How to Build A Galaxy

Noah Muldavin

July 19, 2013

Introduction

The selection of initial conditions for an astrophysical simulation is not a trivial matter. First you must come up with a series of functions which represent the system you're studying in a way which can be modeled (already not easy), then one must come up with some way of sampling from these functions such that the positions and velocities of each particle are assigned correctly. This guide is meant to help out with both processes so that it is a simple matter of inputting the desired functions and parameters to build a composite galaxy (disk, bulge, halo, or more) with realistic initial conditions. It is meant to accompany the galactic initial conditions package which consists of a set of helper functions designed to help assign distributions and an example of how to build a composite galaxy according to the Hernquist method [7].

1 How to Build a Galaxy with `makegalaxy.cpp`

I've included a routine to construct a composite galaxy (bulge, disk, halo) according to the Hernquist method in the file `makegalaxy.cpp`. The code may be compiled with the terminal command `make -f init.make` and run with `./init`

2 How to Generate Random Numbers

For the best results the positions and velocities of ever mass should be randomly selected from a set of probability distributions designed to model a realistic system. Unfortunately assigning random numbers according to an arbitrary distribution is not a trivial matter. The C standard library includes a `rand()` function which will randomly generate an integer between 0 and some maximum value `RAND_MAX`, with an essentially even probability of selecting any integer.[1] The specific value of `RAND_MAX` varies depending on how recent the library is. When I tested it on my computer just now, `RAND_MAX` was $2,147,483,647$. The `rand()` function is only effective when it has has been initialized with a *random seed*. The `rand()` function isn't truly random, it just generates an *approximately* random number using some algorithm that *starts* with a given number, the random seed. It is common practice to seed the random number generator with the *time*, since it can't be predicted exactly before you run the code. This is done with the command `srand ((double) time(NULL));` , which requires the `<ctime>` package.

[1]The nerdz will complain about how `rand()` isn't very random, or doesn't select very evenly. It is true, there are probably better random number generating algorithms out there, maybe as part of the `<random>` package. However for the purposes here, where we're just trying to sample from an already approximate distribution function (not encrypt NSA secrets) `rand()` should do perfectly fine. You can ignore the nerdz.

The challenge when generating random numbers is to turn the random integer generated by `rand()` into a random number fitting a desired probability distribution function. I've included a number of functions in the files `helperfunctions.hpp` and `helperfunctions.cpp` which should allow you to generate a set of random numbers following most distributions. Below I've listed some types of distributions along with the functions that will generate them:

- Uniform Distribution: `randomreal(min, max)`

  The `rand()` function already generates a uniform distribution of integers between 0 and `RAND_MAX`, all you have to do to generate a number in the desired range is scale and shift the range until it fits. First a number in the range $[0, 1]$ is generated by dividing the result of `rand()` by `RAND_MAX`. Then the range is scaled so the width is `max - min`, and shifted so that the smallest number is `min`.

- Gaussian Distribution: `gaussianrandom(sigma)`

  Often one is interested in selecting a random number from the gaussian distribution

  $$g(x) = e^{-x^2/2\sigma^2}$$

  where σ is the *dispersion* or *standard deviation* of the distribution. A gaussian distribution of random numbers can be generated with a *Box-Muller transform.* If two random numbers x_1 and x_2 are selected from an even distribution between 0 and 1, the number

  $$y_1 = \sqrt{-2\sigma \ln(x_1)}\cos(2\pi x_2)$$

  will be generated according to a gaussian distribution. I'm not going to prove it here, you can look it up. The function `gaussianrandom(sigma)` generates a gaussian random number with this method, given a dispersion `sigma`.

- Spherical Distribution: `randomsphere(r)`

  What if you wanted to select the coordinates of a vector such that it lies at a random position on the surface of a sphere of a given radius? This might occur when you're trying to build a spherically symmetric spacial distribution of particles, or in any situation where you were trying to randomize the components of a three-vector (v_x, v_y, v_z) such that the norm of the vector maintains a specified value. Unfortunately you can't just randomly select the coordinates ϕ and θ in spherical coordinates, since they do not map to an even spatial distribution.

