Computer projects
Formation and Evolution of Galaxies

Søren S. Larsen

April 14, 2018

Contents

1 Stellar population synthesis

In the Milky Way and its closest neighbouring galaxies, individual stars can be observed in considerable detail. From colour-magnitude diagrams, we can obtain information about the age distributions and metallicities of stellar populations, and spectroscopy provides a wealth of information about more detailed properties such as chemical abundances and kinematics.

However, in more distant galaxies we cannot generally observe individual stars, and we therefore have to rely on observations of *integrated light* - the combined light originating from all the stars (and other sources of radiation) in the galaxies. The aim of this exercise is to illustrate how we can use such observations to obtain information about the properties of galaxies.

1.1 Prerequisites

This exercise assumes familiarity with a few basic astronomical concepts. We will make use of astronomical magnitudes as explained, for example, in Section 2.1 of Mo, van den Bosch, & White, *Galaxy Formation and Evolution* (hereafter MBW). We will also assume a basic understanding of stellar evolution, as discussed in Section 10.2 of MBW.

1.2 Simple stellar population models

We start by considering the simplest case: a population of stars with a single age and chemical composition. This is often referred to as a *simple stellar population* (SSP). Real galaxies, with a mix of stellar populations with different ages and metallicities, may be thought of as a superposition of a large number of such SSPs. In nature, the closest thing to an SSP is a star cluster, although it is clear that some star clusters themselves consist of multiple stellar populations (especially the ancient *globular clusters*) and thus are not really "simple".

To model the integrated light of an SSP, we need to add together the contributions from all the individual stars that belong to the population. Although, by definition, these all have the same age and chemical composition, they have different *masses*, so we need to know:

1. For a given age and chemical composition, how does the luminosity as a function of wavelength, L_λ, depend on the mass of a star?

2. How many stars are there of a given mass?

For (1), we can rely on theoretical models of stellar evolution. From such models, one can calculate stellar *evolutionary tracks*, which describe how the properties of a star with a given (initial) mass and chemical composition depend on the age of the star (see Fig. 10.3 in MBW). One can then interpolate in evolutionary tracks for stars of different masses to get the properties of stars of a given *age* as a function of their initial masses. One then has a set of *isochrones*, which are very useful for modelling the properties of stellar populations.

As an example, Fig. 1 shows isochrones for ages of 10^7, 10^8, 10^9, and 10^{10} years. Such isochrones are really just data tables that list various properties of the stars (such as temperature, radius, and surface gravity) as a function of their (initial) mass and age. In principle, knowledge of the mass, temperature, luminosity, and chemical composition of each star allows

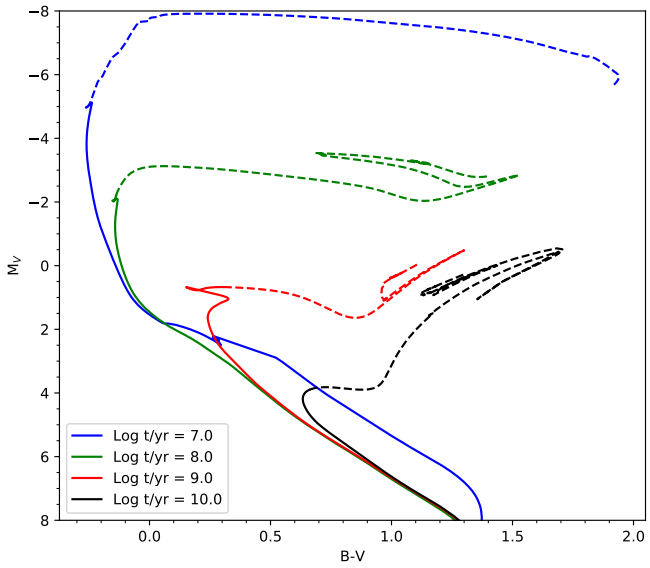


Figure 1: Theoretical isochrones for solar metallicity (here from MIST, `http://waps.cfa.harvard.edu/MIST/`). The dashed parts of the isochrones indicate the post-main sequence phases.

us to calculate its spectrum, L_λ. In practice, isochrone tables often include some basic information about the spectral energy distributions, such as the absolute magnitudes of the stars in selected sets of broad-band filters.

The isochrone tables list stellar properties as a function of mass, but do not contain information about the relative numbers of stars of different masses. This information is needed in order to calculate how much each point on the isochrone will contribute to the total luminosity of the stellar population. For this we need to assume a stellar *initial mass function*, IMF, formally defined as

$$\phi(m) \equiv \frac{\mathrm{d}N}{\mathrm{d}m} \tag{1.1}$$

The IMF can be thought of as a probability distribution, which gives the likelihood that a randomly selected, newly born star will have a mass between m and $m + \mathrm{d}m$. It is important to distinguish between this *initial* mass function, and the *present-day* mass function, which may have been modified by stellar evolution and other effects.

A useful approximation to the IMF is the Salpeter (1955) law,

$$\phi(m) \propto m^{-2.35} \tag{1.2}$$

Normalising this to a total stellar mass of 1 $M_\odot$, we have

$$\phi(m) = 0.35 \left(m_\mathrm{L}^{-0.35} - m_\mathrm{U}^{-0.35}\right)^{-1} m^{-2.35} \tag{1.3}$$

It is not obvious what we should select for the lower and upper limits. In practice, the Salpeter form of the IMF is at best an approximation, and may only apply over a limited mass range. The simple power-law behaviour tends to flatten below $\approx 0.5\,M_\odot$. Using $m_L = 0.15\,M_\odot$ gives about

the right normalisation, and one usually assumes (without any particularly strong justification) $m_U = 100\,M_\odot$.

Having adopted a set of isochrones and assumed an IMF, we can now calculate the integrated properties of SSPs as a function of age. In a real galaxy (and even more so in a star cluster), the integrated light is a sum over a finite number of stars:

$$L_{\lambda,\mathrm{SSP}} = \sum_i L_\lambda(m_i) \tag{1.4}$$

where the masses m_i would be sampled at random from the IMF. In practice, the number of stars in a galaxy is usually so large that the sum in Eq. (1.4) can be replaced with an integral:

$$L_{\lambda,\mathrm{SSP}} = \int_{m_L}^{m_U} \phi(m)\, L_\lambda(m)\, \mathrm{d}m \tag{1.5}$$

However, for stellar clusters the stochastic sampling of the IMF can be important, and represents additional challenges for the modelling and interpretation of integrated-light observations.

1.3 Calculating SSP models in practice

In this exercise we will use isochrones computed by the group at the Astronomical Observatory at Padova University. The most recent versions of the Padova isochrones are available online at: `http://stev.oapd.inaf.it/cgi-bin/cmd`. Since the many options on the website may be a bit of a distraction at first sight, we will here make use of a "standard" set of isochrones that you can find on the course webpage (`http://www.astro.ru.nl/~slarsen/teaching/GFormation/data/`). The files are named `isoc_z004.dat`, `isoc_z008.dat`, `isoc_z019.dat`, where the numbers indicate the metallicity ($Z = 0.004$, $Z = 0.008$, $Z = 0.019$, etc.), where the metallicity of the Sun is assumed to be $Z = 0.019$.

The first few lines of an isochrone file look like this:

```
#       Isochrone       Z = 0.01900            Age =   3.981e+06 yr
# log(age/yr)   M_ini          M_act   logL/Lo logTe   logG    Mbol    Mu      Mb      Mv      Mr      Mi      Mj      Mh      Mk      Flum
        6.60    0.15000001      0.1500  -2.5111 3.5183  5.1444  11.048  16.097  14.565  12.988  11.915  10.477  9.200   8.643   8.392   -9.59274740
        6.60    0.20000000      0.2000  -2.2373 3.5399  5.0819  10.363  14.929  13.517  12.016  11.018  9.740   8.568   7.996   7.757   -6.50542581
        6.60    0.25000000      0.2500  -2.0502 3.5514  5.0377  9.895   14.236  12.873  11.408  10.445  9.254   8.130   7.543   7.310   -4.81334753
        6.60    0.30000001      0.3000  -1.9017 3.5605  5.0048  9.524   13.730  12.388  10.938  10.002  8.902   7.787   7.166   6.942   -3.76315853
```

After the two header lines, the first column gives the log(age), followed by the initial and actual mass of the star. For low-mass stars on the main sequence these are essentially identical, but for more evolved stars that have undergone mass loss the actual mass can be significantly less than the initial mass. The next three columns list physical parameters for the stars: $\log L/L_\odot$ (the logarithm of the luminosity, in units of the solar luminosity), $\log T_{\mathrm{eff}}$ (the log of the effective temperature of the star, in K) and $\log g$ (the log of the surface gravity, in cm s^{-2}). Then follows the absolute bolometric magnitude, and after that the absolute magnitudes in various broad-band filters.

