N-body Barnes-Hut

Mats Thijssen

1 Introduction

2D and 3D approximations to gravitational N-body simulations are constructed using the Barnes-Hut method, resulting in Nlog(N) computational time. The code is to be tested on several scenarios, including computation of rotational curves in galaxies and dynamical friction between Dark Matter halos.

2 Theory: Barnes-Hut

The Barnes-Hut approximation, designed by Josh Barnes & Piet Hut in 1986[1], reduces computational time by approximating long-distance interactions from multiple clustered sources as one object at the cluster's center of mass (COM). In practice, this is achieved by placing the objects of interest in a quadtree/octree (2D/3D). The following discussion will be based on 2D for simplicity, the generalization to 3D is trivial. To create a quadtree, the simulated region is recursively subdivided into square nodes by splitting the region at it's center. Each node is then recursively divided until each particle has a separate node. An example of a finished quadtree for 100 points generated with my code is shown below.

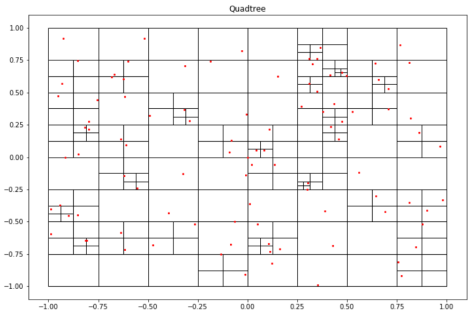

Figure 1: Example of Quadtree

Each node stores the information associated with it: the objects with their coordinates & masses, the center of mass of the node, and its size. When calculating the force on a particular object due to interactions with the others, we traverse the tree from top down. This is where the approximation comes into play. If the following condition is met for a particular node, we use that node's COM and total mass to generate the force from all the node's points:

$$\frac{s}{d} < \theta \tag{1}$$

Where s is the size of the node, d is the distance from the com to the point of interest, and θ is a set parameter. θ is usually chosen to be 0.5. A smaller θ gives a more accurate result, but increased runtime. $\theta = 0$ is equivalent to brute force ($O(n^2)$). If when traversing, the condition is not met, the node's "children" (or subnodes) are investigated in the same way. This continues recursively until the condition is met or we are at a node with only one particle. Approximating the forces this way reduces the calculation to $O(nlog(n))$. Figure 2 shows a simple example of how the tree is recursed in calculating forces.

[1]Barnes, Josh & Hut, Piet - A hierarchical O(N log N) force-calculation algorithm: http://dx.doi.org/10.1038/324446a0

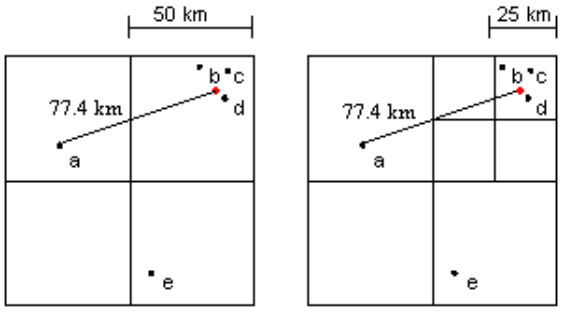

Figure 2: Force is calculated on (a). The first NE region is too big, so we recurse and use the child NE-region.(http://www.cs.princeton.edu/courses/archive/fall03/cs126/assignments/barnes-hut.html)

In the code, two objects are created: Node and Quadtree. As previously mentioned, nodes store the information about every point contained within in, as well the center of mass and node size. The Quadtree object stores the entire tree of points, i.e. a bunch of nodes. The code is simply run by creating arrays of points (coordinates+masses) and initial velocities and initialize the Quadtree, also specifying boxsize. The tree can subsequently be told to evolve into future time. After every time step, the tree is destroyed and recreated, to save memory and not run into problems with subtleties with points moving across nodes.
The integration for time-evolvement is carried out using the leapfrog algorithm. The algorithm updates the motional quantities as follows:

$$x(t+h) = x(t) + h * v(t+h/2)$$
$$v(t+3h/2) = v(t+h/2) + h * f(x(t+h), t+h)$$

To find the first velocity needed (at $t_0 + h/2$), Euler is used. We also find the velocities at the "position-points", if we wish to do an energy calculation. This is done by introducing the following two steps:

$$k = h * f(x(t+h), t+h)$$
$$v(t+h) = v(t+h/2) + k/2)$$

Note that this algorithm is time-symmetric ($h \to -h$) and thus conserves Energy, unlike a lot of other integration schemes. That is of course important in many gravitational systems. The total error in both position and velocity is of order h^2. Euler's method is also implemented, should one want to use that for some reason. In the future, other integration schemes could be implemented relatively easily as well. Details about the code can be seen by going through the code - it is well commented.