  One way to do so is called the golden spiral method. Imagine that the unit sphere is divided into `RAND_MAX` slices along the y axis. A slice of the sphere can be randomly selected with the `rand()` function, and the corresponding position y can be determined easily. Each slice will have radius $r = \sqrt{1 - y^2}$. In the golden spiral method, the angle ϕ at which each point is placed in its respective slice is offset by an amount $\Delta\phi = \pi * (3 - \sqrt{5})$ per integer value. It turns out that offsetting the masses as such produces a pretty even distribution. To be perfectly honest I don't really get it either, but you can try it, it works.

  The function `randomsphere(r)` will return a three-vector in cartesian coordinates that lies at a random position on a sphere of radius r.

- Arbitrary Distribution: `rejectionsample(function, min, max, topbound)`

Transforming the uniform distribution produced by `rand()` to a given distribution is hard. Instead you can jerry-rig an arbitrary distribution by throwing a bunch of trials out. Here's how it works:

Given an arbitrary one-dimensional distribution function $f(x)$, a random number x is generated in the desired range. Next another number y is generated between 0 and $f_{\max}$, the maximum height of the function. If y is greater than the height of the function at x, $f(x)$, x is returned. If not the process is repeated until a set of numbers is selected such that $y \leq f(x)$. This way *the probability of returning a number located at x is directly proportional to the height of the function $f(x)$.*

The problem with this method is that it wastes a lot of time since so many numbers are rejected. However, it is easy to implement and should be sufficient here. The function `rejectionsample(function, min, max, topbound)` will generate a random number between `min` and `max` according to a one-dimensional function specified at `function`. You must also tell it what the maximum height of the function is, at `topbound`. If you don't know the maximum height, the function `findtopbound(function, min, max)` will find it for you, but I would be careful about using this often because it takes a while. Another advantage here is that the function you input doesn't need to be normalized since only proportionality matters. You can forget about all those annoying constants.

3 The Distribution Function

An N-Body simulation is concerned with the evolution of a *distribution* of mass. Mathematically this distribution is represented by the *distribution function* $f(\mathbf{x}, \mathbf{v}, t)$ which is defined such that $f(\mathbf{x}, \mathbf{v}, t)d^3\mathbf{x}d^3\mathbf{v}$ is the probability that a randomly selected star has phase space coordinates located in the volume $d^3\mathbf{x}d^3\mathbf{v}$ located at $(\mathbf{x}, \mathbf{v})$ in phase space, at time t. Naturally f must be normalized so that

$$\int f(\mathbf{x}, \mathbf{v}, t)d^3\mathbf{x}d^3\mathbf{v} = 1. \tag{1}$$

It is reasonably straightforward to derive an equation of motion for the distribution function (see [1] section 4.1 or [5] section 2.3):

$$\frac{\partial f}{\partial t} + \dot{\mathbf{q}} \cdot \frac{\partial f}{\partial \mathbf{q}} + \dot{\mathbf{p}} \cdot \frac{\partial f}{\partial \mathbf{p}} = 0. \tag{2}$$

This is known as the *collisionless Boltzman equation.* Unfortunately it cannot be solved in any but the most trivial cases. An N-Body simulation, in essence, is a method which simulates the evolution of the distribution function by sampling it with discrete masses (for a full discussion see my thesis, Reference [5], chapters 2 and 3). This leaves open the question of initial conditions: How does one sample the initial distribution function $f(\mathbf{x}, \mathbf{v}, 0)$ with discrete masses in order to realistically model the system at hand? This is a difficult task because the distribution function of realistic systems (which might contain a spinning disk, a halo of dark matter, a central bulge, or more) is nearly impossible to specify completely with tractable functions.

