2018/2019 Summer Studentship, Alexander Rohl

March 18, 2019

Superviser: Adam Stevens, GitHub: https://github.com/arhstevens

1 Introduction

Dark Sage is an updated version of the Semi-analytic Galaxy Evolution (SAGE) model [4] that considers galaxies made up of concentric discs [2]. Note that the original feedback prescription assumes stellar feedback to be a function of local gas density only [6].

Although DarkSage is calibrated and appears accurate at redshift zero, notice that the universally averaged star formation rate density (SFRD) peaks at a higher redshift and predicts more star formation activity at earlier epochs than observations suggest [2]. The aim of my project is to implement a new stellar feedback prescription that better predicts the trend on the right.

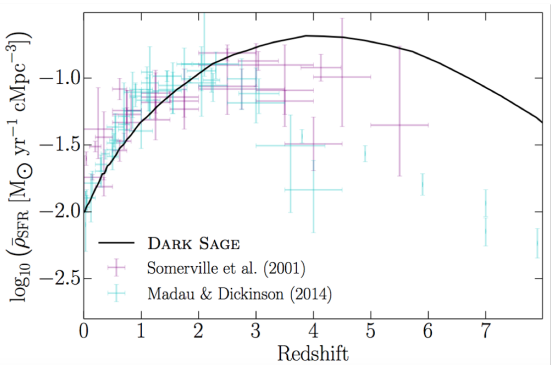

Figure 1: Star formation rate history

2 Modularising Feedback in DarkSage

The following segment from the original DarkSage code shows the original supernovae prescriptions in action.

```
//supernoveRecipeOn>0 -> stellar feedback
if(SupernovaRecipeOn > 0 && (Gal[p].DiscGas[i] > 0.0 && stars>
    MIN_STARS_FOR_SN) //so we have enough mass to make supernovas
{
    if(SupernovaRecipeOn == 1) {
    //uses equation in the paper for supernova feedback
    reheated_mass = FeedbackReheatingEpsilon * stars * pow(Sigma_0gas /
    (Gal[p].DiscGas[i]/area), FeedbackExponent);
    ...
} else if(SupernovaRecipeOn == 2) {
    //uniform reheated fraction
    reheated_mass = FeedbackReheatingEpsilon * stars;
    ...
} else {
    reheated_mass = 0.0;
}
```

This code is executed for each annulus and originally appeared in each of the modules: *model_starformation_and_feedback.c*, *model_disk_instability.c* and *model_mergers.c*. Therefore, in order to save time later down the track where we aim to add a third feedback prescription, it is essential that we modularise this process as much as possible.

To do so, I implemented a new module called *feedback_only.c* that within has a function called recipe that essentially runs the code segment above as well as adding the finer details unique to the three modules we are modularising for. These finer details are controlled by the integer variable **feedback_type** where for the cases in *model_starformation_and_feedback.c*, *model_disk_instability.c* and *model_mergers.c*, **feedback_type** is set to 1,−1 and 0 respectively.

```
def recipe(...
```

```
if(SupernovaRecipeOn > 0 && (Gal[p].DiscGas[i] > 0.0 && stars>
    MIN_STARS_FOR_SN) //so we have enough mass to make supernovas
{
...
}
...
// Inject new metals from SN II
if (feedback_type == 1) {
        metallicity = get_metallicity(NewStars[i],NewStarsMetals[i]);
        ...
}
//updating masses after ejection
if (feedback_type == -1) {
    update_from_ejection(p, centralgal, ejected_sum);
    ...
}
//angular momentum adjustments
if (feedback_type == 0)  {
    j_bin = (DiscBinEdge[i]+DiscBinEdge[i+1])*0.5;
    ...
}
```

