PROBLEM SET #1, ASTR 451/551

Virial Theorem Coding

30 pts; due Monday, September 26, 2016

In this problem you will construct a code to track the movements of a small cluster of five stars. The goal will be to follow how well the virial theorem is satisfied, and to gain some insight into the moment of inertia term in the virial equation. We will use distance units in AU, and time in units of years throughout this problem.

(A) [1 pt] You will have to determine a timestep to move your code along. Calculate the natural gravitational timescale t_{dyn} associated with a cluster of stars where the cluster has a radius r and contains n stars, each of mass M. When you actually need to find t_{dyn} for your cluster you are going to have to decide what 'r' means for that case. We will all likely define that slightly differently, but the results will all have the same answer in absolute time units if the codes work.

(B) [1 pt] Rather than begin with the full problem, let's get a code to work on the orbits of two solar mass stars. Take the stars to have positions (0,10,0) and (0,-10,0), with velocities ($-V_\circ$,0,0) and ($V_\circ$,0,0) respectively. Hence, the motion of the components of this binary ought to be solely in the XY plane. For these units, derive what $V_\circ$ needs to be in order for the stars to travel in exactly circular orbits.

(C) [6 pts] Write a code that uses the gravitational force to move the stars around. Experiment with the timestep to see what fraction of the natural timescale it needs to be in order to keep the stars in circular orbits around their center of mass. Run the code for at least 10 orbits to make sure the code is stable. Show a plot of the XY positions of both stars. Now decrease the velocity and show the XY plot. Does the code behave as you expect?

(D) [4 pts] Once you are confident that the code works with two stars you are ready to put in a cluster. With velocities in AU/yr and distances in AU, here are the initial positions of the five stars. Each star is 1 solar mass. The origin of the coordinate system is set to be the center of mass of the system, and the cluster as a whole is stationary with respect to this origin.

Star	X (AU)	Y (AU)	Z (AU)	V_X(AU/yr)	V_Y(AU/yr)	V_Z(AU/yr)
1	-1	9	-1	-0.7	0.1	0
2	9	-1	-1	0.3	1.1	0
3	-11	-11	4	0.8	-0.4	0
4	4	-1	-6	-0.7	0.1	0
5	-1	4	4	0.3	-0.9	0

N-body codes can be tricky beasts. The trouble is that the timesteps cannot stay fixed. Upon close approaches the timesteps must become smaller to actually track the orbits. However, doing so all the time is a waste and will bog down your code. You will have to decide upon some scheme to choose a reasonable timestep. Describe and justify your scheme.

(E) [4 pts] Run your code for three dynamical times and plot the positions of the stars. Be sure that the trajectories look right - no sudden jumps, gravity pulls stars together, etc. For plotting results we should all use the same conventions. Plot the positions of the stars in two side-by-side graphs: the one on the left a plot of Y vs. X and the one on the right a plot of Z vs. X. At the beginning of the time interval (in this case $\tau =$ 0) plot the symbol of the star, and follow the trajectory of the star with a colored line.

Let's employ the following conventions:

STAR	Color	Shape
1	black	open circle
2	red	open square
3	dark blue	filled triangle
4	purple	open triangle
5	dark red	filled square

Note that in order for two stars to actually be near to one-another they must be close together in both the XY plot and the XZ plot.

(F) [4 pts] Now do the same plots, but for the following approximate intervals:

(a) 3 $t_{dyn} <$ t $>$ 10 t_{dyn}
(b) 10 $t_{dyn} <$ t $>$ 20 t_{dyn}
(c) 20 $t_{dyn} <$ t $>$ 50 t_{dyn}
(d) 50 $t_{dyn} <$ t $>$ 100 t_{dyn}
(e) 100 $t_{dyn} <$ t $>$ 120 t_{dyn}
(f) 120 $t_{dyn} <$ t $>$ 150 t_{dyn}
(g) t $>$ 150 t_{dyn}

These intervals are *approximate* - **you** get to choose the intervals. In any case, your t_{dyn} may be a little difrerent from mine depending on how you defined it. If something exciting is happening somewhere and a lot is going on, say, between $t_{dyn} = 29$ and 33, then create a plot, zoomed in on the interesting region, for that time interval. If a plot gets too jumbled, you'll need to break it up. I will evaluate these on how easy the plots allow you to tell the story. Spend time to make these clear. Zoom in and out as you need. Something like 10 plots ought to suffice. Label the paths to make them easier to follow.

(G) [4 pts] Write a short narrative for each panel to describe what is going on. If something flaky is happening you do not believe and you cannot solve it, you need to note it. If you believe the results, say so. You will need to use judgment here.

(H) [2 pts] Plot the ratio of the total kinetic energy to potential energy for the group of stars as a function of time. How close does it match the simple virial theorem result?

(I) [2 pts] Plot the time derivative of the moment of interia of the system, dI/dt as a function of time. Only zoom in on the part of the plot that is useful; for example, if dI/dt just increases linearly to infinity or flatlines at zero that won't have a term in the virial so there is no reason to plot that part. Can you relate what you see in this graph to your results in section (G)? Try to think about your results here.

(J) [2 pts] Print out your final code and attach it with your plots and narrative.

Some notes on coding: It does not matter what language you use. Popular ones are python, IDL, R, MatLab, Mathematica, C, and fortran. Of these, only C and fortran are likely to run quickly. For what it is worth, I coded in fortran and plotted things with R. You may not use an existing n-body code, assuming you could ever find one that is appropriate. Instead, you should code this by (a) choosing a timestep, (b) calculating the forces between each of the stars, and (c) changing the stellar velocities and positions according to the forces. For plotting, application languages like python, R, IDL, etc. are probably easiest.

There is no need, and it is probably a bad idea, to print out the (X,Y,Z,Vx,Vy,Vz) each step. That will lead to a huge output file. Also, keeping all of those values for each timestep in a matrix is going to chew up a lot of memory and make for a very slow code, difficult to use and hard to debug. Instead, try doing whatever calculations you need on short timescales, but only write out results every once in a while. And generally once you are on, say, timestep 1000, you no longer need to remember where every star was at timestep 893. If you subtract two numbers that are both large, be careful that the calculation has enough precision so that round off errors do not ruin everything.

A good way to test your code (e.g. for part (G)) is to change the timesteps. If the code is working, the general character of the results should not change with the choice of timestep. Often a code gives reproducible results at early times, and diverges more at later times. What might cause this behavior do you suppose?