Indian Institute of Science Education and Research, Mohali

Orbital dynamics in barred galaxies and the role of chaos

Term Project

Course Code - IDC402

Instructor - Dr. Anil Kumar Dasanna

Name	-	Rajnil Mukherjee
Registration Number	-	MS21213
Department	-	Department of Physical Sciences
Date of Submission	-	May 4, 2025

1 Introduction

N-body simulations play a pivotal role in understanding the dynamical processes driving galactic ecosystems and their subsequent evolution. For instance, studying the nature of orbits in galaxies constitutes a very important issue, because they play a central role in constructing self-consistent models of galaxies. In order to study and understand the structure and dynamics of a galaxy it is necessary to identify the type of dynamics characterising the motion of stars (regular or chaotic) and estimate the percentage of each type in the phase space of the galaxy. The aim of this report is to put forward a brief overview and present some of the results obtained from an attempt at such a study.

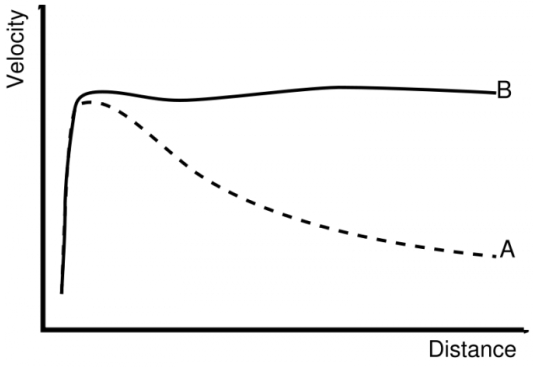
2 The Model

We have tried to model the behaviour of a system of ≈ 1000 particles under the influence of gravity in the hopes of reproducing galaxy-like behaviour. A galaxy like the Milky Way is composed of matter distributed in a disk $\approx 15kpc$ in radius, and a dark matter halo $\approx 30kpc$ in radius. In reality the dark matter halo is extremely important, but in this simulation we will neglect contribution due to dark matter halo and perform the simulation on a 2D plane for ease of plotting. We would like to generate a initial system with similar conditions. This motivates us to work in a system of units where length is in units of kpc, mass is in units of $GM_{\odot}$s (giga solar masses), and time is in units $10Myr$ (10 million years). In this system, G=0.449. Each of the 1000 particles will have mass m = 5 and the simulation box will have side of half-length L = 150kpc.
Matter in the our fiducial galactic disk is distributed according to an exponential profile, where probability density P(R) of finding a star at radius R is given by Equation 2.1 (Nesti 2013 [1]), normalized over $R \in [0, \infty)$.

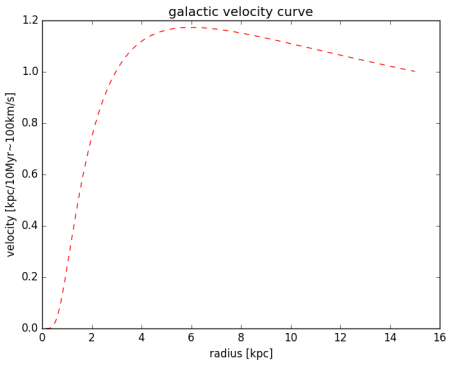
$$P(R) = (1/h_R) \exp(-R/h_R) \tag{2.1}$$

We can draw points $r \in [0, \infty)$ distributed according to Equation 2.1 by Monte Carlo importance sampling. We draw numbers $x \in [0, 1)$ uniformly and calculate r using Equation 1. We can give the point an arbitrary angular position by uniformly drawing from $\theta \in [0, 2\pi)$. This gives us a distribution of particles resembling the Milky Way.

$$r = -h_R \log(1 - x) \tag{2.2}$$
$$\hat{r} = \cos\theta\hat{x} + \sin\theta\hat{y} \tag{2.3}$$



(a) Rotation curve for the milky where B accounts for dark matter and A does not



(b) Velocity function used to give speeds to particles at various R.

Velocity curve in the Milky Way is well known from observational measurements. Theoretical predictions of velocity curve depend on dark matter. However, we proceed with a naive estimate of velocity at radius R is $v(R) \sim \sqrt{GM_{gal}/R}$, orbital speed assuming concentration of galactic mass in the core (M_{gal} is mass of galaxy). This matches behaviour for large R. We've implemented a boltzmann factor in Equation 2.4 that generates non-singular core behaviour.

$$v(R) = \sqrt{GM_{gal}/R}\exp\left(-h_R/R\right) \tag{2.4}$$

$$\hat{v} = -\sin\theta\hat{x} + \cos\theta\hat{y} \tag{1}$$

To account for the bar, we have introduced a rudimentary Ferrer's bar which has a following a mass density given by,

$$\rho = \rho_c\left(1 - m^2\right) \tag{2}$$

only when $m < 1$, and 0 otherwise, where $m = \frac{x^2}{a^2} + \frac{y^2}{b^2}$ and a = 6,b = 0.5, and $\rho_c = \frac{105GM_B}{32\pi ab}$, where M_B is the mass of the bar.

