Wang, Long, Spurzem, Rainer, Aarseth, Sverre, Nitadori, Keigo, Berczik, Peter, Kouwenhoven, M. B. N., & Naab, Thorsten. 2015. NBODY6++GPU: ready for the gravitational million-body problem. *MNRAS* , **450**(4), 4070–4080.

Whitehead, Alfred J., McMillan, Stephen L. W., Vesperini, Enrico, & Portegies Zwart, Simon. 2013. Simulating Star Clusters with the AMUSE Software Framework. I. Dependence of Cluster Lifetimes on Model Assumptions and Cluster Dissolution Modes. *ApJ* , **778**(2), 118.

Wisdom, Jack, & Holman, Matthew. 1991. Symplectic maps for the N-body problem. *AJ* , **102**(Oct.), 1528–1538.

Yoshida, Haruo. 1990. Construction of higher order symplectic integrators. *Physics Letters A*, **150**(5-7), 262–268.

Yoshida, Haruo. 1993. Recent Progress in the Theory and Application of Symplectic Integrators. *Celestial Mechanics and Dynamical Astronomy*, **56**(1-2), 27–43.

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

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

Chapter 4

Stellar Structure and Evolution

> How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?
>
> Sir Arthur Conan Doyle

Stars evolve. From the moment of birth, a star's internal structure and appearance change with time. Some of these changes occur very slowly and hardly affect the local environment, but others are swift and have far-reaching repercussions. In this chapter, we review the basic evolution of stars and show how relatively simple stellar evolution calculations can be conducted.

4.1 In a Nutshell

Stellar evolution is a very different game from Newtonian gravity. Whereas the physical equations underlying gravitational dynamics can be described and solved using basically high-school mathematics, solving the equations for stellar structure and evolution is challenging even for experts. In this chapter, we present an overview of stellar structure and evolution, although it will barely be enough to give an appreciation of the complexities of stellar evolution codes. We start by reviewing some fundamental time scales in stellar evolution, followed by a rudimentary review of the structure equations and some prescriptions for stellar evolution. More extensive discussion can be found in Kippenhahn & Weigert (1967) and Eggleton (2006).

A star is a gravity-confined nuclear fusion reactor (Bethe, 1939). Stars are born when parts of a molecular cloud are compressed by turbulence or other external forces (such as supernovae), collapse under their own weight, and fragment into a collection of prestellar clumps. These clumps continue to contract and accrete, and their central temperatures increase. Some of the excess heat is radiated away, but some of it goes to increase the pressure, ultimately halting the gravitational contraction. These hydrodynamical processes are discussed in more detail in Chapter 5. A clump becomes a star when its central temperature and density reaches the point where nuclear fusion can start. This moment is generally associated with the birth of the star. At this point, the star is said to lie on the *zero-age main sequence*, or ZAMS. From then on, the star

evolves, as nuclear fusion in its core[1] drives irreversible structural changes.

Figure 4.1 shows images of two red supergiant stars, Antares (Ohnaka *et al.*, 2017) and Betelgeuse (Haubois *et al.*, 2009), showing their complex surface temperature structure. Such observations have only recently become possible. A more theoretically oriented view of the internal structure of three stars of different masses is presented in Figure 4.2.

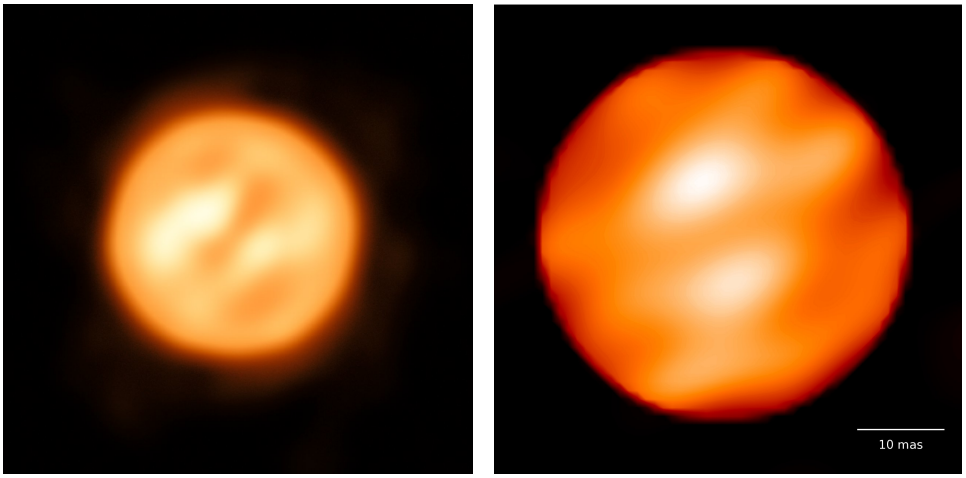

Figure 4.1: Stars are finite objects with structure on their surface, as demonstrated here with the observations of Antares (left, image with the ESA VLTI; reprinted with permission from Ohnaka *et al.*, 2017) and Betelgeuse (right, image by Observatoire de Paris; reprinted with permission from Haubois *et al.*, 2009). Both stars are similar in size—about 4.1–5.5 au (Dolan *et al.*, 2016). For comparison, we also show an image of the nearest white dwarf, Sirius B, with a size of 0.008 $R_\odot$, next to the two red giants.

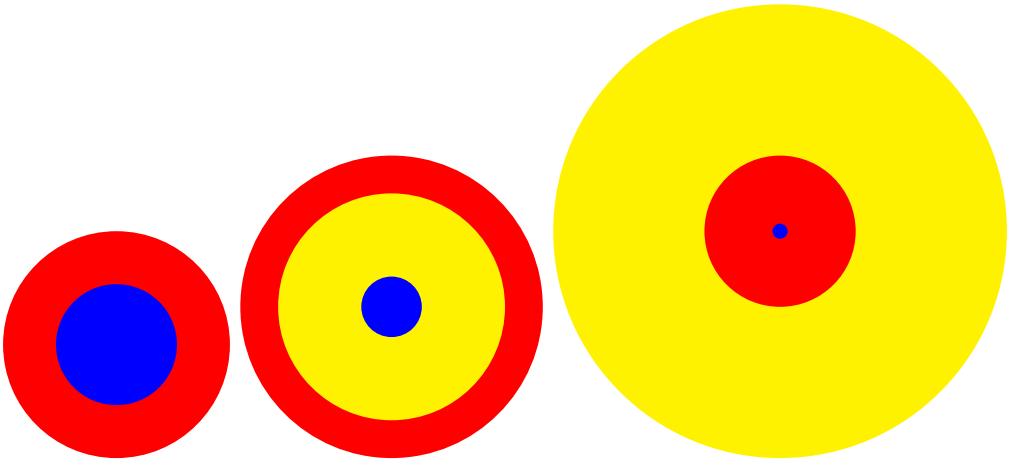

Figure 4.2: Schematic structure of a low-mass star (left), an intermediate-mass star (middle) and a high-mass star (right). They are not to scale, but the various colored regions in the stellar interior indicate the nuclear-burning region (blue), the convective zone (red), and the radiation-dominated region (yellow).

[1]The stellar core is the central portion of the star where nuclear fusion occurs. The transition from fusion to non-fusion is typically quite sharp; see Figure 4.4.

During its lifetime, a star experiences several evolutionary phases, each associated with distinct burning regimes in the central nuclear furnace. This evolution is often presented as a trajectory in temperature–luminosity space, in what is called a Hertzsprung–Russell (H–R) diagram. Figure 4.3 is an H–R diagram illustrating the evolution of three stars of different masses. Colors indicate various stellar evolutionary stages. Very roughly speaking, we can divide the evolution of a star into three distinct phases: the main sequence, the post-main sequence (or giant phase), and the remnant phase, when the star has turned into a compact object. Along the main sequence, stars are generally comparable in size to the Sun. Giants can be much larger—as large as several astronomical units (see Figure 4.1). Remnants are much smaller than the Sun—comparable to or much smaller than Earth.

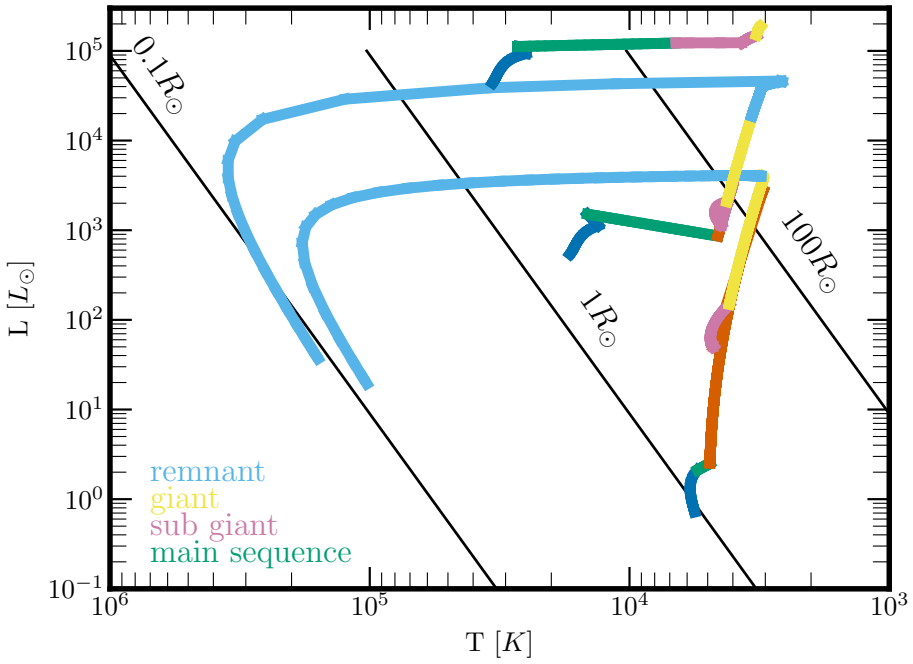

Figure 4.3: Hertzsprung–Russell diagrams (luminosity versus temperature) for three stars of masses 1 $M_\odot$ (bottom set curves), 5 $M_\odot$, and 20 $M_\odot$ (top curves), from the zero-age main sequence to the moments they turn into compact remnants. The colored text in the bottom left identifies the evolutionary stage of the star along its track (with the caveat that a star at the tip of the asymptotic giant branch should probably not be identified as a remnant). The black diagonal lines indicate the run of luminosity and temperature for blackbodies with radii 0.01 $R_\odot$ (left), 1 $R_\odot$, and 100 $R_\odot$ (rightmost line). The figure was made using the script amuse.examples.stellar.plot_stellar_evolution_track.

To guide the eye in Figure 4.3, we plot lines of constant stellar radius r. The radius can be estimated from the star's effective temperature ($T_{\rm eff} \approx T$) and luminosity by approximating the stellar spectrum as a blackbody, resulting in

$$L = 4\pi r^2 \sigma T_{\rm eff}^4, \tag{4.1}$$

where σ is the Stefan–Boltzmann constant. This is a significant simplification because the stellar spectrum is generally not described particularly well as a blackbody, but it is a reasonable first approximation to the true state of the star.

4.1.1 Stellar Time Scales

Three fundamental time scales are relevant for stellar evolution. These are the *dynamical time scale* $t_{\rm dyn}$, the *Kelvin–Helmholtz time scale* $t_{\rm KH}$, and the *nuclear time scale* $t_{\rm nuc}$ (Bradt, 2014). They are time scales on which distinctly different physical processes operate.

4.1.1.1 The Dynamical Time Scale

The stellar analog to the stellar dynamical time scale is defined analogously to the gravitational dynamical time scale discussed in Section 3.1.2. For a star of mass m and radius r, the dynamical time scale is

$$t_{\rm dyn} = \sqrt{\frac{r^3}{Gm}}. \tag{4.2}$$

This is the dynamical (orbital) time scale for a particle orbiting at the stellar surface, but it is also (approximately) the time taken for a sound wave to cross the star. The dynamical time scale for the Sun (m = 1 | units.MSun and r = 1 | units.RSun) can be calculated as follows:

```
def dynamical_time_scale(m, r, G=constants.G):
return numpy.sqrt(r**3/(constants.G*m))

print 'solar dynamical time scale =', \
dynamical_time_scale(1|units.MSun, 1|units.RSun).in_(units.minute)
```

which turns out to be about $t_{\rm dyn} \simeq 27$ minutes. If a comet hits the Sun, the other side of the star learns about the impact in about half an hour.

The related *convective time scale* $t_{\rm conv}$ is the time scale on which material convectively bubbles up from the stellar interior toward the surface. In convection, the stellar luminosity is carried outward by physical motion of the stellar gas rather than by radiation. It can be characterized by the mean distance Δr traveled by a typical rising gas parcel (usually assumed to be related to the local pressure scale height $P/\mid \nabla P \mid$) and the mean speed $\bar{v}$ with which the parcel rises:

$$t_{\rm conv} = \frac{\Delta r}{\bar{v}}. \tag{4.3}$$

For the Sun, $\Delta r \simeq 0.1 {\rm R}_\odot$ and $\bar{v} \simeq 11$ m s^{-1}. We then obtain $t_{\rm conv} \simeq 0.2$ years.

4.1.1.2 The Thermal Time Scale

The Kelvin–Helmholtz (or thermal) time scale is the time scale on which thermal energy leaves a star. It can be written as

$$t_{\rm KH} = \frac{E_{\rm th}}{L}, \tag{4.4}$$

where $E_{\rm th}$ is the star's total thermal energy and L is its luminosity, the total energy leaving the stellar surface per unit time.

Like star clusters, stars in equilibrium (i.e., having no bulk motion) also are governed by the virial theorem (see Equation (4.5)), with the total thermal energy E_{th} replacing the total kinetic energy T (Binney & Tremaine, 2008):

$$2E_{\rm th} + U = 0. \tag{4.5}$$

Thus, neglecting (as usual) extraneous factors of order unity, we can write the Kelvin–Helmholtz time scale as

$$t_{\rm KH} = \frac{Gm^2}{rL}. \tag{4.6}$$

For the Sun (m = 1 | units.MSun, r = 1 | units.RSun and L = 1 | units.LSun), we have

```
def Kelvin_Helmholtz_timescale(m, r, L):
return constants.G*m**2/(r*L)
```

in which case $t_{\rm KH} \simeq 31$ Myr.