Isochrone files.
At this stage, it is good to familiarise yourself with the format of the isochrone files and the data stored in them. Open one of the files in a text editor. Then consider the questions below:

- What are the minimum and maximum ages included in the file? What are the steps in age?
- Why do you think the mass steps for a given age are unequal? Why do the steps become smaller at the high mass end?

To calculate SSP magnitudes for a given age and metallicity, we need to read in the entries for that age from the corresponding isochrone file, and then add up the contributions from each entry in the file, weighted by the IMF. Since we are sampling the IMF at a number of discrete mass intervals, the integral (1.5) becomes a sum:

$$L_{\lambda,\mathrm{SSP}} = \sum_i \phi(m_{\mathrm{ini},i})\, L_{\lambda,\mathrm{iso}}(m_{\mathrm{ini},i})\, \Delta m_i \tag{1.6}$$

To get from the absolute magnitudes, for example $M_V(m)$, to luminosities we apply the definition of magnitudes:

$$L_V = L_{V,\odot}\, 10^{-0.4\times(M_V - M_{V,\odot})} \tag{1.7}$$

where $L_{V,\odot}$ and $M_{V,\odot}$ are the luminosity and absolute magnitude of the Sun in a given filter (here V). It is usually convenient to express luminosities in units of the solar luminosity, i.e. setting $L_{V,\odot} = 1$ in the above expression. To convert back to the absolute SSP magnitude, we apply the inverse relation

$$M_{V,\mathrm{SSP}} = M_{V,\odot} - 2.5 \log_{10}\left(\frac{L_{V,\mathrm{SSP}}}{L_{V,\odot}}\right) \tag{1.8}$$

and we see that $L_{V,\odot}$ and $M_{V,\odot}$ cancel out. As long as we work within the magnitude system, it is thus irrelevant which values we assume for $L_{V,\odot}$ and $M_{V,\odot}$ for the intermediate steps. The absolute magnitudes in the isochrone tables will have been computed using assumptions about these zero-point constants.

Calculated in this way, $M_{V,\mathrm{SSP}}$ is the absolute V magnitude normalised to a total *initial* mass of the stellar population of 1 $M_\odot$ between m_L and m_U. Of course, SSPs with a total mass of $1 M_\odot$ rarely exist in nature, and if they did, it would certainly be very unrealistic to calculate their integrated properties as described here. However, we can easily scale the integrated luminosity calculated from Eq. (1.7) (and hence the integrated magnitude in Eq. (1.8)) to any other total mass. In many cases, we will be more interested in the ratios between luminosities at different wavelengths (or, equivalently, differences between magnitudes) in which case the overall mass scaling becomes irrelevant.

An example `python` programme to read in an isochrone file and calculate the integrated SSP M_V magnitude is listed below. The programme should be mostly self explanatory, although there are a couple of details to note:
1) The parameter `magsun=4.75` is the absolute magnitude of the Sun in a given filter. As noted above, the value of this parameter is actually irrelevant, as it cancels out in the calculation. However, if you should wish to modify the programme to return the luminosity, then this parameter becomes relevant. The default value is for the V-band, but it will have to be modified

for other filters.
2) For an isochrone table with N rows, the mass steps Δm are only defined for $N-1$ bins. We deal with that here by simply "throwing away" the last tabulated row for each age.

Exercise 1.1. The age-metallicity degeneracy.
The age-metallicity degeneracy is briefly discussed in Section 10.3.5 of MBW. In this exercise we will illustrate it by calculating SSP model colours for different ages and metallicities.
Extend the programme in the example to calculate the $B-V$ colour as a function of age, as in the upper panel of Figure 10.6 in MBW. Then make a plot of $B-V$ versus $\log t$ for two metallicities, $Z = 0.019$ (`isoc_z019.dat`) and $Z = 0.004$ (`isoc_z004.dat`).

Q1.a Suppose a star cluster is observed to have a colour of $B-V = 0.80 \pm 0.01$. Assuming that the metallicity of the cluster is in the range $0.004 < Z < 0.019$, what can you say about the age?

Breaking the age-metallicity degeneracy requires extra information. Suppose the following additional measurements of the cluster are available: $V-I = 1.03 \pm 0.10$, $V-K = 2.9 \pm 0.1$, and $M_V = -9.5 \pm 0.1$.

Q1.b Which of these additional measurements is more useful for constraining the metallicity and age of the cluster? What are now your best estimates of the age and metallicity?
In your report, include any additional plots that help illustrate how you found your answers.

Now that we know the metallicity and age of the cluster, we can estimate its total mass from the luminosity. Since the IMF is normalised to a unit mass of 1 $M_\odot$, the ratio between the observed luminosity of the cluster and that predicted by our SSP model can be used to calculate the mass of the cluster.
There is, however, one complication: the most massive stars have ended their lives, and lost most of their mass. Their *remnants* are now present in the form of white dwarfs, neutron stars, and black holes. Calculating their contribution to the total mass requires knowledge of the *initial-final mass relation* for stars, the discussion of which we shall postpone until a later exercise. Here we will simply assume 1) that the mass of remnants is negligible and 2) that all remaining stars have lost no mass. With these assumptions, the current mass can be calculated as the integral over the IMF from the lower limit to the most massive stars currently still alive (and thus present in the isochrone file).

Q1.c Estimate the initial and present-day masses of the cluster, assuming that the difference between the two is caused only by stellar evolution. Can you think of other reasons why the present-day mass might differ from the initial mass of the cluster?

Example 1.1: python programme to read an isochrone file and calculate integrated SSP magnitude

```
import numpy as np
import math

def imf(m, mL=0.15, mU=100.):
    return m**(-2.35) * 0.35 / (mL**(-0.35) - mU**(-0.35))

def sspmag(logtsel, logt, m_ini, magiso, magsun=4.75):

    wt   = np.where(abs(logt-logtsel)<0.01) # Select rows with logt == logtsel
    mag_t = magiso[wt]                      # New array with magnitudes for the selected age
    m_ini_t = m_ini[wt]                     # New array with masses for the selected age

    dm   = m_ini_t[1:] - m_ini_t[:-1]       # Calculate mass steps (Delta M)
    lum_t = 10**(-0.4*(mag_t[:-1]-magsun))  # Convert from Magnitudes to luminosities
    nm   = imf(m_ini_t[:-1])                # Calculate IMF weight for each mass

    lumssp = sum(lum_t*nm*dm)               # Calculate integrated luminosity
    magssp = -2.5*math.log10(lumssp) +magsun # Convert back to magnitude

    return magssp

# Name of file with isochrone data
fn = "isoc_z019.dat"

# Read in data from the file
logt, m_ini, mv = np.loadtxt(fn, usecols=(0,1,9), unpack=True)

# Calculate the integrated V-band magnitude for log(age) = 8.0
mvssp = sspmag(8.0, logt, m_ini, mv)
print mvssp
```

1.4 Models for continuous star formation histories

Having familiarised ourselves with SSP models, it is now time to consider the case of extended star formation histories (SFHs). An extended star formation history can be modelled as a sequence of SSPs of different ages (and metallicities). Since we now know how to compute SSPs, it should be relatively straight forward to generalise our programme to also deal with extended SFHs.

