Computational HW4

Group 1: Julian Marohnic, Harrison Agrusa, Ramsey Karim

November 2017

Problem 1

We first verify our code by solving the two-body problem for equal mass particles, to which we have an analytic solution. We initiate conditions as listed in the prompt: $m_1 = m_2 = 1$, $G = 1$, and no softening, such that $\epsilon = 0$. The integrator runs for 100 orbits in four cases:

- Eccentricity $e = 0.5$ using Leapfrog
- Eccentricity $e = 0.5$ using Runge-Kutta 4^{th} order
- Eccentricity $e = 0.9$ using Leapfrog
- Eccentricity $e = 0.9$ using Runge-Kutta 4^{th} order

Initial conditions for position and velocity are set by the eccentricities and are as stated in the prompt.

The N-body code used for this test runs on a Barnes-Hut tree framework and is written in C, with several wrapper functions controlling it written in Python. For the initial two-body test, we set the critical angle to 0 so that the code runs as a particle-particle simulator, although it should behave the same in either case. It offers a second-order leapfrog integrator as well as a fourth-order Runge-Kutta integrator.

Results are as expected. We include plots of x-y position for each particle, phase diagrams giving r vs v_r, and fractional change in total energy.

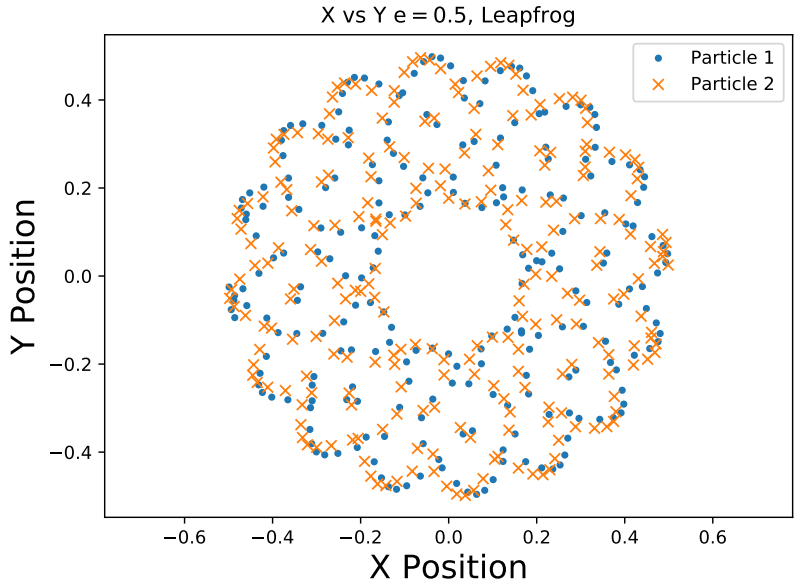

Figure 1: Position graph, leapfrog integrator used for Eccentricity $e = 0.5$

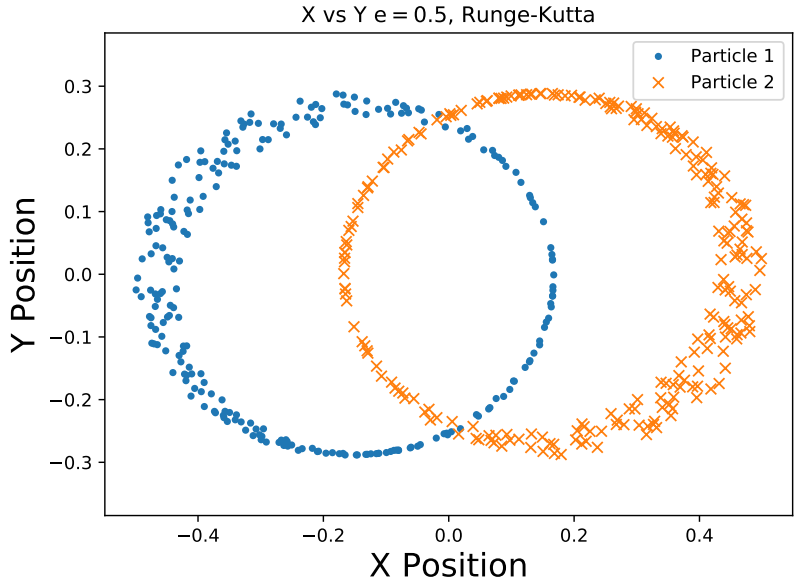

Figure 2: Position graph, Runge-Kutta 4 integrator used for Eccentricity $e = 0.5$

