University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

PHYC30170 Physics Astronomy and Space Lab I

Comparing Computational Methods for the Orbits in the Solar System

Lian Fernandes
Student No.: 22206311

09 September 2025

Abstract

The aim of this experiment is to compare different computational methods for a more accurate orbit movement.

1. Introduction

Classical mechanics has been an intriguing study in the field of physics with it heavily being involved in the movement of the planets in the solar system. Some of the most fundamental concepts involved in classical mechanics are Newton's Law of Gravitation, centripetal forces of orbits and Kepler's Laws (in particular, Kepler's Third Law). The complexity of the orbits of the solar system makes it hard and tedious to solve differential equations numerically. However, relating these equations with numerical methods allow accuracy in answers and repeated coding.

As science has evolved with technology, solving complex differential equations has been much easier with the help of programming languages such as Python, C++, Java and more. In this paper, Python has been used throughout with libraries such as numpy, scipy and matplotlib aiding with analysis and visualisation. With the help of these services, it is easier to model the orbits of the solar system and predict the behaviour and trajectory of the celestial objects when in such system. This experiment simulated the orbital motion of celestial bodies around the Sun using numerical methods (Euler-Cromer and 2nd order Runge-Kutta) and explored the accuracy of these methods under different conditions.

1.1. Orbital Mechanics

Orbital mechanics are modeled using Newton's gravitational law and laws of motion alongside Kepler's law of motion. [1] Newton's gravitational law states that every object in the Universe is attracted to another object with a mass. [2] The equation that relates this is

$$F = G\frac{m_1 m_2}{r^2} \tag{Eq. 1a}$$

where **F** is the gravitational force between the two objects, $m_1 m_2$ is the masses of the two objects, r is the distance between the objects' centers and G is the gravitational constant (6.647×10^{-11} m^2/kg^2). In relation to our Solar System, the Sun is in the middle of the system with celestial bodies (planets, comets, etc.) revolving around it. Due to this, the equation becomes

$$\mathbf{F} = -\frac{GmM}{|r|^3}\mathbf{r} \tag{Eq. 1b}$$

where M is the mass of the Sun.

Using Newton's second law of motion, $F = ma$, the equation for the acceleration of a celestial body, **a**(t), can be obtained assuming the Sun is stationary at the centre.

$$\mathbf{a}(t) = -\frac{GM}{|\mathbf{r}(t)|^3}\mathbf{r}(t) \tag{Eq. 2}$$

where $r = \sqrt{x^2 + y^2}$. The negative sign indicates that the force is in inwards and directed to the Sun. Newton's third law which states that every action has an equal and opposite reaction, can also be seen here. The gravitational force obtained supplies the centripetal force which keeps the body in orbit. [3] This provides the equation

$$\frac{GMm}{R^2} = \frac{mv^2}{R} \qquad \text{(Eq. 3)}$$

where **R** is the radius of the orbit.

The orbit of most celestial bodies are elliptical. However, some bodies do have circular orbits. All objects that have a circular orbit have a circular velocity, v, of

$$v = \sqrt{\frac{GM}{R}} \qquad \text{(Eq. 4)}$$

to keep the body in a stable orbit.

Kepler's laws are just as fundamental to orbital mechanics as Newton's laws of motion. [4] They are stated as follows:

- *Kepler's First Law:* The planet's orbit around the Sun is an elliptical orbit. Due the Sun's gravitational pull, the orbit of the bodies ends up as an ellipse.
- *Kepler's Second Law:* Often called the law of equal areas. The line that joins the body and the Sun sweeps out equal areas in equal time intervals.
- *Kepler's Third Law:* The squares of the orbital periods of the planets are proportional to the cubes of their semi-major axes.

In this experiment, Kepler's third law is prominent and can be used to calculate the planet's trajectories. The equation that follows this is

$$\left(\frac{P_1}{P_2}\right)^2 = \left(\frac{a_1}{a_2}\right)^3 \qquad \text{(Eq. 5)}$$

where P is the period of the orbit in years and a is the semimajor axis.