At a given wavelength, the flux can now be calculated by adding the individual SSPs together. We could write this as an integral

$$L_{\lambda,\mathrm{tot}} = \int_{\tau_1}^{\tau_2} L_{\lambda,\mathrm{SSP}}(\tau)\Psi(\tau)\,\mathrm{d}\tau \tag{1.9}$$

with $\Psi(\tau)$ being the star formation rate at time τ. The star formation history may be modelled in many different ways. A common simplifying assumption is that the star formation rate in a galaxy declines exponentially as a function of τ on a time scale $\tau_\star$:

$$\Psi(\tau) \propto \exp\left(-\frac{\tau}{\tau_\star}\right) \tag{1.10}$$

Once again, since the SSPs are computed at discrete ages, the integral is really a sum:

$$L_{\lambda,\mathrm{tot}} = \sum_i L_{\lambda,\mathrm{SSP}_i}\Psi(\tau_i)\,\Delta\tau_i \tag{1.11}$$

where the $\Delta\tau_i$ now represent the steps in time from the ith to the $(i+1)$th SSP.

In the above, we have used τ to denote time. A potential pitfall when picking the SSP corresponding to a particular time is that the isochrones (and hence the SSPs) are labelled with some *age*, t. Clearly, we have $t = t_0 - \tau$, where t_0 is the age where "the clock starts ticking". For an exponentially declining SFH, t_0 is just a normalisation constant. When coding this in practice, it is then convenient to loop over age t, rather than time τ. We can then use the spacing between the isochrones as time steps, $\Delta t = \Delta\tau$ (we evidently want to keep the steps as positive numbers). We could calculate the Δt's in the same way that we calculated the mass steps for the SSP models. However, since we know that the spacing of the isochrones is uniform in $\log t$, the Δt's are proportional to t, i.e.,

$$\Delta t_i = t_i \times \Delta \log t \times \ln 10 \tag{1.12}$$

We then have

$$L_{\lambda,\mathrm{tot}} = \sum_i L_{\lambda,\mathrm{SSP}_i}\Psi(t_0 - t_i)\,t_i\,\Delta \log t \tag{1.13}$$

up to some normalisation constant.

We are ready for the second part of the assignment.

Exercise 1.2. Extended star formation histories.
Below is an example `python` routine which takes as input an array of log(age) values and another array of SSP luminosities at each age. We will combine this routine with the example programme above to investigate how integrated-light properties behave for a population with an extended star formation history.
Include the SFH routine below in the example programme. First, modify the routine `sfhmag` in the example programme to return the luminosity, rather than absolute magnitude, in a given filter (you may give the routine a new, more appropriate name, e.g. `sfhlum`). Then set up a loop to calculate the integrated SSP luminosities for the relevant ages. Finally, feed the SSP luminosities into the `sfhlum` routine below.
You can assume a metallicity of $Z = 0.019$ throughout this exercise.

Q2.a Compute the $B-V$ colours for maximum ages of 5 and 10 Gyrs for SFHs with $\tau_\star = 10^9$ years, $\tau_\star = 3 \times 10^9$ years, and $\tau_\star = 10^{10}$ years. Compare with the middle panel in Fig. 10.9 in MBW to verify that you get (approximately) similar results.

Q2.b So far, we have assumed a Salpeter IMF. How do you think the model colours would change for a steeper (or shallower) IMF? Try changing the IMF slope (e.g., to $\alpha = -3.35$) and see what happens. Explain!

Example 1.2: `python` subroutine to sum SSPs for extended SFH.

```
def sfhlum(logt, lumt, t0=1e10, taustar=1e15):
# logt = array of log(t) values
# lumt = array of SSP luminosities at log(t). Delta log(t) assumed constant!

    ltot = 0.
    for tt, ll in zip(logt, lumt):    # Loop over logt, lumt
        ti = 10**tt                   # Age of SSP
        tau = t0 - ti                 # Time from t0
        psi = math.exp(-tau/taustar)  # Star formation rate
        ltot = ltot + ll*ti*psi       # Add contribution to total luminosity
    return ltot
```

1.5 Final remarks

Hopefully, this exercise has helped illustrate some of the basic principles behind the modelling of observable properties of stellar populations. A more realistic modelling must include many effects that we have not accounted for here, such as dust absorption and more realistic star formation histories and age-metallicity relations, of which the latter will be the subject of the next computer project. Many stars are members of binary or multiple systems, which will affect their evolution if the stars are sufficiently close to each other to interact and transfer mass. These effects are not captured by standard stellar evolutionary models, which assume that stars evolve in isolation. We have concentrated on emission from stars, but the spectra of galaxies with active star formation will also typically contain emission lines from gas clouds that are ionized by UV radiation from young stars. Furthermore, we have neglected the contributions

from other sources of radiation, such as AGN (active galactic nuclei), X-ray binaries, transient sources such as novae and supernovae, etc.

1.6 The report

At the end of this project, please hand in a brief report with the answers to the exercises, any relevant figures, and listings of the programmes you have used. Proper explanations of how you obtained the answers are as important as the answers themselves!

2 Chemical Evolution models

In the lecture we discussed the basics of models for galactic chemical evolution. In this assignment we will use a computer programme based on the equations for chemical evolution as given in MBW, Chapter 10.4. A more detailed discussion of chemical evolution can be found in the book *Nucleosynthesis and Chemical Evolution of Galaxies* by Bernard E. J. Pagel (Cambridge University Press, 2007). A brief summary follows.

2.1 Basics: keeping track of the mass

The total (baryonic) mass is the sum of the mass of stars M_s and gas M_g:

$$M = M_g + M_s \tag{2.1}$$

In the closed-box approximation, M is constant, but in general we may allow for infall and ejection of gas from the system, quantified by the accretion- and (wind) outflow rates, $\mathcal{A}(t)$ and $\mathcal{W}(t)$. Then the rates of change of M, M_g, and M_s are given by

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \mathcal{A}(t) - \mathcal{W}(t) \tag{2.2}$$

$$\frac{\mathrm{d}M_g}{\mathrm{d}t} = \mathcal{A}(t) - \mathcal{W}(t) + \mathcal{E}(t) - \Psi(t) \tag{2.3}$$

and

$$\frac{\mathrm{d}M_s}{\mathrm{d}t} = \Psi(t) - \mathcal{E}(t) \tag{2.4}$$

where $\Psi(t)$ is the *star formation rate* and the matter ejection rate by stars is $\mathcal{E}(t)$ (i.e., the rate at which gas is returned to the gas phase from previously formed stars). These relations simply follow from the overall requirement that mass is conserved (strictly speaking, we are ignoring the tiny amount of mass that is converted to energy and radiated away).

As we have also discussed in the lectures, a realistic description of the star formation rate is complicated by the fact that there are different gas phases. In this assignment we are only tracking the total amount of gas and, furthermore, the geometry of the gas is not explicitly specified. We will assume that the star formation rate Ψ can be estimated from the amount of gas, M_g, via the simple Schmidt-Kennicutt relation (Kennicutt 1998):

$$\Sigma_{\mathrm{SFR}} \propto \Sigma_{\mathrm{gas}}^{1.4} \tag{2.5}$$

If we thus think of all masses as surface densities, we can use relation (2.5) to get the star formation rates (per unit area). This is the relation implemented in the computer programme, but it can easily be changed.

To determine $\mathcal{E}$ at a given point in time t, we need some extra information. Suppose a star of a given initial mass m has a lifetime $\tau(m)$ and leaves a remnant of mass m_{rem}. Then the mass returned to the gas phase, at a time τ after the star formed, is $m - m_{\mathrm{rem}}$ (in some stars, significant mass loss may occur earlier in their evolution, but we ignore that here.). We thus need to adopt an *initial-final* mass relation; the programme uses a relation given by Lawlor & MacDonald (2006). We also need to know how many stars of mass m formed at time $t - \tau$, which is the

product of the star formation rate $\Psi(t - \tau(m))$ and the initial mass function $\phi(m)$ (see the first computer exercise). Then we have

$$\mathcal{E}(t) = \int_{m_{\tau=t}}^{m_U} (m - m_{\rm rem})\Psi\left[t - \tau(m)\right]\phi(m)\,{\rm d}m \tag{2.6}$$

where the lower limit of the integral corresponds to stars that have a lifetime of $t = \tau$. Stars of even lower masses will then have lifetimes greater than t and will thus still be on the main sequence, where we assume that they do not contribute to chemical enrichment. Note that this lower limit is time-dependent; it shifts to lower masses as time progresses. The upper limit, m_U, is equal to the mass of the most massive stars formed; its exact value is uncertain but is often assumed (somewhat arbitrarily) to be $m_U = 100\,M_\odot$.