Now that we have modularised the process for some independent annuli, later in the project we want to implement a prescription that considers the annuli as dependent, hence next we needed to modularise the entire process of iterating through the annuli as well. This was achieved by constructing the function feedback which simply loops through the annuli, calculates the variables required to execute recipe and runs the final computations required for *model_starformation_and_feedback.c* and *model_mergers.c*.

```
void feedback(...)
{
    ...
    // Loops through annuli
    for(i=0; i<N_BINS; i++)
    {
       if (feedback_type == 1) {
           ...
           strdot = SfrEfficiency * SFE_H2 * Gal[p].DiscH2[i];
           ...
           stars = strdot * dt; //dt time step -> total mass of stars
    formed
       } else if (feedback_type == 0) {
            if(mode == 1)
                eburst = disc_mass_ratio[i];
            else
                //equation in paper, model for stars forming in burst
                eburst = BetaBurst * pow(disc_mass_ratio[i], AlphaBurst)
    ;
            stars = eburst * Gal[p].DiscGas[i];
        }
        if(stars < MIN_STARFORMATION) //minimum threshold
                stars = 0.0;
        if(stars > Gal[p].DiscGas[i])
                stars = Gal[p].DiscGas[i];

        //Calls recipe for each disk,
        struct RecipeOutput output = recipe(p, centralgal, dt, step,
    NewStars, NewStarsMetals, stars_sum, metals_stars_sum, strdotfull,
    ejected_mass, ejected_sum, reheated_mass, metallicity, stars_angmom,
    i, stars, feedback_type, Gal[p].DiscGas[i], Gal[centralgal].Vvir);
```

```
27
28          stars = 1.0*output.stars;
29          stars_sum += stars;
30          ejected_sum += 1.0*output.ejected;
31          NewStars[i] = 1.0*output.NewStarsDisk;
32          NewStarsMetals[i] = 1.0*output.NewMetalsDisk;
33      }
34      if (feedback_type == 1) {
35          // Sum stellar discs together
36          if(NewStarSum>0.0)
37              combine_stellar_discs(p, NewStars, NewStarsMetals);
38          ...
39      } else if (feedback_type == 0) {
40           // Update bulge spin
41           ...
42           // Now adding all new stars directly to the bulge
43           ...
44       }
45  }
```

This modularisation and the implementation of the recipe and feedback functions are summarised in a flowchart in the appendices.

3 Feedback Recipe

The goal of this new feedback recipe is to implement physically sound energy arguments to model the distribution of energy and the amount of gas reheated or ejected as a result of this energy exerted on the system.

We achieve this by calculating the energy required to transport some unit of mass from one phase to the next; specifically, from the cold annuli to the hot halo (CtoH) and from the hot halo to the ejected component (HtoE). Then, we make the assumption that the energy released from supernovae in each annulus is initially used up to reheat the cold gas, and any excess energy is used to eject the hot gas. We also consider coupling parameters 'EnergyEfficiencyC2H' and 'EnergyEfficiencyH2E' to control the proportion of energy responsible for reheating the cold gas and ejecting the hot gas respectively.

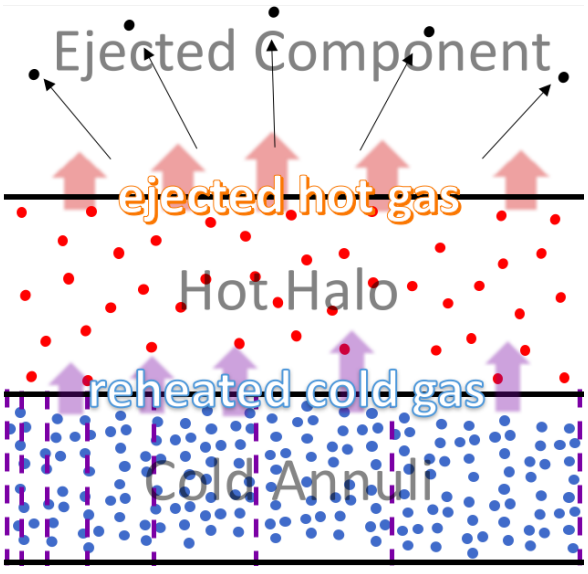

Figure 2: Gas Phases

Terms:
M_{vir}: Mass bounded by the virial radius
V_{vir}: Galaxy virial velocity
V_{rad}: Radial velocity
r_{annuli}: Annuli radius
R_s: Scale radius (written as R_2 in the code)
R_{hot}: Mean hot gas radius
R_{vir}: Virial radius
$\Phi(r)$: NFW Energy Potential
$\frac{E_{\text{cold}}}{M_{\text{cold}}}$: Energy per unit cold mass
$\frac{E_{\text{hot}}}{M_{\text{hot}}}$: Energy per unit hot mass

$\frac{E_{\rm ejected}}{M_{\rm ejected}}$: Energy per unit ejected mass
$\dot{E}_{\rm CtoH}$: Energy required to transport one unit from cold to hot
$\dot{E}_{\rm HtoE}$: Energy required to transport one unit from hot to ejected

We are firstly interested in calculating the total energy per unit mass present in each of the three gas phases. This energy is in the form of either gravitational potential energy or kinetic energy. We write this as:

$$\begin{aligned}\frac{E_{\rm cold}}{M_{\rm cold}} &= \frac{1}{2}V_{\rm rad}^2 + \Phi(r_{\rm annuli}) \\ \frac{E_{\rm hot}}{M_{\rm hot}} &= \frac{1}{2}V_{\rm vir}^2 + \Phi(r_{\rm hot}) \\ \frac{E_{\rm ejected}}{M_{\rm ejected}} &= \frac{1}{2}V_{\rm vir}^2 + \Phi(r_{\rm vir})\end{aligned}$$

Where we use the NFW density profile to calculate the gravitational potential energy. Set $c = R_{\rm vir}/R_s$.