The system evolves under the force of Gravity. Between every pair of particles, the gravitational force from particle 2 on particle 1 is given by equation 2.6.

$$\vec{F} = \frac{Gm_1m_2}{|\overrightarrow{r_2} - \overrightarrow{r_1}|^3}\left(\overrightarrow{r_2} - \overrightarrow{r_1}\right) \tag{2.6}$$

3 Computational Method

Below we describe basic theory of methods used as well as results of implementation. In the next section (Section 4), we will discuss detailed structure of Python implementation of these methods.

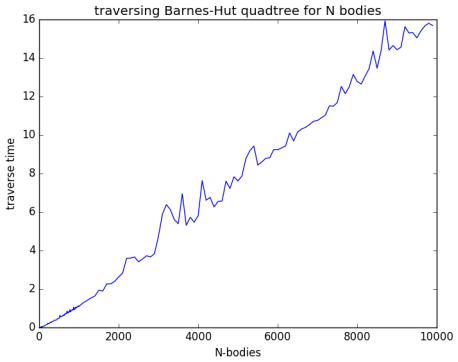
3.1 Computation of Force

Computation of force is the most computationally expensive part of the simulation. For each body, we must compute force from every other body. This makes $N(N-1)$ computations in total. Using this brute force method, we get complexity order $O\left(N^2\right)$. However this is slow and not all of it is necessary.

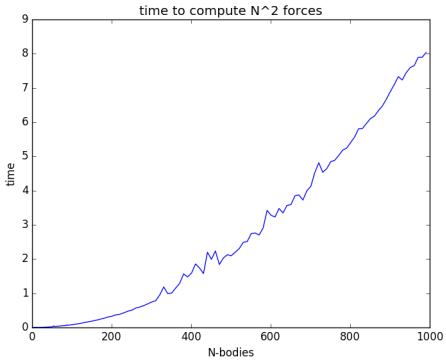
We begin by realizing that Poissonian potentials (like gravity) about a mass are very complex within some radius, but relatively smooth at large radii. This motivates us to try to approximate force due to masses at large distance with some kind of center of mass representative force rather than compute each force individually. This is the basis of the *Barnes-Hut method*, an order O(N log N) method of computing forces in a system of N particles.

The basic idea is as follows. Particles are sorted into a quad tree, which is a recursive structure that divides a square recursively into smaller and smaller squares. Each square is a structure called a node, and each quadrant of the square is a child node. We keep dividing each quadrant of a square into subquadrants until every subquadrant has one and or zero particle in it. One that is achieved for any square, we stop. These squares are called leaf nodes, while each of the preceding squares are called branch nodes. Then all branch nodes have more than one particle in it, and thus must have child nodes. The computational order is O(NlogN), which looks linear on long time scales. If this is not convincing, at least it is obviously not quadratic.

Every node of the tree is the same structure as its parent, so recursion is ideal for the construction of trees as well as accessing nodes. Python recursion however, is not well optimized so it is very slow. This has the disadvantage of being slower than comparable O(NlogN) methods which rely on



(a) Time required to evaluate forces on all particles from root node of Barnes-Hut tree as function of number of bodies



(b) Time required to evaluate forces on all particles using brute force method as function of number of bodies

C programs like Particle Mesh which relies on FFT. In fact, it is around 25 times slower!
Each branch node of the tree stores only its child nodes (which are also trees in their own right), and a representative "aggregate body". The aggregate body is an object with mass equal to the sum of all masses contained in the quadrant represented by the node, and position equal to their centre of mass. It tracks this information easily because whenever a body is inserted into the node, it updates mass and position before inserting the body into the relevant child node. If the node is a leaf node, then it contains no children and its aggregate body is simply the one body it contains. Some children have no bodies and we should call them something else, like stub nodes.

$$\frac{s}{d} < \theta_{BH} \tag{3.1}$$

To compute force on any given particle, we descend from the root node. At any node, if the aggregate body satisfies the inequality in Equation 3.1, then gravitational force from that node is added to force on that particle. Here s is size of node quadrant (side length), d is distance between particle and aggregate body, and θ_{BH} is a computational resolution parameter, where smaller theta increases resolution. We chose $\theta_{BH} = 1$ which means that resolution is fine enough when node body is at least one node length away. If the inequality is not satisfied, then we compute the gravitational force from the node on the body as the sum of the gravitational force from its child nodes. This operation completes in O(NlogN) time compared with $O\left(N^2\right)$ brute force algorithm.