2.2 Chemical enrichment

Next, we look at the equations for chemical enrichment, and wish to follow the evolution of the amount of metals in the gas. Stars forming at a given time will initially have the same composition as the gas. Formally, the general expression for the change in the amount of metals present in the gas is

$$\frac{{\rm d}M_Z}{{\rm d}t} = \frac{\rm d}{{\rm d}t}(M_g Z) = \mathcal{E}_Z - Z\Psi + Z_{\rm acc}\mathcal{A} - Z_{\rm ej}\mathcal{W} \tag{2.7}$$

There are four terms: $\mathcal{E}_Z$ is the rate at which metals are added to the gas phase after having been produced by nucleosynthesis in stars (or after simply having been temporarily locked up in the stars), and $Z\Psi$ accounts for the metals that are being incorporated in new stars via star formation (some of which will be returned back to the gas phase later). The final two terms account for accreted and ejected mass; in general the composition of the accreted gas is likely different from that present in the galaxy at any given time, and we have also allowed for the ejected gas to have a metallicity that may, in principle, be different from the overall metallicity in the gas. We can calculate $\mathcal{E}_Z$ using an expression analogous to Eq. (2.6):

$$\mathcal{E}_Z(t) = \int_{m_{\tau=t}}^{m_U} \left[(m - m_{\rm rem})Z(t - \tau(m)) + mq_Z(m)\right]\Psi\left[t - \tau(m)\right]\phi(m)\,{\rm d}m \tag{2.8}$$

The first term inside the integral accounts for unenriched gas returned at the end of a star's lifetime (this gas thus has the same composition that the gas had at time $t - \tau(m)$, when the star formed), and the second term accounts for the new metals produced in the star. Here, $q_z(m)$ is the amount of metals produced as a fraction of the initial mass of the star.

Dealing with Type Ia supernovae is somewhat more complicated as it involves the evolution of binary stars, which has many uncertain aspects. One must therefore, in principle, account for the fraction of stars that are members of binary systems (this may be mass dependent), the orbital separations, and the mass ratios in order to estimate what fraction of the binaries are likely to eventually explode as Type Ia SNe.

Following Pagel (2007), we adopt a simplified treatment, in which we simply assume that some fraction of all stars with initial mass less than 8 $M_\odot$ explode as Type Ia SNe with a delay $t_{\rm delay}$ after they have become white dwarfs. Furthermore, we assume that all Type Ia SNe return

an identical amount of enriched material, m_Z, to the gas. We have already accounted for the return of unenriched gas above, in Eq. (2.8). Then we can write

$$\mathcal{E}_{\mathrm{Z,Ia}}(t) = \eta_{\mathrm{Ia}} \int_{m_{\tau=t+t_{\mathrm{delay}}}}^{8M_\odot} m_Z \Psi\left(t - \tau(m) - t_{\mathrm{delay}}\right) \phi(m)\, \mathrm{d}m \tag{2.9}$$

where η_{Ia} is an efficiency parameter that specifies which fraction of the stars in the mass range from $m_{\tau=t+t_{\mathrm{delay}}}$ to $8M_\odot$ explode as Type Ia SNe.

2.3 Implementation

The theory described above is implemented in a `Python` programme, called `gce.py`. A listing of this programme is given in Appendix A. The main function of the programme, `gce_model()`, carries out a numerical integration of the equations given above and prints the results on standard output. Calling the programme is simple:

```
> python gce.py
```

or, redirecting the output to a file:

```
> python gce.py > output.txt
```

The data in `output.txt` can then be read by other programmes and manipulated, plotted, etc.

As listed in Appendix A, the ejection and accretion rates are set to zero, so the programme will calculate a closed-box model, starting with a gas density of 10 $M_\odot\,\mathrm{pc}^{-2}$, a value that is close to the current gas surface density near the Sun. Even in the closed-box approximation, however, the programme does *not* assume instantaneous recycling, even for Type II supernovae: each star has a finite lifetime, with the mass-lifetime relation read from the file `t_m.txt`, which must therefore be present in the working directory. The IMF can be changed by modifying the function `IMF()`. The Type II SN yields for O and Fe are given in the arrays at the beginning of the programme and are taken from Woosley & Weaver (1995).

Exercise 2.1. Understanding the output from `gce`

First of all, take some time to look at the code and try to understand how it works. You will need to modify it for some of the exercises below.

Run the programme in its default form, redirecting the output to a text file. Inspect the output:

Q2.1a Look at columns 2 and 3 (M_g and M_s). Is the sum of the two constant, as it should be?

Q2.1b The next column lists the star formation rate. How does this change with time? Why?

Q2.1c Next follows the ejected stellar mass, $\mathcal{E}$. This should be zero in the first few time steps. Why?

Q2.1d After the total $\mathcal{E}$ follows $\mathcal{E}_{^{16}\mathrm{O}}$. This remains zero for another couple of time steps. Why is that so?

Q2.1e Columns 8 and 9 list the Fe produced by Type Ia and Type II SNe. How long does it take for the Type Ia SNe to become "active"? Explain the delay quantitatively.

Q2.1f After how much time do Type Ia SNe start to return more iron to the gas phase than Type II SNe? Why does this happen relatively late in the chemical enrichment history of the galaxy, compared to t_{delay}?

Q2.1g Estimate the effective yields, $y_{\mathrm{eff}} \equiv \frac{Z}{\ln 1/f_{\mathrm{gas}}}$, for oxygen and iron. You can use $f_{\mathrm{gas}} = 0.5$ as the reference point ($f_{\mathrm{gas}} = M_g/(M_g + M_s)$).
You will need these for the next exercise.

Exercise 2.2. Comparison with analytical models

Q2.2a Make a plot of the abundances of [O/H] and [Fe/H] as a function of $f_{\rm gas}$. For the Solar abundances of these elements, you can assume $Z(\mathrm{O}, \odot) = 0.0097$ and $Z(\mathrm{Fe}, \odot) = 0.0014$.

Together with the abundances of these elements computed by the programme, plot the analytical prediction by the closed-box model with instantaneous recycling:

$$Z = -p \ln f_{\rm gas} \tag{2.10}$$

for suitable choices of the yields, $p_{\rm O}$ and $p_{\rm Fe}$.

Does the analytical prediction agree with the numerical calculations?
For each of the two elements O and Fe, why do you think the agreement is good/poor?

Q2.2b Make a plot of [O/Fe] vs. [Fe/H]. Explain what you see in the plot. Compare with Figure 10.10 in MBW.

- How does this relation change if you adopt a flatter IMF (i.e., more high-mass stars)? Explain!
- What happens if you increase the star formation rate for a given gas density (i.e., effectively increasing the star formation efficiency, $\epsilon_{\rm ff}$) by a factor of five? Explain!

Having examined some properties of the closed-box model in detail, we now proceed to modify the model to allow for in- and outflows.

Exercise 2.3. In/Outflow models

First, let us have a look at what happens when wind-driven outflows are included. Change the main subroutine of the programme, `gce_model()`, to include winds. As a representative example, assume $\mathcal{W}(t) = 5 \times \Psi(t)$.

Q2.3a Now what happens to the total mass within model as a function of time? What is the total mass at the end of the simulation?

Q2.3b Include the output from the wind model in the plot from **Q2.2a** above. What difference do the winds make to the abundances as a function of $f_{\rm gas}$? Is this in agreement with the predictions by analytical models?

Next, we look at a model where accretion is included. We first consider the *steady-state* model, in which the gas mass (M_g) remains constant.

Q2.3c Which modification to the programme is necessary in order to keep M_g constant? (hint: see Equation (2.3))

Now run the programme and verify that the gas mass does indeed remain constant.

Q2.3d Compare the age-metallicity relation for the accreting model with that for the closed-box model. How do the two age-metallicity relations differ?