$$\begin{aligned}\text{Given } M_{\rm vir} &= \int_0^{R_{\rm vir}} 4\pi r^2 \rho(r) dr \\ &= 4\pi\rho_0 R_s^3 \left[\ln(1+c) - \frac{c}{1+c}\right] \\ &= 4\pi\rho_0 R_s^3 \left(\ln\left(\frac{R_{\rm vir}+R_s}{R_s}\right) - \frac{R_{\rm vir}}{R_{\rm vir}+R_s}\right) \\ \text{and } \Phi(r) &= -\frac{4\pi G\rho_0 R_s^3}{r}\ln\left(1+\frac{r}{R_s}\right) \\ \text{we have } \Phi(r) &= -\frac{M_{\rm vir}}{r}\left(\ln\left(\frac{R_{\rm vir}+R_s}{R_s}\right) - \frac{R_{\rm vir}}{R_{\rm vir}+R_s}\right)^{-1} \times \left(\ln\left(1+\frac{r}{R_s}\right)\right)\end{aligned}$$

In the code, this is written as:

```
double NFW_profile(double radius, double R_vir, double M_vir, double r_2
    ) {
     double result = (-1) * (M_vir/radius) * pow(ln((R_vir + r_2)/r_2)-(
    R_vir/(R_vir + r_2)),-1) * ln(1 + radius/r_2);
     return result;
}
```

We calculate the scale radius (R_s) in the function CalcR_2:

```
double CalcR_2(int p) {
     // Determine the distribution of dark matter in the halo =====
     double M_DM_tot = Gal[p].Mvir - Gal[p].HotGas - Gal[p].ColdGas - Gal
    [p].StellarMass - Gal[p].ICS - Gal[p].BlackHoleMass; // One may want
    to include Ejected Gas in this too
     double baryon_fraction = (Gal[p].HotGas + Gal[p].ColdGas + Gal[p].
    StellarMass + Gal[p].ICS + Gal[p].BlackHoleMass) / Gal[p].Mvir;
     ...
     c_DM = pow(10.0, a+b*log10(Gal[p].Mvir*UnitMass_in_g/(SOLAR_MASS*1
    e12))); // Dutton & Maccio 2014
     if(Gal[p].Type==0 && Gal[p].StellarMass>0 && Gal[p].Mvir>0)
         c = c_DM * (1.0 + 3e-5*exp_f(3.4*(X+4.5))); // Di Cintio et al
    2014b
```

```
    else
        c = 1.0*c_DM; // Should only happen for satellite-satellite
    mergers, where X cannot be trusted
    r_2 = Gal[p].Rvir / c; // Di Cintio et al 2014b
    return r_2;
}
```

Finally, $R_{\rm hot}$ is calculated by assuming the hot gas lies at some average hot halo radius:

$$\begin{aligned} &\text{Since } \rho_{\rm hot}(r) \propto r^{-2} \\ \text{we have the mean radius, } R_{\rm hot} &= \frac{\int_0^{R_{\rm vir}} r \cdot \rho_{\rm hot}(r)\, dV}{\int_0^{R_{\rm vir}} \rho_{\rm hot}(r)\, dV} \\ &= \frac{\int_0^{R_{\rm vir}} 4\pi r dr}{\int_0^{R_{\rm vir}} 4\pi dr} \\ &= \frac{R_{\rm vir}}{2} \end{aligned}$$

Now that we have the energy per unit masses for each gas phase, we calculated the energy required to move some unit mass from hot to cold and from hot to ejected by

$$\begin{aligned} \dot{E}_{\rm CtoH} &= \frac{E_{\rm hot}}{M_{\rm hot}} - \frac{E_{\rm cold}}{M_{\rm cold}} \text{ and} \\ \dot{E}_{\rm HtoE} &= \frac{E_{\rm ejected}}{M_{\rm ejected}} - \frac{E_{\rm hot}}{M_{\rm hot}}, \text{ respectively.} \end{aligned}$$

Clearly, we need an expression for the amount of energy released by the supernovae in the first place. Referring back to the original sage paper [4], we use the following conversion from stellar mass to energy released by supernovae.

$$E_{\rm SN} = \frac{1}{2}\dot{m}_* V_{\rm SN}^2,\ V_{\rm SN} = 630 {\rm kms}^{-1}$$

Finally, we compute the amount of disk gas reheated and the amount of hot gas ejected by:

$$\begin{aligned} E_{out} &= \epsilon_{\rm efficiencyC2H} \times E_{SN} \\ E_{excess} &= E_{out} - \dot{E}_{CtoH} \times M_{cold} \\ M_{reheated} &= \frac{E_{out}}{\dot{E}_{CtoH}} \\ \text{If } E_{excess} > 0,\ M_{ejected} &= \frac{\epsilon_{\rm efficiencyH2E} \times E_{excess}}{\dot{E}_{HtoE}} \end{aligned}$$

