What Causes the Stellar Levitation?

A Thesis Submitted for

MSci Theoretical Physics

Jialun Liu

(under supervision of Dr. Ralph Schönrich)

April 2021

Department of Physics and Astronomy

University College London

UCL DEPARTMENT OF PHYSICS & ASTROMOMY

Submission of coursework for Physics and Astronomy course PHAS0097/PHAS0048
2020/21

Please sign, date and return this form with your coursework by the specified deadline.

DECLARATION OF OWNERSHIP

I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.

I confirm that all work will also be submitted electronically and that this can be checked using the JISC detection service, Turnitin®.

I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

Signed 

PrintName Jialun Liu

Dated 12/04/2021

Title	Date Received	Examiner	Examiner's Signature	Mark

ABSTRACT

The project aims to understand the involvement of mean-motion resonances (MOR) in the stellar levitation mechanism which plays an important role in explaining the formation of galactic thick discs as well as the buckling of galactic bars. The process of levitation captures the resonant stellar orbits resulting in a transfer of motion from the radial to the vertical component, thereby achieving the effect of lifting stars' orbits higher above the galactic plane. We are particularly interested in the characteristics of 1:1 MOR stellar orbits in the disc and discussing potential factors that might cause and suppress the effect of stellar levitation. If we can establish a firm connection, we will solidify our understanding of the levitation model, thus pave the path to fully understand the structural formations in disc galaxies.

CONTENTS

1. INTRODUCTION

1.1. MOR and levitation

Any object with periodic motion can be treated as an oscillator, its motion corresponds to a certain frequency through Fourier transformation. Resonance occurs when this frequency matches the natural frequency, as a result increasing the amplitude of oscillation significantly. On the same principle, the MOR describes the orbital resonances of moving stars and occurs when the ratio between the radial and vertical oscillation frequencies is an integer number. It is mathematically defined as $f_x : f_y : f_z = l : m : n$ (where l, m, n are integers), according to the notation from Sellwood and Wilkinson (1993). In this paper, the notation of $f_R : f_z$ will be more frequently adapted and the dynamic features of stellar orbits in 1:1 MOR are of our biggest interest.

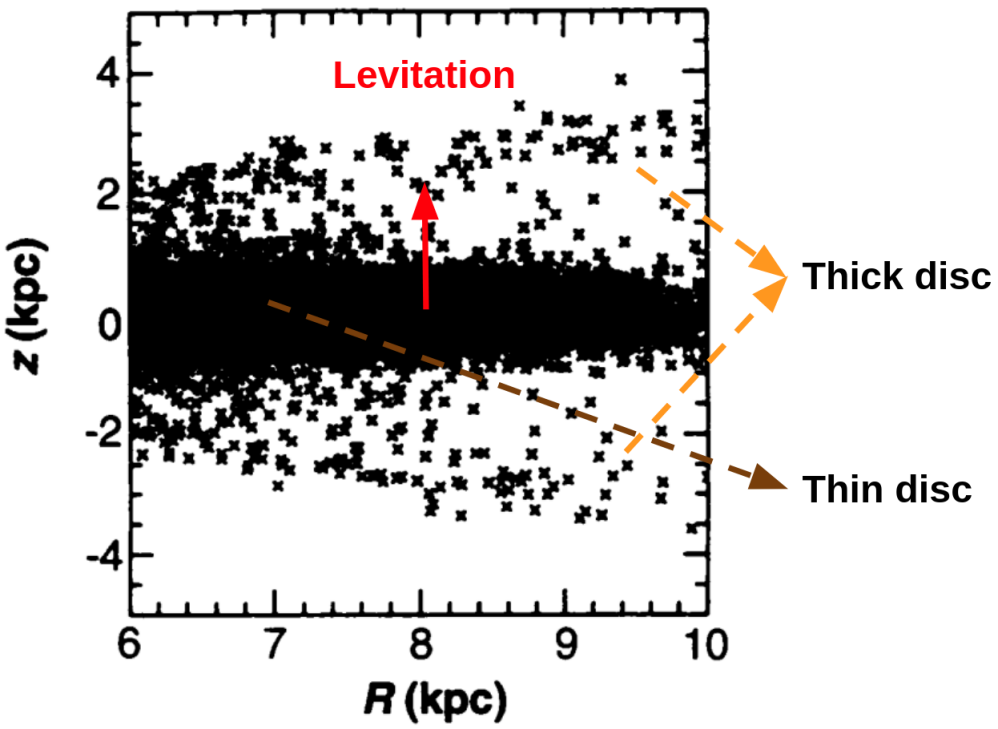

Figure 1.1: A side on view (the (R, z) plane) of the dissection of the galactic disc which have a thin and a thick components. The diagram is adapted from the results of levitation model simulation, developed by Sridhar and Touma (1996a). Some stars levitate from the thin disc to make up the thick disc in the direction of the red arrow labeled.

The mathematical model of levitation was developed by Sridhar and Touma (1996a,b) to explicate the formation of galactic thick discs (see figure 1.1). It is a

mechanism that explains the levitating stellar orbits in a growing disc as a result of adiabatically captured into resonances (Sridhar and Touma, 1996a). The concept of adiabatic process refers to the slow variations in potential compared to the orbital frequencies of stars (Binney and Tremaine, 2008). If the disc potential is increasing adiabatically, the stellar orbits trapped in resonances will transfer their radial momentum into vertical momentum, hence get heightened few kilometres above the galactic plane to form the thick disc. The resonance that sridhar and Touma investigated was the 2:2 radial and vertical epicyle frequencies which are defined as the frequencies (e.g. f_R and f_z) of nearly circular orbits particularly (see the mathematical derivation in section 3.2.2.). An opposite argument challenges their work by doubting whether disc orbits have high enough initial radial actions to convert into vertical actions.

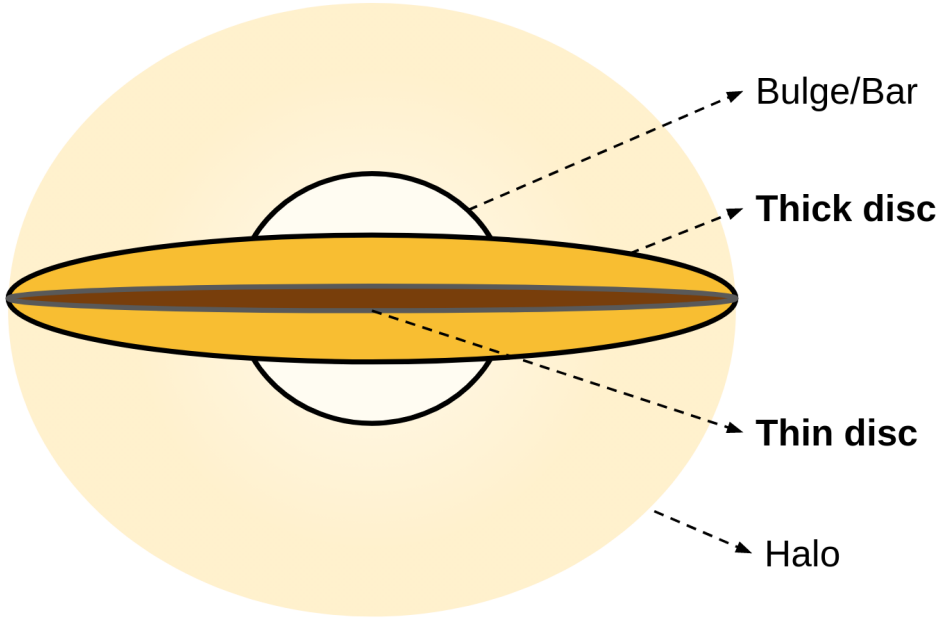

Figure 1.2: A simplified structure of a disc galaxy, with components such as galactic Bar/bulge, galactic thick disc, galactic thin disc and halo.

The concept of MOR, together with the mathematical model of levitation are widely applied to explain galactic dynamical phenomena. In the next subsections, the two following questions will be answered with more refined details. **(1) Where do galactic thick discs come from?** While the galactic thin disc is adiabatically heated/ thickened, the levitation model enables the resonant stellar orbits located in the thin disc to transfer to the thick disc (section 1.2.). **(2) Why do a galactic bar buckles to form a BP/X-shaped morphology?** There is a long story to buckling, but overall resonance is the key factor for the formation of BP/X-shaped bars. Due to the coupling of MORs and

the bar's resonance arises from its rotation, the bar undergoes a gentle and stable thickening process. The two most convincing mechanisms (see section 1.3.); **resonance sweeping** (Weinberg, 1985) and **resonance capturing** (Quillen, 2002) have adapted the mathematical model from the levitation mechanism.

1.2. Galactic thick disc

Galactic disc (thin disc) is the most common structure of a large spiral galaxy, two thirds of spiral galaxies harbour both thick and thin discs in their galactic plane, including our MW. These two components of disc are better differentiated by viewed from edge-on in Figure 1.2. As suggested by their names, the stellar population within a thick disc is distributed vertically higher than the ones in the thin disc. This is highlighted by the scale height of MW's thick disc is around 1 kpc perpendicular to the galactic plane, but the thin disc is only at the scale height of 0.3 kpc (Gilmore and Reid, 1983; Jurić et al., 2008). This segmentation of MW's discs was first postulated by Gilmore and Reid (1983) using the technique of photometric parallax. Through the star count, the observation of double exponential vertical density profiles $\nu(z)$ rather than a single exponential validated the prediction of two separate disc components in the MW. (Bensby and Feltzingand, 2006; Jurić et al., 2008). An opposite point view from Schönrich and Binney (2009a); Roskar et al. (2013) claims the overlap of thick and thin discs, thereby giving us the conclusion that there are no distinct thick discs.

The discovery of thick disc has fundamentally changed our perspective towards the structure and evolution of disc galaxies. The groups of stellar populations in thick discs are dynamically and chemically distinctive in comparison to thin discs. The thin disc is made of young stars (less than than 7 billion years (Schönrich and Binney, 2009b)) and gas, whereas the the thick disc is mainly made of stars, taking up roughly 30 % of disc stars (Bensby and Feltzingand, 2006; Jurić et al., 2008). They are extremely old (more than 10 billion years), have poor metalicity content and higher vertical velocity dispersion. Those old stars in the thick disc carry the kinematic and chemical evidence for our early universe. By analysing the migration of these stars, one can interpret the structural formation of galaxies.

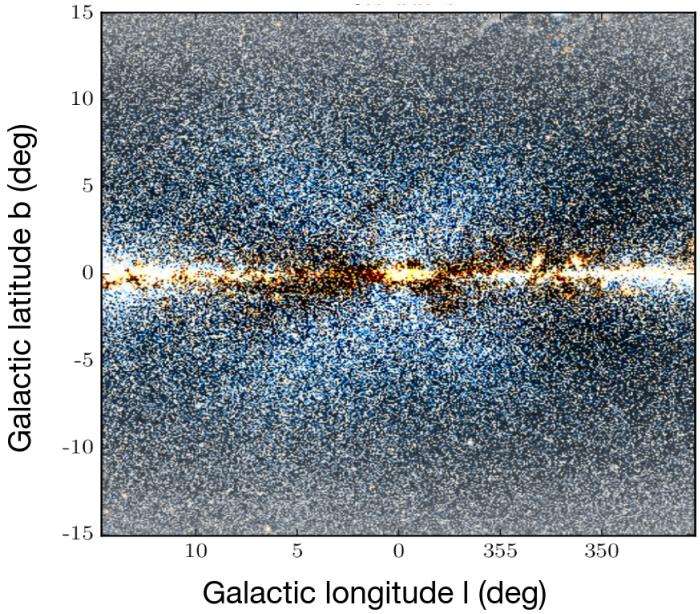

Figure 1.3: The clearest morphological X-shaped bar image of our MW, achieved by Ness and Lang (2016). WISE, an Infrared telescope, was launched to conduct a full-sky photometric survey (Wright et al., 2010) in the central region of the MW.

1.3. BP/X-shaped galactic bar

Most disc galaxies exhibit a central bulge as demonstrated in Figure 1.2. Our MW's bulge/bar has a BP shape, characterised by a central X-shaped structure (see Figure 1.3). This type of bulge is well confirmed to be the thicker component of the strong galactic bars (Bureau et al., 2006; Erwin and Debattista, 2013) which arise from the instability of the disc. McWilliam and Zoccali (2010) interpreted the X morphology from the split of red clump stars in the MW bar. The two equally populated peaks in the density distribution indicate the existence of two populations of stars that forms an X shape (McWilliam and Zoccali, 2010; Nataf et al., 2010). Like the MW, nearly half of external disc galaxies have now been observed to own BP/X structures (Shaw, 1987; Combes et al., 1990; Laurikainen et al., 2011). A central galactic bar plays an essential role in the dynamical evolution of a galaxy: its presence might affect the star formation rates (Hawarden et al., 1986; Hummel et al., 1990), spatial star formation patterns and overall metallicity contents (Friedli et al., 1994; Vera et al., 2016). The relation between stellar populations within the bar and the bar thickening process is still obscure. Hence, understanding the bar formation enables us to interpret the evolutionary history of the MW. More importantly, comparing the MW with other disc galaxies broadens our horizons with the kinematic structure of the external galaxies.

A galactic bar is originally a flat elongated component often regarded as a rotating rigid body with a steady pattern speed (Binney and Tremaine, 2008). So, **what causes the buckling or thickening?** Many studies have proven that a galactic bar grows in size by exchanging the angular momentum from its inner disc to outer discs and dark matter halos (Athanassoula, 2003). As it thickens and slows down (Aumer and Schönrich, 2015; Chiba et al., 2019), the bar bends the stars' orbits out of the disc plane under two main mechanisms based on the theory of resonances, **resonance sweeping** and **resonance capturing.**

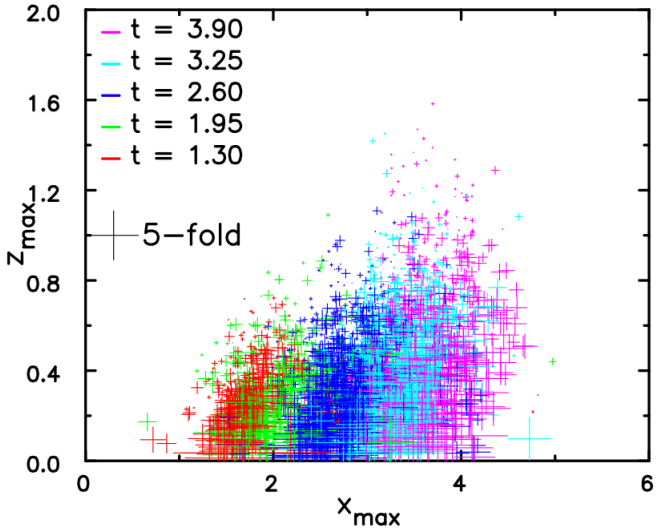

Figure 1.4: Resonant stars' orbits in the x_{max}, z_{max} plane at five different times, t = 1.3, 1.95 2.60, 3.25, 3.90 (Gyr) from Sellwood and Gerhard (2020).

The resonance sweeping model (Weinberg, 1985; Hernquist and Weinberg, 1992; Quillen et al., 2014) predicts the resonance to sweep through the disc as the bar slows down and grows thicker. The bar can be regarded as a decelerated rotating carousel that spins out its stars. Once a star's orbit begins to resonate, it will be pushed further above the plane of the disc and become part of the BP/X morphology. When the BP/X structure moves outwards via forming new resonant orbits, old ones will be detached and pertain their original trajectories. This mechanism was mathematically devised from a perturbed axisymmetric Hamiltonian from a non-axisymmetric bar (Contopoulos and Weinberg, 1975) and computationally examined by Sellwood and Gerhard (2020). In Figure 1.4, x_{max} gradually deviates from the bar's centre when the time passes by; at a given time, the stars' orbits enlarge their vertical oscillation ranges (z_{max}) in the resonance. This is strong evidence for the radially growing bar when the resonance is swept outwards.

Resonance capturing from Quillen (2002), predicts a vertical growth of the bar if stellar orbits are trapped by the 2:1 vertical Lindblad resonances (VLR). The Lindblad resonance can be regarded as the resonance of the bar due to its self-rotations. The bar thickening processes is mainly caused by the 2:1 VLR, with two vertical oscillations per rotation of the bar, $\Omega_x : \Omega_z = 1 : 2$. More precisely, when the stars are caught by the resonant motion of an evolving bar, the orbits of stars will levitate over the mid-plane of the galaxy to broaden their vertical oscillation ranges. This mathematical model was later simulated and proven by Sellwood and Gerhard (2020) using N-body simulations (Pfenniger and Friedli, 1991).

Although both theories arise from the 2:1 VLR from the bar, the differences between them are obvious. **(1)** The resonance width in the resonance capturing is narrow (Quillen, 2002), so that the thickened region on the bar might be too small to match up with the actual size of the BP/X-shaped bars from the observational data. **(2)** Resonance capturing will result in a higher stellar density in the inner bar due to the angular momentum loss of the stars' orbits from the inner bar to the outer halo. In Figure 1.5, the right plot's vertex is noticeably narrower and taller than that of the left.

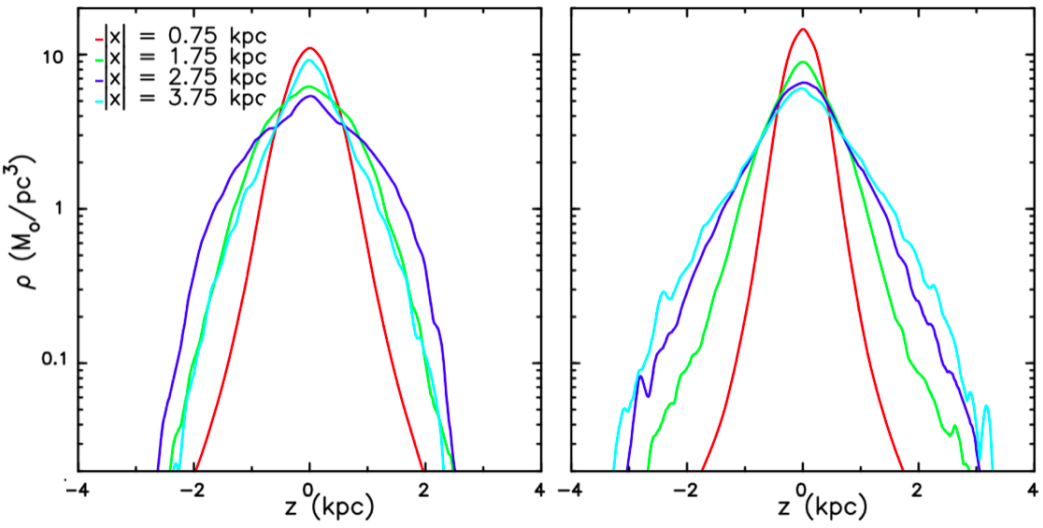

Figure 1.5: The vertical density distributions at varying distances across the bars by the end of the resonance sweeping (left) & capturing (right) simulations. The radial distances are 0.75, 1.75, 2.75, 3.75 (kpc) respectively. Adapted from Sellwood and Gerhard (2020).

The following thesis is organised as follows: In the next section 2, we will review the theory and understand the action of stellar orbits in an adiabatic growing potential. Section 3 describes the numerical methods that evaluate the MOR of regular and disc orbits as well as studying the topological features of the MOR in the phase space. In section 4, a further discussion with results concerning the relationship between stellar levitation and MOR orbits is presented. Section 5 is the final conclusion summarising the important results and future work needed.

2. REVIEW OF ACTION THEORY

In sections 1.2 & 1.3, we have discussed the applications of stellar levitation in the senarios of forming galactic thick discs and BP/X-shaped galactic bars. The slow growth of the galactic potential, implying slow increases in the energy of the system. For instance, the stellar levitation taking place between the thin and thick discs requires the slow growth of the thin disc; the thickening and buckling of the bar indicates the growing bar potential. Therefore, it is of great importance to obtain constant quantities to study the dynamical features of stellar orbits in a slowly growing potential. **Angle-action variables** (θ, J), constant of motions (Binney, 2012) are defined to solve this issue, with the benefit of remaining approximately conserved under the adiabatic changes.

The angle-action variable method calculates the frequencies of stellar motion through Canonical transforming its variables, avoids solving the complicated equations of motion directly. The angle-action coordinate (θ, J) is constructed to demonstrate the integral of stellar motion in the phase space. A good approach for visualising this coordinate system is to imagine an orbital torus, the angle θ is the generalised coordinates (conjugate to actions J (Binney and Tremaine, 2008)) where the torus sits; the actions can be treated as the surface of the torus. There are three action integrals, such as the radial action J_R, vertical action J_z and the azimuthal action J_ψ. The radial and vertical actions describe the radial and vertical motions of a star.

$$J_R = \frac{1}{2\pi} \oint dR\, p_R \tag{1}$$

The azimuthal integral is equivalent to the angular momentum L_z.

$$J_\psi = \frac{1}{2\pi} \oint d\psi\, p_\psi = L_z \tag{2}$$

2.1. Motions near a resonance

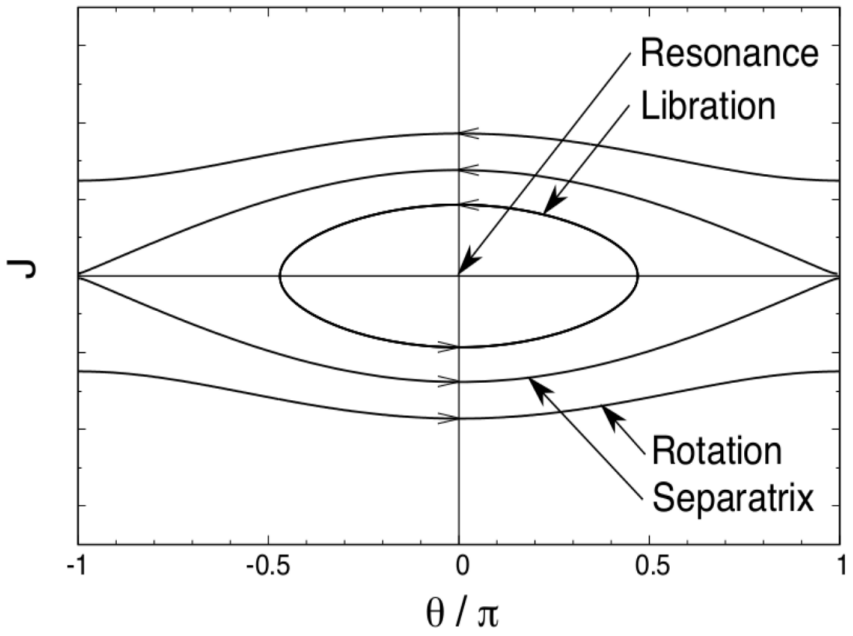

Figure 2.1

To put the theory into practice, Figure 2.1 is the simplest case that demonstrates the dynamics of an unperturbed pendulum in an angle-action coordinate (θ, J). The resonance sits in the centre, and the separatrix is regarded as the boundary between the phase flows of resonant and non-resonant orbits. For example, an orbit gets captured by the resonance will librate within the separatrix, its direction is depicted by the arrow of libration in Figure 2.1. Oppositely, non-resonant orbits circulate outside of the separatrix with an angle θ range of 2.0 π (Lichtenberg and Lieberman, 1992). When the orbit is above the separatrix, the less value of action J it has, the further away it is from the resonance. If the potential changes, the trapped orbits and non-resonant orbits have different possibilities to escape or captured by the resonance (Goldreich and Peale, 1966).