Along with the laws of motion, conversation of energy is key in showing that the n-body system maintains its orbits and its energy. Kinetic energy (KE) is the energy that the celestial body holds due to its motion. The equation for the KE is

$$KE = \frac{1}{2}mv^2 \qquad \text{(Eq. 6)}$$

where m is the mass of the body and v is the total velocity in the x and y coordinate. The potential energy (PE) of the body is caused due to the planet's location in the system. [5] The equation is as Eq. 1b.

Adding Eq. 1b and Eq. 6 equates to the total mechanical energy of the system and shows that mass is conserved.

For a binary (two stars) system with masses of M_1 and M_2, the total sum of the acceleration becomes

$$\mathbf{a}(t) = -G\left(\frac{M_1}{(|r - r_1|)^3}(r - r_1) + \frac{M_2}{(|r - r_2|)^3}(r - r_2)\right) \tag{Eq. 7}$$

Energy is still conserved in such system.

1.2. Euler-Cromer Method

The Euler-Cromer method is an modified version of the Euler method that solves ordinary dfferential equations (ODEs) for various oscillatory systems. For an orbital system, this computes the vector position (r) and the velocity, v, of the body as

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \Delta t \mathbf{a}(t) \tag{Eq. 8a}$$

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \Delta t \mathbf{v}(t + \Delta t) \tag{8b}$$

where Δt is the time step, $\mathbf{v}$ is the velocity vector, $\mathbf{r}$ is the position vector and $\mathbf{a}$ is the acceleration vector.

The updated velocity is used to calculate the advance position of the body. Unlike the Euler method, this algorithm preserves the energy of the system for a longer period of time, allowing the body to stay in orbit for longer. [6], [7]

1.3. Runge-Kutta Method

The Runge-Kutta method is a group of numerical method that uses information to extrapolate the solution to the ODE. This includes the famously known 2nd order Runge-Kutta (RK2) and 4th order Runge-Kutta (RK4) methods. This method predicts the value first and then corrects the solution according to the predicted value.[8] RK2 is a midpoint method that computes the current acceleration of the body to calculate the velocity and position. It then computes the acceleration at the midpoint and updates the velocity and position with the midpoint slope.

For a system of $\dot{y} = f(t, y)$, the values are computed as follows:

$$k_1 = f(t_n, y_n) \tag{Eq. 9a}$$

$$y_{mid} = y_n + \frac{\Delta t}{2} k_1 \tag{9b}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, y_{mid}) \tag{9c}$$

$$y_{n+1} = y_n + \Delta k_2 \tag{9d}$$

where k_1 values are the slopes at the start, y_{mid} are the midpoint values, k_2 is the values of the slope at the midpoint and Δt is the time step size.

The accuracy in this method is better than in the Euler-Cromer method due to the $\frac{1}{2}\Delta t$, time step. However, just like the Euler-Cromer method, it is not suitable for long term stability of the orbit. This means in the long run, the body would be out of its orbit (may spiral out). [9]

1.4. Leapfrog Integration Method

The Leapfrog intergration method uses the similar techniques as the velocity verlet method where differential equations is solved by calculating the position and velocity at different time points. The equations are as follows:

$$v(t + \frac{1}{2}\Delta t) = v(t) + \frac{1}{2}a(t)\Delta t \tag{Eq. 10a}$$

$$r(t + \Delta t) = r(t) + v(t + \frac{1}{2}\Delta t)\Delta t \tag{10b}$$

$$v(t + \Delta t) = v(t + \frac{1}{2}\Delta t) + \frac{1}{2}a(t + \Delta t)\Delta t \tag{10c}$$

The positions and velocities are calcuated at alternate intervals than at the same time step. The velocities are calulated at half-integer time steps whereas the positions are calculated at full time steps. [10] This staggers the computation and conserves energy and angular momentum of the object. With it taking multiple steps at staggered intervals, this can be computed for longer period of time, allowing it to be used for longer time periods and intergration.

