Faculté de physique et ingénierie
Université de Strasbourg

Observatoire astronomique de Strasbourg | ObAS

Order and Chaos in a 2D potential

Numerical Methods and Simulation Project

Junaid Ramzan Bhat and Yaël Moussouni
Observatoire astronomique de Strasbourg (CNRS, Unistra)
Master of Science 2 – Astrophysics and Data Sciences

January 23rd, 2024

Contents

1 Introduction

The goal of this numerical project is to develop computational techniques for simulation and analysis of orbits under the influence of Hénon–Heiles potential. This potential represents a simplified dynamical system where particles can exhibit both regular and chaotic behavior depending on the total energy of the system. It was inspired by whether there was an analytical solution of the motion of stars moving in the potential of the galaxy. It was assumed such a potential had a symmetry axis, and so Hénon &Heiles (1964) investigated whether an axis symmetric potential admits a third isolating integral of motion.

Integrals of motion are the quantities that are conserved along the orbit, e.g. energy, angular momentum, etc. These integrals of motion act as constraints, limiting the regions of the phase space that the object can explore. For the galactic dynamics, as conveyed above it was assumed that the potential had a symmetry axis and was time-independent. The system is a 6-dimensional phase space $(R, \theta, z, \dot{R}, \dot{\theta}, \dot{z})$, therefore it has five independent conservative integrals of the motion (Hénon &Heiles 1964). Each of these integrals represent a hypersurface in the 6D phase space and the trajectory is the intersection of these. These integrals can be categorized into isolating or non-isolating.

The non-isolating integrals fill the whole phase space and give no restriction on the trajectory. Thus, one can say that non isolating integrals have no significance in determining the orbit. It can shown that at least 2 integrals in general are non isolating and at the time of writing of the paper two isolating integrals for the total energy and the angular momentum per unit mass of the star were known. Since, no analytical solution for the motion of the stars under the galactic potential was found it was assumed that the third integral is also non-isolating but the observations of orbits of stars near the sun as well as a number of numerical computations of orbits behaved as if there were three isolating integrals. Hénon–Heiles set to find any proof for the existence of this integral but not limiting themselves to the astronomical nature of the problem, they chose a potential that became to be known as Hénon–Heiles potential, to explore this question in a broader context.[1]

In this numerical project, we tested various numerical integrators like Euler method, 2^{nd} order Runge–Kutta method (RK2) and 4^{th} order Runge kutta method (RK4) to compute the orbits under a potential. We tested our integrators for a potential whose analytical solutions are known like the Kepler potential and make a comparison between the efficiency and accuracy of each of them. After validating our integrators, we used the RK4 method and apply it to the Hénon–Heiles potential to compute the Poincaré sections to visualize both chaotic and regular orbits. Finally, we applied the technique from Gottwald &Melbourne (2004), which is based on the slope of the distance in the phase space along an orbit to find whether an orbit is chaotic or regular and make comparison between the results obtained through two different techniques.

2 Numerical Integrators

In order to simulate the a system from a set of equations governing it and recover its trajectory, we use numerical integrators. As part of the project, we developed different numerical integrators as given below.

2.1 Euler Method

The Euler method is one of the simplest numerical techniques for solving ordinary differential equations (ODEs). It approximates the solution by stepping forward in small increments using the slope of the function at each point. It is a first order method.

For a first-order differential equation of the form:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y) \tag{2.1}$$

[1] see https://jfuchs.hotell.kau.se/kurs/amek/prst/11_hehe.pdf

with $y(x_0) = y_0$. To estimate the values taken by y, we discretize the problem: let us consider a step size h, then for all n:

$$y_{n+1} = y_n + h \cdot f(x_n, y_n) \tag{2.2}$$

where $x_n = x_0 + nh$ and $y_n = y(x_0 + nh) = y(x_n)$

2.2 Second Order Runge–Kutta Method

The second-order Runge–Kutta method (RK2), also known as the midpoint method, is a numerical technique for solving ordinary differential equations (ODEs). It improves upon the Euler method by using an additional intermediate step to estimate the slope more accurately. It is a second order predictor-corrector method,the central idea behind these methods is to first make a guess (predictor step) about the solution at a future point and then refine this guess (corrector step) to achieve higher accuracy.

For the same differential equation (2.1), the RK2 method estimates the next value of y as follows:

Initial slope. Calculate the slope at the beginning of the interval using:

$$k_1 = f(x_n, y_n), \tag{2.3}$$

where k_1 is the slope at the current point (x_n, y_n).

Midpoint slope. Estimate the slope at the midpoint of the interval by predicting y at the midpoint:

$$k_2 = f\left(x_n + \frac{h}{2},\ y_n + \frac{h}{2} \cdot k_1\right), \tag{2.4}$$

where k_2 is the slope at the midpoint.

Update the solution. Using k_2 to compute the next value of y:

$$y_{n+1} = y_n + h \cdot k_2. \tag{2.5}$$

Here:

- h is the step size,
- k_1 represents the slope at the start of the interval,
- k_2 refines the slope estimate using the midpoint value.

2.3 Fourth Order Runge-Kutta Method

Another numerical integration method belonging to the Runge–kutta family is RK4. As evident from the name, it is a fourth order method. In this method, the value of the function i.e. y at the next step is determined by using the present value plus the weighted average of the slopes at different points within the next full step, multiplied by the step size.

We take the same differential equation (2.1) apply the RK4 method as follows to get the next values of y:

$$\begin{aligned} k_1 &= f(x_n,\ y_n) & k_3 &= f\left(x_n + \frac{h}{2},\ y_n + \frac{h}{2} \cdot k_2\right) \\ k_2 &= f\left(x_n + \frac{h}{2},\ y_n + \frac{h}{2} \cdot k_1\right) & k_4 &= f(x_n + h,\ y_n + h \cdot k_3) \end{aligned} \tag{2.6}$$

These increments are combined to compute the next value of y as:

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right),$$

Here:

- h is the step size,
- k_1, k_2, k_3, and k_4 are the slopes calculated at different points within the interval.

3 Kepler Potential

3.1 Main Equations

The Kepler problem refers to a special case of the two body problem, in which the two bodies are interacting through a central force which follows the inverse square law e.g. gravitational force (Strauch 2009). The potential associated with such a force is called the Kepler potential. We consider a gravitational two body problem with point masses to compare the accuracy of our integrators by comparing it with the analytical solutions of the problem as well. This potential is given by

$$V(R) = -\frac{Gm_1m_2}{R} \tag{3.1}$$

where G is the gravitational constant and m_1, m_2 are the masses of the central body and the orbiting particle respectively. In our project, for the sake of simplicity and without the loss of generality, we set $G = 1$ and $m_1 = m_2 = 1$. We use the Hamiltonian formulism to find the set of equations governing the motion and the Hamiltonian of the problem.

The Hamilton equations for a system, in general are:

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\partial \mathcal{H}}{\partial p} \qquad \frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial q}. \tag{3.2}$$

where $\mathcal{H}$ is the Hamiltonian of the system, q represents the generalized position vector and p is the conjugate momentum vector. The Hamiltonian, which is equal to the total energy for a conservative system like the Kepler problem is given as

$$\mathcal{H}(q,p) = T + V = \frac{1}{2}(u^2 + v^2) - \frac{1}{\sqrt{x^2+y^2}} \tag{3.3}$$

therefore:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \frac{\partial \mathcal{H}}{\partial u} = u & \qquad \frac{\mathrm{d}u}{\mathrm{d}t} &= -\frac{\partial \mathcal{H}}{\partial x} = -\frac{x}{r^3} \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \frac{\partial \mathcal{H}}{\partial v} = v & \qquad \frac{\mathrm{d}v}{\mathrm{d}t} &= -\frac{\partial \mathcal{H}}{\partial y} = -\frac{y}{r^3} \end{aligned} \tag{3.4}$$

where $r = \sqrt{x^2+y^2}$. We solve the equations (3.4) numerically using the methods described in section 2 to simulate the orbit for a given time span.

The main reason, we chose this potential is that its analytical solution is known, so that we can compare our numerical schemes with the analytical result. For the particle in circular orbit (our case), the radial distance r_0 as well as the angular velocity $\omega = \sqrt{u_0^2 + v_0^2}/r_0$ are both constant. The position and the velocities are functions of time t and are given parametrically by:

$$x(t) = r_0\cos(\omega t) \qquad y(t) = r_0\sin(\omega t), \tag{3.5}$$

and

$$u(t) = -r_0\omega\sin(\omega t) \qquad v(t) = r_0\omega\cos(\omega t).$$

These parametric equations yield a trajectory that is a perfect circle centered on the origin. The total energy of the system,which is constant over time is given by=

$$E = \frac{[u(t)]^2 + [v(t)]^2}{2} - \frac{1}{r_0}. \tag{3.6}$$

With this we have all the required equations to test the numerical schemes which has been done in the subsequent section.

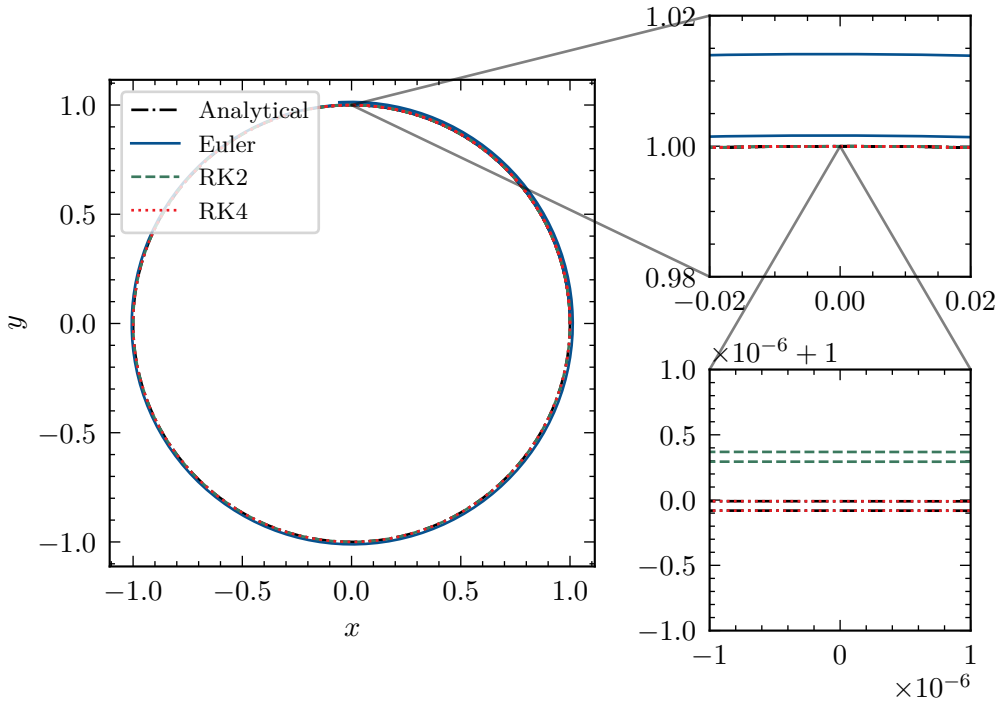

Figure 1: Global view of integrated orbits in a Kepler potential (left) and two zoomed panels (right). Three integrators are used: Euler (solid blue line), RK2 (dashed green line) and RK4 (dotted red line), and the analytical solution is also shown (black dashed–dotted line). The step size here is $h = 0.001$. The first zoom panel (top right) emphasis the rapid deviation of the Euler method, with a first orbit start to deviate from the analytical solution already during the first orbit, and goes away even more during its second orbit. The second zoom panel (bottom right) emphasis the deviation from the RK2 method. In this panel we also notice a deviation between the two analytical orbits, caused by the accumulation of machine error. The RK4 deviation cannot be noticed.

3.2 Accuracy of Numerical Integrators

In figure 1, we show the graphs obtained using the different integration as well as the evolution of the error with different step sizes for various integrators.

Even for such a small step size ($h = 0.001$), the Euler method is already deviating significantly from the analytical trajectory. This behaviour is expected as the Euler method is just a first order method. Even though it appears that RK2 is as good as RK4, we show a zoomed in segment of the orbit to show that is not the case, even with such a small step RK4 is better than RK2. This difference is of the order of 10^{-6} but as the integration time increases the error will increases, deviating significantly from the analytical path.

We show another plot for a higher step size in the figure 2. At such a step size, all the integrators are erroneous but it seems that RK4 is still able to reproduce the analytical orbit.