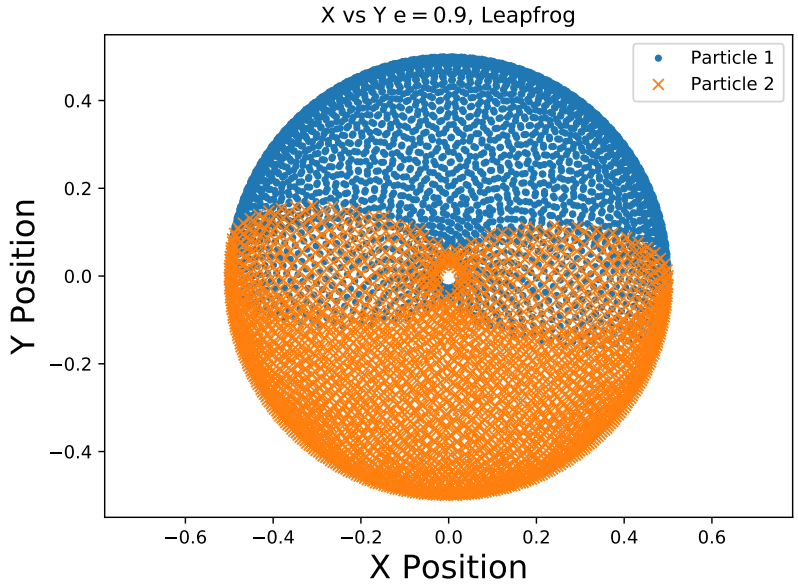

Figure 3: Position graph, leapfrog integrator used for Eccentricity $e = 0.9$

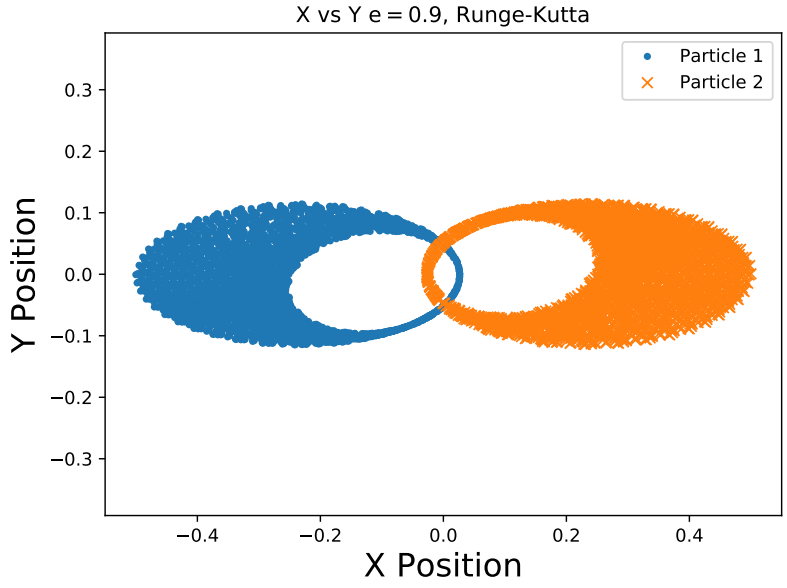

Figure 4: Position graph, Runge-Kutta 4 integrator used for Eccentricity $e = 0.9$

