dp  !K
1115 #endif
1117
1117 #if defined ALUSTOP && ( ALUSTOP > 0 )
1118   call units(scale_l,scale_t,scale_d,scale_v,scale_nH,scale_T2)
1119 #endif
1299 #if defined ALUSTOP && ( ALUSTOP > 0 )
1300 #if NVAR > 2+NDIM+NENER
1301         !flux only if accepting cell is warm enough.
1302         if(ustar>0)then
1303            Tcell=pr/rr*scale_T2
1304            if (Tcell.lt.Tmin26Al)then
1305               fgdnv(i,3+ndim+nener:4+ndim+nener) = 0.0_dp
1306            else
1307               fgdnv(i,3+ndim+nener:4+ndim+nener) = ro*uo* &
1308 &                        qleft(i,3+ndim+nener:4+ndim+nener)
1309            end if
1310         else
1311            Tcell=pl/rl*scale_T2
1312            if (Tcell.lt.Tmin26Al)then
1313               fgdnv(i,3+ndim+nener:4+ndim+nener) = 0.0_dp
1314            else
1315               fgdnv(i,3+ndim+nener:4+ndim+nener) = ro*uo* &
1316 &                        qright(i,3+ndim+nener:4+ndim+nener)
1317            end if
1318         endif
1319 #endif
1320 #if NVAR > 2+NDIM+NENER+2
1321      do ivar = 3+ndim+nener+2,nvar
1322         if(ustar>0)then
1323            fgdnv(i,ivar) = ro*uo*qleft (i,ivar)
1324         else
1325            fgdnv(i,ivar) = ro*uo*qright(i,ivar)
1326         endif
1327      end do
1328 #endif
1329 #else
1330 #if NVAR > 2+NDIM+NENER
1331      do ivar = 3+ndim+nener+2,nvar
1332         if(ustar>0)then
1333            fgdnv(i,ivar) = ro*uo*qleft (i,ivar)
1334         else
1335            fgdnv(i,ivar) = ro*uo*qright(i,ivar)
1336         endif
1337      end do
1338 #endif
1339 #endif
```

Listing C.22: Default units: amr_commons.f90

```
127   real(dp)::units_density=1.e-22_dp ! [g/cm^3]
128   real(dp)::units_time   =1.e11_dp  ! [seconds]
129   real(dp)::units_length =1.e19_dp  ! [cm]
```

Listing C.23: Check energy losses due to outflow of the computational domain: outflow.f90

```
1  module outflow
2   use amr_parameters, only: dp
3   implicit none
4  contains
5  ! subroutine outflow1(ind,ilevel,ind_grid,ind_cell,deltax,summass,sumekin,sumeth)
6  !> \short Print mass flux across grid boundaries
7  !------------------------------------------------------------------------
8  !> \version 1.0
9  !> \author Katharina M. Fierlinger
10 !> \date last modification 15.12.2010
11 !------------------------------------------------------------------------
12 !> \details PURPOSE: Print mass flux across grid boundaries
13 !------------------------------------------------------------------------
14 subroutine outflow1(ind,ilevel,ind_grid,ind_cell,deltax                  &
15 &  ,summass,sumekin,sumeth)!uold
16   use hydro_commons, only : uold, gamma
17   use amr_commons, only : active, xg, dtold  !< index array, coordinates (values
       in interval [0.5,2.5]
18   use amr_parameters, only : dp, icoarse_min, jcoarse_min, kcoarse_min !< floating
        point type, lower [xyz] coarse grid boundaries
19   use poisson_parameters, only : ndim
20   implicit none
21   integer, intent(in) :: ind, ilevel      !< position of new grids
22   integer,dimension(:), intent(in):: ind_grid, ind_cell
23   real(dp), intent(in):: deltax           !< converts cell size
24   real(dp), intent(inout), dimension(1:10) :: summass  !< total mass loss per
       timestep
25   real(dp), intent(inout), dimension(1:10) :: sumekin  !< total kinetic energy
       loss per timestep
26   real(dp), intent(inout), dimension(1:10) :: sumeth   !< total thtermal energy
       loss per timestep
27   real(dp):: dx                           !< cell size (if boxlen = 1)
28   integer :: ii,i,ix,iy,iz,nn !< loop variable, position in coordinate array, new
       grid [xyz] index, random numbers inside driver
29   integer :: ngrid                        !< grid size
30   real(dp):: vx,vy,vz                     !< velocities
31   real(dp):: xcoord,ycoord,zcoord         !< coordinates
32   real(dp):: temperature,oflux,massflux,ek_help, eth_help
33   real(dp), dimension(1:3) :: skip_loc    !< grid boundaries
34   real(dp), dimension(1:3) :: xc          !< center of new grid
35   real(dp), parameter :: minvel=1.e-8_dp !< minimum outflow speed below which mass
       /energy loss is ignored
36   real(dp):: gm1k                         !< (gamma-1._dp)/8.3e-9_dp
37   gm1k=(gamma-1._dp)/8.3e-9_dp ! in Kelvin
38   ngrid = size(ind_grid)
39   dx=0.5_dp**dble(ilevel)
40   !print flux over cell boundaries
41   !ind=1,2**ndim
42   !2d: ind=1,4
43   !3d: ind=1,8
44   ! Set new grids position
45   iz=(ind-1)/4         ! integer division -> 0 or 1
46   iy=(ind-1-4*iz)/2    ! integer division -> 0 or 1
47   ix=(ind-1-2*iy-4*iz)! integer division -> 0 or 1
48   skip_loc=(/0.0_dp,0.0_dp,0.0_dp/)
```

```
49    xc(1)=(dble(ix)-0.5_dp)*dx ! -0.5D0 or +0.5D0
50    skip_loc(1)=dble(icoarse_min)
51  #if NDIM>1
52    xc(2)=(dble(iy)-0.5_dp)*dx ! -0.5D0 or +0.5D0
53    skip_loc(2)=dble(jcoarse_min)
54  #endif
55  #if NDIM>2
56    xc(3)=(dble(iz)-0.5_dp)*dx ! -0.5D0 or +0.5D0
57    skip_loc(3)=dble(kcoarse_min)
58  #endif
59    do i=1,ngrid
60       !xg(:,1)-1.5 ... values in interval [-1,1]
61       xcoord=xg(ind_grid(i),1)+xc(1)-skip_loc(1)
62       if((xcoord.lt.dx).or.(xcoord.gt.1._dp-dx))then !boundary layer cell
63          if(xcoord.lt.dx)then
64             !for outflow vx < 0
65             vx=(-1._dp)*uold(ind_cell(i),2)/uold(ind_cell(i),1)
66          else
67             !for outflow vx > 0
68             vx=uold(ind_cell(i),2)/uold(ind_cell(i),1)
69          end if
70          if(vx.gt.minvel)then
71             ek_help=(uold(ind_cell(i),2)**2                              &
72  #if NDIM>1
73  &                  +uold(ind_cell(i),3)**2                              &
74  #endif
75  #if NDIM>2
76  &                  +uold(ind_cell(i),4)**2                              &
77  #endif
78  &                  )*0.5_dp/uold(ind_cell(i),1)
79             eth_help=(uold(ind_cell(i),ndim+2)-ek_help)
80             temperature=gm1k*eth_help/uold(ind_cell(i),1) ! in Kelvin
81             ii=max(min(floor(log10(temperature)+1.0),10),1)
82             !massflux = dx*dx*v*dt*rho
83             oflux=dtold(ilevel)*(deltax)**2*vx
84             summass(ii)=summass(ii)+oflux*uold(ind_cell(i),1)
85             sumekin(ii)=sumekin(ii)+oflux*ek_help
86             sumeth(ii)=sumeth(ii)+oflux*eth_help
87             ! don't "cycle" since corner cells can have xyz fluxes
88          end if
89       end if
90  #if NDIM>1
91       ycoord=xg(ind_grid(i),2)+xc(2)-skip_loc(2)
92       if((ycoord.lt.dx).or.(ycoord.gt.1._dp-dx))then !boundary layer cell
93          if(ycoord.lt.dx)then
94             !for outflow vy < 0
95             vy=(-1._dp)*uold(ind_cell(i),3)/uold(ind_cell(i),1)
96          else
97             !for outflow vy > 0
98             vy=uold(ind_cell(i),3)/uold(ind_cell(i),1)
99          end if
100         if(vy.gt.minvel)then
101            ek_help=(uold(ind_cell(i),2)**2                              &
102 &                  +uold(ind_cell(i),3)**2                              &
103 #if NDIM>2
```

