Simulating Black Hole Accretion & Feedback During a Collision with a Star

Sarah Seitanakis

Department of Physics
Allegheny College
Meadville, Pennsylvania

April 2019

Simulating Black Hole Accretion & Feedback During a Collision with a Star

By
Sarah Seitanakis
Department of Physics
Allegheny College
Meadville, Pennsylvania

April, 2019

Submitted in fulfillment of the senior thesis requirements of the
Department of Physics at Allegheny College
and approved by the senior thesis committee.

Dr. Lombardi	Date	Dr. Poynor	Date

Pledge

Name: Sarah Seitanakis
Major: Physics
Thesis Committee: Dr. Lombardi, Dr. Poynor

Date: Spring, 2019

Abstract

In this project, three different accretion feedback models have been implemented to *StarSmasher*, a smoothed particle hydrodynamics code. These models can be used to explore the possibility of the formation of gamma ray bursts or other electromagnetic signatures during black hole to star collisions. The three different models have been tested using a 15 solar mass black hole and an 8 solar mass star. These results have been compared to the results from a version of *StarSmasher* that does not have an accretion feedback mechanism. There is a spike at early times in the feedback energy rate, which may be consistent with gamma ray bursts. Additionally, when the star completely surrounds the blackhole there is a sustained period of feedback.

Acknowledgments

Pursuing physics at Allegheny College has been a transformative journey. To all the professors in the department: I thank you for pushing me to be the best I can be.

Dr. Lombardi
Thank you for teaching me everything I know about Astrophysics. Researching in your lab has been the most exciting and fulfilling experience in my academic career. I am grateful to have spent two summers researching under your supervision. You are a wonderful professor and you have truly changed how I approach problems.

Dr. Poynor
Thank you for being so supportive of me, through thick and thin. *Thank god* you have tolerated me for four years. You have given me a reason to never give up, even when I really thought I would quit. Thank you for teaching me perseverance (and Classical Mechanics—loved that class). Thank you for telling me that I can do anything I put my mind to.

Liam, Dipto, Nick, Richie, Josh, Kadeem, Anthony, Harry
You guys are my physics fam. Thank you for sticking with me through the late nights, rough classes, and seemingly endless work. Thank you for sharing laughs, through the good times and the bad... and especially every Friday over donuts. We ride together; we die together. May the heavens shine a light on your beautiful souls. And that goes for every student in the Physics department too—Stay strong, little buggers.

Patrick, Justin, Ethan, Charlie, Andrea, and Ben
Thank you for keeping me sane and helping me escape the stress by going on our DnD adventures every week.

My family
Thank you for being so supportive of me throughout my years of schooling. Would not have made it this far without you.

Benjamin
Thanks for being my rock.

Contents

Chapter 1

Introduction & Background

The mechanism of black hole accretion and feedback is still a mystery. I am working on one piece of the puzzle in hopes that it will lead to a clear picture. For my senior project, I am implementing a model that more accurately treats stellar collisions involving a black hole. This model estimates the 'sub-grid' phenomenon (a phenomenon on a smaller scale than is currently resolvable in a simulation) of black hole accretion during a collision between black hole and a star.

1.1 Black Holes (BHs)

A black hole (BH) forms from a high-mass star at the end of its life. After a high mass star evolves to a certain point, iron builds up in the core of the star (Bennett et al. *The Cosmic Perspective*). Eventually, the pressure from the outside becomes too great for the electrons to move freely around the iron nuclei. At this point, the electrons will fall down onto their iron nuclei, interacting with protons to form neutrons. Note that when electrons and protons combine, a neutron as well as a neutrino form. Now, with the reduction of electrons, the electron degeneracy pressure vanishes. Since the force that was keeping gravity from crushing the star further (this aforementioned "electron degeneracy pressure") has gone, the core collapses. The collapse results in tremendous amounts of energy causing a huge explosion, a.k.a. a supernova, throwing off mass in all directions in the form of ejecta. All that is left behind is a big ball of mostly neutrons (these are the same neutrons that form when the electrons fall onto the nuclei and join with the protons that I referred to above), which is called a neutron star (NS). Sometimes, the star lives on as a NS. However, sometimes this ball of neutrons is so massive ($\gtrsim 3$ *solar masses*, $M_{\odot}$) that the gravity is so strong that it overcomes the *neutron* degeneracy pressure. This causes the star to continue to collapse *to the extreme*, turning itself into a black hole. It is expected that all matter that makes up a BH must crush to an infinitely tiny and dense point at the center; this point is called *singularity* (Bennett et al. *The Cosmic Perspective*). Now that I have gone over BH formation, I will expound on what we know about BHs post-formation.

Note that non-rotating BHs are spherical in shape (Bennett et al. *The Cosmic Perspective*). One important feature of a BH is the event horizon. The event horizon is the border that determines what is 'inside' and what is 'outside' the BH. The location of the event horizon is where the velocity required for an object to escape the gravitational pull of the BH is equal to the speed of light. The *radius* of the event horizon is known as the "Schwarzschild radius". Oftentimes, the Schwarzschild radius is thought of as a BH's "size" (Bennett et al. *The Cosmic Perspective*). The calculation of the Schwarzschild radius depends only on the mass of the particular BH. To further illustrate this property, I will find the Schwarzschild radius of a black hole in terms of its mass, the gravitational constant, and the speed of light. To do this, I would like us to imagine for a moment that we have a projectile, with mass m and we shoot this from a planet with radius R and mass M_{BH}. At the final time, our projectile is infinitely far from everything and has come to a stop so that it has a total energy of zero ($E_{\mathrm{final}} = 0$). From our knowledge of mechanics, we know that the initial energy of the projectile when it is shot off of the planet is given by the following

$$E_{\mathrm{initial}} = \frac{1}{2}mv^2 - \frac{GM_{\mathrm{BH}}m}{R}, \tag{1.1}$$

where v is the escape velocity of the projectile. Because energy must be conserved, the initial energy must equal the final energy. So, we have

$$\frac{1}{2}mv^2 - \frac{GM_{\mathrm{BH}}m}{R} = 0. \tag{1.2}$$

Normally, we would solve for the escape velocity of the particle. But I am going to solve for R so that we can relate this to the Schwarzschild radius of a black hole. We have

$$R = \frac{2GM_{\mathrm{BH}}}{v^2}. \tag{1.3}$$

Now, in order to relate this to the Schwarzschild radius, R_{s}, of a black hole, we need to imagine that instead of a projectile being shot off of the surface of a planet, the projectile is really shot from the event horizon of a black hole (at R_{s}). The escape velocity of a particle at R_{s} is the speed of light, c. So if $R = R_{\mathrm{s}}$, then

$$R_{\mathrm{s}} = \frac{2GM_{\mathrm{BH}}}{c^2}. \tag{1.4}$$

The relationship between the Schwarzschild radius R_{S}, the mass of the BH M_{BH}, and the mass of the Sun $M_{\odot}$ is given by

$$R_{\mathrm{S}} = \frac{M_{BH}}{M_{\odot}} * (2.95\ \mathrm{km}), \tag{1.5}$$

where 2.95 km is the Schwarzschild radius of a BH with the same mass as the Sun (Bennett et al. *The Cosmic Perspective*).

1.2 Gamma Ray Bursts (GRBs)

A gamma-ray burst (GRB) is a quick flare of gamma-rays. There are a few different categories of GRBs. A long GRB usually lasts anywhere from two seconds to a couple of minutes (Elías & Martínez 2018). A short GRB has a duration of less than two seconds; this type of GRB can last as little as a few milliseconds (Elías & Martínez 2018).

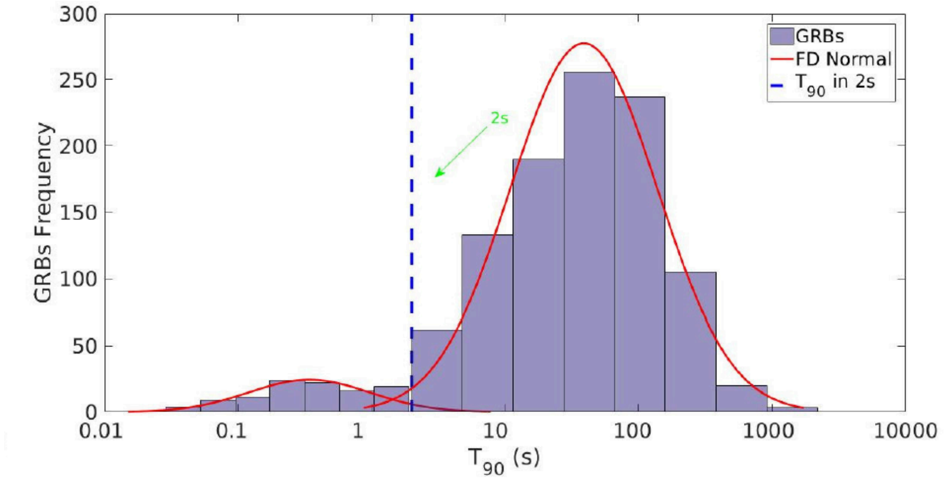

Figure 1.1: This figure shows a two-part distribution of a sample of 994 observed GRBs. The x-axis shows the duration of a GRB in seconds, and the y-axis shows the frequency of GRBs with given duration. The dotted blue line, at 2 seconds, is the cutoff between what is considered a short GRB and long GRB. The red line is a normalized curve that represents the frequency of GRBs (within the sample) with respect to duration in seconds. One can see that both short GRBs (duration < 2 s) and long GRBs (duration > 2 s) are present in this sample. Figure from Elías & Martínez (2018).

From Figure 1.1, we can see that short GRBs most often have a duration of about 0.6 seconds, while long GRBs most frequently have a duration of about a minute. The isotropic energy of long and short GRBs can range from $10^{50} - 10^{52}$ ergs (Elías & Martínez 2018; Gehrels et al. 2004). Observation has shown that *some* GRBs seemed to come from very powerful supernovas that form BHs (Bennett et al. *The Cosmic Perspective*). GRBs that occur during the formation of a BH are typically long GRBs. The *Chandra* and *Hubble* telescopes have collected observational data that show that some GRBs are *not* generated by explosions of massive stars (Bennett et al. *The Cosmic Perspective*). Among these GRBs from unknown sources are what appear to be "ultra-long" GRBs, which have durations of greater than 1000 seconds and can last several thousand seconds (Dagoneau, et al. 2018). Although there are several hypotheses about what could be generating these bursts, the main functions that cause them are still unknown. Perets et al. (2016) puts forth the idea that when a compact object (a BH, white dwarf, or NS) tidally disrupts a star (or planet), a debris disc may form around the compact object. Additionally, Perets et al. (2016) posits that the BH's efficient accretion of debris from this disc may be the cause of events that seem like ultra-long GRBs. It is suggested that micro tidal disruption events (which could be caused by a close encounter of a star with a compact object) may cause very energetic flares which share characteristics with GRBs (gamma ray bursts) or XRFs (X-ray flares) except they last longer and are fainter than most GRB/XRF (Perets et. al 2016).

1.3 Smoothed Particle Hydrodynamics (SPH)

Smoothed Particle Hydrodynamics (SPH) programs use hydrodynamics, the study of how fluids move. Because we are interested in simulating stars, this 'fluid' is gas. It is a 'Smoothed Particle' code because each 'particle' represents a blob of fluid, and these particles interact with each other (see Figure 1.2).

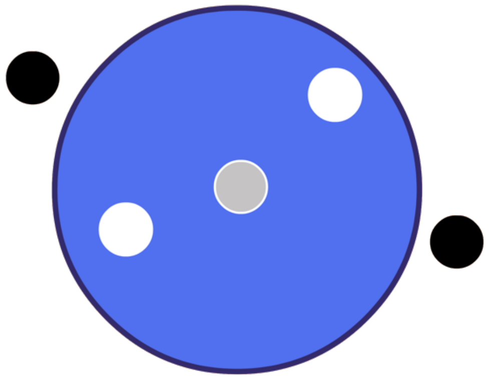

Figure 1.2: Each particle has a region around it called a 'kernel'. The kernel that belongs to the gray particle is represented by the blue circle. When other particles (see white circles) are *inside* the gray particle's kernel, they *are* taken into account when calculating the properties of the gray particle such as hydrodynamic acceleration and density. When other particles (see black circles) are *outside* of the kernel, they have *no impact* on the properties of the gray particle, although they do contribute to its gravitational acceleration. Figure from my 2017 summer work.

1.4 Background Theory

My senior project takes advantage of many core physics concepts. One of these important concepts is Newton's Law of Gravitation, which is a key component of the *StarSmasher* calculations. *StarSmasher* is a hydrodynamic code that runs a series of calculations to simulate the motion of a system of particles. Newton's law of gravity tells us that the gravitational forces between two objects with masses m_1 and m_2 is given by the equation

$$\vec{F}_{12} = -\frac{Gm_1m_2}{r^2}\hat{r}_{12}\,, \tag{1.6}$$

where $\overrightarrow{F}_{12}$ is the force on object 1 from object 2, G is the gravitational constant, r is the distance between the objects, and $\hat{r}_{12}$ is the direction vector pointing from object 1 to object 2. From classical mechanics we know that potential energy W has the following relationship with force:

$$W(\bar{r}) = -\int_{\infty}^{\bar{r}} \overrightarrow{F} \cdot \mathrm{d}\overrightarrow{r}, \tag{1.7}$$

where the limits of integration ensure the usual convention that $U = 0$ at infinity. So, from Newton's Law of Gravitation, I can calculate gravitational potential energy which is given by

$$W = -\frac{Gm_1m_2}{r}. \tag{1.8}$$

In *StarSmasher*, the total gravitational potential energy, which is summed over all particles, is one of the values outputted, along with other types of energy such as the total internal energy, total kinetic energy, and the sum of all three of these energies.

From Newton's laws of motion, we know that kinetic energy of a particle with mass m and speed v is given by $T = \frac{1}{2}mv^2$. Note that in *StarSmasher*, the total kinetic energy is the kinetic energy summed over all particles. Additionally, the versions of *StarSmasher* that I am using have an analytic equation of state. So, the specific internal energy (internal energy per unit mass of particle i) is given by

$$u_i = \frac{3}{2}\frac{kT_i}{\mu_i m_{\mathrm{H}}} + \frac{aT_i^{\ 4}}{\rho_i}, \tag{1.9}$$

where k is the Boltzmann constant, T_i is the temperature of particle i, μ_i is the mean molecular mass of particle i, m_{H} is the mass of hydrogen, ρ_i is the density at particle i, and a is a radiative constant. For the total internal energy U, we have

$$U = \sum_{i=1}^{N} m_i u_i, \tag{1.10}$$

where N is the total number of particles, m_i is the mass of particle i, and u_i is defined by Equation 1.9.

Bear with me for a moment while I go through an example using simple physics; my goal in doing so will be to make sense of an equation from the Methodology section which describes the time-dependent change in energy of a particle being accreted onto a black. The aforementioned equation shows the following relationship:

$$\dot{E}_{\mathrm{fb}} \propto \dot{M}_{\mathrm{BH}} c^2, \tag{1.11}$$

where E_{fb} is the feedback energy, M_{BH} is the mass of the black hole, c is the speed of light, and the dots represent the time derivative. In my senior project, I will be modeling the change in energy over time of a particle that is being accreted onto a black hole. To illustrate this, we will consider the situation where we have a particle with mass ΔM_{BH} initially at infinity, so the gravitational potential energy is zero ($W_{\mathrm{init}} = 0$). Note that I am using this name for the particle mass because we are pretending that it is a small amount of mass that will soon be accreted onto a black hole. At the final time we will bring the particle close to a black hole with mass M_{BH} with $W_{\mathrm{final}} = -\frac{GM_{\mathrm{BH}}(\Delta M_{\mathrm{BH}})}{R_{\mathrm{s}}}$. Hence,

$$\Delta W = W_{\mathrm{final}} - W_{\mathrm{initial}}, \tag{1.12}$$

$$\Delta W = -\frac{GM_{\mathrm{BH}}(\Delta M_{\mathrm{BH}})}{R_{\mathrm{s}}} \tag{1.13}$$

In Equation 1.4 we saw that $R_{\mathrm{s}} = \frac{2GM_{\mathrm{BH}}}{c^2}$. Substituting this into the Equation 1.13, we have

$$\Delta W = -\frac{GM_{\mathrm{BH}}(\Delta M_{\mathrm{BH}})c^2}{2GM_{\mathrm{BH}}}. \tag{1.14}$$

Making the appropriate cancellations, we have

$$\Delta W = -\frac{1}{2}\Delta M_{\mathrm{BH}} c^2. \tag{1.15}$$

Now, if we divide both sides by a change in time Δt, we have the following:

$$\frac{\Delta W}{\Delta t} = -\frac{1}{2}\frac{\Delta M_{\mathrm{BH}}}{\Delta t} c^2, \tag{1.16}$$

$$\frac{\Delta W}{\Delta t} \propto -\frac{\Delta M_{\mathrm{BH}}}{\Delta t} c^2, \tag{1.17}$$

$$\dot{W} \propto -\dot{M}_{\mathrm{BH}} c^2, \tag{1.18}$$

Note that a fraction of the energy that the particle lost becomes available as feedback energy. Hence,

$$\dot{E}_{\mathrm{fb}} \propto -\dot{W}. \tag{1.19}$$

Substituting Equation 1.18 into Equation 1.19, we now have

$$\dot{E}_{\mathrm{fb}} \propto \dot{M}_{\mathrm{BH}} c^2. \tag{1.20}$$

Now, note that Equation 1.20 is the same as Equation 1.11. Hence, I have shown how this equation, which describes the time-dependent change in energy that results from mass being accreted onto a black hole, fits in with the knowledge I have acquired while studying physics at Allegheny College.