In the Keplerian problem the total energy of the body remains conserved, thus we present the error between the analytical energy value and the numerical value after the integration of the orbit with various integrators for different step sizes for same total integration time. As shown in the figure 3. One can now conclusively see that the RK4 integrator with the step size $h = 0.001$ is an apt choice for our further analysis. With this step size we get an error nearly equal to the machine epsilon ($\varepsilon \sim 10^{-16}$). In the graph, one can observe that unlike the other integrators the RK4 has comparatively more error at $h < 0.001$ but we think it is just the artifact of the machine, RK4

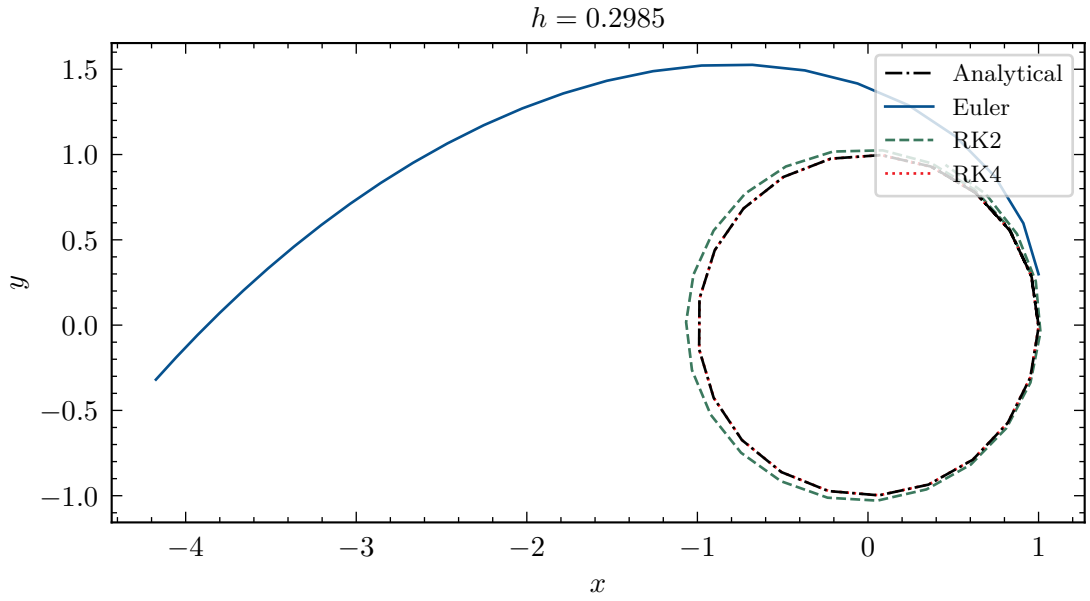

Figure 2: Integrated orbits in a Kepler potential. Three integrators are used: Euler (solid blue line), RK2 (dashed green line) and RK4 (dotted red line), and the analytical solution is also shown (black dashed–dotted line). The step size here is $h = 0.2985$. With this value very high for a step size, the Euler method can no longer produce relevant results, RK2 integrator starts to deviate from the analytical solution while the RK4 seems to produce results close enough from the analytical solution to be unnoticeable

starts to produce results that are more accurate than the machine epsilon,the numerical error is dominated by the limitations of floating-point precision in the system[2].

4 Hénon–Heiles Potential

The potential chosen by the authors to be studied became known as the Hénon-Heiles potential. It is a two-dimensional potential which appears to be a harmonic potential with perturbative terms. It was chosen because of its three properties:

1. It is simple to solve analytically, making the computations easier;
2. It can give chaotic trajectories as well;
3. It represents a general class of potentials which behave in same way even when more perturbative terms are added.

The potential (shown in figure 4) is defined as:

$$V(x, y) = \frac{1}{2}\left(x^2 + y^2 + 2x^2y - \frac{2}{3}y^3\right) \tag{4.1}$$

[2]see appendix A.

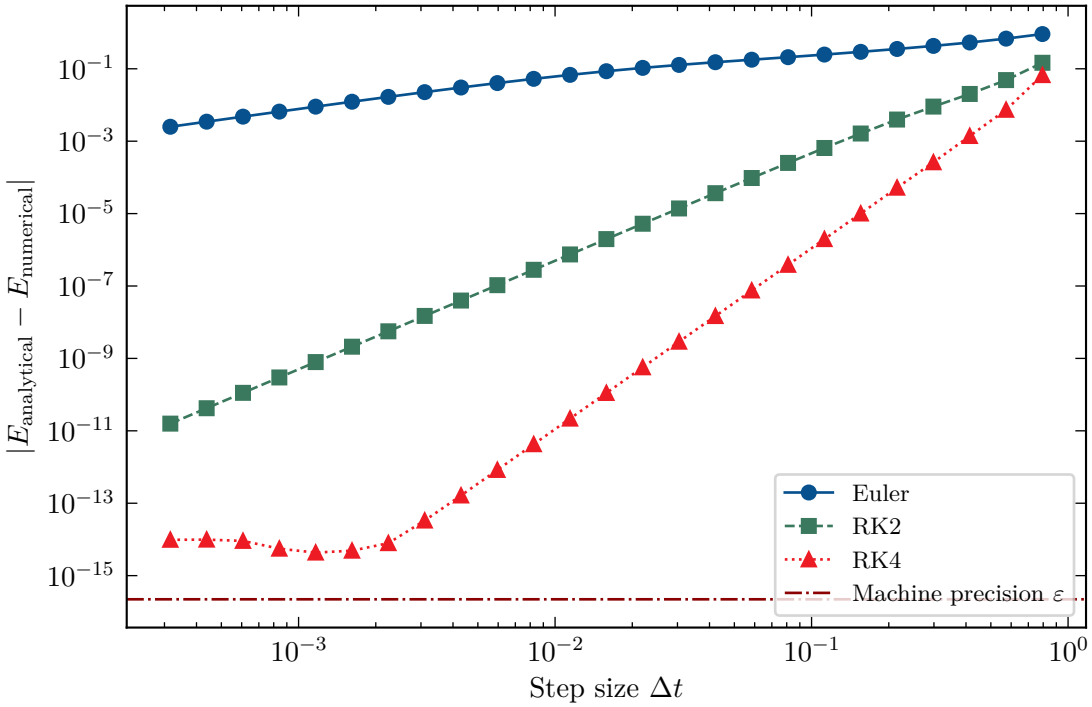

Figure 3: Difference between the analytical energy value and final numerical energy value for various step sizes. The machine precision correspond to the python `float` type precision.

4.1 Equations of Motion

The motion of a particle in the Hénon–Heiles potential is governed by the equations of motion derived from the potential energy $U(x, y)$. The time evolution of the system is described by the following first-order differential equations:

$$\dot{x} = -\frac{\partial U}{\partial x} = -(x + 2xy) \qquad \dot{y} = -\frac{\partial U}{\partial y} = -(x^2 - y^2 + y) \tag{4.2}$$

The equations of motion are integrated numerically using the RK4 method to compute the particle's trajectory, one such trajectory corresponding to $E = 1/12$ with a step size of 0.001 has been provided in the figure 5.

4.2 Initial Conditions

In their paper, Hénon &Heiles (1964) use x and y such that $-1 \leq x \leq 1$ and $-0.5 \leq y \leq 1$[3]. We randomly sampled initial positions (x_0, y_0) in this range, according to the constraint that the potential energy of the particle must be less than the total energy of the particle i.e. $U(x, y) < E$ to avoid leading to negative kinetic energies which is obviously unphysical.

The law of conservation of the energy for the system can be formulated as,

$$U(x, y) + \frac{1}{2}\left(u^2 + v^2\right) = E \tag{4.3}$$

This directly leads to $U(x, y) \leq E$. This constraint defines a bounded region in phase space where the particle's motion will be confined to a certain region otherwise the particle has enough energy

[3]These ranges were taken from the plot of the potential (Hénon &Heiles 1964, figure 2).

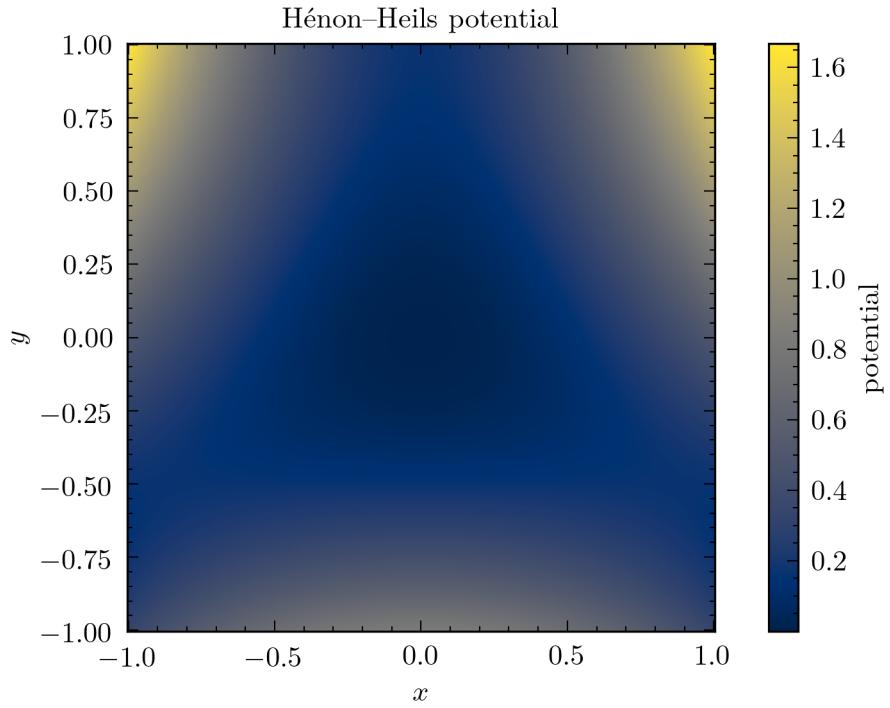

Figure 4: Hénon–Heiles potential in natural units, shown in real (x, y) space. The central blue part is attractive. There is also three "valleys" (top, bottom left and bottom right) that are also attractive, but that can drag particles away if their total energy is too high.

to escape the potential. Then, we generate $N_{\text{part}} = 100$ particles of coordinates (x_0, y_0) in this region, and compute the norm of their velocity with:

$$\|\mathbf{v}_0\| = \sqrt{2[E - U(x_0, y_0)]} \tag{4.4}$$

and compute a u_0 and v_0 components (along the x and y axis respectively) by generating a random direction $\theta_0 \in [0; 2\pi)$

$$u_0 = \sqrt{2[E - U(x_0, y_0)]}\cos(\theta_0) \qquad v_0 = \sqrt{2[E - U(x_0, y_0)]}\sin(\theta_0) \tag{4.5}$$

This gives a set of initial conditions whose time evolution we can track in phase space to find nature of these orbits.

4.3 Chaotic orbits

The behavior of the particle under the potential is chaotic as well for higher energies. Figure 6 shows the one such chaotic trajectory. By varying the initial conditions and total energy, a wide range of orbits have been studied, from regular, periodic trajectories to chaotic ones, whose results will be discussed in the subsequent sections

4.4 Poincare Sections

In dynamical systems like ours, A Poincaré section (Poincaré 1893) is a method for visualizing the behavior of this dynamical system by taking a lower-dimensional "slice" through its phase space. It is created by sampling the trajectories of the system at regular intervals or whenever they intersect a predefined surface, known as the Poincaré surface of section. It reduces a continuous flow to a discrete-time mapping[4]. Mathematically, a Poincaré section is the set of points where a trajectory

[4]see ocw.mit.edu/courses/12-006j-nonlinear-dynamics-chaos-fall-2022/mit12_006jf22_lec15-16.pdf

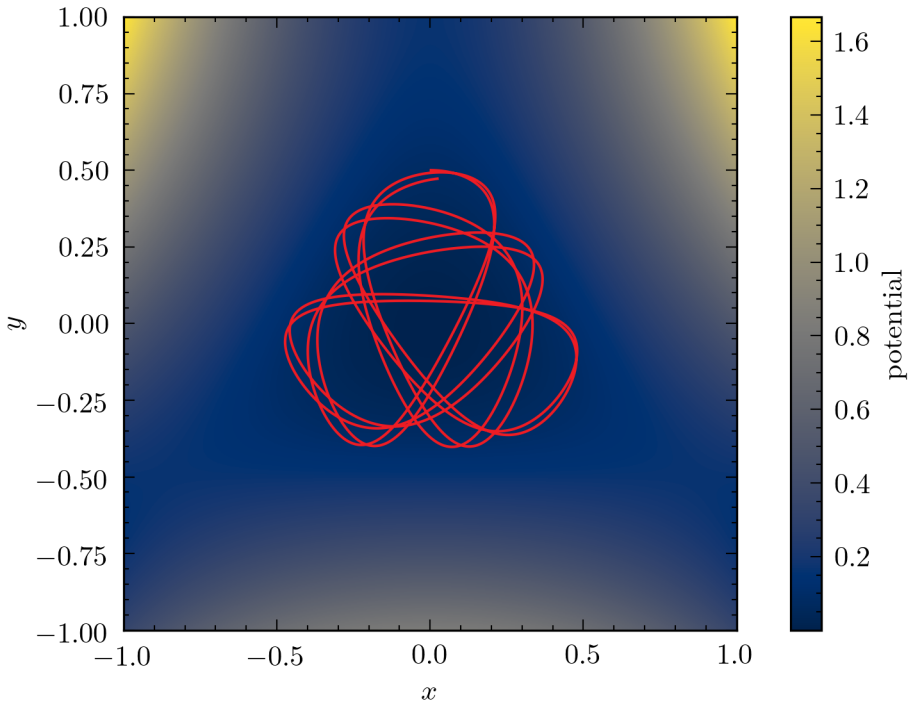

Figure 5: Trajectory of a particle under the influence of the Hénon-Heiles potential. The trefoil motion of the particle is ordered.