2. Methodology

The equations for the different intergration methods were defined. Initial conditions (initial velocity and position) were assigned. The simulation uses AU (astronomical units) as the units for the position of the celestial body and years (yr) as the unit for time. This makes the unit of velocity as AU/yr. The gravitational constant, G, becomes $G = 4\pi^2$ and M, the solar mass, is 1 for easier and simple calculations. For circular orbits, the distance between the Sun and the object is fixed at 1 AU.

The Sun is placed at a fixed position at the origin (0,0) of the coordinate system. The first body was initialized at a position (1,0) with a varying velocity to create circular and elliptical orbits.

Eq. 2 was used to obtain the motion of the object. Throughout this simulation, a numerical method was used to update the position and velocity of the object in discrete time step, Δt. The three numerical methods used were Euler-Cromer, 2nd Order Runge-Kutta and Leapfrog method.

The Euler-Cromer method updates the velocity of the body using the acceleration. The updated velocity is then used to calculate the new position of the body. This reccurs for a certain period time until it reaches the required period and time step. This simple method however leads to large errors over a period of time.

The 2nd Order Runge-Kutta method computes the acceleration of the body at the current position and then estimates the velocity and position at the midpoint of the acceleration and velocity. The midpoint of the acceleration is then calculated using the updated velocity and position. The midpoints are then used to calculate the new velocity and position of the body. This reduces the error when compared to the Euler-Cromer method but does not conserve the energy.

The Leapfrog method computes the position and velocity at staggered intervals. The velocity halfway is first calulated as well as the object's position. The position is then accordingly updated. The new acceleration is then updated at the new position. This is then looped until the period is met.

Python was used to simulate the orbits with the NumPy package used for calulations and Matplotlib for the graphical visualisation of the orbits. Each of these values were stored in lists and later refered to to calculate the new velocities and positions at different time steps. To get a quantitative analysis of the orbit, the energy-time graph was created to understand the conservation of energy in each method. This was repeated to a 3 body problem: two stars at 0.4 AU apart and celestial body orbiting in the binary system.

3. Results

3.1. Comparison of Numerical Methods

3.1.1 Circular Orbits

The three numerical methods used to create the orbits of the solar system were Euler-Cromer, 2nd Order Runge-Kutta and Leapfrog method.

The circular orbit was graphed using the Euler-Cromer method in Figure 1.

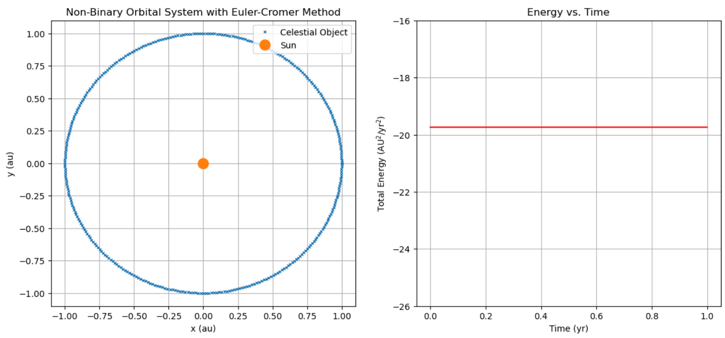

Figure 1: *Circular orbit of the object with $x = 1$ AU and $v_y = 2\pi$ AU/yr using the Euler-Cromer method with the energy-time graph.*