Chapter 2

Methods and Methodology

2.1 StarSmasher

StarSmasher is a three-dimensional SPH code. To better understand how *StarSmasher*, I will walk through how a one-dimensional SPH code works. First, parameters are established in the simulation (including values such as the number of particles and desired final time). *StarSmasher* updates these values by calculating the velocity, position, and specific internal energy of each particle. Then, the code repeats these calculations over, and over, for every particle. The code will then move the model forward in time a very small amount. This process repeats, slowly marching the system forward until the final time is reached. Then, to help check if the program works properly, I look at energy and momentum conservation of the system; the goal is that the momentum and energy are conserved (or nearly conserved). Then, I compare the results to an analytic solution found in literature (Rasio & Shapiro 1991, see Figure 3a and 3b). Below are several figures from my 2017 summer work. During this time, I worked with Mitchell Latchet and Brandon Kallams to write our own one-dimensional SPH code. Figures 2.1a and 2.1b are outputs from the code that we wrote.

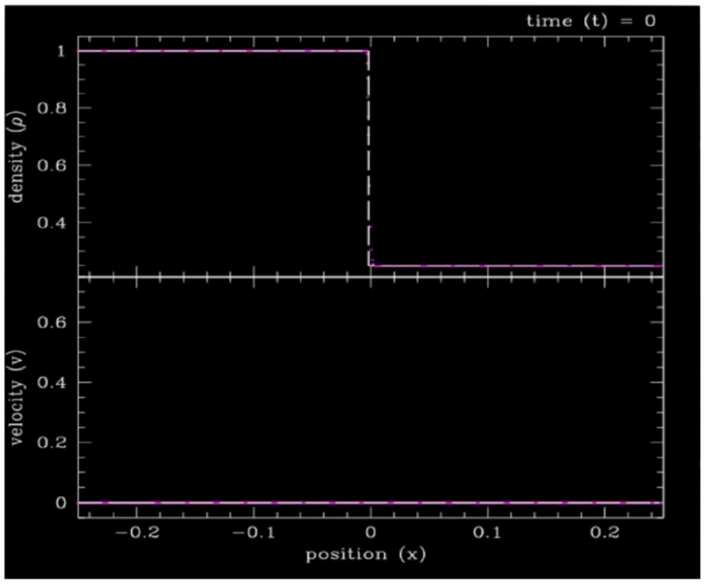

Figure 2.1a: This figure shows the initial conditions of our simulation, as shown by the time in the top right hand corner of the graph. The top graph shows the density of our gas (y-axis) at different positions (x-axis). These graphs compare the output of our code (pink data) to the analytic solution (white dashed line). The density graph shows that half of the space is filled with dense gas (left) and the other half is filled with not very dense gas (right). The bottom graph shows the velocity of the particles (y-axis) at different positions (x-axis). The velocity at this time is constant at zero, which means that none of the particles are moving at first. When we march this forward in time, there will be a wind from left to right as the gas dynamically evolves.

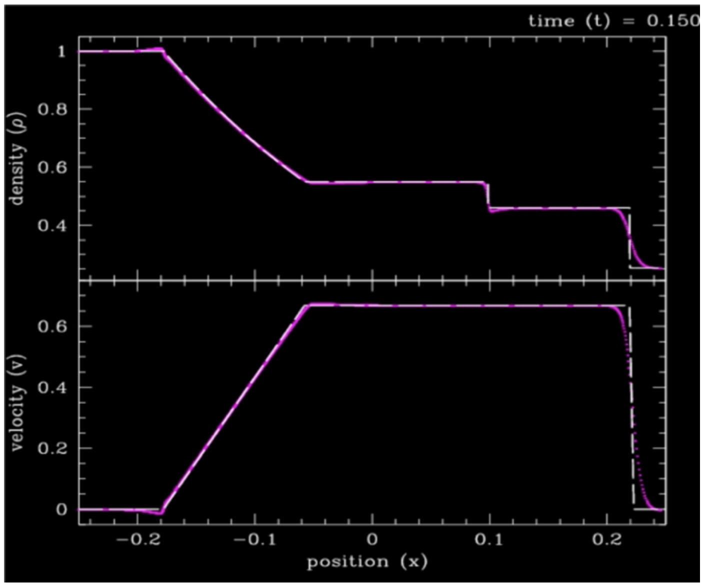

Figure 2.1b: This figure shows the final output of our simulation. These graphs compare the output of our code (pink data) to the analytic solution (white dashed line). As is shown from both the density and velocity graphs, our data are a good match to the analytic solution until the sharp corners on the right side of the graph, where some smoothing occurs. We can reduce smoothing by increasing the number of particles used in our simulation.

Three-dimensional SPH codes, like *StarSmasher*, work much the same. An SPH program starts by using the initial conditions as a 'starting point' for calculating new values, the code calculates values for each particle. The hydro code then marches the model forward in time— the previously new values become the 'starting point' and more new values are calculated from there. At regular time intervals, an output file gets dumped from the code. These output files act as snap-shots of the model; they contain all kinds of useful data about each particle: the position, velocity, acceleration, potential energy, etc.

StarSmasher runs calculations using user specified units. The default for mass units m_{unit} is a solar mass and the default unit for length r_{unit} is solar radii. Also, the default unit for time t_{unit}, energy E_{unit}, and luminosity L_{unit} are all arrangements of r_{unit}, m_{unit}, and G. So, that is

$$\begin{gathered}
m_{\text{unit}} = 1\ \mathrm{M}_\odot = 1.99 \times 10^{33}\ \mathrm{g}, \\
r_{\text{unit}} = 1\ \mathrm{R}_\odot = 6.96 \times 10^{10}\ \mathrm{cm}, \\
t_{\text{unit}} = \sqrt{\frac{{r_{\text{unit}}}^3}{G \cdot m_{\text{unit}}}} = \sqrt{\frac{(1\ \mathrm{R}_\odot)^3}{G \cdot (1\ \mathrm{M}_\odot)}} = 1590\ \mathrm{s}, \\
E_{\text{unit}} = \frac{G \cdot {m_{\text{unit}}}^2}{r_{\text{unit}}} = \frac{G \cdot (1\ \mathrm{M}_\odot)^2}{(1\ \mathrm{R}_\odot)} = 3.79 \times 10^{48}\ \mathrm{erg}, \\
L_{\text{unit}} = \frac{G \cdot {m_{\text{unit}}}^2}{r_{\text{unit}}} \cdot \sqrt{\frac{G \cdot m_{\text{unit}}}{{r_{\text{unit}}}^3}} = \frac{G \cdot (1\ \mathrm{M}_\odot)^2}{(1\ \mathrm{R}_\odot)} \cdot \sqrt{\frac{G \cdot (1\ \mathrm{M}_\odot)}{(1\ \mathrm{R}_\odot)^3}} = 2.38 \times 10^{45}\ \mathrm{erg/s}.
\end{gathered} \tag{2.1}$$

Throughout my project, I use these default units unless otherwise stated.

For those interested in understanding how to use the code, I will refer you to the *StarSmasher* student guide that I created with Dr. Lombardi. The guide can be found at this address: http://starsmasher.allegheny.edu/seitanakiss/StarSmasherOnline/Student_tutorial/StarSmasher_Setup_students.html

2.2 The Eddington Limit

The accretion rate of a BH is limited because the outward flow of photons released from accretion exerts a force on the incoming matter. Assume that there are free electrons and protons from ionized gas surrounding a BH. When a particle interacts with a photon, the photon exerts a force on that particle by transferring momentum. The rate of momentum transfer depends on the particle's cross-section for interaction with photons, the area in which an interaction between the photon and the particle can occur. This cross-section is inversely related to the particle's mass squared (Ryden & Peterson 2010). Since an electron is less massive than a proton, the cross-section for interaction with photons of an electron is larger than that of a proton. Hence, the source of opacity in an ionized gas is from electron scattering of photons. So photons are transferring the most momentum to electrons. The cross-section for an electron, or the Thomson cross-section σ_T is given by

$$\sigma_T = \frac{8\pi}{3}\left(\frac{e^2}{4\pi\epsilon_0 m_e c^2}\right)^2 = 6.65 \times 10^{-29}\ \mathrm{m}^2\,, \tag{2.2}$$

where ϵ_0 is the permeability of free space, c is the speed of light, m_e is the mass of an electron, and e is the charge of an electron.

The Eddington luminosity refers to the maximum possible luminosity a black hole can have. This luminosity occurs when the BH's inward gravitational force is exactly equal in magnitude to the BH's outward radiation force. I will use the example found in Foundations of Astronomy (page 498) to show how to derive an equation for the maximum accretion rate of a BH (the Eddington Limited accretion rate). Suppose we have a black hole with mass M_{bh} with luminosity, L given as

$$L = \epsilon \dot{M} c^2 \, , \tag{2.3}$$

where ϵ represents the “efficiency” of the BH and is a dimensionless number, $\dot{M}$ is the mass accretion rate, and c is the speed of light. Assume, for simplicity, the BH is surrounded by pure ionized hydrogen, which is ionized due to the BH heating infalling matter. At a distance r from the BH, photons have an energy flux Φ_{energy} given by

$$\Phi_{\text{energy}} = \frac{L}{4\pi r^2}, \tag{2.4}$$

where L is the luminosity of the BH. Since each photon has a certain energy E, it also has momentum $p = E/c$. Hence, the momentum flux of the outward flow of electrons is

$$\Phi_{\text{momentum}} = \frac{\Phi_{\text{energy}}}{\text{c}} = \frac{L}{4\pi r^2 c} \, . \tag{2.5}$$

Now, the amplitude of the force an electron experiences due to radiation pressure is the momentum flux multiplied by the Thomson cross-section σ_{T} (see Equation 2.2) (since the cross-section for protons is much smaller than it is for electrons, the cross-section for the proton-electron pair is approximately equal to the cross-section of an electron, which is the Thomson cross-section σ_{T});

$$F_{\text{radiation}} = \sigma_{\text{T}} \Phi_{\text{momentum}} = \frac{\sigma_{\text{T}} L}{4\pi r^2 c} \, . \tag{2.6}$$

We must compare the radiation force to the force of gravity, which we will now calculate using Newton’s Law of Gravitation. Note that the mass of the proton-electron pair is the sum of the electron’s mass and the proton’s mass; since the electron mass is negligible in comparison to the mass of the proton, the mass of the pair is approximately equal to the proton’s mass. The force of gravity F_{g} on the electron-proton pair is given by

$$F_{\text{g}} = - \frac{G M_{\text{bh}} m_{\text{p}}}{r^2} \, , \tag{2.7}$$

where G is the gravitational constant, M_{bh} is the mass of the BH, m_{p} is the mass of a proton, and r is the distance between the BH and the electron-proton pair. As stated before, the Eddington luminosity L_{E} occurs when the BH's inward gravitational force is exactly equal in *magnitude* to the BH's outward radiation force $\left(|F_{\mathrm{g}}| = |F_{\mathrm{radiation}}| \right)$. Hence,

$$\frac{GM_{\mathrm{bh}}m_{\mathrm{p}}}{r^2} = \frac{\sigma_{\mathrm{T}}L_{\mathrm{E}}}{4\pi r^2 c}, \qquad and$$

$$L_{\mathrm{E}} = \frac{4\pi c G M_{\mathrm{bh}} m_{\mathrm{p}}}{\sigma_{\mathrm{T}}}. \tag{2.8}$$

The Eddington luminosity L_{E} gives a maximum accretion rate $\dot{M}_{\mathrm{Edd}}$, which we can find by rearranging Equation 2.3:

$$\dot{M}_{\mathrm{Edd}} = \frac{L_{\mathrm{E}}}{\epsilon c^2}. \tag{2.9}$$

Substituting Equation 2.8 in for L_{E}, we have

$$\dot{M}_{\mathrm{Edd}} = \frac{4\pi G M_{\mathrm{bh}} m_{\mathrm{p}}}{\epsilon \sigma_{\mathrm{T}} c} \tag{2.10}$$

(Ryden & Peterson 2010). Note that, when the scaling factor $\alpha_{\mathrm{E}} = 1$ in Equation 2.14b, Equation 2.10 is identical to Equation 2.14b. As you can see from Equation 2.10, all of the variables in $\dot{M}_{\mathrm{Edd}}$ are constants, with the exception of M_{bh}.

Here I make the argument that, for my purposes, we may consider M_{bh} in Equation 2.10 as a constant as well. Even the most extreme accretion rate, the super-Eddington accretion rate in terms of M_{bh} is given as follows (note that the following has $\epsilon = \epsilon_{\mathrm{r}}$ and the extreme case $\alpha_{\mathrm{E}} = 100$ for a super-Eddington rate; see Equation 2.14b):

$$\dot{M}_{\mathrm{Edd}} = \frac{4\pi\alpha_{\mathrm{E}} G m_{\mathrm{p}}}{\epsilon_{\mathrm{r}} \sigma_{\mathrm{T}} c} M_{\mathrm{bh}} = (7.03 \times 10^{-14}\ \mathrm{s}^{-1}) M_{\mathrm{bh}} = (1.12 \times 10^{-10}\ t_{\mathrm{unit}}{}^{-1}) M_{\mathrm{bh}}, \tag{2.11}$$

where t_{unit} is defined by Equation 2.1. This shows that for each time unit that the code goes through, a negligible percentage of the BH mass is added to the BH due to accretion even with super-Eddington accretion. Therefore, it is acceptable for my code to not update the BH mass at every timestep. The BH mass that I use in the simulations is $M_{\text{bh}} = 15.0\ \text{M}_{\odot} = 2.98 \times 10^{34}$ g. The quotient $M_{\text{bh}}/\dot{M}_{\text{Edd}}$ represents the time it would take for the BH to double in mass, assuming it is accreting mass at the highest rate possible (at the Eddington limited rate with a scaling factor of $\alpha_{\text{E}} = 100$) over the whole time period. Using the BH mass that I use for my simulations, we have

$$\frac{M_{\text{bh}}}{\dot{M}_{\text{Edd}}} = 8.92 \times 10^{9}\ t_{\text{unit}}\,. \tag{2.12}$$

Since my simulation only runs for $1000\ t_{\text{unit}}$ and $8.92 \times 10^{9}\ t_{\text{unit}} \gg 1000\ t_{\text{unit}}$, we can have even more confidence that the mass of the BH does not need to be updated for the timescale of my simulations. Hence, for my simulations, M_{bh} is constant at $15.0\ \text{M}_{\odot}$, which gives the analytical value of the Eddington limited mass accretion rate as

$$\dot{M}_{\text{Edd}} = 2.10 \times 10^{19}\ \text{g/s}\,, \qquad where\ \alpha_{\text{E}} = 1 \tag{2.13a}$$

$$and \quad \dot{M}_{\text{Edd}} = 2.10 \times 10^{21}\ \text{g/s}\,, \qquad where\ \alpha_{\text{E}} = 100. \tag{2.13b}$$

2.3 Accretion & Feedback

As a black hole accretes matter, the matter radiates and heats the surrounding gas. In the model that I am implementing into *StarSmasher*, a BH will accrete matter from surrounding gas (a.k.a. surrounding SPH particles) according to an analytical equation, which I will discuss below. Note that the total number of particles in a simulation using this model will remain constant. The equation I will be using to model the BH accretion rate is as described by the Bondi-Hoyle formula

$$\dot{M}_{\mathrm{B}} = \frac{4\pi\alpha G^2 M_{\mathrm{BH}}^2 \rho}{({c_{\mathrm{s}}}^2 + v^2)^{3/2}} \ , \tag{2.14a}$$

where M_{BH} is the mass of the black hole, G is the gravitational constant, c_{s} is the local speed of sound of the gas, v is the relative velocity of the BH to the gas, ρ is the local density of the gas, and $\alpha = 100$, which is a unit-less scaling factor created to account for underestimation of the local density that occurs at our working resolution (Liu et. al 2016). This accretion rate describes the rate at which mass is added to the BH over time (hence the dot). Due to the Eddington Limit, the amount of mass accreted at any given time is restricted. One can imagine a situation where a large amount of mass is being accreted onto the BH. This causes a consequentially large amount of radiation to be outputted by the BH. However, when radiation is being released, we run into this idea of "radiative pressure". Radiative pressure is pressure caused by radiation pushing back on the gas being accreted. This causes gas to slow, leading to a limit on the rate of mass accretion onto the BH. Hence, the accretion rate cannot be greater than the following due to the Eddington Limit:

$$\dot{M}_{\mathrm{Edd}} = \frac{4\pi\alpha_E G M_{\mathrm{BH}} m_{\mathrm{p}}}{\epsilon_{\mathrm{r}} \sigma_{\mathrm{T}} c} \ , \tag{2.14b}$$

where ϵ_{r} is the efficiency of the radiation (described below), σ_{T} is the Thomson cross-section (see Equation 2.2), m_{p} is the mass of a proton, and c is the speed of light (Liu et al. 2016), and α_E is a scaling factor. This scaling factor α_E is a first attempt to account for super-Eddington accretion (Brightman et al. 2019). In my project, I test both $\alpha_E = 1$, which is the conventional treatment, and $\alpha_E = 100$ which is the super-Eddington treatment (see Chapter 4). In the model, the accretion rate is set to the lower value, $\dot{M}_{\mathrm{B}}$ or $\dot{M}_{\mathrm{Edd}}$ (Liu et al. 2016). Hence, the accretion rate of the BH is calculated in the code by

$$\dot{M}_{\mathrm{BH}} = \min\left(\dot{M}_{\mathrm{Edd}}, \dot{M}_{\mathrm{B}}\right), \tag{2.15}$$

where $\dot{M}_{\mathrm{B}}$ and $\dot{M}_{\mathrm{Edd}}$ are defined by Equations 2.14a and 2.14b respectively. Recall that as mass is accreted onto a BH, the BH radiates energy. In Liu et al. (2016), the paper presents $\epsilon_{\mathrm{r}} = 0.1$ as the fraction of mass (from in-falling gas particles) that gets converted into photon energy by the BH, so this is what I use here. Additionally, this paper uses ϵ_{f} as the portion of the energy produced by the BH that couples to the surrounding gas. The rate at which the BH gives off energy as it accretes matter is given by the equation used by Liu et. al (2016) as follows:

$$\dot{E}_{\mathrm{fb}} = \epsilon_{\mathrm{r}}\epsilon_{\mathrm{f}}\dot{M}_{\mathrm{BH}}c^2 , \tag{2.16}$$

where $\epsilon_{\mathrm{r}} = 0.1$, $\epsilon_{\mathrm{f}} = 0.05$, $\dot{M}_{\mathrm{BH}}$ is defined by Equation 2.15, and c is the speed of light.