```
104 &                    +uold(ind_cell(i),4)**2                                  &
105 #endif
106 &                    )*0.5_dp/uold(ind_cell(i),1)
107             eth_help=(uold(ind_cell(i),ndim+2)-ek_help)
108             temperature=gm1k*eth_help/uold(ind_cell(i),1) ! in Kelvin
109             ii=max(min(floor(log10(temperature)+1.0),10),1)
110             !massflux = dx*dx*v*dt*rho
111             oflux=dtold(ilevel)*(deltax)**2*vy
112             summass(ii)=summass(ii)+oflux*uold(ind_cell(i),1)
113             sumekin(ii)=sumekin(ii)+oflux*ek_help
114             sumeth(ii)=sumeth(ii)+oflux*eth_help
115             ! don't "cycle" since corner cells can have xyz fluxes
116          end if
117       end if
118 #endif
119 #if NDIM>2
120       zcoord=xg(ind_grid(i),3)+xc(3)-skip_loc(3)
121       if((zcoord.lt.dx).or.(zcoord.gt.1._dp-dx))then !boundary layer cell
122          if(zcoord.lt.dx)then
123             !for outflow vz < 0
124             vz=(-1._dp)*uold(ind_cell(i),4)/uold(ind_cell(i),1)
125          else
126             !for outflow vz > 0
127             vz=uold(ind_cell(i),4)/uold(ind_cell(i),1)
128          end if
129          if(vz.gt.minvel)then
130             ek_help=(uold(ind_cell(i),2)**2+uold(ind_cell(i),3)**2       &
131 &                   +uold(ind_cell(i),4)**2)*0.5_dp/uold(ind_cell(i),1)
133
133             eth_help=(uold(ind_cell(i),ndim+2)-ek_help)
134             temperature=gm1k*eth_help/uold(ind_cell(i),1) ! in Kelvin
135             ii=max(min(floor(log10(temperature)+1.0),10),1)
136             !massflux = dx*dx*v*dt*rho
137             oflux=dtold(ilevel)*(deltax)**2*vz
138             summass(ii)=summass(ii)+oflux*uold(ind_cell(i),1)
139             sumekin(ii)=sumekin(ii)+oflux*ek_help
140             sumeth(ii)=sumeth(ii)+oflux*eth_help
141          end if
142       end if
143 #endif
144    end do
145 end subroutine outflow1
146 end module outflow
```

Listing C.24: Reset cooling losses and avoid negative internal energies in set_uold and remove outflows from almost empty cells in godfine1: godunov_fine.f90

```
  1 !> preprocessor: ifdef  ETOT ... increase total energy
  2 !> preprocessor: ifndef ETOT ... reduce speeds
  3 !> preprocessor: ifdef VMAX  ... set speed limit
187    ! Set uold to unew for myid cells
188    do ind=1,twotondim
189       iskip=ncoarse+(ind-1)*ngridmax
190       do ivar=1,nvar
191          do i=1,active(ilevel)%ngrid
```

```
192             uold(active(ilevel)%igrid(i)+iskip,ivar) = unew(active(ilevel)%igrid(i)
                    +iskip,ivar)
193          end do
194       end do
195       ! ... reset cooling losses here if you do not want to sum over a main step
196       do i=1,active(ilevel)%ngrid
197          ind_cell=active(ilevel)%igrid(i)+iskip
198          uold(ind_cell,nvar+1) = 0.0_dp
199          unew(ind_cell,nvar+1) = 0.0_dp
200       end do
201       if(pressure_fix)then
202          ! Correct total energy if internal energy is too small
203          do i=1,active(ilevel)%ngrid
204             ind_cell=active(ilevel)%igrid(i)+iskip
205             d=uold(ind_cell,1)
206             u=uold(ind_cell,2)/d
207 #ifdef VMAX
208             if(u>vmax)then
209              print*, "u>vmax", u, vmax
210              uold(ind_cell,2)= min(vmax*d,uold(ind_cell,2))
211              u=uold(ind_cell,2)/d
212             end if
213 #endif
214 #if NDIM>1
215             v=uold(ind_cell,3)/d
216 #ifdef VMAX
217             if(v>vmax)then
218              print*, "v>vmax", v, vmax
219              uold(ind_cell,3)= min(vmax*d,uold(ind_cell,3))
220              v=uold(ind_cell,3)/d
221             end if
222 #endif
223 #else
224             v=0.0_dp
225 #endif
226 #if NDIM>2
227            w=uold(ind_cell,4)/d
228 #ifdef VMAX
229             if(w>vmax)then
230              print*, "w>vmax", w, vmax
231              uold(ind_cell,4)= min(vmax*d,uold(ind_cell,4))
232              w=uold(ind_cell,4)/d
233             end if
234 #endif
235 #else
236            w=0.0_dp
237 #endif
238             e_kin=0.5*d*(u**2+v**2+w**2)
239 #if NENER>0
240             do irad=1,nener
241                e_kin=e_kin+uold(ind_cell,ndim+2+irad)
242             end do
243 #endif
244             e_cons=uold(ind_cell,ndim+2)-e_kin
245 #ifndef ETOT
```

