Ph 22.4: Simulations of N-body systems using approximation techniques

-v20130102-

Introduction

In an earlier assignment you were asked to write a program to explore the behavior of a large set of N particles subject to their reciprocal gravitational attraction. In such a system, as opposed to one with *local* interactions, such as the Ising model or the *Game of Life*, we must (in principle) compute the effect exerted on each particle by every other particle; so the number of mathematical operations per integration timestep scales as N^2, which makes it impossible to follow the evolution of systems with large N. In this assignment, we will use the approximation technique perfected by Barnes and Hut [J. Barnes and P. Hut, *Nature* **324**, 446 (1986)], where the number of mathematical operations scales only as $N \log N$.

The Barnes-Hut tree algorithm

For a detailed description of the Barnes-Hut scheme you should read their *Nature* letter. Here, however, is an outline.

The Barnes-Hut algorithm relies on approximating a distant distribution of matter as a point particle with mass and position set equal to the the total mass of the distribution and to its center of mass. Consider for instance a spacecraft that is leaving the Earth, traveling through our Solar System, and out into the Galaxy. While the spacecraft is still close to the Earth, it would feel the gravitational effects of the non-homogeneous distribution of mass, such as mountains and tides; however, once it has traveled sufficiently far, to a very good approximation it would just feel a $1/r^2$ force directed toward the center of the Earth, and proportional to its total mass.

In the same way, while the spacecraft is traveling in the Solar System, it would feel the separate gravitational effects of the Sun and the planets; however, once it has left the Solar System and traveled far enough, it would just feel a $1/r^2$ pull directed toward the Solar-System baricenter, and proportional to its total mass.

So how do we go about implementing this approximation principle for a swarm of particles, confined (for simplicity) in the 2-D square $S \equiv [0,1] \times [0,1]$? We start by *recursively* subdividing the original square in sets of four subsquares, in such a way that each square contains at most one particle (see Fig. 1). Each square is known as a *quad*, and its subsquares as its *subquads*; the entire structure is known as a *quadtree*, and indeed it can be represented as a linked tree, where the top node is the root quad S, its children are the four subquads $S_{(00)} \equiv [0, 0.5] \times [0, 0.5]$, $S_{(01)} \equiv [0, 0.5] \times [0.5, 1]$, $S_{(10)} \equiv [0.5, 1] \times [0, 0.5]$, and $S_{(11)} \equiv [0.5, 1] \times [0.5, 1]$, and so on. The effect is to distribute the particles through the tree, in such a way that we have a bound on the distance between all the particles contained within a certain quad. For instance, the distance between each pair of particles contained within the quad $S_{(00)(00)} \equiv [0, 0.25] \times [0, 0.25]$ must be less than the quad diagonal, $\sqrt{2} \cdot 0.25$. In our notation, each set of parentheses denotes a subdivision, and the binary number within them denotes the choice of a subquad: $(00) \equiv$ left-bottom, $(01) \equiv$ left-top, $(10) \equiv$ right-bottom, and $(11) \equiv$ right-top.

For each quad (at every level of refinement), we then compute the total mass that it contains, and the position of its center of mass. To figure out the total force exerted by the quadtree on the particle P_n located at (x, y), we define the following *recursive procedure*. For a given quad S_m:

1. If S_m contains no particles (and therefore no subquads), it exerts no force on P_n.

2. If S_m contains exactly one particle (and therefore no subquads), we compute the exact two-particle force that this particle exerts on P_n.

3. If S_m contains more than one particle (and therefore four subquads, each of which might be further recursively subdivided), we compute the distance d between (x, y) and the center of mass of S_m, and we retrieve the length l of the side of S_m.

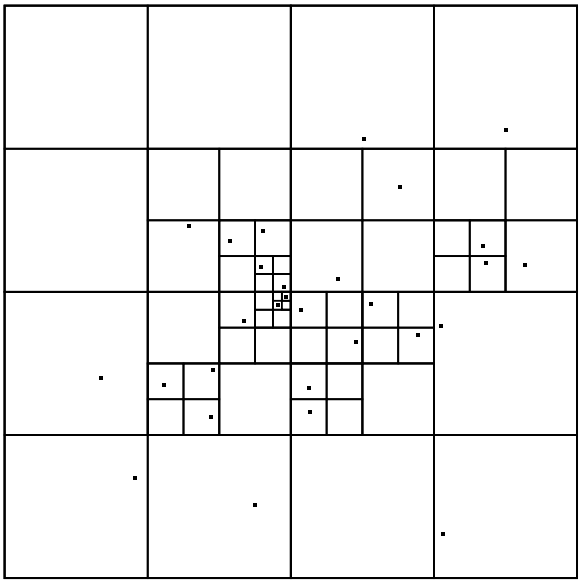

Figure 1: The recursive subdivision of the root *quad* in the Barnes-Hut algorithm. Each of the *subquads* at the maximum level of refinement can only contain one particle.

(a) If the ratio l/d is smaller than a fixed constant θ (known as *opening angle*), we conclude that S_m is far enough from P_n that the gravitational effect of all the particles contained in S_m can be approximated as a single force directed toward their center mass and proportional to their total mass, and we compute this force using the two-particle expression.

(b) If the ratio is larger, the force exerted by S_m on P_n is the sum of the forces obtained by applying this recursive procedure to the subquads $S_{m(00)}$, $S_{m(01)}$, $S_{m(10)}$, $S_{m(11)}$.