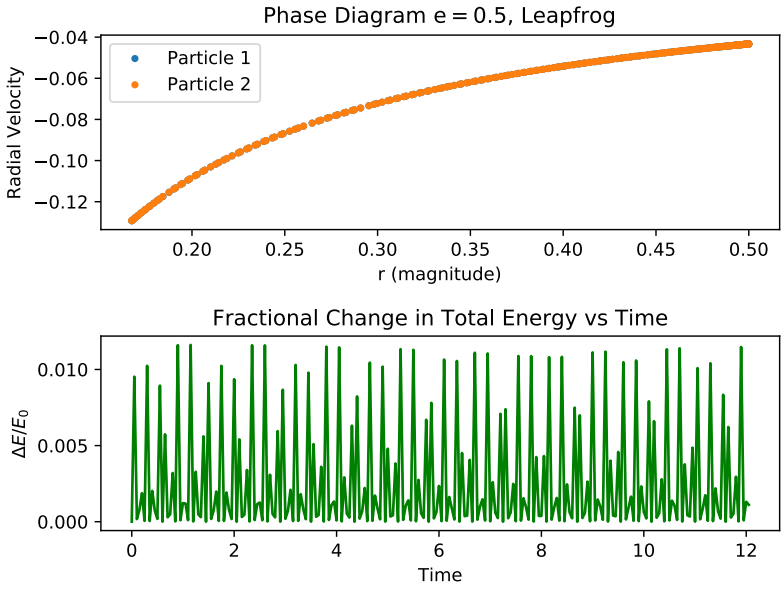

Figure 5: Phase and energy diagrams, leapfrog integrator used for Eccentricity $e = 0.5$

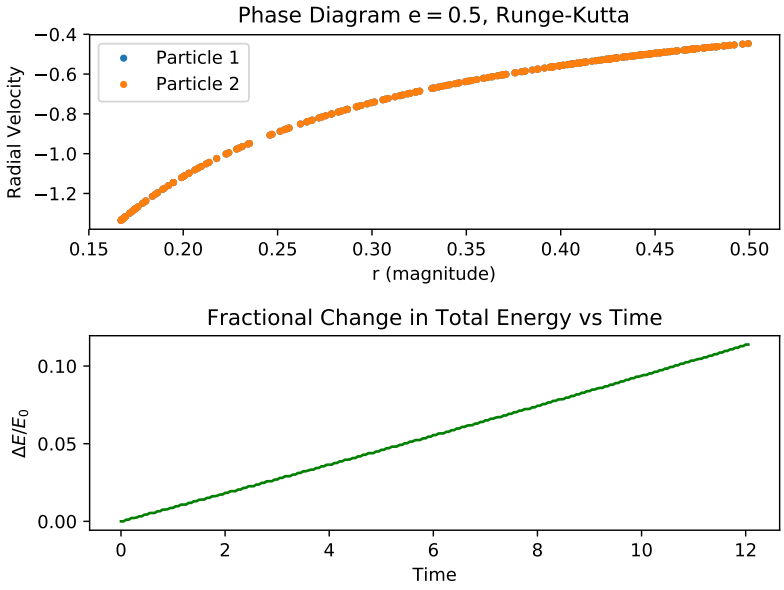

Figure 6: Phase and energy diagrams, Runge-Kutta 4 integrator used for Eccentricity $e = 0.5$