2.4 Implementation of Accretion Feedback

I will implement three different models similar to those used in Liu et al. (2016). The three used in Liu et al. (2016) are the following: the kernel (K), spline (S), and tophat (T) feedback models. The kernel feedback model is standard— energy is given to surrounding gas particles that are within the BH's SPH kernel. The energy for each particle is weighted depending on the SPH kernel function and their mass. The K model is a "fixed mass" scheme. The S and T feedback models use the "fixed volume" approach. For the S model, energy is given to particles within a fixed, user-defined, feedback radius. The amount of energy given to each particle is weighted the same way as it is for the K model. The T model uses a user-defined feedback radius, just like the S model. However, unlike the S model, the T model only uses particle mass to weight the energy per particle.

The way we will distinguish between the three models from each other is by having different specific internal energy (internal energy per unit mass) values, more precisely, specific internal energy due to feedback, which is a component of the specific internal energy.

Motivated by Liu et al. (2016), we present the following similar methods of calculating the time derivative of the specific internal energy for particle i due to BH feedback $\dot{u}_{\mathrm{fb},i}$:

$$\dot{u}_{\mathrm{fb},i} = \dot{E}_{\mathrm{fb}} \frac{W(\bar{r}_i - \bar{r}_{\mathrm{BH}}, h_i)}{\sum_{j=1}^{N} m_j W(\bar{r}_j - \bar{r}_{\mathrm{BH}}, h_j)}, \qquad \textit{(PK model, 2.17)}$$

$$\dot{u}_{\mathrm{fb},i} = \dot{E}_{\mathrm{fb}} \frac{1}{M_{\mathrm{enclosed\ in\ }R_{\mathrm{fb}}}}, \qquad \textit{(T model, 2.18)}$$

$$\dot{u}_{\mathrm{fb},i} = \dot{E}_{\mathrm{fb}} \frac{W(\bar{r}_i - \bar{r}_{\mathrm{BH}}, R_{\mathrm{fb}})}{\sum_{j=1}^{N} m_j W(\bar{r}_j - \bar{r}_{\mathrm{BH}}, R_{\mathrm{fb}})}, \qquad \textit{(S model, 2.19)}$$

where W is a weighting function (the same weighting function used for an SPH kernel), $\bar{r}_i$ is the position vector of particle i, $\bar{r}_{\mathrm{BH}}$ is the position vector of the BH, h_i is the smoothing length for particle i (half the radius of the kernel of the particle), $\dot{E}_{\mathrm{fb}}$ is defined by Equation 2.16, N is the total number of particles, R_{fb} is a fixed feedback radius, and $M_{\mathrm{enclosed\ in\ }R_{\mathrm{fb}}}$ is the mass enclosed in the feedback radius. Here, our PK model (or particle kernel model) differs from the K model used in Liu et al. (2016). For the PK model, we use the kernels of *SPH particles* (rather than the kernel of the BH) to determine whether energy is coupled to a given particle. Hence, for our PK model, when the BH is inside of the kernel of an SPH particle, feedback energy will be imparted to that particle. Note that for the T model, this $\dot{u}_{\mathrm{fb},i}$ is only applied to particles inside of the fixed feedback radius R_{fb}. Additionally, note that for all three models, these equations (2.17, 2.18, and 2.19) are designed such that the sum of the pieces is equal to the whole. That is,

$$\dot{E}_{\mathrm{fb}} = \sum_{i}^{N} m_i \dot{u}_{\mathrm{fb},i} \,. \qquad (2.20)$$

Prior to my project, the specific internal energy in StarSmasher $\dot{u}_{\mathrm{SPH}}$ includes adiabatic expansion and compression plus shock heating (for the full expression, see Appendix A of Gaburov et al. 2010). So, when I implement the additional component of the specific internal energy $\dot{u}_{\mathrm{fb}}$ to account for feedback energy, the new equation for the specific internal energy of particle i is

$$\dot{u}_i = \dot{u}_{\mathrm{SPH},i} + \dot{u}_{\mathrm{fb},i} \, , \tag{2.21}$$

where $\dot{u}_{\mathrm{fb},i}$ is defined by Equation 2.17, 2.18, 2.19 for the PK model, T model, and S model respectively.

2.5 Test Simulations

I test the different accretion feedback models by running simulations with the same star, black hole, and parameters for each of the three different models, as well as a run with no accretion feedback model (this acts as the control case). I use a 15 $\mathrm{M}_\odot$ black hole, a 8 $\mathrm{M}_\odot$ star with a radius of 3.821 $\mathrm{R}_\odot$, a periastron separation of 3.9 $\mathrm{R}_\odot$ (*note to future students*: this is the parameter in the sph.input file called "bimpact"), an initial separation of 10 $\mathrm{R}_\odot$, and a semi-major axis of 20 $\mathrm{R}_\odot$. I have done my initial tests and each different treatment appears to run properly, to my knowledge. I compare results of these tests in Chapter 4. Also note that I have used an analytic equation of state (rather than tabulated) for all of my simulations, as I have had more luck with the analytic equation of state when running simulations involving a BH.

Chapter 3

Preliminary Results

3.1 Comparing Versions of StarSmasher

In order to have a solid foundation off of which I can build, I have performed tests on two different versions of the code. I have done so to determine which version I will use to implement my accretion feedback models. The main difference between the two versions that I have been comparing is the way in which the smoothing lengths of particles (h) are handled. There are two ways of dealing with the smoothing lengths of particles. In the old version of the code, the smoothing length is allowed be zero. In the context of a collision involving a black hole and a star, allowing the smoothing length to be zero can lead to particles clumping onto each other around the black hole. I will refer to this old version of the code as "unlimited-h". In the newer version of the code, we set minimum and maximum values for the smoothing length. Theoretically, this will prevent clumping around the black hole. I will refer to this newer version of the code as "limited-h". Both versions of the code use a second-order integrator. For a second-order integrator, when the time step is doubled in length, the error in energy conservation (the change in total energy) should be quadrupled. Inversely, when the time-step is halved, the change in total energy should be quartered. In order to halve the time-steps in *StarSmasher*, one must double the set of "safety factors" that we call "cn values". To check the performance of the integrators in the two versions of *StarSmasher*, I performed a set of twelve simulations (six simulations using *unlimited-h* and six simulations using *limited-h*) in which I test the energy conservation at different average time-steps (see Figure 3.1). For all of these simulations, I used a star with 10,000 particles, mass 8 $M_{\odot}$, and radius 3.821 $R_{\odot}$ (relaxed using a profile file called "m_8_t_5.7.last1.muse_s2mm"). I used a black hole with mass 10 $M_{\odot}$. I set the initial separation between the star and the BH to 20 $R_{\odot}$, the periastron separation to 3.9 $R_{\odot}$, and the semi major axis to 20 $R_{\odot}$.

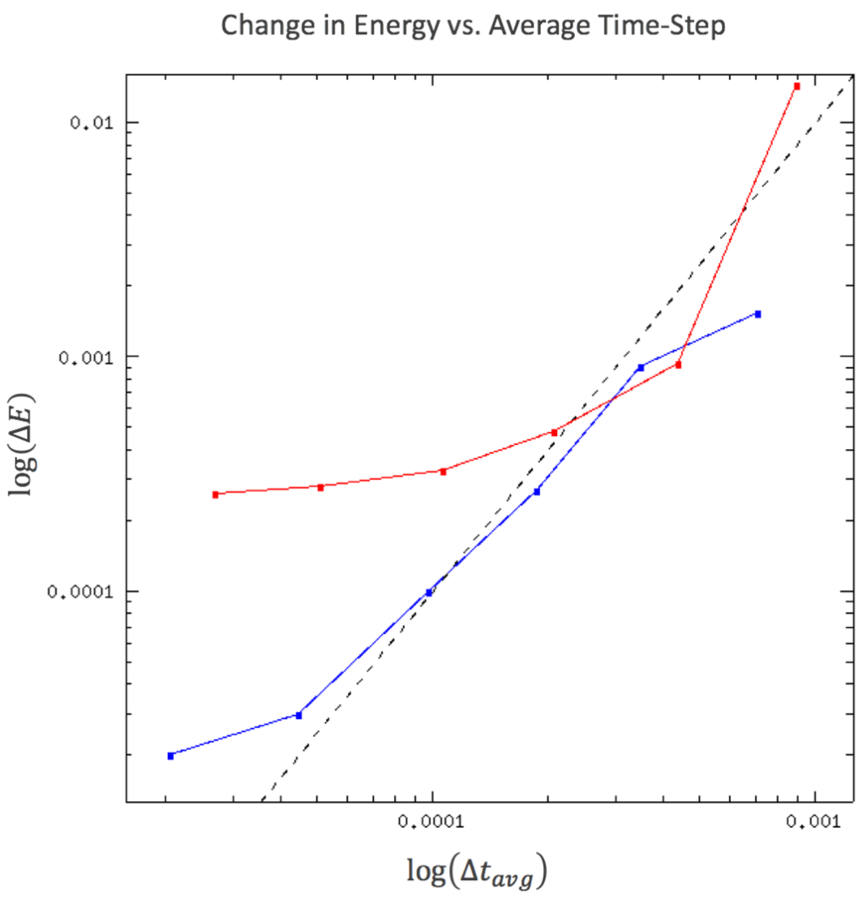

Figure 3.1: This figure shows error in energy conservation ΔE verses the average time-step of the run, both scaled logarithmically. We hope to see quadratic behavior with the relationship between ΔE and average time-step. Since we have a logarithmic scale, a quadratic relationship will appear as a line with a slope of two (see black dashed line). The red data are from *unlimited-h* runs, and the blue data are from *limited-h* runs. I am still considering the error in the two sets of data, which is why there are no error bars at this time. The right-most red point in the figure does not appear to fit in with the rest of the red data. This is possibly due to the timescale becoming too large, causing the code to hit a numerical instability, which leads to poor energy conservation (large ΔE). Note that the units of time and energy are defined by Eq. 2.1.

Note that we found ΔE in Figure 3.1 by subtracting the minimum total energy from the maximum total energy that occurred up to time $t = 180$, and Δt_{avg} is the average time-step up to time $t = 180$. As we can see in Figure 3.1, the blue data (*limited-h*) follows the dashed line that has a slope of 2 (as is expected for a second-order integration scheme). It is wise to note that the left-most blue point may be inaccurate due to round-off error. Notice that the red data (*unlimited-h*) does not come close to following the dashed line. This implies that *unlimited-h* is not working properly. Hence, I will be moving forward with my project using the '*limited-h'* code.

3.2 Limiting Smoothing Length

For the newer version of the code ("sph.passivelyAdvected" version, a.k.a. '*limited-h*' version) the smoothing length h_i of a particular particle i is given as

$$h_i = \frac{1}{\frac{1}{a} + b{\rho_i}^{1/3}} + c\,, \tag{3.1}$$

where the constant c is the smallest smoothing length a particle can have, ρ_i is the density at particle i, and $a + c$ is defined as the maximum value of h_i. In the *StarSmasher* code, c is called $hfloor$ and can be set in the input file ("sph.input"). Similarly, the quantity $a + c$ is called $hceiling$ and can also be set in the input file.

As the density at particle i becomes very large, the smoothing length will become very small. The smoothing length can be imagined like the half-radius of a bubble centered around particle i. We only want our bubble to contain a given number of particles (which is particle i's number of nearest neighbors). Imagine that a lot of particles are clumped close together. For the bubble to contain the correct number of particles, the radius will have to be quite small. With the old code, this situation could cause the smoothing length to approach zero. For the newer version of the code, under the same circumstances, the value of the smoothing length will approach c. This is illustrated by the following limit:

$$\begin{aligned}&\lim_{\rho_i\to\infty} \frac{1}{\frac{1}{a}+b\rho_i^{\frac{1}{3}}}+c\\&=\frac{1}{\infty}+c \quad =0+c\\&=c\end{aligned} \tag{3.2}$$

Hence, as the density approaches infinity, the value of h_i will approach c, which is the $hfloor$ value.

Now, we can take a look at the opposite problem. When the density of the gas at particle i becomes very low, the smoothing length becomes extremely large. Imagine the bubble trying to encapsulate the correct number of neighbors if the particles are very spread out. This would cause the bubble to grow unreasonably large. So, for the newer code, there is a value that the user can set as the maximum value of the smoothing length ($hceiling$). To see what happens to the value of the smoothing length when the density approaches zero, see the following limit:

$$\begin{aligned}&\lim_{\rho_i\to 0} \frac{1}{\frac{1}{a}+b\rho_i^{\frac{1}{3}}}+c\\&=\frac{1}{\frac{1}{a}+0}+c \quad =\frac{1}{\frac{1}{a}}+c\\&=a+c\end{aligned} \tag{3.3}$$

Recall that $a+c$ is defined as $hceiling$. Hence, as the density at particle i approaches 0, the smoothing length of particle i approaches the $hceiling$ value.

Chapter 4

Results

4.1 Comparing Models

Figures 4.1 and 4.2 are the energy profiles for each variation of the code: control (which has no accretion feedback mechanism), PK model, S model, and T model. The panels in each graph, from top to bottom, show the internal energy U of the system over time, the gravitational potential energy W of the system over time, the kinetic energy T of the system over time, and the total energy E of the system over time. The difference between Figures 4.1 and 4.2 is the value of the constant α_E (which is defined in Equation 2.14b). For Figure 4.1, $\alpha_E = 1$, which means that the data in the PK, S, and T graphs use the traditional Eddington limited mass accretion treatment. On the other hand, for Figure 4.2, $\alpha_E = 100$, so the PK, S, and T graphs have the super-Eddington mass accretion treatment as described in Section 2.3.

The total energy vs time in Figure 4.1 shows that all three models (PK, S, and T) are comparable to each other as well as with the control case in terms of energy conservation. Overall, there is a high level of agreement between the four cases. Although I was hoping that at least one of the three models would perform dramatically better than the control case, these results are not bad. While feedback energy may not be exceedingly important in this scenario with a $15\ M_\odot$ BH and a $20\ M_\odot$ star, the method I have implemented for calculating this additional energy does not seem to cause catastrophic failure in the code ☺. The three models can be tested and applied to other situations (for example, a more massive BH, or a different initial separation, etc.) where the models might reveal different behavior.