```
246             if(e_cons.le.0.0)then
247               if(verbose_patches)then
248                 print*, "PATCH: e_cons is too small: e_cons= ", e_cons
249 !               print*, "uold(,ndim+2)= ",uold(ind_cell,ndim+2)
250                 print*, "PATCH: reduce speeds to ",                    &
251 &                0.99_dp*uold(ind_cell,ndim+2)/e_kin," x old speed"
252               end if
253               ! decrease speeds
254               uold(ind_cell,2:ndim+1)=0.99_dp*uold(ind_cell,2:ndim+1)*  &
255 &              uold(ind_cell,ndim+2)/e_kin
256               ! use new speeds to get new energies
257               u=uold(ind_cell,2)/d
258 #if NDIM>1
259               v=uold(ind_cell,3)/d
260 #else
261               v=0.0_dp
262 #endif
263 #if NDIM>2
264               w=uold(ind_cell,4)/d
265 #else
266               w=0.0_dp
267 #endif
268               e_kin=0.5*d*(u**2+v**2+w**2)
269               e_cons=uold(ind_cell,ndim+2)-e_kin
270               if(verbose_patches)then
271                 print*, "new (smaller) velocities:",uold(ind_cell,2:ndim+1)
272                 print*, "e_cons=", e_cons, "e_kin=", e_kin
273               end if
274             end if
275 #endif
276             ! Note: here divu=(-div.u)*dt
277             div=abs(divu(ind_cell))*dx/dtnew(ilevel)
278             e_trunc=beta_fix*d*max(div,3.0*hexp*dx)**2
279             if(e_cons<e_trunc)then
280               e_prim=enew(ind_cell)
281               if(e_prim.gt.0.0)then
282                 if(verbose_patches)print*, "PATCH in set_uold: pressure_fix"
283                 uold(ind_cell,ndim+2)=e_prim+e_kin
284               else
285 #ifdef ETOT
286                 uold(ind_cell,2:ndim+1)=0.99_dp*uold(ind_cell,2:ndim+1)* &
287 &                uold(ind_cell,ndim+2)/e_kin
288                 if(verbose_patches)then
289                   print*, "PATCH in set_uold: e_prim is zero: ",         &
290 &                  "e_prim=", e_prim, "e_kin=", e_kin, "e_cons=", e_cons, &
291 &                  "e_tot=",uold(ind_cell,ndim+2)
292                   print*, "PATCH in set_uold: reduce speeds to ",        &
293 &                  0.99_dp*uold(ind_cell,ndim+2)/e_kin,"x old speed"
294                   print*, "PATCH in set_uold new speeds:",               &
295 &                  uold(ind_cell,2:ndim+1)
296                 u=uold(ind_cell,2)/d
297 #if NDIM>1
298                 v=uold(ind_cell,3)/d
299 #else
300                 v=0.0_dp
```

```
301 #endif
302 #if NDIM>2
303                 w=uold(ind_cell,4)/d
304 #else
305                 w=0.0_dp
306 #endif
307                 e_kin=0.5_dp*d*(u**2+v**2+w**2)
308                 e_cons=uold(ind_cell,ndim+2)-e_kin
309                 print*, "e_cons=", e_cons, "e_kin=", e_kin
310               end if
311 #endif
312               if(verbose_patches)print*, "PATCH: e_prim is zero: ",     &
313 &             "e_prim=", e_prim, "e_kin=", e_kin, "e_cons=", e_cons
314             end if
315           end if
316         end do
317       end if
318   end do
587 subroutine godfine1(ind_grid,ncache,ilevel)
823           if(unew(ind_cell(i),1).le.smallr)then
824             if(verbose_patches)write(*,112) unew(ind_cell(i),1:nvar)
825             !set density
826             unew(ind_cell(i),1)=smallr
827             !set velocities
828             unew(ind_cell(i),2:ndim+1)=0.0_dp
829             !set pressure
830             unew(ind_cell(i),ndim+2)=min(1e-20_dp,unew(ind_cell(i),ndim+2)) !
                  smallp
831             !remove outflow
832             flux(i,i3    ,j3    ,k3    ,1:nvar,idim)=max(0.0_dp,         &
833            & flux(i,i3    ,j3    ,k3    ,1:nvar,idim)) ! inflow
834             flux(i,i3+i0,j3+j0,k3+k0,1:nvar,idim)=min(0.0_dp,         &
835            & flux(i,i3+i0,j3+j0,k3+k0,1:nvar,idim)) ! outflow
836           end if
```

Listing C.25: Remove outflows from almost empty cells and use average pressure of adjacent cells in subroutine ctoprim: umuscl.f90

```
848 subroutine ctoprim(uin,q,c,gravin,dt,ngrid)
860   real(dp),dimension(:), allocatable :: qhelp
861   integer ::i, j, k, l, n, idim, nqhelp
890             ! remove outflows from q, set velocity zero
891             ! and use average pressures of adjacent cells
892             if(uin(l,i,j,k,1).le.smallr)then
893               if(verbose_patches)then
894                 print*, "PATCH: ctoprim: detected too small density"
895                 print*, "rho(uin)", uin(l,i,j,k,1),"<",smallr
896               end if
897               !set velocities
898               !q(l,i,j,k,2:ndim+1) = uin(l,i,j,k,2:ndim+1)*oneoverrho
899               q(l,i,j,k,2:ndim+1)=0.0_dp
900               !remove outflows from empty cells and set pressure
901               allocate(qhelp(1:2*ndim))
902               qhelp=0.0_dp
903               nqhelp=0
904               if(i.gt.iu1)then !this q was already written
```

```
905                      q(l,i-1,j,k,2)=max(q(l,i-1,j,k,2),0.0_dp)
906                      qhelp(1)= q(l,i-1,j,k,ndim+2)
907                      nqhelp=nqhelp+1
908                    end if
909                    if(i.lt.iu2)then !q not yet written
910                      uin(l,i+1,j,k,2)=min(uin(l,i+1,j,k,2),0.0_dp)
911                      qhelp(2)=(gamma-one)*uin(l,i+1,j,k,1)*                    &
912     &                 MAX(smalle,uin(l,i+1,j,k,ndim+2)/uin(l,i+1,j,k,1)-half*&
913     &                 sum((uin(l,i+1,j,k,2:ndim+1)/uin(l,i+1,j,k,1))**2))
914                      nqhelp=nqhelp+1
915                    end if
916     #if NDIM>1
917                    if(j.gt.ju1)then
918                      q(l,i,j-1,k,3)=max(q(l,i,j-1,k,3),0.0_dp)
919                      qhelp(3)= q(l,i,j-1,k,ndim+2)
920                      nqhelp=nqhelp+1
921                    end if
922                    if(j.lt.ju2)then
923                      uin(l,i,j+1,k,3)=min(uin(l,i,j+1,k,3),0.0_dp)
924                      qhelp(4)=(gamma-one)*uin(l,i,j+1,k,1)*                    &
925     &                 MAX(smalle,uin(l,i,j+1,k,ndim+2)/uin(l,i,j+1,k,1)-half*&
926     &                 sum((uin(l,i,j+1,k,2:ndim+1)/uin(l,i,j+1,k,1))**2))
927                      nqhelp=nqhelp+1
928                    end if
929     #if NDIM>2
930                    if(k.gt.ku1)then
931                      q(l,i,j,k-1,4)=max(q(l,i,j,k-1,4),0.0_dp)
932                      qhelp(5)= q(l,i,j,k-1,ndim+2)
933                      nqhelp=nqhelp+1
934                    end if
935                    if(k.lt.ku2)then
936                      uin(l,i,j,k+1,4)=min(uin(l,i,j,k+1,4),0.0_dp)
937                      qhelp(6)=(gamma-one)*uin(l,i,j,k+1,1)*                    &
938     &                 MAX(smalle,uin(l,i,j,k+1,ndim+2)/uin(l,i,j,k+1,1)-half*&
939     &                 sum((uin(l,i,j,k+1,2:ndim+1)/uin(l,i,j,k+1,1))**2))
940                      nqhelp=nqhelp+1
941                    end if
942     #endif
943     #endif
944                    !mean
945                    q(l,i,j,k,ndim+2) = max(smallp,sum(qhelp)/real(nqhelp))
946                    !!median
947                    !! 1d: minval
948                    !! 2d: 3rd largest value (from 4)
949                    !! 3d: 4th largest value (from 6)
950                    !do ihelp=1,ndim
951                    !  qhelp(maxloc(qhelp))=0.0_dp
952                    !end do
953                    !q(l,i,j,k,ndim+2) = maxval(qhelp)
954                    deallocate(qhelp)
955     #if NENER>0
956                  ! Compute thermal pressure
957                    eint = MAX(q(l,i,j,k,ndim+2)/(gamma-one)*oneoverrho-erad,smalle)
958     #endif
959                  else
```

Listing C.26: Makefile