This is summarised in the following code fragment located in the recipe function in *feedback_only.c* where this prescription is executed if **SupernovaRecipeOn** = 3.

```
if(SupernovaRecipeOn == 3) { //new case
  //EnergyEfficiencyC2H: proportion of SN energy going into reheating
    gas
  //EnergyEfficiencyH2E: proportion of excess energy going into ejecting
    gas
  double energy_in =  0.5 * 630 * 630 * stars;  //e_SN, Croton et al.
    pg8
```

```
  double energy_out = EnergyEfficiencyC2H * energy_in;

  //Energy per unit mass for each state: Cold, Hot, Ejected
  double energy_perunit_coldmass =
            0.5 * (V_rot*r_av/Gal[p].Rvir) * (V_rot*r_av/Gal[p].Rvir)
            + NFW_profile(r_av, Gal[p].Rvir, Gal[p].Mvir, CalcR_2(p));

  double r_hot = 0.5 * Gal[p].Rvir;
  double energy_perunit_hotmass =
            0.5 * V_rot * V_rot
            + NFW_profile(r_hot, Gal[p].Rvir, Gal[p].Mvir, CalcR_2(p));

  double energy_perunit_ejectedmass =
            0.5 * V_rot * V_rot
            + NFW_profile(Gal[p].Rvir, Gal[p].Rvir, Gal[p].Mvir, CalcR_2
   (p));

  //Find the energy required to move between states
  double energy_c2h = energy_perunit_hotmass - energy_perunit_coldmass;
  double energy_h2e = energy_perunit_ejectedmass -
   energy_perunit_hotmass;
  assert(energy_h2e>=0.0);
  assert(energy_out>=0.0);

  reheated_mass = energy_out/energy_c2h;
}
// Can't use more cold gas than is available, so balance SF and feedback
if((stars + reheated_mass) > gas_sf && (stars + reheated_mass) > 0.0)
{
        fac = gas_sf / (stars + reheated_mass);
        stars *= fac;
        reheated_mass *= fac;

        energy_excess = energy_out - reheated_mass * energy_c2h;
        if (energy_excess < 0.0) energy_excess = 0.0;
        ...
} else {
        if (SupernovaRecipeOn == 3) {
                //new ejected_mass argument
                if (energy_excess > 0)  {
                        ejected_mass = (EnergyEfficiencyH2E *
   energy_excess) / energy_h2e;
                } else {
                        ejected_mass = 0.0;
                }
         }
         ...
}
```

4 Allowing for Annular Dependence

Now we aim to construct the feedback recipe such that we can allow energy to spread from one annulus to another. This means that we need to know the energy present in each annulus simultaneously in order to update each annulus correctly as the energy is distributed. The difficulty with implementing this is that the procedure of functions required to compute each annuli energy is also responsible for updating related global fields, hence this energy amount would essentially be lagging behind other processes in the model. .

In order to achieve this, I constructed a new module named *feedback_annuli_dispersion.c* which has two new functions recipe_dispersed and feedback_dispersed. recipe_dispersed is identical to recipe, aside from the fact that now we read-in the **annuli_energy** rather than computing the **energy_in** from the **stars** variable inside recipe. To resolve the issue of 'lagging behind the rest of the model' feedback_dispersed now pre-calculates the **stars** variable to construct the **all_stars array**, which we can then use to construct **annuli_energy array** from the E_{SN} derived earlier, which is then afterwards fed into the recipe_dispersed function. We see this in the following code fragment:

```
void feedback(...)
{
    ...
    // Loops through annuli
    for(i=0; i<N_BINS; i++)
    {
       if (feedback_type == 1) {
           ...
           stars = strdot * dt; //dt time step -> total mass of stars
   formed
       } else if (feedback_type == 0) {
            ...
            stars = eburst * Gal[p].DiscGas[i];
        }
        if(stars < MIN_STARFORMATION) //minimum threshold
                stars = 0.0;
        if(stars > Gal[p].DiscGas[i])
                stars = Gal[p].DiscGas[i];

        //New arrays holding the annuli stars/energy data
        all_stars[i] = 1.0*stars;
        annuli_energy_init[i] = 0.5 * 630 * 630 * stars;
    }
    // Loops through annuli again
    for(i=0; i<N_BINS; i++)
    {
        //Calls recipe for each disk
        double energy = 1.0*annuli_energy_init[i];
        double stars = 1.0*all_stars[i];
        struct RecipeOutput output = recipe_dispersed(..., energy);
        ...
    }
    ...
}
```

Whilst this works for calling recipe_dispersed inside of feedback_dispersed, we still need to call recipe_dispersed inside of deal_with_unstable_gas. In other words, we need to pre-calculate the **annuli_energy array** before calling deal_with_unstable_gas. This was done by writing two new functions count_stars_mergers and count_disk_stars located in *model_mergers.c* and *model_disk_instability.c*, respectively. Essentially, these functions construct the **annuli_energy array** and **all_stars array** without altering any global data, allowing the annuli energy to interact dependently. This process is summarised in a flowchart in the appendices.