There is no easy solution, in fact the construction of realistic initial conditions is a very open area of research. Generally the problem is tackled by *breaking up* the distribution function into a number of manageable pieces which together approximate the entire distribution function. Below I describe the Hernquist method (Reference [7]) for constructing the initial conditions of a composite disk galaxy (disk, halo, bulge) starting with the position distribution of each, and proceeding to the velocity distribution.

4 Position Distribution

The exact position distribution of mass in a galaxy is easily obtained from the distribution function. The probability of finding a randomly selected star at location $\mathbf{x}$ regardless of its velocity may be obtained by integrating f over all possible velocities,

$$\nu(\mathbf{x}) = \int f(\mathbf{x}, \mathbf{v}) d^3\mathbf{v} \tag{3}$$

Multiplying ν by the total mass M gives the mass density

$$\rho(\mathbf{x}) = M\nu(\mathbf{x}). \tag{4}$$

The mass density distribution $\rho(\mathbf{x})$ is a quantity which is easy to obtain from observations of the system you're studying. For galaxies the mass density distribution is often broken up into three parts corresponding to the disk, halo of dark matter, and bulge if present.

4.1 Disk

The disk is often modeled (in cylindrical coordinates) with the form

$$\rho_d(r, z) = \frac{M_d}{4\pi h^2 z_0} \exp\left(-r/h\right) \operatorname{sech}^2\left(\frac{z}{z_0}\right) \tag{5}$$

where M_d is the mass of the disk, h is the radial scale length, and z_0 is a vertical scale thickness. In order to create a distribution of this form, one would randomly select the particle radii (in cylindrical coordinates) from the function

$$f_{\text{rad}}(r) = r \exp\left(-r/h\right), \tag{6}$$

where the r has been added to the original density distribution to account for the volume element of cylindrical coordinates. The disk must be truncated at some maximum radius r_{cut} to avoid the exorbitant computational cost of integrating masses in the very outer disk. The azimuthal position ϕ would be selected from a uniform distribution between 0 and 2π. The vertical distribution would be selected from the function

$$f_{\text{vert}}(z) = \operatorname{sech}\left(\frac{z}{z_0}\right). \tag{7}$$

This too should be truncated at some value z_{cut}. Finally the coordinates would be transformed back to cartesian and assigned.

4.2 Halo

The exact distribution of dark matter around galaxies is unknown. It is often modeled as a spherically symmetric distribution with the density profile (in spherical coordinates)

$$\rho_h(r) = \frac{M_h}{2\pi^{3/2}} \frac{\alpha}{r_c} \frac{\exp\left(-r^2/r_c^2\right)}{r^2 + \gamma^2}, \tag{8}$$

where M_h is the total mass of the halo, r_c and γ are scale lengths, and α is a normalization constant. To build a spherical distribution with this profile, one would randomly select radii from the function

$$f_{\text{halo}}(r) = \frac{\exp\left(-r^2/r_c^2\right)}{r^2+\gamma^2} r^2 \tag{9}$$

where the r^2 has been added to account for the volume element in spherical coordinates. There should be some maximum radius so you don't waste computation time integrating particles way out that don't do anything interesting. Once a radius has been assigned, the particle's position on a sphere of that radius should be selected randomly. This can be done with the `randomsphere()` function.

4.3 Bulge

Observations show that bulges in spiral galaxies are often flattened slightly by some net rotation. In this case the particles should be initialized in some elliptical distribution. It is common practice, however, to approximate the distribution with the spherical distribution

$$\rho_b(r) = \frac{M_b}{2ma^2} \frac{1}{r\left(1+r/a\right)^3} \tag{10}$$

where a is a scale length and M_b is the mass of the bulge. In this case the initial conditions may be assigned exactly as with the halo: randomly selecting a radius from the appropriate distribution, then randomizing the position on a sphere.

5 Velocity Structure

Unfortunately the velocity structure of a realistic galaxy is not as simple to construct as the position distribution. In order for the simulation to be useful the system must be *near* equilibrium initially, since it wasn't you might end up with unrealistically violent behavior. Given a position distribution, there should exist a full distribution function (velocity included) which is at equilibrium. Unfortunately these distributions cannot be specified in all but the most trivial cases.