4.1.1.3 Nuclear Fusion and the Nuclear Time Scale

The collapse of a protostellar clump toward the main sequence proceeds on a Kelvin–Helmholtz time scale. During its slow contraction, a star follows the pre-main sequence or *Hayashi track* (Hayashi, 1961). During this period, the star decreases in luminosity but stays at roughly the same surface temperature. The central temperature and pressure increase until nuclear fusion begins to contribute considerably to the total energy budget. The central density and temperature then exceed $\rho_c \gtrsim 7\,{\rm g/cm^3}$ and $T_{\rm c} \gtrsim 10^7\,{\rm K}$.

On the main sequence, where the star spends most of its lifetime, hydrogen (H) fuses into helium (He) through a variety of channels that may be summarized schematically by

$$4(^1{\rm H}) \;\rightarrow\; ^4{\rm He} \;+\; 2\nu_e \;+\; 26.7\,{\rm MeV}, \tag{4.7}$$

where $\nu_{\rm e}$ represents an electron neutrino that streams out of the core, through the star and into interplanetary space. The 26.7 MeV represents the net energy liberated as a result of this reaction. Overall, a fraction $\epsilon \simeq 0.7\%$ of the total nuclear mass is converted into energy in the form of gamma-rays. That energy ultimately contributes to the star's luminosity. Assuming that the innermost 10% of the star's mass is fusing, and assuming the above efficiency, we can write a time scale for nuclear fusion in a main sequence star:

$$t_{\rm nuc} \;=\; 7\times10^{-3}\,\frac{0.1mc^2}{L} \;=\; 7\times10^{-4}\,\frac{mc^2}{L}. \tag{4.8}$$

In AMUSE, a simple estimate of the nuclear time scale for the Sun (m = 1 | units.MSun and L = 1 | units.LSun) gives

```
def nuclear_time_scale(l, L):
    return 7e-4 * m * constants.c**2/L
```

This shows that the Sun has $t_{\rm nuc} \simeq 10$ Gyr. This time scale is the Sun's approximate nuclear burning lifetime. More careful calculations yield 12.462 ± 0.002 Gyr (based on calculations you can do yourself in Section 4.4).

Once the hydrogen in the stellar core has fused into helium, a star may then fuse helium into carbon (C):

$$3^4\mathrm{He} \rightarrow {}^{12}\mathrm{C} \; + \; 7.3\,\mathrm{MeV} \tag{4.9}$$

(the reaction is somewhat more complicated than indicated here, but this is the outcome). Subsequent fusion stages form oxygen (O) and heavier elements (such as Neon, Ne) by alpha capture:

$$^4\mathrm{He} + {}^{12}\mathrm{C} \;\; \rightarrow \;\; {}^{16}\mathrm{O} \; + \; 7.2\,\mathrm{MeV}. \tag{4.10}$$

$$^4\mathrm{He} + {}^{16}\mathrm{O} \;\; \rightarrow \;\; {}^{20}\mathrm{Ne} \; + \; 4.7\,\mathrm{MeV}, \tag{4.11}$$

etc. This progression is sometimes called the *alpha ladder*. How far reactions proceed depends on the mass of the star. The Sun will never make it past oxygen, but the most massive stars will keep producing heavier elements up to iron and nickel. Fusion eventually stops at nickel (^{56}Ni), after which the reaction becomes endothermic: ^{56}Ni does not produce energy when fused.

Figure 4.4 presents the luminosity generated in the stellar interior as a function of time for three stellar masses and at two different moments in time. In each of these cases, the core is easily seen in the sudden drops in density and luminosity.

4.1.2 Physics of the Interior

Unlike the case with self-gravitating systems, we cannot easily solve the fundamental physical equations governing stellar structure and evolution. That would require us to resolve detailed nuclear reaction networks, the ways in which matter and energy flow around the star, and the details of how radiation interacts with matter.

The cross sections for interactions between radiation and matter are in fact quite well known, and nuclear reaction rates can in principle be measured in the laboratory. The reaction rates used in stellar evolution calculations are generally extrapolated from laboratory measurements made at higher temperatures and much lower densities. The fusion reactor ITER[2] will have an operational temperature of 1.5×10^8 K and an electron number density of $n_e \sim 10^{18} - 10^{21}$ m^{-3}, corresponding to an equivalent hydrogen density of $\sim 10^{-12} - 10^{-9}$ g cm^{-3}. For comparison, the temperature and density of the solar core are roughly 1.5×10^7 K and 150 g cm^{-3}, respectively.

4.1.2.1 Underlying Assumptions in Stellar Evolution Models

Like many gravity solvers, stellar evolution codes make a number of simplifying assumptions. Some are hidden in the mathematical or numerical models, while others

[2] See https://www.iter.org.

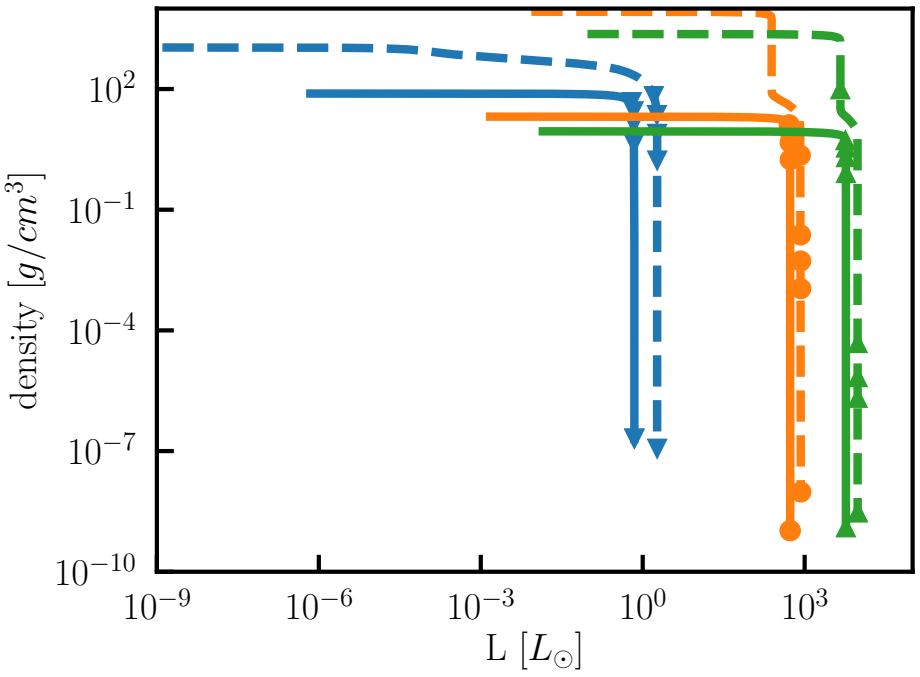

Figure 4.4: Density as a function of total luminosity in the stellar interior, showing the difference between the stellar core and envelope. Results are shown for three stellar masses: 1 $M_\odot$ (blue), 5 $M_\odot$ (red), and 10 $M_\odot$ (yellow). The solid curves are for stellar ages of 1 Myr; the dashed curves are for 11 Gyr, 90 Myr and 20 Myr, for the 1 $M_\odot$, the 5 $M_\odot$, and 10 $M_\odot$ stars, respectively. Bullets and triangles are drawn at 20% intervals of the enclosed mass of the star. The script that produced this figure is `amuse.examples.stellar.plot_stellar_structure`. It should take a couple of minutes to run on a laptop.

are more explicit. Instead of solving for the microphysics, we generally solve for the macroscopic state of the star, which can only be realized by neglecting several aspects of the physics. The neglect of these effects does not (seem to) affect the zeroth-order calculation for most stars, and to some extent can be included as a first-order correction. Here, we list some of the most important assumptions.

1. **Spherical symmetry**. The assumption of spherical symmetry is a sweeping approximation that reduces the stellar structure problem from a complex three-dimensional set of partial differential equations to a one-dimensional set of ordinary differential equations. In the process, we lose the possibility of including stellar rotation, internal magnetic activity, and any other non-radial internal motion. In this approximation, the star is viewed as a series of 1D arrays of macroscopic parameters, such as density, temperature, mean molecular weight, etc., stored in radial bins running from the center to the surface.

   This greatly simplifies the calculation, but most actual stars are observed to have magnetic fields of varying strengths, do rotate, and exhibit a wealth of non-spherical features. One only has to look at an image of the Sun, with its many sunspots and prominences, to realize that spherical symmetry is only an approximation (see Figure 4.5). (On the other hand, solar activity—even at its maximum—is only a tiny fraction of the Sun's energy budget.)

2. **Local thermodynamic equilibrium**. Radiation plays an important role in transporting energy through a star. Let's temporarily ignore the possibility of convection (see below), and assume that all energy generated in the stellar core moves through the stellar interior in the form of radiation. If the total luminosity within a sphere of radius r is $L(r)$, then the energy flux across radius r is

$$F(r) = \frac{L(r)}{4\pi r^2}. \tag{4.12}$$

   In equilibrium, all the luminosity produced inside the star eventually reaches the surface and escapes into space.

   The outward energy flux F is driven by a temperature gradient within the star. Basically, if the temperature decreases with radius, then at any point in the star, there will be slightly more photons moving outward from within than moving inward from outside. However, a system in thermodynamic equilibrium would have constant temperature throughout, and therefore cannot have temperature gradients. Therefore, a star cannot be in perfect thermodynamic equilibrium. Instead, we must assume that the star is very close to thermodynamic equilibrium at any radius, but there is a slight radial temperature gradient driving the outward flux. This assumption is called *local thermodynamic equilibrium*.

   The mean free path of a photon is much smaller than the stellar radius, so each photon interacts frequently with local stellar material on its way out. This supports the assumption that the gas and the radiation are in local thermodynamic equilibrium: the radiation and gas temperatures are the same at every point in the star, even though the temperature decreases with radius.

3. **Convection and mixing**. Radiation is not the only energy transport mechanism operating in most stars. Convection is another important mechanism that plays a vital role in determining stellar structure and evolution. Stars of very low mass tend to be fully convective, the Sun has a convective envelope, and massive stars tend to be convective in their central regions. The imprint of convection on the surface of the Sun is nicely visible in Figure 4.5.

   When the energy flux through part of a star cannot be transported efficiently by radiation (for example, when the flux becomes very large, or in a region where the opacity is very high), the energy is instead transported by convection—turbulent motion in the stellar interior. Stellar material is heated, rises, cools, and sinks back, forming convective cells that clearly cannot be properly described in a 1D spherical stellar model: How can two parcels of gas pass one another on a line?

   Turbulence gives rise to many numerical complications and is not realistically resolved in current state-of-the-art 1D stellar structure and evolution models. Rather, it is handled in an ad hoc fashion, assuming that convective velocities are sufficient to carry the required energy flux and that the radial extent of a convective cell is a fixed fraction of the pressure scale height, $\Delta r = \alpha P/|dP/dr|$,

where α is a parameter that can be adjusted to calibrate models against the Sun. It is generally assumed that any region of a star that becomes convectively unstable is mixed by the convection process.

Convection is particularly important in the earliest stellar evolution phases (such as along the Hayashi track before the star reaches the main sequence) and in much later phases (for example, when the star reaches its maximum radius along the asymptotic giant branch), but some regions of most stars will become convectively unstable at some time during their evolution.

Stellar material can be efficiently redistributed by convection. For example, nuclear-processed elements can be transported to the stellar surface, or fresh hydrogen from the stellar envelope can be brought into the core. The latter process is generally identified with convective overshoot (where the momentum of the convected material carries it beyond the unstable region, into the overlying stable gas) and has the effect of prolonging the lifetime of a star.

4. **Mass loss by stellar winds**. In addition to radiating energy into space, stars also lose mass. The amount of mass lost is very hard to calculate from first principles. Consequently, stellar winds are generally implemented as ad hoc prescriptions calibrated against observations (which are also uncertain). One problem here is that the stars that lose the most mass generally have the shortest lifetimes. As a result, the number of examples against which approximate methods can be calibrated is quite limited. Our imperfect theoretical understanding of the physics and the expense of a detailed numerical prescription means that mass loss in stellar evolution codes is often parameterized, rather than solved explicitly.

5. **Equation of state**. Although it is not, strictly speaking, an assumption, any calculation of stellar structure requires an equation of state to relate the gas pressure (P) to the temperature (T) and density (ρ).

   In many cases, an ideal gas equation of state may be used. This relation is simple: at sufficiently high temperatures, the gas is fully ionized and is described well as an ideal gas, in which case (Clapeyron, 1834; Clausius, 1857)

$$P = nkT = \frac{\rho kT}{\mu}. \tag{4.13}$$

   Here, n is the number of particles per unit volume, k is the Boltzmann constant, and μ is the mean molecular weight. For a fully ionized gas with hydrogen fraction (by mass) X, helium fraction Y, and heavy element fraction $Z = 1 - X - Y$, because each hydrogen atom contributes one electron, each helium atom contributes two electrons, and (we assume) each heavy atom of atomic weight A contributes on average $A/2$ electrons, the total number density is

$$n = \frac{\rho}{m_H}\left(2X + \tfrac{3}{4}Y + \tfrac{1}{2}Z\right), \tag{4.14}$$

where $m_{\rm H}$ is the mass of a hydrogen atom. Setting $Z = 1 - X - Y$, we find

$$\mu = 2m_{\rm H}\left(1 + 3X + \frac{1}{2}Y\right)^{-1}. \tag{4.15}$$

For solar ZAMS composition, $X = 0.7$, $Z = 0.02$, and $\mu \simeq 0.62 m_{\rm H}$.

Other pressure terms may be important in some circumstances. In hot stars, radiation pressure must be taken into account, so the expression for the pressure becomes

$$P = \frac{\rho k T}{\mu} + \frac{1}{3}aT^4, \tag{4.16}$$

where a is the radiation constant. This equation of state is a good description of most of the interiors of most stars for most of their lives. At very high densities, quantum mechanical electron degeneracy pressure may dominate:

$$P_e = \left(\frac{3}{8\pi}\right)^{2/3} \frac{h^2}{5m_e} n_e^{5/3}. \tag{4.17}$$

Here, n_e is the electron number density and m_e is the electron mass. Degeneracy pressure is particularly important in the cores of evolved stars and remnants. Finally, near the stellar surface, the gas may be only partly ionized and the ionization fraction of each species must be computed from the Saha–Langmuir equation (Saha, 1921; Kingdon & Langmuir, 1923). In addition, molecules may form, further complicating the calculation of μ.