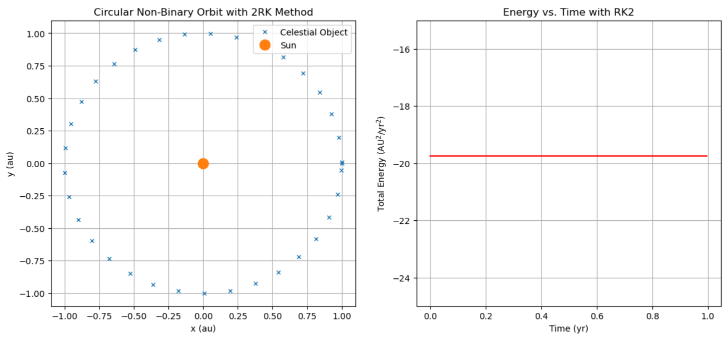

Figure 2: *Circular orbit of the object with $x = 1$ AU and $v_y = 2\pi$ AU/yr using the RK2 method with the energy-time graph.*

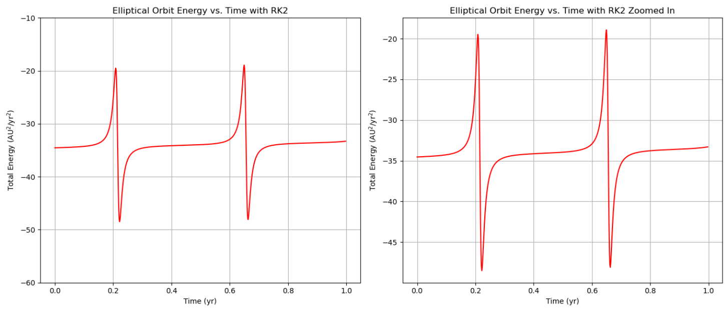

Figure 3: *The energy-time graph of a circular orbit with the initial conditions as in Figure 1 created using the RK2 method.*

3.1.2 Elliptical Orbits

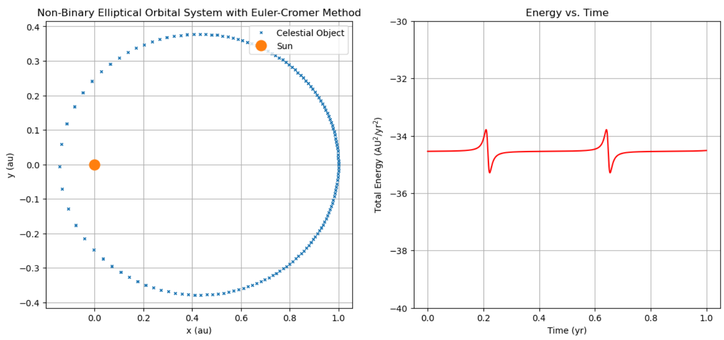

Figure 4: *Elliptical orbit of the object with $x = 1$ AU and $v_y = \pi$ AU/yr using the Euler-Cromer method and the energy-time graph.*

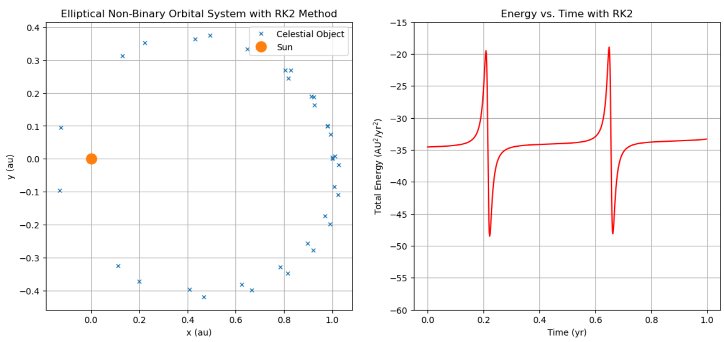

Figure 5: *Elliptical orbit of the object with $x = 1$ AU and $v_y = \pi$ AU/yr using the RK2 method and the energy-time graph.*