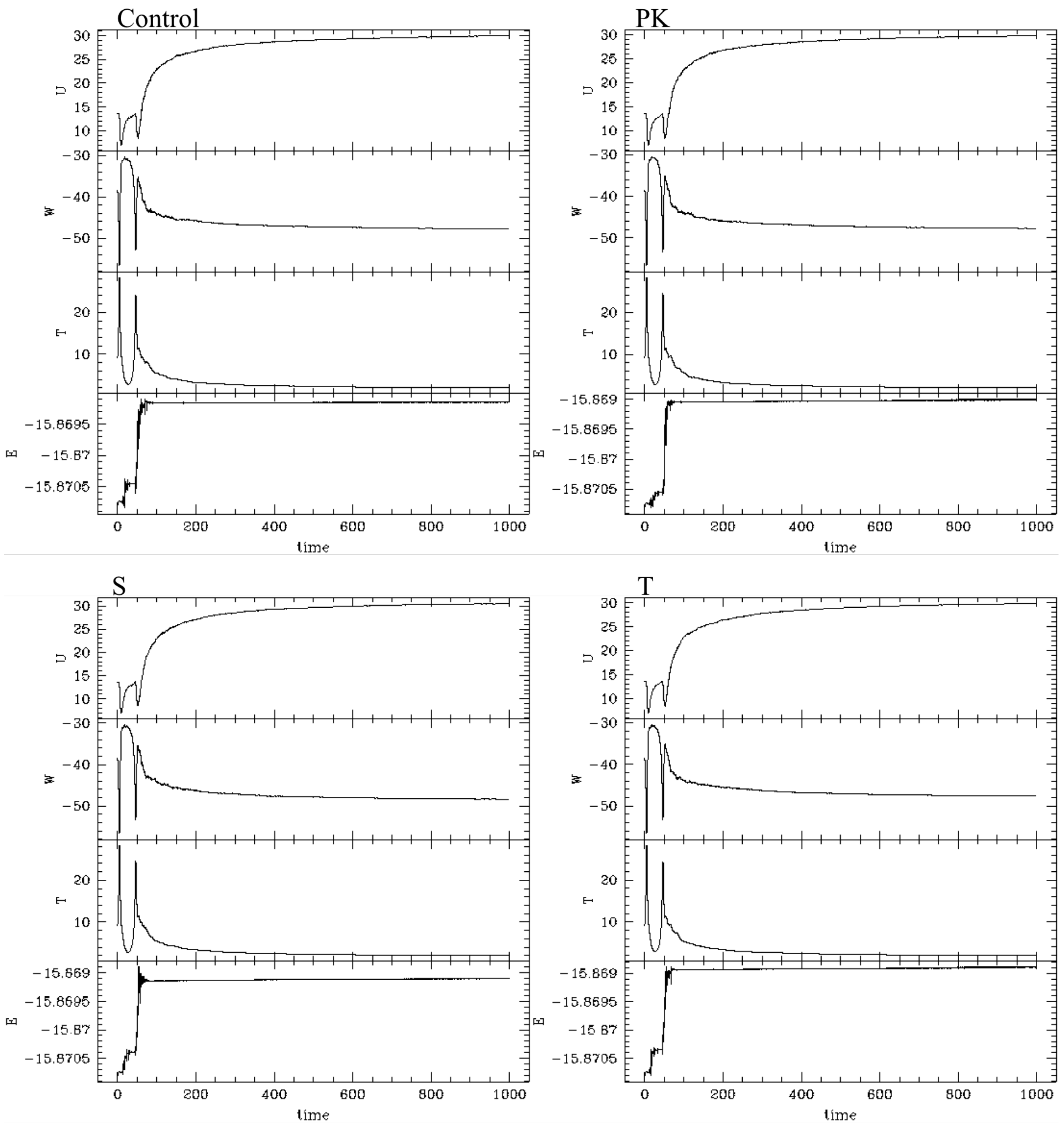

Figure 4.1: This figure shows energy vs time. The top left plot shows the energy profile of the control code (the code that does not take into account feedback energy). The top right, bottom left, and bottom right show the energy profiles for the PK, S, and T models respectively. The top panel of each plot shows the internal energy, second shows the gravitational potential energy, third shows the kinetic energy, and bottom shows the total energy. The units of time on the x-axis are t_{unit}, and the units of energy on the vertical axis are E_{unit} (see Eq. 2.1). For this figure, $\alpha_E = 1$.

In Figure 4.2a we can see that all three feedback models show a constant increase in total energy at later times. I suspect that this is because at later times, feedback energy may be less relevant; however, we are still adding energy to the system at every timestep. To test this, I have chosen two points from each dataset and found the slope between these two points. I have used the raw numbers from the log0.sph file to preserve accuracy. See Table 4.1 below.

	t_1	E_1	t_2	E_2	*Slope*	$\dot{E}_{\mathrm{fb}}$
PK Model	215.00004	-15.8682	799.00011	-15.8659	3.938×10^{-6}	3.961×10^{-6}
S Model	214.99998	-15.8693	777.00036	-15.8671	3.922×10^{-6}	3.961×10^{-6}
T Model	218.00024	-15.8693	799.00008	-15.8670	3.959×10^{-6}	3.961×10^{-6}

Table 4.1: This table shows points from the data files that produced each of the PK, S, and T graphs from Fig. 4.2a and shows the slope between those two points. It also shows the feedback energy rate at late times from the fort.100 file, the log of which is shown in Fig. 4.5.

Comparing the slope and $\dot{E}_{\mathrm{fb}}$ in the final two columns of Table 4.1, we can see that the data supports the idea that this energy drift is due to the feedback energy. Note that this energy drift in the total energy appears when we use the super-Eddington mass accretion rate with $\alpha_E = 100$ (Figure 4.2a) and does not when using the Eddington limited rate with $\alpha_E = 1$ (see Figure 4.1).

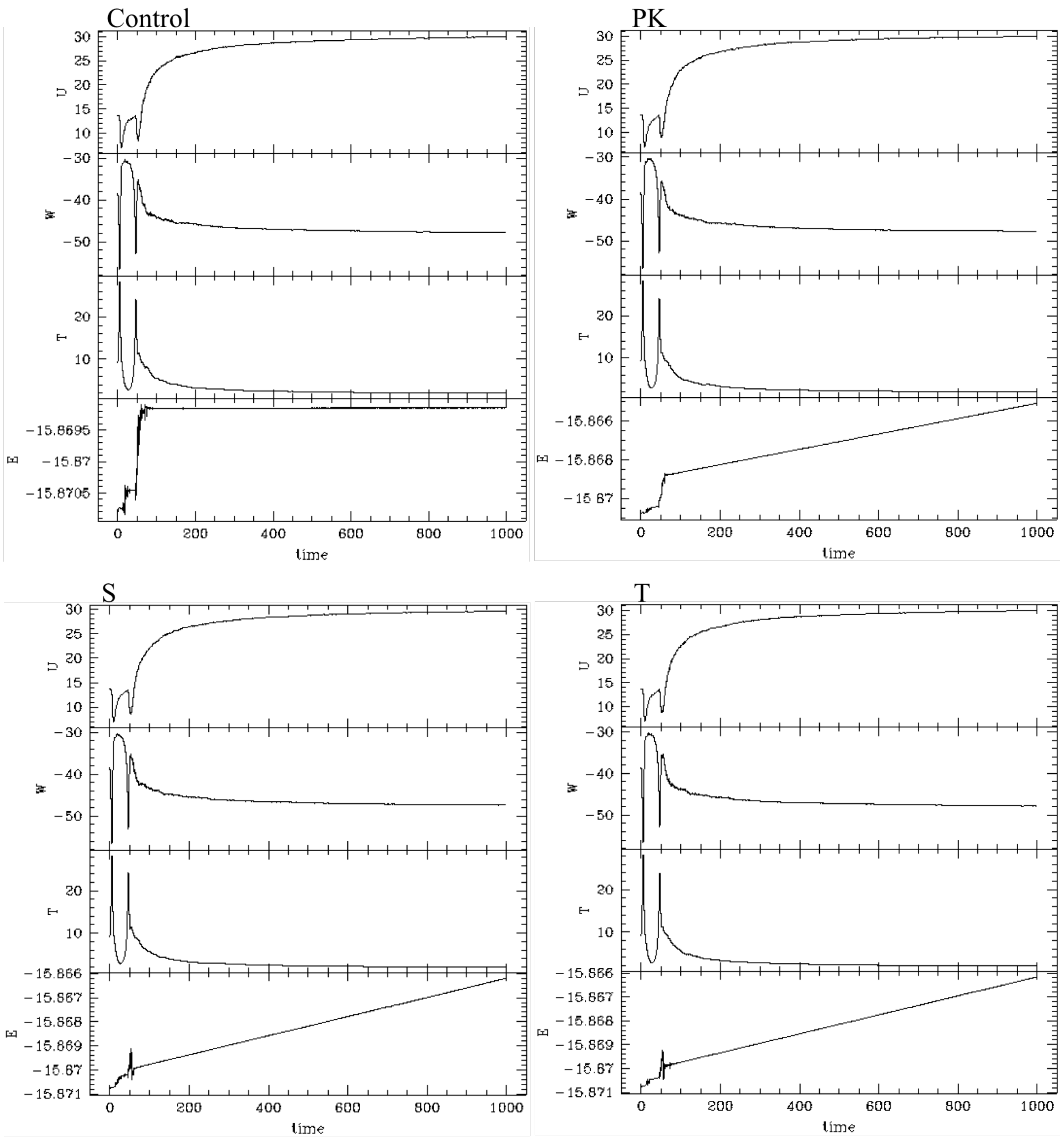

Figure 4.2a: This figure shows energy vs time. The top left plot shows the energy profile of the control code (the code that does not take into account feedback energy). The top right, bottom left, and bottom right plots show the energy profiles for the PK, S, and T models respectively. The top panel of each plot shows the internal energy, second shows the gravitational potential energy, third shows the kinetic energy, and bottom shows the total energy. The units of time on the x-axis are t_{unit}, and the units of energy on the vertical axis are E_{unit} (see Eq. 2.1). For this figure, $\alpha_E = 100$.

Interestingly, in Figure 4.2b, the total energy vs. time for the control and PK model are very similar, and total energy vs. time for the S and T models are very similar. This is most likely due to the way energy coupling is treated in each variation. In the S and T models, we have a fixed feedback radius. In the PK model, the energy coupled to a particle is dictated by which particles' kernels overlap with the black whole. In the control case, there is no energy coupling due to accretion feedback. It is not completely clear why the total energy is affected in this way by the energy coupling, and, hence, requires more research to confirm this as the reason for the similarity. It is also interesting to note that the similarity between the control and the PK models as well as the similarity between the S and the T models did not appear in the Eddington limited accretion version. It seems that the super-Eddington accretion version has revealed some properties of the models that were not able to be detected otherwise.

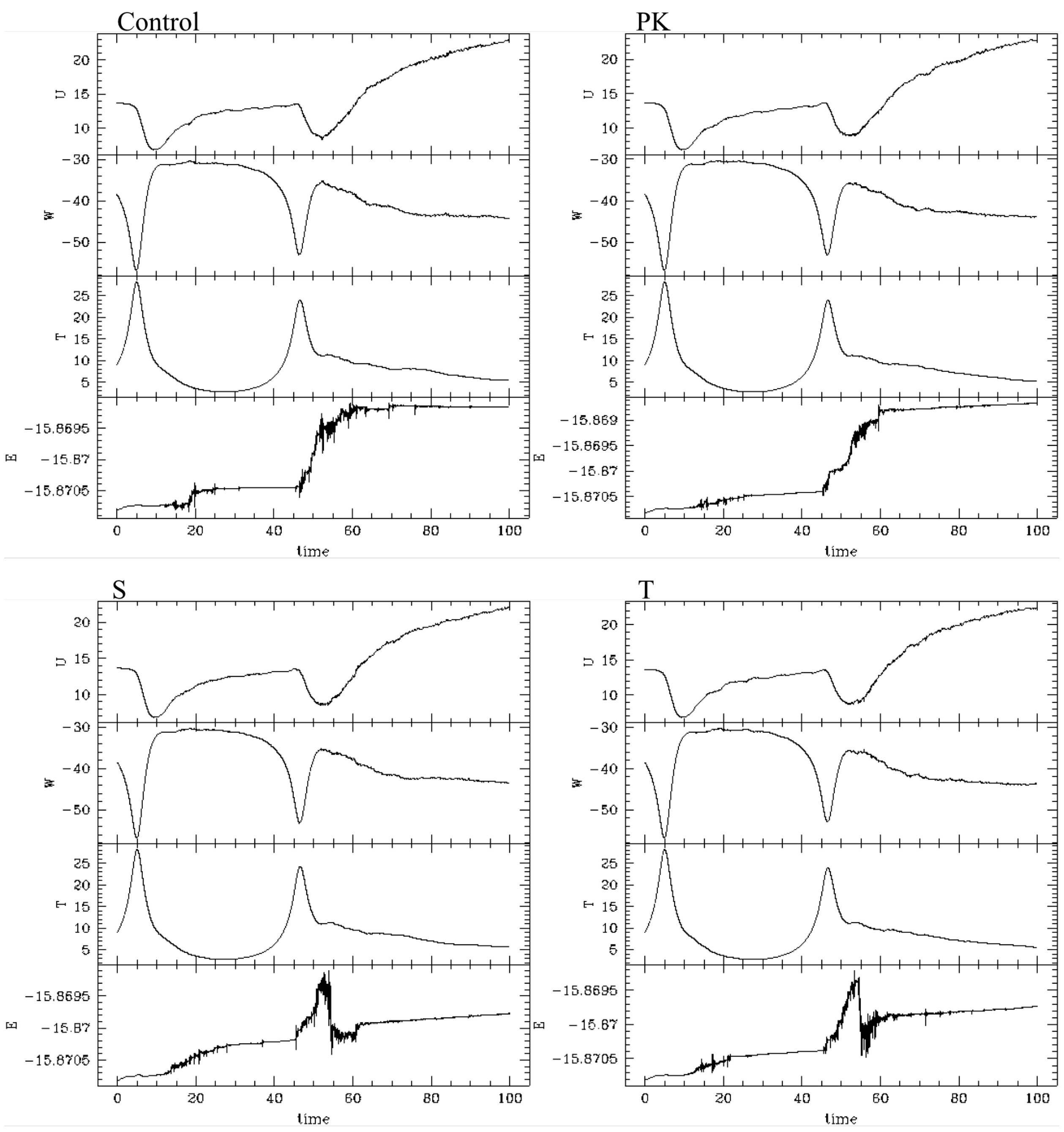

Figure 4.2b: This figure plots the energy vs. time, the same as Figure 4.2a, except the time interval for this is from time $t = 0$ to time $t = 100$. For this figure, $\alpha_E = 100$.

Figure 4.3 shows the number of particles affected by the respective model's feedback energy throughout the simulation from time $t = 0$ and $t = 1000$. From the graphs in Figure 4.3, we can see that the S and T models behave almost identically for the number of particles affected by feedback energy over time, while the PK model differs. The explanation for this pertains to the way that each model couples energy to particles surrounding the BH.

For both the S and T models, feedback energy is coupled with any particle that enters the feedback radius of the BH (recall that the feedback radius is user-defined, and in these test simulations, for both the S and T models, the feedback radius is set to 4 $R_\odot$). At early times, the particles in the star are still tightly packed (the BH has not yet ripped the star apart). So, when the star passes the BH, many particles enter the feedback radius of the BH and we can see a spike in the number of particles affected at these times (see times $t = 5$ and $t = 46$ in Figure 4.3 for S and T graphs). Furthermore, at late times, the star has been ripped apart; hence, the particles are spread out. Since the feedback radius is fixed in these two cases (S and T), there are less particles within the feedback radius of the BH, and, consequentially, the number of particles affected by feedback energy settles out to be fairly low (just shy of 200 particles, see S and T graphs in Figure 4.3).

In the case of the PK model, energy is coupled to any particle whose *kernel* overlaps with the BH. At time $t = 0$, there are no particle kernels that overlap with the BH; so, the number of particles affected by the feedback energy is zero, as expected (see PK graph in Figure 4.3). At early times the star is still tightly packed, so the kernels can be fairly small to encompass the correct number of nearest neighbors. However, at late times, once the star has been ripped apart, the particles are spread far apart, and the particle kernels become large in order to maintain the correct number of nearest neighbors. This can explain why the number of particles affected by the feedback radius continue to increase over time. During very late times in the simulation (for example, see $t = 850$ to $t = 1000$ for the PK graph in Figure 4.3), the particles are drifting apart less, so the particle kernels are not growing very much at this point; hence, the number of particles affected by the feedback energy will settle out to be approximately constant.