3.2 Softening Length

A major problem encountered by all inverse square forces is that on finite timescales, should two particles take a step that takes them to within minuscule distance from each other, their force will be tremendous, maybe propelling them to infinity within a single time step. Energy conservation could go out the window. To prevent this, it is common practice to implement a softening length, that is the minimum length at which particles can interact. This modifies the force given in Equation 2.6 to Equation 3.2 where ϵ is usually the average distance between particles

$$\vec{F} = \frac{Gm_1m_2}{\left(|\overrightarrow{r_2} - \overrightarrow{r_1}| + \epsilon^2\right)^3}\left(\overrightarrow{r_2} - \overrightarrow{r_1}\right) \tag{3.2}$$

3.3 Integration of Motion ODE

The computational challenge is to calculate the force on every particle and integrate position and velocity in time. Suppose force F^i on every body were known at every point. The problem reduces to an ODE integration problem.

$$dv^i/dt = F^i(t) \tag{3.3}$$
$$dr^i/dt = v^i(t) \tag{3.4}$$

Leapfrog method would be ideal for simulations of systems under conservative forces (like gravity) for long time periods because energy is conserved. The discrete difference equations are given in Equation 3.5-3.7 with the first half step in velocity to start off the integrator shown. The error associated with this method is O(dt).

$$v^i\left(t_{1/2}\right) = v^i\left(t_0\right) + (dt/2) \times F^i\left(t_0\right) \tag{3.5}$$
$$v^i\left(t_{j+1/2}\right) = v^i\left(t_{j-1/2}\right) + dt \times F^i\left(t_j\right) \tag{3.6}$$
$$r^i\left(t_{j+1}\right) = r^i\left(t_j\right) + dt \times v^i\left(t_{j-1/2}\right) \tag{3.7}$$

Since Barnes-Hut simulation computes forces only on particles in the root node (the 150 kpc box), we have implemented periodic boundary conditions to keep particles from escaping the simulation (and taking energy away). This is effected simply by Equation 3.8 which gives the root node the topology of a torus (or classic arcade game).

$$r = \mathrm{mod}(r, 150kpc) \tag{3}$$

Energy may fluctuate wildly locally, but on the long term energy is conserved to within 10%.

4 Python Implementation

An object oriented implementation of the methods described in Section 3 was written in Python.

BHSim.py is a script that runs the Barnes-Hut simulation. It has various user controllable parameters such as number of particles, radius of the simulation domain, mass of the galaxy, scale length of the galaxy (density drops to 1/e at scale length), Barnes-Hut resolution (θ_{BH}), softening length, time step, and total simulation time. The script produces an animation showing time evolution of the system.

Body.py defines the Body class, which is an object representing a physical particle in 2D space. It carries the particle's mass (float, units of GMs), position (2D array, units of kpc), velocity (2D array, units of kpc/10Myr), force (2D array, units of $GMs * kpc/(10Myr)^2$) box boundary (in kpc, for periodic boundary condition), and color (for plotting). Class carries functions for performing leapfrog steps (implementing equations in Equation 3.7), functions for computing distance between two body objects and checking position w.r.t. quadrants, functions for computing kinetic energy and potential energy, mutators for force on body, and a function for plotting.

Quad.py defines the Quad class, which is an object representing a quadrant in 2 D space. It carries the position of its lower left corner (2D array), and its side length (float). Class carries functions that return Quad objects representing each of its subquadrants, and a function for plotting.

BHTree.py defines the BHTree class, which is an object representing a node of the Barnes-Hut tree. It carries a Quad object representing the quadrant that the node occupies. If it is not a stud node, it carries a Body object representing the aggregate particle in the node. If it has been given more than one particle, it carries 4 BHTree objects that are its child nodes. Class carries a function for inserting Bodies into the node. If it is a stub node, it accepts the Body and becomes a leaf node. If it is a leaf node, it changes its Body object into a aggregate particle, and gives its two Bodies to the appropriate children. If it is a branch node, it updates its aggregate Body and gives the new

Body to its appropriate child. Class also carries a function for computing force of the node on a Body. If Body and the node's aggregate Body satisfy Equation 3.1, then force is computed between the two. If not, then force is computed from each of the node's children on the Body instead. If the node is a stub, then no force is added to the Body. Class also carries a function for recursively plotting the entire BHTree, with all non-stub children, grandchildren, etc.

MCgalaxy.py contains functions for generating various initial distributions of particles using Monte Carlo methods. It can generate a galaxy using methods described in Section 2, and it can also generate a uniform distribution of particles with random velocities.