3 Theory: Rotation Curves

For pure circular motion in a spherically symmetric potential, the velocity is given by:

$$v(r) = r\frac{d\phi}{dt} = \left[\frac{GM(r)}{r}\right]^{1/2} \tag{2}$$

where M(r) is the mass enclosed. For motion in a galactic disk the analysis is more involved. For the ideal case, the density is zero except when z=0. The approach is to solve Laplace's equation $\nabla^2\phi = 0$ for $z \neq 0$ and apply boundary conditions at z=0. The results for z=0 is quoted below. For a full analysis, see for example sect 2.6 of *GalacticDynamics* by Binney and Tremaine.

$$\phi(R,0) = -2\pi G \int_0^\infty dk J_0(kR) \int_0^\infty dR R J_0(kR) \Sigma(R) \tag{3}$$

where Σ is the surface mass density of the disk and J_0 is a Bessel function. For an exponential disk, $\Sigma(R) = \Sigma_0 exp(-R/h_r)$, the integrals can be solved to get:

$$\phi(R,0) = -\pi G \Sigma_0 R \left[I_0\left(\frac{R}{2h_r}\right) K_1\left(\frac{R}{2h_r}\right) - I_1\left(\frac{R}{2h_r}\right) K_0\left(\frac{R}{2h_r}\right)\right] \tag{4}$$

where $I_0, K_0, J_1 and K_1$ are modified Bessel functions. For pure circular motion, we get the result:

$$v(R)^2 = \pi G \Sigma_0 \frac{R^2}{2h_r} \left[I_0\left(\frac{R}{2h_r}\right) K_0\left(\frac{R}{2h_r}\right) - I_1\left(\frac{R}{2h_r}\right) K_1\left(\frac{R}{2h_r}\right) \right] \tag{5}$$

The speed as a function of radius (rotation curve) is plotted below.

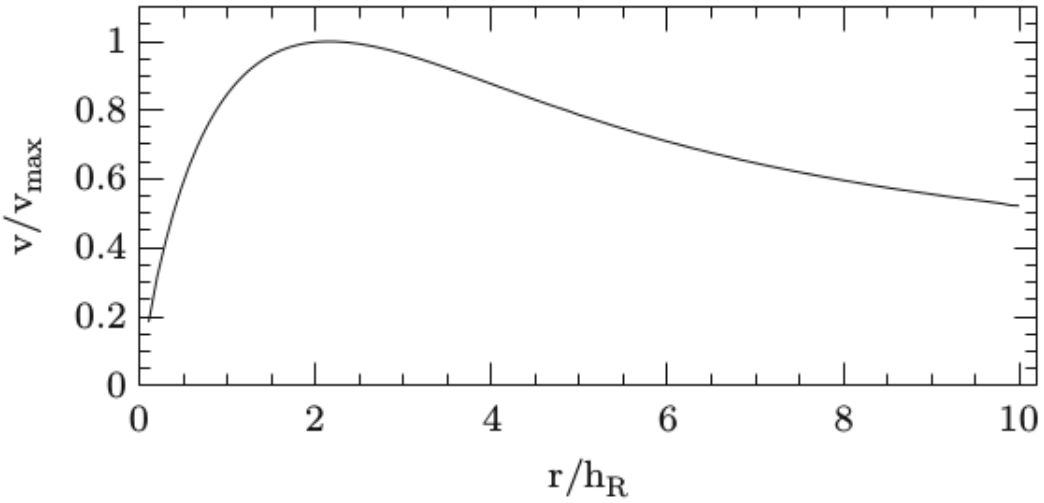

Figure 3: Rotation Curve

Real disks obviously aren't generally exponential. Still, as a general rule, we expect(ed) them to decrease in the outer regions, if they only contain the stars and gas we can see and measure. However, starting in the 70s, we started to measure rotation curves that did not match these predictions. Today, we instead see that some rotation curves fall only a little bit, some rise a bit, and some remain approximately flat. This was one of the first arguments for dark matter - a form of matter we do not observe directly, only by gravitational interaction. If there is a halo of dark matter in a galaxy, it can explain the observed rotational curves without messing with the theory of gravity. A simple test is to run simulation with and without DM particles/DM potential and see how rotation curves are affected.

4 Theory: Dynamical Friction

As mentioned in the above section, it is believed that DM orbits in halos in galaxies. However, in more recent years, there has been numerical evidence that there is significant substructure to these halos. In particular, when a small halo interacts with a larger halo, it can form a "subhalo" and orbit in the potential well of the larger halo. When the two halos are in this orbit, we expect there to be dynamical friction. This can be thought of as the subhalo moving through a particle mesh (the larger halo), accelerating the DM particles in it. As they get accelerated, they gain energy, and so the subhalo has to lose energy and slow down. Another point of view is to say that as an object moves through a particle mesh, clusters form behind it, causing the object two slow down. I prefer the first point of view, but regardless of stance the outcome is the same: we should see a loss of energy and angular momentum is the subhalo due to angular friction. This can be tested my simulating two halos orbiting inside each other, looking at the evolvement of the mechanical quantities of them.