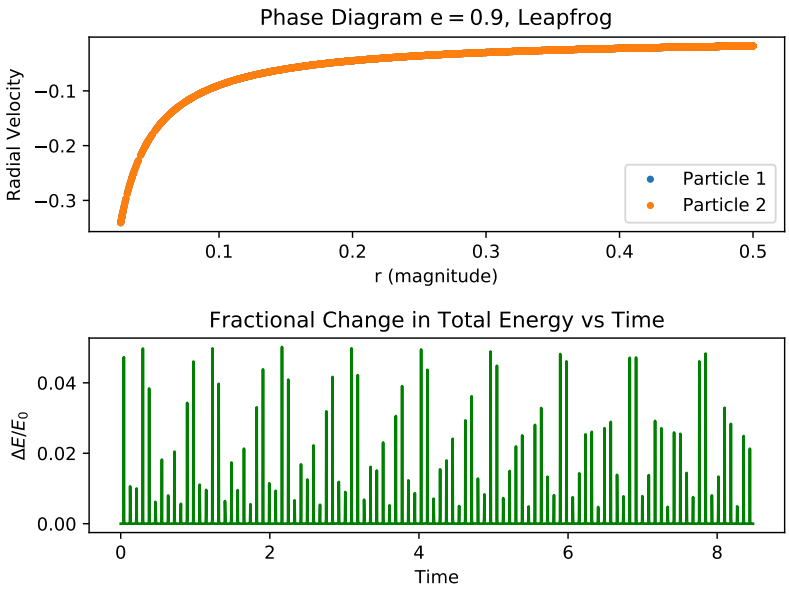

Figure 7: Phase and energy diagrams, leapfrog integrator used for Eccentricity $e = 0.9$

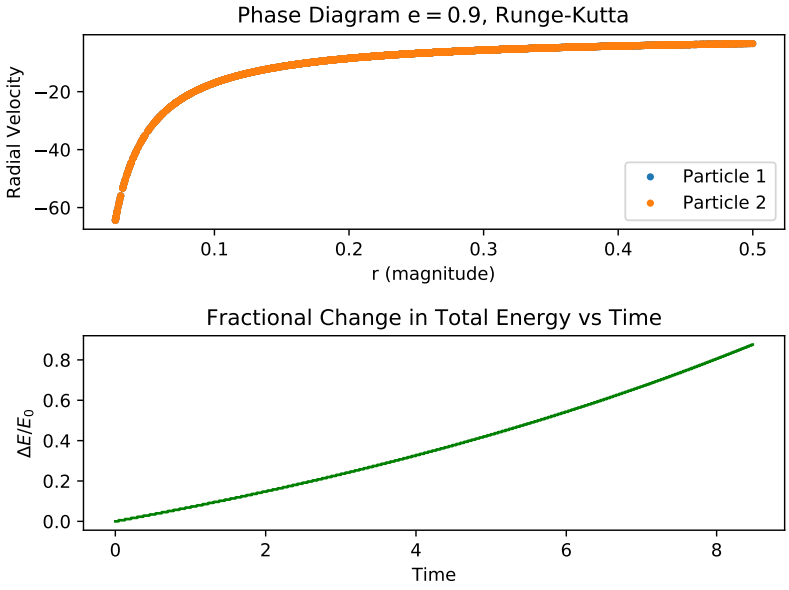

Figure 8: Phase and energy diagrams, Runge-Kutta 4 integrator used for Eccentricity $e = 0.9$

Problem 2

We generate initial conditions that correspond to a rotating sphere of particles with radius 250. We generate 1000 particles with masses randomly distributed between 1 x 10^4 and 1 x 10^6. The critical angle used for the tree is 0.1. We use a softening parameter of 1 in order to allow for the possibility of "clumps" forming in our non-collisional simulation. The initial density of the system is given by $\rho = M_{avg}N/V = 7.72$ and we set $G = 1$, so we can estimate the dynamical time of the system as follows:

$$\tau_D \simeq \frac{3}{\sqrt{G\rho}} \approx 1.$$

The rule of thumb $\delta t \sim \tau_D/30$ suggests a time step of about 0.03. We choose 0.01 in order to see more detail near the beginning of the simulation, where the system evolves very quickly. We save the system state once every two steps. We allow the system to evolve for 1000 time steps using a second order leapfrog integrator, which takes 39.1 seconds to run. This gives a simulation that spans about 30 dynamical times- this should be sufficient to allow the system to settle into a steady state. When we run the simulation, we see that the sphere very rapidly collapses into a disk (after $0.3\tau_D$) and then expands again into a cloud of orbiting particles (by $3.5\tau_D$) which is much larger than the original sphere. Many of the particles are ejected entirely. An animation of the evolution of this system is included, titled "sphere.mp4". To track changes in the energy of the system over time, we sample the total energy of every particle at five time steps equally spaced throughout the run. Unfortunately, global energy conservation is violated, even though we use a leapfrog integrator. This is because of the relatively large softening parameter that we use and because we are using a tree code instead of a particle-particle summation.