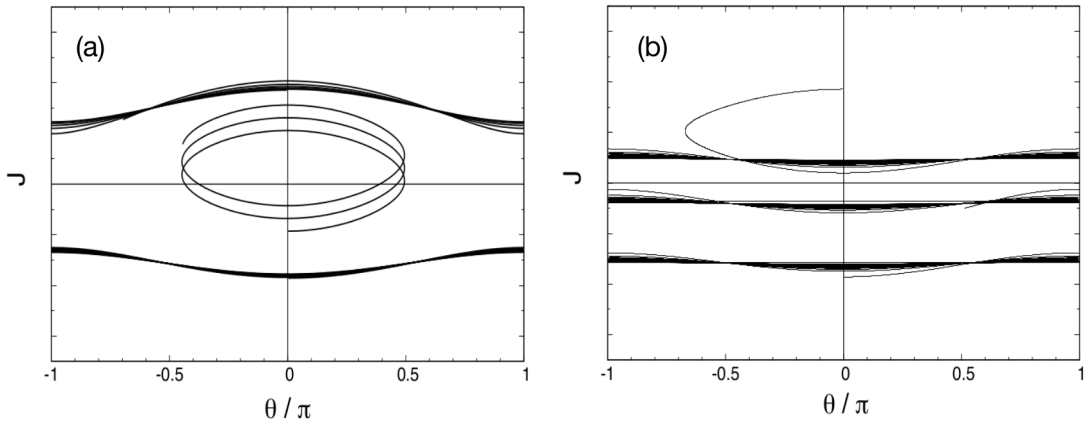

Figure 2.2

Figure 2.2 discusses the responses of stellar orbits in (θ, J) coordinate towards changes in potentials with two different rates. The adiabatic process in (a) changes the system's potential slow enough to conserve the actions. This process is called the resonant dragging, which is the same resonant capturing effect claimed in the levitation mechanism. The overall actions of the resonant and non-resonant orbits have a small systematic shift in the positive direction due to the changes in the potential. In Figure 2.2 (b), the rate of changing in the potential is much greater than the orbital frequency of the pendulum, thus the initially resonant orbit cannot complete a single libration cycle before the resonance has moved away. We can relate the two completely different processes by analogy to the phenomena of the motion of pulling a cloth under a glass of water. In this scenario, the resonance is the cloth and the initially trapped orbit is the glass. If the cloth is pulled fast enough, it will be separated from the glass without spilling the water, otherwise, the glass will move slowly with the cloth. In this paper, to obtain conserved actions in a growing potential, its rate of change is set to be 10^{-4} Myr^{-1} which is significantly smaller than the stellar orbital frequencies in an order of 10^{-2} Myr^{-1}.

3. SIMULATIONS

In this chapter, we introduce methods to investigate and analyse the levitation of stellar orbits in an axisymmetric potential (e.g. disc's potential). Future work is required to evaluate the case when the potential is perturbed by a non-axisymmetric component (e.g. bar's potential) using the same techniques. The chapter is organized by firstly setting up the initial conditions for simple orbit simulations (section 3.1.), including adapting constant and growing disc potentials (section 3.1.1.), introducing two different numerical frequency analysis techniques (section 3.2.), locating and visualising the MORs in a 2-D phase space through the surfaces of section (section 3.3.). To illustrate these methods, we will simulate a 2:3 MOR test orbit with energy per solar mass E equals -0.13965 kpc^{2}Myr^{-2} as an example. The initial position of this star in Cartesian coordinate is (8.0, 0, 0) kpc, its initial velocity vector is (96.32, 220.0, 26.88) km/s.

3.1. Initial conditions

In this paper, the cylindrical coordinate $(R, z, \phi, v_R, v_z, v_\phi)$ is used to describe the dynamical system and keep track of the stellar orbit in a 6-dimensional (6-D) phase space. The circular velocity v_c is taken to be 239.1 km/s, the galactocentric distance R_0 is 8.29 kpc.

3.1.1. Potential of the disc

The galactic potential used is the McMillan (2017) potential, consisting of a disc and a halo. We refer this particular potential as the standard potential in this paper. The potential of discs is axisymmetric which means the symmetry about an axis, in this case, it is the symmetry in vertical z axis. Despite real disc galaxies do harbour a non-axisymmetric bar potential in the centre, the disc potential $\Phi(R, z)$ can still be approximated to as being symmetric in the mid-plane (z =0 kpc). Therefore, determining stellar trajectories in such potential has become simpler because the angular momentum L_z will always be conserved.

Disc potential can be described separately by a radial and a vertical profiles. The

exponential surface density of the disc $\Sigma(R)$ is formulated as follows:

$$\Sigma(R) \;=\; \Sigma_0\, exp\left(\; -\; \frac{R_0}{R}\; -\; \frac{R}{R_d} + \epsilon\, cos\left(\frac{\pi\, R}{R_d}\right)\right) \tag{3}$$

Where Σ_0 is the central surface density in units of $M_{\odot} kpc^{-2}$, R_0 is the inner cutoff radius, R is the cylindrical radius in kpc, R_d is the scaling of the disc in kpc, ϵ is a perturbation of the disc. The vertical density profile of the disc $\rho(R, z)$ is also exponential, defined as:

$$\rho(R, z) \;=\; \frac{\Sigma(R)}{2\, z_d}\, exp\left(\frac{-\,|\, z\,|}{z_d}\right) \tag{4}$$

Where z_d is the scale height in kpc.

Stellar levitation occurs in an adiabatically growing disc potential. It requires an algorithm that increase the stellar mass of the disc slowly over time. In such a model, the potential of halo remains constant, the initial mass of the disc is set to be 0% to 10% of its final mass to mimic a compact thin disc. For simplicity, the thin disc gets ramped up linearly with time.

3.1.2. Simple orbit simulations

A stellar orbit is the trajectory of a star under the influence of gravitational potential. It evolves over time, containing information about the star's position, velocity and change of potential/force at each time resolution. I have expanded and adapted an existing C++ based on the leapfrog algorithm to integrate stellar orbits. The algorithm incorporates a drift-kick-drift scheme to approximate the orbit. This ensures time symmetry and energy conservation. In this way, the particle's orbit is constrained in the galactic potential and is prevented from bumping into the galaxy's centre. The time-step factor of the orbital integrator is chosen with a fixed small value (0.005 Myr) to ensure higher precision of the force or the change of the potential/force for each step of the simulation.

The points on the simulated orbit randomly extracted by the code have varying time intervals (the green points in Figure 3.1), they do not fit the Fourier transform well. Hence, I interpolated between neighbouring points to repopulate their configurations in the orbit with a fixed time intervals: **(1)** Cubic spline interpolation was implemented to smooth out the lines passing through the original data points in the orbit. **(2)** 16384

points with equal time intervals were re-selected from the stellar trajectory. **(3)** This new time interval is defined as the fraction of total time period and the number of points. In Figure 3.1, the time interval after interpolation (shown by the purple points) is found to be 1.16912 Myr, and the total time period of this stellar orbit is 19.15 Gyr.

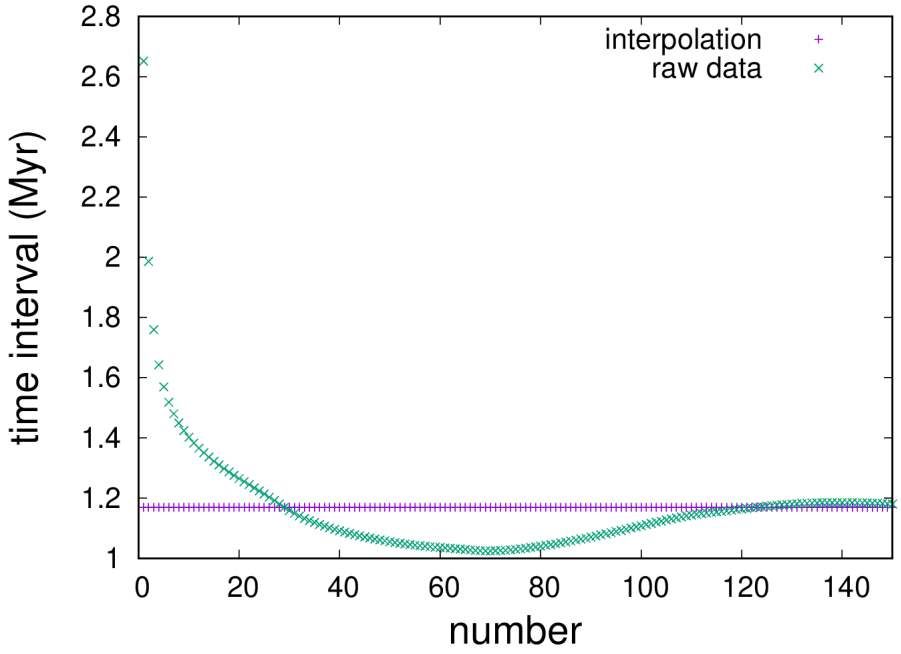

Figure 3.1: The time intervals of the first 150 out of 16384 points before and after interpolations. The points in green are first-hand data generated by the leapfrog algorithm, the purple ones have invariant time intervals through the interpolation.

3.2. Orbital frequencies analysis

3.2.1. Method A: Fourier analysis

We intend to develop a method to identify the MOR of a regular stellar orbit. The most straightforward approach is to take a proper Fourier transformation of the radial $R(t)$ and vertical $z(t)$ arrays of the orbit to obtain corresponding frequencies. By comparing the radial and vertical frequencies, we achieve the exact value of MOR. This technique is a short cut to the angle-action variables method. We analyse the frequencies and angles of an orbit separately without solving the derivatives of the Hamiltonian with respect to a new generalized momenta.

Firstly, we introduce a variable $\theta(t)$ to express star's vertical motions rather than the vertical oscillation ranges $z(t)$. $\theta(t)$ is the angle between a star's radial and vertical positions in the galactic coordinate system as shown in Figure 3.2, and mathematically

defined as $\theta(t) = \arctan\left(\frac{z(t)}{R(t)}\right)$. The reason to use θ is owing to the fact that the shape of a regular stellar orbit is better related to the actual angle variable connected to the vertical action. Hence this variable counts up almost linearly and provides a decent result from the Fourier transform. Therefore, the frequency ratio between R and θ identifies the exact MOR in a regular star's orbit.

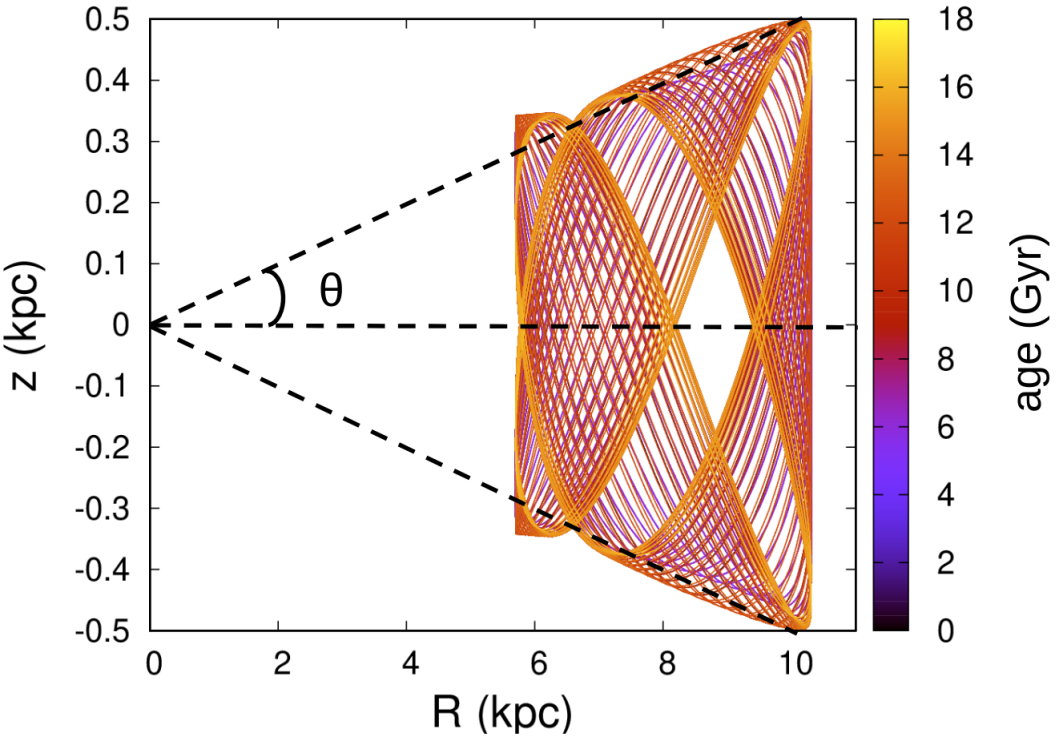

Figure 3.2: The stellar trajectory of the test star in the (R, z) plane. θ is the angle between the radial and vertical positions of the test star in a cylindrical coordinate.

In Figure 3.2, the orbit has been integrated long enough (18 Gyr) to see the effect of libration. It means the shape of the resonant stellar orbit changes gradually throughout the time by exchanging the vertical and radial energies/motions. The variation in colour (from blue to yellow) depicts the systematic orbital shift after each vertical oscillation.

In Figure 3.3, the fundamental frequencies are a group of non-zero frequencies with the highest peak. They are the most significant contributors for the determination of MOR. The vertical fundamental frequency with the highest peak is at 0.012 Myr^{-1} and the radial fundamental frequency roughly equals to 0.008 Myr^{-1}. The ratio between radial and vertical frequencies is in terms of two integer numbers, 2 and 3, thus this orbit is in a 2:3 MOR. The fundamental vertical frequencies have a weaker beat around 0.004 Myr^{-1}, caused by the interaction of the radial and vertical frequencies. The other

weaker frequencies are overtones with higher frequency orders, which arose from the non-sinusoidal wave-function of the stellar orbit as well as the coupling from beats between the two orbital frequencies.

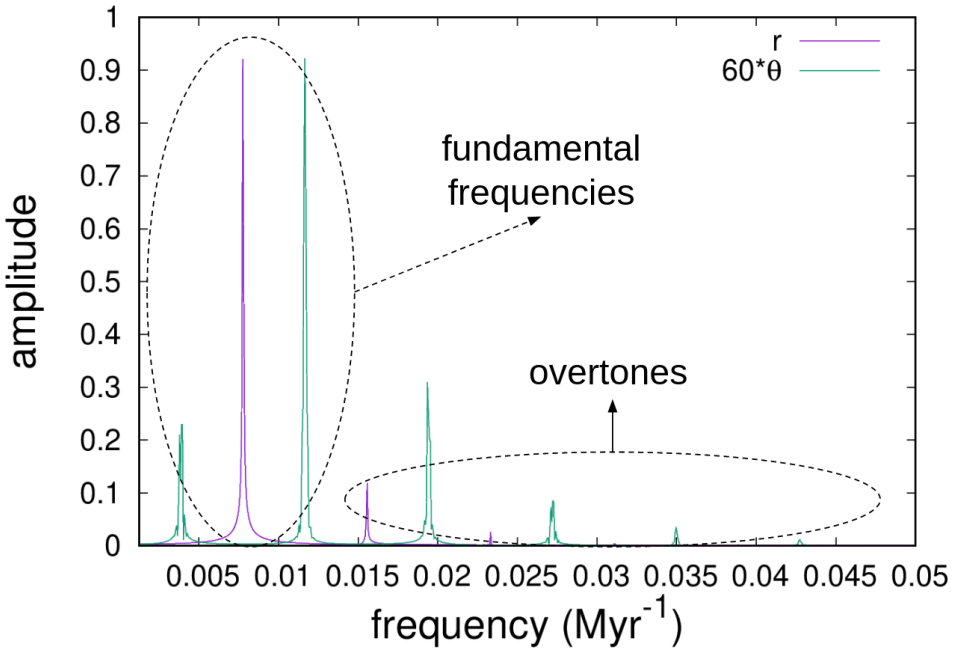

Figure 3.3: The radial frequency f_R of the test star's orbit plots over f_θ which is related to the vertical frequency. The peaks of f_R are in purple, the green ones are f_θ.

3.2.2. Method B: Epicycle frequency analysis

This technique is developed with the goal of tracking where and when the 1:1 MOR passes through disc orbits, thus determining the exact moment of stellar levitation. The advantage of this method is the involvement of circular orbits which are fixed and only have dependency with the strength of the galactic potential. Since most of the disc orbits are almost circular, astrophysicists describe them as perturbed circular orbits with the concepts of radial and vertical epicycle frequencies κ and ν. **They are measures of how much a stellar orbit deviates from the frequency of a circular orbit in the radial and vertical directions respectively.** κ is defined as the radial epicycle frequency, mathematically expressed as:

$$\kappa^2(R_g) = \frac{\partial^2 \Phi_{eff}}{\partial R^2}\bigg|_{(R_g,0)} \tag{5}$$

ν is the vertical epicycle frequency,

$$\nu^2(R_g) \;=\; \frac{\partial^2 \Phi_{eff}}{\partial z^2}\bigg|_{(R_g,0)} \tag{6}$$

Φ_{eff} is the effective potential defined in equation (7), R_g is the guiding-centre radius where a circular orbit is located in a particular Φ_{eff}.

Theory part: To derive the expressions in equation (5, 6), we start with the motion of an star's orbit in the mid-plane of a disc. In a cylindrical coordinate system, the angular momentum L_z is conserved in an axisymmetric potential $\Phi(R, z)$. Hence, we treat the L_z as a constant and combine it with the potential to define the effective potential Φ_{eff} in terms of (R, z, Lz) as follows:

$$\Phi_{eff}(R, z, L_z) \;=\; \Phi(R, z) \;+\; \frac{L_z^2}{2R^2} \tag{7}$$

The next important quantity to look at is the energy of the system which is better described as effective Hamiltonian $H_{eff}(R, z, v_R, v_z, L_z)$. It is a combination of the potential of the system and kinetic energy of the orbit. A note to take from here is that velocities and momentum are equivalent in this paper due to the mass of stars is 1.

$$H_{eff}(R, z, v_R, v_z, L_z) \;=\; \Phi_{eff}(R, z, L_z) \;+\; \frac{1}{2}\left(v_R^2 \;+\; v_z^2\right) \tag{8}$$

We Taylor expand the Φ_{eff} from equation (7) at the plane where R = R_g $z = 0$, and re-express the $\Phi_{eff}(R, z, L_z)$ of a nearly circular orbit as the $\Phi_{eff}(R_g, 0)$ of an circular orbit with radially and vertically perturbed potentials in terms of radial and vertical epicycle frequencies; where x is $R - R_g$.

$$\begin{aligned}
\Phi_{eff}(R, z, L_z) \;&\simeq\; \Phi(R_g, 0) \;+\; \frac{1}{2}\,\frac{\partial^2 \Phi_{eff}}{\partial R^2}\bigg|_{(R_g,0)} x^2 \;+\; \frac{1}{2}\,\frac{\partial^2 \Phi_{eff}}{\partial z^2}\bigg|_{(R_g,0)} z^2 \\
&\simeq\; \Phi(R_g, 0) \;+\; \frac{1}{2}\,\kappa^2(R_g)\,x^2 \;+\; \frac{1}{2}\,\nu^2(R_g)\,z^2
\end{aligned} \tag{9}$$

Numerical part: In the simulation, the first step of calculating κ and ν is to devise the effective potential $\Phi_{eff}(R, z, L_z)$ of nearly circular orbits according to equation (9). To do so, we need to determine the variables, the potential $\Phi(R, z)$ and the angular momentum L_z (equation (7)). Potential can be directly extracted from the simple orbit algorithm; L_z is a function of circular velocity v_c and guiding radius R_g, mathematically defined as $L_z(R_g)$ = $R_g v_c(R_g)$. As a result, if we can express v_c in terms of R_g, we will

be able to evaluate $L_z(R_g)$ and hence determine $\Phi_{eff}(R, z, L_z)$. In fact the relation between $v_c(R_g)$ and R_g is the rotation curve of the galaxy. It describes the velocities of circular stellar orbits at different radius from the galactic centre. The discrepancy between the theoretical prediction and experimental observation of its shape had raised up the dark matter problem first posited by Oort (1932).

According to equation (5, 6), we can approximate $\kappa(R_g)$ and $\nu(R_g)$ respectively by differentiate the achieved $\Phi_{eff}(R, z, L_z)$ with respect to a range of R and z at (R_g, z) plane. In Figure 3.4, $\kappa(R_g) : \nu(R_g)$ is significantly higher when the corresponding R_g range is within 2.0 kpc due to the strong central gravitational potential. The ratio drops below 1.0 if the radius of the stellar obits is in the region where the slope of the gravitational potential is shallower. Later, we will apply the same method to investigate the location of MOR by considering a time-dependent epicycle ratio.

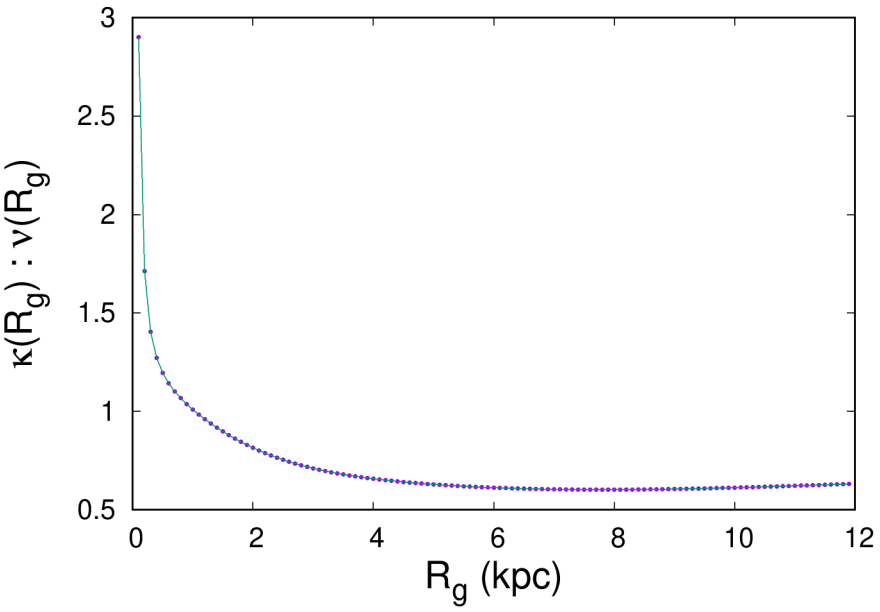

Figure 3.4: The guiding centre radius R_g against epicycle frequency ratio $\kappa(R_g) : \nu(R_g)$ in a constant standard potential.

3.3. Surfaces of section (SOS)

From the previous section 3.3, we discussed two numerical methods to calculate the MOR of a star's orbit. However to clearly demonstrate the shape and location of the resonant orbits, especially the 1:1 MOR, we need to consider the SOS. This technique tracks the stellar orbits in the phase space as well as identifies the position of the MOR.

The concept of SOS is to reduce the 6-D phase space to a 2-D plane for the ease of

visualisation, invented by Poincaré (Rasband, 1990). This method works as follows: firstly, the 6-D phase space $(R, z, \phi, v_R, v_z, v_\phi)$ has 4-D motions (R, z, v_R, v_z). Due to the constant effective Hamiltonian $H_{eff}(R, z, L_z)$, this 4-D phase space reduced to a 3-D volume (R, z, v_R) with positive v_z. In another words, the energy integral of the orbit allows one to interpret v_R and v_z by fixing one of their signs. Lastly, slice through this 3-D volume at $z = 0$ to create the 2-D SOS (R, v_R).

Owing to the axisymmetric disc potential and time invariance, angular momentum L_z and orbital energy E are constants of motions. The third integral of motion I (Binney and Tremaine, 2008) confines the curves in a 2-D phase space to 1-D discrete points as illustrated by the two orbits in Figure 3.5. Those discrete points in the non-resonant orbit forms an oval-shaped loop, whereas, the points in the 2:3 mean-motion resonant orbit partially covers the curve. This is because a resonant orbit cannot fully cover the volume in the given phase space, therefore analysing the SOS of a stellar orbit is an efficient way to identify resonant orbites.