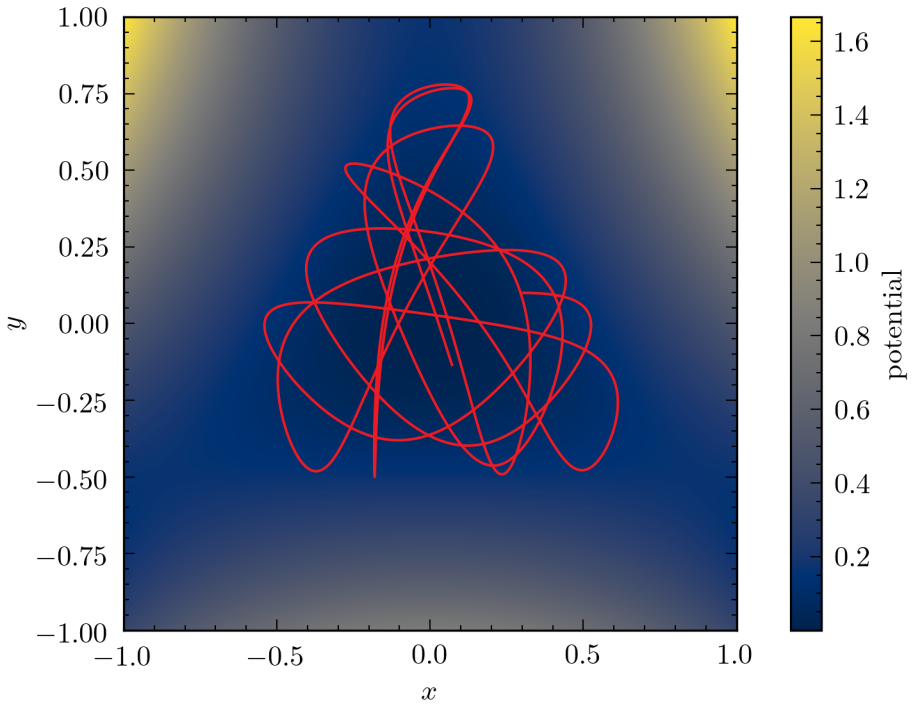

Figure 6: Chaotic trajectory of a particle under the influence of the Hénon-Heiles potential.

of a higher-dimensional system intersects a hypersurface in the phase space. By doing so, we reduce the dimensionality while still retaining the behavior of the system.

One can interpret the behavior of the dynamics by visualizing the Poincaré maps. A periodic orbit can produce a single fixed point (if the period coincides with the return time of the section) or a finite set of points,a quasi-periodic dynamics will form smooth continuous closed loops and for the chaotic dynamics, the Poincaré section will appear scattered over a region of the section, there will be no simple repeating structure but if the chaos is deterministic the section can display some order but not simple curves (Cheb-Terrab &De Oliveira 1996).

In our study, we compute the Poincaré section by identifying the points where the particle's trajectory crosses the plane $x = 0$ in phase space. At each crossing we record the y-coordinate and the velocity v_y, interpolating to ensure accuracy. Through this approach, we reduce the dimensionality, projecting the dynamics onto the y–v_y plane. The figure 7 represents the recorded Poincaré sections.

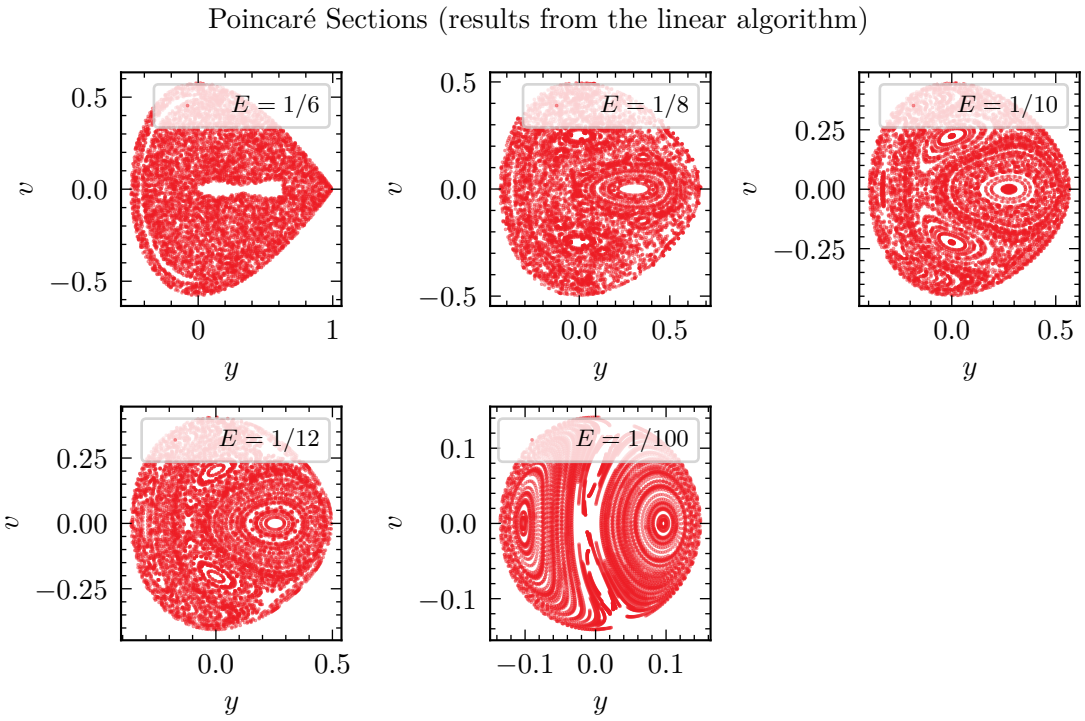

Figure 7: Poincaré sections of the Hénon-Heiles potential for energies $E = 1/6$, $1/8$, $1/10$, $1/12$ and $1/100$

.

We can see that for the lower energy values the section has closed curves with all the points lying on the curves filling completely the available areas which shows that the particle has stable behavior but the Poincaré section for $E = 1/8$ tells a different story, there are still closed curves in the various parts of the diagram (islands of stability) but it is evident these curves don't fill the whole area, there are a lot of isolated points through which one can not draw any curve, it represents a transition towards instability and chaotic dynamics. These apparently randomly distributed points fill the space between curves. As the energy is increased to $E = 1/6$, we see that the islands of stability are gone and there are just random points that appear to fill the whole phase space[5] i.e. ergodic behavior. The open circles (see figure 14) in the diagram for this energy correspond to a

[5]The outer line of all the diagrams is the limit given by $U(0, y) + \dot{y}/2 = E$ for our chosen surface of $x = 0$ in the phase space.

trajectory of new kind, intermediate between the closed curves and ergodic behavior. The particle is now totally in a chaotic regime, a drastic change from $E = 1/12$ to $1/6$.

5 Chaos Analysis

In order to characterize the chaotic behavior of the Hénon-Heiles potential, we will use two different based, one based on the computation of the orbits and another based on the slope of the distance in the phase space along a single orbit.

5.1 Area Analysis

To study the transition in greater detail, we computed the proportion of the total allowable area in the $(y, \dot{y})$ plane covered by curves for various values of E. The method used to determine whether a point P_i belongs to a curve or to an ergodic orbit involves a secondary initial point P_i', taken very close to P_i, at a distance of 10^{-7} in our case. For each point, a sequence of successive transforms, were calculated for both P_i and P_i'. If P_i and P_i' lie in a region occupied by curves, the distance P_iP_i' will increase slowly, approximately linearly, with the number of transforms i. Conversely, if P_i and P_i' are in the ergodic region, this distance will grow rapidly, roughly exponentially. To quantify this behavior, we compute the quantity :

$$\mu = \sum_{i=1}^{25} (\text{distance } P_iP_i')^2 \tag{5.1}$$

The summation is over the last 25 states of the initial condition and perturbed initial condition. A point P_i, along with its transforms, was classified as belonging to the ergodic region if $\mu > \mu_c$, or to a curve if $\mu \leq \mu_c$, where $\mu_c = 10^{-4}$ is a chosen constant. This criterion allowed for the distinction between regular and ergodic regions in phase space. As for the value of μ_c, even though the authors suggest a cutoff of 10^{-4}, we wanted to verify its behavior for various energies, figure 8 represents the μ values for different initial conditions corresponding to various energies, the authors state that the μ values range from 10^{-12} to 10^{+1} and our diagram is consistent with that, even from this plot it is evident that behavior is chaotic at high energies and stable at lower energies.

The results of the equation (5.1) have been given in the figure 9, as expected it complements all the findings so far both from μ value calculation and Poincaré sections, there appears to be a critical energy ($E = 0.09$) up to which the curves cover the whole area and as we go up, the area covered decreases rapidly going to zero at $E \approx 0.167$, any energy higher than this value means that the particle will eventually escape to infinity, hence it is called energy of escape, the phase space volume goes to infinity, consequently the area of our Poincaré section goes to infinity and relative area losses its meaning. Another important thing to state here is that the critical energy from Hénon &Heiles (1964) is $E = 0.11$, the method is very sensitive to initial conditions, total time of orbit integration, the μ_c criteria, we believe these are the cause of discrepancy between but the overall behavior of the function is similar to the one recorded by authors.

5.2 Gottwald and Melbourne Scheme

The earlier method of chaos determination required computation of the orbits, which is computationally expensive. Melbourne and Gottwald suggested another method called the 0-1 test (Gottwald &Melbourne 2004) which is both independent of the dimensionality of the phase space as well as the underlying equations. It is based on the slope of the distance in phase space along an orbit. The input for this method is a time series data and the output is 0 or 1, depending on whether the dynamics is non-chaotic or chaotic. In this method, we consider an observable $\phi(x)$, it could be anyone from $x, \dot{x}, y, \dot{y}$ or a combination of these. One has to then create a real valued function $p(t)$ as follows:

$$\theta(t) = ct + \int_0^t \phi[x(s)]\,\mathrm{d}s \qquad p(t) = \int_0^t \phi[x(s)]\cos[\theta(s)]\,\mathrm{d}s \tag{5.2}$$

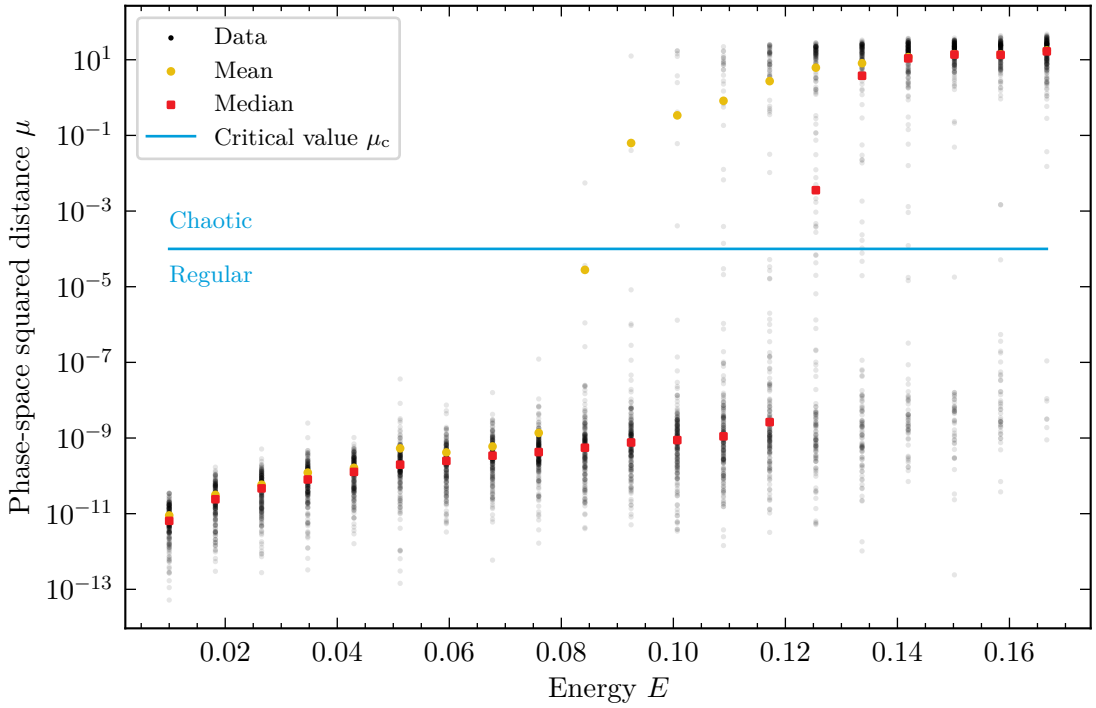

Figure 8: Behavior of μ for various initial conditions for different total energies. Additionally, the mean (yellow circles) and median (red squares) values for each energy is plotted, and the critical value μ_c is showed in light blue.

where c has to be chosen arbitrarily and must be greater than zero ($c > 0$).The main observation is that if the system is non-chaotic, $p(t)$ remains bounded, while for chaotic systems, $p(t)$ behaves similarly to a Brownian motion. In the figure 10, one can see the exact behavior, for the energies less than the critical energy ($E = 0.11$) $p(t)$ is bounded and for higher energies it behaves akin to the path under Brownian motion[6] (Kostrykin et al. 2010). Thus, the former trajectories are regular while the latter ones are chaotic (Gottwald &Melbourne 2004).

To quantitatively analyze the behavior of $p(t)$, the mean-square displacement (MSD) has to be computed as:

$$M(t) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(p(t+\tau) - p(\tau)\right)^2 \, \mathrm{d}\tau \tag{5.3}$$

The growth rate of MSD, defined as $K = \lim_{t\to\infty} \log[M(t)]/\log(t)$ determines the nature of the system[7]: $K = 0$ indicates non-chaotic behavior, while $K = 1$ signifies chaos. The growth rate can be determined numerically using linear regression of $\log[M(t)]$ versus $\log(t)$, Due to lack of time we were not able to rectify our MSD calculation function so we did see the expected behavior of K, we have given the computed value of K for different energies in the table 1, we see the expected increase in the value of K as energy increases.