5 Energy Distribution

This section will discuss how we distribute the energy across annuli. This is done in the following function, located in *feedback_annuli_dispersion.c*. We then discuss the mathematics implemented in this function.

```
void distribute_energy(double all_stars[N_BINS], double
    annuli_energy_init[N_BINS], int p) {
  ...
  for(i=0; i<N_BINS; i++) {
    //Calculate volumetric proportions for each annulus
    inner = i-1;
    outer = i+1;
    h_inner = 11*11 / (2*M_PI * GRAVITY * (all_stars[inner]+Gal[p].
    DiscGas[inner]));
    h_outer = 11*11 / (2*M_PI * GRAVITY * (all_stars[outer]+Gal[p].
    DiscGas[outer]));
    energy = 1.0*annuli_energy_init[i];
    ...
    double innerVol = (M_PI/3) * (r_av - r_inner) * (r_av + 2*r_inner) *
    h_inner;
    double outerVol = (M_PI/3) * (r_outer - r_av) * (r_av + 2*r_outer) *
    h_outer;
    double totalVol = (M_PI/3) * (r_outer - r_inner) * (h_inner*(r_inner
    + 2*r_outer) + h_outer*(r_outer + 2*r_inner));
    if (totalVol > 0.0) {
      remainVol = totalVol - innerVol - outerVol;
      inner_prop = innerVol / totalVol;
      remaining_prop = remainVol / totalVol;
      outer_prop = outerVol / totalVol;
    }
    ...
    //Disperse inward energy towards the center
    disperse_energy_inward(energy * inner_prop, annuli_energy_final,
    inner);
    //Update the annulus energy based on the remaining proportion
    annuli_energy_final[i]     += energy * remaining_prop;
    //Disperse outward energy towards edge of galaxy
    disperse_energy_outward(energy * outer_prop, annuli_energy_final,
    outer);
  }
}
```

On the right, we have a side-on view of an annulus. We consider the supernovae to occur at some average radius (R_{av}). From here we consider the energy released to either eject up/down and thus remain in the annulus, move inwards and dissipate towards the centre of the galaxy, or move outwards and dissipate towards the outer edge of the galaxy.

To do so, we construct a geometric argument where we denote R_{in}, R_{out}, H_{in} and H_{out} to be the inner radius length, outer radius length, inner disk height and outer disk height respectively.

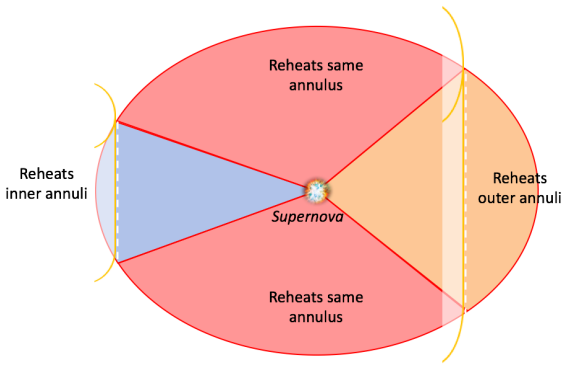

Figure 3: Side-on Annulus

We calculate this partitioning of the annuli as follows:

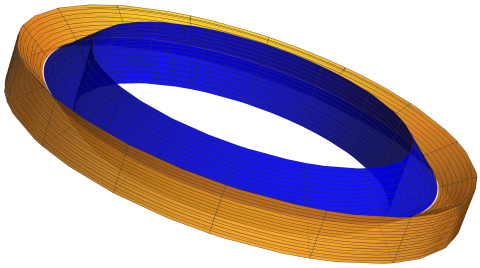

Figure 4: Inner Volume (blue).
Outer Volume (orange).

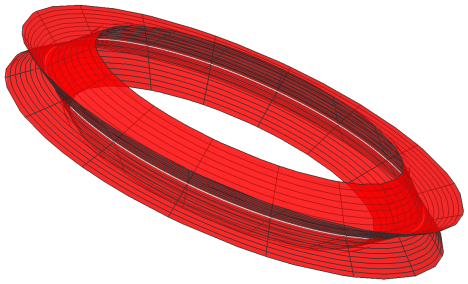

Figure 5: Remaining Volume.