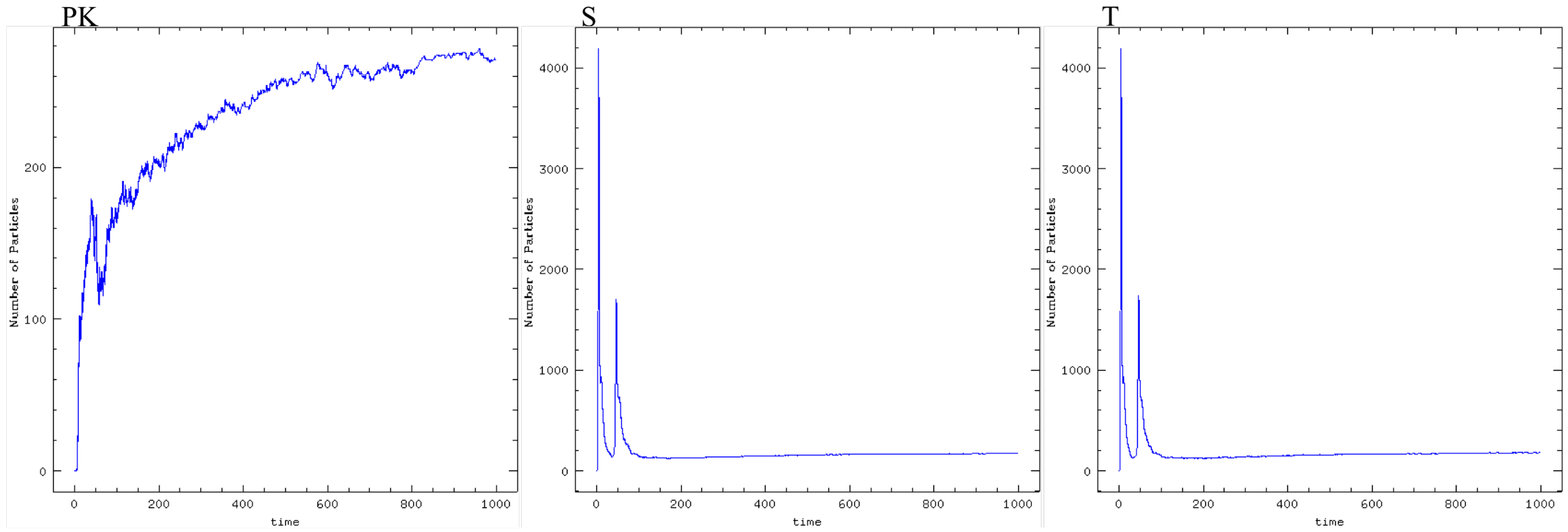

Figure 4.3: This figure shows the number of particles affected by feedback energy vs. time, where time is in t_{unit} (see Eq. 2.1). The top left, top right, and bottom left shows data from the PK, S, and T models respectively. Note that there is no plot for the control case since there is no feedback energy in the control case. For the S and T models we can see clear peaks in the number of particles affected during the times that the star passes the BH. The peak is less obvious on the PK graph, however there is a large increase in number of particles affected during the passages.

Figures 4.4 and 4.5 show the log of the mass accretion rate vs. time and the log of the feedback energy rate vs. time. In both Figure 4.4 and 4.5 the red data represents the Bondi-Hoyle mass accretion rate $\dot{M}_{\mathrm{B}}$, which is defined by Equation 2.14a, and the blue data represents the Eddington mass accretion rate $\dot{M}_{\mathrm{Edd}}$, which is defined by Equation 2.14b. Figure 4.4 shows the Eddington limited accretion rate data where $\alpha_E = 1$. Figure 4.5 shows the super-Eddington accretion rate with $\alpha_E = 100$. The code takes the minimum of $\dot{M}_{\mathrm{B}}$ and $\dot{M}_{\mathrm{Edd}}$ to be the actual mass accretion rate $\dot{M}_{\mathrm{BH}}$. As you can see, at late times, the mass accretion rate is dictated by $\dot{M}_{\mathrm{Edd}}$. Additionally, in both Figures, we see that $\dot{M}_{\mathrm{Edd}}$ is constant, as discussed in Section 2.2. For Figure 4.4 we see that the Eddington accretion rate (red data) is $\dot{M}_{\mathrm{Edd}} = 0.210 \times 10^{20}$ g/s, which agrees with Equation 2.13a. Furthermore, in Figure 4.5 we can see that the Eddington accretion rate (red data) is $\dot{M}_{\mathrm{Edd}} = 0.210 \times 10^{22}$ g/s which is also in agreement with Equation 2.13b. Also, note that, at late times, $\dot{E}_{\mathrm{fb}}$ in Figure 4.4 is two orders of magnitude less than it is in Figure 4.5. Looking back to Equation 2.16, we see that $\dot{E}_{\mathrm{fb}}$ is directly proportional to $\dot{M}_{\mathrm{BH}}$. At late times, $\dot{M}_{\mathrm{BH}} = \dot{M}_{\mathrm{Edd}}$, and $\dot{M}_{\mathrm{Edd}}$ in Figure 4.5 is 100 times greater than it is for Figure 4.4. So, it makes sense that $\dot{E}_{\mathrm{fb}}$ in Figure 4.4 is two orders of magnitude less than it is in Figure 4.5.

The most notable feature of these two figures (Figures 4.4 and 4.5) is the spike in the feedback energy rate at early times. This spike may be consistent with a burst of energy similar to a gamma ray burst. Additionally, when the star completely surrounds the blackhole there is a sustained period of feedback which could hint at a different electromagnetic signature.

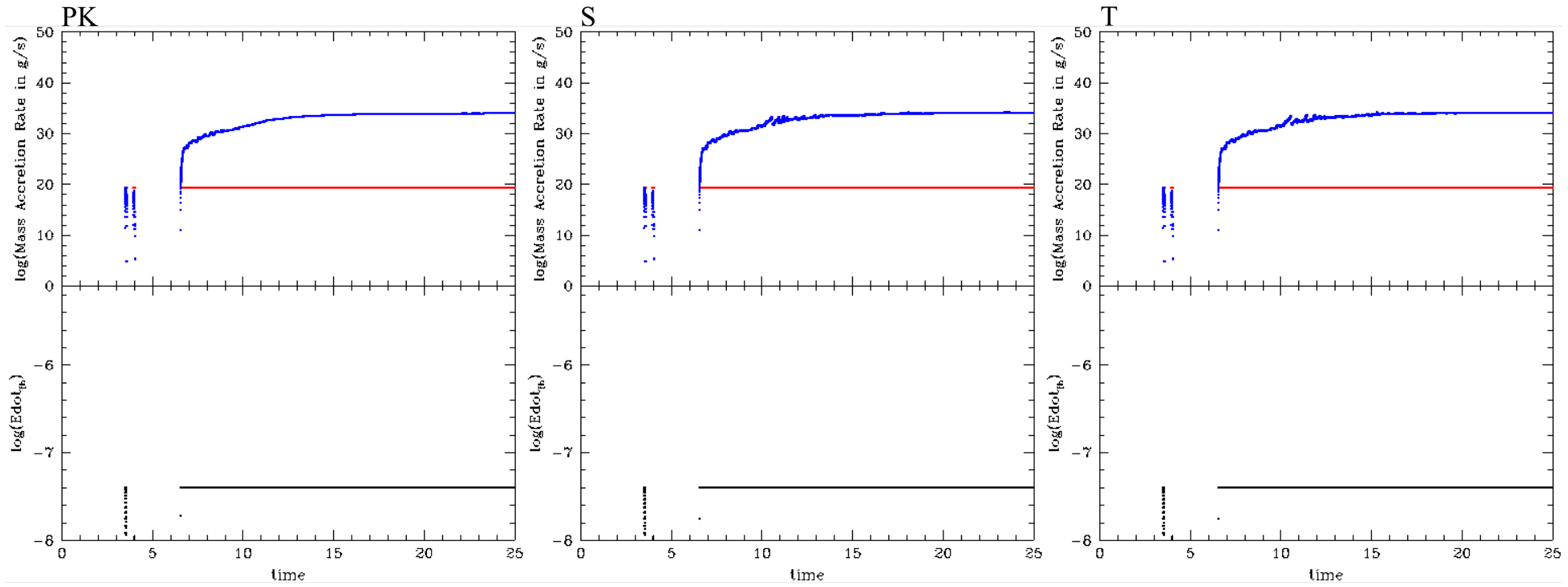

Figure 4.4: This figure shows the log of the mass accretion rate vs. time and the log of the feedback energy rate vs. time for the PK, S, and T models. The red data represents the Eddington mass accretion rate $\dot{M}_{\mathrm{Edd}}$ (defined by Eq. 2.14b). The blue data represents the Bondi-Hoyle mass accretion rate $\dot{M}_{\mathrm{B}}$ (defined by Eq. 2.14a). The code takes the actual mass accretion rate $\dot{M}_{\mathrm{BH}}$ to be the minimum of $\dot{M}_{\mathrm{B}}$ and $\dot{M}_{\mathrm{Edd}}$ (see Eq. 2.15). For this figure, $\alpha_E = 1$. On the lower panel, we have the log of $\dot{E}_{\mathrm{fb}}$vs time. $\dot{E}_{\mathrm{fb}}$ is in L_{unit} and time is in t_{unit} (see Eq. 2.1).

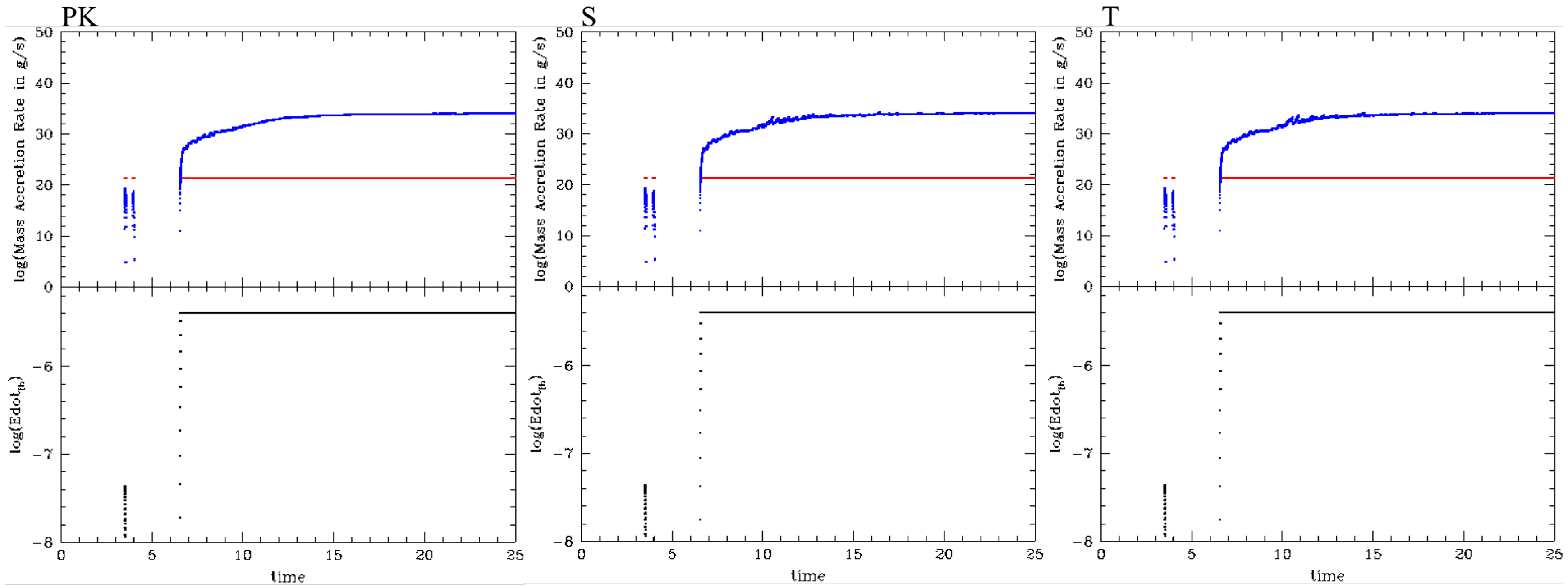

Figure 4.5: This figure shows the log of the mass accretion rate vs time and the log of the feedback energy rate vs time for the PK, S, and T models. The red data represents $\dot{M}_{\rm Edd}$ (defined by Eq. 2.14b). The blue data represents $\dot{M}_{\rm B}$ (defined by Eq. 2.14a). The mass accretion rate $\dot{M}_{\rm BH}$ is defined as the minimum of $\dot{M}_{\rm B}$ and $\dot{M}_{\rm Edd}$ (see Eq. 2.15). For this figure, $\alpha_E = 100$. On the lower panel, we have the log of $\dot{E}_{\rm fb}$vs time. $\dot{E}_{\rm fb}$ is in $L_{\rm unit}$ and time is in $t_{\rm unit}$ (see Eq. 2.1).

4.2 Conclusions

As I have stated earlier in this chapter, I have found evidence that the feedback energy may not be extremely important to energy conservation in my test scenario of a $15\ M_{\odot}$ BH and a $20\ M_{\odot}$ star. However, I have implemented a new technique for treating BH accretion and feedback that can be used for a myriad of other stellar/BH simulations. The three models can be tested and applied to other situations where the models could potentially reveal different behavior. I look forward to seeing how this code will be used by students in the future.

4.3 Future Work

Prior to this project, *StarSmasher* did not have an accretion feedback mechanism. Hence, I focused my project on implementation. So, there are still plenty of aspects pertaining to the behavior of the different models that need to be explored. The models that I have put forth in my project can easily be applied to other cases of BH to star simulations. To further investigate the differences between the behavior of the three models, one could vary the mass of the BH and run simulations for each feedback model to see if this affects how well each model performs. Future work could also include a more robust look into super-Eddington accretion and how to better account for this in the models. For students interested in analyzing my results, I feel that there needs to be more investigation into Figures 4.4 and 4.5 in particular. More specifically, one might look into why there is only one spike in the feedback energy rate instead of two.

References

Bennett J, Donahue M, Schneider N, Voit M. 2009. *The cosmic perspective,* 1301 Sansome St., San Francisco, CA 94111: Addison-Wesley. Fifth ed.

Brightman, M., Bachetti, M., Earnshaw, H. P., et al. 2019, Breaking the limit: Super-Eddington accretion onto black holes and neutron stars

Dagoneau N, Schanne S, Gros A, et al. 2018. *Detection capability of Ultra-Long Gamma-Ray Bursts with the ECLAIRs telescope aboard the SVOM mission*.

Elías M & Martínez O M. 2018. *Estimation of the star formation rate using long-gamma ray bursts observed by SWIFT*. Revista Mexicana de Astronomía y Astrofísica.

Gaburov E, Lombardi J C & Zwart S P. 2010. *On the onset of runaway stellar collisions in dense star clusters - II. Hydrodynamics of three-body interactions*. Monthly Notices of the Royal Astronomical Society. 402 : 105

Gehrels N. 2004. *The Swift γ-ray burst mission*. New Astronomy Reviews. 48 : 431

Liu M, Di Matteo T, Feng Y. 2016. *The effects of AGN feedback and SPH formulation on black hole growth in galaxies.* Monthly Notices of the Royal Astronomical Society. 458 : 1402–1416

Monaghan JJ. 1992. *Smoothed particle hydrodynamics.* Annual Review of Astronomy and Astrophysics. 30 : 543-74

Okamoto T, Nemmen RS, Bower RG. 2008. *The impact of radio feedback from active galactic nuclei in cosmological simulations: Formation of disc galaxies.* Monthly Notices of the Royal Astronomical Society. 385 :

Perets HB, Li Z, Lombardi JC, Milcarek SR. 2016. *Micro-tidal disruption events by stellar compact objects and the production of ultra-long GRBs.* The Astrophysical Journal. 823 : 113

Rasio FA, Shapiro SL. 1991. *Collisions of giant stars with compact objects - hydrodynamical calculations.* 559

Ryden, B. & Peterson, B. 2010, Foundations of Astrophysics (1301 Sansome St., San Francisco, CA 94111: Pearson Addison-Wesley)

Springel V. 2005. *The cosmological simulation code gadget-2.* Monthly Notices of the Royal Astronomical Society. 364 : 1105-34

Appendix

Relevant Files: balAV3.f & pressure.f

Here I show the files balAV3.f and pressure.f for each of the three models that I implemented (PK, S, and T), which are both files that existed in the source code for StarSmasher prior to my project. I have highlighted the parts of the files that I wrote. I include the full balAV3.f file for the PK model. The other two models' non-highlighted portions of balAV3.f are identical to the non-highlighted parts of balAV3.f for the PK model. Hence, I will only include the highlighted portions for the S and T models. Note that pressure.f is identical for all three models.