```
1  ############################################################################
2  # If you have problems with this makefile, contact Romain.Teyssier@cea.fr
3  ############################################################################
4  # Compilation time parameters
5  NVECTOR = 500 # ... default: NVECTOR = 500
6  NDIM = 3
7  NPRE = 8
8  NVAR = 7 #... default: NVAR = NDIM+2+2+1 (rho, vx, vy, vz, ui, 26Al, 60Fe, aton)
9  NENER = 0
10 SOLVER = hydro
11 #undef WITHOUTMPI     !... for single processor runs
12 #undef QUADHILBERT
13 #undef SOLVERmhd      !... use MHD
14 #define NOSYSTEM 1    !... avoid system calls
15 ############################################################################
16 # Katharina's compilation time parameters
17 #define DEBUG_2          !... debugging output
18 #define DEBUG_3          !... more debugging output
19 #define DEBUG_0          !... no debugging output
20 DEBUG = 0
21 #
22 #define CLOUDY_1         ! use CLOUDY cooling implemented by Eva Ntormousi (2011,
       ApJ 731, 13)
23 CLOUDY = 1
24 #define COOLINGWEIGHTS_0 !... use unweighted cooling losses in all cells
25 #define COOLINGWEIGHTS_2 !... use user defined coolplus from namelist, reduce
       cooling near feedback region
26 #define COOLINGWEIGHTS_1 !... use coolplus = 0, use mask, no coolingin cells with
       feedback
27 COOLINGWEIGHTS = 0
28 #
29 #define DEBUGCOOLING_1   !... debugging output if smallnum_cooling condition is
       violated
30 DEBUGCOOLING = 1
31 #define KAHANBABUSKA_1   !... check if the calculating the sum of all densities
       runs into problems
32 KAHANBABUSKA = 1
33 #define KMFCLEAN_1       !... cells with minimal density (uu(k,1).eq.smallr) are
       allowed, time steps from almost empty cells are ignored
34 KMFCLEAN = 1
35 #
36 #define MAXDRIVERGRID_1  !... enhance refinement in feedback region
37 MAXDRIVERGRID = 1
38 #define SMOOTH_DRIVER_EDGE_1 !... calculate weights for cells partly inside the
       driver region
39 SMOOTH_DRIVER_EDGE = 1
40 #define ZEROREDSHIFT_1   ! ignore redshifts in cooling_module.
41 ZEROREDSHIFT = 1
42 #undef REFINE_BUBBLE  !... refine all cells with densities below 0.01*d_region(1)
43 REFINE_BUBBLE = 1
44 #undef CARINA          !... sequential star formation
45 #undef DECAYINTERVAL  !... set lower limit for the time interval for the 26Al decay
46 #undef EKIN           !... insert kinetic energy, not thermal energy
47 #undef ETOT           !... increase total energy, don't reduce speeds
```

```
48 #undef IGNOREX        !... ignore xn to refine only close to x axis
49 #undef MASSFLUX_1     !... print mass flux
50 #undef MHD            !... use MHD in FromangTeyssier2006/init_flow_fine.f90
51 #undef RANDZELLEN     !... interpolate partly filled cells of spherical regions
52 #undef SPH            !... read SPH data
53 #undef THII           !... set T = 10.000 Kelvin in driver region
54 #undef TMAX           !... don't allow temperatures above 5.0e6 Kelvin
55 #undef TMIN           !... check if the total energy is larger than the kinetic
       energy
56 #undef VMAX           !... set speed limit in ramses_wind_cleanlowdens_patches/
       godunov_fine.f90
57 #undef WITHTURB       ! for maclow_eva/init_flow_fine.f90
58 ########################################################################
59 #PATCH0 =../mypatch/aton
60 #PATCH1 =../mypatch/ramses_wind_ISM_phases
61 #PATCH1 =../mypatch/cooling_module_eva
62 PATCH2 =../mypatch/ramses_wind_cleanlowdens_patches
63 PATCH3 =../mypatch/ramses_wind_standard_patches
64 EXEC = ramsesWind_
65 #PROFILER = -pg -fno-inline-functions
66 #ATON_FLAGS = -DATON  # Uncomment to enable ATON.
67 ########################################################################
68 COMPILEPARS = -DNVECTOR=$(NVECTOR) -DNVAR=$(NVAR) -DNDIM=$(NDIM) -DNPRE=$(NPRE) -
       DNENER=$(NENER) -DSOLVER$(SOLVER) -DDEBUG=$(DEBUG) -DCOOLINGWEIGHTS=$(
       COOLINGWEIGHTS) -DDEBUGCOOLING=$(DEBUGCOOLING) -DCLOUDY=$(CLOUDY) -
       DKAHANBABUSKA=$(KAHANBABUSKA) -DKMFCLEAN=$(KMFCLEAN) -DMAXDRIVERGRID=$(
       MAXDRIVERGRID) -DSMOOTH_DRIVER_EDGE=$(SMOOTH_DRIVER_EDGE) -DZEROREDSHIFT=$(
       ZEROREDSHIFT) -DREFINE_BUBBLE=$(REFINE_BUBBLE) $(ATON_FLAGS)
69 ########################################################################
70 # Fortran compiler options and directives
72
72 # --- No MPI, ifort ---------------------------------------
73 #F90 = /home/katharina/intel/bin/ifort # optimal.universe-cluster.de
74 #F90 = /opt/intel/bin/ifort            # 10.155.59.244 # 10.155.59.15
75 #F90 = /opt/intel/Compiler/11.1/069/bin/intel64/ifort # 10.155.59.237
76 #F90 = /opt/intel/Compiler/11.1/046/bin/ia32/ifort    # 10.155.59.82
77 #FFLAGS =  -O0 -Warn -g -traceback -fpe0 -ftrapuv -cpp -DNOSYSTEM # for debugging
       only
78 #FFLAGS =  -O3 -cpp -DWITHOUTMPI  -DNOSYSTEM
79 #FFLAGS =      -cpp -DWITHOUTMPI  -DNOSYSTEM #default
81
81 # --- MPI, ifort syntax -----------------------------------
82 #F90 = /usr/bin/mpif90 #10.155.59.244, optimal.universe-cluster.de (default: ifort
       )
83 #F90 = /usr/local/OpenMPI-intel/bin/mpif90 #10.155.59.237 (default: ifort)
84 #F90 = /usr/local/mpich2-1.0/bin/mpif90 -f90=ifort #10.155.59.237
85 #F90 = /usr/bin/mpif90 -g -traceback
86 #FFLAGS = -O0 -cpp -DNOSYSTEM
87 #FFLAGS = -O2 -cpp -DNOSYSTEM
88 #FFLAGS = -O3 -cpp -DNOSYSTEM
89 #FFLAGS = -cpp -fast -DNOSYSTEM #default
91
91 # --- No MPI, gfortran ------------------------------------
92 F90 = gfortran -O3 -frecord-marker=4 -fbacktrace -ffree-line-length-none -g
93 FFLAGS = -x f95-cpp-input -DWITHOUTMPI
```