5 Nature of Orbits: NAFF Software

Frequency analysis of orbits is a powerful way to understand both the orbits them- selves and for large samples of orbits that are self-consistent with a potential (e.g. orbits in a N-body potential) they can provide a means to understand the phase spaces structure of the potential. And although they do provide a means towards orbit classification, classification based solely on frequency ratios is insufficient. This is because orbit families with identical frequency ratios can exhibit divergent properties, including variations in energy ranges, stability, extent, and shapes. However, automated classification of orbits, based on fundamental frequencies, is a well-developed technique that has been widely utilized. To this end, we utilized the Numerical Analysis of Fundamental Frequencies (NAFF) software, written in Fortran. The auto- classification method in NAFF incorporates multiple quantities in addition to the orbital frequencies, such as the apocenter radius of orbits, and the maximum values of x and y. [1] NAFF employs the frequency drift parameter (diffusion rate i.e. the fractional change in fundamental frequency with the largest amplitude) to measure the fraction of chaotic orbits. The frequency drift parameter $log_{10}(\Delta f)$ is computed by the change of the frequency in two equal orbital time segments, and a value of > -1.2 is a good empirical criterion for determining chaotic orbits. Orbits classified are thus tagged with *regl* (regular) or *chao* (chaotic). Hence, using NAFF, we can conveniently study the chaotic nature of orbits in a N-Body system.

6 Results and code availability

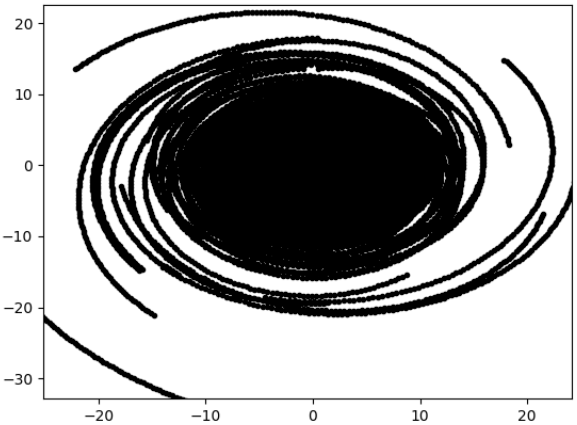


Figure 3: A simulation snapshot of 1000 particles in our fiducial galaxy. Unfortunately the bar is not visible in this snap, however in the video clip it is (faintly) visible. Axes units are in kpc.

[1] https://bitbucket.org/cjantonelli/naffrepo/src/master/

With a simulation box of 1000 particles, we extracted orbit information across all the time steps of the simulation. After making all the orbit files in a format specified in NAFF documentation, we fed the files to NAFF. Out of all the orbits, NAFF classified 90% of the orbits as chaotic. This is not very surprising–in a complicated system such as ours, periodicity in orbits should no longer be the norm. This means that particle (or stellar) trajectories are highly dependent on initial velocity and potential parameters. This is observed in many regions of the Milky Way as well, especially for systems near the central AGN Sagittarius $A*$, or when a halo star crosses the disk, where its acceleration, and hence the curvature of its orbit, changes suddenly. The cumulative effects of these sudden changes can induce chaos, though not in all orbits and not in all potentials. Most of the orbits which become chaotic stay relatively close to the disk, and range widely in the radial direction. Both heavier disks and increased halo flattening enhance the extent of the chaos.
Considering the simplifications/models we have used to model the system, it is indeed prudent to take these results with a grain of salt, and observe any variations in results with better modeling/improved computational power. It is strongly recommended to visit the GitHub pages website linked here: `https://aaevelion.github.io/IDC402_Term_Project/` for the results (which also include a video file of the simulation) and the code used in the project.

References

[1] Josh Barnes and Piet Hut. A Hierarchical 0 Nlogn Force Calculation Algorithm. *Nature*, 324:446–449, 1986.

[2] C. Hunter. Chaotic Orbits in the Galaxy. In *AAS/Division of Dynamical Astronomy Meeting #36*, volume 36 of *AAS/Division of Dynamical Astronomy Meeting*, page 02.07, May 2005.

[3] T. Manos and E. Athanassoula. Regular and chaotic orbits in barred galaxies – i. applying the sali/gali method to explore their distribution in several models. *Monthly Notices of the Royal Astronomical Society*, 415(1):629–642, 07 2011.

[4] Fabrizio Nesti and Paolo Salucci. The dark matter halo of the milky way, ad 2013. *Journal of Cosmology and Astroparticle Physics*, 2013(07):016–016, July 2013.

[5] Behzad Tahmasebzadeh, Shashank Dattathri, Monica Valluri, Juntai Shen, Ling Zhu, Vance Wheeler, Ortwin Gerhard, Sandeep Kumar Kataria, Leandro Beraldo e Silva, and Kathryne J. Daniel. Orbital support and evolution of cx/ox structures in boxy/peanut bars, 2024.

[6] Monica Valluri, Juntai Shen, Caleb Abbott, and Victor P. Debattista. A Unified Framework for the Orbital Structure of Bars and Triaxial Ellipsoids. , 818(2):141, February 2016.