$$
\begin{aligned}
\text{Disk Height} &= \frac{\sigma_z^2(R)}{2\pi G\Sigma(R)} = \frac{11^2}{2\pi G\left(stars_{\text{disk}} + gas_{\text{disk}}\right)} \\
\text{InnerVol} &= 2\pi \int_0^{\frac{H_{\text{in}}}{2}} \left(\left(R_{\text{av}} - \frac{h(R_{\text{av}} - R_{\text{in}})}{\frac{H_{\text{in}}}{2}} \right)^2 - R_{\text{in}}^2 \right) dh \\
&= \frac{\pi}{3} H_{\text{in}}(R_{\text{av}} - R_{\text{in}})(R_{\text{av}} + 2R_{\text{in}}) \\
\text{OuterVol} &= 2\pi \int_0^{\frac{H_{\text{out}}}{2}} \left(R_{\text{out}}^2 - \left(\frac{h(R_{\text{out}} - R_{\text{av}})}{\frac{H_{\text{out}}}{2}} + R_{\text{av}} \right)^2 \right) dh \\
&= \frac{\pi}{3} H_{\text{out}}(R_{\text{out}} - R_{\text{av}})(R_{\text{av}} + 2R_{\text{out}}) \\
\text{TotalVol} &= \pi \left(R_{\text{out}}^2 - R_{\text{in}}^2 \right) H_{\text{in}} + 2\pi \int_0^{\frac{H_{\text{out}} - H_{\text{in}}}{2}} \left(\left(R_{\text{out}} - \frac{h(R_{\text{out}} - R_{\text{in}})}{\frac{H_{\text{out}} - H_{\text{in}}}{2}} \right)^2 - R_{\text{in}}^2 \right) dh \\
&= \frac{\pi}{3}(R_{\text{out}} - R_{\text{in}})(H_{\text{in}}(R_{\text{in}} + 2R_{\text{out}}) + H_{\text{out}}(2R_{\text{in}} + R_{\text{out}})) \\
\text{RemainingVol} &= \text{TotalVol} - \text{OuterVol} - \text{InnerVol}
\end{aligned}
$$

We then simply take the proportion as the ratio of each volume with respect to the total volume. Now that some proportion of the energy released is directed inwards and some outwards, we consider an exponential decay model to disperse this energy out to the innermost and outermost annuli respectively. This is defined recursively in the following code fragments.

```
1  //applies the exponential decay model until the outermost annulus is
       reached
2  void disperse_energy_outward(double energy, double annuli_energy[N_BINS
       ], int affected_index) {
3      if (affected_index < N_BINS) {
4          double new_energy = EnergyEfficiencyC2H * energy;
5          double excess_energy = energy - new_energy;
6          annuli_energy[affected_index] += new_energy;
7          //recursive call
8          disperse_energy_outward(excess_energy, annuli_energy,
       affected_index+1);
9      }
10     return;
11 }
12
```

```
//applies the exponential decay model until the innermost annulus is reached
void disperse_energy_inward(double energy, double annuli_energy[N_BINS], int affected_index) {
    if (affected_index >= 0) {
        double new_energy = EnergyEfficiencyC2H * energy;
        double excess_energy = energy - new_energy;
        annuli_energy[affected_index] += new_energy;
        //recursive call
        disperse_energy_inward(excess_energy, annuli_energy, affected_index-1);
    }
    return;
}
```

6 Results

The following plot clearly shows the improvement of the new model. However, there is still the issue of recalibrating the other plots as shown below as well.