```
95
95 # --- MPI, gfortran syntax ---------------------------------
96 #F90 = mpif90 -O3
97 #FFLAGS = -x f95-cpp-input
99
99 ########################################################################
100 MOD = mod
101 ########################################################################
102 # MPI librairies
103 #LIBMPI = -lmpi_cxx
104 #LIBMPI = -lfmpi -lmpi -lelan
106
106 # --- CUDA libraries, for Titane ---
107 #LIBCUDA = -L/usr/local/cuda/lib64 -lm -lcuda -lcudart
108 #LIBCUDA = -L/opt/cuda/lib -lm -lcuda -lcudart
110
110 LIBS = $(LIBMPI)
111 ########################################################################
112 # Sources directories are searched in this exact order
113 VPATH = $(PATCH0):$(PATCH1):$(PATCH2):$(PATCH3):../$(SOLVER):../aton:../hydro:../
    pm:../poisson:../amr
114 ########################################################################
115 # All objects
116 MODOBJ = amr_parameters.o amr_commons.o random.o pm_parameters.o pm_commons.o
    poisson_parameters.o poisson_commons.o hydro_parameters.o hydro_commons.o
    cooling_module.o bisection.o sparse_mat.o clfind_commons.o gadgetreadfile.o
    driver.o geneva_models.o maclow.o outflow.o sph.o
117 AMROBJ = read_params.o init_amr.o init_time.o init_refine.o adaptive_loop.o
    amr_step.o update_time.o output_amr.o flag_utils.o physical_boundaries.o
    virtual_boundaries.o refine_utils.o nbors_utils.o hilbert.o load_balance.o
    title.o sort.o cooling_fine.o units.o light_cone.o movie.o
118 # Particle-Mesh objects
119 PMOBJ = init_part.o output_part.o rho_fine.o synchro_fine.o move_fine.o newdt_fine
    .o particle_tree.o add_list.o remove_list.o star_formation.o sink_particle.o
    feedback.o clump_finder.o clump_merger.o flag_formation_sites.o
120 # Poisson solver objects
121 POISSONOBJ = init_poisson.o phi_fine_cg.o interpol_phi.o force_fine.o
    multigrid_coarse.o multigrid_fine_commons.o multigrid_fine_fine.o
    multigrid_fine_coarse.o gravana.o boundary_potential.o rho_ana.o
    output_poisson.o
122 # Hydro objects
123 HYDROOBJ = init_hydro.o init_flow_fine.o write_screen.o output_hydro.o
    courant_fine.o godunov_fine.o uplmde.o umuscl.o interpol_hydro.o godunov_utils
    .o condinit.o hydro_flag.o hydro_boundary.o boundana.o read_hydro_params.o
    synchro_hydro_fine.o
124 # All objects
125 AMRLIB = $(AMROBJ) $(HYDROOBJ) $(PMOBJ) $(POISSONOBJ)
126 # ATON objects
127 ATON_MODOBJ = timing.o radiation_commons.o rad_step.o
128 ATON_OBJ = observe.o init_radiation.o rad_init.o rad_boundary.o rad_stars.o
    rad_backup.o ../mypatch/aton/atonlib/libaton.a
129 ########################################################################
130 ramses: $(MODOBJ) $(AMRLIB) ramses.o
131 	$(F90) $(MODOBJ) $(AMRLIB) ramses.o -o $(EXEC)$(NDIM)d $(LIBS)
132 #	$(F90) $(PROFILER) $(AMRLIB) ramses.o -o $(EXEC)$(NDIM)d $(LIBS)
```

```
ramses_aton: $(MODOBJ) $(ATON_MODOBJ) $(AMRLIB) $(ATON_OBJ) ramses.o
	$(F90) $(MODOBJ) $(ATON_MODOBJ) $(AMRLIB) $(ATON_OBJ) ramses.o -o $(EXEC)$(NDIM)d $(LIBS) $(LIBCUDA)
#########################################################################
%.o:%.f90
	$(F90) $(FFLAGS) $(COMPILEPARS)  -c $^ -o $@
#	$(F90) $(PROFILER) $(FFLAGS) $(COMPILEPARS)  -c $^ -o $@
#########################################################################
clean :
	rm *.o *.$(MOD)
#########################################################################
```

Listing C.27: Example of a namelist: IC_snwind_3d.nml

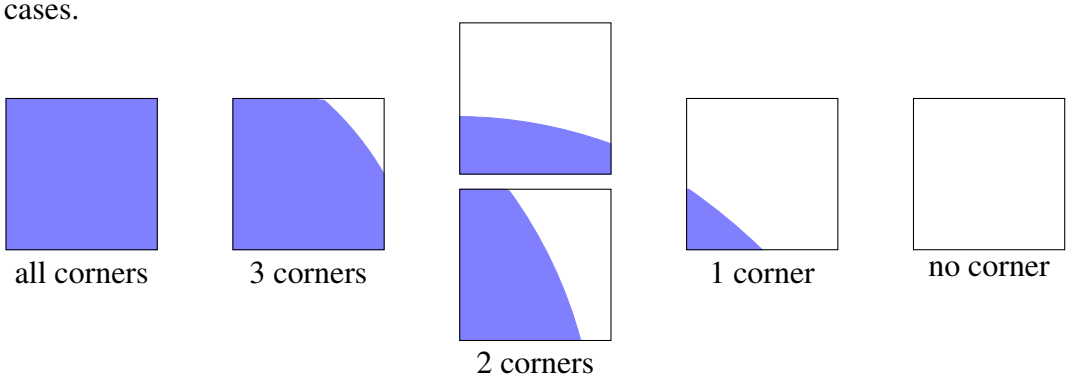
```
This namelist contains various input parameters for RAMSES runs

&DRIVER_PARAMS
file_driver='wind.dat'
file_sn='sn.dat'
r_driver=0.75d0
x_driver=0.0d0
y_driver=0.0d0
z_driver=0.0d0
coolplus=0.0d0
n_stars=1.0d0
/

&RUN_PARAMS
hydro=.true.
debug=.true.
ncontrol=10
nsubcycle=2
nremap=10
nrestart=0
verbose_patches=.true.
/