Exercise 2.4. Starbursts

Galaxies sometimes experience bursts of star formation, which often appear to be triggered by interactions with other galaxies. The most dramatic examples of this are major mergers (like the Antennae galaxies), but bursts may also be triggered by close encounters that may trigger compression of the gas.

Q2.4a Modify the programme so that the star formation rate (for a given gas density) is increased by a factor of ten for ages between 1.5 Gyr and 2 Gyr. One way to do this would be to pass the age as a parameter to the `calcsfr()` subroutine and multiply the computed SFR by 10 for the corresponding age interval.

- Verify that the SFR increases, as intended.

Q2.4b Compare the [O/Fe] vs. [Fe/H] relation in the closed-box model with and without the burst. What difference do you see? Explain.

2.4 The report

At the end of this project, please hand in a brief report with the answers to the exercises and any relevant figures. Also, describe how you modified the `gce()` programme to obtain your results.

Feel free to explore additional scenarios - there are rich possibilities for varying the inflow/outflow rates, IMF, initial-final mass relation, star formation recipes, Type Ia delay times,

etc., etc., and examining how modifications to the input assumptions affect the model output.

Proper explanations of how you obtained the answers are as important as the answers themselves!

3 Galaxy interactions

Close encounters can dramatically alter the appearance of galaxies. The most striking examples are major gas-rich mergers, in which two spiral galaxies can be transformed to an elliptical galaxy. However, tidal interactions can also have less dramatic effects. For example, a smaller companion galaxy may perturb the disc of a larger galaxy and thereby induce grand-design spiral structure (as in the M51 pair) or cause *warping* of the disc. Head-on collisions may also lead to the formation of *polar ring* galaxies.

In general, the outcome of an interaction is determined by many parameters, including the geometry of the encounter and the masses and detailed structure of the two galaxies. A realistic modelling of galaxy encounters is computationally very demanding, and must take into account the gravitational interaction between a large number of particles belonging to the discs, bulges, halos, and other components of the galaxies. In addition, modelling of the gas phase requires hydrodynamical simulations. Such simulations generally require powerful supercomputers and are well beyond the scope of this exercise.

Fortunately, it is possible to gain some insight into the problem using simpler techniques. The numerical analysis of galaxy interactions was pioneered in the 1970s by Alar and Juri Toomre, who showed that many of the observed features of interacting galaxies can be reproduced in relatively simple simulations. In the Toomre models, each galaxy was modelled as a central point mass ("bulge"), surrounded by mass-less particles that initially follow circular orbits ("disc"). When two galaxies interact, each disc particle feels the gravitational force from both bulges, and the two bulges feel each other, but the disc particles do not interact among themselves. Hence, the number of force calculations scales linearly with the number of disc particles N, unlike a true N-body calculation in which the number scales as N^2.

In this exercise we will use the "Toomre approximation" to study galaxy interactions, using a `Python` code written by Vincent Henault-Brunet. On modern desktop computers, this type of simulation takes less than a minute. The programme listing is included in Appendix B.

3.1 Initial conditions

At the beginning of the simulation, each of two galaxies consists of a central point mass, surrounded by concentric rings of disc particles that follow circular orbits around the centre. The masses, coordinates, and velocities of the two bulges are specified in the subroutine `set_ICs()`. The disc stars initially orbit in the (x', y') plane of a coordinate system which may be tilted with respect to the reference coordinate system. The orientation of the disc coordinate system is specified by the *Eulerian* coordinates (θ, ψ, ϕ), where:

- θ = nutation. This is the angle between the z' axes (i.e., the normals to the discs) and the z-axis of the reference system.
- ψ = precession. Angle between the intersection of the (x', y') and (x, y) planes, measured with respect to the $x-$axis.
- ϕ = rotation. Rotation of the x' axis with respect to the intersection line. As the discs are assumed to be axisymmetric initially, this latter angle is irrelevant here.

In the default set-up, these angles are all set to zero, so that both discs lie in the (x, y) plane.

3.2 Evolving the encounter

Once the initial set-up has been defined, it is just a matter of letting gravity do its work. This is done in the subroutine `run_sim()`, which integrates the simulation forward, one step at a time. First, the acceleration of each halo (due to the other halo) is computed and the halo velocities and positions are updated. Second, the acceleration of each disc particle due to the two halos is calculated, and the disc particle velocities and positions are updated.

These simulations lend themselves well to visualisation through animations. The code here uses the `FuncAnimation()` function in the `matplotlib.animation` library. `FuncAnimation()` calls the `run_sim()` subroutine, which then carries out the integration as described above and updates the plots. At the end, the sequence of plots is saved as a movie. Under Ubuntu Linux, you can take screenshots from the `Edit` menu within the Movie Player. These are then stored in the "Pictures" folder.

3.3 Gravitational softening

A common problem in this type of simulation is that the gravitational force diverges as the separation between two point masses approaches zero, leading to spuriously large accelerations:

$$\vec{F} = G\vec{R}\frac{m_1 m_2}{R^3} \tag{3.1}$$

In nature there are no point masses; although many real galaxies have a black hole at their centre, this accounts for only a tiny fraction of the total mass of the galaxy, and even a black hole has a finite event horizon, after all.

The usual solution is to introduce a *softening length*, r_s, when calculating the force:

$$\vec{F} = G\vec{R}\frac{m_1 m_2}{(R^2 + r_s^2)^{3/2}} \tag{3.2}$$

For separations $R \gg r_s$, Eq. (3.2) gives the same force as Eq. (3.1), but for small radii ($R \ll r_s$) the force remains finite.

Exercise 3.1. Getting to know the code.

Running the code is as simple as

```
> python run_toomre.py
```

This will produce an output movie, `toomre.mp4`, in which you can see the two galaxies approaching each other, interacting, and then separating again.

Q3.1a In the default set-up, what are the initial coordinates, velocities, and masses of the bulges? Remember to include the units!

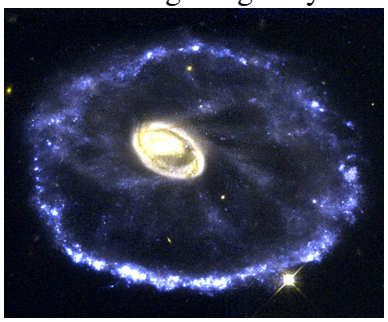
Q3.1b Include a screenshot showing the final frame of the simulation in your report. What has happened to the discs? What has happened to the positions and velocities of the two bulges?

Q3.1c What happens if you now reverse the spin direction of one of the discs? Include screenshots near the time of the closest encounter and at the end of the simulation. (you may want to enlarge the plot window a bit to show all the particles).

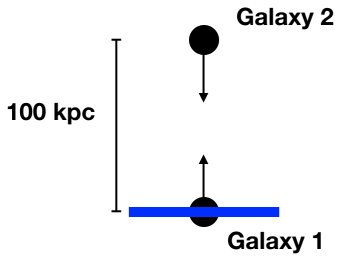
Discuss how the outcome of this simulation differs from that of the previous one - can you explain the difference(s)?

3.4 The Cartwheel Galaxy

The image below shows the *Cartwheel Galaxy*. The peculiar outer ring is thought to be the result of a head-on encounter between the original galaxy and a smaller companion.



To simulate the encounter that may have led to the Cartwheel galaxy, we need to change the setup in our simulation. As shown below, we want Galaxy 2 to collide face-on with Galaxy 1, passing right through the center of Galaxy 1. We can then adjust the relative velocities of the two galaxies and the ratio of their masses in order to reproduce the Cartwheel as closely as possible.