4.1.3 Final Stages of Stellar Evolution

A star continually fights its own gravitational pull, but gravity inevitably wins, sooner or later causing the central parts of the star to turn into a compact remnant—a white dwarf, neutron star, or black hole.

In low- and intermediate-mass stars—those with masses at the end of the giant branch of less than about 8 $M_\odot$—the core grows until it reaches a state where electron degeneracy pressure prevents further contraction, and fusion ends somewhere along the alpha ladder (see Section 4.1.1.3). How far the star ascends the alpha ladder depends on the temperature and density in the stellar interior. Figure 4.6 shows the central temperature as a function of the core density. Stars are born in the lower left corner, at low temperature and density. In this regime, the star fuses hydrogen (see Equation (4.7)). As the star ages, its central density and temperature increase, allowing it to ascend the ladder. Only the most massive stars reach the top rung, fusing ^{52}Fe to ^{56}Ni. The portion of the script that carries out the evolutionary calculation is shown in Listing 4.1.

Listing 4.1: Calculating the core temperature and density of a stellar model (used to make Figure 4.6). The full script can be found in amuse.examples.plot_core_temperature_density.

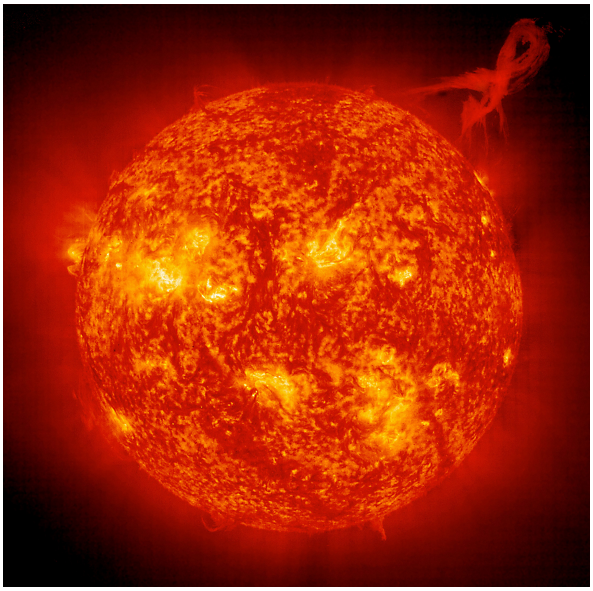

Figure 4.5: Surface image of the Sun (image from the SOHO mission at EIT 304 Å on 1999 September 14). The light and dark orange patches (as opposed to the yellow areas, which are regions of solar activity) show granulation, the result of temperature variations due to convection just below the surface. Image from ESA/NASA/SOHO/EIT Courtesy of SOHO/EIT consortium. SOHO is a project of international cooperation between ESA and NASA.

The core density and temperature of the most massive star in Figure 4.6 grow steadily, whereas the stars of lower mass make some interesting excursions during their evolution. The differences between these tracks are caused by the various energy transport mechanisms and changes in the equation of state in the stellar interior. For the Sun and other low-mass stars, the alpha ladder stops when carbon and helium fuse to form oxygen; for more massive stars, it ends at oxygen and/or neon (see Section 4.1.1.3). Eventually, nuclear fusion stops because (due to electron degeneracy) the contraction of the stellar core cannot produce the density and temperature required to reach the next rung of the alpha ladder. As the star runs out of fuel, instabilities arise and propagate to the outer layers, eventually ejecting them into space in the form of a dense stellar wind. This final phase is called the post-asymptotic giant branch (AGB). The leftover cinder of processed material becomes a white dwarf, the composition of which depends on how far the star ascended the alpha ladder.

The evolution of a higher-mass star ends much more violently. Once all available fuel is spent, and the stellar core is mainly composed of iron-group elements, further fusion is not possible. As the core mass grows due to nuclear fusion in the overlying layers, the only remaining route is collapse. The resulting phenomenon is called a supernova, although this term is also often applied to the core-collapse process. A wide variety of supernovae have been identified observationally. Roughly speaking, we can recognize two basic types, II and I[**b/c – 2018 version**], but many sub-categories have been defined (Smartt, 2009). To make a rough distinction, type II supernovae are the result of core collapse in a massive hydrogen-rich star, as just described, leading to the formation of a neutron star or black hole (Chiu, 1964). Type Ib/c supernovae result from the collapse of helium-rich (Wolf–Rayet) stars and may lead to black hole

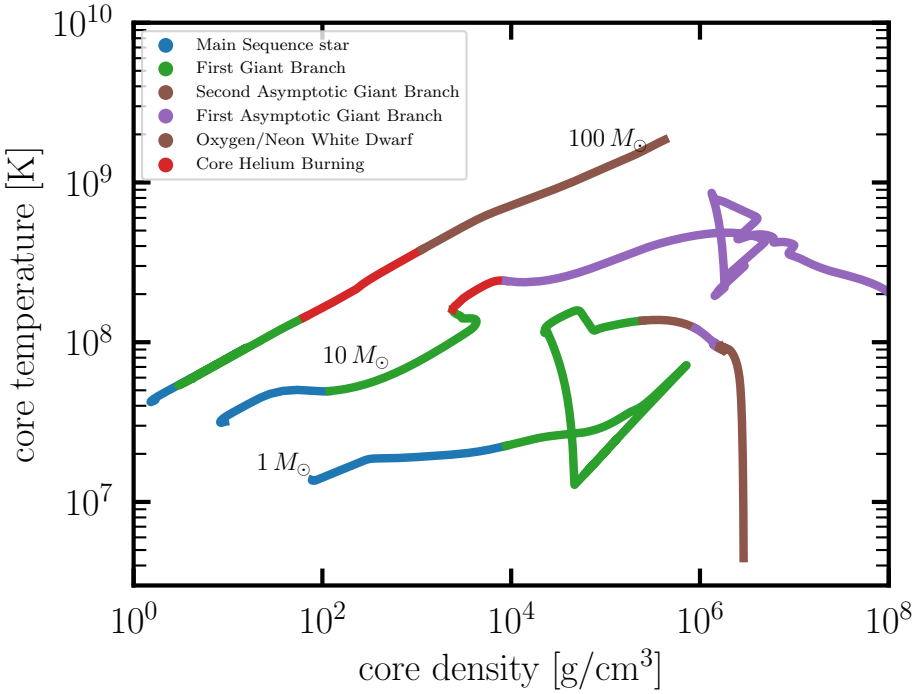

Figure 4.6: Central temperature as a function of core density for a 1 $M_\odot$ (bottom), a 10 $M_\odot$ (middle), and a 100 $M_\odot$ star (top) of solar metallicity. The colors identify various evolutionary phases (see legend), starting with the main sequence at the lower left to giants and white dwarfs to the right. The code to produce this plot can be found in amuse.examples.stellar.calculate_core_temperature_density (see Listing 4.1). To plot the results it may be more practical to use the pre-calculated data in the simple plotting script amuse.examples.stellar.plot_core_temperature_density, because the Henyey stellar evolution code MESA (see Table 4.1) used to generate this figure takes a few hours for each single star. In particular the 1 $M_\odot$ star tends to be slow along the asymptotic giant branch just before the green part of the curve; magnifying this part of the evolution will reveal interesting sinusoidal variations in the core density and core temperature of the star.

formation, although this remains uncertain (Georgy *et al.*, 2009). Type Ia supernovae may mark the collapse of a massive white dwarf, possibly formed via a white dwarf merger.

Supernovae are not necessarily symmetrical, and the compact object may receive a velocity kick as a result of the explosion. This is often referred to as a *natal kick* of the neutron star or black hole in a type Ib/c or type II supernova. These velocity kicks are uncertain, but may be as high as 1000 km s^{-1} (Helfand & Tademaru, 1977; Lyne & Lorimer, 1994).

4.2 Simulating Stellar Evolution

The most detailed stellar evolution codes solve the underlying (1D) differential equations for stellar structure and evolution (for details, see Eggleton, 2006)). They advance the nuclear burning and stellar structure equations at every time step, and by taking energy transport and nuclear energy generation into account, they converge to a time-dependent solution. These methods are accurate, within the simplifying assumptions listed earlier, but relatively slow. As a result, a second class of much more rapid meth-

Table 4.1: Stellar evolution codes incorporated in AMUSE. 1: Paxton *et al.* (2010, 2011); 2: Eggleton *et al.* (2011); Eggleton (1971, 2006); 3: Hurley *et al.* (2000); 4: Portegies Zwart & Verbunt (1996); Portegies Zwart & Yungelson (1998); Toonen *et al.* (2012); Portegies Zwart & Verbunt (2012); 5: (Hut *et al.*, 2003).

name	ref	type of code	language	binary evolution	internal structure	parallel
MESA	1	Henyey	F95	no	yes	yes
EVTWIN	2	Henyey	F77	yes	yes	no
SSE/BSE	3	parametric	F77	yes	no	no
SeBa	4	parametric	C++	yes	no	yes
Toy	5	scaling relations	C++	no	no	no

ods has appeared. They use fits to the more detailed results, generally expressed in the form of look-up tables (Meynet *et al.*, 1994; Meynet & Maeder, 2005; Brott *et al.*, 2011; Köhler *et al.*, 2015), fitting formulae (Eggleton *et al.*, 1989; Hurley *et al.*, 2000), or other parameterizations (Varshavskii & Tutukov, 1975; Tutukov & Fedorova, 2001). We refer to the first class of stellar evolution codes as direct solvers. The second class, we call derived stellar evolution prescriptions. Both are part of AMUSE.

Direct stellar evolution solvers use so-called Henyey stellar evolution models (Henyey *et al.*, 1959, 1964), which provide a 1D representation of a star, using radial distributions of stellar mass, density, composition, etc. Hybrid Henyey codes exist, in which the basic differential equations are expanded to include atmosphere models, rotation, and some form of convective overshoot and mixing, but those models are, strictly speaking, still one-dimensional (Schaerer *et al.*, 1996; Stasińska & Schaerer, 1997). There is currently only one true 3D stellar evolution code (Bazán *et al.*, 2003).

Derived stellar evolution methods come in two varieties: parametric and look-up. Parametric evolution codes use fitting formulae to reproduce the various stages of stellar evolution. They adopt polynomials with mass and sometimes composition—typically expressed as metallicity—as free parameters, and yield the stellar radius, temperature, and luminosity as functions of time since the zero-age main sequence. Parametric stellar evolution models generally have very limited internal structure information. Sometimes they include some notion of a core–envelope structure, and possibly additional parameters such as the radius of gyration, but their structural content is minimal.

Look-up parametric stellar evolution methods (indicated in Table 4.1 as *parametric*) generally interpolate from precomputed tables of stellar evolution tracks (mass, radius, temperature, luminosity, etc). In some sense, the look-up and parametric stellar evolution codes are equivalent, but they differ in their computational requirements in terms of both memory management and CPU performance. In general, however, both are much faster than Henyey codes. The evolution of a 1 $M_\odot$ star from zero-age main sequence until it leaves the asymptotic giant branch takes about a second for a parametric code and up to a day for a Henyey code.

4.2.1 Initial conditions for Stars

A star is born—hydrogen fusion starts—with a certain mass and metallicity, at which point we call it a *zero-age main-sequence star*. For a parameterized stellar evolution code (see Section 4.2) the specification stops there—the future evolution of the star is completely determined via fitting formulae and/or tables.

For a Henyey code (see Section 4.2) these parameters generally also describe the star, but a number of additional parameters must also be specified. In this type of code the nuclear reaction network, the hydrodynamics and the radiative transport are generally solved in a number of shells—a star in a Henyey code is layered like an onion. One of the parameters in such a code is the number of layers (onion shells) to use in the code's description of stellar structure. In general, we must specify the run of temperature, density, etc. throughout the (1-D, spherically symmetric) star. The `MESA` and `EVtwin` stellar evolution codes can generate initial stellar models based on mass and metallicity, but we can also have these codes start from completely arbitrary structures, under our control. It is possible, for example, to start with a pre-main-sequence star, in which case the evolution starts somewhere on the Hayashi track (see Chapter 4). Another interesting initial stellar model is the hydrogen-exhausted core of a giant without an envelope, which could be interpreted as a zero-age helium main-sequence star.

Listing 4.2 presents a simple script to initialize a star of mass M and metallicity z, and plots the associated density profile. Figure 4.7 shows the density structure of an $M = 2\,\mathrm{M}_\odot$ star simulated using the two Henyey codes, `EVtwin` and `MESA`. In addition, we plot the structure of two $1\,\mathrm{M}_\odot$ stars that merge into a single star at a very early stage in their evolution (see Chapter 4 and Listing 4.7 for details). Such a collision would provide a good reason to start with a non-standard stellar initial model. In this case, one could use `MakeMeAMassiveStar` (MMAMS; Lombardi *et al.*, 2002; Gaburov *et al.*, 2008) to generate a stellar initial model, as we will discuss in Chapter 4.

```
13 def get_density_profile(code=Evtwin, mass=2.0 | units.MSun, metallicity=0.02):
14     stellar = code()
15     stellar.parameters.metallicity = metallicity
16     stellar.particles.add_particle(Particle(mass=mass))
17     radius = stellar.particles[0].get_radius_profile()
18     rho = stellar.particles[0].get_density_profile()
19     stellar.stop()
20     return radius, rho
```

Listing 4.2: Script to initialize a single zero-age main-sequence star using EVTwin. Code fragment from `amuse.examples.initialize_single_star`.

It is interesting to note that the profiles calculated by the three codes are not the same. The largest differences are in the outer parts of the stars, and probably these do not make a huge difference to global stellar evolution. But they may affect quite dramatically how the star appears in a Hertzsprung–Russell diagram, which plots the star's temperature against its luminosity (see for example Figure 4.11). A larger star translates to a dimmer and cooler stellar surface.