The result of applying this procedure to the root quad S is the total force on the particle P_n (see Fig. 2). The constant θ (of order 1) controls the level of the approximation: for $\theta = 0$, each of the particles exerts a separate force; for increasing θ, the particles are grouped into larger and larger subclusters that exert a force only through their center of mass. As a result, the number of force evaluations for each particle is only proportional to $\log N$. Can you convince yourself that this is true? If not, discuss it with your TA.

The Assignment

1. Write a python program to implement the Barnes-Hut algorithm as described above. Using graphical output to visualize the positions of the particles and the structure of the quadtree (as in Fig. 1) will help you debug your program.

   Use the symplectic, first-Euler Euler method to integrate the equations of motion, and use a *softened* two-particle force to reduce the effects of mass lumpiness:

$$F_{(2\,\mathrm{on}\,1)} = m_1 m_2 \frac{\hat{\Delta\mathbf{x}}}{|\Delta\mathbf{x}|^2 + a^2}, \tag{1}$$

   where $\Delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_2$, $\hat{\Delta\mathbf{x}}$ is the corresponding unit vector, $G = 1$, and $a^2 \lesssim 0.1$. Note that the quadtree will have to be recomputed at each timestep.

2. Set up a system of particles randomly distributed within a circle of size one with a set of random initial velocities, and run the simulation. Gauge the value of the masses, of the initial velocities, and

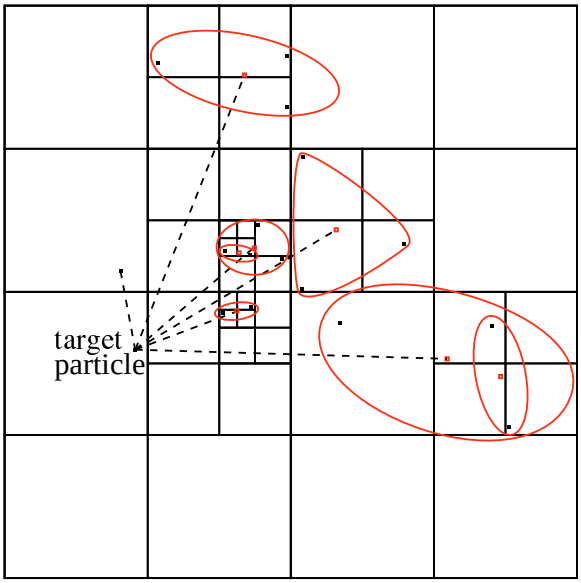

Figure 2: Computation of the total force on the left-bottom particle, according to the Barnes-Hut algorithm. The dashed lines represent the forces that must be summed to obtain the total force. Close particles contribute individually, but farther clusters contribute through their centers of mass (labeled by the red dot at the center of the closed curves).

of the timestep so that the cluster does not fly apart. Use graphical output to visualize the motion of the particles. Compare the qualitative evolution of the cluster for different θ's (remember that setting $\theta = 0$ yields the direct N^2 scheme). Compare also the runtimes, which should improve as you increase θ. What is the largest cluster that you can watch evolving (i.e., that completes a timestep within a reasonable time, not counting the time taken to display the cluster)?

3. Now do a quantitative comparison of a few evolutions that have the same parameters and initial conditions (initialize the random number generator with the same seed), but different θ's: choose a simple numerical *observable* (for example the total energy, the total angular momentum, or the radial density of particles at a few reference radii) and compare its evolution for different θ's. Devise a definition of *approximation error* based on your observable, and find the θ that yields a 1% error for a given total time of integration.

Implementation notes

This is assignment involves some challenging programming, so allow enough time and be patient in debugging. Some hints:

Data structures. This assignment is a good place for simple object-oriented techniques. In particular, the quads and the particles could be represented as classes in python. A typical quad class would contain variables corresponding to its position and size and to the total mass contained and the corresponding center of mass, plus pointers to its children or to the class for the single particle that can be contained in a quad. A typical particle class would contain mass, position, and velocity.

Functions/methods. The most important functions/methods that you will need to define involve:

1. At each timestep, you should create the quadtree by starting from the root quad, and inserting the particles one by one. You can do this by navigating down through the tree structure and choosing branches according to the position of the particle (say, P_n), until you find a quad

without children. If the quad already contains a particle (say, P_m), you need to create children for the quad, move P_m to its appropriate child, and then try to fit P_n within its appropriate child (if both P_n and P_m go to the same child, this will also have to be subdivided). All of this is best done by a recursive procedure/method.

2. After each timestep, you should destroy the quadtree, so that memory can be reused.
3. Another recursive procedure/method, to call after having created the quadtree, could prune all the quads that have neither children nor a particle, and compute the total mass and center of mass for the ones that do. A bottom-top method works best: if a quad has children, its total mass is the sum of the masses of its children, and its center of mass is the weighted average of the centers of mass of its children; if a quad has a particle, its total mass and center of mass are simply the mass and position of the particle.
4. Finally, the last important procedure/method would compute the force on a particle as described in the main assignment text.

Tricks. First, to allow for particles that wander off the root quad S, simply recreate the root quad at each timestep with size determined by the farthest particle. Second, be sure to avoid computing the force of a particle on itself!

Good luck!