&AMR_PARAMS
levelmin=7
levelmax=7
ngridmax=340000
boxlen=21.25
/

&BOUNDARY_PARAMS
nboundary = 6
bound_type= 2, 2,  2,  2,  2,  2
ibound_min=-1, 1, -1, -1, -1, -1
ibound_max=-1, 1,  1,  1,  1,  1
jbound_min= 0, 0, -1,  1, -1, -1
jbound_max= 0, 0, -1,  1,  1,  1
kbound_min= 0, 0,  0,  0, -1,  1
kbound_max= 0, 0,  0,  0, -1,  1
/

&INIT_PARAMS
```

```
nregion=2
region_type(1)='square'
region_type(2)='square'
x_center=10.625,10.625
y_center=10.625,10.625
z_center=10.625,10.625
length_x=21.25,15.5
length_y=21.25,15.5
length_z=21.25,15.5
exp_region=10.0,2.0
d_region=0.0166,1.66
u_region=0.0,0.0
v_region=0.0,0.0
w_region=0.0,0.0
p_region=1.38e-6,1.38e-6
/

&OUTPUT_PARAMS
tend=1262.43
delta_tout=15.78
/

&HYDRO_PARAMS
gamma=1.66667
courant_factor=0.8
slope_type=1
scheme='muscl'
riemann='acoustic'
pressure_fix=.true.
beta_fix=0.d0
smallr=1.e-7
/

&PHYSICS_PARAMS
cooling=.true.
T_min_fix=100.
metal=.false.
z_ave=1.0d0
T2_star=0.0
/