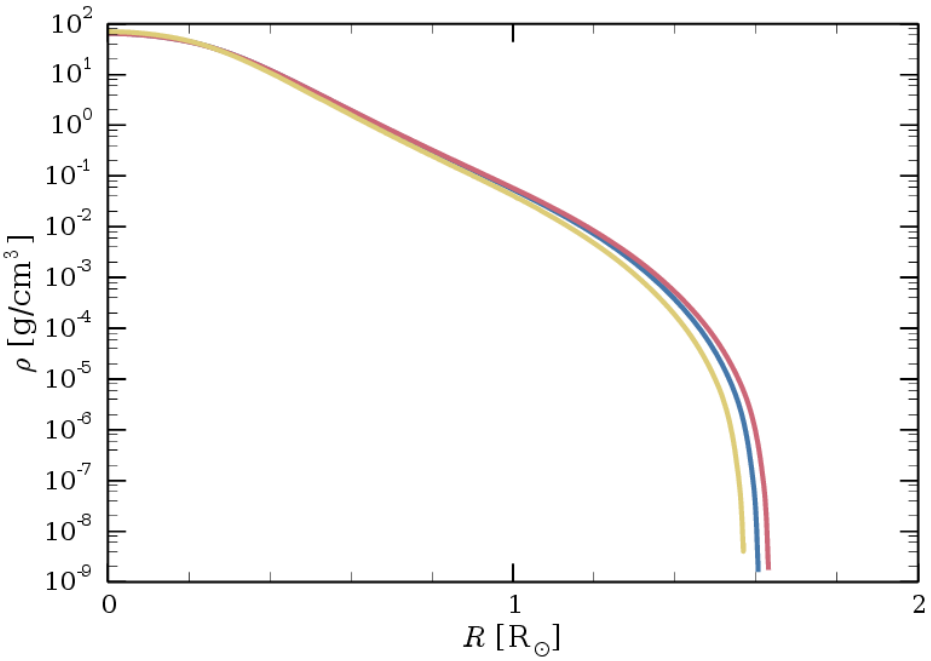

Figure 4.7: Density profile of a star on the zero-age main sequence. All three curves are for a single $2\,M_\odot$ star. Two curves are calculated using the stellar structure and evolution codes `EVTWIN` (blue curve) and `MESA` (red). The yellow curve is generated from a merger of two $1\,M_\odot$ stars, which are combined conservatively using `MMAMS` for constructing a $2\,M_\odot$ in dynamical equilibrium, whose structure is then calculated using `MESA`. The merger calculation is performed using the script in Listing 4.7. The script to generate this figure can be found in `amuse.examples.stellar.merger_stellar_density_profile`.

4.2.2 Stellar Evolution Modules in AMUSE

Currently, in AMUSE, we can choose between two Henyey codes and two parametric prescriptions for stellar evolution. Table 4.1 presents some fundamental parameters for these codes.

Just as we can easily run a simple N-body simulation in AMUSE, we can also run a simple stellar evolution calculation. The snippet in Listing 4.3 looks quite similar to the code presented in Listing 3.2 of Section 3.1.1, and indeed, the interface function names are deliberately chosen to be similar from one module to another. Listing 4.3 omits the details of parsing the command line to obtain the desired parameters and passing them to the main function. To incorporate that, we could include the Python option parser, modified within AMUSE to incorporate unit handling, as discussed in Appendix B (see Appendix B.4.2).

Instead of declaring a gravitational N-body code as we did in Section 3.1.1, here we initialize the stellar evolution code, specify the metallicity, and add a single particle of mass m and metallicity z, as follows:

```
stellar = MESA()
stellar.parameters.metallicity = z
stellar.particles.add_particle(Particle(mass=m))
```

The default metallicity for most codes is the solar value: $z = 0.02$, but as can be

```
from amuse.units import units
from amuse.datamodel import Particle

# from amuse.community.mesa import Mesa
from amuse.community.seba import Seba


def stellar_minimal(mass, metallicity, model_time):
    # stellar = Mesa(version="15140")
    # stellar = Mesa(version="2208")
    stellar = Seba()
    stellar.parameters.metallicity = metallicity
    stellar.particles.add_particle(Particle(mass=mass))

    initial_luminosity = stellar.particles.luminosity
    time_step = 0.1 | units.Myr

    while stellar.model_time < model_time:
        stellar.evolve_model(stellar.model_time + time_step)

        print(
            f"at T={stellar.model_time.in_(units.Myr)} "
            f"L(t=0)={initial_luminosity}, "
            f"L (t={stellar.particles.age.in_(units.Myr)})="
            f"{stellar.particles.luminosity.in_(units.LSun)}, "
            f"m={stellar.particles.mass.in_(units.MSun)}, "
            f"R={stellar.particles.radius.in_(units.RSun)}"
        )

    luminosity = stellar.particles.luminosity

    print(
        f"Time={stellar.model_time.in_(units.Myr)} "
        f"L={np.log10(luminosity.value_in(units.erg / units.s))}"
    )

    stellar.stop()
```

Listing 4.3: Minimal routine for evolving a single star of mass M and metallicity z to some end time model_time. This snippet was taken from the source code in amuse.stellar.minimal

seen, this parameter is easily changed. In order to run multiple stars with different metallicities, we would have to instantiate multiple stellar evolution codes, as we will demonstrate later in Section 4.2.4.

The stellar evolution code chosen here is MESA (short for Modules for Experiments in Stellar Astrophysics Paxton *et al.*, 2011, 2013, 2015, 2018; Gaburov *et al.*, 2018), which is a Henyey stellar evolution code (see Table 4.1). The single star is added to the repository of stars in the running MESA instantiation. In the line

```
initial_luminosity = stellar.particles.luminosity
```

we fetch the stellar luminosity directly from the stellar evolution code. We do something similar in the eventual print statement

```
print "at T=", stellar.model_time.in_(units.Myr), \
"L(t=0)=", initial_luminosity, \
", L (t=", stellar.particles.age.in_(units.Myr), \
")=", stellar.particles.luminosity.in_(units.LSun), \
", m=", stellar.particles.mass.in_(units.MSun), \
", R=", stellar.particles.radius.in_(units.RSun)
```

where we print some of the stellar properties after the star has been evolved to model_time. This is mainly a matter of convenience. For efficiency, we might alternatively have used a copy() function on a predefined communication channel, as discussed for the N-body case in **??**.

Stars in AMUSE are categorized into types, from pre-main sequence to black hole. This parameter is called the stellar_type. It is convenient for referring to the evolutionary state, but it should only be used as a rough measure of the stellar structure. We provide a complete list of AMUSE stellar types in Appendix B.

For a different metalicity, it is often better to start the stellar evolution code from a pre-main-sequence star because pre-build models are only stored in the AMUSE repository for Solar metallicity. One way to achieve this is

```
if z == 0.02:
    add_particle = stellar_evolution.native_stars.add_particles
else:
    add_particle = stellar_evolution.pre_ms_stars.add_particles
```

4.2.3 Improving the Stellar Evolution Solver

We can improve the stellar evolution solver by adding two new features to the code. The first is to store the results for later use. This is accomplished in much the same way as when we evolved a cluster of stars in the gravity solver and graphed the results (similar to the snippet presented in Listing 3.5). After storing the stellar data at the end of each step, we can plot it immediately in the same script, or save it and use a second small Python/AMUSE script to visualize the results.

Evolving a single star is somewhat limited. The second improvement is to evolve an entire population to study the isochrones of the stellar system—shown as curves on the H–R diagram showing stars of the same age. We can generate a list of masses from a stellar mass function (see **??**, Equation (2.1)). Any distribution will do, but from an observational perspective, the Salpeter (1955) mass function, a power law with index alpha = -2.35, is popular.[3] AMUSE contains a special function, new_salpeter_mass_distribution, to initialize masses according to this distribution, but here we use a more general power-law mass function to accomplish the task. The code

```
m = new_powerlaw_mass_distribution(N, Mmin, Mmax, alpha=-2.35)
```

[3] Another commonly used mass function, due to Kroupa (2001), is new_kroupa_mass_distribution().

```
stellar.particles.add_particles(Particles(mass=m))
```

returns an array of N masses chosen randomly between Mmin and Mmax with a Salpeter power-law distribution, then adds them to the stellar evolution solver.

Figure 4.8 is a H–R diagram of 1000 stars drawn from a Salpeter mass function with initial masses between 0.1 $M_\odot$ and 100 $M_\odot$, evolved to an age of 4.5 Gyr. Due to the random sampling the minimum mass turned out to be 0.100008472297 $M_\odot$ and the maximum is 7.85091223641 $M_\odot$. We assumed solar metallicity and evolved the stars using SeBa and MESA. The Henyey code crashed on several of the stars, particularly those of mass $\gtrsim 1.2\,M_\odot$; we have omitted those stars from Figure 4.8. In Section 7.4.5, we discuss how AMUSE deals with a crash of one of the underlying codes. Table 4.2 presents a small performance comparison between the four AMUSE stellar evolution modules.

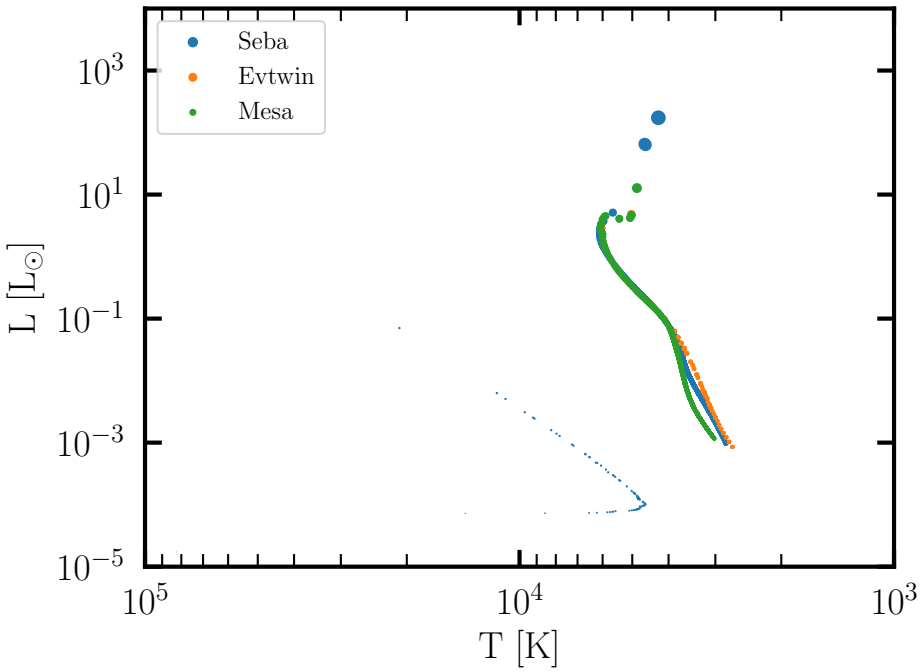

Figure 4.8: Hertzsprung–Russell diagram of 3000 stars with solar metallicity and initial masses between 0.1 $M_\odot$ and 100 $M_\odot$, evolved to an age of 4.5 Gyr. The stars were evolved using SeBa (blue), MESA (green), and EVtwin (red); the symbol sizes are proportional to the square root of the stellar radius. Only SeBa managed to evolve the stars through the white dwarf phase. This figure was produced with script amuse.examples.stellar.isochrone.py. Using MESA and EVtwin, the evolution of stars to the remnant stage is very time-consuming; this script will take many days to run, even on a multi-core workstation.

4.2.4 Evolving an Inhomogeneous Stellar Population

In the stellar evolution prescriptions implemented in AMUSE, time and metallicity are properties of the code, rather than of the star. This is somewhat confusing, but it is how the Henyey codes are structured. Changing the metallicity, in particular, requires the code to load a new set of opacity tables describing the stellar interior, and this is

Table 4.2: Comparison of the Four Available Stellar Evolution Codes in AMUSE

name	$t_{\rm CPU}$	$M/{\rm M}_\odot$	${\rm R}/{\rm R}_\odot$	$L/{\rm L}_\odot$	T/K
MESA	0:35.29	0.996	4.11	8.01	4792
EVTWIN	3:20.07	0.993	18.6	98.0	4212
SSE/BSE	0:0.63	0.999	2.58	3.40	4884
SeBa	0:0.62	0.999	2.58	6.40	4878

Notes. Here, we calculated the evolution of a 1 $\rm M_\odot$ solar metallicity star to an age of 11.75 Gyr. For EVtwin, the 1 $\rm M_\odot$ star turned into a white dwarf slightly after this. The runs were performed on a 2.4 GHz Core-i7 processor. CPU times are measured in hours:minutes.

simply too time-consuming to do efficiently every time we switch our attention from one star to another. An analogy might be the accuracy parameter in an N-body code, which also is an attribute of the code rather than of the individual particles. Thus, evolving a stellar population of a single age or metallicity is easy, but a multi-age or multi-metallicity population requires a little extra work.

The easiest way to evolve a population of stars of different ages is by initiating as many stellar evolution codes as there are stellar age groups, or maybe even one code for each individual star. Listing 4.4 presents a small script for evolving several stars, each to a different age and with its own stellar evolution code. When simulating a limited number of stars, the method in Listing 4.4 is probably adequate because it is simple and transparent. Performance is not really an issue, because each stellar evolution code can run independently, which makes for an embarrassingly parallel problem. For large numbers of stars, however, the memory requirements of this approach may become prohibitive.

```
10 def evolve_with_different_stellar_model():
11     times = [10, 100, 1000] | units.Myr
12     stars = Particles(mass=[1, 2, 4] | units.MSun)
13     stellars = [Seba(), Mesa(), Sse()]
14     channels = []
15     for star, stellar in zip(stars, stellars):
16         stellar.particles.add_particle(star)
17         channels.append(stellar.particles.new_channel_to(stars))
18 
19     for time, channel, stellar in zip(times, channels, stellars):
20         stellar.evolve_model(time)
21         channel.copy()
22 
23     for time, star in zip(times, stars):
24         print(f"Time= {time} M= {star.mass}")
25         stellar.stop()
```

Listing 4.4: Example code to initiate three stellar evolution codes to evolve three stars, each to a different age. This code fragment is taken from the source code in amuse.examples.multiple_stellar_code

4.2.5 Multiprocessing Codes

We can implement multiprocessing in this example by slightly rewriting the main evolve function in Listing 4.4. In Listing 4.5 we adopt a similar approach as in Listing 4.4, but we can enable multiprocessing by introducing the specific stellar evolution code as an argument. This routine takes code as an argument and uses it to evolve a (10 $M_\odot$) star to an age of (1 Myr). We could expand its functionality by introducing more arguments, such as mass, end time, and metallicity, but that would make the source listing longer and not necessarily easier to understand.

```
58 def evolve_single_star(code):
59     stars = Particles(mass=10 | units.MSun)
60     stellar = code()
61     stellar.particles.add_particles(stars)
62     channel = stellar.particles.new_channel_to(stars)
63
64     stellar.evolve_model(1 | units.Myr)
65     channel.copy()
66     print(
67         f"Star evolved to time= {stellar.model_time} "
68         f"M= {stars.mass} R= {stars.radius}"
69     )
70     stellar.stop()
```

Listing 4.5: Rewritten routine (from Listing 4.4) for evolving a single 10 $M_\odot$ star to an age of 1 Myr with the stellar evolution code code as an argument. Code fragment from amuse.examples.multiple_stellar_threaded.