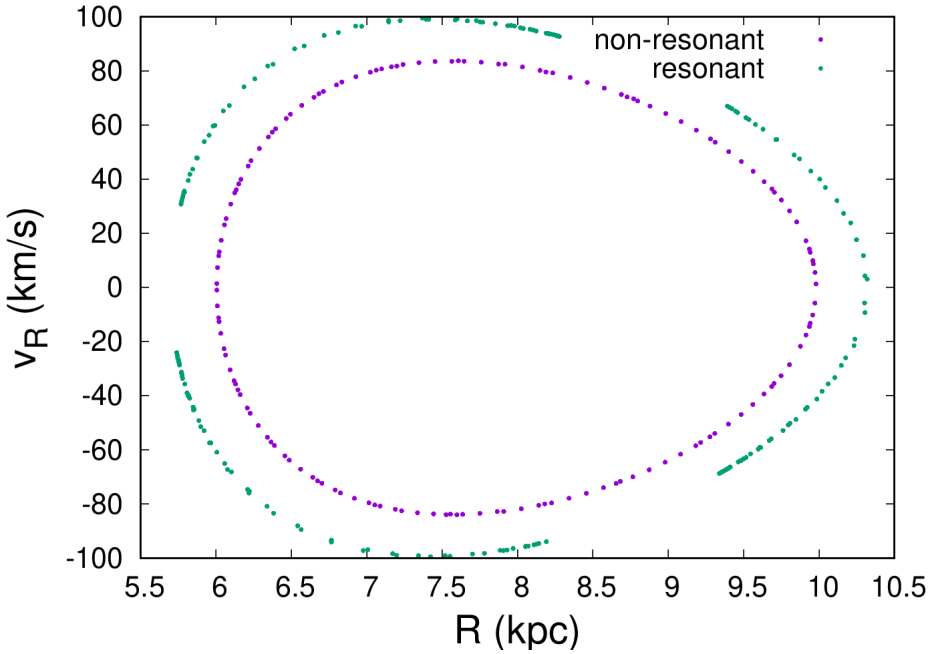

Figure 3.5: The SOS of the 2:3 resonant orbit and a non-resonant orbit with $f_R : f_z$ = 0.863014. The resonant orbit is in green colours and the non-resonant one is in purple.

4. RESULTS AND DISCUSSION

4.1. Stellar levitation

Before getting into the study of how the MOR behave in the process of levitation, we look at a simpler case of stellar levitation.

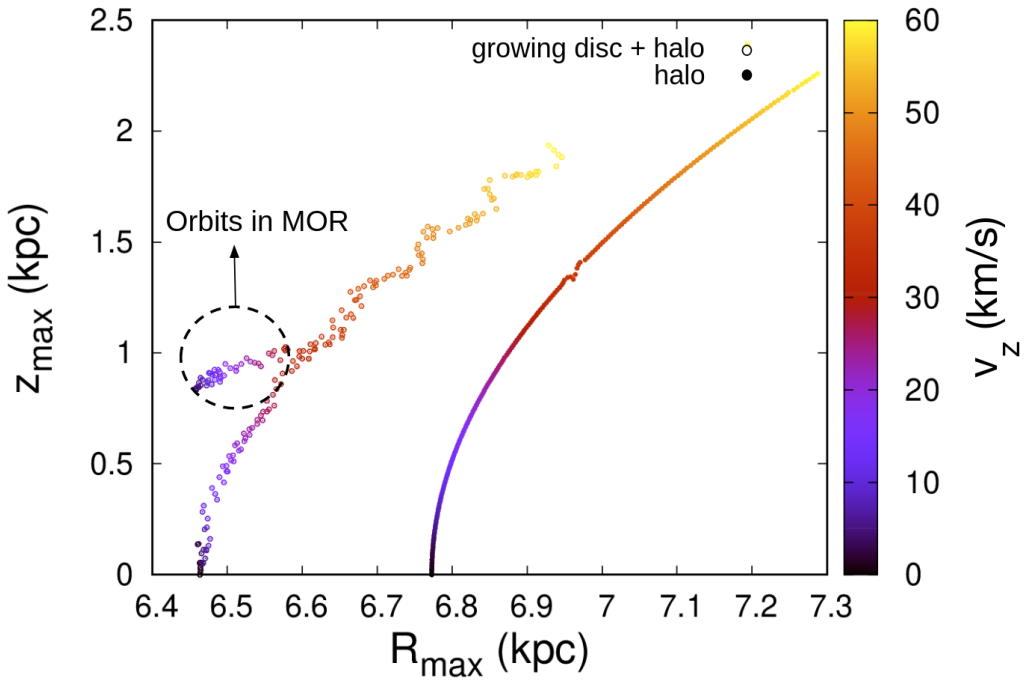

Figure 4.1: Two groups of stellar orbits in the (R_{max}, z_{max}) plane, circles represent the orbits in both a standard halo potential and an adiabatically growing disc potential, and the dots refer to the ones in a constant standard halo potential. There are 600 stars' orbits in each group, every circle/dot corresponds to an orbit. The initial positions of all stars are (4.0, 0, 0) kpc away from the galaxy centre in a Cartesian coordinate. The initial velocity vector is (80.0, 180.0, v_z) km/s, and v_z varies from 0.0 km/s to 60.0 km/s as indicated by the colour bar.

The essence of the levitation mechanism is the increasing vertical oscillation range of a star's orbit due to the adiabatic growth of the disc potential. Therefore, it is of great importance to investigate an orbit's vertical extent before and after the growth of a disc. For simplicity, we assume the disc is growing slowly in a constant standard halo potential with zero initial mass over 18.0 Gyr. Therefore, actions are conserved. Figure 4.1 compares the maximum vertical extents z_{max} that a group of 600 stellar orbits can reach in two sets of potentials, constant halo potential and a combination of a halo and a growing disc potentials. All of stars are initially set up to be released at the same location and radial kinetic energy but different vertical velocities.

In Figure 4.1, two factors affect the values of z_{max} : **(1)** Initial vertical velocities. When the initial position and radial motion of stars are fixed, z_{max} increases exponentially with a linear increase in stars' initial vertical velocities. The answer is simply that the more vertical kinetic energy a star has, the higher the distance above the mid-plane it can reach. Hence, the two curves in the diagram reflect the shapes of the galactic potentials. To elaborate further, the potential of the halo is constant, therefore, the smooth line formed by dots suggests a constant halo potential which attracts moving stars. **(2)** MOR. The split of stellar orbits that exceeds the curve in a growing potential (marked in Figure 4.1) is a sign of the MOR occurring in these orbits. This phenomenon allows the stellar orbit to exchange momentum in radial and vertical components which results in a decrease or increase of the z_{max} around such resonance.

Subtracting z_{max} of the same stellar orbit in both potentials to eliminate the effect of the halo potential, we obtain the result of stellar levitation caused by the MOR in purely growing disc potential. In Figure 4.2, a small number of stars with low initial vertical velocities arise at the top of the majority of stars, suggesting an increase in their vertical actions due to the MOR. To clarify, the differences between stars' initial vertical velocities result in their varied frequency ratios $f_R : f_z$. The MOR happens to appear at the orbits with initial v_z around the range of 0 km/s to 30 km/s. To push forward our understanding of actions in stellar levitation, one may ask **what happens to the radial action?** Since the disc potential is growing under adiabatic process, with the concept of conserved actions in mind, radial actions are expected to decrease for the resonant stellar orbits whose vertical actions are increased. If we look back to Figure 4.1, it is clear to see the radial action reduction (decrease in R_{max}) for stars' orbits in the growing potentials respect to the ones in the constant potential. The majority of the stars in Figure 4.2 experience the action contraction because their orbits are non-resonant.

In physical contexts, the resonant orbits levitate above the thin disc plane to form a thick disc compartment, while the non-resonant ones contract to the thin disc; the levitating stellar orbits in a thickened galactic bar has been captured by the resonance of the bar (VLR) and the BP/X morphology is a result of the shapes of the mean-motion resonant and non-resonant stellar orbits (Portail et al., 2015b; Abbott et al., 2017).

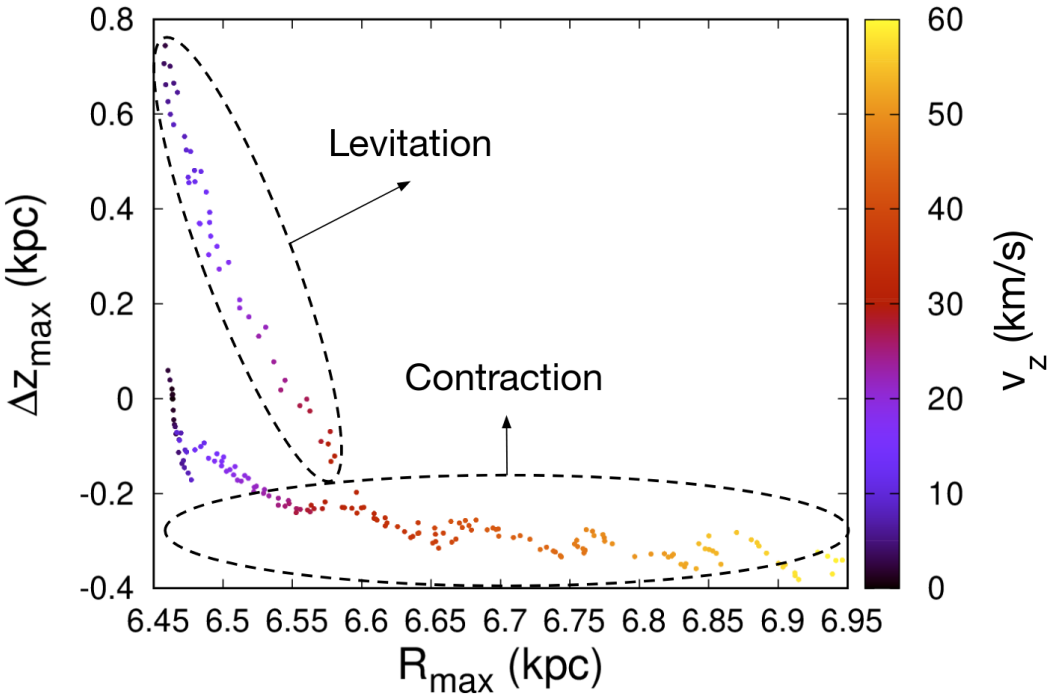

Figure 4.2: Stellar levitation and adiabatic contraction of 600 stars in the R_{max}, Δz_{max} plane. The initial positions and motions of the stellar orbits are the same as described in Figure 4.1.

Despite observing the direct effect of stellar levitation, the result in Figure 4.2 still does not provide us with the full picture. **(1)** The orbits chosen for this simple test are highly eccentric, unlike a typical disc orbit, we are interested in the effect of levitation of disc orbits hence explaining the formation of thick discs. **(2)** The location and passage of the MOR are unclear from the figures. These concerns will be solved in section 4.3. **(3) Why levitation only happens to a minority of stars?** The leading factor in stellar levitation is the MOR. It happens that the MOR passes through a small number of stars' orbits and aids with lifting them above the galactic plane. The remaining problem occurs to when and where the MOR is in the process of levitation (see section 4.3.). **(4) Why the contracted stellar orbits bump up and down?** They are either caused by the rate of growth in the disc potential exceeding the limit of the adiabatic process or due to simulation errors, such as the time evolution period not being long enough.

4.2. Features of 1:1 MOR

In theory, there are an infinite amount of the MOR in the combination of integer ratio. We are particularly interested in the role that MOR plays during the process of stellar levitation. **In this section, we probe into the features of 1:1 MOR in the phase space.**

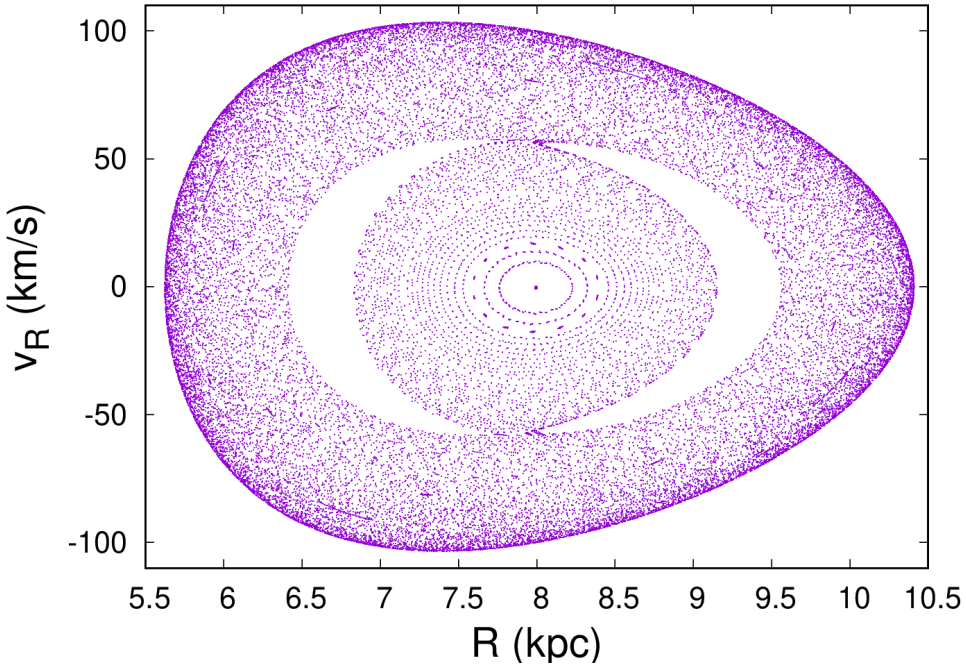

Figure 4.3: The double-moon structured resonance island of 800 stellar orbits with a constant energy equals to -0.13965 kpc^{2}Myr^{-2}. The initial positions of all stars are (8.0, 0, 0) kpc away from the galaxy centre. The initial velocity vector is (v_x, 180.0, v_z) km/s, v_z varies from 100.0 km/s to 0.0 km/s, and v_x changes accordingly to conserve the energy.

4.2.1. Resonance island

To firstly locate the 1:1 mean-motion resonant star's orbit in a constant standard potential, we applied the SOS method (section 3.4.) **to 200 stellar orbits and present the resonance island in a 2-D phase space** (R, v_R)**.** In Figure 4.3, each closed loop represents a stellar orbit in the (R, v_R) phase space without overlapping with the others. With the fixed effective Hamiltonian H_{eff}, the orbits from inside out in spatial order have decreasing initial vertical velocities and increasing initial radial momentum. As a result, the outermost trajectory has the highest initial radial velocity, whereas the innermost one has the lowest.

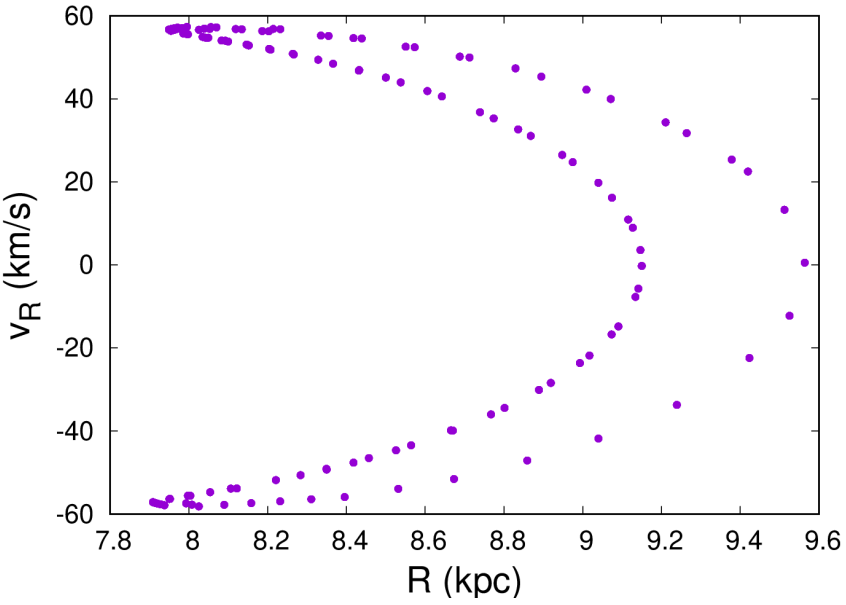

Figure 4.4: SOS of a 1:1 MOR from the previous resonance island. Its initial velocity vector is (57.42, 220.0 81.88) km/s.

Regarding Figure 4.3, there are several interesting questions to ask to fully understand this particular resonance island. **Why is the general distribution of the resonance island oval-shaped?** It is caused by the gravitational potential difference. Gravity is gradually weakened when the orbit gets further away from the galaxy centre (larger R), hence the change in v_R slows down and reaching the slowest at the furthest R. This explains the sharper end of the phase space trajectory. **Why is the SOS of the innermost star's orbit a point?** The point orbit in the centre is called the parent orbit or the circular orbit. This is because there are no perturbed oscillations in the radial direction for a perfectly circular orbit at z = 0 kpc plane, nor the variations in v_R. Therefore, a 2-D loop reduces to a single point in the (R, v_R) plane. This type of orbit represents families of orbits. To be more precise, the rest of the orbits makes up a few families of orbits. In particular the resonant and non-resonant orbits are different families. Therefore, parent orbits are extremely useful to keep track of different orbital families, such as the x1 family tree, which form part of the X morphology in a galactic bar, demonstrates banana shapes like "$\smile$ " or "$\frown$ " when viewed from the side (Pfenniger and Friedli, 1991). **Where do the gaps within the resonance island come from?** Through Method A, Fourier analysis (section 3.3.1.), we obtain the frequency ratio $f_R : f_z$ of each orbit. The 1:1 MOR was verified to cause the gaps in the resonance island and the structure of 1:1 mean-motion resonant stellar orbits is topologically different to the non-resonant ones. As shown by Figure 4.4, the SOS of a 1:1 MOR looks like a crescent moon, with a varying R range and allows for the exchange of the

vertical and radial energies. This explains the double-moon structure of the resonance island in Figure 4.3.

4.2.2. Libration

To have a deeper insight into the properties of the 1:1 MOR, a possibility would be to perceive its dynamical behaviour in an angle-action plane (introduced in section 2.). An important quantity, libration angle ϕ, needs to be defined to form an angle-action coordinate (ϕ, J_R). As discussed in section 2.1., libration occurs when stellar orbits are trapped by the resonance, with an angle range of π in radians.

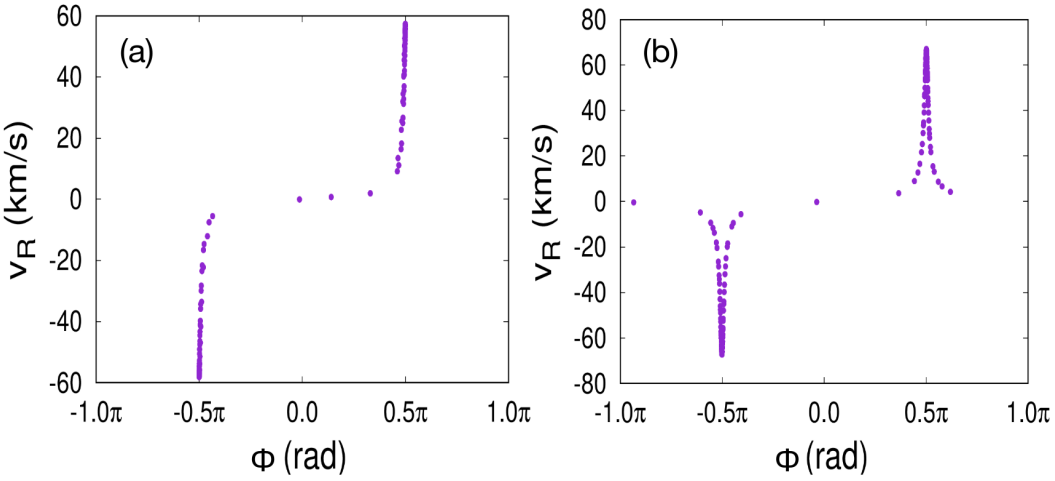

Figure 4.5: Plots of the libration angle ϕ against the radial velocity v_R of (a) a 1:1 MOR; (b) a non-resonant orbit with a frequency ratio $f_R : f_z$ at 0.954545. I remark there is a numerical error in these diagrams because they were simulated without considering the normalisation factor N. Despite the error, they are still useful to understand the libration and circulation ranges of resonant and non-resonant orbits.

ϕ can be approximated as the the angle between the values of R and v_R in the epicycle approximation. Since the parent orbit is the circular orbit and located in the centre of the resonance island. Thereby, regarding the location (R_g) of the circular orbit as the origin, any discrete point in Figure 4.3 can be expressed as in a polar coordinate (r, ϕ). The radial coordinate r between the pole and a discrete point is $\sqrt{(R - R_g)^2 + v_R^2}$, and A is a normalising factor in units of Myr to make the term $\frac{v_R}{R - R_g}$ unit-less.

$$\phi \simeq tan^{-1}\left(A\,\frac{v_R}{R\,-\,R_g}\right) \tag{10}$$

We are particularly interested in the relationship between the radial action and libration angle ϕ of a stellar orbit in 1:1 MOR. An equivalent result can be obtained by considering the correlation between ϕ and radial velocity v_R. In Figure 4.5 (a), the ϕ range of the 1:1 mean-motion resonant orbit varies from -0.5 π to 0.5 π and agrees with the libration range of resonant pendulum's orbits which is demonstrated in section 2.1.. We have also acknowledged that the phase flow of a non-resonant orbit lies outside of the separatrix, with a rotation range of 2.0 π. In Figure 4.5 (b), the range of ϕ covers the 2.0 π period, showing the circulation of a non-resonant orbit.

Having understood the features of a 1:1 MOR in the (ϕ, v_R) plane, we are also interested in the libration in the vertical phase space. In a constant potential, the old action of a resonant orbit forms a new action throughout time. The vertical and radial action get degenerated and form fast and slow actions eventually. The fast action is a linear combination of radial and vertical actions which corresponds to the coupled beat in Figure 3.3. The slow action is libration. **Another important property of libration** is its gradual change on the shape of the stellar orbit over time by exchanging the vertical and radial energies/motions. The effect of libration is clear in Figure 4.6, represented by the systematic shift in colours.

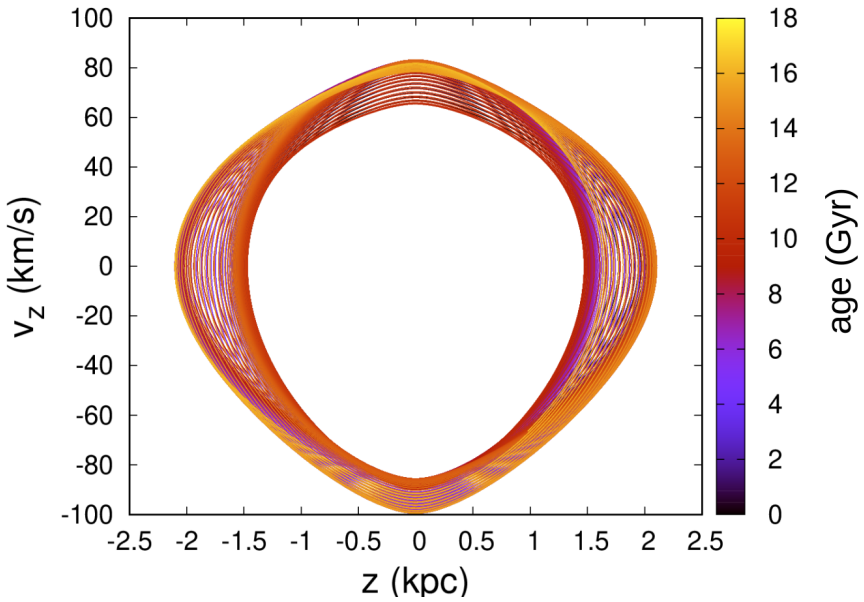

Figure 4.6: The vertical phase space profile $(z : v_z)$ of the same stellar orbit in 1:1 MOR. The colour bars indicate the progress of time in Gyr.