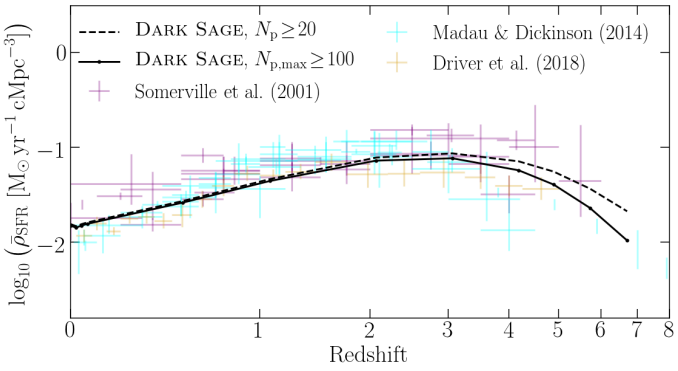

Figure 6: Updated star formation rate density plot

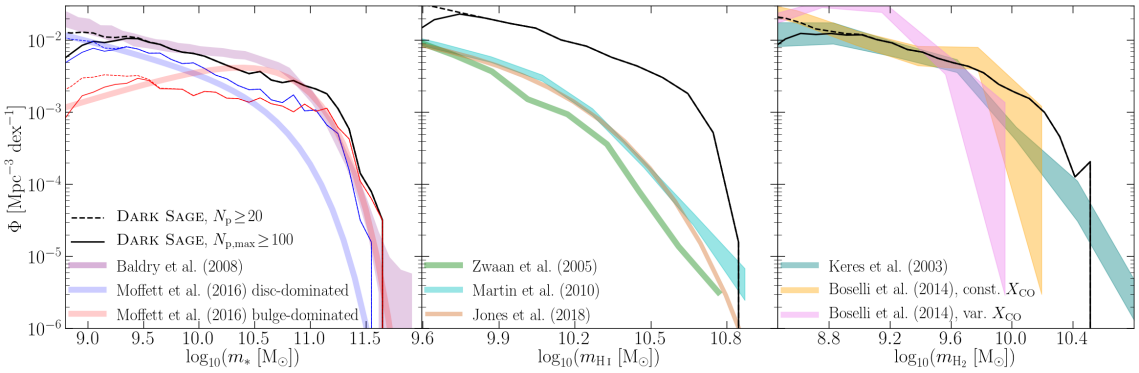

Figure 7: Stellar mass function for updated model

7 Summary

We have successfully implemented a new energy model that uses a physically motivated energy argument to better predicts universal star formation density at earlier epochs. Because of practical time constraints, parts of the project fell outside of the scope. If time permitted, the next step would be to do a full recalibration of the model on both the *millennium* data set and the *SMDPL* data set. This would be followed by further rigorous testing of the new feedback scheme.

The new structure of the implementation also opens up exciting possibilities to easily change the equations governing energy flow between annuli and easily alter equations controlling how the gas is moving between the cold, hot and ejected phases.

References

[1] C. M. Baugh *A primer on hierarchical galaxy formation: the semi-analytical approach.* https://arxiv.org/pdf/1605.00647.pdf, 2016.

[2] Adam R. H. Stevens, Darren J. Croton and Simon J. Mutch *Building disc structure and galaxy properties through angular momentum: The Dark Sage semi-analytic model.* https://arxiv.org/pdf/1605.00647.pdf, 2016.

[3] Adam R. H. Stevens, Claudia del P. Lagos, Danail Obreschkow and Manodeep Sinha *Connecting and dissecting galaxies? angular momenta and neutral gas in a hierarchical universe: cue Dark Sage.* arxiv.org/pdf/1806.07402.pdf, 2018.

[4] Darren J. Croton, Adam R. H. Stevens, Chiara Tonini, Thibault Garel, Maksym Bernyk, Antonio Bibiano, Luke Hodkinson, Simon J. Mutch, Gregory B. Poole and Genevieve M. Shattow *Semi-analytic Galaxy Evolution (SAGE): model calibration and basic results.* https://arxiv.org/pdf/1601.04709.pdf, 2016.

[5] Philip F. Hopkins, Andrew Wetzel, Duan Kere, Claude-Andr FaucherGigure, Eliot Quataert, Michael Boylan-Kolchin, Norman Murray, Christopher C.Hayward and Kareem El-Badry *How To Model Supernovae in Simulations of Star and Galaxy Formation.* https://arxiv.org/pdf/1707.07010.pdf, 2018.

[6] Jian Fu, Qi Guo, Guinevere Kauffmann and Mark R. Krumholz *The atomic-to-molecular transition and its relation to the scaling properties of galaxy discs in the local Universe.* Monthly Notices of the Royal Astronomical Society, Volume 409, Issue 2, 1 December 2010.