Exercise 3.2. Simulating the Cartwheel galaxy

The default softening length in the code is very small, 5×10^{-4} kpc. Since we are here attempting to model a head-on collision, this becomes critical, and it is reasonable to set the softening length to a scale that is more representative of the mass distribution in galaxies. A softening length of 1.5 kpc works fairly well.
Set up a simulation in which the initial separation of the galaxies is 100 kpc and their relative velocity is 400 km/s (i.e., each galaxy has a velocity of 200 km/s). If you let the galaxies move along the y direction in the (x, y) plane, you will need to rotate the disc coordinate system of Galaxy 1 by 90 degrees, as in the figure above. Set the mass of Galaxy 2 to half of that of Galaxy 1, i.e., $5 \times 10^{10} M_\odot$ for Galaxy 2 and $10^{11} M_\odot$ for Galaxy 1. We are not interested in any disc associated with Galaxy 2 in this simulation, so you can comment out the statements where the disc particles for Galaxy 1 are plotted (near the end of the `run_sim()` subroutine and under "plot stars" in the main programme). It will be useful to show the encounter both in the (x, y) plane (i.e., edge-on view of the disc) and in the (x, z) plane (face-on view of the disc).

Q3.2a Make a modified version of the code, in which the movie shows the encounter in the (x, z) plane instead of the (x, y) plane.

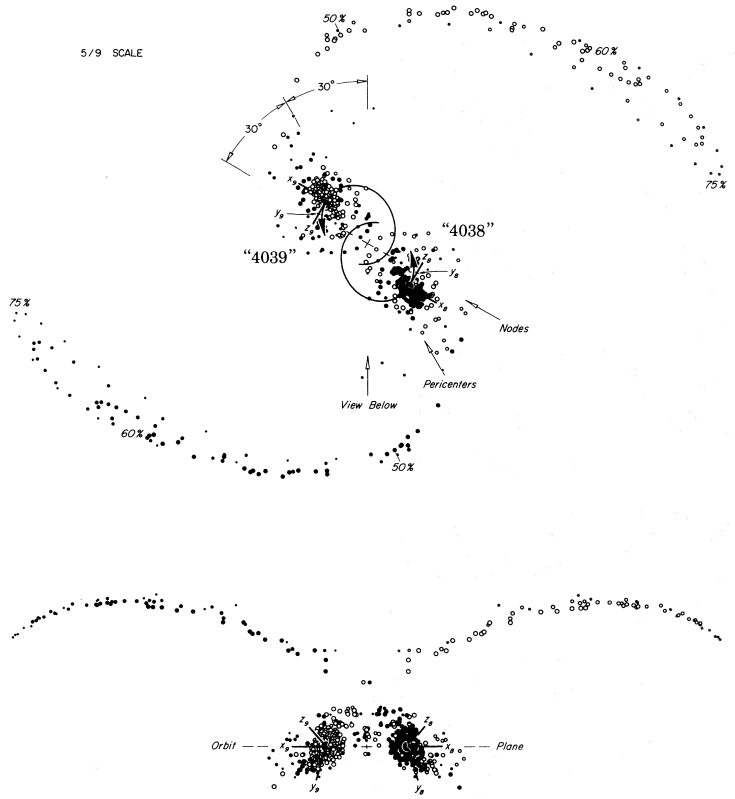
Q3.2b Now run the simulation with the parameters suggested above. Describe what you see. At what time does the simulation bear the closest resemblance to the Cartwheel system? Include screenshots of the face-on and edge-on views of the simulations at this time.

Q3.2c What happens if the encounter is not exactly head-on? Try shifting the initial coordinates of Galaxy 2 by 10 kpc in the x-direction, then rerun the simulation. Describe what you see.

3.5 The Antennae

One of the most famous (and the closest) examples of a major spiral-spiral merger is the "Antennae", NGC 4038/39. It is among the set of galaxies originally investigated by Toomre & Toomre (1972), and it has been the subject of many subsequent studies. The Toomre brothers first showed that the main features of the Antennae – the symmetry, and the long, apparently intersecting tidal tails – can be well reproduced in a set-up where the two spiral discs are highly inclined (by about 60°) with respect to the orbital plane, and our line-of-sight nearly coincides with the orbital plane.

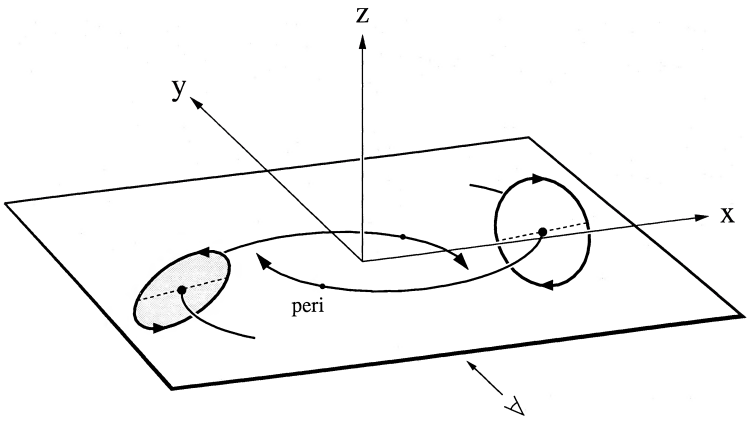
The Toomre & Toomre simulation is shown below. The figure shows a snapshot of the system shortly after the pericentre passage and it can be seen that the tails do not actually intersect physically, but only in projection.



The Toomre & Toomre (1972: ApJ, 178, 623) model of The Antennae. Here two identical discs orbiting in the (x, y) *plane with inclinations* $\theta_1 = 60°$, $\theta_2 = -60°$ *have experienced an encounter. The top panel shows the projection onto the* (x, y) *plane, the bottom panel the projection on the* (x, z) *plane.*

The geometry of the encounter is perhaps somewhat clearer in the following figure, taken from Barnes (1988) who carried out more realistic simulations of the Antennae, including extended mass distributions from the bulges, dark matter halos, and discs.

In this assignment we will try to reproduce the Antennae.



Geometry of the Antennae encounter, from Barnes (1988: ApJ, 331, 699). Our viewing direction is indicated by the arrow

Exercise 3.3. The Antennae

As indicated in the figure from Barnes (1988), the set-up is symmetrical. Here, we give each galaxy a mass of $2 \times 10^{11}\, M_\odot$ and we make the discs more compact than in the previous simulations by reducing the spacing between the rings to 5 kpc. The galaxies are initially at $x = \pm 50$ kpc from the origin of the coordinate system, and the discs have inclinations of $\pm 60°$.

Q3.3a The velocities in the y-direction define the eccentricity of the relative orbits of the two galaxies and need to be "just right". What happens if the y-velocities are too large? Too small? What is a good value for the initial velocities? What spin direction do the discs need?

Q3.3b At what time does your simulation most closely resemble the actual "Antennae" merger pair (you can easily find images of it with Google). Include the (x, y) and (x, z) snapsnots of your best-matching simulation.