4.3. Levitation of disc orbits

Now we consider the levitation of nearly circular orbits in a growing disc potential using Method B, epicycle frequency analysis (see section (3.3.2)). Method B particularly aids showing the passage of a MOR in the levitation process and the shifts of the ratio of epicycle frequencies.

In Figure 4.7, the disc potential grows from 10 % of a standard potential for the present disc slowly for 18.0 Gyr. This is a long enough period for the disc to grow adiabatically, thus the vertical action of the orbit is conserved. The nearly circular orbit chosen in Figure 4.7 (a) has R_g equals 8.0 kpc and located in the mid-plane of the disc.

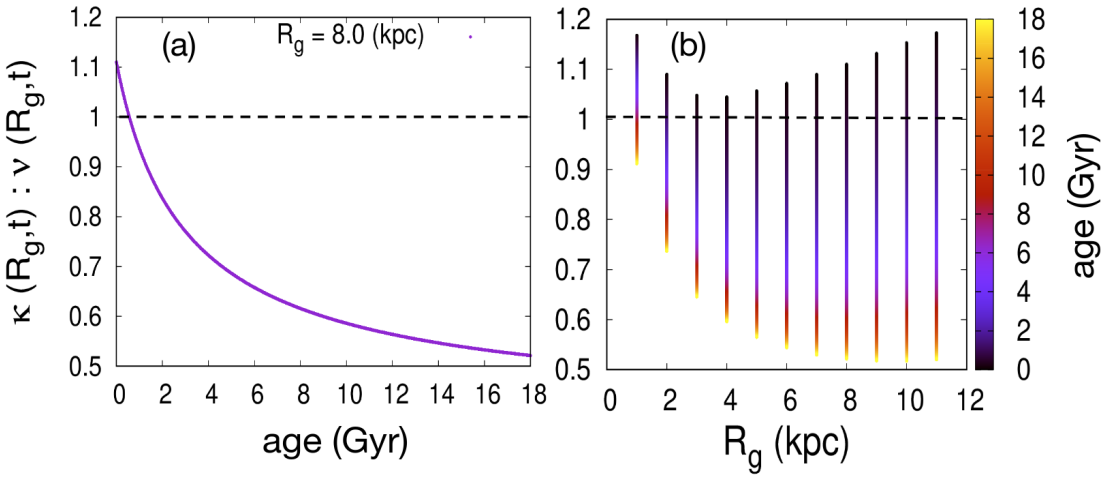

Figure 4.7: Locating and position and passage of the MOR via time-dependent epicycle frequency ratio $\kappa(R_g, t) : \nu(R_g, t)$ in a growing potential. (a) the time against the ratio when R_g = 8.0 kpc. (b) the evolution of the ratio at varies guiding center radius R_g.

The first thing to be aware of from the diagram (a) is the ratio of epicycle frequencies which sits higher above 1.0 (the dashed line) in the very beginning. This is strong evidence of the existence of MOR in the galactic plane before the disc starts growing. Roughly at the dashed line, the time for the epicycle frequency ratio to decrease below 1.0 is the moment when the MOR rises above the mid-pane. For example, the MOR no longer exist in the mid-plane region if the disc potential initially starts growing at 2 Gyrs or so. As the resonance has proved to acquire a finite width, it should stretch a bit into the frequency ratio to be slightly off 1.0. The extent of deviation is up to either analytical calculation or empirical results. In Figure 4.7 (b), when the initial potential of the disc is set to be 10% of the standard potential, disc orbits in range of all radius R_g have passed

the dashed line. This has proven that the MOR should locate at the mid-plane of a disc when its initial potential is weak and rise up above the plane when the disc is growing adiabatically long enough.

The epicycle frequency ratio decreases exponentially over time in both plots, implying the rising of the stellar orbit in the vertical extent. Envisage the vertical potential of a disc as a piece of rubber sheet that can be stretched by putting a heavy mass on its surface. The most stretched or lowest point in the sheet corresponds to the greatest potential in the mid-plane of the disc. Both the vertical disc potential profile and the surface of the sheet get more flattened once away from this deepest point. Now, if we throw a light ball around the central heavy mass, assuming its dynamical behaviour is equivalent to a simple harmonic oscillator, its rate of motion and frequency will keep dropping once moved to a more flattened spot in the sheet or potential. Equivalently, when the epicycle ratio of the nearly circular orbit decreases, this star's orbit arrives at a higher location above the mid-plane where the gravitational potential is weaker. Thus, the dropping of $\kappa(R_g, t) : \nu(R_g, t)$ would cause the MOR to rise above the galactic plane. To conclude, when levitation happens, the MOR has to migrate from a lower vertical action orbit to higher vertical action orbits. Furthermore, if we can evaluate how much faster a vertically extended orbit is compared to the in-plane orbits, it will most likely determine the exact position of the MOR above the galactic plane.

5. CONCLUSION

In the literature presented in the introduction, we have noticed and discussed the similarity in the formations of galactic thick disc and BP/X-shaped galactic bar. Stellar levitation is the process responsible for the occurrence of these phenomena. This paper focuses on finding the key factors that cause or suppress the stellar levitation in disc orbits.

We have adapted and developed multiple analytical and numerical methods to **(1)** study the features of 1:1 mean-motion resonant stellar orbits in the SOS and angle-action coordinate; **(2)** locate the position and observed the passage of MOR in nearly circular (disc) orbits; **(3)** build up the connections between the MOR and stellar levitation. We hence concluded that disc orbits get captured by the MOR which rises above the disc plane in an adiabatically growing disc potential. The resonant stellar orbits increase vertical oscillation ranges by transferring the radial action to the vertical action. Future work is required to investigate how further above a MOR can arise and how much radial action is transferred to the vertical one.

The disc potential is axisymmetic, thereby the angular momentum Lz is conserved. To mimic a slowly growing thin disc, the potential of the disc was simulated to grow linearly and adiabatically over time with a low initial value (e.g. 0% to 10% of the standard disc potential). This linear time-dependent model can be improved by considering the star formation rate, such as the rate of adding stars. A further refinement would involve accounting for the heating of giant molecular clouds (GMC) because they provide the major source of heat for a thin disc. GMC heating is usually more effective at disc's early age due to its short life period (Aumer et al., 2016).

Starting with a simple test on stellar levitation, our results are in agreement with Sridhar and Touma (1996a). Stellar orbits with high initial radial action become resonant in an adiabatically growing potential and subsequently increase their vertical oscillation ranges. Actions of each orbit are conserved since the process is adiabatic. Thus the resonant orbits levitate by feeding on their radial actions. However, the findings in Sridhar and Touma (1996a,b) have left two main problems wide open: **(1)** Is there really MOR moving out of the disc? **(2)** Are resonant orbits just librating horizontal actions into

vertical actions? A potential approach to investigate this issue would involve identifying where the resonance passes in the disc. A series of simulations of the disc stars' orbits around the resonance can be produced to observe if levitation occurs

To locate the position of the MOR in disc orbits while the galactic thin disc is undertaking an adiabatic growth, Method B, epicycle frequency analysis was implemented to find the epicycle frequency ratios $(\kappa(R_g, t) : \nu(R_g, t))$ with respect to time. Highlighted by the results in Figure 4.7, the inital frequency ratio surpasses the dashed line at 1.0, implying the MOR was originally found to sit in the disc plane when the initial potential of the disc was 10% of the standard disc potential. If we start the process with a more massive initial thin disc (roughly more than 20% of the standard disc potential), the initial position of the MOR is more likely to be off the mid-plane, hence stopping the stellar levitation from happening. The decay of $\kappa(R_g, t) : \nu(R_g, t)$ directly shows the rise of MOR above the galactic plane. Specifically, it starts when the ratio is around 1.0.

The features of MOR become clearer through phase space analysis such as looking at the SOS of stellar orbit in the (R, v_R) plane. We tracked the 1:1 MOR which is a double-moon structured resonance island in Figure 4.3 & 4.4. Despite the normalisation error in Figure 4.5, it was found that the orbits trapped by resonance in the (ϕ, v_R) coordinate (which is equivalent to the angle-action coordinate) librates within the separatrix, while the non-resonant ones circulate outside the separatrix.

Further, we were interested whether resonant dragging (section 2.1) happens in the disc plane with our located MOR. Further iterations to refine the approach would involve: Currently, we know that MOR is initially located in the disc plane. By exposing nearly circular orbits next to the MOR, we can determine if they are captured by the resonance. This can be achieved by implementing Method A, Fourier analysis to keep track of the $f_R : f_z$ of stellar orbits until they match with the expected MOR. To ascertain the orbits are trapped by the MOR, they have to complete at least a full cycle of libration. There are two possibilities where an orbit cannot be captured, and they are also the potential factors that suppress the levitation: **(1)** the fast growth of the disc potential. If the growth rate of the disc potential is relatively fast (greater than 10^{-4} Myr^{-1}), the MOR will rise up quickly above the plane, being unable to capture the orbits. The quick decay of

$\kappa(R_g, t) : \nu(R_g, t)$ in Figure 4.7, meaning a quick rise of MOR, may lead to this situation. The solution to this issue is to grow the disc potential under extremely slow routine, with a growth rate less than 10^{-5} Myr^{-1}. **(2)** Orbits can jump over and miss the MOR. To avoid this situation, the stellar orbits that we are testing should have small initial vertical oscillation ranges that are close to the disc plane.

Lastly, the same mechanism happened through the thickening process of a BP/X shaped bar. Instead of a growing disc potential, the bar's potential changes adiabatically while it slows down over time. From the previous literature, stellar orbits levitate by getting captured by the resonance of the bar (VLR). With our findings of MOR, future works are needed to explain the levitation process occurring in the bar as a coupling of both VLR and MOR.

BIBLIOGRAPHY

C. G. Abbott, M. Valluri, J. Shen, and V. P. Debattista. *MNRAS*, 470, 1526–1541, 2017.

E. Athanassoula. *MNRAS*, 341, 1179-1198, 2003.

M. Aumer and R. Schönrich. *MNRAS*, 454, 3166-3184, 2015.

M. Aumer, J. Binney, and R. Schönrich. *MNRAS*, 459, 3326–3348, 2016.

T. Bensby and S. Feltzingand. *MNRAS*, 367, 1181–1193, 2006.

J. Binney. *arXiv e-prints, arXiv1202.3403*, 2012.

J. Binney and S. Tremaine. *Galactic Dynamics. 2nd edn.* Princeton Univ. Press, Princeton, 2008.

M. Bureau, G. Aronica, E. Athanassoula, J. Dettmar, A. Bosma, and K. C. Freeman. *MNRAS*, 370, 753, 2006.

R. Chiba, J. K. S. Friske, and R. Schönrich. *arXiv e-prints, arXiv:1912.04304*, 2019.

F. Combes, F. Debbasch, D. Friedli, and D. Pfenniger. *A&A*, 233, 82, 1990.

G. Contopoulos and M. D. Weinberg. *ApJ*, 201, 566, 1975.

P. Erwin and V. P. Debattista. *MNRAS*, 431, 3060, 2013.

D. Friedli, W. Benz, and R. Kennicutt. *ApJ*, 430, L105, 1994.

G. Gilmore and N. Reid. *MNRAS*, 202, 1025–1047, 1983.

P. Goldreich and S. Peale. *AJ*, 71, 425, 1966.

T. G. Hawarden, C. M. Mountain, S. K. Leggett, and P. J. Puxley. *MNRAS*, 221, 41P-45P, 1986.

L. Hernquist and M. D. Weinberg. *ApJ*, 400, 80, 1992.

E. Hummel, J. M. van-der Hulst, R. C. Kennicutt, and W. C. Keel. *A & A*, 236, 333, 1990.

M. Jurić, Z. Ivezić, A. Brooks, R. H. Lupton, D. Schlegel, D. Finkbeiner, N. padmanabhan, N. Bond, B. Sesar, and C. M. Rockosi. *ApJ*, 673, 864, 2008.