For future students looking to find my working directories on gonzales:
`/data/jalombar/starsmasher/sarah/analytic_eos`

The following are the names of the files that contain the simulations for my project:

`Control_model` (no accretion feedback energy mechanism)

$\alpha_{\mathrm{E}} = 100$	$\alpha_{\mathrm{E}} = 1$
`PK_model`	`PK_model_fixEdd`
`S_model`	`S_model_fixEdd`
`TophatModel`	`T_model_fixEdd`

pressure.f

```
      subroutine pressure
      include 'starsmasher.h'
      include 'mpif.h'
      real*8 pgas,prad
      real*8 rhocgs,ucgs,beta1,temperature,gam1,useeostable
      integer i
      integer mylength, ierr, irank

      if(neos.eq.0) then
         if(nintvar.eq.1) then
c     do i=1,ntot
            do i=n_lower,n_upper
c     p=a*rho^gam, so p/rho^2=a*rho^(gam-2.d0)
               por2(i)=u(i)*rho(i)**(gam-2.d0)
            enddo
         else
            do i=n_lower,n_upper
c     p=(gam-1)*rho*u, so p/rho^2=(gam-1)*u/rho
               if(u(i).ne.0.d0) por2(i)=(gam-1)*u(i)/rho(i)
            enddo
         endif
      else if(neos.eq.1) then
c         do i=1,ntot
         do i=n_lower,n_upper
            if(u(i).ne.0.d0) then
               rhocgs=rho(i)*munit/runit**3.d0
               if(nintvar.eq.1) then
                  ucgs=u(i)*rho(i)**(gam-1.d0)/(gam-1.d0)
     $                 *gravconst*munit/runit
               else
                  ucgs=u(i)*gravconst*munit/runit
               endif

               call gettemperature(qconst*rhocgs/meanmolecular(i),
     $              -ucgs*rhocgs/arad,temperature)
               pgas=rhocgs*boltz*temperature/meanmolecular(i)
               prad=arad*temperature**4/3.d0
               beta1=pgas/(pgas+prad)
               gam1=(32.d0-24.d0*beta1-3.d0*beta1**2) /
     $              (24.d0-21.d0*beta1)
               if(pgas .ne. pgas)then
                write(900+myrank,'(12g16.8)')i,ucgs,u(i),pgas,prad,
     $                 temperature,meanmolecular(i),qconst,rhocgs,arad
                  sToP
               endif

               if(gam1.lt.0.999*4.d0/3.d0 .or. gam1.gt.1.001*5.d0/3.d0) then
                  write(69,*)'warning gam1=',gam1,'at i=',i
                  write(69,*) beta1,pgas,prad,temperature,rho(i),
     $                 meanmolecular(i),rhocgs,ucgs,x(i),y(i),z(i),hp(i),
     $                 qconst*rhocgs/meanmolecular(i),
     $                 -ucgs*rhocgs/arad
                  stop 'gam1 value does not make sense'
               endif
               por2(i)=(pgas+prad)/rho(i)**2/punit

            else
               por2(i)=0.d0
            endif
         enddo
      else if(neos.eq.2) then
c     use tabulated eos here!

c     we need to use the tabulated
c     eos to get por2 (pressure over rho squared) from a table that uses specific
c     internal energy u and density rho as input variables.  my recommendation is to
c     make such a table for a gam=2 eos.  then make sure the code behaves the same
c     when neos=0 and when neos=2.

         do i=n_lower,n_upper
```

```
            if(u(i).ne.0.d0) then
               rhocgs=rho(i)*munit/runit**3.d0
               if(nintvar.eq.1) then
                  ucgs=u(i)*rho(i)**(gam-1.d0)/(gam-1.d0)
     $                 *gravconst*munit/runit
               else
                  ucgs=u(i)*gravconst*munit/runit
               endif
               por2(i)=useeostable(ucgs,rhocgs,meanmolecular(i),3)
     $              /rho(i)**2/punit
            else
               por2(i)=0.d0
            endif
         enddo

      endif

c     this part will share the POR2 value between processors
      mylength=n_upper-n_lower+1
      do irank=0,nprocs-1
         if(myrank.ne.irank)then
           call mpi_gatherv(por2(n_lower),mylength,mpi_double_precision,
     $           por2, recvcounts, displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
           call mpi_gatherv(rho(n_lower),mylength,mpi_double_precision,
     $           rho, recvcounts, displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)

         else
           call mpi_gatherv(mpi_in_place,mylength, mpi_double_precision,
     $           por2, recvcounts, displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
           call mpi_gatherv(mpi_in_place,mylength, mpi_double_precision,
     $           rho, recvcounts, displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         endif
      enddo


      return
      end
```