The driving sequence for the main evolution routine in Listing 4.5 is presented in Listing 4.6. Here, we use the Python packages queue and threading for synchronizing queues and enabling threading, respectively, along with the process-based threading interface multiprocessing. We recognize that this example uses several fairly sophisticated multiprocessing concepts that the novice reader may find intimidating. If that is the case, you can safely skip to the next subsection.

We construct the task queue as a global variable code_queue, which in our case contains the list of codes used in our experiment. The actual work is done in the function remote_worker_code, which reads the next task from the queue, calls the evolution function evolve_single_star, and notifies the queue when the task has finished. The routine evolve_with_different_stellar_model forms the heart of the multiprocessing task. It fills the queue with the various codes, checks how many cores are available (to prevent overloading your system), and starts the threads. It ends by blocking each thread until all items in the queue have finished. The entire routine to launch multiple stellar evolution codes is given in amuse.examples.stellar.multiple_stellar_threaded.

Now we can evolve the same (10 $M_\odot$) star to the same age (1 Myr), using four different stellar evolution codes in parallel. These codes are listed in an array and passed as an argument to the driving routine:

```
codes = [SeBa, SSE, MESA, EVtwin]
evolve_with_different_stellar_model(codes)
```

Note that simulating the evolution of a large number of stars may run into fundamental

```
33 code_queue = queue.Queue()
34
35
36 def remote_worker_code():
37     code = code_queue.get()
38     evolve_single_star(code)
39     code_queue.task_done()
40
41
42 def evolve_with_different_stellar_model(codes):
43     for code in codes:
44         if code is not None:
45             code_queue.put(code)
46     n_cpu = multiprocessing.cpu_count()
47     for i in range(n_cpu):
48         thread = threading.Thread(target=remote_worker_code)
49         thread.daemon = True
50         thread.start()
51     code_queue.join()  # block until all tasks are done
```

Listing 4.6: Multiprocess version of the main loop for evolving stars, again from amuse.examples.multiple_stellar_threaded.

limitations of the operating system. Modern operating systems are limited, by their standard settings, to $2^{15} = 32{,}768$ concurrent jobs (even on large supercomputers), and it often requires root access to change this.

4.2.6 Enforcing Stellar Mass Loss/Gain

The stellar evolution codes in AMUSE have predetermined mass-loss prescriptions. They are generally not very elaborate because stellar mass loss is not understood particularly well. For this reason, and in order to allow the user to enforce mass loss or gain from other sources (imposed by hand in your script), we have enabled user-specified mass changes. If you invoke this feature, it is important to realize that you will have to switch off the standard mass-loss procedures. Built-in winds on the RGB and on the AGB must be turned off via:

```
1 stellar.parameters.RGB_wind_scheme=0
2 stellar.parameters.AGB_wind_scheme=0
```

This will switch off the mass-loss prescription in the instantiation of the stellar evolution code, and all stars evolved with this code are consequently affected.

Stellar mass loss can subsequently be imposed on a particular particle i in the code by

```
1 stellar.particles[i].mass_change = dmdt
```

where dmdt is the mass loss rate of star i. The other stars are unaffected and will not lose any mass. For example, for an Eta-Carinae-like luminous blue variable star, one might say

```
dmdt = -2.e-2 | units.MSun/units.yr
```

to model the remarkably high mass-loss rate of that star. A positive value of dmdt indicates that the star will gain mass; a negative value means mass loss.

Henyey stellar evolution codes are generally not prepared for large changes in the stellar mass, and may respond unexpectedly when the user makes an adjustment. One of the major issues here is that the internal time step is calculated on the assumption that mass loss is handled internally. Incorporating a much larger mass-loss rate—for example when simulating Roche-lobe overflow and calculating the mass loss from the radius of the Roche lobe relative to the stellar radius—may cause the internal time step to become too large to accommodate the externally imposed rate. It may even cause the stellar evolution code to crash—which might be the best possible outcome, as the code's response to an excessive mass change may shift the evolution onto a new and entirely different track. This evolutionary track is not necessarily wrong, as it could represent a viable solution to the underlying differential equations, but one may wonder if it really represents an observable star (stars are not onions; **[after – Steven]** Steig, 2008)).

Changing the mass loss rate of a particular star therefore also requires changing the stellar evolution time step of that star. This might be realized for star i by allowing it to lose less than dm = -0.02 | units.MSun within one time step, as follows:

```
new_time_step = min(dm/dmdt, stellar.particles[i].time_step)
stellar.particles[i].time_step = new_time_step
stellar.evolve_model()
```

Here, we start by calculating the new time step, after which we overwrite the star's time step with a smaller one and evolve the stars. The stellar evolution code will try to evolve the star to the specified next time step. However, this may fail because it also has the requirement that the star remain in thermal and dynamical equilibrium, in which case the actual time step taken by the code may be different.

In our example of evolving Eta Carinae as a 90 $M_\odot$ zero-age main sequence star, we obtain an initial evolution time step of 1.3 yr with MESA. Changing the mass-loss rate as just described and reducing the time step to 0.5 yr, checking the progress of the code indicates that the star was actually evolved for only 0.495 yr. After the first step is taken with the enhanced mass-loss rate and the reduced time step, subsequent time steps will be determined internally again. To guarantee that the star evolves according to the desired mass-loss rate and with subsequent small time steps, these parameters will have to be set again before the next step is taken.

4.2.7 Accessing Stellar Interiors

The direct (Henyey) stellar evolution codes in AMUSE solve for the entire structure of the stellar interior. The internal structure of a star is stored in zones, each of which

represents a grid point at which the stellar structure equations are solved. Modern stellar evolution codes typically use between 200 and 2000 zones. The number of zones for star i in the stellar evolution code stellar can be queried by

```
n_zones = stellar.particles[i].get_number_of_zones()
```

To access information on the stellar interior, use the particles attribute function get_xxx_profile(), which returns an array of data corresponding to the stellar internal structure parameter xxx, which may be mass, radius, temperature, luminosity, density, entropy, thermal_energy, etc.

For example, the internal temperature structure can be obtained by

```
temperature_profile = stellar.particles[i].get_temperature_profile(n_zones)
```

The returned array (here called temperature_profile) contains the temperature profile of the star, starting at the center (the first element of the array) and running all the way to the photosphere (the last element). To see the available options, it is probably best to look in the interface.py source code in the module's AMUSE directory. For example, for EVtwin, this will be in the file: amuse.community.evtwin.interface.[**Fixme: use inspect module! – Steven**] Note that the zones in the AMUSE stellar evolution codes are not distributed linearly in mass, radius, or any other parameter, but are spaced to ensure optimal functioning of the code. It is therefore important always to obtain the mass or radius profile along with the desired parameter. The code snippet in Listing 4.1 shows how to access the internal density and temperature profile of a star. Other internal stellar parameters can be accessed in similar ways. The number of access functions for the stellar interior is quite large, mainly because stellar evolution is a complex process, involving a wide variety of physics.

We encourage the reader to experiment with querying the internal stellar structure, and even changing it. To reduce the temperature in the outermost shell of a 10 $M_\odot$ star, one could say

```
T_profile[-1] = 0.9*T_profile[-1]
stellar.particles[i].set_temperature_profile(T_profile)
stellar.particles[i].time_step = 0.5*stellar.particles[i].time_step
stellar.evolve_model()
```

Here, we also reduce the time step (arbitrarily cutting it in half) to allow the stellar evolution code to adjust to the change. It is very easy to crash a stellar evolution code by fiddling with the stellar structure, which can easily cause the code to fail to converge. A typical example of output you should expect, after 117 lines of debugging messages, will eventually look like

CodeException: Exception when calling function 'evolve_for', of code

'MESAInterface', exception was 'lost connection to code'

Seeing how a code crashes and understanding why (here because it was forced on the code by small changes to the stellar interior) is an excellent way to appreciate the complexity of stellar evolution calculations.

4.2.8 Modeling Stellar Mergers

Stellar structure is critical when we merge two stars. A fast and convenient approximate approach to determining the interior structure of a merger product is provided by the MakeMeAStar code (MMAS; Lombardi *et al.*, 2003) and its extension MakeMeAMassiveStar (MMAMS; Gaburov *et al.*, 2008). These codes use data on the interiors of the two colliding stars to construct a new self-consistent stellar model, sorting mass shells by buoyancy to create the merger product, which can then be recommitted into a Henyey stellar evolution code to continue the evolution. Mass loss in the merger event is calibrated against a set of independent hydrodynamical simulations (mainly using smoothed particle hydrodynamics; see Section 5.2.1). The result is a fast method for obtaining a first-order estimate of the structure of the product of a low-velocity, roughly head-on merger of two stars.

Listing 4.7 presents a simple script in which two main sequence stars of masses Mprim and Msec are merged using MMAMS. Here, the primary and secondary masses are sent to the merger routine, which creates and evolves the two stars to time t_coll and then uses MakeMeAMassiveStar to merge them. The two lines

```
primary = stellar.particles.add_particle(bodies[0].as_set())
secondary = stellar.particles.add_particle(bodies[1].as_set())
```

initialize the primary and secondary stars. The particles in bodies contain only mass; they have no position or velocity information because we never initialize those quantities.

```
def merge_two_stars(mass_primary, mass_secondary, time_collision):
    """
    Merges two stars using the Make Me a Massive Star code and returns the
    resulting density profile.
    """
    bodies = Particles(mass=[mass_primary, mass_secondary] | units.MSun)

    stellar = Mesa()
    primary = stellar.particles.add_particles(bodies[0].as_set())
    secondary = stellar.particles.add_particles(bodies[1].as_set())

    stellar.evolve_model(time_collision)

    stellar.merge_colliding(
        primary.copy(), secondary.copy(), Mmams, return_merge_products=["se"]
    )
    radius = stellar.particles[0].get_radius_profile()
    rho = stellar.particles[0].get_density_profile()
    stellar.stop()
    return radius, rho
```

Listing 4.7: Simple routine to merge two stars (see amuse.examples.stellar.merge_two_stars). The script should run in about a minute on a laptop.

Note that functions like new_plummer_model(), which generate initial conditions,

generally return a particle set, but Particles(...) only returns a list. Thus, before adding the particles to the stellar instantiation, they first have to be converted into a particle set. In addition, the return value of add_particles is a pointer to the particle last added in the community code. (This was also true in the numerous previous examples using add_particles, but we ignored this fact in those cases.) In this case, it is convenient to maintain explicit access to the primary and secondary stars in the community code, rather than referring to them as stellar.particles[0] and stellar.particles[1].

After the initialization of the two stars, we evolve them to time t_coll. Because both the primary and secondary point directly to community code memory (which happens unseen here, via the MPI channel or a socket), they contain the same internal properties of the star as in the module memory. After the evolution step, we merge the two stars using the routine merge_colliding in MakeMeAMassiveStar. We work here with copies of the stars because both will be removed from the stellar particle set when the merger product is created. On return, the merger product is in stellar_particles[0]. The phrase return_merge_products=["se"] ensures that it has a Henyey structure describing its internal properties. The function merge_two_stars then returns the radius and density attributes of this structure. This script was used to generate Figure 4.7. In Section 4.3.2, we elaborate on this method by performing simulations to try to reconstruct an observed blue straggler.

This may look like a bit of dark AMUSE magic, but keep in mind that all it does is hand two stars to the merger routine and return two arrays describing the internal structure of the merger product.

4.2.9 Interrupting Stellar Evolution

Certain events in stellar evolution calculations, such as the supernova explosion of a massive star, can be quite sudden. The Henyey codes are not designed to continue after the supernova. We can flag and handle these events using stopping conditions, much as we discussed for gravitational dynamics in Section 7.4.1. The evolution of the remnant can then be continued using another code better suited to the task.

We enable the stopping condition in the stellar code by

```
detect_supernova = stellar.stopping_conditions.supernova_detection
detect_supernova.enable()
```

In the event loop, we can use the following snippet:

```
if detect_supernova.is_set():
for ci in range(len(detect_supernova.particles(0))):
print detect_supernova.particles(0)
code_body = Particles(particles=detect_supernova.particles(0))
local_body = particles_in_supernova\
.get_intersecting_subset_in(bodies)
```

Here, code_body is a pointer to the exploding star in the stellar evolution code stellar, while local_body is the same star in the local particle set. The former has the most up-to-date data from the stellar evolution code, but the latter may contain additional information used by other modules or additional local calculations.

4.2.10 Binary Evolution

Some stellar evolution codes can manage binary star evolution. It is more complicated to initialize a binary star than a single star, because the individual stars and the binary system itself are treated as separate particles in the particle set; consequently, there are a lot more free parameters to deal with.