E. Laurikainen, H. Salo, R. Buta, and J. H. Knapen. *MNRAS*, 418, 1452–1490, 2011.

A. J. Lichtenberg and M. A. Lieberman. *Regular and Chaotic Dynamics*. 1992.

P. J. McMillan. *MNRAS*, 465, 76, 2017.

A. McWilliam and M. Zoccali. *ApJ*, 724, 1491-1502, 2010.

D. M. Nataf, A. Udalski, A. Gould, and et al. *ApJ*, 721, L28, 2010.

M. Ness and D. Lang. *ApJ*, 152, 14, 2016.

J. H. Oort. *Bull. Astron. Inst. Neth.*, 6, 249–87, 1932.

D. Pfenniger and D. Friedli. *A&A*, 252, 75, 1991.

M. Portail, C. Wegg, and O. Gerhard. *MNRAS*, 450, L66–L70, 2015b.

A. C. Quillen. *AJ*, 124, 722, 2002.

A. C. Quillen, I. Minchev, and et al. *MNRAS*, 437, 1284, 2014.

S. N. Rasband. *Chaotic Dynamics of Nonlinear Systems*. Wiley-vch, New York, 1990.

R. Roskar, V. P. Debattista, and S. R. Loebman. *MNRAS*, 433, 976–985, 2013.

R. Schönrich and J. Binney. *MNRAS*, 396, 203 (SB09), 2009a.

R. Schönrich and J. Binney. *MNRAS*, 399, 1145–1156, 2009b.

J. A. Sellwood and O. Gerhard. *MNRAS*, 495, 3175–3191, 2020.

J. A. Sellwood and A. Wilkinson. *Rep. Prog. Phys.*, 56, 173, 1993.

M. A. Shaw. *MNRAS*, 229, 691, 1987.

S. Sridhar and J. Touma. *MNRAS*, 271, 973-975, 1996a.

S. Sridhar and J. Touma. *MNRAS*, 279, 1263-1273, 1996b.

M. Vera, S. Alonso, and G. Coldwell. *A & A*, 95, A63, 2016.

M. D. Weinberg. *MNRAS*, 213, 451, 1985.

E. L. Wright, P. R. M. Eisenhardt, R. M. Peter, and et al. *ApJ*, 140, 1868-1881, 2010.

APPENDIX

The attached C++ algorithm in the Appendix is what we have used to generate results for this paper, it consists of a simple leapfrog algorithm, a simple stellar levitation algorithm, a Fourier analysis, a time-dependent epicycle frequency analysis, SOS techniques. I have adapted the simple leapfrog algorithm from Dr. Ralph Schönrich and developed the rest of codes.

```cpp
#include<iostream>
#include<fstream>
#include<string>
#include<functional>
#include<cmath>
#include<random>
#include<thread>
#include<mutex>
#include<algorithm>
#include"./GalPot-master/src/GalPot.h"


#include <math.h>
#include <sstream>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_vector_double.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_matrix_double.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_fft_real.h>
#include <gsl/gsl_fft_halfcomplex.h>
#include <gsl/gsl_spline.h>

#define DIMENSION 3
#define THREADNO 6


#define PI 3.14159265358979323846264338328
#define parsec 3.08567759756e+16
#define VC 239.1
#define RSUN 8.29


#define NBASE   500
#define NPERBASE  10
#define TIMESTEPFAC 0.005        //0.0002   //the time-step factor
#define MINSTEP 0.2  //the minimum step
#define MAXSTEP 200.0



using namespace std;
using std::setprecision;

// parameters of normalization array
#define CVALN 40
#define GVALN 27
#define ZVALN 23                //50
```

```
#define CVL 0.05
#define GVL 0.8
#define ZVL 9.0


#define VPHIPLACE 0.215
#define VPHIPLACESIG 0.045
#define VRPLACEFAC 1.6
#define VZPLACEFAC 2.0
#define RPLACEL 4.0
#define ZPLACEL 0.6
#define RMIN 6.79
#define RMAX 9.79
#define VCG 0.235 //in 1000km/s
#define RCR 6.0 //in kpc

// real and imag for fft
#define REAL(z,i) ((z)[2*(i)])
#define IMAG(z,i) ((z)[2*(i)+1])

#define FROMFILE1 "Halo1.Tpot"
#define FROMFILE2 "Disc1.Tpot"

//Fix the dis potential
#define CONTIME 10000.0
#define FIXTIME 10000.0
#define TIMEXPRESS (FIXTIME/CONTIME)


//global variable problem, instantiating Potential
std::ifstream fromfile("./GalPot-master/pot/PJM17_best.Tpot");
GalaxyPotential PhiGalPot(fromfile);
/*
std::ifstream fromGalPot("./barPJM16_best.Tpot");
GalaxyPotential PhiGalPotBar(fromGalPot);

std::ifstream fromBarsmall("./barsmalla2212Normal.Tpot");
GalaxyPotential PhiBarSmall(fromBarsmall);

std::ifstream fromBarlarge("./barsmalla2212Normal.Tpot");
GalaxyPotential PhiBarLarge(fromBarlarge);

std::ifstream fromBarsmallgrow("./barsmalla2212Normal.Tpot");
GalaxyPotential PhiBarSmallGrow(fromBarsmallgrow);

std::ifstream fromBarlargegrow("./barsmalla2212Normal.Tpot");
GalaxyPotential PhiBarLargeGrow(fromBarlargegrow);
*/


std::mutex mutex_var;

std::random_device random_var;
std::mt19937 randomGen(random_var());
```

```cpp
const double delta = 10e-7;
const double divDelta = 1.0 / delta;


double timefunc(double time) {
    //return 0.1+time/CONTIME;//Growing potential
    return 0.1;//TIMEXPRESS;//Fixed potential, time in units of Myr, when TIMEXPRESS=1.0, we achieve the full MW2017 potential
}

//function for returning the azimuthal angle of the bar
double barphi(double time){
    double Omegabar = VCG/RCR; //in Myr //angular velocity of bar
    return Omegabar*time; //gives the azimuthal angle of bar at the specific time: phi_b
}

//function returning the omega_bar
double barOm(double time){
    const double tinytime = 0.01; //10 thousand years //remember 1 unit is 1 Myr
    //note to me: in the thesis, write down actual timelengths of typical dynamical motions of the milky way

    //gives the angular velocity omega in terms of tinytime (and angular displacement)
    return (barphi(time+tinytime) - barphi(time))/tinytime;
    }

//so that amplitude grows with time
double baramp(double A, double time){
    double growtime = 500.0; //the growth time in units of Myr, can set however i want
    //to slowly grow the amplitude, mimicking a growing bar
    //choose any type of growing function i want
    return 0.0;//A*2.0/PI; //by Simon ;      //*atan(time*time/(growtime*growtime));//*2.0/PI;
}


void fPot(double coord[3], double grad[], GalaxyPotential &pH) {
    double R = std::sqrt(coord[0] * coord[0] + coord[1] * coord[1]);
    double dR; double dz;
    pH(R, coord[2], dR, dz);
        grad[0] = -dR * coord[0] / R;
        grad[1] = -dR * coord[1] / R;
        grad[2] = -dz;
}


void fPot(double coord[3], double grad[], GalaxyPotential &pH, GalaxyPotential &pD, double time) {
    double R = std::sqrt(coord[0] * coord[0] + coord[1] * coord[1]);
    double phi = std::atan2(coord[1], coord[0]);
    double dR; double dz;
    double f2 = timefunc(time);
    pH(R, coord[2], dR, dz);
```

```
    double dR2; double dz2;
    pD(R, coord[2], dR2, dz2);
    grad[0] = -(dR + f2*dR2)*coord[0]/R;
    grad[1] = -(dR + f2*dR2)*coord[1]/R;
    grad[2] = -(dz + f2*dz2);
    //double pot_unper = (pH(R,coord[2]) + f2*pD(R,coord[2]));

    const double A = 0.017; const double b =  0.28; const double vc = VCG; const
    double m = 2.0; const double Rcr = RCR;    //gives the azimuthal angle of bar //
    in rads
    double phib = barphi(time);
    //gives the angle difference between bar and coordinate frame //in rads
    double deltaphi = phi - phib;
    double Phim = - (baramp(A,time)*(vc*vc/m)) * (R/Rcr)*(R/Rcr) * pow(((b + 1.0)/(b
    + R/Rcr)),5.0);
    double potper = Phim*cos(deltaphi*m);
    double gradfR = -potper * ((2.0/R) - (5.0)*((b + R/Rcr)/(b + 1.0))*(1.0/
    Rcr)*(b+1.0)/((b + R/Rcr)*(b+R/Rcr))); //         (1.0/(b+(R/R_cr))));
    double gradfphi = Phim * (m/R) * sin(deltaphi*m);
    double gradfx = (gradfR *coord[0]/R) - gradfphi * coord[1]/R; // x = R*cos(phi) -
    phi*sin(phi)
    double gradfy = (gradfR *coord[1]/R) + gradfphi * coord[0]/R; // y = R*sin(phi) +
    phi*cos(phi)
    grad[0] += gradfx;
    grad[1] += gradfy;
    //grad[2] += gradf_z; // make the bar 3-D
}

void fPot2(double coord[3], double grad[], GalaxyPotential &pH, GalaxyPotential &pD2)
{

    double R = std::sqrt(coord[0] * coord[0] + coord[1] * coord[1]);
    double phi = std::atan2(coord[1], coord[0]);
    double dR; double dz;
    pH(R, coord[2], dR, dz);
    double dR2; double dz2;
    pD2(R, coord[2], dR2, dz2);
    grad[0] = -(dR+dR2 )*coord[0]/R;
    grad[1] = -(dR+dR2 )*coord[1]/R;
    grad[2] = -(dz+dz2 );
}

void fPotL(double coord[3], double grad[], GalaxyPotential &pH, GalaxyPotential &pD,
double time) {
    double R = std::sqrt(coord[0] * coord[0] + coord[1] * coord[1]);
    double phi = std::atan2(coord[1], coord[0]);
    double dR; double dz;
    pH(R, coord[2], dR, dz);
    double dR2; double dz2;
    pD(R, coord[2], dR2, dz2);
    double f2 = 0.1;//+time/CONTIME; // 10 per cent initial mass
    grad[0] = -(dR + f2*dR2)*coord[0]/R;
    grad[1] = -(dR + f2*dR2)*coord[1]/R;
    grad[2] = -(dz + f2*dz2);
}
double gPot(double coord[3], GalaxyPotential &pH1, GalaxyPotential &pH2, double time)
```

```
{
    double R = std::sqrt(coord[0] * coord[0] + coord[1] * coord[1]);
    double phi = std::atan2(coord[1], coord[0]);
    double f2 = timefunc(time);
    const double A = 0.017; const double b =  0.28; const double vc = VCG; const
    double m = 2.0; const double Rcr = RCR; //in kpc
    double phib = barphi(time);
    double deltaphi = phi - phib;
    double Phim = - (baramp(A,time)*(vc*vc/m)) * (R/Rcr)*(R/Rcr) * pow(((b + 1.0)/(b
    + R/Rcr)),5.0);
    double potunper = (pH1(R,coord[2]) + f2*pH2(R,coord[2]));
    double potper = Phim* cos(deltaphi*m);
    return (potper + potunper);
}

double gPotconst(double coord[3], double R, GalaxyPotential &pH1, GalaxyPotential
&pH2) {
    double f2 = 1.0;
    double potunper = (pH1(R,coord[2]) + f2*pH2(R,coord[2]));
    return potunper;
}

double gPottime(double coord[3], double R, GalaxyPotential &pH1, GalaxyPotential
&pH2, double time) {
    double f2 = 0.1 + time/CONTIME;//full potential
    double potunper = (pH1(R,coord[2]) + f2*pH2(R,coord[2]));
    return potunper;
}

double effPot(double coord[3], double R, double lz, GalaxyPotential &pH1,
GalaxyPotential &pH2) {
    double f2 = 1.0;//full potential
    double potunper = (pH1(R,coord[2]) + f2*pH2(R,coord[2]));
    double poteff = potunper + lz*lz/(2*R*R);
    return poteff;
}

double effPottime(double coord[3], double R, double lz, GalaxyPotential &pH1,
GalaxyPotential &pH2, double time) {
    double f2 = 0.1 + time/CONTIME;//full potential
    double potunper = (pH1(R,coord[2]) + f2*pH2(R,coord[2]));
    double poteff = potunper + lz*lz/(2*R*R);
    return poteff;
}


class particle {
public:
    int stepnumber = 0;
    double weight = 1.0;
    double timestep = 1.0;
    double htimestep = 0.5;
    double time = 0.0;
    double timefine = 0.0;
    double x[DIMENSION];
    double p[DIMENSION];
```

```
    double f[DIMENSION];
    void drift();
    double getR();
    double getz();
    double getr();
    double gettime();
    void settime();
    void timefineadd();
    double getE(GalaxyPotential &pH);
    void kick(GalaxyPotential &pH);
    void step(GalaxyPotential &pH);
    void zstep(GalaxyPotential &pH, GalaxyPotential &pD);
    void zkick(GalaxyPotential &pH, GalaxyPotential &pD);
    void zdrift();
    void zleapfrog(GalaxyPotential &pH, GalaxyPotential &pD);
    void getJz(GalaxyPotential &pH, GalaxyPotential &pH2, double JrJz[2]);
    void leapfrog(GalaxyPotential &pH);
    void print(GalaxyPotential &pH);
    void print(GalaxyPotential &pH, ostringstream &ostr);
//    void print(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream &ostr);
    void getsimpleperiods(GalaxyPotential &pH,double periods[2], double maxtime);
    void integrate(double integtime, GalaxyPotential &pH);
    void kick(GalaxyPotential &pH, GalaxyPotential &pD);
    void step(GalaxyPotential &pH, GalaxyPotential &pD);
    void kick2(GalaxyPotential &pH, GalaxyPotential &pD2);
    void step2(GalaxyPotential &pH, GalaxyPotential &pD2);
    void kickL(GalaxyPotential &pH, GalaxyPotential &pD, double time);
    void stepL(GalaxyPotential &pH, GalaxyPotential &pD, double time);
    void leapfrog(GalaxyPotential &pH, GalaxyPotential &pD);
    void leapfrog2(GalaxyPotential &pH, GalaxyPotential &pD2);
    void leapfrogL(GalaxyPotential &pH, GalaxyPotential &pD, double time);
    void print(GalaxyPotential &pH, GalaxyPotential &pD);
    void print(GalaxyPotential &pH, GalaxyPotential &pD, ostringstream &ostr);
//  void Nprint(GalaxyPotential &pH, GalaxyPotential &pD, ostringstream &ostr);
//    void getsimplestartperiods(GalaxyPotential &pH, GalaxyPotential &pD,
periods[2], double maxtime);
    void integrate(double integtime, GalaxyPotential &pH, GalaxyPotential &pD);
    void integrate2(double integtime, GalaxyPotential &pH, GalaxyPotential &pD2);
    void integrateL(double integtime, GalaxyPotential &pH, GalaxyPotential &pD,
    double time);
    void phishift(std::uniform_real_distribution<double> phasedist);
    void phishift(double delphi);
    void phiset(double R, double phi, double vR, double vphi);
    void phitrans(double &R, double &phi, double &vR, double &vphi);
    void clone(particle &pp);
    void epicycle(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream &ostr1,
    ostringstream &ostr21, ostringstream &ostr22, ostringstream &ostr31,
    ostringstream &ostr32);
    void epicycletime(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream
    &ostr1, ostringstream &ostr21, ostringstream &ostr22, ostringstream &ostr31,
    ostringstream &ostr32);
    void fourier(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream &ostr1,
    ostringstream &ostr11, ostringstream &ostr12, ostringstream &ostr2, ostringstream
    &ostr21, ostringstream &ostr3);
    void sos(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream &ostr1,
    ostringstream &ostr11, ostringstream &ostr12, ostringstream &ostr2);
```

```
    void dragging(GalaxyPotential &pH1, GalaxyPotential &pH2, GalaxyPotential &pH3,
    ostringstream &ostr1, ostringstream &ostr2);
    void simplelevitation(GalaxyPotential &pH1, GalaxyPotential &pH2, GalaxyPotential
    &pH3, ostringstream &ostr);
private:
    ;
};

double particle::getz() {
    return x[2];
}

void particle::getJz(GalaxyPotential &pH1, GalaxyPotential &pH2, double JRJz[2]) {
    JRJz[0] = 0.0; JRJz[1] = 0.0;
    double z0 = x[2];
    double z = x[2];
    double zold = z0 - p[2];
    double zold2 = zold - p[2];
    double delz = z- zold;
    double timestepsave = timestep;
    for (int i = 0; i < 2; i++) {
      zleapfrog(pH1, pH2);
        zold2 = zold;
        z = x[2];
        delz = z - zold;
        JRJz[1] += delz*p[2];
        zold = z;
//        cout << "1 " << " " << htimestep << " " << zold << " " << zold2 << " " <<
x[2] << " " << p[2] << "  " << JRJz[1] << "\n";
    }
    while (((zold - z0)*(zold2 - z0)) >= 0.0) {
        zleapfrog(pH1, pH2);
        zold2 = zold;
        z = x[2];
        delz = z - zold;
        JRJz[1] += delz*p[2];
        zold = z;
//        cout << "1 " << " " << htimestep << " " << zold << " " << zold2 << " " <<
x[2] << " " << p[2] << "  " << JRJz[1] << "\n";
    }
    for (int i = 0; i < 3; i++) {
       zleapfrog(pH1, pH2);
        zold2 = zold;
        z = x[2];
        delz = z - zold;
        JRJz[1] += delz*p[2];
        zold = z;
//        cout << "2 " << " " << x[2] << " " << p[2] << "  " << JRJz[1] << "\n";
    }
    while (((zold - z0)*(zold2 - z0)) >= 0.0) {
        zleapfrog(pH1, pH2);
        zold2 = zold;
        z = x[2];
        delz = z - zold;
        JRJz[1] += delz*p[2];
        zold = z;
```

```
//        cout << "3 " <<  " " << x[2] << " " << p[2] << "  " << JRJz[1] << "\n";
    }
    timestep = timestepsave;
}

void particle::zleapfrog(GalaxyPotential &pH, GalaxyPotential &pD) {
    double grad[3];
    for (int i = 0; i < 3; ++i) {
        grad[i] = 0.0;
    }
    fPot(x, grad, pH, pD, gettime());
    double acc = std::sqrt(grad[2] * grad[2]);
    double df = (f[2] - grad[2])*(f[2] - grad[2]);
    df = 10.0*sqrt(df)/acc;
    df = 1.0/(df + 1.0e-14);
//    cout << 0.0001 << " " << df << " " << sqrt(q2)/acc << " " << 1.0/acc << "\n";
    if (1.0/acc < df) {df = 1.0/acc;}
//        else {
//            cout << "dfrule " << df << " " << 1.0/acc << "\n";
//        }
    if (df > MINSTEP) {
//        cout << "WARNING " << df << " " << x[0] << " " << x[1] << " " << x[2] << "
" << 1.0/acc << "\n";
        df = MINSTEP;
    }
    double tnew = TIMESTEPFAC*df;
    if ((tnew > timestep*2.0) || (tnew < htimestep)){
        if (stepnumber > 2) {
            if (tnew < htimestep) {
//                cout << "WARNING tnew " << tnew << " " << timestep << " " <<
stepnumber << "\n";
            }
        }
        if (tnew > timestep) {
            tnew = timestep*1.2;
        }
    }
    timestep = tnew;
    htimestep = 0.5*timestep;
    for (int j = 0; j < 4; j++) {
        zstep(pH, pD);
    }
}

void particle::timefineadd() {
    if (timefine > 0.00001*time) {
        time += timefine; //adding timefine to time
        timefine = 0.0;
    }
}

double particle::getR() {
    return sqrt(x[0]*x[0] + x[1]*x[1]);
}

double particle::getr() {
```

```
    return sqrt(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);
}


double particle::gettime() {
    return (time + timefine);
}

void particle::settime() {
    time = 0.0;
    timefine = 0.0;
    stepnumber = 0;
}

void particle::kick(GalaxyPotential &pH) {
    fPot(x, f, pH);
    for (int i = 0; i < DIMENSION; i++) {
        p[i] += timestep*f[i];
    }
}

void particle::kick(GalaxyPotential &pH, GalaxyPotential &pD) {
    fPot(x, f, pH, pD, gettime());
    for (int i = 0; i < DIMENSION; i++) {
        p[i] += timestep*f[i];
    }
}

void particle::kick2(GalaxyPotential &pH, GalaxyPotential &pD2) {
    fPot2(x, f, pH, pD2);
    for (int i = 0; i < DIMENSION; i++) {
        p[i] += timestep*f[i];
    }
}

void particle::kickL(GalaxyPotential &pH, GalaxyPotential &pD, double time) {
    fPotL(x, f, pH, pD, time);
    for (int i = 0; i < DIMENSION; i++) {
        p[i] += timestep*f[i];
    }
}

void particle::zkick(GalaxyPotential &pH, GalaxyPotential &pD) {
    fPot(x, f, pH, pD, time);
        p[2] += timestep*f[2];
}


void particle::zdrift() {
        x[2] += htimestep*p[2];
}


void particle::drift() {
    for (int i = 0; i < DIMENSION; i++) {
        x[i] += htimestep*p[i];
```

```
    }
}

void particle::step(GalaxyPotential &pH) {
    drift();
    timefine += htimestep;
    kick(pH);
    drift();
    timefine += htimestep;
}

void particle::zstep(GalaxyPotential &pH, GalaxyPotential &pD) {
    zdrift();
    timefine += htimestep;
    zkick(pH, pD);
    zdrift();
    timefine += htimestep;
}


void particle::step(GalaxyPotential &pH, GalaxyPotential &pD) {
    drift();
    timefine += htimestep;
    kick(pH, pD);
    drift();
    timefine += htimestep;
}

void particle::step2(GalaxyPotential &pH, GalaxyPotential &pD2) {
    drift();
    timefine += htimestep;
    kick2(pH, pD2);
    drift();
    timefine += htimestep;
}

void particle::stepL(GalaxyPotential &pH, GalaxyPotential &pD, double time) {
    drift();
    timefine += htimestep;
    kickL(pH, pD, time);
    drift();
    timefine += htimestep;
}

void particle::leapfrog(GalaxyPotential &pH) {
    stepnumber++;
    double grad[3];
    for (int i = 0; i < 3; ++i) {
        grad[i] = 0.0;
    }

        //difference here is the fPot taking 5 arguments
    //taking in the coordinates, force, pH, pD, and gettime
    fPot(x, grad, pH);
        //the total acceleration of all 3 space-dimension
    double acc = (grad[0] * grad[0] + grad[1] * grad[1] + grad[2] * grad[2]);
```

```
    double sacc = sqrt(acc);

    double q2 = 0.0;//.....................................................
    double df = 0.0;
    for (int j = 0; j < DIMENSION; j++) { //for each coordinates
        q2 += x[j]*x[j]; //position^2
        df += (f[j] - grad[j])*(f[j] - grad[j]);//force difference ^2
    }//.....................................................................
    df = sqrt(df/acc); //some sort of normalisation
    double tssug = (1.0/(10.0*df + 1.0e-14 + sacc + (1.0/MAXSTEP))) + MINSTEP;

    double tnew = TIMESTEPFAC*tssug;
    if ((tnew > timestep*2.0) || (tnew < htimestep)){ // if ... OR ...
        if (stepnumber > 2) {
            if (tnew < htimestep) {
                double saccold = sqrt(f[0]*f[0] + f[1]*f[1] + f[2]*f[2]);
                cout << "WARNING tnew " << tnew << " " << timestep << " " << tssug <<
                " " << 1.0/sacc << " " << 1.0/saccold << " " << 1.0/(10.0*df) << " "
                << stepnumber << "\n";
            }
        }
        if (tnew > timestep) {
            tnew = timestep*1.2;
        }
    }
    timestep = tnew;
    htimestep = 0.5*timestep;
    //cout << gettime() << " "  << timestep << " " << df << " " << acc << " " <<
    sqrt(x[0]*x[0] + x[1]*x[1]) << "\n";
//.....................................................................
    //implementing the step function
    for (int j = 0; j < 8; j++) {
        step(pH);  //here calling the step function with 2 potentials
    }
    if (timefine > 0.00001*time) {
        time += timefine;
        timefine = 0.0;
    }
}

void particle::leapfrog(GalaxyPotential &pH, GalaxyPotential &pD) {
    stepnumber++;
    double grad[3];
    for (int i = 0; i < 3; ++i) {
        grad[i] = 0.0;
    }

        //difference here is the fPot taking 5 arguments
    //taking in the coordinates, force, pH, pD, and gettime
    fPot(x, grad, pH, pD, gettime());
        //the total acceleration of all 3 space-dimension
    double acc = (grad[0] * grad[0] + grad[1] * grad[1] + grad[2] * grad[2]);
    double sacc = sqrt(acc);

    double q2 = 0.0;//.....................................................
```

```
    double df = 0.0;
    for (int j = 0; j < DIMENSION; j++) { //for each coordinates
        q2 += x[j]*x[j]; //position^2
        df += (f[j] - grad[j])*(f[j] - grad[j]);//force difference ^2
    }//.....................................................................
    df = sqrt(df/acc); //some sort of normalisation
    double tssug = (1.0/(10.0*df + 1.0e-14 + sacc + (1.0/MAXSTEP))) + MINSTEP;

    double tnew = TIMESTEPFAC*tssug;
    if ((tnew > timestep*2.0) || (tnew < htimestep)){ // if ... OR ...
        if (stepnumber > 2) {
            if (tnew < htimestep) {
                double saccold = sqrt(f[0]*f[0] + f[1]*f[1] + f[2]*f[2]);
                cout << "WARNING tnew " << tnew << " " << timestep << " " << tssug <<
                " " << 1.0/sacc << " " << 1.0/saccold << " " << 1.0/(10.0*df) << " "
                << stepnumber << "\n";
            }
        }
        if (tnew > timestep) {
            tnew = timestep*1.2;
        }
    }
    timestep = tnew;
    htimestep = 0.5*timestep;
    //cout << gettime() << " "  << timestep << " " << df << " " << acc << " " <<
    sqrt(x[0]*x[0] + x[1]*x[1]) << "\n";
//.....................................................................
.................
    //implementing the step function
    for (int j = 0; j < 8; j++) {
        step(pH, pD);  //here calling the step function with 2 potentials
    }
    timefineadd();

}//
=============================================================================
==================

void particle::leapfrog2(GalaxyPotential &pH, GalaxyPotential &pD2) {
    stepnumber++;
    double grad[3];
    for (int i = 0; i < 3; ++i) {
        grad[i] = 0.0;
    }

        //difference here is the fPot taking 5 arguments
    //taking in the coordinates, force, pH, pD, and gettime
    fPot2(x, grad, pH, pD2);
        //the total acceleration of all 3 space-dimension
    double acc = (grad[0] * grad[0] + grad[1] * grad[1] + grad[2] * grad[2]);