8 Appendix

Figure 8: Colour Scheme

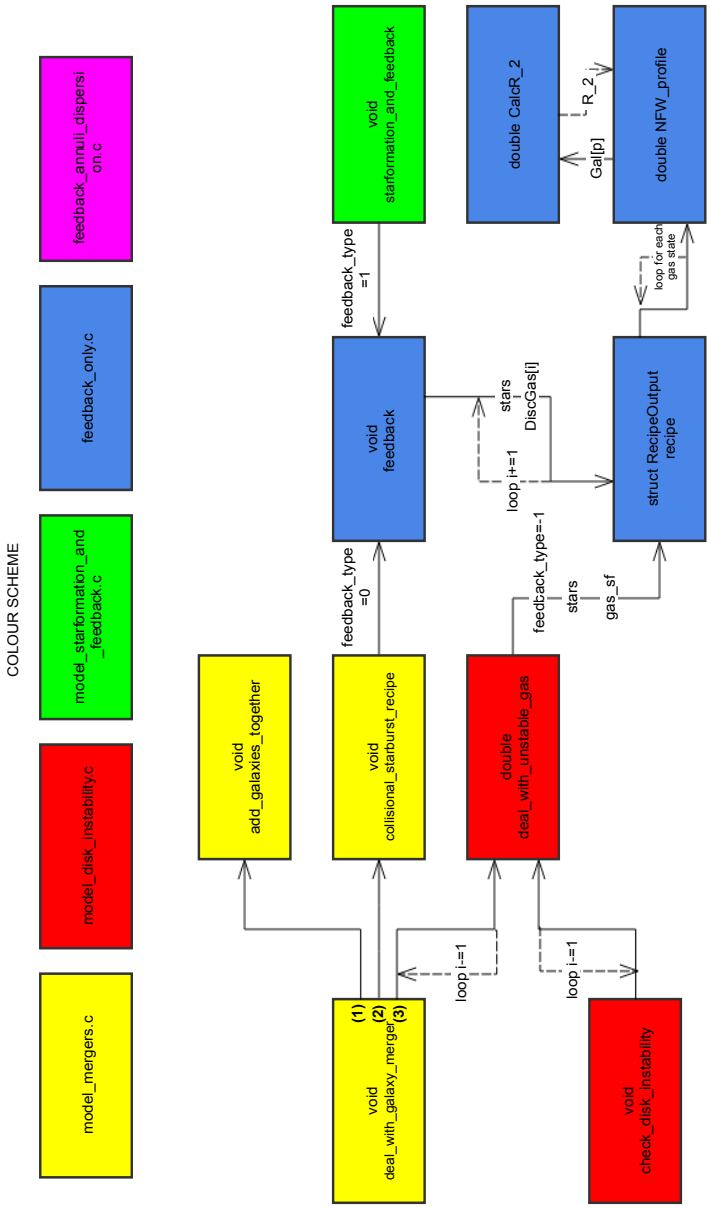

Figure 9: Function diagram for EnergyDispersion = 0 (i.e. restricted to annulus)

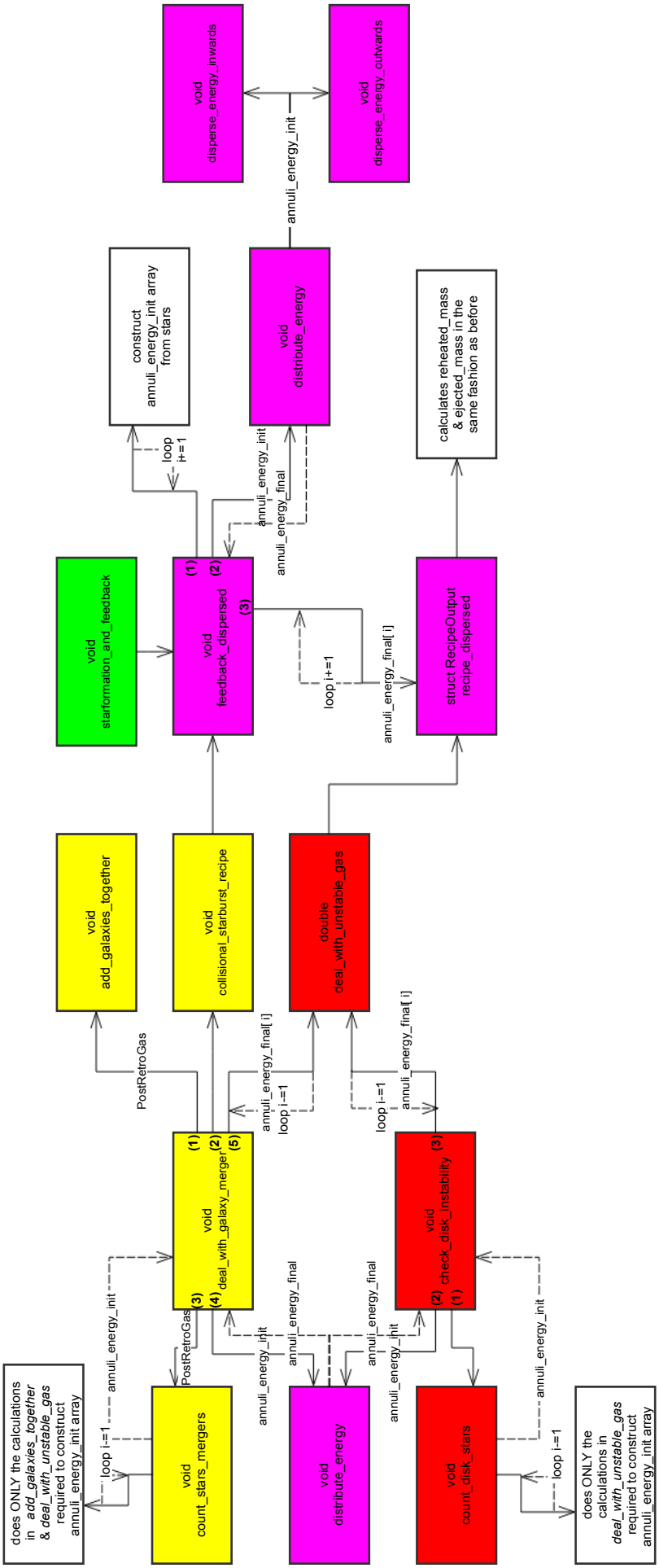

Figure 10: Function diagram for EnergyDispersion = 1 (i.e. dispersed across annuli)