Appendix A

gce.py: Programme to calculate models for Galactic chemical evolution.
Needs the data file t_m.txt with stellar lifetimes and masses.

```
import math
import numpy as np
import scipy.interpolate as ip

# Initial-final mass relation (from Lawlor and MacDonald 2006, MNRAS 371, 263)

minit = [0.80, 0.85, 0.90, 0.95, 1.000, 1.200, 1.500, 2.000, 2.500, 3.000, 4.000, 5.
mfin  = [0.54, 0.54, 0.54, 0.54, 0.541, 0.557, 0.563, 0.595, 0.632, 0.678, 0.778, 0.
fmfin = ip.interp1d(minit, mfin)

# Yields for Type II SNe, from Woosley and Weaver 1995, ApJ Suppl. 101, 181

mww = [11.0, 12.0, 13.0, 15.0, 18.0, 19.0, 20.0, 22.0, 25.0, 30.0, 35.0, 40.0]
m16O = [1.36e-01, 2.10e-01, 2.72e-01, 6.80e-01, 1.13, 1.43, 1.94, 2.38, 3.25, 3.65,
mFe  = [0.0804, 0.0554, 0.1458, 0.1297, 0.0828, 0.1177, 0.1063, 0.2247, 0.1509, 0.31

f16O = ip.interp1d(mww,m16O)
fFe  = ip.interp1d(mww,mFe)

# IMF slope
alpha = -2.35

def calcsfr(mg):
# Simple Schmidt-Kennicutt law, from Kennicutt (1998)
# Here, mg is the gas surface density (in Msun pc^-2)
    logMg = math.log10(mg)
    logSFR = logMg*1.4 - 3.5
    SFR = 1e-6 * 10**logSFR
    return SFR

def mrem(m):
# Calculate the remnant mass for a star of initial mass m
# For stars more massive than max(minit) we assume, arbitrarily, a remnant
# mass of 2 Msun

    if (m < max(minit)):
       mm = fmfin(m)
    else:
       mm = 2.0

# Alternatively, Iben & Tutukov 1984 (Pagel 1997, p. 239):
#    if (m < 9.5):
```

```
#       mm = 0.11*m + 0.45
#    else:
#       mm = 1.5

    return mm

def fq16O(m):
# Mass fraction of 16O returned by a star of mass m
    if ((m > min(mww)) and (m < max(mww))):
        return f16O(m)/m
    else:
        return 0.

def fqFe(m):
# Mass fraction of Fe returned by a star of mass m
    if ((m > min(mww)) and (m < max(mww))):
        return fFe(m)/m
    else:
        return 0.

def IMF(m):
# Evaluate the Salpeter (1955) IMF for a star of mass m
    mmin = 0.15
    mmax = 100.
    return m**alpha * -(alpha+2) / (mmin**(alpha+2) - mmax**(alpha+2))

def mej(ti, at, aSFR):
# Calculate the ejected mass, from stellar evolution, at time ti.
# at and aSFR are arrays containing the time and star formation rate.
# Stellar masses and lifetimes are read from the file t_m.txt

    logt, m = np.loadtxt('t_m.txt',usecols=(0,1), unpack=True)
    t = 10**logt
    fm = ip.interp1d(t, m)
    ft = ip.interp1d(np.flipud(m), np.flipud(t))

    if (ti < min(t)):      # No stars have evolved off MS yet
        ee = 0.
    elif (len(at) < 2):
        ee = 0.
    else:
        fSFR = ip.interp1d(at, aSFR)     # Interpolate in (t, SFR) arrays
        mt = fm(ti)                      # Lower limit of the integral
        mU = max(m)                      # Upper limit
        mm = mt
        ee = 0.
        while (mm < mU):
```

```
            dm = mm / 25.
            tSFR = max(ti - ft(mm),0)    # Calculate SFR when star of mass
                                         # m was born
            if (tSFR > max(at)): tSFR = max(at)
            de = (mm - mrem(mm)) * fSFR(tSFR) * IMF(mm) * dm  # ejected mass
            ee = ee + de
            mm = mm + dm

    return ee


def mejZ_SNIa(ti, at, aSFR, mZ):
# Calculate the amount of metals returned by Type Ia SNe
# mZ is the mass of the element produced by one SN explosion

    mSNIamax = 8.0   # Maximum mass of stars that become Type Ia SNe
    fSNIa    = 0.02  # Fraction of those stars that actually become Type Ia SNe
    delay = 1e8      # Time delay, after endpoint of normal stellar evolution

    logt, m = np.loadtxt('t_m.txt',usecols=(0,1), unpack=True)
    t = 10**logt + delay     # Remember to add the delay
    fm = ip.interp1d(t, m)
    ft = ip.interp1d(np.flipud(m), np.flipud(t))
    w = np.where(m < mSNIamax)

    if (ti < min(t[w])):
        ee = 0.
    elif (len(at) < 2):
        ee = 0.
    else:
        fSFR = ip.interp1d(at, aSFR)
        mt = fm(ti)
        mm = mt
        ee = 0.
        while (mm < mSNIamax):
            dm = mm / 25.
            tSFR = max(ti - ft(mm),0)
            if (tSFR > max(at)): tSFR = max(at)
            de = fSNIa * mZ * fSFR(tSFR) * IMF(mm) * dm
            ee = ee + de
            mm = mm + dm

    return ee


def mejZ(ti, at, aSFR, aZ, fqZ):
# Calculate the amount of metals returned by Type II SNe
```

```
# aZ should contain the metallicity of the gas at times at
# fqZ(m) is a function that should return the mass fraction of element Z
# returned by a star with initial mass m

# The rest is largely equivalent to the function mej() above

    logt, m = np.loadtxt('t_m.txt',usecols=(0,1), unpack=True)
    t = 10**logt
    fm = ip.interp1d(t, m)
    ft = ip.interp1d(np.flipud(m), np.flipud(t))

    if (ti < min(t)):
        ee = 0.
    elif (len(at) < 2):
        ee = 0.
    else:
        fSFR = ip.interp1d(at, aSFR)
        fZ = ip.interp1d(at, aZ)
        mt = fm(ti)
        mU = max(m)
        mm = mt
        ee = 0.
        while (mm < mU):
            dm = mm / 50.
            tSFR = max(ti - ft(mm),0)
            if (tSFR > max(at)): tSFR = max(at)
            de = ((mm - mrem(mm))*fZ(tSFR) + mm*fqZ(mm)) * fSFR(tSFR) * IMF(mm) * dm
            ee = ee + de
            mm = mm + dm

    return ee


def gce_model(tmax=1.0e10, Mg0=10., Ms0=0.):
# Calculate the actual chemical evolution model.
# This largely follows "Nucleosynthesis and Chemical Evolution of Galaxies"
# by Pagel (2007), Section 7.4 (pp 243-244), also MBW Section 10.4
# Mg0 is the initial gas mass (surface density; Msun pcˆ-2) and
# Ms0 is the initial stellar mass (surface density)

    t = 0.
    Mg = Mg0       # Gas mass
    Ms = Ms0       # Stellar mass
    M16O = 0.      # Mass in 16O in gas phase
    MFe  = 0.      # Mass in Fe in gas phase
    Z16Oacc = 0.   # Composition of accreted gas
    ZFeacc = 0.
```

```
# Arrays to store the enrichment history and related variables
    at, aMg, aMs, aSFR, aZ16O, aZFe = [], [], [], [], [], []

    while (t < tmax):

        dt = 1e6 + t/1e2    # Time steps: minimum 1e6 yr, then 1% of age

        SFR = calcsfr(Mg)
        et  = mej(t, at, aSFR)

        e16O = mejZ(t, at, aSFR, aZ16O, fq16O) # O from Type II SNe
        Z16O = M16O/Mg

        eFeI = mejZ_SNIa(t, at, aSFR, 0.61)    # Fe from Type Ia SNe
        eFeII = mejZ(t, at, aSFR, aZFe, fqFe)  # Fe from Type II SNe
        eFe = eFeI + eFeII
        ZFe = MFe/Mg

# Gas accretion
        racc = 0.        # No accretion

# Gas outflow via winds
        Z16Oej = Z16O     # Assume that ejected gas has same composition
                          # as current gas composition
        ZFeej = ZFe
        rwind = 0.        # No Winds

# Update variables
        dMg = (racc - rwind + et - SFR) * dt
        dMs = (SFR - et) * dt
        dM16O = (e16O - Z16O*SFR + Z16Oacc*racc - Z16Oej*rwind) * dt
        dMFe  = (eFe  -  ZFe*SFR +  ZFeacc*racc -  ZFeej*rwind) * dt
        Mg = Mg + dMg
        Ms = Ms + dMs
        M16O = M16O + dM16O
        MFe  = MFe  + dMFe

        at.append(t)
        aSFR.append(SFR)
        aMg.append(Mg)
        aMs.append(Ms)
        aZ16O.append(M16O/Mg)
        aZFe.append(MFe/Mg)

        print("%6.3e %11.5e %11.5e %6.3e %6.3e %6.3e %9.6e %6.3e %6.3e %9.6e" % (t,
        t = t + dt
```

```
    return

# Run the model

gce_model()
```