balAV3.f   ~PK model

```
      subroutine uvdots
c     evaluate right-hand sides of equations of motion:
c=======================================================================
c  balvdots creates arrays of the accelerations of each particle, using
c  balsara's form of the artificial viscosity pi_{ij}.
c  this incarnation of balvdots implements the form in which only the pi_{ij}
c     term is multiplied by the gradient of a symmetrized w_{ij}.
c     the p/rho^2 terms are each multiplied by the gradient of the kernel
c     evaluated with the smoothing length of that particle only.
c     see springel and hernquist's mnras article (astro-ph/0111016)
c     and monaghan's compressible turbulence article (astro-ph/0204118).
c=======================================================================
      include 'starsmasher.h'
      include 'mpif.h'
      double precision curlvxi,curlvyi,curlvzi
      real*8 csij,udbij,fi,ci,r2,
     $     abscurli,divvi,vxi,vyi,vzi,ci2,ami,por2i,h5,
     $     h2,hpi,csiju,csijgc
      integer itab,j,in,i,ierr
      real*8 uijmax(nmax)
      common/uijmax/ uijmax
      real*8 dwijin,pijin,sxijin,syijin,szijin,divin, diwkin
      real*8 sxijingc,syijingc,szijingc
      logical switchedav
      double precision divv(nmax)
      common/commdivv/divv
      integer mylength, mygravlength
      integer comm_worker
      common/gravworkers/comm_worker
      real*8 myvxdot(ntot),myvydot(ntot),myvzdot(ntot)
      real*8 myvxdotgc(ntot),myvydotgc(ntot),myvzdotgc(ntot)
      integer irank
      real*8 invh, invh2, invh5
      real*8 dx, dy, dz
      real*8 dvx, dvy, dvz
      real*8 invbonnet
      real*8 alpha1, beta1
      integer offset, nni
c     variables used for radiative cooling portion of the code:
      real*8 uunit,udotunit,rhocgs,ucgs,temperature,kappa
      real*8 opacity,pgas,prad
      real*8 gradponrho,scaleheight,columnden
      real*8 kappar
      real*8 ueqfunction,useeostable,uupperlimit!,ulowerlimit
      external ueqfunction
      real*8 zeroin,mucgs
      common/ueqstuff/rhocgs,teq,mucgs
      real*8 grpottot(nmax),uraddoti
!      real*8 hpitilde,h2tilde,invh2tilde
!      integer itabtilde
c     variables used for Accretion/Feedback Model
      real*8 Rfb, Rfb2, Mencl_Rfb, xrelBH, yrelBH, zrelBH
      real*8 dist_to_BH2, vx_BHrelgas, vy_BHrelgas, vz_BHrelgas
      real*8 rhoBH, v_BHrelgas2, pBH, cs_BH2, gamma
      real*8 Mdot_BH,Mdot_B, Mdot_Edd, Edot_fb
      real*8 m_proton,epsilon_r,epsilon_f,alph,alphE,sigmaT,POR2BH
      real*8 udot_PKmodel, vunit, invhpj2,invhpj3,invh3
      real*8 rhoBH_cgs,Edot_fb_cgs
      integer itabBH, icount, myicount
c     set Rfb and gamma and other relavent constants here
      Rfb=4.0d0
      Rfb2=Rfb*Rfb
      gamma=5d0/3d0
      m_proton= 1.6726219d-24 !in grams
      epsilon_r=0.1d0
      epsilon_f=0.05d0
      sigmaT= 6.6524d-25      !in cm**2
      alph=100d0
      alphE=1d0
      myicount=0
      vunit=(gravconst*munit/runit)**0.5d0 !used to convert velocities to cgs units

      if(nav.eq.0) then
         if(nintvar.eq.1)then
            do i=1,n
               udot(i)=0.d0
            enddo
         else
            print *,'turn off av by setting alpha=beta=0'
            stop
         endif
      endif

      switchedav=.false.
      if(nrelax.eq.1
     $     .and. t.lt.treloff .and. alpha.eq.0.d0
     $     .and. beta.eq.0.d0) then
         switchedav=.true.
         alpha=1.d0
         beta=2.d0
         if(myrank.eq.0)
     $        write(69,*)'vdots: during relaxation alpha,beta=',alpha,beta
      endif

c  initialize dv/dt before accumulating:
      do i=1,n
         myvxdot(i)=0.d0
```

```
            myvydot(i)=0.d0
            myvzdot(i)=0.d0
            myVXDOTgc(I)=0.d0
            myVYDOTgc(I)=0.d0
            myVZDOTgc(I)=0.d0
            uijmax(i)=0.d0
            udot(i)=0.d0
         end do

         if (nrelax.eq.1 .and.t.lt.treloff) then
            if(myrank.eq.0)write(69,*)
     $        'baludots is not using pi_{ij} right now'
         endif

c     the factor of 2d0 in the following lines is because our pi_{ij} below
c     is going to involve only p_i/rho_i^2 and not, as is often done,
c     (p_i/rho_i^2 + p_j/rho_j^2).  to keep alpha and beta values at the same
c     strength as in other implementations of sph, we multiply by 2 here.
      alpha1 = -2.d0 * alpha
      beta1  =  2.d0 * beta

cccccccccccccccccccccccccccccccc
c        Sarah's Edits         c
cccccccccccccccccccccccccccccccc
c     Initialize relative velocities to zero (T model)
      vx_BHrelgas=0.d0
      vy_BHrelgas=0.d0
      vz_BHrelgas=0.d0
      rhoBH=0.d0 !initialize smoothed density to zero
      POR2BH=0.d0   !initialize smoothed pressure to zero
      cs_BH2=0.d0

c share hp values with ranks
      mylength=n_upper-n_lower+1
      do irank=0,nprocs-1
         if(myrank.ne.irank)then
            call mpi_gatherv(hp(n_lower), mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         else
            call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         endif
      enddo


      do j=1, (ntot-1)
c        ****************************
c        * Calculate distance to BH *
c        ****************************
c       find particle's position relative to BH
         xrelBH=x(j)-x(ntot) !note: x(ntot) is the x position of the BH
         yrelBH=y(j)-y(ntot)
         zrelBH=z(j)-z(ntot)
         dist_to_BH2= xrelBH**2 + yrelBH**2 + zrelBH**2

c        ***************************************
c        * Calculate rel velocity of BH to gas *
c        ***************************************
         invhpj2=1/(hp(j)**2) !this is the inverse of the smoothing length squared (1/h^2)
         invhpj3=invhpj2/hp(j)
         if(dist_to_BH2 .lt. 4*hp(j)**2) then
            itab=int(ctab*dist_to_BH2*invhpj2)+1   !don't freak out; ctab is a known value
            vx_BHrelgas= vx_BHrelgas+am(j)*wtab(itab)*vx(j)*invhpj3
            vy_BHrelgas= vy_BHrelgas+am(j)*wtab(itab)*vy(j)*invhpj3
            vz_BHrelgas= vz_BHrelgas+am(j)*wtab(itab)*vz(j)*invhpj3
            rhoBH= rhoBH+am(j)*wtab(itab)*invhpj3  !calculate smoothed density at BH
            if (POR2(j) .le. 0) then  !checks to make sure that all particles have POR2 values
               write(10+myrank,*) "POR2 undefined for particle ", j
               stop
            endif
c            POR2BH= POR2BH+am(j)*wtab(itab)*POR2(j)*invhpj3 !calculate smoothed pressure at BH
```

```
               cs_BH2= cs_BH2+am(j)*wtab(itab)*POR2(j)*invhpj3*rho(j)
            endif
         enddo


         if (rhoBH .gt. 0d0)then
            vx_BHrelgas=vx_BHrelgas/rhoBH !when finding smoothed value at BH, you must divide by rhoBH after loop
            vy_BHrelgas=vy_BHrelgas/rhoBH
            vz_BHrelgas=vz_BHrelgas/rhoBH
c            POR2BH=POR2BH/rhoBH
            cs_BH2=cs_BH2/rhoBH

            vx_BHrelgas= (vx_BHrelgas -vx(ntot))
            vy_BHrelgas= (vy_BHrelgas -vy(ntot))
            vz_BHrelgas= (vz_BHrelgas -vz(ntot))

            v_BHrelgas2= vx_BHrelgas**2 + vy_BHrelgas**2 + vz_BHrelgas**2 !use components to find v^2 of BH relative
to gas
            v_BHrelgas2=v_BHrelgas2*vunit**2  !convert to cgs units

c            pBH= POR2BH*rhoBH**2   !when finding smoothed value at BH, you must divide by rhoBH after loop
            rhoBH_cgs= rhoBH*munit/runit**3 !convert to cgs units
c            pBH=pBH*gravconst*munit**2/runit**4 !convert to cgs units

c            cs_BH2=gamma*pBH/rhoBH_cgs !calculate local sound speed (in cgs units)
            cs_BH2=cs_BH2*gamma*vunit**2 !calculate local sound speed (in cgs units)

c           *************************************************
c           * Calculate mass accretion rate of BH (Mdot_BH) *
c           *************************************************

c        calculate Mdot_B, the Bondi-Hoyle mass accretion rate (cgs)
            Mdot_B=4*pi*alph*(gravconst**2)*(am(ntot)**2*munit**2)*rhoBH_cgs/
     $        ((cs_BH2+v_BHrelgas2)**1.5d0)

c        calculate Mdot_Edd, the Eddington-limitted mass accretion rate (cgs)
            Mdot_Edd=4*pi*alphE*gravconst*(am(ntot)*munit)*m_proton/
     $        (epsilon_r*sigmaT*crad)

c        for the acctual accretionrate of the BH, Mdot_BH, take minimum of prev two (cgs)
            Mdot_BH= min(Mdot_Edd,Mdot_B)

         else
            Mdot_Edd=0d0    !)
            Mdot_B=0d0      !> for when we print out these values, they need a value
            cs_BH2=0d0      !)

            Mdot_BH=0d0
         endif
c           *******************************************
c           * Calculate rate of BH's Feedback Energy  *
c           *******************************************

         Edot_fb_cgs = epsilon_r*epsilon_f*Mdot_BH*crad2
         Edot_fb= Edot_fb_cgs*runit**2.5d0/(gravconst**1.5d0*munit**2.5d0) !convert Edot_fb term to code units

c        Search for "PK Model Term" to see the udot compontent corresponding to PK Model equation

cccccccccccccccccccccccccccccccc
c        End of Sarah's edits     c
cccccccccccccccccccccccccccccccc

c        for each particle:
         do i=n_lower,n_upper ! line 244 of balAV3.f in src_from_2006_paper
            hpi=hp(i)
!            hpitilde=hpi - hfloor
            h2=hpi**2
!            h2tilde=hpitilde**2
            h5=hpi*h2**2
            invh  = 1.0/hpi
            invh2 = 1.0/h2
            invh3 = invh*invh2
!            invh2tilde = 1.0/h2tilde
```

```
         invh5 = 1.0/h5
         ami=am(i)
         por2i=por2(i)
         ci2=gam*por2i*rho(i)
         ci=sqrt(ci2)
         vxi=vx(i)
         vyi=vy(i)
         vzi=vz(i)
         call getderivs(i,curlvxi,curlvyi,curlvzi,divvi)
         divv(i)=divvi
         abscurli=sqrt(curlvxi**2+curlvyi**2+curlvzi**2)

c     calculate gradwij's to all neighbors:
         offset = first(i)
         nni    = nn(i)
         do in=1,nni   ! line 259 in balAV3.f
            j=list(offset+in)

            dx = x(i) - x(j)
            dy = y(i) - y(j)
            dz = z(i) - z(j)
            dvx = vxi - vx(j)
            dvy = vyi - vy(j)
            dvz = vzi - vz(j)
            r2 = dx*dx + dy*dy + dz*dz
            itab=int(ctab*r2*invh2)+1

            if(r2.lt.4d0*h2) then
!            if(r2.lt.4d0*h2tilde) then
!               itabtilde=int(ctab*r2*invh2tilde)+1
!               diwkin = dgtab(itabtilde) * invh2tilde
               diwkin = dgtab(itab) * invh2
            else
               write(69,*) 'Should never get here....'
               sToP
               diwkin = 0d0
            endif

            dwijin = dwtab(itab) * invh5  ! line 263

            if(u(j).ne.0.d0 .and. u(i).ne.0) then
              divin = dx*dvx + dy*dvy + dz*dvz

               if (divin.lt.0.d0) then
c                  r2=(x(i)-x(j))**2.d0+(y(i)-y(j))**2.d0
c     $                 +(z(i)-z(j))**2.d0
c     cj=sqrt(5.d0/3.d0*por2(j)*rho(j))

c     calculate dinshaw balsara's (db) \mu_{ij}: see equation (a12) of
http://adsabs.harvard.edu/abs/2012apj...745...71p
                  fi=abs(divvi)/(abs(divvi)+abscurli+
     $                 0.00001d0*ci*invh)
c                 fj=abs(divv(j))/(abs(divv(j))+
c     $                    sqrt(curlvx(j)**2+curlvy(j)**2+curlvz(j)**2)+
c     $                    0.00001d0*cj/hp(j))
c     modified from very old version:
                  if(r2.gt.0) then
                     if(nav.eq.3) then
                        udbij=divin/(ci*r2**0.5d0)*fi
                     else
                        udbij=hpi*divin/(ci*(r2+0.01d0*hpi**2))*fi
                     endif
                  else
                     udbij=0d0
                  endif

c     old version:
c     udbij=div(in)/((ci+cj)*r2**0.5d0)*(fi+fj)
c     udbij=divin/(ci*r2**0.5d0)*fi
                  if(abs(udbij).gt.1.d8) then
                     write(69,*)'c warning large udbij...',i,j, udbij,divin,ci,
     $                    am(i),am(j),u(i),u(j)
                     if(dt.lt.1d-17)then
```

```
                        write(69,*)'unreasonably small dt. abort run.'
                        call mpi_finalize(ierr)
                        stop
                     endif
                  endif
c                              a     +    b * c   (multiply-add, cpu likes these)
                  pijin=f(i)*por2i*(alpha1 + beta1*udbij)*udbij
               else
                  pijin=0.d0
               endif
!c     hydro correction part.  here we are implementing corrections as shown
!c     in equations (a11) and (a12) of gaburov et al. (2010):
!               invbonnet = 1.0/(bonet_0mega(i)*am(j))

c     csij will not contain AV (pijin) contributions so that we can
c     later use the acceleration derived from it to calculate scale height.
c     All AV contributions to the acceleration will be accounted for with
c     the help of csijgc below. (gc=gravity correction... this is for terms
c     associated with the gravity correction *and* with AV)
!               csij = por2i * (-bonet_omega(i) * invbonnet * diwkin
!     $              + dwijin)
!               csijgc= 0.5d0 * pijin * dwijin

               csij=f(i)*por2i*dwijin
               csijgc=0.5d0*pijin*dwijin

c     hydro part
               if (nrelax.eq.1 .and.t.lt.treloff) then
                  csiju= csij ! No heating from AV during relaxation
               else
                  csiju= csij + csijgc
               endif


               if(nintvar.eq.2) then
                  udot(i)=udot(i)+am(j)*csiju*divin ! line 197
               elseif (nrelax.eq.0 .or. t.ge.treloff) then
                  udot(i)=udot(i)+am(j)*pijin*dwijin*divin ! line 180
               endif

               uijmax(i)=max(uijmax(i),
     $              (f(i)*por2i + 0.5d0 * pijin)*rho(i))
c               uijmax(j)=max(uijmax(j),
c     $              (por2i + 0.5d0 * pijin)*rho(i))

               if(ngr.ne.0) then
ccc     gravity correction part
ccc                 csij = csij - 0.5d0*diwkin * bonet_psi(i) * invbonnet
cc                 csijgc = csijgc - 0.5d0*diwkin * bonet_Psi(i) * invBonnet


c                  if(u(i).ne.0) csijgc = csijgc + 0.5d0*f(i)*zeta(i)*dwijin
                  csijgc = csijgc + 0.5d0*f(i)*zeta(i)*dwijin


               endif
            else
               csij=0.d0
               csijgc=0.d0
            endif

            sxijin = csij * dx !(x(i) - x(j))
            syijin = csij * dy !(y(i) - y(j))
            szijin = csij * dz !(z(i) - z(j))
            sxijingc = csijgc * dx !(x(i) - x(j))
            syijingc = csijgc * dy !(y(i) - y(j))
            szijingc = csijgc * dz !(z(i) - z(j))

c     finally compute each component of dv/dt:

c     note: sxijin has been calculated using pi/rhoi^2.  so in the "gather"
c     part of the sum, everything is as you would expect.  but the gather part
c     of the sum only gives half of the contributions for any given particle
```

```
c     i.  for the sake of concreteness, let's say i=5.  the other half of the
c     contributions to the acceleration of particle 5 comes from the scatter
c     portion of the code with j=5 (note: this is *j*, not i).  in the scatter
c     portion, the index i will be a number other than 5, and so the p/rho^2
c     being used inside of sxijin will come from that particle with index
c     different than 5.  the gather contributions to particle 5 all happen
c     together in one big group.  the scatter contributions to particle 5 will
c     happen haphazardly as the loop over particle index i gradually covers
c     all the particles that have 5 as a neighbor.

c     "gather" part of the sum:
            myvxdot(i)=myvxdot(i)-am(j)*sxijin
            myvydot(i)=myvydot(i)-am(j)*syijin
            myvzdot(i)=myvzdot(i)-am(j)*szijin
c     "scatter" part of the sum:
            myvxdot(j)=myvxdot(j)+ami*sxijin
            myvydot(j)=myvydot(j)+ami*syijin
            myvzdot(j)=myvzdot(j)+ami*szijin
c     "Gather" part of the sum:
            myvxdotgc(i)=myvxdotgc(i)-am(j)*sxijingc
            myvydotgc(i)=myvydotgc(i)-am(j)*syijingc
            myvzdotgc(i)=myvzdotgc(i)-am(j)*szijingc
c     "Scatter" part of the sum:
            myvxdotgc(j)=myvxdotgc(j)+ami*sxijingc
            myvydotgc(j)=myvydotgc(j)+ami*syijingc
            myvzdotgc(j)=myvzdotgc(j)+ami*szijingc
         enddo

c     p=(gam-1)*rho*u=a*rho^gam, so a=(gam-1)*rho^(1-gam)*u
         if(nintvar.eq.1 .and. u(i).ne.0.d0)
     $        udot(i)=udot(i)*0.5d0*(gam-1.d0)*rho(i)**(1.d0-gam) ! line 82 of balAV3.f of 2007 code

         if(nintvar.eq.2) then
c          ************************************
c          * Calculate PK Model Term for udot *
c          ************************************
            if(i .ne. ntot)then
               if(rhoBH .gt. 0d0)then
                  dist_to_BH2=(x(i)-x(ntot))**2+(y(i)-y(ntot))**2+ !calculate particle's dist to BH
     $                 (z(i)-z(ntot))**2
                  if(dist_to_BH2 .lt. 4*h2)then
                     myicount=myicount+1
                     itabBH=int(ctab*dist_to_BH2*invh2)+1
                     udot_PKmodel=(Edot_fb*(wtab(itabBH)*invh3)/rhoBH) !***** PK Model Term ***** (code units)
                     udot(i)=udot(i)+udot_PKmodel
                  endif
               endif
            endif
c          ************************************
         endif

      enddo
c      write(6,'(a)')'hydrompi'

      mylength=n_upper-n_lower+1
      do irank=0,nprocs-1
         if(myrank.ne.irank)then
            call mpi_gatherv(udot(n_lower), mylength, mpi_double_precision,
     $           udot, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         else
            call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $           udot, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         endif
      enddo
c      call mpi_allgatherv(mpi_in_place, mylength, mpi_double_precision,
c     $     udot, recvcounts,displs, mpi_double_precision,
c     $     mpi_comm_world, ierr)

      if(ncooling.ne.0) then
c     to get the pressure scale height, we will need the v{x,y,z}dot arrays *without* the
c     gravitational acceleration included.  note: the myv{x,y,z}dot arrays are not
```

```
c     affected by the following mpi calls.  instead, they are just being summed up to give
c     the v{x,y,z}dot arrays.
         call mpi_allreduce(myvxdot,vxdot,n,mpi_double_precision,mpi_sum,
     $        mpi_comm_world,ierr)
         call mpi_allreduce(myvydot,vydot,n,mpi_double_precision,mpi_sum,
     $        mpi_comm_world,ierr)
         call mpi_allreduce(myvzdot,vzdot,n,mpi_double_precision,mpi_sum,
     $        mpi_comm_world,ierr)

         punit=gravconst*(munit/runit**2)**2
         uunit=gravconst*munit/runit
         udotunit=(gravconst*munit)**1.5d0/runit**2.5d0
c     if(myrank.eq.0) write(69,*) 'uunit,udotunit=',uunit,udotunit

         if(neos.eq.1)then
            do i=n_lower,n_upper
               if(u(i).ne.0.d0) then
                  rhocgs=rho(i)*munit/runit**3.d0
                  if(nintvar.eq.1) then
                     ucgs=u(i)*rho(i)**(gam-1.d0)/(gam-1.d0)
     $                    *uunit
                  else
                     ucgs=u(i)*uunit
                  endif
                  call gettemperature(qconst*rhocgs/meanmolecular(i),
     $                 -ucgs*rhocgs/arad,temperature)
                  pgas=rhocgs*boltz*temperature/meanmolecular(i)/punit
                  prad=arad*temperature**4/3.d0/punit

                  gradponrho=(vxdot(i)**2+vydot(i)**2+vzdot(i)**2)**0.5d0
                  scaleheight=(pgas+prad)/(gradponrho*rho(i))
                  columnden=rho(i)*scaleheight*1.06d0*munit/runit**2
c                 kappa is local opacity; kappar is pseudo-mean opacity
                  call usetable(rhocgs,temperature,kappa,kappar)
                  uraddoti=4*sigma*(teq**4-temperature**4)/
     $                 (columnden**2*kappar + 1/kappa)
     $                 /udotunit
                  ueq(i)=(arad*teq**4/rhocgs
     $                 +1.5d0*boltz*teq/meanmolecular(i))
     $                 /uunit

                  tthermal(i)=(ueq(i)-u(i))/uraddoti

c     approximate the equilibrium temperature... not really correct...
c     teq=(10.d0**4+udot(i)*(columnden**2*kappa+ 1/kappa)
c     $           *udotunit/(4*sigma))**0.25d0

               else
                  tthermal(i)=1.d30
                  ueq(i)=0.d0
               endif
            enddo
         else if(neos.eq.2)then
            do i=n_lower,n_upper
               if(u(i).ne.0.d0) then
                  rhocgs=rho(i)*munit/runit**3.d0
                  mucgs=meanmolecular(i)
                  if(nintvar.eq.1) then
                     ucgs=u(i)*rho(i)**(gam-1.d0)/(gam-1.d0)
     $                    *uunit
                  else
                     ucgs=u(i)*uunit
                  endif
                  temperature = useeostable(ucgs,rhocgs,
     $                 meanmolecular(i),1)
                  gradponrho=(vxdot(i)**2+vydot(i)**2+vzdot(i)**2)**0.5d0
                  scaleheight=useeostable(ucgs,rhocgs,
     $                 meanmolecular(i),3)/(punit*gradponrho*rho(i))
                  columnden=rho(i)*scaleheight*1.06d0*munit/runit**2
c                 kappa is local opacity; kappar is pseudo-mean opacity
                  call usetable(rhocgs,temperature,kappa,kappar)
                  uraddoti=4*sigma*(teq**4-temperature**4)/
     $                 (columnden**2*kappar + 1/kappa)
     $                 /udotunit

c     using eos table, solve for u_eq such that temperature t=teq
                  uupperlimit=1.1d0*(1.5d0*boltz*teq/meanmolecular(i)
     $                 +arad*teq**4/rhocgs)
 1234             continue
                  ucgs=zeroin(0d0,uupperlimit,ueqfunction,1.d-11)
                  if(ueqfunction(ucgs).gt.1d-6) then
                     uupperlimit=2*uupperlimit
                     write(69,*)'Having trouble with u_eq for particle'
     $                    ,i,'at time',t,ueqfunction(ucgs)
                     ucgs=zeroin(0d0,uupperlimit,ueqfunction,1.d-11)
                     write(69,*)'Found',ucgs,'for'
     $                    ,i,'at time',t,ueqfunction(ucgs)
                  endif
                  ueq(i)=ucgs/uunit

                  tthermal(i)=(ueq(i)-u(i))/uraddoti

               else
                  tthermal(i)=1.d30
                  ueq(i)=0
               endif
            enddo
         else
            print *,'radiative cooling requires neos=1 or 2'
            stop
         endif

         mylength=n_upper-n_lower+1
         do irank=0,nprocs-1
            if(myrank.ne.irank)then
               call mpi_gatherv(tthermal(n_lower), mylength, mpi_double_precision,
     $              tthermal, recvcounts,displs, mpi_double_precision, irank,
     $              mpi_comm_world, ierr)
               call mpi_gatherv(ueq(n_lower), mylength, mpi_double_precision,
     $              ueq, recvcounts,displs, mpi_double_precision, irank,
     $              mpi_comm_world, ierr)
            else
               call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $              tthermal, recvcounts,displs, mpi_double_precision, irank,
     $              mpi_comm_world, ierr)
               call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $              ueq, recvcounts,displs, mpi_double_precision, irank,
     $              mpi_comm_world, ierr)
            endif
         enddo
      endif

c      write(6,'(a)')'hydrompi_complete'
      if(ngr.ne.0 .and. myrank.lt.ngravprocs)then
         if(myrank.eq.ngravprocs-1) call cpu_time(time0)
         if(nusegpus.eq.1)then
            call lasthalf_grav_forces(ntot, gx, gy, gz, grpot)
         else
            call get_gravity_using_cpus
         endif
         if(myrank.eq.ngravprocs-1) call cpu_time(time1)
         if(ngravprocs.gt.1) then
            if(nusegpus.eq.1)then
               mygravlength=ngrav_upper-ngrav_lower+1
               if(myrank.ne.0) then
                  call mpi_gatherv(grpot(ngrav_lower), mygravlength, mpi_double_precision,
     $                 grpot, gravrecvcounts, gravdispls, mpi_double_precision, 0,
     $                 comm_worker, ierr)
               else
                  call mpi_gatherv(mpi_in_place, mygravlength, mpi_double_precision,
     $                 grpot, gravrecvcounts, gravdispls, mpi_double_precision, 0,
     $                 comm_worker, ierr)
               endif
            else
               call mpi_reduce(grpot, grpottot, n, mpi_double_precision,
     $              mpi_sum, 0, mpi_comm_world, ierr)
```

```
               if(myrank.eq.0) then
                  do i=1,n
                     grpot(i)=grpottot(i)
                  enddo
               endif
            endif
         endif
         if(myrank.eq.ngravprocs-1)then
            write(6,'(a,f6.3,a,g10.3,a,i4)')'2ndhalf:',time1-time0
         endif
         if(nusegpus.eq.1)then
            do i=ngrav_lower,ngrav_upper
               myvxdot(i)=myvxdot(i)+gx(i)
               myvydot(i)=myvydot(i)+gy(i)
               myvzdot(i)=myvzdot(i)+gz(i)
            enddo
         else
            do i=1,n
               myvxdot(i)=myvxdot(i)+gx(i)
               myvydot(i)=myvydot(i)+gy(i)
               myvzdot(i)=myvzdot(i)+gz(i)
            enddo
         endif
      endif

cccccccccccccccccccccccccccccccccccccccccc
c     reassemble acceleration arrays here
cccccccccccccccccccccccccccccccccccccccccc

c     use mpi_allreduce to collect subtotals, sum them, and redistribute
c     to all processes

c     Now that scaleheight has been calculated, we can include all
c     relevant terms in the calculation of the acceleration:
      do i=1,n
         myvxdot(i)=myvxdot(i)+myvxdotgc(i)
         myvydot(i)=myvydot(i)+myvydotgc(i)
         myvzdot(i)=myvzdot(i)+myvzdotgc(i)
      enddo

      call mpi_allreduce(myvxdot,vxdot,n,mpi_double_precision,mpi_sum,
     $     mpi_comm_world,ierr)
      call mpi_allreduce(myvydot,vydot,n,mpi_double_precision,mpi_sum,
     $     mpi_comm_world,ierr)
      call mpi_allreduce(myvzdot,vzdot,n,mpi_double_precision,mpi_sum,
     $     mpi_comm_world,ierr)

cc     if this is a relaxation calculation, add centrifugal and drag forces:
c      if (nrelax.ne.0 .and. .not. gonedynamic ) then
         call relax        ! add drag and fictitious forces if necessary
c     endif

      if(switchedav)then
         alpha=0.d0
         beta=0.d0
      endif

c      write(69,*)'balav3: finished'

c    ****************************
c    *    Print Sarah's Data    *
c    ****************************
      call mpi_reduce(myicount, icount, 1, mpi_integer, !ranks report myicount to rank0 for sum which is stored
into icount
     $     mpi_sum, 0, mpi_comm_world, ierr)

!      call mpi_allreduce(myicount,icount,1,mpi_integer,mpi_sum, !ranks report myicount to all ranks and all
ranks sum to icount
!     $      mpi_comm_world,ierr)

      if(myrank .eq. 0)then
         write(100+myrank,'(99g16.8)')  t, dt, Mdot_B, Mdot_Edd, Mdot_BH,
     $        Edot_fb, rhoBH, cs_BH2,v_BHrelgas2,vunit, vx_BHrelgas,
     $        vy_Bhrelgas,vz_BHrelgas,vx(ntot),vy(ntot),vz(ntot),
     $        vx(6970),vy(6970),vz(6970), icount
      endif
c    ****************************

      return
      end
```