Energy	0.0100	0.0667	0.0833	0.1111	0.1250	0.1667
Chaos Indicator K	0.0003	0.0088	0.0145	0.0251	0.0423	0.0605

Table 1: K values for different energies.

[6]This behavior is represented significantly in the example shown in Appendix B.1.
[7]To avoid negative logarithms, the authors suggest to use $\lim_{t\to\infty} \log[M(t)+1]/\log(t)$

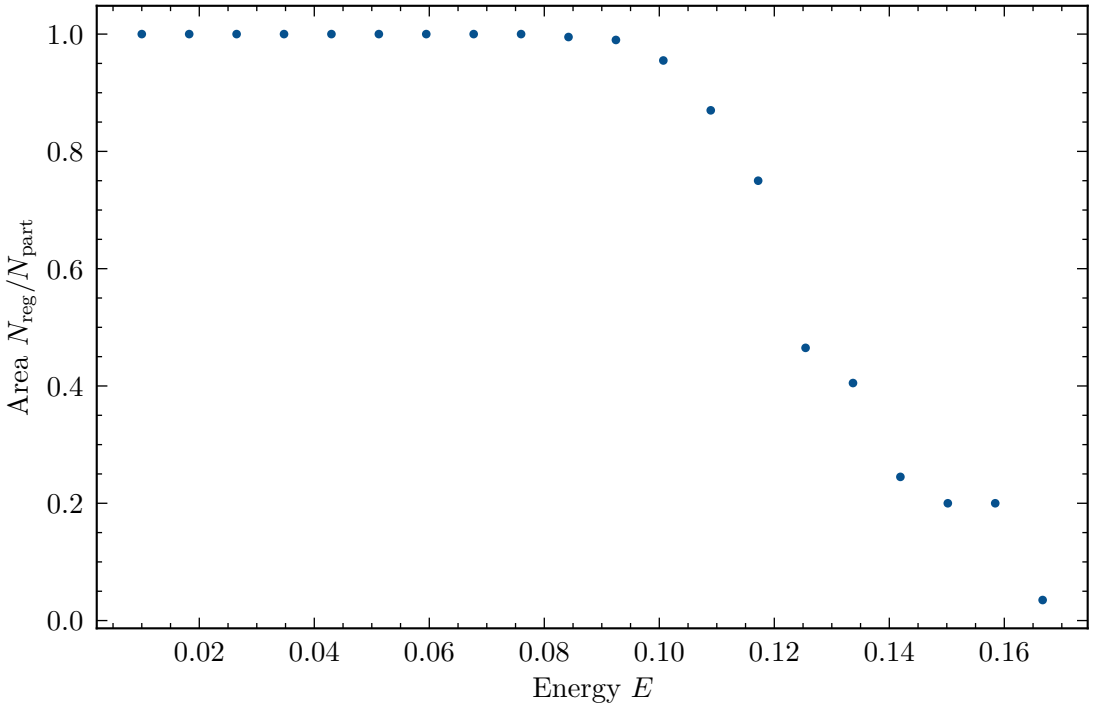

Figure 9: Relative area of the curves as function of energy.

6 Parallelization of the Code

In this section, we will focus on the problem of finding Poincaré sections: first, we generate $N_{\text{part}} = 100$ particles, and integrate their trajectories in the Hénon-Heils potential for $N_{\text{iter}} = 30\,000$ iterations (with a time-step $h = 0.01$). Then, for each particle trajectory, we find two successive points on each sides of $x = 0$, and interpolate its position y and velocity v at $x = 0$.

The most computationally expensive part of the algorithm is the integration of the trajectories. In our first approach of this problem, the algorithm we made performed serial computations, therefore it was not very efficient, and quite long to run.

However, we designed our code to be easily improved by using Numpy arrays and its ability to perform data-based parallelism. Soon after, we made a new version of this algorithm to take advantages of vectorization.

To be able to quantify the improvement of the vectorization, we kept the two versions of the code, and wrote a script to test the wall clock time required to run each code. We also make sure to keep the same (realistic) conditions for the two runs. to get better statistics, the code is run for 5 times (with different total energy values), and we compute the mean wall clock time per iteration of the code. Only the integration and Poincaré section search are timed, the initialization and result writing parts are not counted. We performed two tests, on two different CPU: an 8-core Apple M1 3.2 GHz (1) and the `astromaster` 16-core Intel Xeon CPU E5-2609 v4, 1.70 GHz. (2) Our results are:

1. On Apple M1:
   - for the serial algorithm: $t = (33.13 \pm 0.15)$ s per energy iteration;
   - for the parallel algorithm: $t = (1.38 \pm 0.15)$ s per energy iteration.
2. On `astromaster`:

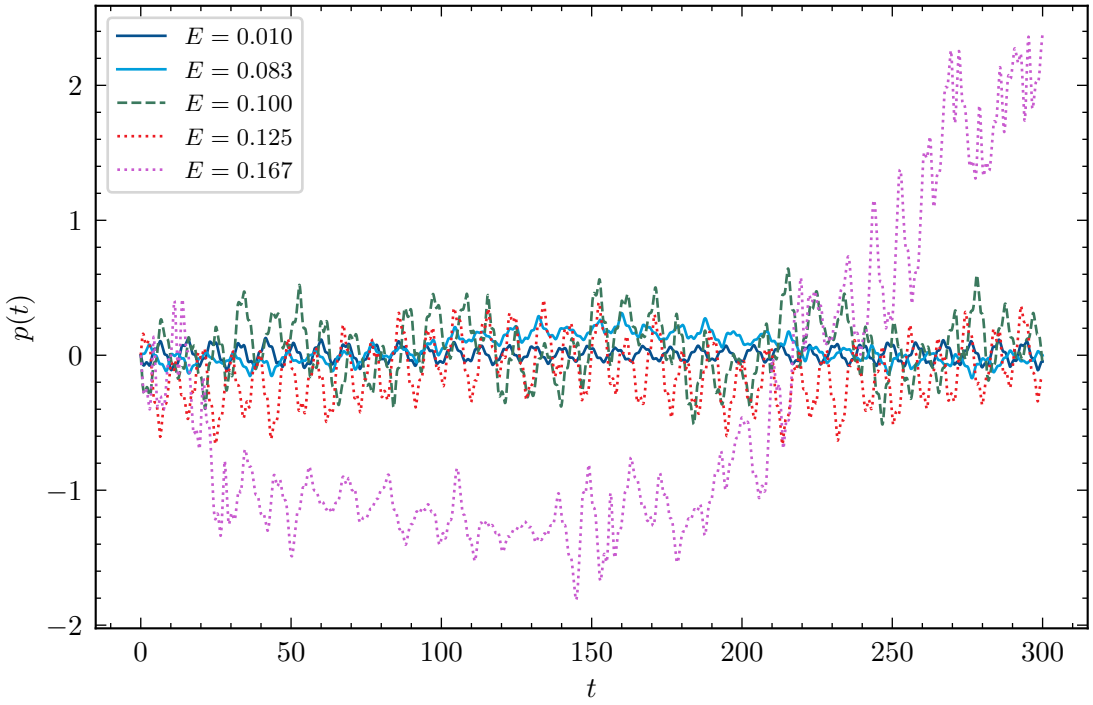

Figure 10: Behavior of $p(t)$ for various energies. The orange trajectory $E = 0.125$ is above the critical energy but here we see a bounded behavior because a significant portion of orbits are still regular,so a significant portion of initial conditions will yield bounded trajectories.

- for the serial algorithm: $t = (123.9 \pm 2.8)$ s per energy iteration;
- for the parallel algorithm: $t = (5.5 \pm 2.8)$ s per energy iteration.

The uncertainty correspond to the standard deviation. Using Numpy arrays reduced the time required to run the script by a factor 20–25 in both machines.

Another possibility for parallelization of the program would have been to implement a task-based parallelization, for instance by computing each of the energy iteration on one core (assuming there are at least five cores available). In this case, the time per energy iteration would stay roughly the same, but the total wall clock time would be divided by roughly a factor 5.

7 Conclusion

In this project, we studied the Hénon–Heiles potential by using numerical methods to understand the dynamics of a particle under this potential and the transition from regular to chaotic behavior. Before employing Runge–Kutta 4^{th} order integrator on the Hénon–Heiles potential, we compared and tested the accuracy as well as the efficiency of the Euler method, RK2 and RK4 for the Kepler two body problem. After establishing that indeed RK4 is the best choice among them, we computed the trajectories of particles under this potential, one in regular regime and another in chaotic regime.

For analysis of overall behavior of the potential, we computed the Poincaré sections, for lower energy values the sections were smooth continuous loops, meaning the orbits were stable and for higher energies the section turned into a sea of chaos. There were also some energy values for which islands of stability were there but they did not fill the whole space, indicating that there is a critical energy separating the regular and chaotic dynamics. In order to find this energy, we plotted the area in the phase space plane covered by curves and found a critical energy of $E = 0.09$

below which all the allowable area is covered by curves and above this it starts to decrease going to nearly 0 at $E = 0.167$, a totally chaotic behavior.

We also tried to verify our previous findings with the Melbourne-Gottwald 0–1 scheme based on the slope of distance in an orbit, we did find the behavior to be chaotic for energies above the critical energy, inferred by plotting $p(t)$ as a function of integration time but were unable to reproduce nearly 0 or nearly 1 values for the growth rate of the mean square displacement as the method suggests, we believe there was some inaccuracy in the calculation of the mean square displacement, but nevertheless we saw an increase in the growth rate of MSD as energies increased, which is expected as $p(t)$ is first bounded for lower energies and then shows Brownian motion-like behavior for higher energies.

Furthermore, to enhance the computational efficiency, we implemented vectorized parallelization technique and also discussed how it can be further improved. In nutshell, we demonstrated the chaotic as well as the regular dynamics of the particles under the Hénon-Heiles potential.

References

Cheb-Terrab, E. & De Oliveira, H. 1996, Computer physics communications, 95, 171

Gottwald, G. A. & Melbourne, I. 2004, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 460, 603

Hénon, M. & Heiles, C. 1964, The Astronomical Journal, 69, 73

Kostrykin, V., Potthoff, J., & Schrader, R. 2010, arXiv preprint arXiv:1008.3761

Poincaré, H. 1893, Les méthodes nouvelles de la mécanique céleste, Vol. 2 (Gauthier-Villars et fils, imprimeurs-libraires)

Strauch, D. 2009, in Classical Mechanics: An Introduction (Springer), 157–182

Appendix

A Evolution of the Energy Error

The fig.11 represents the error between the analytical value of energy and the numerical values of energy calculated using various integrators as the orbit evolves. All these computations were done for equal integration time, the behavior is as expected, with the RK4 being the most accurate one, but what is of interest is that for early orbit integration time, the error between the RK4 method and the analytical expression is less than machine error, so it means the floating point error can't reliably distinguish between two values, the error no longer represents the true physical or mathematical difference between results rather it is dominated by the limits of machine.

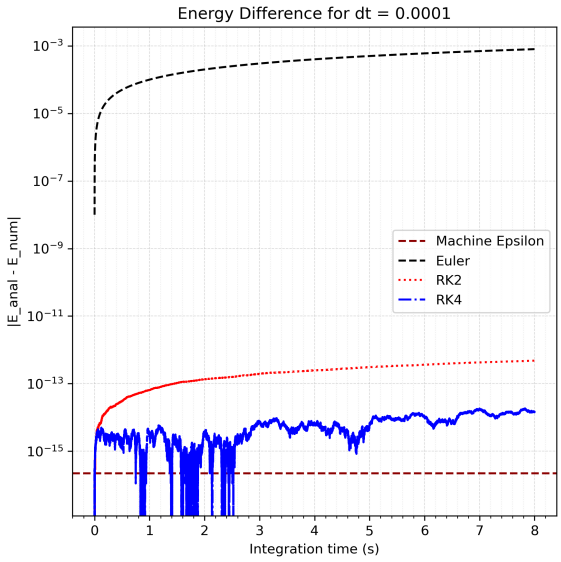

Figure 11: energy error evolution for $\mathrm{d}t = h = 0.0001$

We also calculated the computation time for all integrators for different time steps as given in figure 12, as expected the Euler method is the fastest, followed by RK2 and the RK4. RK4 is almost one order of magnitude slower than the Euler method but given that for $\mathrm{d}t = h = 0.001$ it is more than 4 orders of magnitude better than RK2 and 10 orders of magnitude better than the Euler method(see figure 3).

B Linear Interpolation

Between two consecutive time steps, the particle might have cross $x = 0$ between two time steps, in that case the exact crossing point is not explicitly recorded. To accurately determine the y-coordinate and v_y-velocity at the exact point where a trajectory crosses the plane $x = 0$, linear interpolation is used. The interpolated values are calculated as:

$$y_0 = y[i] + \frac{y[i+1] - y[i]}{x[i+1] - x[i]} \cdot (0 - x[i]),$$

$$v_0 = v[i] + \frac{v[i+1] - v[i]}{x[i+1] - x[i]} \cdot (0 - x[i]).$$

Linear interpolation is valid as the time step is very small $\mathrm{d}t = h = 0.001$ and the particle's motion is nearly linear in this interval.