4.2.10.1 Initializing and Evolving a Binary

To initialize a binary, one must first initialize the individual stars, then make a new particle set containing the binary properties, semimajor axis, and eccentricity. The binary particle is then given a pointer to each of the component stars. The two parameterized codes, SeBa and BSE, are currently binary-aware. The two Henyey stellar evolution codes EVtwin and MESA can also evolve binaries, but their interfaces differ slightly and they are prone to unanticipated interrupts (i.e., they tend to crash). **????** present a simple example of how to create and evolve a binary in AMUSE, and how to create a time series of orbital parameters.

```
15 def create_double_star(mass_primary, mass_secondary, semi_major_axis, eccentricity):
16     primary_stars = Particles(mass=mass_primary)
17     secondary_stars = Particles(mass=mass_secondary)
18 
19     stars = Particles()
20     primary_stars = stars.add_particles(primary_stars)
21     secondary_stars = stars.add_particles(secondary_stars)
22 
23     double_star = Particles(semi_major_axis=semi_major_axis, eccentricity=
       eccentricity)
24     double_star.child1 = list(primary_stars)
25     double_star.child2 = list(secondary_stars)
26     return double_star, stars
```

Listing 4.8: Routine to initialize a single binary star, as called in **??**.

```
def evolve_double_star(
    mass_primary,
    mass_secondary,
    semi_major_axis,
    eccentricity,
    time_end,
    number_of_steps,
):
    double_star, stars = create_double_star(
        mass_primary, mass_secondary, semi_major_axis, eccentricity
    )
    time = 0 | units.Myr
    time_step = time_end / number_of_steps

    code = Seba()
    code.particles.add_particles(stars)
    code.binaries.add_particles(double_star)

    channel_from_code_to_model_for_binaries = code.binaries.new_channel_to(
    double_star)

    t = [] | units.Myr
    a = [] | units.RSun
    e = []
    while time < time_end:
        time += time_step
        code.evolve_model(time)
        channel_from_code_to_model_for_binaries.copy()
        t.append(time)
        a.append(double_star[0].semi_major_axis)
        e.append(double_star[0].eccentricity)
    code.stop()
    return t, a, e
```

Listing 4.9: Routine to evolve a single binary star with mass_primary, mass_secondary, semi_major_axis, and eccentricity. The full script can be found in `amuse.examples.binary_evolution`. It should run in a few seconds on a laptop.

4.2.10.2 Rejuvenation Due to Collision or Accretion

When a star accretes material, by mass transfer from a companion or following a collision, a number of important changes occur. It starts to burn its nuclear fuel at a higher rate, and becomes hotter and brighter. Possibly the most important change, however, is the possibility that some of the accreted material finds its way into the stellar core, where nuclear fusion occurs. As a result, the accreting star may appear younger than the surrounding stars in a stellar cluster. Such "rejuvenated" stars are called *blue stragglers* (see Section 4.3.2). We will discuss the physics of stellar collisions in more detail in Section 5.3.3.

Rejuvenation is an important aspect of stellar and binary evolution, but different codes adopt very different approaches to the process. A Henyey code manages accretion onto a star by adding fresh gas to the outermost shell in the one-dimensional stellar structure. The internal solvers for the structure equations of the star will then resolve this addition and arrive at a new structure with the appropriate burning rates, etc.,

and ultimately mix the new gas into the interior. In a parameterized code, stars evolve along pre-calculated tracks and new tracks cannot easily be generated. Therefore, these codes incorporate an approximate prescription for on what track, and where on that track, an accreting star should reappear. This, of course, is a complete fudge, but it works reasonably well in practice. We warn the reader that these prescriptions are chosen because they seem reasonable and often work, but they are generally not tested against detailed calculations.

The two parameterized codes in AMUSE employ similar approaches toward rejuvenation, although they differ in detail. Following accretion, the star in SSE is restarted on the zero-age main sequence appropriate to the new mass. In SeBa, the evolution of a star is separated into partial tracks that represent the different stellar evolution phases (see Table B.1). After accretion, a star is placed at the same relative position between the beginning and end of the same track of a star with the increased mass. The relative age of the star along this track is reduced according to the fraction of accreted mass relative to the original mass of the star. In this way, a star can accrete small amounts of mass in consecutive phases of accretion.

For example, suppose that a 5 $M_\odot$ star halfway through the main sequence stage—at an age of 0.5 $t_{\rm tams}(5\ M_\odot)$—accretes 0.1 $M_\odot$ following a phase of rapid mass transfer. Here, $t_{\rm tams}$ refers to the *terminal-age main sequence*, the point where the star moves from the main sequence to the subgiant branch on the H–R diagram. According to the prescription in SSE, this new 5.1 $M_\odot$ star will start its evolution as a zero-age star at this mass. In SeBa, this same star will be restarted at a fraction of (0.5 $M_\odot$ - 0.1 $M_\odot$ /5 $M_\odot$) $\times\, t_{\rm tams}(5.1\,M_\odot) = 0.48 t_{\rm tams}(5.1 M_\odot)$.

A nice aspect of parameterized stellar evolution codes is their predictable behavior. The Henyey codes behave less predictably, but they provide a better, if more expensive, answer to this complicated problem.

4.2.11 Reading and Writing Binary Evolution Files

The data structure for stellar binaries (and higher-order hierarchies) is somewhat more elaborate than for a single particle set. Storing stellar multiples is therefore also somewhat more elaborate. Writing stellar binaries can be realized by separately storing the individual stars (here called stars) and the binaries (here called binaries). We write them in two separate files.

```
write_set_to_file(stars, "individual_tars.hdf5", "amuse")
write_set_to_file(binaries, "stars_binary_pairs.hdf5", "amuse")
```

We can read the data from the duplicate files with the reverse operation.

```
stars = read_set_from_file('individual_stars.hdf5','amuse')
binaries = read_set_from_file('stars_binary_pairs.hdf5','amuse')
```

4.3 Examples

4.3.1 Response of a Star to Mass Loss

An important issue in stellar evolution is the response of a star to changes in its mass. This response is generally expressed as the dimensionless adiabatic or the thermal (Kelvin–Helmholtz) response, $\zeta_{\rm ad}$ or $\zeta_{\rm th}$, where we define, for mass loss dm

$$\zeta = \frac{d\ln r}{d\ln m} \tag{4.18}$$

(Hjellming & Webbink, 1987), where $r(m)$ is the radius in the star containing mass m and the expression is evaluated at the surface of the star (where $m = M$). The response ζ can be compared with the change in the size of the Roche lobe in order to determine the rate at which the star loses mass due to Roche-lobe overflow (Passy *et al.*, 2012).

In binary population synthesis codes, one often adopts a very simple parameterization of ζ—it is generally held constant, independent of dm/dt, mass, or the evolutionary stage (Ge *et al.*, 2010). In SeBa, for example, $\zeta_{\rm th} = 0$ for main sequence stars less massive than 0.4 $M_\odot$, $\zeta_{\rm th} = 0.55$ for stars more massive than 1.5 $M_\odot$, and $\zeta_{\rm th} = 0$ for all other stars, except for those in the Hertzsprung gap, for which $\zeta_{\rm th} = -2$, independent of mass. The negative sign here indicates that the star shrinks as a result of mass loss. In these parameterizations, $\zeta_{\rm th}$ is assumed to be independent of dm/dt, which is not realistic because the response of the star to mass loss must depend on the mass-loss rate.

In AMUSE we can run a direct stellar evolution code, EVtwin or MESA, to calculate ζ and also study the dependence of ζ on the rate of mass loss. Listing 4.10 presents a simple routine to calculate the ζ coefficients for a particular star. This function is called by the main routine in Listing 4.11, in which a star of initial mass M is evolved. Figure 4.9 presents the evolution of ζ as a function of time for a few stars and mass-loss rates. We used MESA in these cases and continued the calculations until the code became impractically slow, which generally happened somewhere on the first asymptotic giant branch (see Figure 4.3).

4.3.2 Blue Stragglers in M67

M67 was discovered between 1772 and 1779 by Johann Gottfried Koehler, who described it as a "rather conspicuous nebula in elongated figure, near Alpha of Cancer." Charles Messier described the cluster on 1780 April 6 as a "cluster of small stars with nebulosity below the southern claw of the Crab." He gave the cluster its current identification M67. In 1783, William Herschel counted a total of 20 stars. Today, M67 is known to host nearly 1900 stars (Uribe *et al.*, 2007) and goes under many names, among them HR3515, OCl549.0, C0847 + 120, NGC2682, and [KPR2004b]212.

The age of a cluster is often measured by the distribution of stars in the H–R diagram. Key to this age determination is the main sequence turn-off—the temperature and luminosity at which stars leave the relatively hot main sequence and start to cross

```
def calculate_zeta(star, metallicity, dmdt):
    stellar = Mesa()
    stellar.parameters.metallicity = metallicity
    stellar.particles.add_particles(star)
    stellar.commit_particles()
    rold = star.radius
    star.mass_change = dmdt
    mass_step = 0.01 * star.mass
    star.time_step = mass_step / dmdt
    stellar.particles.evolve_one_step()
    rnew = stellar.particles[0].radius
    dlnr = (rnew - rold) / rold
    dlnm = (stellar.particles[0].mass - star.mass) / star.mass
    zeta = dlnr / dlnm
    stellar.stop()
    return zeta
```

Listing 4.10: Routine to calculate $\zeta_{\rm th}$ for a star. This function is called by the main function Listing 4.11.

```
def stellar_massloss_response(mass, metallicity, dmdt):
    stellar = Mesa()
    stellar.parameters.metallicity = metallicity
    bodies = Particles(mass=mass)
    stellar.particles.add_particles(bodies)
    stellar = turnon_massloss(stellar)
    channel_to_framework = stellar.particles.new_channel_to(bodies)
    copy_argument = ["age", "mass", "radius", "stellar_type"]
    while stellar.particles[0].stellar_type < Second_Asymptotic_Giant_Branch:
        stellar.particles.evolve_one_step()
        channel_to_framework.copy_attributes(copy_argument)
        star = stellar.particles.copy()
        zeta = calculate_zeta(star, metallicity, dmdt)
        print(
            "Zeta=",
            zeta[0],
            bodies[0].age,
            bodies[0].mass,
            bodies[0].radius,
            dmdt,
            bodies[0].stellar_type,
        )
    stellar.stop()
```

Listing 4.11: Main function to evolve a single star of mass M in time. Here, we use the internal stellar-evolution code parameter Second_Asymptotic_Giant_Branch = 6 | units.stellar_type, which is implemented as a part of the units package. This function calls Listing 4.10 to calculate ζ. The full script can be found in amuse.examples.massloss_response.

the Hertzsprung-gap to cooler regions. This turn-off point is associated with a certain stellar mass and hence age. Stars more massive than the turn-off mass have already started to cross the Hertzsprung gap, whereas lower-mass stars are still on the main sequence.

For the cluster M67, the turn-off mass is $\sim$1.4 $M_\odot$ (Boffin *et al.*, 2015, p.51). There is also a population of stars that lie along the main sequence, but are brighter and

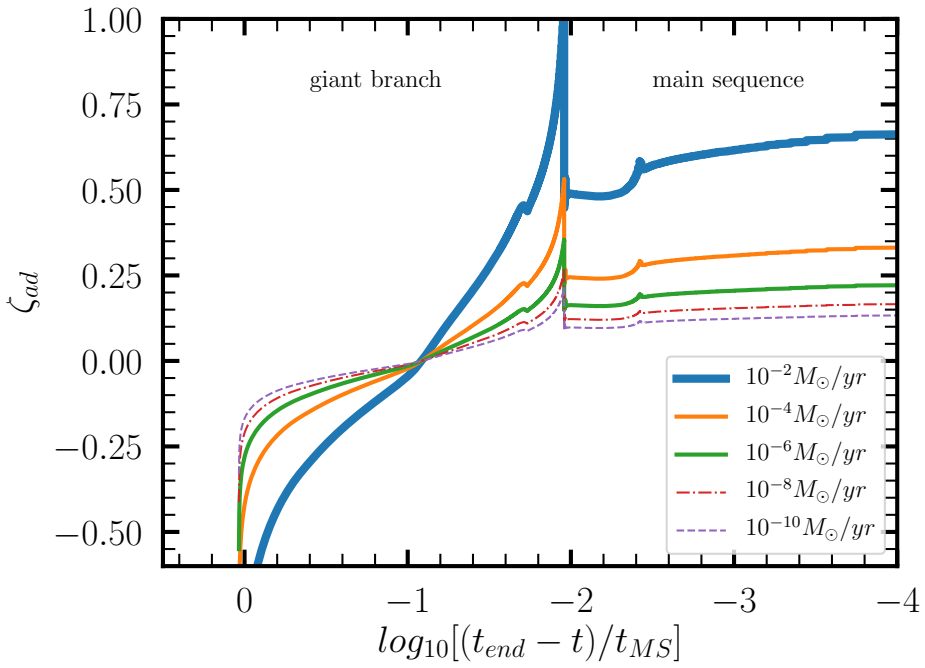

Figure 4.9: Response of an $M = 1$ $M_\odot$ star to mass loss, expressed as a function of time (relative to the main sequence lifetime). We performed the calculation for mass-loss rates $dm/dt = 10^{-2}\,M_\odot$/yr (top curve), $dm/dt = 10^{-4}\,M_\odot$/yr, $10^{-6}\,M_\odot$/yr, $10^{-8}\,M_\odot$/yr, and $10^{-10}\,M_\odot$/yr (bottom curve). The figure was created using a script based on amuse.examples.stellar.massloss_response. The calculation took several hours on a laptop, so we include the data file (Zeta_M1MSun_for_various_dmdt.data in the data directory) for a 1 $M_\odot$ star and plot this using the routine plot_stellar_massloss_response.py .

Table 4.3: Observed Parameters for Some M67 Blue Stragglers

name	$T_{\rm eff}$ [K]	V mag	L [$L_\odot$]	$\log g$
S984(F134)	6170	12.259	10.55	3.9
S975	6820	11.078	3.558	4.4
S997	6675	12.127	9.350	4.4
S1082	7050	11.226	4.078	4.5
S2204	6650	12.892	18.91	4.6

Notes. M67 is about 4 Gyr old (Shetrone & Sandquist, 2000). The luminosity is calculated from $10^{(V-\beta)/2.5}$. Here, $\beta = V - M_v = 9.7$ is the distance modulus.

hotter than the main sequence turn-off. These stars are called blue stragglers, and were first discovered in the globular cluster M3 (Sandage, 1953). Blue stragglers are commonly visible in any star cluster for which a turn-off can be clearly determined, but their origin remains somewhat elusive. Table 4.3 lists some fundamental and derived parameters for five blue stragglers in M67. Their temperatures and luminosities can be compared directly with the output of any stellar evolution module in AMUSE.