5 Results: Simple Test (Solar System)

As an initial test of the code, it is applied to the solar system and all it's nine planets. The planets are (very physically correct) lined up on the negative x-axis, with initial velocities taken from NASA[2]. The system is then evolved for a number of years. An animation of the process can

[2]https://nssdc.gsfc.nasa.gov/planetary/factsheet/

be found in the repo. To check if the system behaves as we expect it to, we plot the energy and angular momentum of the system:

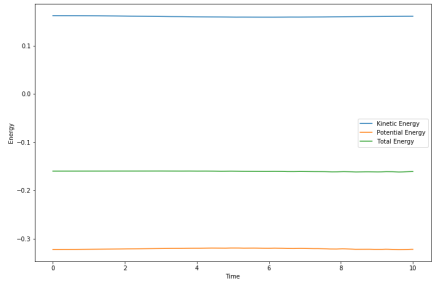

Figure 4: Total energy of Solar System

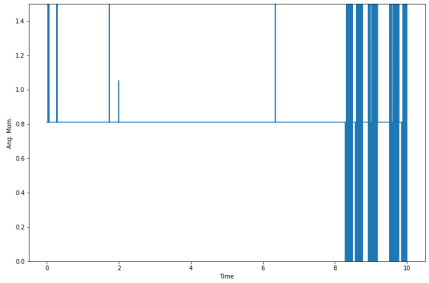

Figure 5: Total angular momentum of Solar System

As can be seen from the figure, the Energy in the system is perfectly conserved: thank you Verlet. The angular momentum has some strange spikes, but is otherwise a straight line (even at small scale). It's hard to say where these spikes come from, especially since we don't have them in the energy-plot, ruling out velocity-spikes. My best guess is something weird is happening in NumPy's cross product function, or in their norm-function, that might give some weird spikes - possible if the sun gets to close to zero. Disregarding the spikes however, it seems that angular momentum is indeed also conserved.

Another thing to test about the code is whether it is in fact of order nlog(n) as expected. To do so, we compute the forces on a number of particles - plotted below for 1000 time steps. Computational time vs number of points is plotted below.

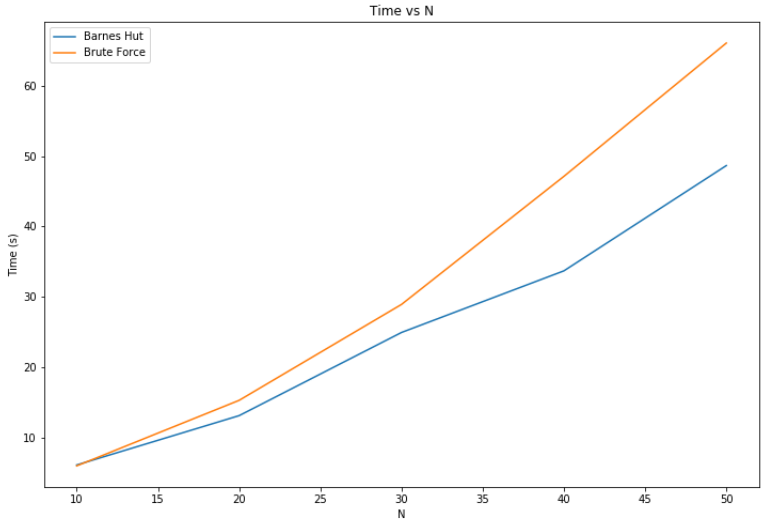

Figure 6: Barnes-Hut is O(nlog(n))

The Barnes-Hut algorithm is indeed faster than purely going brute force for a number of random points - and indeed it behaves approximately like nlog(n). The value of the efficiency of course increases greatly as the number of points increases.

6 Results: Galaxies

Sadly, I have not been able to produce any nice results for galaxies (rotation curves or subhalos). I have managed to make some decent 2D galaxy simulations, which can be seen in the repo. However, my computer is very old and subsequently very slow and it takes a very long time to do these runs. To tune the galaxy to the point of the attached animation took several days with several hour-long runs. In weeks of finals and grad-school application, I did not have the time to do many of these runs. I really wanted to explore some of these concepts - in 3D (the code is written), but time stopped me due to the poor computational power I have access to. I ask for consideration to be taken due to the lack of available resources to me.
I hope to be able to do some of these runs in the future, possible during winter break, and I will update with results if I get any, even though it is past the semester.

7 Conclusion

A successful Barnes-Hut algorithm was written - able to do N body simulations at order NlogN. The code was tested on a simple solar system case and the results were in agreement with expectations. Due to lack of processing power, little further progress was made, but it is a project that can be enhanced in the future. I want to generalize the code for optimal general use in Python, as well as getting some proper results for my own, though that might have to wait until I get a new computer.