Name: _____ , _____

PHYSICS 401 : SPRING SEMESTER 2018

Project #2: Keplerian Orbits and Java Objects

Please note: Answer questions, do not just hand in graphs without stating which part of the assignment you are doing.

Reference Reading: Sections 2.3; Appendix 3A; 5.1-5.3

Downloads: Download the three files KeplerFileApp.java, MovingObject.java, and KeplerMoveApp.java from my website.

1. Compile and run KeplerFileApp, plot the output "traject.txt", and check that everything is working.

a) Examine the file KeplerFileApp.java. Note that the initial conditions are $\mathbf{r}_0 = (10, 0)$, $\mathbf{v}_0 = (0, 1/10)$. What are the semi-major axis a, the value of $\mathcal{P}$, the eccentricity e and the period T (in our reduced units)?

b) Knowing T, set $dt = T/1000$ and the loop number $n = 5000$ so that the program outputs 5 complete orbits. Hand in the resulting graph (never mind that dt is too large for an accurate orbit, the point here is just to see what kind of errors will result). What's wrong with the orbit? (In gnuplot, try all three of the following and see which one does the best job: `plot "traject.txt"`, `plot "traject.txt" with lines` or `plot "traject.txt" with linespoints`.)

c) Examine how the output file "traject.txt" is created, emulate and create another output file "energy.txt", and output into it, the energy error $(E - E_0)/E_0$ as a function of t/T. The energy is given by $E = \frac{1}{2}(v_x^2 + v_y^2) - 1/\sqrt{x^2 + y^2}$ and E_0 is the initial energy.

2. Compile MovingObject.java first, then compile KeplerMoveApp.java, and then run KeplerMoveApp. Check that the latter produces the same initial output as KeplerFileApp.

a) Carefully retracing your steps in 1c), create and output the energy error into "energy.txt". Note that the energy is already defined in the object MovingObject, you just need to retrieve it. You don't have to calculate E in the main program.

b) Examine how the subroutine, or the method `velocitystep(double cof)` is constructed in MovingObject.java and how `accel` is called within it, create another method named `sym1astep(double cof)` that simply calls `velocitystep(double cof)` then `positionstep(double cof)` within it. Now replace the two calls of `planet.velocitystep(1.0)` and `planet.positionstep(1.0)` in KeplerMoveApp.java by just a single call to `planet.sym1astep(1.0)`. Check that the energy error produced by `sym1astep` is the same as that in 2a). Save the energy error file as "energy1a.txt".

c) Create another method `sym1bstep` with the order of calling `velocitystep` and `positionstep` *reversed*. Now call `sym1bstep` in the main program. How does its energy as compared to `sym1astep`? Save the energy file as "energy1b.txt"

d) Now create the method `sym2astep(double cof)` which calls `sym1astep(0.5*cof)` followed by `sym1bstep(0.5*cof)`. In the main program call it as `planet.sym2astep(1.0)`. Save its energy file as "energy2a.txt".

e) What is `sym2astep(double cof)` directly in terms of `velocitystep(double cof)` and `positionstep(double cof)`? What is the name of this algorithm?

f) Plot all three energy files in the same graph and hand this graph in. To see the shape of the energy error, plot only from $t/T = 0.45$ to $t/T = 0.55$ (use gnuplot command:`set xrange[0.45:0.55]`. (To remove the range restriction and can get back to the auto mode: type `set auto`.)

3. Create the method `RK2step()` with no parameters. Now in the main program, create another moving object called `asteroid`. Move `asteroid` with `RK2step`, move `planet` with `sym2astep` for five periods, output both trajectories into "traject.txt" having four columns of numbers, and both energy errors into "energy.txt" with three colums of numbers. In gnuplot, plot both trajectories by `plot "traject.txt" using 1:2 with linespoints` followed by `replot "traject.txt" using 3:4 with linespoints`. Do the same for the energy errors. What is the most notable difference in the two energy error curves? Hand in both the trajectory and the energy error graph.