&REFINE_PARAMS
interpol_var=0
interpol_type=2
err_grad_d=0.1
err_grad_p=0.1
/
```

C.1 Analytic formulas for the feedback region volume

For (pseudo) 2D simulations (`nz=1`) the stellar feedback energy and mass is homogeneously distributed over the feedback region, which is a cylinder of given radius (`rdriver`) and scaled with the ratio of the volume of a one cell high cylinder with this radius to the volume of a sphere of the

same radius ($\frac{\pi r_{\rm fb}^2 \Delta x}{\frac{4\pi}{3} r_{\rm fb}^3} = \frac{3\Delta x}{4 r_{\rm fb}}$). In 3D runs the newly inserted energy and mass are homogeneously distributed over a sphere of given radius (`rdriver`). In 2D the percentage of the cell volume that is inside the feedback region can be calculated analytically. To set the integration limits, the feedback routine checks how many of the corners of the cell are inside the feedback region. The routine uses the absolute values of the x, y and z distances of the cell corners to reduce the number of different cases.

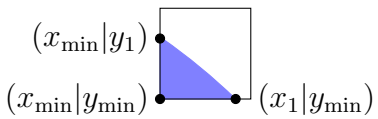


The cases "no corner" and "all corners" are trivial (0% or 100% inside).

C.1.1 2D: one corner inside the feedback region

In the 2D case with only the corner $(x_{\rm min}|y_{\rm min})$ inside the feedback region, the fraction the cell volume inside the feedback region ($p_{\rm fb}$) can be calculated with:

$$
\begin{aligned}
p_{\rm fb} &= \frac{\int_{x_{\rm min}}^{x_1} {\rm d}x \int_{y_{\rm min}}^{\sqrt{r^2-x^2}} {\rm d}y}{V_{\rm cell}} \\
&= \frac{\int_{x_{\rm min}}^{x_1} \sqrt{r^2-x^2}{\rm d}x - y_{\rm min}\left(x_1 - x_{\rm min}\right)}{(\Delta x)^2} \\
&= \frac{\frac{1}{2}\left(x\sqrt{r^2-x^2} + r^2 \arcsin\frac{x}{r}\right)_{x_{\rm min}}^{x_1} - y_{\rm min}\left(x_1 - x_{\rm min}\right)}{(\Delta x)^2} \\
&= \frac{x_{\rm min} y_{\rm min} - \frac{x_{\rm min} y_1}{2} - \frac{x_1 y_{\rm min}}{2} + \frac{r^2}{2}\left(\arcsin\frac{x_1}{r} - \arcsin\frac{x_{\rm min}}{r}\right)}{(\Delta x)^2}
\end{aligned}
$$

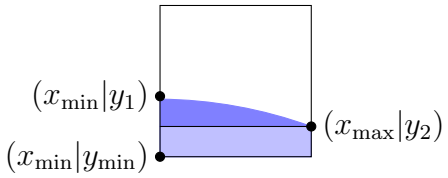


C.1.2 2D: 2 corners inside the feedback region

If there are two corners of the 2D cell inside the feedback region, these corners are $(x_{\rm min}|y_{\rm min})$ and $(x_{\rm max}|y_{\rm min})$ or $(x_{\rm min}|y_{\rm max})$. In the case $x_{\rm min} > y_{\rm min}$ the x and y coordinates are swapped to get an

x-integral. The fraction the cell volume inside the feedback region ($p_{\rm fb}$) can be calculated with:

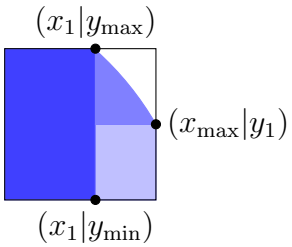
$$\begin{aligned} p_{\rm fb} &= \frac{\int_{x_{\rm min}}^{x_{\rm max}} {\rm d}x \int_{y_{\rm min}}^{\sqrt{r^2-x^2}} {\rm d}y}{V_{\rm cell}} \\ &= \frac{\int_{x_{\rm min}}^{x_{\rm max}} \sqrt{r^2-x^2}{\rm d}x - y_{\rm min}\Delta x}{(\Delta x)^2} \\ &= \frac{\frac{1}{2}\left(x\sqrt{r^2-x^2} + r^2 \arcsin\frac{x}{r}\right)_{x_{\rm min}}^{x_{\rm max}} - y_{\rm min}\Delta x}{(\Delta x)^2} \\ &= \frac{\frac{x_{\rm max}y_2}{2} - \frac{x_{\rm min}y_1}{2} + \frac{r^2}{2}\left(\arcsin\frac{x_{\rm max}}{r} - \arcsin\frac{x_{\rm min}}{r}\right) - y_{\rm min}\Delta x}{(\Delta x)^2} \end{aligned}$$



C.1.3 2D: 3 corners inside the feedback region

If only the corner $(x_{\rm max}|y_{\rm max})$ lies outside the feedback region, $p_{\rm fb}$ can be calculated with:

$$\begin{aligned} p_{\rm fb} &= \frac{\int_{x_1}^{x_{\rm max}} {\rm d}x \int_{y_{\rm min}}^{\sqrt{r^2-x^2}} {\rm d}y + (x_1 - x_{\rm min})\Delta x}{V_{\rm cell}} \\ &= \frac{\int_{x_1}^{x_{\rm max}} \sqrt{r^2-x^2}{\rm d}x - y_{\rm min}\left(x_{\rm max} - x_1\right) + (x_1 - x_{\rm min})\Delta x}{(\Delta x)^2} \\ &= \frac{\frac{1}{2}\left(x\sqrt{r^2-x^2} + r^2 \arcsin\frac{x}{r}\right)_{x_1}^{x_{\rm max}} - y_{\rm min}\left(x_{\rm max} - x_1\right) + (x_1 - x_{\rm min})\Delta x}{(\Delta x)^2} \\ &= \frac{\frac{x_{\rm max}y_1}{2} - \frac{x_1 y_{\rm max}}{2} + \frac{r^2}{2}\left(\arcsin\frac{x_{\rm max}}{r} - \arcsin\frac{x_1}{r}\right) - y_{\rm min}\left(x_{\rm max} - x_1\right) + (x_1 - x_{\rm min})\Delta x}{(\Delta x)^2} \end{aligned}$$



In 3D the percentage of the cell inside the feedback region is calculated with Monte-Carlo if it is not a trivial case (0% or 100%). For all three directions ν random variables are calculated. The fraction the cell volume inside the feedback region $p_{\rm fb}$ is the number of random points inside the feedback region ($|(x_i|y_i|z_i)| < r$) divided by the total number of random points n.

C.1.4 Integral for 3D feedback region boundary cells

$$
\begin{aligned}
x &= [x_{\min}, x_{\max}], x_{\max} = \sqrt{R^2 - y_{\min}^2 - z_{\min}^2} \\
y &= [y_{\min}, y_{\max}], y_{\max} = \sqrt{R^2 - x^2 - z_{\min}^2} \\
z &= [z_{\min}, z_{\max}], z_{\max} = \sqrt{R^2 - x^2 - y^2}
\end{aligned}
$$

$$
\begin{aligned}
p_{\text{fb}} &= \int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} \int_{z_{\min}}^{z_{\max}} \mathrm{d}z\mathrm{d}y\mathrm{d}x \\
&= \int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} \left(\sqrt{R^2 - x^2 - y^2} - z_{\min} \right) \mathrm{d}y\mathrm{d}x \\
&\qquad \text{with Integral 113 in Netz (1986)} \qquad \int \sqrt{a^2 - x^2}\mathrm{d}x = \frac{1}{2}\left(x\sqrt{a^2 - x^2} + a^2 \arcsin\frac{x}{a} \right) \\
&= \int_{x_{\min}}^{x_{\max}} \left(\frac{1}{2}\left(y\sqrt{R^2 - x^2 - y^2} + (R^2 - x^2)\arcsin\frac{y}{\sqrt{R^2 - x^2}} \right) - z_{\min} \right)_{y_{\min}}^{y_{\max}} \mathrm{d}x
\end{aligned}
$$

$$
\begin{aligned}
p_{\text{fb}} = \frac{1}{2}\int_{x_{\min}}^{x_{\max}} \Bigg(& \sqrt{R^2 - x^2 - z_{\min}^2}\sqrt{R^2 - x^2 - R^2 + x^2 + z_{\min}^2} - y_{\min}\sqrt{R^2 - x^2 - y_{\min}^2} \\
+ \; & (R^2 - x^2)\arcsin\frac{\sqrt{R^2 - x^2 - z_{\min}^2}}{\sqrt{R^2 - x^2}} - (R^2 - x^2)\arcsin\frac{y_{\min}}{\sqrt{R^2 - x^2}} \\
& - 2z_{\min}\sqrt{R^2 - x^2 - z_{\min}^2} + 2z_{\min}y_{\min} \Bigg) \mathrm{d}x
\end{aligned}
$$

$$
\text{with } \arcsin\sqrt{1 - \left(\frac{z_{\min}}{\sqrt{R^2 - x^2}}\right)^2} = \arccos\frac{z_{\min}}{\sqrt{R^2 - x^2}} \qquad \text{for } \left(\frac{z_{\min}}{\sqrt{R^2 - x^2}} \geq 0\right)
$$

$$
\begin{aligned}
= \; & \frac{1}{2}\int_{x_{\min}}^{x_{\max}} \Bigg((R^2 - x^2)\left(\arccos\frac{z_{\min}}{\sqrt{R^2 - x^2}} - \arcsin\frac{y_{\min}}{\sqrt{R^2 - x^2}} \right) \\
& - y_{\min}\sqrt{R^2 - x^2 - y_{\min}^2} - z_{\min}\sqrt{R^2 - x^2 - z_{\min}^2} + 2z_{\min}y_{\min} \Bigg) \mathrm{d}x
\end{aligned}
$$

Wolfram Mathematica online integrator (http://integrals.wolfram.com/index.jsp):

$$
\begin{aligned}
& \int (R^2 - x^2)\left(\arccos\frac{z}{\sqrt{R^2 - x^2}} - \arcsin\frac{y}{\sqrt{R^2 - x^2}} \right) dx = \\
& -\frac{x(3R^2 - x^2)}{3}\left(\arcsin\frac{y}{\sqrt{R^2 - x^2}} - \arccos\frac{z}{\sqrt{R^2 - x^2}} \right) \\
& + \frac{2R^3}{3}\arctan\frac{xy}{R\sqrt{R^2 - x^2 - y^2}} + \frac{2R^3}{3}\arctan\frac{xz}{R\sqrt{R^2 - x^2 - z^2}} \\