The workaround is to break the velocity distribution up into its first and second *moments*. The first moment is simply the average velocity at a given location in space. To derive this from the distribution function, note first that the probability distribution of velocities at a location $\mathbf{x}$, $P_{\mathbf{x}}(\mathbf{v})$, may be obtained by dividing f by the position-probability distribution ν:

$$P_{\mathbf{x}}(\mathbf{v}) = \frac{f(\mathbf{x},\mathbf{v})}{\nu(\mathbf{x})}. \tag{11}$$

From this it is easy to calculate the mean velocity at a location $\mathbf{x}$,

$$\overline{\mathbf{v}}(\mathbf{x}) = \int P_{\mathbf{x}}(\mathbf{v})\, \mathbf{v}\, d^3\mathbf{v} = \frac{1}{\nu(\mathbf{x})} \int f(\mathbf{x},\mathbf{v})\mathbf{v} d^3\mathbf{v}. \tag{12}$$

The second moment of the velocity distribution characterizes the "spread" in velocities around $\overline{\mathbf{v}}(\mathbf{x})$. In its most general form, this spread is characterized by the velocity dispersion tensor

$$\sigma_{ij}^2(\mathbf{x}) = \frac{1}{\nu(\mathbf{x})} \int (v_i - \overline{v_i})(v_j - \overline{v_j}) f(\mathbf{x},\mathbf{v}) d^3\mathbf{v} \tag{13}$$

where v_i and v_j are *components* of velocity. Evaluating each integral returns a more familiar definition of velocity dispersion,

$$\sigma_{ij}^2(\mathbf{x}) = \overline{v_i v_j} - \overline{v_i}\,\overline{v_j}. \tag{14}$$

The interpretation of the tensorial character of the velocity dispersion is rather murky. It is clear that σ_{ij}^2 is symmetric, therefore there are six independent components. The diagonal components represent the usual statistical dispersion in each direction. The off-diagonal components are more confusing, but luckily it turns out that if the coordinate system you're using is related to the symmetries of the potential the off diagonal elements are zero [1]. Thus when it boils down to the construction of initial conditions one only need to specify the dispersion in each coordinate: in cartesian coordinates, $(\sigma_x, \sigma_y, \sigma_z)$, in cylindrical coordinates $(\sigma_r, \sigma_\phi, \sigma_z)$, etc.

The first and second moments of the velocity distribution together should approximate the full distribution function. This leaves the task of determining appropriate functions from which to select $\overline{\mathbf{v}}(\mathbf{x})$ and the dispersions such that the system is at or near equilibrium. This is an achievable but still very difficult task which requires the use of *moments* of the collisionless Boltzmann equation. This one is best left to the pros, but luckily as long as you cite it's perfectly ok to steal their methods. Below I describe in detail the Hernquist method for assigning velocities for each component of a composite galaxy, starting with the halo since the spherical symmetry makes things a bit easier.

5.1 Halo

In a spherical distribution of stars which is isotropic (meaning that the velocity distribution looks the same in all directions) the average velocity $\overline{\mathbf{v}}(\mathbf{x})$ will be zero everywhere. This doesn't mean that the mass isn't moving, just that as many stars are moving out as in, left as right, etc. Thus the entire velocity distribution is determined by the second moment, the velocity dispersion in each coordinate. Using the second moment of the distribution function it can be shown [7] that the radial dispersion is given by

$$\sigma_r^2 = \overline{v_r^2} = \frac{1}{\rho_h(r)} \int_r^\infty \rho_h(r) \frac{GM(r)}{r^2} dr, \tag{15}$$

where $\rho_h(r)$ is the known analytic form of the halo mass density and $M(r)$ is the cumulative mass distribution of the entire system (the mass interior to radius r).