balAV3.f   ~S model

```
c     variables used for Accretion/Feedback Model
      real*8 Rfb, Rfb2, Mencl_Rfb, xrelBH, yrelBH, zrelBH
      real*8 dist_to_BH2, vx_BHrelgas, vy_BHrelgas, vz_BHrelgas
      real*8 rhoBH, v_BHrelgas2, pBH, cs_BH2, gamma
      real*8 Mdot_BH,Mdot_B, Mdot_Edd, Edot_fb
      real*8 m_proton,epsilon_r,epsilon_f,alph,alphE,sigmaT,POR2BH
      real*8 halfRfb, invhalfRfb2, rhoRfb
      real*8 udot_Smodel, vunit, invhpj2,invhpj3,invh3
      real*8 rhoBH_cgs, Edot_fb_cgs
      integer itabBH, itabRfb, icount, myicount
c     set Rfb and gamma and other relavent constants here
      Rfb=4.0d0
      Rfb2=Rfb*Rfb
      halfRfb= 0.5d0*Rfb
      invhalfRfb2= 1.d0/halfRfb**2
      gamma=5d0/3d0
      m_proton= 1.6726219d-24 !in grams
      epsilon_r=0.1d0
      epsilon_f=0.05d0
      sigmaT= 6.6524d-25      !in cm**2
      alph=100d0
      alphE=1d0
      myicount=0
      vunit=(gravconst*munit/runit)**0.5d0 !used to convert velocities to cgs units
```

```
cccccccccccccccccccccccccccccccc
c        Sarah's Edits         c
cccccccccccccccccccccccccccccccc
c     Initialize relative velocities to zero (T model)
      vx_BHrelgas=0.d0
      vy_BHrelgas=0.d0
      vz_BHrelgas=0.d0
      rhoBH=0.d0 !initialize smoothed density to zero
      POR2BH=0.d0   !initialize smoothed pressure to zero
      cs_BH2=0.d0
      rhoRfb=0.d0

c share hp values with ranks
      mylength=n_upper-n_lower+1
      do irank=0,nprocs-1
         if(myrank.ne.irank)then
            call mpi_gatherv(hp(n_lower), mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         else
            call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         endif
      enddo

      do j=1, (ntot-1)
c        ****************************
c        * Calculate distance to BH *
c        ****************************
```

```
c       find particle's position relative to BH
         xrelBH=x(j)-x(ntot) !note: x(ntot) is the x position of the BH
         yrelBH=y(j)-y(ntot)
         zrelBH=z(j)-z(ntot)
         dist_to_BH2= xrelBH**2 + yrelBH**2 + zrelBH**2

c        ***************************************
c        * Calculate rel velocity of BH to gas *
c        ***************************************
         invhpj2=1/(hp(j)**2) !this is the inverse of the smoothing length squared (1/h^2)
         invhpj3=invhpj2/hp(j)
         if(dist_to_BH2 .lt. 4*hp(j)**2) then
            itab=int(ctab*dist_to_BH2*invhpj2)+1   !don't freak out; ctab is a known value
            vx_BHrelgas= vx_BHrelgas+am(j)*wtab(itab)*vx(j)*invhpj3
            vy_BHrelgas= vy_BHrelgas+am(j)*wtab(itab)*vy(j)*invhpj3
            vz_BHrelgas= vz_BHrelgas+am(j)*wtab(itab)*vz(j)*invhpj3
            rhoBH= rhoBH+am(j)*wtab(itab)*invhpj3  !calculate smoothed density at BH
            if (POR2(j) .le. 0) then  !checks to make sure that all particles have POR2 values
               write(10+myrank,*) "POR2 undefined for particle ", j
               stop
            endif
c            POR2BH= POR2BH+am(j)*wtab(itab)*invhpj3*POR2(j) !calculate smoothed pressure at BH
            cs_BH2= cs_BH2+am(j)*wtab(itab)*POR2(j)*invhpj3*rho(j)
         endif
         if(dist_to_BH2 .lt. Rfb2) then
            itabRfb=int(ctab*dist_to_BH2*invhalfRfb2)+1
            rhoRfb= rhoRfb+am(j)*wtab(itabRfb)*invhpj3
         endif
      enddo

      if (rhoBH .gt. 0d0)then
         vx_BHrelgas=vx_BHrelgas/rhoBH !when finding smoothed value at BH, you must divide by rhoBH after loop
         vy_BHrelgas=vy_BHrelgas/rhoBH
         vz_BHrelgas=vz_BHrelgas/rhoBH
         cs_BH2=cs_BH2/rhoBH

         vx_BHrelgas= (vx_BHrelgas -vx(ntot))
         vy_BHrelgas= (vy_BHrelgas -vy(ntot))
         vz_BHrelgas= (vz_BHrelgas -vz(ntot))

         v_BHrelgas2= vx_BHrelgas**2 + vy_BHrelgas**2 + vz_BHrelgas**2 !use components to find v^2 of BH relative
to gas
         v_BHrelgas2=v_BHrelgas2*vunit**2  !convert to cgs units


c         pBH= POR2BH*rhoBH**2   !when finding smoothed value at BH, you must divide by rhoBH after loop
         rhoBH_cgs= rhoBH*munit/runit**3 !convert to cgs units
c         pBH=pBH*gravconst*munit**2/runit**4 !convert to cgs units

c         rhoRfb_cgs= rhoRfb*munit/runit**3 !convert to cgs units

         cs_BH2=cs_BH2*gamma*vunit**2 !calculate local sound speed (in cgs units)

c        **************************************************
c        * Calculate mass accretion rate of BH (Mdot_BH) *
c        **************************************************

c     calculate Mdot_B, the Bondi-Hoyle mass accretion rate (cgs)
         Mdot_B=4*pi*alph*(gravconst**2)*(am(ntot)**2*munit**2)*rhoBH_cgs/
     $        ((cs_BH2+v_BHrelgas2)**1.5d0)

c     calculate Mdot_Edd, the Eddington-limitted mass accretion rate (cgs)
         Mdot_Edd=4*pi*alphE*gravconst*(am(ntot)*munit)*m_proton/
     $        (epsilon_r*sigmaT*crad)

c     for the acctual accretionrate of the BH, Mdot_BH, take minimum of prev two (cgs)
         Mdot_BH= min(Mdot_Edd,Mdot_B)

      else
         Mdot_Edd=0d0    !)
         Mdot_B=0d0      !> for when we print out these values, they need a value
         cs_BH2=0d0      !)
         Mdot_BH=0d0
      endif
c        *******************************************
c        * Calculate rate of BH's Feedback Energy  *
c        *******************************************

      Edot_fb_cgs = epsilon_r*epsilon_f*Mdot_BH*crad2
      Edot_fb= Edot_fb_cgs*runit**2.5d0/(gravconst**1.5d0*munit**2.5d0) !convert Edot_fb term to code units

c     Search for "S Model Term" to see the udot compontent corresponding to PK Model equation

cccccccccccccccccccccccccccccccc
c     End of Sarah's edits     c
cccccccccccccccccccccccccccccccc
```

```
         invh2 = 1.0/h2
         invh3 = invh*invh2
```

```
         if(nintvar.eq.2) then
c          ************************************
c          * Calculate S Model Term for udot  *
c          ************************************
            if(i .ne. ntot)then
               if(rhoRfb .gt. 0d0)then
                  dist_to_BH2=(x(i)-x(ntot))**2+(y(i)-y(ntot))**2+ !calculate particle's dist to BH
     $                 (z(i)-z(ntot))**2
                  if(dist_to_BH2 .lt. Rfb2)then
                     myicount=myicount+1
                     itabBH=int(ctab*dist_to_BH2*invh2)+1
                     itabRfb=int(ctab*dist_to_BH2*invhalfRfb2)+1
                     udot_Smodel=(Edot_fb*wtab(itabRfb)*invh3/rhoRfb) !***** S Model Term ***** (code units)
                     udot(i)=udot(i)+udot_Smodel
                  endif
               endif
            endif
c          ************************************
         endif
```

```
c    ****************************
c    *    Print Sarah's Data    *
c    ****************************
      call mpi_reduce(myicount, icount, 1, mpi_integer, !ranks report myicount to rank0 for sum which is stored
into icount
     $     mpi_sum, 0, mpi_comm_world, ierr)

!      call mpi_allreduce(myicount,icount,1,mpi_integer,mpi_sum, !ranks report myicount to all ranks and all
ranks sum to icount
!     $      mpi_comm_world,ierr)

      if(myrank .eq. 0)then
         write(100+myrank,'(99g16.8)')  t, dt, Mdot_B, Mdot_Edd, Mdot_BH,
     $        Edot_fb, rhoBH, cs_BH2,v_BHrelgas2,vunit, vx_BHrelgas,
     $        vy_Bhrelgas,vz_BHrelgas,vx(ntot),vy(ntot),vz(ntot),icount
      endif
c    ****************************
```

balAV3.f ~T model

```
c     variables used for Accretion/Feedback Model (this is Tophat Model)
      real*8 Rfb, Rfb2, Mencl_Rfb, xrelBH, yrelBH, zrelBH
      real*8 dist_to_BH2, vx_BHrelgas, vy_BHrelgas, vz_BHrelgas
      real*8 rhoBH, v_BHrelgas2, pBH, cs_BH2, gamma
      real*8 Mdot_BH,Mdot_B, Mdot_Edd, Edot_fb
      real*8 m_proton,epsilon_r,epsilon_f,alph,alphE,sigmaT,POR2BH
      real*8 udot_Tmodel, vunit, invhpj2, invhpj3
      real*8 rhoBH_cgs,Edot_fb_cgs
      integer icount, myicount
c     set Rfb and gamma and other relavent constants here
      Rfb=4.0d0
      Rfb2=Rfb*Rfb
      gamma=5d0/3d0
      m_proton= 1.6726219d-24 !in grams
      epsilon_r=0.1d0
      epsilon_f=0.05d0
      sigmaT= 6.6524d-25      !in cm**2
      alph=100d0
      alphE=1d0
      myicount=0
      vunit= (gravconst*munit/runit)**0.5d0 !used to convert velocities to cgs units
```

```
cccccccccccccccccccccccccccccccc
c        Sarah's Edits         c
cccccccccccccccccccccccccccccccc
c     Initialize relative velocities to zero (T model)
      vx_BHrelgas=0.d0
      vy_BHrelgas=0.d0
      vz_BHrelgas=0.d0
      rhoBH=0.d0 !initialize smoothed density to zero
      POR2BH=0.d0   !initialize smoothed pressure to zero
      cs_BH2=0.d0
      Mencl_Rfb=0.d0 !Initialize Mencl_Rfb to zero (T model)

c share hp values with ranks
      mylength=n_upper-n_lower+1
      do irank=0,nprocs-1
         if(myrank.ne.irank)then
            call mpi_gatherv(hp(n_lower), mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         else
            call mpi_gatherv(mpi_in_place, mylength, mpi_double_precision,
     $           hp, recvcounts,displs, mpi_double_precision, irank,
     $           mpi_comm_world, ierr)
         endif
      enddo

      do j=1, (ntot-1)
c        ***********************
c        * Calculate Mencl_Rfb *
c        ***********************
c       find particle's position relative to BH
         xrelBH=x(j)-x(ntot) !note: x(ntot) is the x position of the BH
         yrelBH=y(j)-y(ntot)
         zrelBH=z(j)-z(ntot)
         dist_to_BH2= xrelBH**2 + yrelBH**2 + zrelBH**2
c       find mass enclosed in feedback radius (Mencl_Rfb)
         if(dist_to_BH2 .LE. Rfb2) Mencl_Rfb = Mencl_Rfb + am(j)

c        ***************************************
c        * Calculate rel velocity of BH to gas *
c        ***************************************
         invhpj2=1/(hp(j)**2) !this is the inverse of the smoothing length squared (1/h^2)
         invhpj3=invhpj2/hp(j)
         if(dist_to_BH2 .lt. 4*hp(j)**2) then
            itab=int(ctab*dist_to_BH2*invhpj2)+1   !don't freak out; ctab is a known value
            vx_BHrelgas= vx_BHrelgas+am(j)*wtab(itab)*vx(j)*invhpj3
            vy_BHrelgas= vy_BHrelgas+am(j)*wtab(itab)*vy(j)*invhpj3
```

```
               vz_BHrelgas= vz_BHrelgas+am(j)*wtab(itab)*vz(j)*invhpj3
               rhoBH= rhoBH+am(j)*wtab(itab)*invhpj3  !calculate smoothed density at BH
               if (POR2(j) .le. 0) then  !checks to make sure that all particles have POR2 values
                  write(10+myrank,*) "POR2 undefined for particle ", j
                  stop
               endif
c               POR2BH= POR2BH+am(j)*wtab(itab)*invhpj3*POR2(j) !calculate smoothed pressure at BH
               cs_BH2= cs_BH2+am(j)*wtab(itab)*POR2(j)*invhpj3*rho(j)
            endif
         enddo

         if (rhoBH .gt. 0d0)then
            vx_BHrelgas=vx_BHrelgas/rhoBH !when finding smoothed value at BH, you must divide by rhoBH after loop
            vy_BHrelgas=vy_BHrelgas/rhoBH
            vz_BHrelgas=vz_BHrelgas/rhoBH
            cs_BH2=cs_BH2/rhoBH

            vx_BHrelgas= (vx_BHrelgas -vx(ntot))
            vy_BHrelgas= (vy_BHrelgas -vy(ntot))
            vz_BHrelgas= (vz_BHrelgas -vz(ntot))

            v_BHrelgas2= vx_BHrelgas**2 + vy_BHrelgas**2 + vz_BHrelgas**2 !use components to find v^2 of BH relative
to gas
            v_BHrelgas2=v_BHrelgas2*vunit**2  !convert to cgs units


c            pBH= POR2BH*rhoBH**2   !when finding smoothed value at BH, you must divide by rhoBH after loop
            rhoBH_cgs= rhoBH*munit/runit**3 !convert to cgs units
c            pBH=pBH*gravconst*munit**2/runit**4 !convert to cgs units

            cs_BH2=cs_BH2*gamma*vunit**2 !calculate local sound speed (in cgs units)

c           *************************************************
c           * Calculate mass accretion rate of BH (Mdot_BH) *
c           *************************************************

c        calculate Mdot_B, the Bondi-Hoyle mass accretion rate
            Mdot_B=4*pi*alph*(gravconst**2)*(am(ntot)**2*munit**2)*rhoBH_cgs/
     $           ((cs_BH2+v_BHrelgas2)**1.5d0)

c        calculate Mdot_Edd, the Eddington-limitted mass accretion rate
            Mdot_Edd=4*pi*alphE*gravconst*(am(ntot)*munit)*m_proton/
     $           (epsilon_r*sigmaT*crad)

c        for the acctual accretionrate of the BH, Mdot_BH, take minimum of prev two
            Mdot_BH= min(Mdot_Edd,Mdot_B)

         else
            Mdot_Edd=0d0    !)
            Mdot_B=0d0      !> for when we print out these values, they need a value
            cs_BH2=0d0      !)

            Mdot_BH=0d0
         endif
c           *******************************************
c           * Calculate rate of BH's Feedback Energy  *
c           *******************************************

         Edot_fb_cgs = epsilon_r*epsilon_f*Mdot_BH*crad2
         Edot_fb= Edot_fb_cgs*runit**2.5d0/(gravconst**1.5d0*munit**2.5d0) !convert Edot_fb term to code units

c        Search for "T Model Term" to see the udot compontent corresponding to T Model equation

cccccccccccccccccccccccccccccccc
c        End of Sarah's edits     c
cccccccccccccccccccccccccccccccc
```

```
         if(nintvar.eq.2) then
c          ************************************
c          * Calculate T Model Term for udot  *
c          ************************************
            if(i .ne. ntot)then
               if(Mencl_Rfb .gt. 0d0)then
                  dist_to_BH2=(x(i)-x(ntot))**2+(y(i)-y(ntot))**2+ !calculate particle's dist to BH
     $                 (z(i)-z(ntot))**2
                  if(dist_to_BH2 .lt. Rfb2)then
                     myicount=myicount+1
                     udot_Tmodel=(Edot_fb/Mencl_Rfb) !***** T Model ***** (code units)
                     udot(i)=udot(i)+udot_Tmodel
                  endif
               endif
            endif
c          ************************************
         endif
```

```
c    ****************************
c    *    Print Sarah's Data    *
c    ****************************
      call mpi_reduce(myicount, icount, 1, mpi_integer, !ranks report myicount to rank0 for sum which is stored
into icount
     $     mpi_sum, 0, mpi_comm_world, ierr)

!      call mpi_allreduce(myicount,icount,1,mpi_integer,mpi_sum, !ranks report myicount to all ranks and all
ranks sum to icount
!     $      mpi_comm_world,ierr)

      if(myrank .eq. 0)then
         write(100+myrank,'(99g16.8)')  t, dt, Mdot_B, Mdot_Edd, Mdot_BH,
     $        Edot_fb, rhoBH, cs_BH2,v_BHrelgas2,vunit, vx_BHrelgas,
     $        vy_Bhrelgas,vz_BHrelgas,vx(ntot),vy(ntot),vz(ntot),icount
      endif
c    ****************************
```