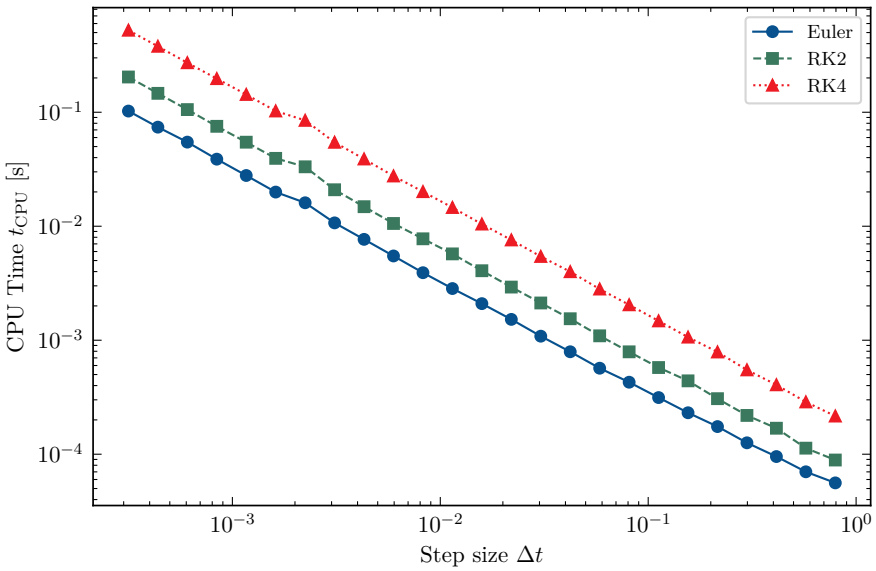

Figure 12: Computation time for various time steps for Euler, RK2 and RK4 scheme

B.1 Melbourne-Gottwald 0–1 test

Here, we present another example of the behavior of $p(t)$, we can clearly see the Brownian motion like behavior of the function above critical energy.

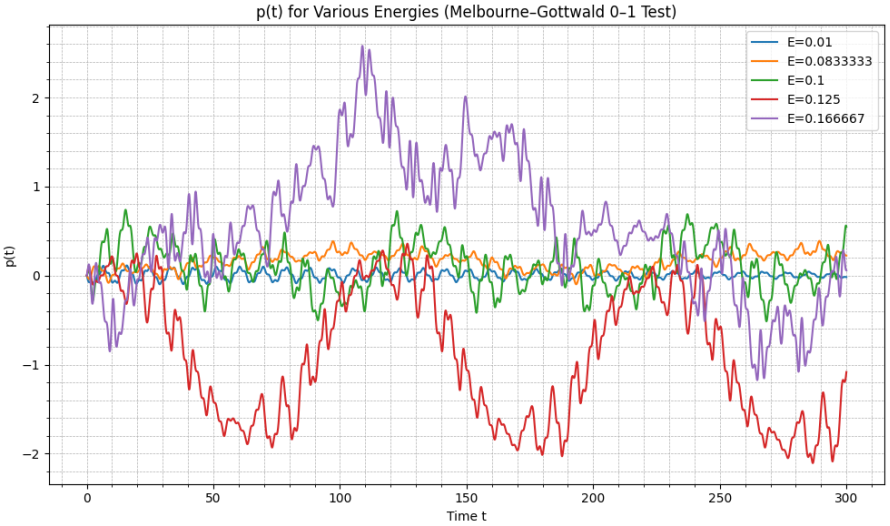

Figure 13: Behavior of $p(t)$ for various energies

B.2 Poincaré Sections

Figure 14 and 15 present a magnified view of the Poincaré sections for various energies in order to see the features of the phase space that are not evident explicitly in the main diagram included in the report.

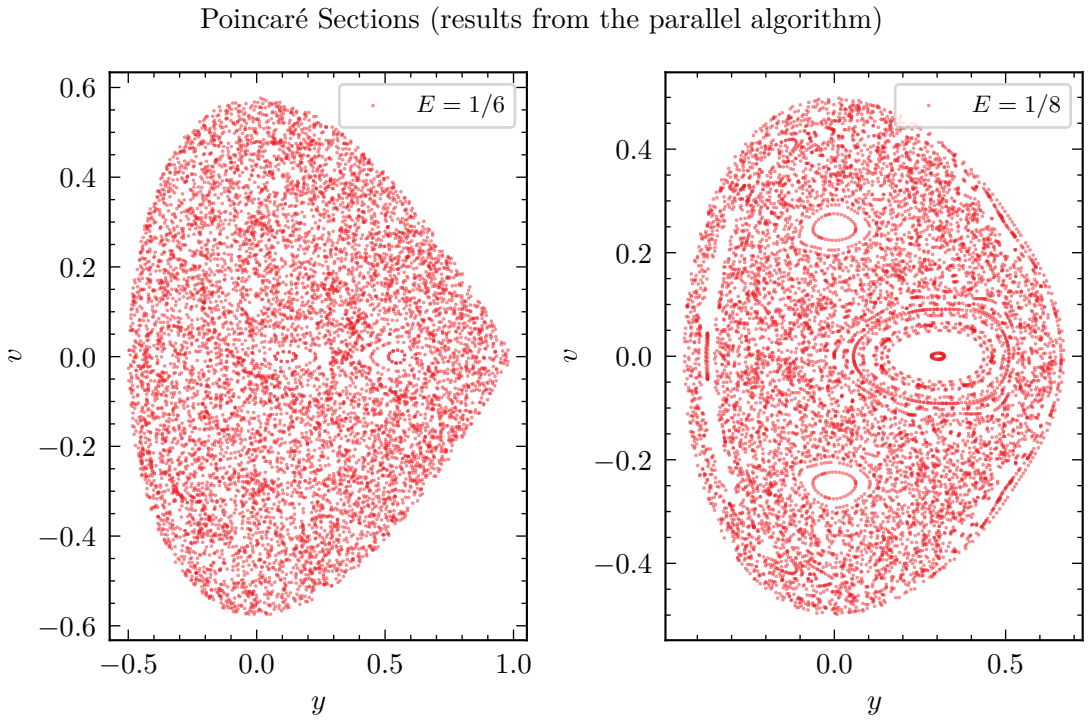

Figure 14: Poincaré section for $E = \frac{1}{6}, \frac{1}{8}$

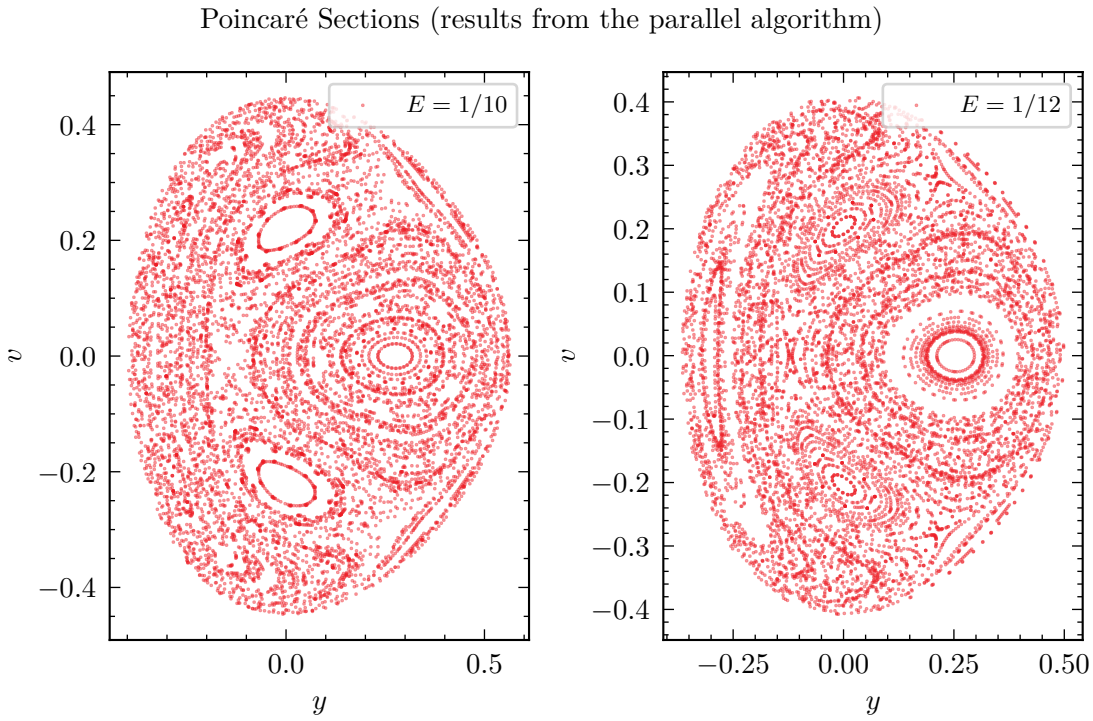

Figure 15: Poincaré section for $E = \frac{1}{10}, \frac{1}{12}$

C Listings

The listings of the codes are provided here. However, the source code is also available on Github[8] (with installing and usage instructions).

C.1 Library Files

file: `energies.py`

```python
#!/usr/bin/env python
"""
Energies

Compute energies (kinetic or total)

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

energies.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
"""

import numpy as np

def kinetic(W: np.ndarray) -> np.ndarray:
    """Computes the kinetic energy.
    @param
        - W: Phase-space vectors
    @returns
        - T: Kinetic energy
    """
    U = W[1,0]
    V = W[1,1]
    # If U or V is not an array (or a list), but rather a scalar, then we
    # create a list of one element so that it can work either way
    if np.ndim(U) == 0: U = np.array([U])
    if np.ndim(V) == 0: V = np.array([V])

    return (U**2 + V**2)/2

def total(W: np.ndarray,
          potential,
          kinetic = kinetic) -> np.ndarray:
    return potential(W) + kinetic(W)
```

[8] https://github.com/Yael-II/MSc2-Project-Chaos

file: `initial_conditions.py`

```python
#!/usr/bin/env python
"""
Initial Conditions

Generate initial conditions depending on different criteria

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

initial_conditions.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
"""

import numpy as np
import matplotlib.pyplot as plt
POS_MIN = -1
POS_MAX = +1
VEL_MIN = -1
VEL_MAX = +1
N_PART = 1

def mesh_grid(N: int = N_PART,
              xmin: float = POS_MIN,
              xmax: float = POS_MAX,
              ymin: float = POS_MIN,
              ymax: float = POS_MAX) -> np.ndarray:
    """Generates a set of regularly sampled particles with no velocity
    @ params:
        - N: number of particles
        - xmin: minimum value for position x
        - xmax: maximum value for position x
        - ymin: minimum value for position y
        - ymax: maximum value for position y
    @ returns:
        - W: phase-space vector
    """
    X = np.linspace(xmin, xmax, N)
    Y = np.linspace(ymin, ymax, N)
    X,Y = np.meshgrid(X,Y, indexing="ij")
    return np.array([X,Y])

def one_part(x0: float = 0,
             y0: float = 0,
             u0: float= 0,
             v0: float = 0) -> np.ndarray:
    """Generates a particle at position (x0, y0)
    with velocities (u0,v0)
    @ params:
```

```
        - N: number of particles
        - x0: initial position x
        - y0: initial position y
        - u0: initial velocity u
        - v0: initial velocity v
    @ returns:
        - W: phase-space vector
    """
    X = x0
    Y = y0
    U = u0
    V = v0
    return np.array([[X,Y], [U,V]])

def n_energy_part(potential,
                  N: int = N_PART,
                  E: float = 0,
                  xmin: float = -1,
                  xmax: float = +1,
                  ymin: float = -0.5,
                  ymax: float = +1):
    """Generates N particles with an energy E in a potential.
    @ params:
        - potential: gravitational potential
        - N: number of particles
        - E: total energy
        - xmin: minimum value for position x
        - xmax: maximum value for position x
        - ymin: minimum value for position y
        - ymax: maximum value for position y
    @ returns:
        - W: an array of all the positions and velocities.
    """
    X = []
    Y = []
    POT = []
    U = []
    V = []
    while len(X) < N:
        x = np.random.random()*(xmax-xmin)+xmin
        y = np.random.random()*(ymax-ymin)+ymin
        w = np.array([x, y])
        pot = potential(w, position_only=True)[0]
        if pot <= E:
            X.append(x)
            Y.append(y)
            POT.append(pot)
    X = np.array(X)
    Y = np.array(Y)
    POT = np.array(POT)
    U = np.zeros_like(X)
    V = np.zeros_like(Y)
    C = np.sqrt(2 * (E - POT))
    THETA = np.random.random(N)*2*np.pi
    U = C*np.cos(THETA)
    V = C*np.sin(THETA)
    return np.array([[X, Y], [U, V]])

def n_energy_2part(potential,
                   N: int = N_PART,
                   E: float = 0,
                   sep: float = 1e-7,
                   xmin: float = -1,
                   xmax: float = +1,
                   ymin: float = -0.5,
                   ymax: float = +1):
    """Generate a sample of 2N particles with the energy E in a potential in
    two sets: one "normal" set (see n_energy_part), and a slightly shifted set
    with a separation sep.
    @ params:
```

```
140         - potential: gravitational potential
141         - N: number of particles
142         - E: total energy
143         - sep: the separation between the two sets
144         - xmin: minimum value for position x
145         - xmax: maximum value for position x
146         - ymin: minimum value for position y
147         - ymax: maximum value for position y
148     @ returns:
149         - (W1, W2): the two arrays of all the positions and velocities for
150         each set.
151     """
152     W_1 = n_energy_part(potential, N, E)
153     W_2 = np.zeros_like(W_1)
154     alpha = np.random.uniform(0, 2*np.pi, N)
155     W_2[0, 0] = W_1[0, 0] + sep*np.cos(alpha)
156     W_2[0, 1] = W_1[0, 1] + sep*np.sin(alpha)
157     W_2[1, 0] = W_1[1, 0]
158     W_2[1, 1] = W_1[1, 1]
159     return (W_1, W_2)
```