In order to calculate $\overline{v_r^2}$ equation 15 must be integrated numerically. To do so in a reasonable amount of time, the cumulative mass distribution $M(r)$ must be estimated at a number of evenly spaced discrete steps. Once the number of steps is set (I used 1000), the mass within the radius corresponding to each step is summed. The result is a list M_i of the cumulative mass at a set of discrete radii separated by some amount Δr. To integrate equation 15 the starting index is determined and the sum

$$\overline{v_r^2} \simeq \frac{1}{\rho_h(r)} \sum_{i=\text{start}}^{\text{max}} \rho_h(i\Delta r) M_i \Delta r. \tag{16}$$

is evaluated.

Since the halo is isotropic by assumption, the dispersion in each cartesian coordinate should be equal at any point in space. Thus once $\overline{v_r^2}$ has been calculated the *speed* v of the particle may be selected from

$$F(v, r) = Av^2 \exp\left(-v^2/2\overline{v_r^2}\right) \tag{17}$$

where A is a normalization constant. The cartesian components are then assigned randomly and uniformly such that the magnitude remains v. This may be done using the `randomsphere()` function since the surface in velocity space such with constant magnitude v that does not favor a coordinate is a sphere.

There are two potential problems with this method. The first is that when randomly selecting from $F(v, r)$ you might accidentally select a speed which is higher than the escape velocity at that location, thus causing that mass to fly to infinity. This may be prevented by limiting the speed to the escape velocity

$$v_{\rm esc} = \sqrt{\frac{2GM(r)}{r}}. \tag{18}$$

The second problem is that this method of assigning velocities assumes that the *entire* system is spherically symmetric, including the global potential. Because of the contribution of the disk this assumption is untrue. At large radii the non-spherically symmetric contribution of the disk is negligible since most of the potential will be due to the halo. Near the center this approximation might be less accurate. This may cause some initial disequilibrium near the center of the system. Empirical tests show that this disequilibrium is minor [7].

5.2 Bulge

Since the bulge is also spherically symmetric and isotropic, its velocities may be initialized in exactly the same manner as the halo with the substitution of $\rho_b(r)$ for $\rho_h(r)$.

5.3 Disk

The disk is a bit more complicated, since it's flat and spinning. Thus initializing the velocities comes in several steps. In cylindrical coordinates, the velocity structure of the disk may be approximated by the radial dispersion $\overline{v_r^2}$, the vertical dispersion $\overline{v_z^2}$, the azimuthal dispersion σ_ϕ^2, and the azimuthal streaming velocity $\overline{v_\phi}$.

It is easiest to start with the vertical dispersion. Observationally it is true that the vertical thickness of galactic disks is independent of radius. In order for this to be true the vertical dispersion must be given by

$$\overline{v_z^2} = \pi G \Sigma(r) z_0 \tag{19}$$

where $\Sigma(r)$ is the *surface density* of the disk,

$$\Sigma(r) = \frac{M_d}{2\pi h^2} \exp\left(-r/h\right). \tag{20}$$

Naturally $\Sigma(r)$ is just the total disk density integrated over the vertical axis.

Observations suggest that the radial dispersion should also be proportional to the surface density so that

$$\overline{v_r^2} = A\, \Sigma(r) \tag{21}$$

where A is some constant. How should this constant be assigned?

In 1964 the mathematician Alar Toomre [2] showed that the local stability of a uniformly rotating disk may be parametrized in terms of the quantity

$$Q = \frac{\sigma_r \kappa}{3.36\, G\, \Sigma(r)} \tag{22}$$

where κ is the *epicyclic frequency* of oscillations around a perfectly circular orbit at a radius r given by

$$\kappa^2(r) = \frac{3}{r}\frac{\partial \Phi}{\partial r} + \frac{\partial^2 \Phi}{\partial r^2} \tag{23}$$

where Φ is the potential. Q can be thought of as a temperature scale for the disk since it is directly correlated to the velocity dispersion σ_r. A "hot" disk has a large velocity dispersion and a high Q, a "cool" disk has a small velocity dispersion and a low Q, while a "cold" disk has zero velocity dispersion and $Q = 0$. The latter case corresponds to a disk in which every mass rotates about the center on a perfectly circular orbit.