Because they are brighter and hotter than the turn-off, it has been suggested that blue stragglers result from stellar mergers during binary evolution or following stellar

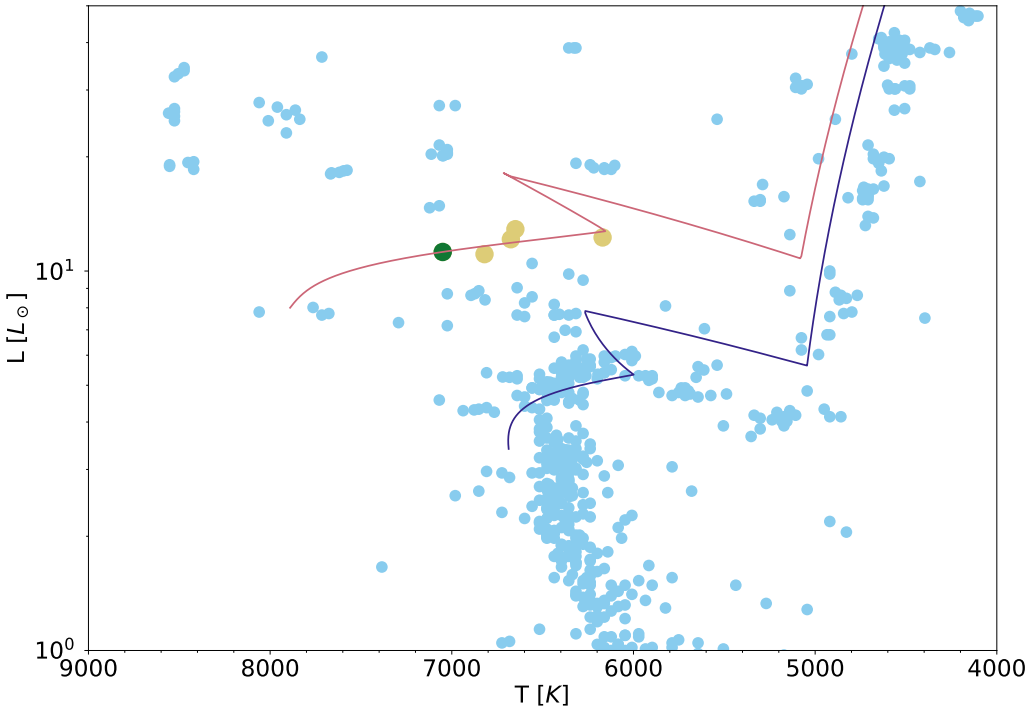

Figure 4.10: Hertzsprung–Russell diagram for the star cluster M67. The data are from http://webda.physics.muni.cz/ (Eggen & Sandage, 1964). The orange (the stars S984, S975, S997, and S2204) and blue (S1082) bullets are the blue stragglers listed in Table 4.3. Two lines indicate the evolutionary track for a star near the turn-off of 1.4 $M_\odot$ (blue) and 1.7 $M_\odot$ (red), both at twice the solar metallicity ($Z = 0.02$). This figure was generated using the script amuse.examples.objects.plot_m67_with_blue_straggler_tracks. It should take 5–10 s to run on a laptop.

collisions in the dense central region of star clusters. Their higher masses and younger ages are then direct consequences of their collisional history. Here, we discuss their possible association with stellar evolution; we will return to the possibility of a collisional origin in Section 7.4.4.

Figure 4.10 presents a historical H–R diagram for M67, including the five blue stragglers listed in Table 4.3. We overplot two evolutionary tracks to demonstrate how the turn-off stars evolve to an age of 4 Gyr. The upper 1.7 $M_\odot$ evolutionary track runs nicely through the blue stragglers, but the tip of the curve is off the plot. This indicates that these stars would already have evolved to the white dwarf stage, ergo they would be absent in this observation if they really were 4 Gyr old. The other track (blue curve) gives the evolutionary track of a 1.4 $M_\odot$ star to the same age. These stars are still visible in the cluster near the end of the turn-off or early Hertzsprung gap.

4.4 Validation

Stellar evolution codes are very hard to validate. One major problem is that we can't observe the full evolution of any star. Generally, there are two observable parameters that can be tested directly—the effective temperature (or something equivalent), and the luminosity. Recent advances in observational techniques and the discovery of planets orbiting many stars have given us new insights into stellar atmospheres, pulsations, etc., but these are often not solved for in stellar evolution codes (however, see also Christensen-Dalsgaard *et al.*, 1996; Aerts *et al.*, 2010).

The equations of stellar structure and evolution are generally quite well-behaved. There are few significant nonlinear effects, and there is no fundamental reason to think that solving the various differential equations should not provide the right answer (mathematically speaking), so long as the star is in thermal and dynamical equilibrium. Subsequent validation is carried out by comparing stellar evolution tracks with star cluster isochrones, or even with the positions of individual stars in the H–R diagram. This is the basic reason why stellar evolution theory has been so successful.

The only star for which a more thorough validation can be carried out is the Sun, because its surface parameters are quite well determined. In Figure 4.11, we compare the solar temperature and luminosity with the results of the four stellar evolution codes currently available in AMUSE. Although the differences are not large, they are still significant, particularly considering the small error bars on the observed values, which are smaller than the plotted symbols (see Williams, 2013).

It remains challenging to compare individual stellar evolution codes, particularly if they give slightly different results for the same input parameters (mass, metallicity, and age). Often, the differences are fairly small, and they are not a major concern for many coupled problems that involve hydrodynamics, radiative transport, or gravitational dynamics. The global mass loss of a population of stars in the central portion of a star cluster is the result of a complicated combination of dynamical and stellar evolution, and may depend only to first order on the specific stellar evolution code used. It remains important, however, to be aware of these small differences and recognize that a certain subspace of initial parameters may give different results, depending on the adopted stellar evolution code.

4.5 Assignments

4.5.1 Stellar Comparison

Rewrite the simple script plot_solar_comparison.py used in Figure 4.11 to compare the computed temperatures and luminosities of some other stars. The best stars to try this on are the closest and brightest—Rigel, Betelgeuse (see Figure 4.1), the red supergiant R Sculptoris, and both stars in the Spica binary system (see Table 4.4). Note that, even for these stars, the direct stellar evolution codes may be unable to converge to a solution, in which case they will simply break.

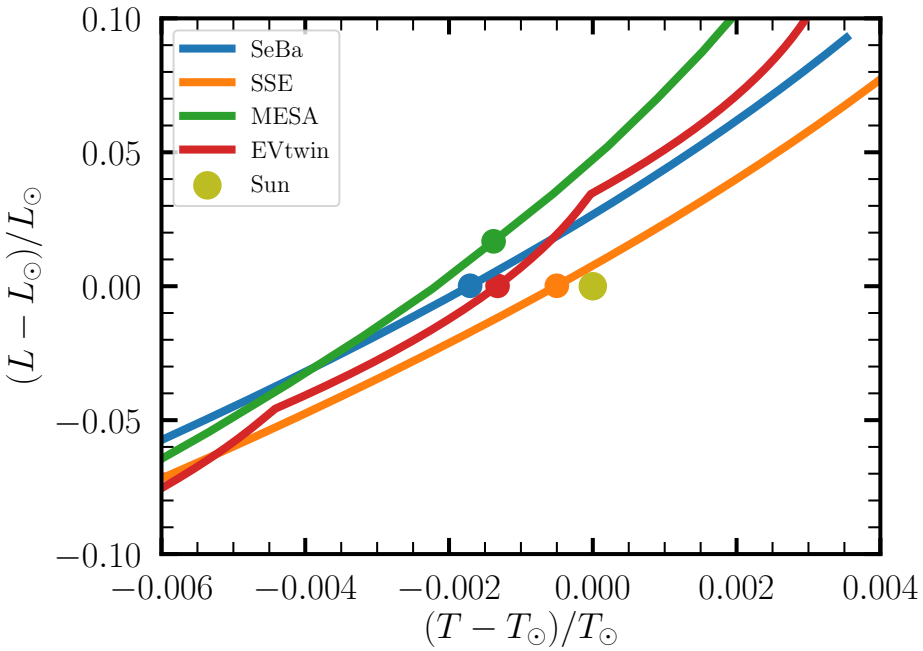

Figure 4.11: Comparison of the effective temperature and luminosity for the Sun, calculated using the four stellar evolution codes in AMUSE. The calculations were performed for a star with solar metallicity and an initial mass of 1 $M_\odot$, evolved from 4 Gyr to 5.2 Gyr (curves from left to right). The points indicate the moments (using each code's internal time resolution) when the Sun is best represented by the models. These occurred at 5.18 Gyr, 5.20 Gyr, 4.33 Gyr and 4.30 Gyr for SeBa, SSE, MESA, and EVtwin, respectively. The solar temperature and luminosity are taken from Allen's Astronomical Quantities (Cox, 2000). This figure was generated using the script amuse.examples.stellar.plot_solar_comparison, which should take about 20 s to run on a laptop.

Table 4.4: Observed and measured parameters for some nearby bright stars. References 1: Przybilla *et al.* (2006, 2010); 2: Lyubimkov *et al.* (1995); Harrington *et al.* (2009); 3: Palate *et al.* (2013); Tkachenko *et al.* (2016); 4: Ramírez *et al.* (2000); Mohamed *et al.* (2012); 5: Wittkowski *et al.* (2017).

name	ref	mass [$M_\odot$]	age [Myr]	T [K]	L [$L_\odot$]	R [$R_\odot$]	[Fe/H]
Rigel	1	21 ± 3	8 ± 1	12,100	$1.0–1.40 \times 10^5$	78.9 ± 7.4	−0.06
Spica A	2	11.4 ± 1.2	12.5	22,400	12,100	7.40 ± 0.57	—
Spica B	3	7.21 ± 0.75	12.5	18,500	1,500	3.64 ± 0.28	—
Betelgeuse	4	7.7–20	7.3	3140–3641	$9–15 \times 10^5$	950–1200	0.05
R Sculptoris	5	1.3 ± 0.6	—	2640 ± 80	5500	355 ± 55	—

4.5.2 Ages of the M67 Blue Stragglers

The five blue stragglers in M67 listed in Table 4.3 show a remarkable coherence in terms of their mass. In Section 4.3.2 we demonstrated that each of them lies close to the main sequence location of a 1.7 $M_\odot$ star of twice solar metallicity. Their ages, however, are clearly not the same, because each of them is positioned differently along the 1.7 $M_\odot$ evolutionary track.

Using any of the stellar evolution codes in AMUSE, can you determine the ages of each of the blue stragglers in Table 4.3, and perhaps even improve on the listed mass

estimate, assuming that they were formed as normal zero-age main sequence stars?

4.5.3 Constructing Isochrones

Write an AMUSE script to construct a series of 10 isochrones equally spaced in time from 1 Gyr to 10 Gyr for stars of 0.2 $M_\odot$ to 20 $M_\odot$, with solar metallicity. Plot the isochrones in the temperature (K)–luminosity ($L_\odot$) plane.

Convert the H–R diagram into a color–magnitude diagram. Luminosity can be converted to magnitudes in the usual way, by taking distance modulus and interstellar reddening [$E(B-V) = 0.04$] (Taylor, 2007) into account.

$$m_{\rm star} = m_\odot - 2.5 \log \left(\frac{L_{\rm star}}{L_\odot} \left(\frac{\rm pc}{d_{\rm star}} \right)^2 \right) \tag{4.19}$$

Here, the apparent magnitude of the Sun is $m_\odot = -26.73$, and its distance is $d_\odot \simeq 2.25 \times 10^{-8}$ pc (Allen, 1955; Cox, 2000). Color $(B-V)$ can be converted to temperature using the following polynomial fit:

$$\log T(K) \simeq [14.551 - (B-V)]/3.684. \tag{4.20}$$

Address the following questions:

- What is the best stellar evolution module to use for this task?
- What age best fits the observed star cluster M67? The data for M67 can be found at http://www.univie.ac.at/webda/cgi-bin/ocl_page.cgi?cluster=M67.
- What distance do you have to assume for the best fit?
- What age or metallicity spread do you need in order to obtain a better comparison between the simulated cluster and the observations?
- What additional parameters could you introduce in order to improve the fit?

Bibliography

Aerts, Conny, Christensen-Dalsgaard, Jørgen, & Kurtz, Donald W. 2010. *Asteroseismology.*

Allen, Clabon Walter. 1955. *Astrophysical quantities.*

Bazán, G., Dearborn, D. S. P., Dossa, D. D., Eggleton, P. P., Taylor, A., Castor, J. I., Murray, S., Cook, K. H., Eltgroth, P. G., Cavallo, R. M., Turcotte, S., Keller, S. C., & Pudliner, B. S. 2003 (Jan.). Djehuty, a code for modeling stars in three dimensions. *Page 1 of:* Turcotte, Sylvain, Keller, Stefan C., & Cavallo, Robert M. (eds), *3D Stellar Evolution.* Astronomical Society of the Pacific Conference Series, vol. 293.

Bethe, H. A. 1939. Energy Production in Stars. *Physical Review*, **55**(5), 434–456.

Binney, James, & Tremaine, Scott. 2008. *Galactic Dynamics: Second Edition.*

Boffin, H. M. J., Carraro, G., & Beccari, G. 2015. *Ecology of Blue Straggler Stars*.

Bradt, Hale. 2014. *Astrophysics Processes*.

Brott, I., de Mink, S. E., Cantiello, M., Langer, N., de Koter, A., Evans, C. J., Hunter, I., Trundle, C., & Vink, J. S. 2011. Rotating massive main-sequence stars. I. Grids of evolutionary models and isochrones. *A&A* , **530**(June), A115.

Chiu, H. Y. 1964. Supernovae, neutrinos, and neutron stars. *Annals of Physics*, **26**(3), 364–410.

Christensen-Dalsgaard, J., Dappen, W., Ajukov, S. V., Anderson, E. R., Antia, H. M., Basu, S., Baturin, V. A., Berthomieu, G., Chaboyer, B., Chitre, S. M., Cox, A. N., Demarque, P., Donatowicz, J., Dziembowski, W. A., Gabriel, M., Gough, D. O., Guenther, D. B., Guzik, J. A., Harvey, J. W., Hill, F., Houdek, G., Iglesias, C. A., Kosovichev, A. G., Leibacher, J. W., Morel, P., Proffitt, C. R., Provost, J., Reiter, J., Rhodes, E. J., Jr., Rogers, F. J., Roxburgh, I. W., Thompson, M. J., & Ulrich, R. K. 1996. The Current State of Solar Modeling. *Science*, **272**(5266), 1286–1292.