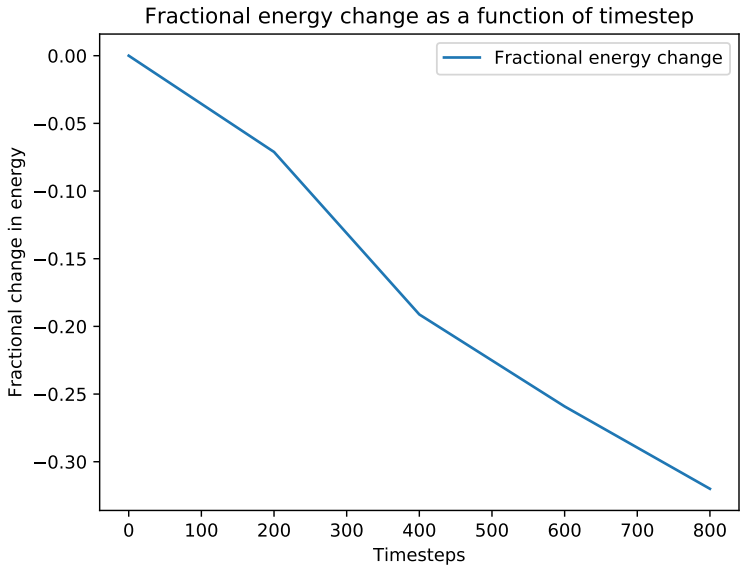

Figure 9: Plot of kinetic energy over time

Problem Fun

To test the performance of the tree code, we looked at time to complete 1000 time steps as a function of the number of particles in the system. Figure 10 shows the computation time as a function of the number of particles. A theta of 0.0 corresponds to doing the full particle-particle calculation which scales as $\mathcal{O}(N^2)$ as expected. Using larger values of theta greatly improves the speed of the calculation to $\mathcal{O}(N \log N)$, however there will be a loss in accuracy. Theta = 1.0 is extremely fast, only taking ~ 12 seconds to do 2000 particles, however there is a significant loss in accuracy here. In most scenarios a theta of around 0.2 is ideal, being twice as fast as the particle-particle method for 2000 particles yet still retaining accuracy.

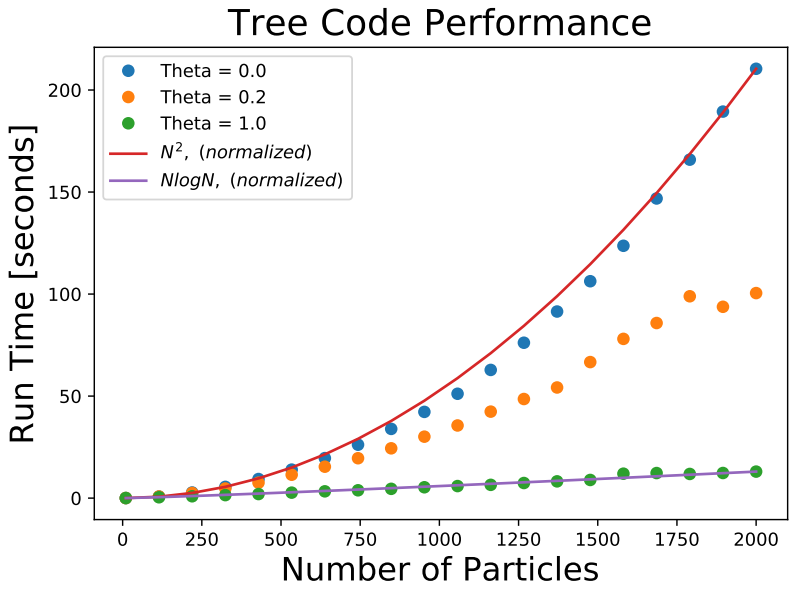

Figure 10: Plot of run time vs number of particles, using the 2nd Order Leapfrog Scheme.