& -\frac{xy}{6}\sqrt{R^2 - x^2 - y^2} - \frac{y}{6}(3R^2 + y^2)\arcsin\frac{x}{\sqrt{R^2 - y^2}} \\
& -\frac{xz}{6}\sqrt{R^2 - x^2 - z^2} - \frac{z}{6}(3R^2 + z^2)\arcsin\frac{x}{\sqrt{R^2 - z^2}}
\end{aligned}
$$

$$
\begin{aligned}
p_{\mathrm{fb}} = \frac{1}{12}\Bigg(&-2x(3R^2-x^2)\left(\arcsin\frac{y_{\min}}{\sqrt{R^2-x^2}}-\arccos\frac{z_{\min}}{\sqrt{R^2-x^2}}\right)\\
&+4R^3\arctan\frac{xy_{\min}}{R\sqrt{R^2-x^2-y_{\min}^2}}+4R^3\arctan\frac{xz_{\min}}{R\sqrt{R^2-x^2-z_{\min}^2}}\\
&-xy_{\min}\sqrt{R^2-x^2-y_{\min}^2}-y_{\min}(3R^2+y_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-y_{\min}^2}}\\
&-xz_{\min}\sqrt{R^2-x^2-z_{\min}^2}-z_{\min}(3R^2+z_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-z_{\min}^2}}\\
&-3xy_{\min}\sqrt{R^2-x^2-y_{\min}^2}-3y_{\min}(R^2-y_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-y_{\min}^2}}\\
&-3xz_{\min}\sqrt{R^2-x^2-z_{\min}^2}-3z_{\min}(R^2-z_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-z_{\min}^2}}\\
&+12z_{\min}y_{\min}x\Bigg)_{x_{\min}}^{x_{\max}}
\end{aligned}
\tag{C.1}
$$

$$
\begin{aligned}
= \frac{1}{12}\Bigg(&-2x(3R^2-x^2)\left(\arcsin\frac{y_{\min}}{\sqrt{R^2-x^2}}-\arccos\frac{z_{\min}}{\sqrt{R^2-x^2}}\right)\\
&+4R^3\arctan\frac{xy_{\min}}{R\sqrt{R^2-x^2-y_{\min}^2}}+4R^3\arctan\frac{xz_{\min}}{R\sqrt{R^2-x^2-z_{\min}^2}}\\
&-4xy_{\min}\sqrt{R^2-x^2-y_{\min}^2}-y_{\min}(6R^2-2y_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-y_{\min}^2}}\\
&-4xz_{\min}\sqrt{R^2-x^2-z_{\min}^2}-z_{\min}(6R^2-2z_{\min}^2)\arcsin\frac{x}{\sqrt{R^2-z_{\min}^2}}\\
&+12z_{\min}y_{\min}x\Bigg)_{x_{\min}}^{x_{\max}}
\end{aligned}
\tag{C.2}
$$

$$
\begin{aligned}
= \frac{1}{6}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}(2R^2 + y_{\min}^2 + z_{\min}^2)\left(\arccos\frac{z_{\min}}{\sqrt{y_{\min}^2 + z_{\min}^2}} - \arcsin\frac{y_{\min}}{\sqrt{y_{\min}^2 + z_{\min}^2}}\right) \\
+\frac{R^3}{3}\arctan\frac{y_{\min}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{Rz_{\min}} + \frac{R^3}{3}\arctan\frac{z_{\min}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{Ry_{\min}} \\
-\frac{y_{\min}}{3}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}\sqrt{R^2 - (R^2 - y_{\min}^2 - z_{\min}^2) - y_{\min}^2} \\
-\frac{y_{\min}}{6}(3R^2 - y_{\min}^2)\arcsin\frac{\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{\sqrt{R^2 - y_{\min}^2}} \\
-\frac{z_{\min}}{3}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}\sqrt{R^2 - (R^2 - y_{\min}^2 - z_{\min}^2) - z_{\min}^2} \\
-\frac{z_{\min}}{6}(3R^2 - z_{\min}^2)\arcsin\frac{\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{\sqrt{R^2 - z_{\min}^2}} \\
+y_{\min}z_{\min}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2} \\
+\frac{x_{\min}}{6}(3R^2 - x_{\min}^2)\left(\arcsin\frac{y_{\min}}{\sqrt{R^2 - x_{\min}^2}} - \arccos\frac{z_{\min}}{\sqrt{R^2 - x_{\min}^2}}\right) \\
-\frac{R^3}{3}\arctan\frac{x_{\min}y_{\min}}{R\sqrt{R^2 - x_{\min}^2 - y_{\min}^2}} - \frac{R^3}{3}\arctan\frac{x_{\min}z_{\min}}{R\sqrt{R^2 - x_{\min}^2 - z_{\min}^2}} \\
+\frac{x_{\min}y_{\min}}{3}\sqrt{R^2 - x_{\min}^2 - y_{\min}^2} + \frac{y_{\min}}{6}(3R^2 - y_{\min}^2)\arcsin\frac{x_{\min}}{\sqrt{R^2 - y_{\min}^2}} \\
+\frac{x_{\min}z_{\min}}{3}\sqrt{R^2 - x_{\min}^2 - z_{\min}^2} + \frac{z_{\min}}{6}(3R^2 - z_{\min}^2)\arcsin\frac{x_{\min}}{\sqrt{R^2 - z_{\min}^2}} \\
-z_{\min}y_{\min}x_{\min}
\end{aligned}
\tag{C.3}
$$

$$
\begin{aligned}
= \frac{1}{6}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}(2R^2 + y_{\min}^2 + z_{\min}^2)\left(\arccos\frac{z_{\min}}{\sqrt{y_{\min}^2 + z_{\min}^2}} - \arcsin\frac{y_{\min}}{\sqrt{y_{\min}^2 + z_{\min}^2}}\right) \\
+\frac{R^3}{3}\arctan\frac{y_{\min}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{Rz_{\min}} + \frac{R^3}{3}\arctan\frac{z_{\min}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2}}{Ry_{\min}} \\
-\frac{R^3}{3}\arctan\frac{x_{\min}y_{\min}}{R\sqrt{R^2 - x_{\min}^2 - y_{\min}}^2} - \frac{R^3}{3}\arctan\frac{x_{\min}z_{\min}}{R\sqrt{R^2 - x_{\min}^2 - z_{\min}}^2} \\
+\frac{x_{\min}}{6}(3R^2 - x_{\min}^2)\left(\arcsin\frac{y_{\min}}{\sqrt{R^2 - x_{\min}^2}} - \arccos\frac{z_{\min}}{\sqrt{R^2 - x_{\min}^2}}\right) \\
+\frac{y_{\min}}{6}(3R^2 - y_{\min}^2)\left(\arcsin\frac{x_{\min}}{\sqrt{R^2 - y_{\min}^2}} - \arccos\frac{z_{\min}}{\sqrt{R^2 - y_{\min}^2}}\right) \\
+\frac{z_{\min}}{6}(3R^2 - z_{\min}^2)\left(\arcsin\frac{x_{\min}}{\sqrt{R^2 - z_{\min}^2}} - \arccos\frac{y_{\min}}{\sqrt{R^2 - z_{\min}^2}}\right) \\
+\frac{y_{\min}z_{\min}}{3}\sqrt{R^2 - y_{\min}^2 - z_{\min}^2} + \frac{x_{\min}y_{\min}}{3}\sqrt{R^2 - x_{\min}^2 - y_{\min}^2} + \frac{x_{\min}z_{\min}}{3}\sqrt{R^2 - x_{\min}^2 - z_{\min}^2} \\
-z_{\min}y_{\min}x_{\min}
\end{aligned}
\tag{C.4}
$$

The wind speeds at the borders of the feedback region are expected to be too small to be resolved – hence the kinetic wind energy ($0.5v^2\mathrm{d}m$ [code-mass-unit/code-length-unit/code-time-unit^2] with density increase $\mathrm{d}m$ [code-mass-unit/code-length-unit^3]) is treated as unresolved kinetic energy and hence added to the internal energy (see Code Listing C.1).

If mass loss is used, it is assumed that this gas moves with the bulk speed. Thus, inserting mass usually creates kinetic energy, since the bulk speed in the feedback region is almost always nonzero. To insert the fixed amount of stellar feedback energy the code subtracts this additional kinetic energy (that corresponds to the bulk speed of the cell inside the feedback region and the mass that has been newly inserted into this cell) from the newly inserted internal energy in this cell.

The name of the variable `rhodriver` used in Code Listing C.1 might be misleading – it is used for the density increase due to the mass ejection caused by stellar feedback. The module `driver` (stored in the file `driver.f90`) provides arrays and subroutines to handle the feedback. It is shown in Code Listing C.1 and reads data from a file called "driver" that is located in the local directory. This file has five entries per line that are separated by blanks: time, internal energy gain (per time unit and for the whole feedback region), mass loss (per time unit and for the whole feedback region), wind speed and ^{26}Al yields (percentage of mass loss). The number of lines in the driver file can either be specified or determined by the code at run-time. As already mentioned, the code has two ways of adding internal energy: internal energy (column 2 in the driver file) that is not connected to mass loss of the feedback region and unresolved kinetic energy (computed from column 3 and 4 in the driver file) that takes mass feeding into account.

The shape (slightly asymmetric feedback regions, 4-symmetric feedback regions) and scaling of the stellar feedback (energy, mass) is explained in Sect. 4.3 on Sedov-Taylor blasts and in Sect. 4.4.1 on constant winds.