Appendix B

```
import numpy as np
import pylab
import matplotlib.pyplot as pl
import matplotlib.animation as animation
import sys
import math

# This code solves the restricted 3-body problem as in Toomre & Toomre
# (e.g. 1972, ApJ, 178, 623).
# It considers gravitational encounters between two galaxies. The bulk (halo)
# of each galaxy is representing by a central point mass.

# The two point masses arrive at the scence of the encounter surrounded
# by a flat annular disc of 120 non-interacting test particles. For details see
# section II of Toomre & Toomre 1972.

# Define 'Particle' class with mass, position, velocity and spin attributes
class Particle(object):
   def __init__(self, mass, pos, vel, spin):
      self.mass = float(mass)
      self.pos = np.array(pos, dtype='float')
      self.vel = np.array(vel, dtype='float')
      self.spin = float(spin)


# Define constants and units (use SI units for calculations)
G = 6.67e-11         # [N m^2/kg^2]
kpc_to_m = 3.086e19  # conversion factor betweenn kpc and m (for plotting)
yr_to_s = 3.15569e7  # conversion factor between yr and s
Msun = 1.989e30      # conversion factor between Msun and kg


# Define main simulation parameters
nb_rings  = 5                           # number of rings in the stellar disc
nb_stars_per_ring = [12, 18, 24, 30, 36] # number of stars in each ring
ring_delta_r  = 10 * kpc_to_m           # initial distance between rings
delta_t = 4e6 * yr_to_s                 # size of time steps
nb_steps   = 400                        # number of time steps of simulations
soft_length = 5e-4 * kpc_to_m           # gravitational softening length



# Compute the force applied on particle 2 exerted by particle 1
def grav_force(particle1, particle2):
```

```
    distance_vector = particle2.pos - particle1.pos
    distance_mag = np.sqrt(np.sum(distance_vector**2))
    force = G * particle1.mass * particle2.mass \
             * distance_vector / (distance_mag**2 + soft_length**2)**1.5
    return force


#Compute circular velocity
def circ_vel(r, Mass):
    return np.sqrt(G*Mass/r)


# Rotate coordinates
def rotate(xyzp, euler):
    theta = euler[0]*math.pi/180.
    psi   = euler[1]*math.pi/180.
    phi   = euler[2]*math.pi/180.
    c1 = math.cos(theta)
    c2 = math.cos(psi)
    c3 = math.cos(phi)
    s1 = math.sin(theta)
    s2 = math.sin(psi)
    s3 = math.sin(phi)

    l1 = c2*c3 - c1*s2*s3
    l2 = -c2*s3 - c1*s2*c3
    l3 = s1*s2
    m1 = s2*c3 + c1*c2*s3
    m2 = -s2*s3 + c1*c2*c3
    m3 = -s1*c2
    n1 = s1*s3
    n2 = s1*c3
    n3 = c1

    xp = xyzp[0]
    yp = xyzp[1]
    zp = xyzp[2]

    x  = l1*xp + l2*yp + l3*zp
    y  = m1*xp + m2*yp + m3*zp
    z  = n1*xp + n2*yp + n3*zp

    return np.array([x, y, z])

# Specify initial conditions for halos and stars
# euler1 and euler2 specify the Euler angles for the coordinate systems
# of galaxy1 and galaxy2 (nutation, precession, rotation).
```

```
# See Bronshtein et al., "Handbook of Mathematics" Section 3.5.3.2

def set_ICs(mass1 = 1e11, mass2 = 1e11, pos1 = [-50,-50,0], pos2 = [50, 50, 0],
              vel1 = [0,1e5,0], vel2 = [0,-1e5,0],
              euler1 = [0, 0, 0], euler2 = [0, 0, 0],
              spin1 = 1, spin2 = 1):
    #Halos
    mass1 = mass1*Msun
    mass2 = mass2*Msun

    pos1 = np.array(pos1) * kpc_to_m
    pos2 = np.array(pos2) * kpc_to_m

    vel1 = np.array(vel1)
    vel2 = np.array(vel2)

    spin1 = spin1
    spin2 = spin2

    halo1 = Particle(mass1, pos1, vel1, spin1)
    halo2 = Particle(mass2, pos2, vel2, spin2)
    halo_array = [halo1, halo2]
    euler      = [euler1, euler2]

    # Discs of stars
    star_array = []
    for halo in halo_array:
        star_array.append([])

    for halo in range(len(halo_array)):
        for ring in range(nb_rings):
            for star in range(nb_stars_per_ring[ring]):
                angle_star = 2*np.pi*star/nb_stars_per_ring[ring]
                radius_ring  = ring_delta_r * (ring + 1)
                star_pos = rotate(np.array([np.cos(angle_star),
                                  np.sin(angle_star), 0])*radius_ring,
                                  euler[halo]) + halo_array[halo].pos
                v_circ = circ_vel(ring_delta_r * (ring+1), halo_array[halo].mass)
                star_vel = rotate(np.array([-np.sin(angle_star),
                                  np.cos(angle_star), 0])*v_circ
                                  * halo_array[halo].spin, euler[halo]) \
                                 + halo_array[halo].vel

                star_particle =  Particle(1e7*Msun, star_pos, star_vel, 1)
                        # mass of star particle not
                        # relevant for restricted 3-body problem
                star_array[halo].append(star_particle)
```

```
    return halo_array, star_array


#Get positions of halos in the x-y plane
def halos_xy(halo_array):
   xpos_halo = [halo.pos[0] for halo in halo_array]
   ypos_halo = [halo.pos[1] for halo in halo_array]
   xy_pos = np.array([xpos_halo, ypos_halo]).T
   return xy_pos


#Get positions of star particles in the x-y plane
def stars_xy(star_array, halo):
   xpos_star = [star.pos[0] for star in star_array[halo]]
   ypos_star = [star.pos[1] for star in star_array[halo]]
   xy_pos = np.array([xpos_star, ypos_star]).T
   return xy_pos


#Integrate forward in time
def run_sim(t_step):
    # Compute gravitational acceleration and new
    # positions/velocities at each time step
    for halo in range(len(halo_array)):
        #Calculate gravitational force on halo
        if halo == 0: force = grav_force(halo_array[halo], halo_array[halo+1])
        if halo == 1: force = grav_force(halo_array[halo], halo_array[halo-1])
        acceleration_halo = force/halo_array[halo].mass

        #Update position and velocity of halo
        halo_array[halo].vel += acceleration_halo * delta_t
        halo_array[halo].pos += halo_array[halo].vel * delta_t

        #Update position and velocity of stars due to gravity of halos
        for star in star_array[halo]:
            # Calculate acceleration
            acceleration_star = (grav_force(star, halo_array[0])
                  + grav_force(star, halo_array[1])) / star.mass
            star.vel += acceleration_star*delta_t
            star.pos += star.vel*delta_t

    #Plot new time step
    halo_plot.set_offsets(halos_xy(halo_array)/kpc_to_m)
    star_plot0.set_offsets(stars_xy(star_array, 0)/kpc_to_m)
    star_plot1.set_offsets(stars_xy(star_array, 1)/kpc_to_m)
```

```
    time_count.set_text('t='+ str(float(t_step+1) * delta_t / yr_to_s/1e6)+' Myr')



################################
################################

#Set initial conditions
halo_array, star_array = set_ICs()
print 'Toomre simulation started...'

########
#Initialise plot
pl.clf()
fig = pl.figure(1)
pl.axis([-170, 170, -170, 170])

#Plot halos
halo_pos = halos_xy(halo_array)
halo_plot = pl.scatter(halo_pos[:,0]/kpc_to_m, halo_pos[:,1]/kpc_to_m,
               marker='o', s=[300 for halo in halo_array], c=['k','k'])

#Plot stars
star_pos0 = stars_xy(star_array, 0)
star_pos1 = stars_xy(star_array, 1)
star_plot0 = pl.scatter(star_pos0[:,0]/kpc_to_m, star_pos0[:,1]/kpc_to_m,
               marker='*', s=[50]*len(star_pos0), c=['b' for s in star_pos0])
star_plot1 = pl.scatter(star_pos1[:,0]/kpc_to_m, star_pos1[:,1]/kpc_to_m,
               marker='*', s=[50]*len(star_pos1), c=['r' for s in star_pos1])

pl.xlabel('X [kpc]', fontsize=16)
pl.ylabel('Y [kpc]', fontsize=16)
time_count = pl.text(70,140,'t= 0 Myr', fontsize=16)
#########

#Create animation and save
anim = animation.FuncAnimation(fig, run_sim, frames=nb_steps,
                   blit=False, repeat=False)

anim.save('toomre.mp4', writer='ffmpeg', fps=30)
print 'Done.'
```