Toomre showed that there is a sweet spot for the formation of non-axisymmetric perturbations such as bars and spirals in the range

$$1 < Q < 2. \tag{24}$$

If $Q < 1$ the disk is violently unstable. If $Q > 2$ there is too much random motion for interesting perturbations to develop. Long story short: Q is an important measure of the "temperature" of disk. It can and should be used as a parameter in the simulation to set the normalization constant A.

Practically this is achieved by calculating the average *critical dispersion*,

$$\sigma_r|_{\mathrm{crit}} = \frac{3.36 G \Sigma(r)}{\kappa} \tag{25}$$

in a narrow annulus around some reference radius r_{ref}. The epicyclic frequency κ must be derived from force calculation. Equation 23 may be rewritten in terms of the radial acceleration $a_r = -\partial\Phi/\partial r$:

$$\kappa^2 = -\left(\frac{3}{r}a_r + \frac{\partial a_r}{\partial r}\right). \tag{26}$$

The derivative $\partial f_r/\partial r$ may be calculated by comparing the force at two points separated by some small amount Δr. The `helperfunctions.cpp` file contains a function `epicyclicfrequency()` which approximates κ for a single particle. This takes two force calculations per particle, so running it for every particle in the disk will take some time. In order to use it you must have built a tree with `maketree()`. Since you want to know the force as accurately as possible, make sure to use a very low value of the θ_{crit}. Once calculated, the constant A may be assigned such that (21) returns the correct value at r_{ref}:

$$\left(Q\,\sigma_r|_{\mathrm{crit}}\right)^2 = \overline{v_r^2} = A\,\Sigma(r). \tag{27}$$

Unfortunately the assumptions underlying the above relations for determining $\overline{v_r^2}$ and $\overline{v_z^2}$ tend to break down near the center of the disk where the surface density $\Sigma(r)$ is changing quickly. Once fix is to instead select dispersions from the softened relations

$$\overline{v_r^2} = A\,\Sigma\left(-\sqrt{r^2 + 2a_s^2}/h\right), \tag{28}$$

$$\overline{v_z^2} = \pi G\,\Sigma\left(-\sqrt{r^2 + 2a_s^2}/h\right) z_0, \tag{29}$$

where a_s is a softening length. Empirical tests show that setting $a_s \sim h/4$ is sufficient.

This leaves the two azimuthal moments σ_ϕ and $\overline{v_\phi}$. The azimuthal dispersion is generally related to the radial dispersion by the epicyclic approximation:

$$\sigma_\phi^2 = \overline{v_r^2}\frac{\kappa^2}{4\Omega^2}, \tag{30}$$

where Ω is the angular frequency of a circular orbit, which is calculated from the radial acceleration:

$$\Omega = \sqrt{\frac{a_r}{r}}. \tag{31}$$

The final piece of the puzzle is the azimuthal streaming velocity $\overline{v_\phi}$. The streaming velocity should be related to the velocity of a perfectly circular orbit $v_c = \Omega r$. Observations in the solar neighborhood show that the streaming velocity $\overline{v_\phi}$ of stars often lags behind the circular velocity. This is a phenomenon known as *asymmetric drift* (see [1] section 4.8.2 (a)). It turns out that this phenomenon is explained by the second moment of the collisionless boltzmann equation (I don't really understand it either, you can try too if you want but I'm just taking their word for it). For an *exponential disk* such as the one in the Hernquist model, the streaming velocities may be selected from the relation

$$\overline{v_\phi}^2 - v_c^2 = \overline{v_r^2}\left(1 - \frac{\kappa^2}{4\Omega^2} - 2\frac{r}{h}\right). \tag{32}$$

In order to set the total azimuthal velocity, the streaming velocity $\overline{v_\phi}$ is added to a random contribution pulled from a gaussian with dispersion σ_ϕ.