file: integrator.py

```python
#!/usr/bin/env python
"""
Integrator

Integrate differential equations.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

integrator.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import numpy as np

def euler(t0: float,
          W0: np.ndarray,
          h: float,
          n: int,
          func):
    """Euler method adapted for state vector [[x, y], [u, v]]
    @ params
        - t0: initial time value
        - W0: initial state vector [[x, y], [u, v]]
        - h: step size (time step)
        - n: number of steps
        - func: RHS of differential equation
    @returns:
        - t, W: time and state (solution) arrays
    """
    time = np.zeros(n)
    W = np.zeros((n,) + np.shape(W0))

    t = t0
    w = W0
    for i in range(n):
        k1 = func(t, w)
        w = w + h*k1
        t = t + h

        time[i] = t
        W[i] = w
    return time, W

def rk2(t0: float,
        W0: np.ndarray,
        h: float,
        n: int,
```

```
70          func):
71      """RK2 method adapted for state vector [[x, y], [u, v]]
72      @ params
73          - t0: initial time value
74          - W0: initial state vector [[x, y], [u, v]]
75          - h: step size (time step)
76          - n: number of steps
77          - func: RHS of differential equation
78      @returns:
79          - t, W: time and state (solution) arrays
80      """
81      time = np.zeros(n)
82      W = np.zeros((n,) + np.shape(W0))
83
84      t = t0
85      w = W0
86      for i in range(n):
87          k1 = func(t, w)
88          k2 = func(t + h/2, w + h/2*k1)
89
90          w = w + h*k2
91          t = t + h
92
93          time[i] = t
94          W[i] = w
95      return time, W
96
97  def rk4(t0: float,
98          W0: np.ndarray,
99          h: float,
100         n: int,
101         func):
102     """RK4 method adapted for state vector [[x, y], [u, v]]
103     @ params
104         - t0: initial time
105         - W0: initial state vector [[x, y], [u, v]]
106         - h: step size (time step)
107         - n: number of steps
108         - func: RHS of differential equation
109     @returns:
110         - t, W: time and state (solution) arrays
111     """
112     time = np.zeros(n)
113     W = np.zeros((n,) + np.shape(W0))
114     # to accommodate the state vector
115     t = t0
116     w = W0
117     for i in range(n):
118         k1 = func(t, w)
119         k2 = func(t + h/2, w + h/2*k1)
120         k3 = func(t + h/2, w + h/2*k2)
121         k4 = func(t + h, w + h*k3)
122
123         w = w + h*(k1/6 + k2/3 + k3/3 + k4/6)
124         t = t + h
125
126         time[i] = t
127         W[i] = w
128     return time, W
129
130 def integrator_type(t0, W0, h, n, func, integrator):
131     return integrator(t0, W0, h, n, func)
132
133 def kepler_analytical(t0: float,
134                       W0: np.ndarray,
135                       h: float,
136                       n: int):
137     """Computes the evolution from the Kepler potential derivative
138     @ params
139         - t0: initial time value
```

```
140         - W0: initial state vector [[x, y], [u, v]]
141         - h: step size (time step)
142         - n: number of steps
143     @returns:
144         - t, W: time and state (solution) arrays
145     """
146     X0 = W0[0 ,0]
147     Y0 = W0[0, 1]
148     U0 = W0[1, 0]
149     V0 = W0[1, 1]
150 
151     time = np.arange(t0, t0 + n*h, h)
152     W = np.zeros((n,) + np.shape(W0))
153 
154     R0 = np.sqrt(X0**2 + Y0**2)
155     Omega0 = np.sqrt(U0**2 + V0**2)/R0
156 
157     X = R0 * np.cos(Omega0 * time)
158     Y = R0 * np.sin(Omega0 * time)
159     U = -R0 * Omega0 * np.sin(Omega0 * time)
160     V = R0 * Omega0 * np.cos(Omega0 * time)
161 
162     W = np.array([[X, Y], [U, V]])
163     W = np.swapaxes(W, 0, 2)
164     W = np.swapaxes(W, 1, 2)
165     return time, W
```

file: poincare_sections.py

```
#!/usr/bin/env python
"""
Poincare Sections

Computes the Poincare Sections

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

poincare_sections.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
"""

import numpy as np
def pcs_find(pos_x, pos_y, vel_x, vel_y):
    """Find Poincare sections (PCS; x = 0)
    @ params:
        - pos_x: position along the x axis
        - pos_y: position along the y axis
        - pos_x: velocity along the x axis
        - pos_y: velocity along the y axis
    @ returns: (tuple)
        - pcs_pos_y: position of the points in the PCS along the y axis
        - pcs_vel_y: velocity of the points in the PCS along the y axis
    """
    if np.ndim(pos_x) == 1:
        pos_x = np.array([pos_x])
        pos_y = np.array([pos_y])
        vel_x = np.array([vel_x])
        vel_y = np.array([vel_y])
    pcs_pos_y = []
    pcs_vel_y = []
    for j in range(len(pos_x[0])): # for each particle
        i = 0
        x = pos_x[:,j]
        y = pos_y[:,j]
        u = vel_x[:,j]
        v = vel_y[:,j]
        while i < len(x) - 1:
            if x[i] * x[i+1] < 0:
                y0 = y[i] \
                        + (y[i+1] - y[i])/(x[i+1] - x[i]) \
                        * (0 - x[i])
                v0 = v[i] \
                        + (v[i+1] - v[i])/(x[i+1] - x[i])\
                        * (0 - x[i])
                pcs_pos_y.append(y0)
                pcs_vel_y.append(v0)
```

```
            i += 1
    return pcs_pos_y, pcs_vel_y

def pcs_find_legacy(pos_x, pos_y, vel_x, vel_y):
    """DEPRECIATED - DO NOT USE
    Depreciated legacy function that should not be used
    """
    if np.ndim(pos_x) == 1:
        pos_x = np.array([pos_x])
        pos_y = np.array([pos_y])
        vel_x = np.array([vel_x])
        vel_y = np.array([vel_y])
    pcs_pos_y = []
    pcs_vel_y = []
    for j in range(len(pos_x)): # for each particle
        i = 0
        x = pos_x[j]
        y = pos_y[j]
        u = vel_x[j]
        v = vel_y[j]
        while i < len(x) - 1:
            if x[i] * x[i+1] < 0:
                y0 = y[i] \
                        + (y[i+1] - y[i])/(x[i+1] - x[i]) \
                        * (0 - x[i])
                v0 = v[i] \
                        + (v[i+1] - v[i])/(x[i+1] - x[i])\
                        * (0 - x[i])
                pcs_pos_y.append(y0)
                pcs_vel_y.append(v0)
            i += 1
    return pcs_pos_y, pcs_vel_y
```

file: potentials.py

```python
#!/usr/bin/env python
"""
Potentials

Functions of the different potentials (and their derivatives for the evolution)

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

potentials.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""

import numpy as np

MAX_VAL = 1e3

def kepler_potential(W_grid: np.ndarray,
                     position_only: bool = False) -> np.ndarray:
    """Computes the Kepler potential: V(R) = -G*m1*m2/R
    (assuming G = 1, m1 = 1, m2 = 1)
    assuming the point mass at (x = 0, y = 0).
    @params:
        - W: Phase-space vector
        - position_only: True if W is np.array([X, Y])
    @returns:
        - computed potential
    """
    if position_only:
        X = W_grid[0]
        Y = W_grid[1]
    else:
        X = W_grid[0,0]
        Y = W_grid[0,1]
    # If X or Y is not an array (or a list), but rather a scalar, then we
    # create a list of one element so that it can work either way
    if np.ndim(X) == 0: X = np.array([X])
    if np.ndim(Y) == 0: Y = np.array([Y])
    R = np.sqrt(X**2 + Y**2)
    return -1/R

def kepler_evolution(t: np.ndarray, W: np.ndarray):
    """Computes the evolution from the Kepler potential derivative
    @params
        - t: Time (not used)
        - W: Phase space vector
    &returns
```

```
70          - dot W: Time derivative of the phase space vector
71      """
72      X = W[0 ,0]
73      Y = W[0, 1]
74      U = W[1, 0]
75      V = W[1, 1]
76      R = np.sqrt(X**2 + Y**2)
77      DX = U
78      DY = V
79      DU = -X/R**3
80      DV = -Y/R**3
81      return np.array([[DX, DY], [DU, DV]])
82
83  def hh_potential(W_grid: np.ndarray,
84                   position_only=False) -> np.ndarray:
85      """Computes the Henon-Heiles potential.
86      @params:
87          - W: Phase-space vector
88          - position_only: True if W is np.array([X, Y])
89      @returns:
90          - POT: Potential
91      """
92      if position_only:
93          X = W_grid[0]
94          Y = W_grid[1]
95      else:
96          X = W_grid[0, 0]
97          Y = W_grid[0, 1]
98
99      # If X or Y is not an array (or a list), but rather a scalar, then we
100     # create a list of one element so that it can work either way
101     if np.ndim(X) == 0: X = np.array([X])
102     if np.ndim(Y) == 0: Y = np.array([Y])
103
104     POT = (X**2 + Y**2 + 2*X**2*Y - 2*Y**3/3)/2
105     return POT
106
107 def hh_evolution(t: np.ndarray, W: np.ndarray):
108     """Computes the evolution from the HH potential derivative
109     @params
110         - t: Time (not used)
111         - W: Phase space vector
112     &returns
113         - dot W: Time derivative of the phase space vector
114     """
115     X = W[0 ,0]
116     Y = W[0, 1]
117     U = W[1, 0]
118     V = W[1, 1]
119     DX = U
120     DY = V
121     DU = -(2*X*Y + X)
122     DV = -(X**2 - Y**2 + Y)
123     return np.array([[DX, DY], [DU, DV]])
```

C.2 Main Files

file: `main_area.py`

```python
#!/usr/bin/env python
"""
Main: Compute Relative Area

Computes the relative area covered bu the curves for different energies, to
study ordered and chaotic regimes.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

main_area.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import numpy as np
from scipy.optimize import curve_fit

import potentials as pot
import energies as ene
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

OUT_DIR = "./Output/"
FILENAME_PREFIX = "phase_separation_"
EXTENSION = ".csv"
DEFAULT_N_iter = int(1e5)
DEFAULT_N_part = 200
DEFAULT_h = 0.005
E_all = np.linspace(1/100, 1/6, 20)

def compute_mu(E: float,
               N_iter: int = DEFAULT_N_iter,
               N_part: int = DEFAULT_N_part,
               h: float = DEFAULT_h) -> tuple:
    """
    Computes the phase-space squared distances for particles of given energy E.
    @params:
        - E: the total energy of each particles
        - N_iter: the number of iteration
        - N_part: the number of particles
        - h: integration steps
    @returns:
        - mu: phase-space squared distance
    """
    W_1, W_2 = init.n_energy_2part(pot.hh_potential, N_part, E)
```

```
68      t_1, positions_1 = itg.rk4(0, W_1, h, N_iter, pot.hh_evolution)
69      x_1 = positions_1[:, 0, 0]
70      y_1 = positions_1[:, 0, 1]
71      u_1 = positions_1[:, 1, 0]
72      v_1 = positions_1[:, 1, 1]
73
74      t_2, positions_2 = itg.rk4(0, W_2, h, N_iter, pot.hh_evolution)
75      x_2 = positions_2[:, 0, 0]
76      y_2 = positions_2[:, 0, 1]
77      u_2 = positions_2[:, 1, 0]
78      v_2 = positions_2[:, 1, 1]
79      dist_sq = (x_2[-25:] - x_1[-25:])**2 \
80              + (y_2[-25:] - y_1[-25:])**2 \
81              + (u_2[-25:] - u_1[-25:])**2 \
82              + (v_2[-25:] - v_1[-25:])**2
83
84      mu = np.sum(dist_sq, axis=0)
85      return mu
86
87  if __name__ == "__main__":
88      mu_all = []
89      for i in range(len(E_all)):
90          mu = compute_mu(E_all[i])
91          filename = OUT_DIR + FILENAME_PREFIX\
92                   + str(i) + EXTENSION
93          np.savetxt(filename, mu)
```

