Final Honour School of Mathematical and Theoretical Physics Part C and MSc Mathematical and Theoretical Physics

Galactic and Planetary Dynamics

Please typeset your solutions using a font size of at least 10 pt for the main body of your report.

The periodic cube provides a particularly simple (if unrealistic) arena for studying stellar dynamics. Many of its properties have been worked out by Weinberg (1994, ApJ 410, 543) and Magorrian (2021, MNRAS 507, 4840). In this miniproject you will use it to investigate one aspect of relaxation in stellar systems. You should read the preceding papers carefully before starting work on the project.

(i) The listing overleaf presents a simple N-body code that follows the evolution of N stars in the periodic cube, each having mass $m = M/N$. What assumptions does it make about the initial phase-space distribution of stars? What does it assume about their interaction potential? Explain why the value of the timestep chosen for integrating orbits is reasonable for this system.

(ii) The program outputs the evolution of the mean-square change in stars' velocities,

$$\Delta v^2(t) \equiv \frac{1}{N}\sum_{n=1}^{N}(\mathbf{v}_n(t) - \mathbf{v}_n(0))^2. \tag{1}$$

By running it for different cube masses M and plotting the results, show – at least for $10 \lesssim t \lesssim 100$ and for masses $M \lesssim 0.1(2\pi)^2$, where M and t are expressed in the same units as used in the code – that $\Delta v^2(t)$ increases approximately linearly with time,

$$\Delta v^2(t) \simeq \text{constant} \times \frac{M^2}{N} t. \tag{2}$$

Explain this scaling with M and N qualitatively. Why does it break down for larger values of M?

(iii) Now consider the situation in which an ensemble of cubes are prepared, each having total mass M generated by N equal-mass stars drawn from a common distribution function $F(\mathbf{v})$. Stating clearly any assumptions that you make, but making as few as you can about $F(\mathbf{v})$, derive an expression for the ensemble-averaged $\Delta v^2(t)$ in terms of N, M and $F(\mathbf{v})$. Compare the evolution predicted from your expression against your measurements from part (ii) above.

(iv) In setting up the simulations it is assumed that the joint N-particle distribution function is of the form $F_N(\mathbf{x}_1, \mathbf{v}_1, ..., \mathbf{x}_N, \mathbf{v}_N) = \prod_{n=1}^{N} F(\mathbf{v}_n)$. Is this relation maintained as the system evolves? If not, outline two ways in which deviations from it might be quantified from the simulations. [You are not expected to implement these!]

```
#!/usr/bin/env python3

# Simple N-body code for the periodic cube.  You can download this script from
# https://www-thphys.physics.ox.ac.uk/people/JohnMagorrian/cm21/cubeNbody.py
# To run it you will need python3 and numpy: see https://numpy.org/install/

import numpy as np
pi = np.pi

class PotentialCalculator:
    def __init__(self, nmax, nx=64):
        self.nx = nx
        self.dx = 2*pi/nx
        self.fftkernel = np.zeros((nx,nx,nx))
        for n1 in range(0,nx):
            n1sgn = n1 if n1<nx/2 else n1-nx
            for n2 in range(0,nx):
                n2sgn = n2 if n2<nx/2 else n2-nx
                for n3 in range(0,nx):
                    n3sgn = n3 if n3<nx/2 else n3-nx
                    nsq = n1sgn**2 + n2sgn**2 + n3sgn**2
                    if nsq>0 and nsq<nmax**2+0.1:
                        self.fftkernel[n3,n2,n1] = -4*pi/nsq;
        self.mesh = np.zeros((nx,nx,nx),dtype=float)

    def ptles2density(self, xs, ms):
        ndxs = (xs/self.dx).astype(int)
        for (ndx,mfac) in zip(ndxs,ms/(2*pi)**3):
            self.mesh[ndx[0],ndx[1],ndx[2]] += mfac

    def density2pot(self):
        fftmesh = np.fft.fftn(self.mesh)
        fftmesh *= self.fftkernel
        self.mesh = np.fft.ifftn(fftmesh).real*self.nx**3

    def pot2accels(self, xs):
        mesh, nx, dx = self.mesh, self.nx, self.dx
        mesh = self.mesh
        ndx = (xs/dx).astype(int).T
        return -np.array([
            mesh[ (ndx[0]+1)%nx, ndx[1], ndx[2] ] - mesh[ (nx+ndx[0]-1)%nx, ndx[1], ndx[2] ],
            mesh[ ndx[0], (ndx[1]+1)%nx, ndx[2] ] - mesh[ ndx[0], (nx+ndx[1]-1)%nx, ndx[2] ],
            mesh[ ndx[0], ndx[1], (ndx[2]+1)%nx ] - mesh[ ndx[0], ndx[1], (nx+ndx[2]-1)%nx] ]).T/(2*dx)

    def accels(self, xs, ms):
        """Given a distribution of particles at locations 'xs' having masses 'ms',
        calculate the corresponding potential and return the acceleration experienced by each particle."""
        self.mesh *= 0
        self.ptles2density(xs,ms)
        self.density2pot()
        return self.pot2accels(xs)

if __name__ == "__main__":
    MJeans = 2*pi**2
    nmax = 4
    N = 100000
    nstep, dt = 5000, 0.02
    M = 0.01*MJeans

    # Initial conditions.  Positions xs, velocities vs, masses ms.
    dv= np.array([ 1.0, 1.0, 1.0 ])
    xs = np.random.uniform(0,2*pi,size=3*N).reshape(N,3)
    vs = dv*np.random.normal(loc=0,scale=1,size=3*N).reshape(N,3)
    ms = np.ones(N)*M/N

    vs0 = vs*1.0 # make a copy of initial velocities
    mesh = PotentialCalculator(nmax)
    for istep in range(nstep):
        t = istep*dt
        xs += 0.5*dt*vs
        xs = np.fmod(4*pi+xs,2*pi) # map to [0,2*pi)
        vs += dt*mesh.accels(xs,ms)
        xs += 0.5*dt*vs
        print(t,np.std(vs-vs0)**2)
```