    double sacc = sqrt(acc);

    double q2 = 0.0;//.....................................................
    double df = 0.0;
    for (int j = 0; j < DIMENSION; j++) { //for each coordinates
        q2 += x[j]*x[j]; //position^2
```

```
        df += (f[j] - grad[j])*(f[j] - grad[j]);//force difference ^2
    }//.....................................................................
    df = sqrt(df/acc); //some sort of normalisation
    double tssug = (1.0/((10.0*df + 1.0e-14 + sacc + (1.0/MAXSTEP))) + MINSTEP;

    double tnew = TIMESTEPFAC*tssug;
    if ((tnew > timestep*2.0) || (tnew < htimestep)){ // if ... OR ...
        if (stepnumber > 2) {
            if (tnew < htimestep) {
                double saccold = sqrt(f[0]*f[0] + f[1]*f[1] + f[2]*f[2]);
                cout << "WARNING tnew " << tnew << " " << timestep << " " << tssug <<
                " " << 1.0/sacc << " " << 1.0/saccold << " " << 1.0/(10.0*df) << " "
                << stepnumber << "\n";
            }
        }
        if (tnew > timestep) {
            tnew = timestep*1.2;
        }
    }
    timestep = tnew;
    htimestep = 0.5*timestep;
    //cout << gettime() << " "  << timestep << " " << df << " " << acc << " " <<
    sqrt(x[0]*x[0] + x[1]*x[1]) << "\n";
//.....................................................................
.....................
    //implementing the step function
    for (int j = 0; j < 8; j++) {
        step2(pH, pD2);  //here calling the step function with 2 potentials
    }
    timefineadd();

}//
=====================================================================
==================

void particle::leapfrogL(GalaxyPotential &pH, GalaxyPotential &pD, double time) {
    stepnumber++;
    double grad[3];
    for (int i = 0; i < 3; ++i) {
        grad[i] = 0.0;
    }

        //difference here is the fPot taking 5 arguments
    //taking in the coordinates, force, pH, pD, and gettime
    fPotL(x, grad, pH, pD, time);
        //the total acceleration of all 3 space-dimension
    double acc = (grad[0] * grad[0] + grad[1] * grad[1] + grad[2] * grad[2]);
    double sacc = sqrt(acc);

    double q2 = 0.0;//.....................................................
    double df = 0.0;
    for (int j = 0; j < DIMENSION; j++) { //for each coordinates
        q2 += x[j]*x[j]; //position^2
        df += (f[j] - grad[j])*(f[j] - grad[j]);//force difference ^2
    }//.....................................................................
    df = sqrt(df/acc); //some sort of normalisation
```

```
    double tssug = (1.0/(10.0*df + 1.0e-14 + sacc + (1.0/MAXSTEP))) + MINSTEP;

    double tnew = TIMESTEPFAC*tssug;
    if ((tnew > timestep*2.0) || (tnew < htimestep)){ // if ... OR ...
        if (stepnumber > 2) {
            if (tnew < htimestep) {
                double saccold = sqrt(f[0]*f[0] + f[1]*f[1] + f[2]*f[2]);
                cout << "WARNING tnew " << tnew << " " << timestep << " " << tssug <<
                " " << 1.0/sacc << " " << 1.0/saccold << " " << 1.0/(10.0*df) << " "
                << stepnumber << "\n";
            }
        }
        if (tnew > timestep) {
            tnew = timestep*1.2;
        }
    }
    timestep = tnew;
    htimestep = 0.5*timestep;
    //cout << gettime() << " "  << timestep << " " << df << " " << acc << " " <<
    sqrt(x[0]*x[0] + x[1]*x[1]) << "\n";
//.....................................................................................
    //implementing the step function
    for (int j = 0; j < 8; j++) {
        stepL(pH, pD, time);  //here calling the step function with 2 potentials
    }
    timefineadd();

}//
=============================================================================================

double particle::getE(GalaxyPotential &pH) {
    double Ekin = 0.0;
    for (int i = 0; i < 3; i++) {
        Ekin += 0.5*p[i]*p[i];
    }
    return (Ekin + pH(sqrt(x[0]*x[0] + x[1]*x[1]), x[2]));
}


void particle::print(GalaxyPotential &pH, ostringstream &ostr) {
    double Ekin = 0.0;
    for (int i = 0; i < 3; i++) {
        ostr << x[i] << " " << p[i] << " " << f[i] << " ";
        Ekin += 0.5*p[i]*p[i];
    }
    double R,phi,vR,vphi;
    phitrans(R,phi,vR,vphi);
    ostr << R << " " << phi << " " << vR << " " << vphi << " " << stepnumber << " ";
    ostr << timestep << " " << htimestep << " " << Ekin << " " << pH(R, x[2]) << " "
    << pH(R, 0.0) << " " << time+timefine;
    ostr << "\n";


}
```

```
void particle::print(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream &ostr)
{
    double Ekin = 0.0;
    for (int i = 0; i < 3; i++) {
        ostr << x[i] << " " << p[i] << " " << f[i] << " ";
        Ekin += 0.5*p[i]*p[i];
    }
    double R,phi,vR,vphi;
    phitrans(R,phi,vR,vphi);
    double coord[3];
    coord[0] = x[0]; coord[1] = x[1]; coord[2] = 0.0;
    ostr << R << " " << phi << " " << vR << " " << vphi << " " << stepnumber << " ";
    ostr << timestep << " " << htimestep << " " << Ekin << " " << gPot(x, pH1, pH2,
    gettime()) << " " << gPot(coord, pH1, pH2, gettime()) << " " << gettime();
    ostr << "\n";
}

void particle::epicycle(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream
&ostr1, ostringstream &ostr21, ostringstream &ostr22, ostringstream &ostr31,
ostringstream &ostr32){
    #define NR 1500 // number of R
    #define NZ 200// number of Z
    #define NRG 12 // number of Rg

   //double Rg = 8.0;
    double delR = 0.001;
    double delz = 0.0001;
    double delRg = 0.1;
/*    // for circular velocity
    double pot [NRG][NR];
    double vc [NRG];
    double dpotdR [NRG][NR];
    // for epicycle frequencies
    double Rg [NRG];
    double Lz [NRG];
    double R [NRG][NR];
    double effpot [NRG][NR];
    double dphidR [NRG][NR];
    double kap [NRG][NR];

    // not working
    double z [NRG][NZ];
    double effpotz [NRG][NZ];
    double dphidz [NRG][NZ];
    double nu [NRG][NZ];
*/
    double **pot = new double*[NRG];   //the stars mean pointers, the **lvplane is a
    pointer to pointers which gets initialised as two-hundred pointers
    double *vc = new double[NRG];
    double **dpotdR = new double*[NRG];
    double *Rg = new double[NRG];
    double *Lz = new double[NRG];
    double **R = new double*[NRG];
    double **effpot = new double*[NRG];
    double **dphidR = new double*[NRG];
```

```
    double **kap = new double*[NRG];
    double **z = new double*[NRG];
    double **potz = new double*[NRG];
    double **dphidz = new double*[NRG];
    double **nu = new double*[NRG];
    for (int i = 0; i < NRG; i++) { //this is a loop using the index i running from 0
to 199
        pot[i] = new double[NR];
        dpotdR[i] = new double[NR];
        R[i] = new double[NR];
        effpot[i] = new double[NR];
        dphidR[i] = new double[NR];
        kap[i] = new double[NR];
        z[i] = new double[NZ];
        potz[i] = new double[NZ];
        dphidz[i] = new double[NZ];
        nu[i] = new double[NZ];
    }
    // find circular velocities vc(Rg)
    int i; int j;
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (j = 0; j < NR; j++){
            settime();
            x[0] = 0.0 + delR* j; x[1] = 0.0; x[2] = 0.0;
            double r = std::sqrt(x[0] * x[0] + x[1] * x[1]);
            pot[i][j] = gPotconst(x, r, pH1, pH2);
        }
    }
    for (i = 0; i < NRG; i++){
        for (j = 0; j < NR-1; j++){
            dpotdR[i][j] = (pot[i][j+1] - pot[i][j])/delR;

        }
    }
    for (i = 0; i < NRG; i++){
        int nRg = Rg[i]/delR;
        vc[i]=std::sqrt(Rg[i]*dpotdR[i][nRg]);
        ostr1 << Rg[i] << " "<< vc[i] << " ";
        ostr1<<"\n";
    }

    ////////////////////////////////////////////
    // radial epicycle frequency
    for (i = 0; i < NRG; i++){
        Lz[i]= Rg[i] * vc[i];//(8.29kpc*(233.14/977.77)kpc/Myr)
        for (j = 0; j < NR; j++){
            settime();
            x[0] = 0.0 + delR* j; x[1] = 0.0; x[2] = 0.0;
            double r = std::sqrt(x[0] * x[0] + x[1] * x[1]);
            double lz = Lz [i];
            R[i][j] = r;
            effpot[i][j] = effPot(x, r, lz, pH1, pH2);
            ostr21<<R[i][j]<<" "<<effpot[i][j]<<" ";
            ostr21<<"\n";
        }
```

```
    }


    //differentiation to find kap
    for (i = 0; i < NRG; i++){
        for (j = 0; j < NR-1; j++){
            dphidR[i][j] = (effpot[i][j+1] - effpot[i][j])/delR;
            //ostr2 << dphidR [i][j] << " ";
            //ostr2<<"\n";
        }
    }
    for (i = 0; i < NRG; i++){
        for (j = 0; j < NR-1; j++){
            kap[i][j] = std::sqrt((dphidR[i][j+1] - dphidR[i][j])/delR);
            //ostr2 << kap[i][j] << " ";
            //ostr2<<"\n";
        }
    }
    for (i = 0; i < NRG; i++){
        int nRg = Rg[i]/delR;
        ostr22 << Rg[i] << " "<< kap[i][nRg] << " ";
        ostr22<<"\n";
    }

    ////////////////////////////////////////////
    // vertical epicycle frequency
    //double coord[3];
    for (i = 0; i < NRG; i++){
        for (j = 0; j < NZ; j++){
            settime();
            x[0] = Rg[i]; x[1] = 0.0; x[2] = -0.1+delz * ((double)(j));
            double r = std::sqrt(x[0] * x[0] + x[1] * x[1]);
            z[i][j] = x[2];
            //coord[0] = Rg[i]; coord[1] = 0.0; coord[2] =0.0;
            potz[i][j] = gPotconst(x, r, pH1, pH2);//-gPotconst(coord, r, pH1, pH2);
            ostr31<<z[i][j]<<" "<<potz[i][j]<<" ";
            ostr31<<"\n";
        }
    }

    //differentiation to find nu
    //#define NR 1500 // number of R #define NZ 1000 // number of Z #define NRG 120
    for (i = 0; i < NRG; i++){
        for (j = 0; j < NZ-1; j++){
            dphidz[i][j] = (potz[i][j+1] - potz[i][j])/delz;
            //ostr32 << dphidz [i][j] << " ";
            //ostr32<<"\n";
        }
    }


    for (i = 0; i < NRG; i++){
        for (j = 0; j < NZ-1; j++){
            nu[i][j] =std::sqrt(fabs(dphidz[i][j+1] - dphidz[i][j])/delz);

            //ostr32 << nu[i][j] << " ";
```

```
            //ostr32<<"\n";
        }
    }


    int nz = NZ*delz/2; // find nz when z=0

    for (i = 0; i < NRG; i++){
        int nRg = Rg[i]/delR;
        ostr32 << Rg[i] << " "<< kap[i][nRg]<< " "<<nu[i][nz] << " ";
        ostr32<<"\n";
    }


    for (i = 0; i < NRG; i++){

        delete [] pot[i];       //free all memory at all pointers in lvplane
        delete [] dpotdR[i];
        delete [] R[i];
        delete [] effpot[i];
        delete [] dphidR[i];
        delete [] kap[i];
        delete [] z[i];
        delete [] potz[i];
        delete [] dphidz [i];
        delete [] nu[i];

        }

    delete [] pot;      //free all memory at all pointers in lvplane
    delete [] vc;
    delete [] dpotdR;
    delete [] Rg;
    delete [] Lz;
    delete [] R;
    delete [] effpot;
    delete [] dphidR;
    delete [] kap;
    delete [] z;
    delete [] potz;
    delete [] dphidz ;
    delete [] nu;

}


void particle::epicycletime(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream
&ostr1, ostringstream &ostr21, ostringstream &ostr22, ostringstream &ostr31,
ostringstream &ostr32){
    #define A 9000

/*

    double T [NRG][A];
    // for circular velocity
    double pot [NRG][A][NR];
```

```
    double vc [NRG][A];
    double dpotdR [NRG][A][NR];
    // for epicycle frequencies

    double Lz [NRG][A];
    double R [NRG][A][NR];
    double effpot [NRG][A][NR];
    double dphidR [NRG][A][NR];
    double kap [NRG][A][NR];

    double z [NRG][A][NZ];
    double effpotz [NRG][A][NZ];
    double dphidz [NRG][A][NZ];
    double nu [NRG][A][NZ];
*/
    //#define NR 1500 // number of R #define NZ 200 // number of Z
    //double Rg = 8.0;
    double delRg = 1.0;
    double delR = 0.01;
    double delz = 0.001;
    double deltime = 2.0;

    double *Rg = new double[NRG];
    double ***pot = new double**[NRG];   //the stars mean pointers, the **lvplane is
    a pointer to pointers which gets initialised as two-hundred pointers
    double **vc = new double*[NRG];
    double ***dpotdR = new double**[NRG];
    double **T = new double*[NRG];
    double **Lz = new double*[NRG];
    double ***R = new double**[NRG];
    double ***effpot = new double**[NRG];
    double ***dphidR = new double**[NRG];
    double ***kap = new double**[NRG];
    double ***z = new double**[NRG];
    double ***effpotz = new double**[NRG];
    double ***dphidz = new double**[NRG];
    double ***nu = new double**[NRG];
    int i; int ii; int j;
    for (int i = 0; i < NRG; i++) {
        pot[i] = new double*[A];
        vc[i] = new double[A];
        dpotdR[i] = new double*[A];
        T[i] = new double[A];
        Lz[i] = new double[A];
        R[i] = new double*[A];
        effpot[i] = new double*[A];
        dphidR[i] = new double*[A];
        kap[i] = new double*[A];
        z[i] = new double*[A];
        effpotz[i] = new double*[A];
        dphidz[i] = new double*[A];
        nu[i] = new double*[A];
        for (int ii = 0; ii < A; ii++) { //this is a loop using the index i running
        from 0 to 199
            pot[i][ii] = new double[NR];
            dpotdR[i][ii] = new double[NR];
```

```
            R[i][ii] = new double[NR];
            effpot[i][ii] = new double[NR];
            dphidR[i][ii] = new double[NR];
            kap[i][ii] = new double[NR];
            z[i][ii] = new double[NZ];
            effpotz[i][ii] = new double[NZ];
            dphidz[i][ii] = new double[NZ];
            nu[i][ii] = new double[NZ];
        }
    }

    // find circular velocities vc(Rg)

    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (ii = 0; ii < A; ii++){
            int time = deltime*ii;
            T[i][ii] = time;
            for (j = 0; j < NR; j++){
                settime();
                x[0] = 0.0 + delR* j; x[1] = 0.0; x[2] = 0.0;
                double r = std::sqrt(x[0] * x[0] + x[1] * x[1]);
                pot[i][ii][j] = gPottime(x, r, pH1, pH2, time);
                //ostr1 << pot[i][ii][j] << " ";
                //ostr1<<"\n";
            }
        }
    }


    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            for (j = 0; j < NR-1; j++){
                dpotdR[i][ii][j] = (pot[i][ii][j+1] - pot[i][ii][j])/delR;

            }
        }
    }
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (ii = 0; ii < A; ii++){
            int nRg = Rg[i]/delR;
            vc[i][ii]=std::sqrt(Rg[i]*dpotdR[i][ii][nRg]);
            ostr1 << Rg[i] << " "<< T[i][ii] << " "<< vc[i][ii] << " ";
            ostr1<<"\n";
        }
    }

     ////////////////////////////////////////
    // radial epicycle frequency
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (ii = 0; ii < A; ii++){
            Lz[i][ii]= Rg[i] * vc[i][ii];//(8.29kpc*(233.14/977.77)kpc/Myr)
            int time = deltime*ii;
            for (j = 0; j < NR; j++){
```

```
                settime();
                x[0] = 0.0 + delR* j; x[1] = 0.0; x[2] = 0.0;
                double r = std::sqrt(x[0] * x[0] + x[1] * x[1]);
                double lz = Lz [i][ii];
                R[i][ii][j] = r;
                effpot[i][ii][j] = effPottime(x, r, lz, pH1, pH2, time);
                //ostr21<<R[i][ii][j]<<" "<<effpot[i][ii][j]<<" ";
                //ostr21<<"\n";
            }
        }

    }
    //differentiation to find kap
    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            for (j = 0; j < NR-1; j++){
                dphidR[i][ii][j] = (effpot[i][ii][j+1] - effpot[i][ii][j])/delR;
                //ostr2 << dphidR [i][ii][j] << " ";
                //ostr2<<"\n";
            }
        }
    }
    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            for (j = 0; j < NR-1; j++){
                kap[i][ii][j] = std::sqrt((dphidR[i][ii][j+1] - dphidR[i][ii][j])/
                delR);
                //ostr2 << kap[i][ii][j] << " ";
                //ostr2<<"\n";
            }
        }
    }
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (ii = 0; ii < A; ii++){
            int time = deltime*ii;
            T[i][ii] = time;
            int nRg = Rg[i]/delR;
            ostr22 << T[i][ii] << " "<< kap[i][ii][nRg] << " ";
            ostr22<<"\n";
        }
    }


    ////////////////////////////////////////////
    // vertical epicycle frequency
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
        for (ii = 0; ii < A; ii++){
            Lz[i][ii]= Rg[i] * vc[i][ii];//(8.29kpc*(233.14/977.77)kpc/Myr)
            int time = deltime*ii;
            for (j = 0; j < NZ; j++){
                settime();
                x[0] = Rg[i]; x[1] = 0.0; x[2] = -0.1+delz * j;
                z[i][ii][j]= x[2];
                double lz = Lz [i][ii];
```

```
                double r = x[0];
                effpotz[i][ii][j] = effPottime(x, r, lz, pH1, pH2, time);
                //ostr21<<z[i][ii][j]<<" "<<effpotz[i][ii][j]<<" ";
                //ostr21<<"\n";
            }
        }

    }
/*
    for (i = 0; i < 1; i++){
        Rg[i]=8.0;
        for (ii = 0; ii < A; ii++){
            Lz[i][ii]= Rg[i] * vc[i][ii];//(8.29kpc*(233.14/977.77)kpc/Myr)
            int time = deltime*ii;
            T[i][ii] = time;
            for (j = 0; j < NZ; j++){
                settime();
                x[0] = Rg[i]; x[1] = 0.0; x[2] = -0.1+delz * j;
                z[i][ii][j]= x[2];
                double lz = Lz [i][ii];
                double r = x[0];
                effpotz[i][ii][j] = effPottime(x, r, lz, pH1, pH2, time);
                ostr31<<T[i][ii]<<" "<<z[i][ii][j]<<" "<<effpotz[i][ii][j]<<" ";
                ostr31<<"\n";
            }
        }

    }

*/
    //differentiation to find nu
    //#define NR 1500 // number of R #define NZ 1000 // number of Z #define NRG 120
    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            for (j = 0; j < NZ-1; j++){
                dphidz[i][ii][j] = (effpotz[i][ii][j+1] - effpotz[i][ii][j])/delz;
                //ostr2 << dphidz[i][ii][j] << " ";
                //ostr2<<"\n";
            }
        }
    }
    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            for (j = 0; j < NZ-1; j++){
                nu[i][ii][j] = std::sqrt((dphidz[i][ii][j+1] - dphidz[i][ii][j])/
delz);
                //ostr2 << nu[i][ii][j] << " ";
                //ostr2<<"\n";
            }
        }
    }


    int nz = NZ*delz/2; // find nz when z=0
    for (i = 0; i < NRG; i++){
        Rg[i]=i*delRg;
```

```
        for (ii = 0; ii < A; ii++){
            int nRg = Rg[i]/delR;
            ostr31 << Rg[i] << " "<< T[i][ii] << " "<< kap[i][ii][nRg]<< " "<<nu[i]
        [ii][nz] << " ";
            ostr31<<"\n";
        }
    }

    for (i = 0; i < NRG; i++){
        Rg[i]=8.0;
        for (ii = 0; ii < A; ii++){
            int nRg = Rg[i]/delR;
            ostr32 << Rg[i] << " "<< T[i][ii] << " "<< kap[8][ii][nRg]<< " "<<nu[8]
        [ii][nz] << " ";
            ostr32<<"\n";
        }
    }


    for (i = 0; i < NRG; i++){
        for (ii = 0; ii < A; ii++){
            delete [] pot[i][ii];      //free all memory at all pointers in lvplane
            delete [] dpotdR[i][ii];
            delete [] R[i][ii];
            delete [] effpot[i][ii];
            delete [] dphidR[i][ii];
            delete [] kap[i][ii];
            delete [] z[i][ii];
            delete [] effpotz[i][ii];
            delete [] dphidz[i][ii];
            delete [] nu[i][ii];

        }
        delete [] pot[i];      //free all memory at all pointers in lvplane
        delete [] vc[i];
        delete [] dpotdR[i];
        delete [] T[i];
        delete [] Lz[i];
        delete [] R[i];
        delete [] effpot[i];
        delete [] dphidR[i];
        delete [] kap[i];
        delete [] z[i];
        delete [] effpotz[i];
        delete [] dphidz[i];
        delete [] nu[i];

    }

    delete [] pot;      //free all memory at all pointers in lvplane
    delete [] vc;
    delete [] dpotdR;
    delete [] T;
    delete [] Rg;
    delete [] Lz;
    delete [] R;
```

```
    delete [] effpot;
    delete [] dphidR;
    delete [] kap;
    delete [] z;
    delete [] effpotz;
    delete [] dphidz ;
    delete [] nu;

}
void particle::fourier(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream
&ostr1, ostringstream &ostr11, ostringstream &ostr12,ostringstream &ostr2,
ostringstream &ostr21, ostringstream &ostr3){
    //Instructions for the gnuplot:
    //(printorbit1) 1:5=time againsts r; 1:6=time againsts z
    //(resonance) plot "printfourier" u 1:2 title "r" with lines, "printfourier" u 1:
    ($3*5.0) title "z" with lines
    //(Phase space) plot "printorbit1" u 7:9 title "" with lines
    //(surface of sections) plot "printorbit1" u 5:7:9 title "" with lines palette,
    "SOS" u 5:10:9 title "" with points palette
    #define M 16384
    double settime();
    //generate the data of an orbital trajectory
    double t[M];
    double rinitial[M];
    double zinitial[M];
    double periods[2];
    double maxtime = 10000.0;
    double magnitude=std::sqrt(M);

    //Potential
    double coord[3];
    coord[0] = 8.0; coord[1] = 0.0; coord[2] = 0.0;//The velocities of an orbit
    required to find the potential
    double potential = gPot(coord, pH1, pH2, gettime());
    //Energy
    double px = 0.1; double py = 0.22; double pz = 0.0; // The velocities of an orbit
    required to find E
    double E = 0.5*(px*px+py*py+pz*pz) + potential; //Energy is constant.
    E=0.5*(vx^2+vy^2+vz^2)+potential

    double number = 1.0;//100.0;
    double del = 0.000125;// delta is the difference between each orbit in p[0]
    //near-circular orbit in 10% of McMillian17 disc, x is in kpc, p is in Mm/s
    //fix the x position and vary the radial velocity
    x[0] = 8.0; x[1] = 0.0; x[2] = 0.0;

    p[0] = 0.02; p[1] =0.22; p[2] = 0.02; //resonant
/*
    double potential1=gPot(x, pH1, pH2, gettime());

    p[1] = py;
    p[2] = sqrt((E-potential1)*2.0 -p[1]*p[1]-0.0)-del*145.0;//-del*145.0 (1:1 MOR) -
    del*585.0 (0.664921 2:3 MOR)
    //-del*585.0 (0.664921 2:3 MOR), -del*578.0 (0.668421), -del*566.0 (0.679144)
    p[0] = sqrt((E-potential1)*2.0 -p[1]*p[1] -p[2]*p[2]); //sqrt((E-potential)*2.0 -
    p[1]*p[1])
```

```
*/


    // intital conditions for the surface of sections
    double oldtime=0.0;
    double oldx=x[0];
    double oldy=x[1];
    double oldz=x[2];
    double oldvx = p[0];
    double oldvy = p[1];
    double oldvz = p[2];
    double oldR,oldphi,oldvR,oldvphi;
    phitrans(oldR,oldphi,oldvR,oldvphi);
    double oldlibratephi = 4.0;//atan2(1000.0*oldvR,(oldR-8.0));

    t[0]=0.0;
    rinitial[0]=x[0];
    zinitial[0]=0.0;

    double Ri,phii,vRi,vphii;
    phitrans(Ri,phii,vRi,vphii);
    ostr1<<0.0<<" "<<x[0]<<" "<< 0.0<<" "<<0.0<<" ";
    ostr1<<Ri<<" "<<x[2]<<" "<<vRi<<" "<<p[0]<<" "<<p[2]<<" "<<vphii<<" ";
    ostr1 << "\n";
    integrate(maxtime, pH1, pH2);
    for (int i = 1; i < M; i++) {
        for (int ii = 0; ii < 1; ii++) {
            leapfrog(pH1, pH2);
        }

        double R,phi,vR,vphi;
        phitrans(R,phi,vR,vphi);
        double libratephi = atan2(1000.0*vR,(R-8.0)); // angle of libration
        double vx = p[0];
        double vy = p[1];