Clapeyron, É. 1834. *Mémoire sur la puissance motrice de la chaleur*. Jacques Gabay.

Clausius, R. 1857. Ueber die Art der Bewegung, welche wir Wärme nennen. *Annalen der Physik*, **176**(3), 353–380.

Cox, Arthur N. 2000. *Allen's astrophysical quantities*.

Dolan, Michelle M., Mathews, Grant J., Lam, Doan Duc, Quynh Lan, Nguyen, Herczeg, Gregory J., & Dearborn, David S. P. 2016. Evolutionary Tracks for Betelgeuse. *ApJ* , **819**(1), 7.

Eggen, O. J., & Sandage, A. R. 1964. New photoelectric observations of stars in the old galactic cluster M 67. *ApJ* , **140**(July), 130–143.

Eggleton, P. P., Tout, Christopher, Pols, Onno, Izzard, Rob, Eldridge, John, Lesaffre, Pierre, Stancliffe, Richard, Church, Ross, & Lau, Herbert. 2011 (July). *STARS: A Stellar Evolution Code*. Astrophysics Source Code Library, record ascl:1107.008.

Eggleton, Peter. 2006. *Evolutionary Processes in Binary and Multiple Stars*.

Eggleton, Peter P. 1971. The evolution of low mass stars. *MNRAS* , **151**(Jan.), 351.

Eggleton, Peter P., Fitchett, Michael J., & Tout, Christopher A. 1989. The Distribution of Visual Binaries with Two Bright Components. *ApJ* , **347**(Dec.), 998.

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

Gaburov, Evghenii, Lombardi, James C. Jr., Portegies Zwart, Simon, & Rasio, F. A. 2018 (May). *StarSmasher: Smoothed Particle Hydrodynamics code for smashing stars and planets*. Astrophysics Source Code Library, record ascl:1805.010.

Ge, Hongwei, Hjellming, Michael S., Webbink, Ronald F., Chen, Xuefei, & Han, Zhanwen. 2010. Adiabatic Mass Loss in Binary Stars. I. Computational Method. *The Astrophysical Journal*, **717**(2), 724.

Georgy, C., Meynet, G., Walder, R., Folini, D., & Maeder, A. 2009. The different progenitors of type Ib, Ic SNe, and of GRB. *A&A* , **502**(2), 611–622.

Harrington, David, Koenigsberger, Gloria, Moreno, Edmundo, & Kuhn, Jeffrey. 2009. Line-profile Variability from Tidal Flows in Alpha Virginis (Spica). *ApJ* , **704**(1), 813–830.

Haubois, X., Perrin, G., Lacour, S., Verhoelst, T., Meimon, S., Mugnier, L., Thiébaut, E., Berger, J. P., Ridgway, S. T., Monnier, J. D., Millan-Gabet, R., & Traub, W.

2009. Imaging the spotty surface of <ASTROBJ>Betelgeuse</ASTROBJ> in the H band. *A&A* , **508**(2), 923–932.

Hayashi, C. 1961. Stellar evolution in early phases of gravitational contraction. *Publ. Astr. Soc. Japan* , **13**(Jan.), 450–452.

Helfand, D. J., & Tademaru, E. 1977. Pulsar velocity observations: correlations, interpretations, and discussion. *ApJ* , **216**(Sept.), 842–851.

Henyey, L. G., Wilets, L., Böhm, K. H., Lelevier, R., & Levee, R. D. 1959. A Method for Automatic Computation of Stellar Evolution. *ApJ* , **129**(May), 628.

Henyey, L. G., Forbes, J. E., & Gould, N. L. 1964. A New Method of Automatic Computation of Stellar Evolution. *ApJ* , **139**(Jan.), 306.

Hjellming, Michael S., & Webbink, Ronald F. 1987. Thresholds for Rapid Mass Transfer in Binary System. I. Polytropic Models. *ApJ* , **318**(July), 794.

Hurley, Jarrod R., Pols, Onno R., & Tout, Christopher A. 2000. Comprehensive analytic formulae for stellar evolution as a function of mass and metallicity. *MNRAS* , **315**(3), 543–569.

Hut, Piet, Shara, Michael M., Aarseth, Sverre J., Klessen, Ralf S., Lombardi, James C. Jr., Makino, Junichiro, McMillan, Steve, Pols, Onno R., Teuben, Peter J., & Webbink, Ronald F. 2003. MODEST-1: Integrating stellar evolution and stellar dynamics. *New Astron.* , **8**(4), 337–370.

Kingdon, K. H., & Langmuir, Irving. 1923. The Removal of Thorium from the Surface of a Thoriated Tungsten Filament by Positive Ion Bombardment. *Physical Review*, **22**(2), 148–160.

Kippenhahn, R., & Weigert, A. 1967. Entwicklung in engen Doppelsternsystemen. I. Massenauschtausch vor und nach Beendigung des zentralen Wasserstoff-Bremens. *Zeitschr. f. Astroph.*, **65**, 251–273.

Köhler, K., Langer, N., de Koter, A., de Mink, S. E., Crowther, P. A., Evans, C. J., Gräfener, G., Sana, H., Sanyal, D., Schneider, F. R. N., & Vink, J. S. 2015. The evolution of rotating very massive stars with LMC composition. *A&A* , **573**(Jan.), A71.

Kroupa, Pavel. 2001. On the variation of the initial mass function. *MNRAS* , **322**(2), 231–246.

Lombardi, J. C., Thrall, A. P., Deneva, J. S., Fleming, S. W., & Grabowski, P. E. 2003. Modelling collision products of triple-star mergers. *MNRAS* , **345**(3), 762–780.

Lombardi, James C. Jr., Warren, Jessica S., Rasio, Frederic A., Sills, Alison, & Warren, Aaron R. 2002. Stellar Collisions and the Interior Structure of Blue Stragglers. *ApJ* , **568**(2), 939–953.

Lyne, A. G., & Lorimer, D. R. 1994. High birth velocities of radio pulsars. *Nat* , **369**(6476), 127–129.

Lyubimkov, L. S., Rachkovskaya, T. M., Rostopchin, S. I., & Tarasov, A. E. 1995. The binary system alpha Vir (Spica): Fundamental parameters of the components and differences in their helium abundance. *Astronomy Reports*, **39**(2), 186–194.

Meynet, G., & Maeder, A. 2005. Stellar evolution with rotation. XI. Wolf-Rayet star populations at different metallicities. *A&A* , **429**(Jan.), 581–598.

Meynet, G., Maeder, A., Schaller, G., Schaerer, D., & Charbonnel, C. 1994. Grids of massive stars with high mass loss rates. V. From 12 to 120 M_{sun}_ at Z=0.001, 0.004,

0.008, 0.020 and 0.040. *A&AS* , **103**(Jan.), 97–105.

Mohamed, S., Mackey, J., & Langer, N. 2012. 3D simulations of Betelgeuse's bow shock. *A&A* , **541**(May), A1.

Ohnaka, K., Weigelt, G., & Hofmann, K. H. 2017. Vigorous atmospheric motion in the red supergiant star Antares. *Nat* , **548**(7667), 310–312.

Palate, M., Koenigsberger, G., Rauw, G., Harrington, D., & Moreno, E. 2013. Spectral modelling of the α Virginis (Spica) binary system. *A&A* , **556**(Aug.), A49.

Passy, Jean-Claude, Herwig, Falk, & Paxton, Bill. 2012. The Response of Giant Stars to Dynamical-timescale Mass Loss. *ApJ* , **760**(1), 90.

Paxton, Bill, Bildsten, Lars, Dotter, Aaron, Herwig, Falk, Lesaffre, Pierre, & Timmes, Frank. 2010 (Oct.). *MESA: Modules for Experiments in Stellar Astrophysics*. Astrophysics Source Code Library, record ascl:1010.083.

Paxton, Bill, Bildsten, Lars, Dotter, Aaron, Herwig, Falk, Lesaffre, Pierre, & Timmes, Frank. 2011. Modules for Experiments in Stellar Astrophysics (MESA). *ApJS* , **192**(1), 3.

Paxton, Bill, Cantiello, Matteo, Arras, Phil, Bildsten, Lars, Brown, Edward F., Dotter, Aaron, Mankovich, Christopher, Montgomery, M. H., Stello, Dennis, Timmes, F. X., & Townsend, Richard. 2013. Modules for Experiments in Stellar Astrophysics (MESA): Planets, Oscillations, Rotation, and Massive Stars. *ApJS* , **208**(1), 4.

Paxton, Bill, Marchant, Pablo, Schwab, Josiah, Bauer, Evan B., Bildsten, Lars, Cantiello, Matteo, Dessart, Luc, Farmer, R., Hu, H., Langer, N., Townsend, R. H. D., Townsley, Dean M., & Timmes, F. X. 2015. Modules for Experiments in Stellar Astrophysics (MESA): Binaries, Pulsations, and Explosions. *ApJS* , **220**(1), 15.

Paxton, Bill, Schwab, Josiah, Bauer, Evan B., Bildsten, Lars, Blinnikov, Sergei, Duffell, Paul, Farmer, R., Goldberg, Jared A., Marchant, Pablo, Sorokina, Elena, Thoul, Anne, Townsend, Richard H. D., & Timmes, F. X. 2018. Modules for Experiments in Stellar Astrophysics (MESA): Convective Boundaries, Element Diffusion, and Massive Star Explosions. *ApJS* , **234**(2), 34.

Portegies Zwart, S. F., & Verbunt, F. 1996. Population synthesis of high-mass binaries. *A&A* , **309**(May), 179–196.

Portegies Zwart, S. F., & Verbunt, F. 2012 (Jan.). *SeBa: Stellar and binary evolution*. Astrophysics Source Code Library, record ascl:1201.003.

Portegies Zwart, S. F., & Yungelson, L. R. 1998. Formation and evolution of binary neutron stars. *A&A* , **332**(Apr.), 173–188.

Przybilla, N., Butler, K., Becker, S. R., & Kudritzki, R. P. 2006. Quantitative spectroscopy of BA-type supergiants. *A&A* , **445**(3), 1099–1126.

Przybilla, N., Firnstein, M., Nieva, M. F., Meynet, G., & Maeder, A. 2010. Mixing of CNO-cycled matter in massive stars. *A&A* , **517**(July), A38.

Ramírez, Solange V., Sellgren, K., Carr, John S., Balachandran, Suchitra C., Blum, Robert, Terndrup, Donald M., & Steed, Adam. 2000. Stellar Iron Abundances at the Galactic Center. *ApJ* , **537**(1), 205–220.

Saha, M. N. 1921. On a Physical Theory of Stellar Spectra. *Proceedings of the Royal Society of London Series A*, **99**(697), 135–153.

Salpeter, Edwin E. 1955. The Luminosity Function and Stellar Evolution. *ApJ* , **121**(Jan.), 161.

Sandage, A. R. 1953. The color-magnitude diagram for the globular cluster M 3. *AJ* , **58**(Jan.), 61–75.

Schaerer, D., de Koter, A., Schmutz, W., & Maeder, A. 1996. Combined stellar structure and atmosphere models for massive stars. I. Interior evolution and wind properties on the main sequence. *A&A* , **310**(June), 837–848.

Shetrone, Matthew D., & Sandquist, Eric L. 2000. Spectroscopy of Blue Stragglers and Turnoff Stars in M67 (NGC 2682). *AJ* , **120**(4), 1913–1924.

Smartt, Stephen J. 2009. Progenitors of Core-Collapse Supernovae. *ARA&A* , **47**(1), 63–106.

Stasińska, G., & Schaerer, D. 1997. Combined stellar structure and atmosphere models for massive stars. IV. The impact on the ionization structure of single star H II regions. *A&A* , **322**(June), 615–623.

Steig, W. 2008. *Shrek!* Square Fish.

Taylor, B. J. 2007. The Benchmark Cluster Reddening Project. II. A Reddening Value for M67. *AJ* , **133**(2), 370–386.

Tkachenko, A., Matthews, J. M., Aerts, C., Pavlovski, K., Pápics, P. I., Zwintz, K., Cameron, C., Walker, G. A. H., Kuschnig, R., Degroote, P., Debosscher, J., Moravveji, E., Kolbas, V., Guenther, D. B., Moffat, A. F. J., Rowe, J. F., Rucinski, S. M., Sasselov, D., & Weiss, W. W. 2016. Stellar modelling of Spica, a high-mass spectroscopic binary with a β Cep variable primary component. *MNRAS* , **458**(2), 1964–1976.

Toonen, S., Nelemans, G., & Portegies Zwart, S. 2012. Supernova Type Ia progenitors from merging double white dwarfs. Using a new population synthesis model. *A&A* , **546**(Oct.), A70.

Tutukov, A. V., & Fedorova, A. V. 2001. Evolution of Stars with High Metallicities. *Astronomy Reports*, **45**(11), 882–887.

Uribe, A., Barrera-Rojas, R. S., & Brieva, E. 2007 (Nov.). Membership in the Open Cluster M67: Working from Proper Motions via the EM Algorithm. *Page 403 of:* Babu, G. J., & Feigelson, E. D. (eds), *Statistical Challenges in Modern Astronomy IV*. Astronomical Society of the Pacific Conference Series, vol. 371.

Varshavskii, V. I., & Tutukov, A. V. 1975. Evolution of massive stars. AZh, **52**(Apr.), 227.

Williams, R. D. 2013 (July). Sun Fact sheet. *In: NASA Goddard Space Flight Center*.

Wittkowski, M., Hofmann, K. H., Höfner, S., Le Bouquin, J. B., Nowotny, W., Paladini, C., Young, J., Berger, J. P., Brunner, M., de Gregorio-Monsalvo, I., Eriksson, K., Hron, J., Humphreys, E. M. L., Lindqvist, M., Maercker, M., Mohamed, S., Olofsson, H., Ramstedt, S., & Weigelt, G. 2017. Aperture synthesis imaging of the carbon AGB star R Sculptoris. Detection of a complex structure and a dominating spot on the stellar disk. *A&A* , **601**(May), A3.