file: main_poincare_sections_linear.py

```python
#!/usr/bin/env python
"""
Main: Computes Poincare Sections (Linear Algorithm)

Computes the Poincare Sections with a linear algorithm
(i.e. no parallel computing).

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

main_poincare_sections_linear.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import numpy as np

import potentials as pot
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

# Parameters
OUT_DIR = "./Output/"
FILENAME_PREFIX = "poincare_sections_linear_"
EXTENSION = ".csv"
DEFAULT_N_iter = 30000
DEFAULT_N_part = 100
DEFAULT_h = 0.01
E_all = np.array([1/100, 1/12, 1/10, 1/8, 1/6])

text_E = ["1/100", "1/12", "1/10", "1/8", "1/6"]

def compute_poincare_sections_linear(E: float,
                                     N_iter: int = DEFAULT_N_iter,
                                     N_part: int = DEFAULT_N_part,
                                     h: float = DEFAULT_h) -> tuple:
    """
    Computes the Poincare sections for a given energy E.
    @params:
        - E: the total energy of each particles
        - N_iter: the number of iteration
        - N_part: the number of particles
        - h: integration steps
    @returns:
        - y_section, v_section: arrays containing the y and v coordinates of
          the Poincare sections
    """
    W_all_part = init.n_energy_part(pot.hh_potential, N_part, E)
```

```
70      y_section = []
71      v_section = []
72      for i in range(N_part):
73          W_part = W_all_part[:,:,i]
74
75          # Perform integration
76          t_part, coord_part = itg.rk4(0, W_part, h, N_iter, pot.hh_evolution)
77
78          # Extract positions and velocities
79          x_part = coord_part[:, 0, 0]
80          y_part = coord_part[:, 0, 1]
81          u_part = coord_part[:, 1, 0]
82          v_part = coord_part[:, 1, 1]
83
84          # Find Poincare section points for the current initial condition
85          y_pcs, v_pcs = pcs.pcs_find_legacy(x_part, y_part, u_part, v_part)
86          # The legacy is important here, the algorithm is the same but the
87          # data format is different...
88
89          # Append the current Poincare section points to the overall lists
90          y_section += y_pcs
91          v_section += v_pcs
92      return y_section, v_section
93
94  if __name__ == "__main__":
95      y_section_all = []
96      v_section_all = []
97      for i in range(len(E_all)):
98          y_section, v_section = compute_poincare_sections_linear(E_all[i])
99          section = np.array([y_section, v_section])
100         filename = OUT_DIR + FILENAME_PREFIX\
101                 + str(text_E[i][2:]) + EXTENSION
102         np.savetxt(filename, section)
```

file: `main_poincare_sections_parallel.py`

```python
#!/usr/bin/env python
"""
Main: Computes Poincare Sections (Parallel Algorithm)

Computes the Poincare Sections with a parallel algorithm (with Numpy)

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

main_poincare_sections_parallel.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import numpy as np

import potentials as pot
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

# Parameters
OUT_DIR = "./Output/"
FILENAME_PREFIX = "poincare_sections_parallel_"
EXTENSION = ".csv"
DEFAULT_N_iter = 30000
DEFAULT_N_part = 100
DEFAULT_h = 0.01
E_all = np.array([1/100, 1/12, 1/10, 1/8, 1/6])

text_E = ["1/100", "1/12", "1/10", "1/8", "1/6"]

def compute_poincare_sections_numpy(E: float,
                                    N_iter: int = DEFAULT_N_iter,
                                    N_part: int = DEFAULT_N_part,
                                    h: float = DEFAULT_h) -> tuple:
    """
    Computes the Poincare sections for a given energy E.
    @params:
        - E: the total energy of each particles
        - N_iter: the number of iteration
        - N_part: the number of particles
        - h: integration steps
    @returns:
        - y_section, v_section: arrays containing the y and v coordinates of
          the Poincare sections
    """
    W_part = init.n_energy_part(pot.hh_potential, N_part, E)
    y_section = []
```

```
    v_section = []

    # Perform integration
    t_part, coord_part = itg.rk4(0, W_part, h, N_iter, pot.hh_evolution)

    # Extract positions and velocities
    x_part = coord_part[:, 0, 0]
    y_part = coord_part[:, 0, 1]
    u_part = coord_part[:, 1, 0]
    v_part = coord_part[:, 1, 1]

    # Find Poincare section points for the current initial condition
    y_section, v_section = pcs.pcs_find(x_part, y_part, u_part, v_part)
    return y_section, v_section

if __name__ == "__main__":
    y_section_all = []
    v_section_all = []
    for i in range(len(E_all)):
        y_section, v_section = compute_poincare_sections_numpy(E_all[i])
        section = np.array([y_section, v_section])
        filename = OUT_DIR + FILENAME_PREFIX\
                 + str(text_E[i][2:]) + EXTENSION
        np.savetxt(filename, section)
```

C.3 Plot Files

file: `plot_area.py`

```python
#!/usr/bin/env python
"""
Plot: Area

Plots areas

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

plot_area.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import os
import numpy as np
import matplotlib.pyplot as plt

if "YII_1" in plt.style.available: plt.style.use("YII_1")

OUT_DIR = "./Output/"
FILENAME_PREFIX = "phase_separation_"
EXTENSION = ".csv"

def plot_area(filelist: list, mu_c = 1e-4) -> int:
    """
    Plot all the Poincare sections in the file list.
    @params:
        - filelist: the list of files in the output directory, with the format
        "poincare_sections_{linear, parallel}_[1/E].csv"
        - title: title of the figure
    @returns:
        - 0.
    """
    orderlist = np.argsort([(int(file
                                 .replace(FILENAME_PREFIX, "")
                                 .replace(EXTENSION, "")))
                            for file in filelist])
    filelist = np.array(filelist)[orderlist]
    N = len(filelist)
    E =  np.linspace(1/100, 1/6, N)
    mu = []
    for filename in filelist:
        with open(OUT_DIR + filename) as file:
            data = file.readlines()
            data = [np.float64(d.replace("\n", "")) for d in data]
            mu.append(data)
```

```
    mu = np.array(mu)

    fig, ax = plt.subplots(1)
    ax.scatter([], [], s=1, color="k", label="Data")
    for i in range(len(mu)):
        Y = mu[i]
        ax.scatter([E[i]]*len(Y), Y, s=1, color="k", alpha=0.1)
    ax.scatter(E, np.mean(mu, axis=1), s=5,
               color="C1", marker="o", label="Mean")
    ax.scatter(E, np.median(mu, axis=1), s=5,
               color="C3", marker="s", label="Median")
    ax.plot(E, [mu_c]*len(E),
            color="C5", label="Critical value $\\mu_\\mathrm{{c}}$")
    ax.text(0.01, 1e-5, "Regular", va="bottom", ha="left", color="C5")
    ax.text(0.01, 1e-3, "Chaotic", va="top", ha="left", color="C5")
    ax.set_xlabel("Energy $E$")
    ax.set_ylabel("Phase-space squared distance $\\mu$")
    ax.set_yscale("log")
    ax.legend()
    fig.savefig("Figs/mu.pdf")

    fig, ax = plt.subplots(1)
    N_reg = np.count_nonzero(mu < mu_c, axis=1)
    N = np.shape(mu)[1]
    Area = N_reg / N
    ax.scatter(E, Area, s=5, color="C0")
    ax.set_xlabel("Energy $E$")
    ax.set_ylabel("Area $N_\\mathrm{{reg}}/N_\\mathrm{{part}}$")
    fig.savefig("Figs/area.pdf")
    return 0

filelist = [f for f in os.listdir(OUT_DIR) if FILENAME_PREFIX in f]
plot_area(filelist)
plt.show()
```

file: plot_poincare_sections.py

```python
#!/usr/bin/env python
"""
Plot: Poincare Sections (Linear and Parallel)

Plots the Poincare sections for different energies, computed either with linear
or parallel algorithms.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

plot_poincare_sections.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import os
import numpy as np
import matplotlib.pyplot as plt

if "YII_1" in plt.style.available: plt.style.use("YII_1")

OUT_DIR = "./Output/"
FILENAME_PREFIX = "poincare_sections_"
EXTENSION = ".csv"

def plot_poincare_sections(filelist: list, title:str = "") -> int:
    """
    Plot all the Poincare sections in the file list.
    @params:
        - filelist: the list of files in the output directory, with the format
        "poincare_sections_{linear, parallel}_[1/E].csv"
        - title: title of the figure
    @returns:
        - 0.
    """
    orderlist = np.argsort([(int(file
                                 .replace(FILENAME_PREFIX, "")
                                 .replace(EXTENSION, "")
                                 .replace("linear_", "")
                                 .replace("parallel_", "")))
                            for file in filelist])
    filelist = np.array(filelist)[orderlist]
    N = len(filelist)
    fig, axs = plt.subplot_mosaic("ABC\nDEF")
    axs = list(axs.values())
    fig.suptitle(title)
    for i in range(N):
        ax = axs[i]
        filename = filelist[i]
```

```
        inv_E = (filename
                 .replace(FILENAME_PREFIX, "")
                 .replace(EXTENSION, "")
                 .replace("linear_", "")
                 .replace("parallel_", ""))
        data = np.loadtxt(OUT_DIR + filename)
        y_section = data[0]
        v_section = data[1]
        ax.scatter(y_section, v_section,
                   s=.1, color="C3", marker=",", alpha=0.5,
                   label="$E = 1/{}$".format(inv_E))
        ax.set_xlabel("$y$")
        ax.set_ylabel("$v$")
        ax.legend(loc="upper right")
    while i < N:
        i += 1
        axs[i].axis('off')
    if "linear" in title: kind = "linear"
    elif "parallel" in title: kind = "parallel"
    else: kind = "error"
    fig.savefig("Figs/pcs_{}.pdf".format(kind))
    return 0
print("\033[32m"
      + "[P]arallel or [L]inear algorithm result, or [B]oth?"
      + "\033[0m")
answer = input("\033[32m" + "> " + "\033[0m").upper()

if answer == "P":
    FILENAME_PREFIX += "parallel_"
elif answer == "L":
    FILENAME_PREFIX += "linear_"

filelist = [fname for fname in os.listdir(OUT_DIR) if FILENAME_PREFIX in fname]

if answer in ["L", "B"]:
    filelist_linear = [fname for fname in filelist if "linear_" in fname]
    plot_poincare_sections(filelist_linear,
                           title=("Poincare Sections "
                                  "(results from the linear algorithm)"))
if answer in ["P", "B"]:
    filelist_parallel = [fname for fname in filelist if "parallel_" in fname]
    plot_poincare_sections(filelist_parallel,
                           title=("Poincare Sections "
                                  "(results from the parallel algorithm)"))

plt.show()
```

C.4 Test Files

file: `test_evolution.py`

```
#!/usr/bin/env python
"""
Test: Evolution

Evolve a particle with a given energy in a potential and show the result path
followed by the particle in a given time.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

test_evolution.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""

import numpy as np
import matplotlib.pyplot as plt
import potentials as pot
import energies as ene
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

if "YII_1" in plt.style.available: plt.style.use("YII_1")
# Loading matplotlib style...

# Constants
N_grid = 5000
h = 0.01
N_inter = 5000
eps = 1e-2

x_0 = 0.0
y_0 = 0.5
E = 0.125

# Main
if __name__ == "__main__":
    W_part = init.one_part(x_0, y_0,0,0)
    POT = pot.hh_potential(W_part)[0]
    u_0 = np.sqrt(2 * (E - POT))
    v_0 = 0.0

    W_part[1,0] = u_0
    W_part[1,1] = v_0

```

```
68      pos_t, positions = itg.rk4(0, W_part, h, N_inter, pot.hh_evolution)
69
70      pos_x = positions[:,0,0]
71      pos_y = positions[:,0,1]
72      vel_x = positions[:,1,0]
73      vel_y = positions[:,1,1]
74
75      W_grid = init.mesh_grid(N_grid)
76      X_grid = W_grid[0]
77      Y_grid = W_grid[1]
78      potential = pot.hh_potential(W_grid, position_only=True)
79
80      fig, ax = plt.subplots(1)
81
82      pcm = ax.pcolormesh(X_grid, Y_grid, potential)
83      line = ax.plot(pos_x, pos_y, color="C3")
84      fig.colorbar(pcm, label="potential")
85
86      ax.set_xlabel("$x$")
87      ax.set_ylabel("$y$")
88      ax.set_aspect("equal")
89
90      plt.savefig("Figs/evolution.png")
91
92      plt.show(block=True)
```

file: test_evolution_chaotic.py

```
#!/usr/bin/env python
"""
Test: Evolution

Evolve a particle with a given energy in a potential and show the result path
followed by the particle in a given time.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

test_evolution.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""

import numpy as np
import matplotlib.pyplot as plt
import potentials as pot
import energies as ene
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

if "YII_1" in plt.style.available: plt.style.use("YII_1")
# Loading matplotlib style...

# Constants
N_grid = 5000
h = 0.01
N_inter = 5000
eps = 1e-2

x_0 = 0.3
y_0 = 0.1
E = 1/6

# Main
if __name__ == "__main__":
    W_part = init.one_part(x_0, y_0,0,0)
    POT = pot.hh_potential(W_part)[0]
    u_0 = np.sqrt(2 * (E - POT))
    v_0 = 0.0

    W_part[1,0] = u_0
    W_part[1,1] = v_0

    pos_t, positions = itg.rk4(0, W_part, h, N_inter, pot.hh_evolution)