        double vz = p[2];
        t[i] = gettime()-10000.0;
        rinitial[i] = R;
        zinitial[i] = x[2];

        // ostri1 is the printorbit1
        ostr1<<gettime()-10000.0<<" "<<x[0]<<" "<< x[1]<<" "<<x[2]<<" ";
        ostr1<<rinitial[i]<<" "<<zinitial[i]<<" "<<vR<<" "<<vx<<" "<<vz<<" "<<vphi<<"
    ";
        ostr1 << "\n";

        // detect the change of sign
        if (oldz*x[2] < 0) {
            if (x[2] >= 0){ // Select z with positive sign

            //ostr11 is the SOS, surface of sections
            ostr11<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<" "<<R<<"
    "<<phi<<" "; // positions
            ostr11<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "<<libratephi<<" ";
    // velocities
            ostr11 << "\n";
```

```
            }
                if (p[2] >= 0){
                        //ostr12 is the SOS of positive/rising vz
                        ostr12<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                        "<<R<<" "<<phi<<" "; // positions
                        ostr12<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<"
                        "<<libratephi<<" "; // velocities
                        ostr12<<i+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                    }
            else {
                ostr11<<oldtime<<" "<<oldx<<" "<<oldy<<" "<<oldz<<" "<<oldR<<"
                "<<oldphi<<" "; // positions
                ostr11<<oldvx<<" "<<oldvy<<" "<<oldvz<<" "<<oldvR<<" "<<oldvphi<<"
                "<<oldlibratephi<<" "; // velocities
                ostr11 << "\n";
            }
                if (p[2] >= 0){
                        //ostr12 is the SOS of positive/rising vz
                        ostr12<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                        "<<R<<" "<<phi<<" "; // positions
                        ostr12<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<"
                        "<<oldlibratephi<<" "; // velocities
                        ostr12<<i+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                    }
        }
        oldtime=gettime()-10000.0;oldx=x[0]; oldy=x[1]; oldz=x[2]; oldR=R;
        oldphi=phi;
        oldvx=vx; oldvy=vy; oldvz=vz; oldvR=vR; oldvphi=vphi;
    }

    //interpolate between points to have fixed time intervals
    int i=0;
    double totaltime = gettime()-10000.0;
    double delta=(totaltime)/M;
    double ti, ri, zi;
    double rfinal[2*M];
    double zfinal[2*M];
    double thetafinal[2*M];//theta=arctan(z/r)

    gsl_interp_accel *accr
      = gsl_interp_accel_alloc ();
    gsl_spline *spliner
      = gsl_spline_alloc (gsl_interp_cspline, M);
    gsl_interp_accel *accz
      = gsl_interp_accel_alloc ();
    gsl_spline *splinez
      = gsl_spline_alloc (gsl_interp_cspline, M);

    gsl_spline_init (spliner, t, rinitial, M);
    gsl_spline_init (splinez, t, zinitial, M);

    // test for interpolations for r and z
    for (ti = t[0]; ti < t[M-1]; ti += delta)
      {
        ri = gsl_spline_eval (spliner, ti, accr);
        zi = gsl_spline_eval (splinez, ti, accz);
```

```
    }


  //ostr2 gives the interpolation
  for (int i = 0; i < M; i++) {
      ti=1.0*delta*i;
      REAL(rfinal,i) = gsl_spline_eval (spliner, ti, accr); IMAG(rfinal,i) = 0.0;
      REAL(zfinal,i) = gsl_spline_eval (splinez, ti, accz); IMAG(zfinal,i) = 0.0;
      REAL(thetafinal,i) = atan2(REAL(zfinal,i),REAL(rfinal,i)); IMAG(thetafinal,i)
      = 0.0;
      //ostr2<<rfinal<<" "<<zfinal<<" ";
      ostr2<<i*delta<<" "<<REAL(rfinal,i)<<" "<<REAL(zfinal,i)<<"
      "<<REAL(thetafinal,i)<<" ";
      ostr2 << "\n";
    }
  //free the storage
  gsl_spline_free (spliner);
  gsl_interp_accel_free (accr);
  gsl_spline_free (splinez);
  gsl_interp_accel_free (accz);


  //Fourier analysis
  //test1: fourier transform the fist half, then fourier transform the late half.
  //Fourier analysis
  gsl_fft_complex_radix2_forward (rfinal, delta, M);
  gsl_fft_complex_radix2_forward (zfinal, delta, M);
  gsl_fft_complex_radix2_forward (thetafinal, delta, M);

  // store fr, fz, magnitudes
  double frequency[M];
  double sqrtmagr[M];
  double sqrtmagz[M];
  double sqrtmagtheta[M];

  //find maximum magnitude
  double magr[M];
  double magz[M];
  double magtheta[M];
  double magrmax=0.0;
  double magzmax=0.0;
  double magthetamax=0.0;
  double nr=0.0;//the location when magrmax occurs
  double nz=0.0;//the location when magzmax occurs
  double ntheta=0.0;//the location when magthetamax occurs

  //Find the maximum magnitudes and corresponding frequencies to understand the
  MOR in the first half
  for (i = 1; i < M/2; i++){
      magr[i]=sqrt(REAL(rfinal,i)*REAL(rfinal,i)+IMAG(rfinal,i)*IMAG(rfinal,i));
      magz[i]=sqrt(REAL(zfinal,i)*REAL(zfinal,i)+IMAG(zfinal,i)*IMAG(zfinal,i));
      magtheta[i]=sqrt(REAL(thetafinal,i)*REAL(thetafinal,i)
      +IMAG(thetafinal,i)*IMAG(thetafinal,i));

      if (magr[i] > magrmax) {
          magrmax = magr[i];
```

```
            nr = i;
        }

        if (fabs(magz[i]) > magzmax) {
            magzmax = fabs(magz[i]);
            nz = i;
        }
        if (fabs(magtheta[i]) > magthetamax) {
            magthetamax = fabs(magtheta[i]);
            ntheta = i;
        }
        if (i==(M/2-1)){
            double fr = nr*1.0/(M);
            double fz = nz*1.0/(M);
            double ftheta = ntheta*1.0/(M);
            ostr21 << fr << " "<< ftheta<< " ";
            ostr21<< magrmax/M << " "<< magthetamax/M << " "<< (nr/nz)<< " ";
            //<< magrmax/M << " " << magzmax/M << " "<< magthetamax/M << " ";
            ostr21 << "\n";
        }
    }

    //Fourier shift (centre)
    int j;
    for (i = 0; i < M; i++)
    {
        // shifting the second half to the negative side
        if (i<M/2){
            j = i+M/2;
            sqrtmagr[j]=sqrt(REAL(rfinal,j)*REAL(rfinal,j)
            +IMAG(rfinal,j)*IMAG(rfinal,j));
            sqrtmagz[j]=sqrt(REAL(zfinal,j)*REAL(zfinal,j)
            +IMAG(zfinal,j)*IMAG(zfinal,j));
            sqrtmagtheta[j]=sqrt(REAL(thetafinal,j)*REAL(thetafinal,j)
            +IMAG(thetafinal,j)*IMAG(thetafinal,j));
            frequency[j]=-1.0+j*1.0/(M);
        }
        else{
            j= i-M/2;
            sqrtmagr[j]=sqrt(REAL(rfinal,j)*REAL(rfinal,j)
            +IMAG(rfinal,j)*IMAG(rfinal,j));
            sqrtmagz[j]=sqrt(REAL(zfinal,j)*REAL(zfinal,j)
            +IMAG(zfinal,j)*IMAG(zfinal,j));
            sqrtmagtheta[j]=sqrt(REAL(thetafinal,j)*REAL(thetafinal,j)
            +IMAG(thetafinal,j)*IMAG(thetafinal,j));
            frequency[j]=j*1.0/(M);
        }
        ostr3<<frequency[j]<<" "<<sqrtmagr[j]/M<<" "<<sqrtmagz[j]/M<<"
        "<<sqrtmagtheta[j]/M<<" ";
        ostr3 << "\n";
    }

}

void particle::sos(GalaxyPotential &pH1, GalaxyPotential &pH2, ostringstream
&ostr1,ostringstream &ostr11,ostringstream &ostr12, ostringstream &ostr2){
```

```
    //(surface of sections) plot "printorbit1" u 5:7:9 title "" with lines palette,
"SOSN" u 5:10:9 title "" with points palette

    double t[M];
    double rinitial[M];
    double xinitial[M];
    double zinitial[M];
    double rfinal[2*M];
    double xfinal[2*M];
    double zfinal[2*M];
    double thetafinal[2*M];//theta=arctan(z/r)

    int i;int j;
    double number = 200.0;//100.0;
    double del = 0.0003;// delta is the difference between each orbit in p[0]

    for (j = 0; j < number; j++){

        settime();
        double periods[2];
        double maxtime = 10000.0;
        t[0]=0.0;

        //constant potential, from maximum pz to maximum pR
        x[0] = 4.0; x[1] = 0.0; x[2] = 0.0;
        double potential1=gPot(x, pH1, pH2, gettime());
        p[1] = 0.18;
        p[2] = del*j;
        p[0] = 0.08;
        //Potential
        double potential = gPot(x, pH1, pH2, gettime());
        //Energy
        double E = 0.5*(p[0]*p[0]+p[1]*p[1]+p[2]*p[2]) + potential; //Energy is
constant. E=0.5*(vx^2+vy^2+vz^2)+potential

        rinitial[0]=x[0];
        zinitial[0]=x[2];

        ////////////////////the surface of sections////////////////////
        // intital conditions
        double oldtime=0.0;
        double oldx=x[0];
        double oldy=x[1];
        double oldz=x[2];
        double oldvx = p[0];
        double oldvy = p[1];
        double oldvz = p[2];
        double oldR,oldphi,oldvR,oldvphi;
        phitrans(oldR,oldphi,oldvR,oldvphi);
        ostr11<<E<<" "<<potential1<<" "<<p[0]<<" "<<p[1]<<" "<<p[2]<<" "<< "\n";

        integrate(maxtime, pH1, pH2);
        for (int i = 1; i < M; i++) {
            for (int ii = 0; ii < 1; ii++) {
                leapfrog(pH1, pH2);
            }
```

```
        double R,phi,vR,vphi;
        phitrans(R,phi,vR,vphi);
        double vx = p[0];
        double vy = p[1];
        double vz = p[2];
        double Lz = vphi; // Lz is the angular momentum p_phi or v_phi
        t[i] = gettime()-10000.0;
        rinitial[i]=R;
        zinitial[i]=x[2];

        // detect the change of sign
        if (oldz*x[2] < 0) {
            if (x[2] >= 0){ // Select z with positive sign

                //ostr1 is the SOS, surface of sections
                ostr1<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                "<<R<<" "<<phi<<" "; // positions
                ostr1<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                velocities
                ostr1<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
            }
                if (p[2] >= 0){
                    //ostr12 is the SOS of positive/rising vz
                    ostr12<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                    "<<R<<" "<<phi<<" "; // positions
                    ostr12<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                    velocities
                    ostr12<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                }

            else {
                ostr1<<oldtime<<" "<<oldx<<" "<<oldy<<" "<<oldz<<" "<<oldR<<"
                "<<oldphi<<" "; // positions
                ostr1<<oldvx<<" "<<oldvy<<" "<<oldvz<<" "<<oldvR<<" "<<oldvphi<<"
                "; // velocities
                ostr1<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
            }
                if (p[2] >= 0){
                    //ostr12 is the SOS of positive/rising vz
                    ostr12<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                    "<<R<<" "<<phi<<" "; // positions
                    ostr12<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                    velocities
                    ostr12<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                }
        }
        oldtime=gettime()-10000.0;oldx=x[0]; oldy=x[1]; oldz=x[2]; oldR=R;
        oldphi=phi;
        oldvx=vx; oldvy=vy; oldvz=vz; oldvR=vR; oldvphi=vphi;
    }

    ////////////////////Fourier analysis and find the MOR////////////////////

    //Interpolation
    gsl_interp_accel *accr
```

```
        = gsl_interp_accel_alloc ();
        gsl_spline *spliner
        = gsl_spline_alloc (gsl_interp_cspline, M);
        gsl_interp_accel *accz
        = gsl_interp_accel_alloc ();
        gsl_spline *splinez
        = gsl_spline_alloc (gsl_interp_cspline, M);


        gsl_spline_init (spliner, t, rinitial, M);
        gsl_spline_init (splinez, t, zinitial, M);

        double totaltime = gettime()-10000.0;
        double delta=(totaltime)/M;
        for (int i = 0; i < M; i++) {
            double ti=1.0*delta*i;
            REAL(rfinal,i) = gsl_spline_eval (spliner, ti, accr); IMAG(rfinal,i) =
            0.0;
            REAL(zfinal,i) = gsl_spline_eval (splinez, ti, accz); IMAG(zfinal,i) =
            0.0;
            REAL(thetafinal,i) = atan2(REAL(zfinal,i),REAL(rfinal,i));
            IMAG(thetafinal,i) = 0.0;
        }

        //free the storage
        gsl_spline_free (spliner);
        gsl_interp_accel_free (accr);
        gsl_spline_free (splinez);
        gsl_interp_accel_free (accz);


        //Fourier analysis
        //test1: fourier transform the fist half, then fourier transform the late
        half.
        //Fourier analysis
        gsl_fft_complex_radix2_forward (rfinal, delta, M);
        gsl_fft_complex_radix2_forward (zfinal, delta, M);
        gsl_fft_complex_radix2_forward (thetafinal, delta, M);

        // store fr, fz, magnitudes
        double frequency[M];
        double sqrtmagr[M];
        double sqrtmagz[M];
        double sqrtmagtheta[M];

        //find maximum magnitude
        double magr[M];
        double magz[M];
        double magtheta[M];
        double magrmax=0.0;
        double magzmax=0.0;
        double magthetamax=0.0;
        double nr=0.0;//the location when magrmax occurs
        double nz=0.0;//the location when magzmax occurs
        double ntheta=0.0;//the location when magthetamax occurs
```

```
        //Find the maximum magnitudes and corresponding frequencies to understand the
        MOR in the first half
        for (i = 1; i < M/2; i++){
            magr[i]=sqrt(REAL(rfinal,i)*REAL(rfinal,i)
            +IMAG(rfinal,i)*IMAG(rfinal,i));
            magz[i]=sqrt(REAL(zfinal,i)*REAL(zfinal,i)
            +IMAG(zfinal,i)*IMAG(zfinal,i));
            magtheta[i]=sqrt(REAL(thetafinal,i)*REAL(thetafinal,i)
            +IMAG(thetafinal,i)*IMAG(thetafinal,i));

            if (magr[i] > magrmax) {
                magrmax = magr[i];
                nr = i;
            }

            if (fabs(magz[i]) > magzmax) {
                magzmax = fabs(magz[i]);
                nz = i;
            }
            if (fabs(magtheta[i]) > magthetamax) {
                magthetamax = fabs(magtheta[i]);
                ntheta = i;
            }
            if (i==(M/2-1)){
                double fr = nr*1.0/(M);
                double fz = nz*1.0/(M);
                ostr2 << fr << " "<< fz<< " ";
                ostr2<< magrmax/M << " "<< magthetamax/M << " "<< (nr/nz)<< " ";
                //<< magrmax/M << " " << magzmax/M << " "<< magthetamax/M << " ";
                ostr2 << "\n";
            }
        }


    }
}
void particle::dragging(GalaxyPotential &pH1, GalaxyPotential &pH2, GalaxyPotential
&pH3,ostringstream &ostr1,ostringstream &ostr2){

    int M1 = 2048;//4096.0;
    int M2 = 4096;
    double t1[M1];
    double rinitial1[M1];
    double xinitial1[M1];
    double yinitial1[M1];
    double zinitial1[M1];
    double vxinitial1[M1];
    double vyinitial1[M1];
    double vzinitial1[M1];
    double rfinal1[2*M1];
    double xfinal1[2*M1];
    double yfinal1[2*M1];
    double zfinal1[2*M1];
    double vxfinal1[2*M1];
    double vyfinal1[2*M1];
    double vzfinal1[2*M1];
```

```
    double thetafinal1[2*M1];//theta=arctan(z/r)

    double periods[2];
    double maxtime = 10000.0;
    t1[0]=0.0;
    //constant potential, from maximum pz to maximum pR
    x[0] = 8.0; x[1] = 0.0; x[2] = 0.0;
    double potential1=gPot(x, pH1, pH2, gettime());
    p[0] = 0.01;
    p[2] = 0.01 ;
    p[1] = 0.22;
    //ostr2<<p[1]<<" ";
    //ostr2 << "\n";
    rinitial1[0]=x[0];
    xinitial1[0]=x[0];
    yinitial1[0]=x[1];
    zinitial1[0]=x[2];
    vxinitial1[0]=p[0];
    vyinitial1[0]=p[1];
    vzinitial1[0]=p[2];
    integrate(maxtime, pH1, pH2);
    for (int i = 1; i < M1; i++) {
        for (int ii = 0; ii < 1; ii++) {
            leapfrog(pH1, pH2);
        }

        double R,phi,vR,vphi;
        phitrans(R,phi,vR,vphi);
        t1[i] = gettime()-10000.0;
        rinitial1[i]=R;
        xinitial1[i]=x[0];
        yinitial1[i]=x[1];
        zinitial1[i]=x[2];
        vxinitial1[i]=p[0];
        vyinitial1[i]=p[1];
        vzinitial1[i]=p[2];
    }

    ////////////////////Interpolation////////////////////

    gsl_interp_accel *accr1
    = gsl_interp_accel_alloc ();
    gsl_spline *spliner1
    = gsl_spline_alloc (gsl_interp_cspline, M1);
        gsl_interp_accel *accx1
    = gsl_interp_accel_alloc ();
    gsl_spline *splinex1
    = gsl_spline_alloc (gsl_interp_cspline, M1);
        gsl_interp_accel *accy1
    = gsl_interp_accel_alloc ();
    gsl_spline *spliney1
    = gsl_spline_alloc (gsl_interp_cspline, M1);
    gsl_interp_accel *accz1
    = gsl_interp_accel_alloc ();
    gsl_spline *splinez1
    = gsl_spline_alloc (gsl_interp_cspline, M1);
```

```
    gsl_interp_accel *accvx1
= gsl_interp_accel_alloc ();
gsl_spline *splinevx1
= gsl_spline_alloc (gsl_interp_cspline, M1);
    gsl_interp_accel *accvy1
= gsl_interp_accel_alloc ();
gsl_spline *splinevy1
= gsl_spline_alloc (gsl_interp_cspline, M1);
    gsl_interp_accel *accvz1
= gsl_interp_accel_alloc ();
gsl_spline *splinevz1
= gsl_spline_alloc (gsl_interp_cspline, M1);


gsl_spline_init (spliner1, t1, rinitial1, M1);
gsl_spline_init (splinex1, t1, xinitial1, M1);
gsl_spline_init (spliney1, t1, yinitial1, M1);
gsl_spline_init (splinez1, t1, zinitial1, M1);
gsl_spline_init (splinevx1, t1, vxinitial1, M1);
gsl_spline_init (splinevy1, t1, vyinitial1, M1);
gsl_spline_init (splinevz1, t1, vzinitial1, M1);

double totaltime1 = gettime()-10000.0;
double delta1=(totaltime1)/M1;
for (int i = 0; i < M1; i++) {
    double ti1=1.0*delta1*i;
    REAL(rfinal1,i) = gsl_spline_eval (spliner1, ti1, accr1); IMAG(rfinal1,i) =
    0.0;
    REAL(xfinal1,i) = gsl_spline_eval (splinex1, ti1, accx1); IMAG(xfinal1,i) =
    0.0;
    REAL(yfinal1,i) = gsl_spline_eval (spliney1, ti1, accy1); IMAG(yfinal1,i) =
    0.0;
    REAL(zfinal1,i) = gsl_spline_eval (splinez1, ti1, accz1); IMAG(zfinal1,i) =
    0.0;
    REAL(vxfinal1,i) = gsl_spline_eval (splinevx1, ti1, accvx1); IMAG(vxfinal1,i)
    = 0.0;
    REAL(vyfinal1,i) = gsl_spline_eval (splinevy1, ti1, accvy1); IMAG(vyfinal1,i)
    = 0.0;
    REAL(vzfinal1,i) = gsl_spline_eval (splinevz1, ti1, accvz1); IMAG(vzfinal1,i)
    = 0.0;
    REAL(thetafinal1,i) = atan2(REAL(zfinal1,i),REAL(rfinal1,i));
    IMAG(thetafinal1,i) = 0.0;
    //ostr1<<i*delta1<<" "<<REAL(rfinal1,i)<<" "<<REAL(zfinal1,i)<<"
    "<<REAL(thetafinal1,i)<<" ";
    //ostr1<<i*delta1<<" "<<sqrt(REAL(xfinal1,i)*REAL(xfinal1,i)
    +REAL(yfinal1,i)*REAL(yfinal1,i))<<" ";
    //ostr1<<REAL(xfinal1,i)<<" "<<REAL(yfinal1,i)<<" "<<REAL(zfinal1,i)<<"
    "<<REAL(vxfinal1,i)<<" "<<REAL(vyfinal1,i)<<" "<<REAL(vzfinal1,i)<<" ";
    //ostr1 << "\n";
}

//free the storage
gsl_spline_free (spliner1);
gsl_interp_accel_free (accr1);
gsl_spline_free (splinex1);
gsl_interp_accel_free (accx1);
```

```
    gsl_spline_free (spliney1);
    gsl_interp_accel_free (accy1);
    gsl_spline_free (splinez1);
    gsl_interp_accel_free (accz1);
        gsl_spline_free (splinevx1);
    gsl_interp_accel_free (accvx1);
        gsl_spline_free (splinevy1);
    gsl_interp_accel_free (accvy1);
        gsl_spline_free (splinevz1);
    gsl_interp_accel_free (accvz1);

    ////////////////////////////////////////////////////////////////////////////////
    double t[M1];
    double rinitial[M1];
    double zinitial[M1];
    double rfinal[2*M1];
    double zfinal[2*M1];
    double thetafinal[2*M1];//theta=arctan(z/r)
    // intital conditions
    double oldtime=0.0;
    double oldx=x[0];
    double oldy=x[1];
    double oldz=x[2];
    double oldvx = p[0];
    double oldvy = p[1];
    double oldvz = p[2];
    double oldR,oldphi,oldvR,oldvphi;
    phitrans(oldR,oldphi,oldvR,oldvphi);
    int i;int j;
    for (j = 1; j < M1; j++){
        settime();
        double periods[2];
        double maxtime = 10000.0;
        double resolution = delta1*j;
        x[0] = REAL(xfinal1,j); x[1] = REAL(yfinal1,j); x[2] = REAL(zfinal1,j);
        p[0] = REAL(vxfinal1,j); p[1] = REAL(vyfinal1,j); p[2] = REAL(vzfinal1,j);
        double R = sqrt(x[0]*x[0] + x[1]*x[1]);
        double potential = gPot(x, pH1, pH2, gettime());
        ostr2<<resolution<<" "<<potential<<" ";
        ostr2<<R<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<" ";
        //ostr2 << "\n";
        rinitial[0]=sqrt(x[0]*x[0] + x[1]*x[1]);
        zinitial[0]=x[2];

        integrateL(maxtime, pH1, pH2,resolution);
        for (int i = 1; i < M1; i++) {
            for (int ii = 0; ii < 1; ii++) {
                leapfrogL(pH1, pH2, resolution);
            }

            t[i] = gettime()-10000.0;
            rinitial[i]=sqrt(x[0]*x[0] + x[1]*x[1]);
            zinitial[i]=x[2];
            ////////////////SOS/////////////////////////////////
            double R,phi,vR,vphi;
            phitrans(R,phi,vR,vphi);
```

```
            double vx = p[0];
            double vy = p[1];
            double vz = p[2];
            double Lz = vphi; // Lz is the angular momentum p_phi or v_phi

            // detect the change of sign
            if (oldz*x[2] < 0) {
                if (x[2] >= 0){ // Select z with positive sign

                    //ostr1 is the SOS, surface of sections
                    //ostr1<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                "<<R<<" "<<phi<<" "; // positions
                    //ostr1<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                velocities
                    //ostr1<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                }
                    if (p[2] >= 0){
                        //ostr12 is the SOS of positive/rising vz
                        ostr1<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                    "<<R<<" "<<phi<<" "; // positions
                        ostr1<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                    velocities
                        ostr1<<j+1.0<< "\n";
                    }

                else {
                    //ostr1<<oldtime<<" "<<oldx<<" "<<oldy<<" "<<oldz<<" "<<oldR<<"
                "<<oldphi<<" "; // positions
                    //ostr1<<oldvx<<" "<<oldvy<<" "<<oldvz<<" "<<oldvR<<"
                "<<oldvphi<<" "; // velocities
                    //ostr1<<j+1.0<<" "<<E<<" "<<potential<<" "<< "\n";
                }
                    if (p[2] >= 0){
                        //ostr12 is the SOS of positive/rising vz
                        ostr1<<gettime()-10000.0<<" "<<x[0]<<" "<<x[1]<<" "<<x[2]<<"
                    "<<R<<" "<<phi<<" "; // positions
                        ostr1<<vx<<" "<<vy<<" "<<vz<<" "<<vR<<" "<<vphi<<" "; //
                    velocities
                        ostr1<<j+1.0<< "\n";
                    }
            }
            oldtime=gettime()-10000.0;oldx=x[0]; oldy=x[1]; oldz=x[2]; oldR=R;
            oldphi=phi;
            oldvx=vx; oldvy=vy; oldvz=vz; oldvR=vR; oldvphi=vphi;
        }
        ////////////////////Fourier analysis and find the MOR//////////////////////

        //Interpolation
        gsl_interp_accel *accr
        = gsl_interp_accel_alloc ();
        gsl_spline *spliner
        = gsl_spline_alloc (gsl_interp_cspline, M1);
        gsl_interp_accel *accz
        = gsl_interp_accel_alloc ();
        gsl_spline *splinez
        = gsl_spline_alloc (gsl_interp_cspline, M1);
```

```
        gsl_spline_init (spliner, t, rinitial, M1);
        gsl_spline_init (splinez, t, zinitial, M1);