Finally, the cylindrical velocities are transformed back to cartesian velocities according to the relations

$$v_x = v_r \cos\phi - v_\phi \sin\phi, \tag{33}$$
$$v_y = v_r \sin\phi + v_\phi \cos\phi, \tag{34}$$
$$v_z = v_z. \tag{35}$$

That was long and convoluted, so I've created an ordered list of the processes required to assign velocities in the disk:

1. Assign radial dispersion normalization constant A in relation to the Toomre Q parameter at the reference radius r_{ref}.

2. For each particle

   (a) Calculate $\overline{v_z^2}$ from (29) and choose random vertical velocity.
   (b) Calculate $\overline{v_r^2}$ from (28) and choose random radial velocity.
   (c) Calculate Ω and v_c from the local radial acceleration.
   (d) Calculate κ from local radial acceleration and its derivative.
   (e) Calcualte σ_ϕ from (30) and choose random azimuthal velocity.
   (f) Assign $\overline{v_\phi}$ from (32).
   (g) Sum streaming azimuthal velocity $\overline{v_\phi}$ and random azimuthal velocity to give total.
   (h) Transform to cartesian velocities and assign.

6 The Important Parts

The Hernquist method discussed here will work for any galaxy consisting of a disk which follows an exponential distribution, and a spherically symmetric isotropic bulge and/or halo. You'll notice that for the most part the velocity distribution depends entirely on the position distribution. Once the position distribution for each of the galaxy components is set, the velocity structure is determined entirely given a value of the Toomre Q parameter to set the "temperature" of the disk. Here's a list of the important parameters:

1. $\rho_d(r, z)$ and accompanying parameters $(M_d, h, z_0, r_{\text{cut}}, z_{\text{cut}})$. Any exponential surface distribution is fine, but a non-exponential distribution won't work with the current method of selecting the azimuthal streaming velocity.

2. $\rho_h(r)$ and accompanying parameters $(M_h, r_c, \gamma, r_{\text{max}})$. Any spherically symmetric distribution will work.

3. $\rho_b(r)$ and accompanying parameters $(M_b, a, r_{\text{max, bulge}})$. Any spherically symmetric distribution will work.

4. Toomre Q parameter to control how "hot" the disk is.

The code in `makegalaxy.cpp()` should allow you to construct a composite galaxy with the desired parameters extremely easily. If you want to change something (make the disk hotter or colder, the halo more or less concentrated, take away the bulge) all you have to do is edit the parameters.

7 Final Note About `makegalaxy.cpp`

The N-Body integrator included in the TreeCode package takes a parameter Δt_{max} which determines the maximum timestep over which a particle may be integrated *and* the frequency at which visualization files are generated. Δt_{max} should be related to some fundamental timescale of the system at hand. For galaxies this timescale is often characterized by the *crossing time*,

$$t_{\text{cross}} = \frac{R}{v} \tag{36}$$

where R is the size of the system and v is the typical speed of a particle. You may wish to set the maximum timestep to some fraction of the crossing time (1/50, 1/200, whatever). For this reason I've added one last piece to the end of `makegalaxy.cpp` which calculates and reports the average velocity of particles in the disk, the size of the disk R which is set to the cutoff radius, and the crossing time $R/\overline{v}$.

When all is said and done, a file"initfile.txt" is generated in the data folder which contains all of the necessary initial condition information.

References

[1] J. Binney and S. Tremaine, *Galactic Dynamics* (Princeton University Press, Princeton, NJ, 2008), 2nd ed

[2] A. Toomre, Astrophys. J. **139**, 1217 (May 1964)

[3] J. E. Barnes, *Treecode Guide*, `http://www.ifa.hawaii.edu/~barnes/treecode/treeguide.html`

[4] D. M. Moore, thesis, Reed College, 1997

[5] N. Muldavin, thesis, Reed College, 2013

[6] E. Athanassoula, J. C. L. E Fady, and A. Bosma, Mon. Not. R. Astron. Soc. **000**, 1 (1999).

[7] L. Hernquist, Astrophys. J. Suppl. Ser. **86**, 389 (1993)