```

```
70      pos_x = positions[:,0,0]
71      pos_y = positions[:,0,1]
72      vel_x = positions[:,1,0]
73      vel_y = positions[:,1,1]
74
75      W_grid = init.mesh_grid(N_grid)
76      X_grid = W_grid[0]
77      Y_grid = W_grid[1]
78      potential = pot.hh_potential(W_grid, position_only=True)
79
80      fig, ax = plt.subplots(1)
81
82      pcm = ax.pcolormesh(X_grid, Y_grid, potential)
83      line = ax.plot(pos_x, pos_y, color="C3")
84      fig.colorbar(pcm, label="potential")
85
86      ax.set_xlabel("$x$")
87      ax.set_ylabel("$y$")
88      ax.set_aspect("equal")
89
90      plt.savefig("Figs/evolution_chaotic.png")
91
92      plt.show(block=True)
```

file: test_initial_E.py

```python
#!/usr/bin/env python
"""
Test: Initial Conditions With a Given Energy

Sample random particles with the same given energy in a valid coordinates range.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

test_initial_E.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.

"""
import numpy as np
import matplotlib.pyplot as plt

import potentials as pot
import energies as ene
import integrator as itg
import initial_conditions as init
import poincare_sections as pcs

if "YII_1" in plt.style.available: plt.style.use("YII_1")
# Loading matplotlib style...
# parameters
N_grid = 1000
N_inter = 30000
N_part = 100
E = 1/12
h = 0.01

if __name__ == "__main__":
    # Initial conditions
    W_grid = init.mesh_grid(N_grid, xmin=-1, xmax=1, ymin=-1, ymax=1)
    X_grid = W_grid[0]
    Y_grid = W_grid[1]
    potential = pot.hh_potential(W_grid, position_only=True)
    pot_valid = np.ma.masked_where(potential > E, potential)

    W_all_part = init.n_energy_part(pot.hh_potential, N_part, E)

    # Plot
    fig, ax = plt.subplots(1)

    pcm = ax.pcolormesh(X_grid, Y_grid, pot_valid, vmin = 0)
    sct = ax.scatter(W_all_part[0, 0], W_all_part[0, 1], s=1, color="C3")
    fig.colorbar(pcm, label="potential")
    ax.set_title("$E = {:.2f}$".format(E))
```

```
70      ax.set_xlabel("$x$")
71      ax.set_ylabel("$y$")
72      ax.set_aspect("equal")
73
74      fig.savefig("Figs/initial_E.png")
75
76      plt.show(block=True)
```

file: `test_integrators.py`

```
"""
Test: Integrators

Demonstrating Keplerian 2-body orbits using various integrators,
and comparing accuracy and runtime over a range of step sizes.

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

test_integrators.py
Copyright (C) 2025 Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
"""

import numpy as np
import matplotlib.pyplot as plt
import time

import integrator as itg
import initial_conditions as init
import potentials as pot
import energies as ene

from matplotlib.patches import ConnectionPatch

if "YII_1" in plt.style.available: plt.style.use("YII_1")

# ----------------------------
# 1. Setup & global parameters
# ----------------------------
t0 = 0.0
T_final = 8.0
W0 = init.one_part(1, 0, 0, 1)  # [x, y, vx, vy]

h_range = np.logspace(-3.5, -0.1, 25)
h_range = np.append(h_range, 0.001)

# For plotting lines/colors
methods = [
    ("Euler", itg.euler, 'o--', 'C0'),
    ("RK2",   itg.rk2,   's--', 'C2'),
    ("RK4",   itg.rk4,   '^--', 'C3')
]
colors = {'Analytical': 'k',
          'Euler': 'C0',
          'RK2': 'C2',
          'RK4': 'C3'}

# Compute machine epsilon
eps = 1.0
```

```
while 1.0 + eps/2 > 1.0:
    eps /= 2.0
print(f"Machine epsilon: {eps}")

# Arrays to store final energy errors & times
err_euler, err_rk2, err_rk4 = [], [], []
time_euler, time_rk2, time_rk4 = [], [], []

# ----------------------------------------
# 2. Main loop over step sizes h in h_range
# ----------------------------------------

fig, ax = plt.subplots(1)

for h in h_range:
    ax.cla()
    N = int(T_final / h)

    # Analytical solution
    t_ana, W_ana = itg.kepler_analytical(t0, W0, h, N)
    W_ana_E = np.swapaxes(W_ana, 0, 2)
    W_ana_E = np.swapaxes(W_ana_E, 0, 1)
    E_analytical_final = ene.total(W_ana_E, pot.kepler_potential)

    # Numerical integrators + timing
    all_solutions = {}
    for (label, method, *_), store_err, store_t in zip(
        methods,
        [err_euler, err_rk2, err_rk4],
        [time_euler, time_rk2, time_rk4]
    ):
        start_time = time.time()
        t_num, W_num = itg.integrator_type(t0, W0, h, N, pot.kepler_evolution,
            method)
        elapsed = time.time() - start_time

        store_t.append(elapsed)

        # Final energy error
        W_num_E = np.swapaxes(W_num, 0, 2)
        W_num_E = np.swapaxes(W_num_E, 0, 1)
        E_numerical_final = ene.total(W_num_E, pot.kepler_potential)
        store_err.append(np.max(abs(E_analytical_final - E_numerical_final)))

        all_solutions[label] = W_num

    # Orbit plot (optional, can comment out if too many figures)
    eu_vals  = all_solutions["Euler"]
    rk2_vals = all_solutions["RK2"]
    rk4_vals = all_solutions["RK4"]


    ax.plot(W_ana[:, 0, 0],
            W_ana[:, 0, 1],
            "-.",
            color=colors['Analytical'],
            label="Analytical",
            zorder=4)
    ax.plot(eu_vals[:, 0, 0],
            eu_vals[:, 0, 1],
            "-",
            color=colors['Euler'],
            label="Euler")
    ax.plot(rk2_vals[:, 0, 0],
            rk2_vals[:, 0, 1],
            "--",
            color=colors['RK2'],
            label="RK2")
    ax.plot(rk4_vals[:, 0, 0],
            rk4_vals[:, 0, 1],
```

```
            ":",
            color=colors['RK4'],
            label="RK4")

    ax.set_title("$\\Var{{t}} = {:.4f}$".format(h))
    ax.set_xlabel("$x$")
    ax.set_ylabel("$y$")
    ax.set_aspect("equal")
    ax.legend(loc="upper right")
    fig.tight_layout()
    fig.savefig("Figs/orbit_dt_{:.4f}.pdf".format(h))

    if h == h_range[-1]:
        mosaic = ("AB\n"
                  "AC")
        fig, axs = plt.subplot_mosaic(mosaic)
        axs = list(axs.values())
        for i in [0,1,2]:
            axs[i].plot(W_ana[:, 0, 0],
                        W_ana[:, 0, 1],
                        "-.",
                        color=colors['Analytical'],
                        label="Analytical")
            axs[i].plot(eu_vals[:, 0, 0],
                        eu_vals[:, 0, 1],
                        "-",
                        color=colors['Euler'],
                        label="Euler")
            axs[i].plot(rk2_vals[:, 0, 0],
                        rk2_vals[:, 0, 1],
                        "--",
                        color=colors['RK2'],
                        label="RK2")
            axs[i].plot(rk4_vals[:, 0, 0],
                        rk4_vals[:, 0, 1],
                        ":",
                        color=colors['RK4'],
                        label="RK4")

            axs[i].set_aspect("equal")

        #fig.suptitle("$\\Var{{t}} = {:.4f}$".format(h))
        axs[0].set_xlabel("$x$")
        axs[0].set_ylabel("$y$")
        axs[0].legend(loc="upper left")

        win_1 = 0.02
        axs[1].set_xlim(0 - win_1, 0 + win_1)
        axs[1].set_ylim(1 - win_1, 1 + win_1)
        #axs[0].indicate_inset_zoom(axs[1], lw=1)
        win_2 = 1e-6
        axs[2].set_xlim(0 - win_2, 0 + win_2)
        axs[2].set_ylim(1 - win_2, 1 + win_2)
        #axs[1].indicate_inset_zoom(axs[2], lw=1)

        ln1 = ConnectionPatch(xyA=(0,1), xyB=(0-win_1,1+win_1),
                              coordsA="data", coordsB="data",
                              axesA=axs[0], axesB=axs[1],
                              color="k", lw=1, alpha=0.5)
        ln2 = ConnectionPatch(xyA=(0,1), xyB=(0-win_1,1-win_1),
                              coordsA="data", coordsB="data",
                              axesA=axs[0], axesB=axs[1],
                              color="k", lw=1, alpha=0.5)
        fig.add_artist(ln1)
        fig.add_artist(ln2)

        ln3 = ConnectionPatch(xyA=(0,1), xyB=(0-win_2,1+win_2),
                              coordsA="data", coordsB="data",
                              axesA=axs[1], axesB=axs[2],
                              color="k", lw=1, alpha=0.5)
```

```
        ln4 = ConnectionPatch(xyA=(0,1), xyB=(0+win_2,1+win_2),
                              coordsA="data", coordsB="data",
                              axesA=axs[1], axesB=axs[2],
                              color="k", lw=1, alpha=0.5)
        fig.add_artist(ln3)
        fig.add_artist(ln4)
        #fig.tight_layout()
        fig.savefig("Figs/orbit_dt.pdf")



# ------------------------------------------------------------
# 3. Summary Plots: CPU time and final energy error (Log-Log)
# ------------------------------------------------------------

# --- Step size vs. CPU Time (Log-Log) ---
fig, ax = plt.subplots()
ax.plot(h_range[:-1], time_euler[:-1], 'o-', color='C0', label="Euler")
ax.plot(h_range[:-1], time_rk2[:-1],   's--', color='C2', label="RK2")
ax.plot(h_range[:-1], time_rk4[:-1],   '^:', color='C3', label="RK4")

ax.set_xscale("log")
ax.set_yscale("log")

ax.set_xlabel("Step size $\\Var{{t}}$")
ax.set_ylabel("CPU Time $t_\\mathrm{CPU}\\axunit{{s}}$")
#ax.minorticks_on()
#ax.grid(True, which="major", linestyle="--", linewidth=0.5, alpha=0.7)
#ax.grid(True, which="minor", linestyle=":", linewidth=0.5, alpha=0.5)
ax.legend(loc="best")
fig.tight_layout()
fig.savefig("Figs/dt_vs_cpu_time_loglog.pdf")

# --- Step size vs. Final Energy Error (Log-Log) ---
fig, ax = plt.subplots()

ax.plot(h_range[:-1], err_euler[:-1], 'o-', color='C0', label="Euler")
ax.plot(h_range[:-1], err_rk2[:-1],   's--', color='C2', label="RK2")
ax.plot(h_range[:-1], err_rk4[:-1],   '^:', color='C3', label="RK4")

ax.set_xscale("log")
ax.set_yscale("log")
# Machine Epsilon line (horizontal)
ax.axhline(eps, color='darkred', ls='-.',
           label='Machine precision $\\epsilon$')

ax.set_xlabel("Step size $\\Var{{t}}$")
ax.set_ylabel("$\\abs{{E_{\\mathrm{analytical}} - E_{\\mathrm{numerical}}}}$")
#ax.minorticks_on()
#ax.grid(True, which="major", linestyle="--", linewidth=0.5, alpha=0.7)
#ax.grid(True, which="minor", linestyle=":", linewidth=0.5, alpha=0.5)

# Ensure 'Machine Epsilon' is in legend
"""
handles, labels = ax.get_legend_handles_labels()
if 'Machine Epsilon' not in labels:
    import matplotlib.lines as mlines
    h_me = mlines.Line2D([], [], color='darkred', ls='--', label='Machine Epsilon')
    handles.append(h_me)
    labels.append('Machine Epsilon')
ax.legend(handles, labels, loc="best", fontsize=12)
"""
ax.legend()
fig.tight_layout()
fig.savefig("Figs/timestep_vs_final_energy_error_loglog1.pdf")
plt.show()
```

file: test_potentials.py

```python
#!/usr/bin/env python
"""
Test: Potential

Draw the Kepler potential and the Henon--Heils potential

@ Author: Moussouni, Yael (MSc student) & Bhat, Junaid Ramzan (MSc student)
@ Institution:  Universite de Strasbourg, CNRS, Observatoire astronomique
                de Strasbourg, UMR 7550, F-67000 Strasbourg, France
@ Date: 2025-01-01

Licence:
Order and Chaos in a 2D potential
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

test_potentials.py
Copyright (C) 2025 Yael Moussouni (yael.moussouni@etu.unistra.fr)
                   Bhat, Junaid Ramzan (junaid-ramzan.bhat@etu.unistra.fr)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
"""
import numpy as np
import matplotlib.pyplot as plt

import potentials as pot
import initial_conditions as init

if "YII_1" in plt.style.available: plt.style.use("YII_1")
# Loading matplotlib style...

W = init.mesh_grid(300)

def kepler(W):
    """Plots the Kepler potential"""
    X = W[0]
    Y = W[1]
    POT = pot.kepler_potential(W, position_only=True)
    fig, ax = plt.subplots(1)
    ax.set_title("Kepler potential")
    pcm = ax.pcolormesh(X, Y, POT)
    fig.colorbar(pcm, label="potential")
    ax.set_aspect("equal")
    ax.set_xlabel("$x$")
    ax.set_ylabel("$y$")

    fig.savefig("Figs/pot_kepler.png")
    return 0

def hh(W):
    """Plots the Henon--Heils potential"""
    X = W[0]
    Y = W[1]
    POT = pot.hh_potential(W, position_only=True)
    fig, ax = plt.subplots(1)
    ax.set_title("Henon--Heils potential")
    pcm = ax.pcolormesh(X, Y, POT)
    fig.colorbar(pcm, label="potential")
```

```
70      ax.set_aspect("equal")
71      ax.set_xlabel("$x$")
72      ax.set_ylabel("$y$")
73
74      fig.savefig("Figs/pot_hh.png")
75      return 0
76
77  if __name__ == "__main__":
78      kepler(W)
79      hh(W)
80
81      plt.show(block=True)
```