        double totaltime = gettime()-10000.0;
        double delta=(totaltime)/M1;
        for (int i = 0; i < M1; i++) {
            double ti=1.0*delta*i;
            REAL(rfinal,i) = gsl_spline_eval (spliner, ti, accr); IMAG(rfinal,i) =
            0.0;
            REAL(zfinal,i) = gsl_spline_eval (splinez, ti, accz); IMAG(zfinal,i) =
            0.0;
            REAL(thetafinal,i) = atan2(REAL(zfinal,i),REAL(rfinal,i));
            IMAG(thetafinal,i) = 0.0;
        }

        //free the storage
        gsl_spline_free (spliner);
        gsl_interp_accel_free (accr);
        gsl_spline_free (splinez);
        gsl_interp_accel_free (accz);


        //Fourier analysis
        //test1: fourier transform the fist half, then fourier transform the late
        half.
        //Fourier analysis
        gsl_fft_complex_radix2_forward (rfinal, delta, M1);
        gsl_fft_complex_radix2_forward (zfinal, delta, M1);
        gsl_fft_complex_radix2_forward (thetafinal, delta, M1);

        // store fr, fz, magnitudes
        double frequency[M1];
        double sqrtmagr[M1];
        double sqrtmagz[M1];
        double sqrtmagtheta[M1];

        //find maximum magnitude
        double magr[M1];
        double magz[M1];
        double magtheta[M1];
        double magrmax=0.0;
        double magzmax=0.0;
        double magthetamax=0.0;
        double nr=0.0;//the location when magrmax occurs
        double nz=0.0;//the location when magzmax occurs
        double ntheta=0.0;//the location when magthetamax occurs

        //Find the maximum magnitudes and corresponding frequencies to understand the
        MOR in the first half
        for (i = 1; i < M1/2; i++){
            magr[i]=sqrt(REAL(rfinal,i)*REAL(rfinal,i)
            +IMAG(rfinal,i)*IMAG(rfinal,i));
            magz[i]=sqrt(REAL(zfinal,i)*REAL(zfinal,i)
            +IMAG(zfinal,i)*IMAG(zfinal,i));
```

```
            magtheta[i]=sqrt(REAL(thetafinal,i)*REAL(thetafinal,i)
            +IMAG(thetafinal,i)*IMAG(thetafinal,i));

            if (magr[i] > magrmax) {
                magrmax = magr[i];
                nr = i;
            }

            if (fabs(magz[i]) > magzmax) {
                magzmax = fabs(magz[i]);
                nz = i;
            }
            if (fabs(magtheta[i]) > magthetamax) {
                magthetamax = fabs(magtheta[i]);
                ntheta = i;
            }
            if (i==(M1/2-1)){
                double fr = nr*1.0/(M1);
                double fz = nz*1.0/(M1);
                ostr2 << fr << " "<< fz<< " "<< (nr/nz)<< " ";
                ostr2 << "\n";
            }
        }

    }
}

void particle::simplelevitation(GalaxyPotential &pH1, GalaxyPotential &pH2,
GalaxyPotential &pH3, ostringstream &ostr){
    //Instructions for the gnuplot: (printorbitNa) 4:5=heated; 6:7=unperturbed;
    4:8=levitation
    double steps = 16384.0;
    int j;
    // For the adiabaticly heated particle
    for (int j = 0; j < 200; j++) {
        settime();
        x[0] = 8.0; x[1] = 0.0; x[2] = 0.0;
        p[0] = 0.1; p[1] =0.2;p[2] = 0.0005*j;

        ostr<< p[0]<< " "<< p[1]<< " "<< p[2]<< " ";
        double periods[2];
        double maxtime = 10000.0;

        integrate(maxtime, pH1, pH2);
        double RmaxA=0;
        double zmaxA=0;
        double RA = getR();
        double zA = getz();
        double nrA=0;//number of nr for Rmax to occur
        double nzA=0;//number of nz for zmax to occur
        for (int i = 0; i < steps; i++) {
            for (int ii = 0; ii < 1; ii++) {
                leapfrog(pH1, pH2);
            }
            //print(pH1, pH2, ostr);
            double RA,phiA,vRA,vphiA;
```

```
            phitrans(RA,phiA,vRA,vphiA);
            //extract z step by step
            double zA;
            zA = fabs(x[2]);

            if (RA > RmaxA) {
                    RmaxA = RA;
                    nrA = i;
            }

            if (zA > zmaxA) {
                    zmaxA = zA;
                    nzA = i;
            }
            if (i==(steps-1)){
                ostr << RmaxA << " " << zmaxA << " "; //<< nrA+1 << " "<< nzA+1 << "
                "
            }
        }


        settime();
        // For the contracted particle
        x[0] = 8.0; x[1] = 0.0; x[2] = 0.0;
        p[0] = 0.1; p[1] =0.2;p[2] = 0.0005*j;

        integrate2(maxtime, pH1, pH3);
        double RmaxB=0;
        double zmaxB=0;
        double RB = getR();
        double zB = getz();
        for (int i = 0; i < steps; i++) {
            for (int ii = 0; ii < 1; ii++) {
                leapfrog2(pH1, pH3);
            }
            double RB,phiB,vRB,vphiB;
            phitrans(RB,phiB,vRB,vphiB);
            //extract z step by step
            double zB;
            zB = fabs(x[2]);

            if (RB > RmaxB) {
                    RmaxB = RB;
            }
            if (zB > zmaxB) {
                    zmaxB = zB;
            }
            if (i==(steps-1)){
                ostr << RmaxB << " " << zmaxB << " "<< zmaxA-zmaxB << " " << "\n";
            }
        }
    }

}

void particle::print(GalaxyPotential &pH) {
```

```
    double Ekin = 0.0;
    for (int i = 0; i < 3; i++) {
        cout << x[i] << " " << p[i] << " " << f[i] << " ";
        Ekin += 0.5*p[i]*p[i];
    }
    cout << timestep << " " << Ekin << " " << pH(sqrt(x[0]*x[0] + x[1]*x[1]), x[2])
    << " " << time+timefine;
    cout << "\n";
}


void particle::print(GalaxyPotential &pH, GalaxyPotential &pD) {
    double Ekin = 0.0;
    for (int i = 0; i < 3; i++) {
        cout << x[i] << " " << p[i] << " " << f[i] << " ";
        Ekin += 0.5*p[i]*p[i];
    }
    cout << timestep << " " << Ekin << " " << pH(sqrt(x[0]*x[0] + x[1]*x[1]), x[2]) +
    pD(sqrt(x[0]*x[0] + x[1]*x[1]), x[2]) << " " << time+timefine;
    cout << "\n";
}


void particle::phitrans(double &R, double &phi, double &vR, double &vphi) {
    phi = 0.0;
    if (x[1] != 0.0 || x[0] != 0.0) {
        phi = atan2(x[1],x[0]);
        R = sqrt(x[0]*x[0] + x[1]*x[1]);
//    double vs = p[0]*p[0] + p[1]*p[1];
        vR = (x[0]*p[0] + x[1]*p[1])/R;
        vphi = (-x[1]*p[0] + x[0]*p[1])/R;
    }
    else {
       R = 0.0; phi = 0.0; vR = 0.0; vphi = 0.0;
       cout << "WARNING do not use origin in phitrans() \n";
    }
}


void particle::phiset(double R,double phi,double vR, double vphi) {
    x[0] = R*cos(phi);
    x[1] = R*sin(phi);
    p[0] = vR*cos(phi) - vphi*sin(phi);
    p[1] = vR*sin(phi) + vphi*cos(phi);
}

void particle::phishift(std::uniform_real_distribution<double> phasedist) {
    double delphi = phasedist(randomGen);
    double R; double phi; double vR; double vphi;
    phitrans(R,phi,vR,vphi);
    phi += delphi;
    phiset(R,phi,vR,vphi);
}

void particle::phishift(double delphi) {
    double R; double phi; double vR; double vphi;
```

```
    phitrans(R,phi,vR,vphi);
    phi += delphi;
    phiset(R,phi,vR,vphi);
}




void particle::clone(particle &pp) {
    for (int i = 0; i < DIMENSION; i++) {
        x[i] = pp.x[i];
        p[i] = pp.p[i];
        f[i] = pp.f[i];
    }
    weight = pp.weight;
    time = pp.time;
    timestep = pp.timestep; htimestep = pp.htimestep;
}

void particle::integrate(double integtime, GalaxyPotential &pH) {
    stepnumber = 0;
    double ttarget = gettime() + integtime;
    while (gettime() < ttarget) {
        leapfrog(pH);
    }
}

void particle::integrate(double integtime, GalaxyPotential &pH, GalaxyPotential &pD)
{
    stepnumber = 0;
    double ttarget = gettime() + integtime;
    while (gettime() < ttarget) {
        leapfrog(pH, pD);
    }
}

void particle::integrate2(double integtime, GalaxyPotential &pH, GalaxyPotential
&pD2) {
    stepnumber = 0;
    double ttarget = gettime() + integtime;
    while (gettime() < ttarget) {
        leapfrog2(pH, pD2);
    }
}

void particle::integrateL(double integtime, GalaxyPotential &pH, GalaxyPotential &pD,
double time) {
    stepnumber = 0;
    double ttarget = gettime() + integtime;
    while (gettime() < ttarget) {
        leapfrogL(pH, pD, time);
    }
}

void compint(int i, std::uniform_int_distribution<int> intdist, int *savearray) {
```

```
    for (int j = 0; j < i; ++j) {
        if (savearray[j] == savearray[i]) {
            savearray[i] = intdist(randomGen);
            compint(i, intdist, savearray);
        }
    }
}

void diffinteg(int no, std::uniform_int_distribution<int> intdist, int *savearray) {
    for (int i = 0; i < no; ++i) {
        savearray[i] = intdist(randomGen);
        compint(i, intdist, savearray);
    }
}


void DoDisk() {
    std::ifstream fromfile1(FROMFILE1); std::ifstream fromfile2(FROMFILE2);
    GalaxyPotential pH1(fromfile1);
    GalaxyPotential pH2(fromfile2);
        double sigmaVrIn = 0.04;
    double sigmaVzIn = 0.03;
//    double frequarray[500];
//  frequarray[0] = 0.0;

    double alpha = 1.0;
    double rd = 2.9;

    //normalization array for weights

    double ***intval = new double**[CVALN];
    for (int n = 0; n < CVALN; n++) {
        intval[n] = new double*[GVALN];
        for (int m = 0; m < GVALN; m++) {
            intval[n][m] = new double[ZVALN];
            for (int p = 0; p < ZVALN; p++) {
                intval[n][m][p] = 0.0;
            }
        }
    }

    thread *persei = new thread[THREADNO];
    int perbes[THREADNO];
    for (int i = 0; i < THREADNO; i++) {
//        persei[i];
        perbes[i] = 0;
    }
    int countper = 0;
    for (int i = 1; i < CVALN; i++) {
        if (perbes[countper] == 1) {
            if (persei[countper].joinable()) {
                persei[countper].join();
                perbes[countper] = 0;
            }
        }
        persei[countper] = thread(getintpar, intval, alpha, rd, i);
```

```
        perbes[countper] = 1;
        countper++;
        if (countper >= THREADNO) {
            countper = 0;
        }
    }
    for (int i = 0; i < THREADNO; i++) {
        if (perbes[i] > 0) {
            if (persei[i].joinable()) {
                persei[i].join();
                perbes[i] = 0;
            }
        }
    }
    delete[] persei;


    cout << "done caluclating normarray\n";
    double sigmaVr = sigmaVrIn;
    double sigmaVz = sigmaVzIn;

    double param[11];
    for (int n = 0; n < 11; n++) {
        param[n] = 0.0;
    }

    /*
    double a = param[0];  //normalization
    double s0 = param[1];  //horizontal vel. dispersion at local galactocentric
    radius
    double h0 = param[2];  //scale height
    double vc = param[3];  //circular velocity
    double l = param[4];   //horizontal vel. dispersion scale length in km/s
    double m = param[5];   //vertical vel. dispersion scale length
    double nx = param[6];   //disc scale length
    double hh = param[7];   //altitude of measurement
    double alpha = param[8];  //potential adiabatic index alpha
    */

    double vcs = VC;
    double Rsun = RSUN;
    param[0] = 0.1;
    param[1] = sigmaVr * 1000.0;        //35.0;
    param[2] = 360.0;                       /// changed to scale height for thin disc
    param[3] = vcs;
    param[4] = 7.5 * vcs / Rsun;            ///7.5*lz ~ 200
    param[5] = 6.0 * vcs / Rsun;            ///~150
    double scalelength = 2.9;   //disc scalelength
    param[6] = scalelength * vcs / Rsun;  /// get rd in velocity units
    param[7] = 0.0; //defined in loop
    param[8] = 1.0;                         ///alpha

    param[10] = sigmaVz * 1000.0;
    double sigphi = VPHIPLACESIG;

    double rInird = RPLACEL;
```

```
    double zInizd = ZPLACEL;

    std::uniform_real_distribution<double> rInidist(0.0, rInird* (exp(-RMIN / rInird)
    - exp(-RMAX / rInird)));
    std::uniform_real_distribution<double> zInidist(-zInizd * (1.0 - exp(-2.0 /
    zInizd)), zInizd * (1.0 - exp(-2.0 / zInizd)));

    std::uniform_real_distribution<double>  phasedist(0.0, 2.0 * PI);
    std::normal_distribution<double> pphidistribution(VPHIPLACE, sigphi);
    std::normal_distribution<double> pzdistribution(0.0, /*2.0* */ param[10] *
    VZPLACEFAC / 1000.0);
    std::normal_distribution<double> prdistribution(0.0, /*2.0* */ param[1] *
    VRPLACEFAC / 1000.0);

    particle **ppa = new particle*[THREADNO];
    particle **ppa0 = new particle*[THREADNO];
    int nbb = NBASE;
    int nperb = NPERBASE;
    std::thread *weightthreads = new std::thread[THREADNO];
    for (int i = 0; i < THREADNO; i++) {
        ppa0[i] = new particle[NBASE*NPERBASE];
        ppa[i] = new particle[NBASE*NPERBASE];
    }

    cout << "launch the threads \n";
        for (int u = 0; u < THREADNO; u++)
    {
        weightthreads[u] = std::thread(weighting, nbb, nperb, ppa[u], phasedist,
        pphidistribution, sigphi, prdistribution, pzdistribution, rInidist, zInidist,
        intval, param, scalelength, rInird, zInizd, u);
    }
//  weighting(THREADNO*particlePerThread, particleNumber, PSvector, phasedist,
pphidistribution, sigphi, prdistribution, pzdistribution, rInidist, zInidist, intval,
param, scalelength, rInird, zInizd, THREADNO);

    cout << "join weighting \n";
    for (int u = 0; u < THREADNO; u++)
    {
        weightthreads[u].join();
        cout << "thread " << u << " joined \n";
    }

    cout << "delete weighting threads \n";
    delete[] weightthreads;

    for (int n = 0; n < CVALN; n++) {
        for (int m = 0; m < GVALN; m++) {
            delete[] intval[n][m];
            }
        delete[] intval[n];
        }
    delete[] intval;

    //------------------------------end intital
    conditions------------------------------
    cout << "done weighing basisorbits\n";
```

```
   int nsub = NBASE*NPERBASE;
   for (int i = 0; i < THREADNO; i++) {
       for (int jj = 0; jj < nsub; jj++) {
           ppa0[i][jj].clone(ppa[i][jj]);
       }
   }
   cout << "cloned particles \n";
   std::thread *perseint = new std::thread[THREADNO];
   double integtime = 10000.0;

   for (int i = 0; i < THREADNO; i++) {
        perseint[i] = std::thread(integset, ppa[i], nsub, integtime, i);
   }
   for (int i = 0; i < THREADNO; i++) {
       perseint[i].join();
       cout << "thread " <<  i << " joined \n";
   }
   delete [] perseint;

   ostringstream ostr;
   for (int i = 0; i < THREADNO; i++) {
       for (int jj = 0; jj < nsub; jj++) {
        ppa[i][jj].print(pH1, pH2, ostr);
        ppa0[i][jj].print(pH1, ostr);
       }
   }

   FILE *allout = fopen("allparticles4", "w");
   string stri = ostr.str();
   char *ch = &stri[0];
   fputs(ch, allout);
   fflush(allout);
   fclose(allout);

   for (int i = 0; i < THREADNO; i++) {
       delete [] ppa[i];
       delete [] ppa0[i];
   }
   delete [] ppa;
   delete [] ppa0;

}

void Dotest1() {

    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");//MW2017
    GalaxyPotential pD2(fromfile2);

    particle pp;
    ostringstream ostr1;
    ostringstream ostr11;
```

```
    ostringstream ostr12;
    ostringstream ostr2;
    ostringstream ostr21;
    ostringstream ostr3;
    //fix the potential
    //fourier analysis
    pp.fourier(pH,pD,ostr1,ostr11,ostr12,ostr2,ostr21,ostr3);
    //pD is 10% of pD2
    FILE *fout1 = fopen("printorbit1", "w");
    string stri1 = ostr1.str();
    char *ch1 = &stri1[0];
    fputs(ch1, fout1);
    fclose(fout1);
    FILE *fout11 = fopen("SOS", "w");
    string stri11 = ostr11.str();
    char *ch11 = &stri11[0];
    fputs(ch11, fout11);
    fclose(fout11);
    FILE *fout12 = fopen("SOS+vz", "w");
    string stri12 = ostr12.str();
    char *ch12 = &stri12[0];
    fputs(ch12, fout12);
    fclose(fout12);
    FILE *fout2 = fopen("interpolation", "w");
    string stri2 = ostr2.str();
    char *ch2 = &stri2[0];
    fputs(ch2, fout2);
    fclose(fout2);
    FILE *fout21 = fopen("max", "w");
    string stri21 = ostr21.str();
    char *ch21 = &stri21[0];
    fputs(ch21, fout21);
    fclose(fout21);
    FILE *fout3 = fopen("printfourier", "w");
    string stri3 = ostr3.str();
    char *ch3 = &stri3[0];
    fputs(ch3, fout3);
    fclose(fout3);

}

void Dotest2() {
    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");//MW2017
    GalaxyPotential pD2(fromfile2);
    particle pp;
    ostringstream ostr1;
    ostringstream ostr11;
    ostringstream ostr12;
    ostringstream ostr2;
    //sos of 10 orbits
    pp.sos(pH,pD,ostr1,ostr11, ostr12,ostr2);
    FILE *fout1 = fopen("SOSN", "w");
```

```
    string stri1 = ostr1.str();
    char *ch1 = &stri1[0];
    fputs(ch1, fout1);
    fclose(fout1);
    FILE *fout11 = fopen("test", "w");
    string stri11 = ostr11.str();
    char *ch11 = &stri11[0];
    fputs(ch11, fout11);
    fclose(fout11);
    FILE *fout12 = fopen("SOSN+vz", "w");
    string stri12 = ostr12.str();
    char *ch12 = &stri12[0];
    fputs(ch12, fout12);
    fclose(fout12);
    FILE *fout2 = fopen("FFTN", "w");
    string stri2 = ostr2.str();
    char *ch2 = &stri2[0];
    fputs(ch2, fout2);
    fclose(fout2);
}
void Dotest3() {
    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");
    GalaxyPotential pD2(fromfile2);
    ostringstream ostrNa;
    particle pp;
    // making particles
    pp.simplelevitation(pH,pD,pD2,ostrNa);
    FILE *foutNa = fopen("printorbitNa", "w");
    string striNa = ostrNa.str();
    char *chNa = &striNa[0];
    fputs(chNa, foutNa);
    fclose(foutNa);
}
void Dotest4() {
    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");//MCM2017
    GalaxyPotential pD2(fromfile2);
    particle pp;
    ostringstream ostr1;
    ostringstream ostr2;
    //sos of 10 orbits
    pp.dragging(pH,pD,pD,ostr1,ostr2);
    FILE *fout1 = fopen("L1", "w");
    string stri1 = ostr1.str();
    char *ch1 = &stri1[0];
    fputs(ch1, fout1);
    fclose(fout1);
    FILE *fout2 = fopen("L2", "w");
    string stri2 = ostr2.str();
```

```
    char *ch2 = &stri2[0];
    fputs(ch2, fout2);
    fclose(fout2);
}

void Dotest5() {
    //epicycle frequency and effective potential
    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");//MCM2017
    GalaxyPotential pD2(fromfile2);
    particle pp;
    ostringstream ostr1;
    ostringstream ostr21;
    ostringstream ostr22;
    ostringstream ostr31;
    ostringstream ostr32;
    pp.epicycle(pH,pD,ostr1,ostr21,ostr22,ostr31,ostr32);
    FILE *fout1 = fopen("vc", "w");
    string stri1 = ostr1.str();
    char *ch1 = &stri1[0];
    fputs(ch1, fout1);
    fclose(fout1);
    FILE *fout21 = fopen("effpotR", "w");
    string stri21 = ostr21.str();
    char *ch21 = &stri21[0];
    fputs(ch21, fout21);
    fclose(fout21);
    FILE *fout22 = fopen("epicycleR", "w");
    string stri22 = ostr22.str();
    char *ch22 = &stri22[0];
    fputs(ch22, fout22);
    fclose(fout22);
    FILE *fout31 = fopen("effpotz", "w");
    string stri31 = ostr31.str();
    char *ch31 = &stri31[0];
    fputs(ch31, fout31);
    fclose(fout31);
    FILE *fout32 = fopen("epicyclez", "w");
    string stri32 = ostr32.str();
    char *ch32 = &stri32[0];
    fputs(ch32, fout32);
    fclose(fout32);
}

void Dotest6() {
    //epicycle frequency and effective potential with respect to time
    std::ifstream fromfile("Halo1.Tpot");
    GalaxyPotential pH(fromfile);
    std::ifstream fromfile1("Disc1.Tpot");
    GalaxyPotential pD(fromfile1);
    std::ifstream fromfile2("Disc2.Tpot");//MCM2017
    GalaxyPotential pD2(fromfile2);
    particle pp;
```

```
    ostringstream ostr1;
    ostringstream ostr21;
    ostringstream ostr22;
    ostringstream ostr31;
    ostringstream ostr32;
    pp.epicycletime(pH,pD,ostr1,ostr21,ostr22,ostr31,ostr32);
    FILE *fout1 = fopen("Tvc", "w");
    string stri1 = ostr1.str();
    char *ch1 = &stri1[0];
    fputs(ch1, fout1);
    fclose(fout1);
    FILE *fout21 = fopen("TeffpotR", "w");
    string stri21 = ostr21.str();
    char *ch21 = &stri21[0];
    fputs(ch21, fout21);
    fclose(fout21);
    FILE *fout22 = fopen("TepicycleR", "w");
    string stri22 = ostr22.str();
    char *ch22 = &stri22[0];
    fputs(ch22, fout22);
    fclose(fout22);
    FILE *fout31 = fopen("Tratio", "w");
    string stri31 = ostr31.str();
    char *ch31 = &stri31[0];
    fputs(ch31, fout31);
    fclose(fout31);
    FILE *fout32 = fopen("Tratio8", "w");
    string stri32 = ostr32.str();
    char *ch32 = &stri32[0];
    fputs(ch32, fout32);
    fclose(fout32);
}

void Dotest10() {
    //fourier transform of a SHO
    #define T 10 //periods
    #define TIME 64
    ostringstream ostrf1;
    double positionx[TIME];
    double omg=2.0*PI/T;
    double sqrtmag[TIME];
    double freq[TIME];

    int i; double data[2*TIME];
    for (i = 0; i < TIME; i++)
    {
       REAL(data,i) = cos(1.0*i*omg); IMAG(data,i) = 0.0;
    }

    gsl_fft_complex_radix2_forward (data, 1, TIME);
    for (i = 0; i < TIME; i++)
    {
      printf ("%d %e %e\n", i,
              REAL(data,i)/sqrt(TIME),
              IMAG(data,i)/sqrt(TIME));
    }
```

```
/*
    for (i = TIME/2; i < TIME; i++){
        positionx[i]=sin(1.0*i*omg);
        sqrtmag[i]=sqrt(REAL(data,i)*REAL(data,i)+IMAG(data,i)*IMAG(data,i));
        freq[i]=-1.0/(T)+i*1.0/(T*TIME);
        //ostrf1<<freq[i]<<" "<<sqrtmag[i]<<" ";
        //ostrf1 << "\n";
    }
*/
    int j;
    for (i = 0; i < TIME; i++)
    {
        if (i<TIME/2){
            j = i+TIME/2;
            positionx[j]=cos(1.0*i*omg);
            sqrtmag[j]=sqrt(REAL(data,j)*REAL(data,j)+IMAG(data,j)*IMAG(data,j));
            freq[j]=-1.0+j*1.0/(TIME);
        }
        else{
            j= i-TIME/2;
            positionx[j]=cos(1.0*i*omg);
            sqrtmag[j]=sqrt(REAL(data,j)*REAL(data,j)+IMAG(data,j)*IMAG(data,j));
            freq[j]=j*1.0/(TIME);
        }

        //ostrf1<<i*1.0/(T*TIME)<<" "<<positionx[i]<<" "<<REAL(data,i)/sqrt(TIME)<<"
"<<IMAG(data,i)/sqrt(TIME)<<" ";
        //ostrf1<<positionx[j]<<" ";
        ostrf1<<freq[j]<<" "<<sqrtmag[j]/TIME<<" ";
        ostrf1 << "\n";
    }

    FILE *foutf1 = fopen("printf1", "w");
    string strif1 = ostrf1.str();
    char *chf1 = &strif1[0];
    fputs(chf1, foutf1);
    fclose(foutf1);
}


int main() {
    Dotest4();

    return 0;
}

/*
#define S 100
int
main (void)
{
  int i;
  double xi, yi, x[S], y[S];
  ostringstream ostrint1;
  ostringstream ostrint2;
  printf ("#m=0,S=17\n");
```

```
  for (i = 0; i < S; i++)
    {
      x[i] = i + 0.5 * sin (i);
      y[i] = sin(1.0*i)+cos(1.0*i);
      printf ("%g %g\n", x[i], y[i]);
      ostrint1<<x[i]<<" "<<y[i]<<" ";
      ostrint1<< "\n";
    }

  printf ("#m=1,S=0\n");


    gsl_interp_accel *acc
      = gsl_interp_accel_alloc ();
    gsl_spline *spline
      = gsl_spline_alloc (gsl_interp_cspline, S);

    gsl_spline_init (spline, x, y, S);
    //double delta=gettime()/M;
    for (xi = x[0]; xi < x[99]; xi += 0.1345632)
      {
        yi = gsl_spline_eval (spline, xi, acc);
        ostrint2<<xi<<" "<<yi<<" ";
        ostrint2<< "\n";
      }
    gsl_spline_free (spline);
    gsl_interp_accel_free (acc);


    FILE *foutint1 = fopen("printint1", "w");
    string striint1 = ostrint1.str();
    char *chint1 = &striint1[0];
    fputs(chint1, foutint1);
    fclose(foutint1);
    FILE *foutint2 = fopen("printint2", "w");
    string striint2 = ostrint2.str();
    char *chint2 = &striint2[0];
    fputs(chint2,foutint2);
  return 0;
}
*/
```