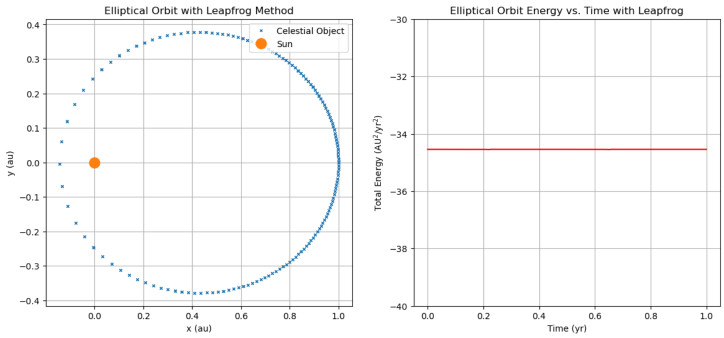

Figure 6: *The energy-time graph of an elliptical orbit with the initial conditions as in Figure 5 created using the RK2 method.*

3.2. Effect of Time Step

3.3. Numerical intergration in a Binary System

4. Discussion

T

5. Conclusion

A

6. References

[1] NASA. ”Chapter 7 Fundamentals of Orbital Mechanics.” NASA. Accessed: 19 Sept. 2025. [Online]. Available: https://spsweb.fltops.jpl.nasa.gov/portaldataops/mpg/MPG_Docs/MPG%20Book/Release/Chapter7-OrbitalMechanics.pdf.

[2] https://phys.libretexts.org/Bookshelves/Conceptual_Physics/Introduction_to_Physics_(Park)/02%3A_Mechanics_I_-_Motion_and_Forces/02%3A_Dynamics/2.09%3A_Newtons_Universal_Law_of_Gravitation

[3] https://www.sciencedirect.com/science/article/pii/B9780124158450000037

[4] Exploring the system lab

[5] https://orbital-mechanics.space/constants-of-orbital-motion/energy-is-conserved-in-orbital-motion.html

[6] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://homepages.dias.ie/ydri/TP3-en.pdf

[7] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://liceocuneo.it/oddenino/wp-content/uploads/sites/2/Alan-Cromer-Stable-solutions-using-the-Euler-Approximation-American-Journal-of-Physics-49-455-1981.pdf

[8] https://www.sciencedirect.com/science/article/pii/B9780128097304000276

[9] https://www.sciencedirect.com/science/article/pii/B9780128097304000276

[10] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.colorado.edu/amath/sites/default/files/attached-files/sttime.pdf

7. Appendix

7.1. Coding

7.1.1 The Euler-Cromer Method for a Non-Binary System

```
# %% [markdown]
# ## The Euler-Cromer Method of Non-Binary Orbital System

# %%
# import the packages
import numpy as np
import matplotlib.pyplot as plt

# %%
# the constants

G = 4*(np.pi**2) # gravitational constant units: AU^3/yr^2
M = 1 # solar mass ... makes equations easier
R = 1 # AU for circular orbits

# %%
# initialising variables / initial conditions

# initial position = x0 and y0 (in AU)
# initial velocity = v_x and v_y (in AU/yr)
# these can be changed accordingly

# initial position
x0 = 1
y0 = 0

# initial velocity
v_x = 0
v_y = 4

# %%
# Using Kepler's Third Law, the equation of a period of a
    circular orbit is

T = np.sqrt(((4*(np.pi**2))/G*M)*R**3)

# the time step - small iteration for the loop to show the
    approximate motion at that time for that x amount of time

dt = 0.00015
step = int(T/dt)    # how many years iteration

# to store the trajectories, use arrays
```

```python
xvalues = [x0]
yvalues = [y0]

# initialise the variables so that it uses after the ones stored in the array
x = x0
y = y0

# to be used when the volecity is getting updated
vx = v_x
vy = v_y

energies = []
times = []

# %%
# Using the Euler-Cromer loop

for i in range(step):
    r = np.sqrt(x**2 + y**2)    # distance from the Sun circular orbit and modulus
    ax = -(G*M*x)/(r**3)    # acceleration in the x direction
    ay = -(G*M*y)/(r**3)    # acceleration in the y direction

    # updating the velocity
    vx = vx + dt*ax
    vy = vy + dt*ay

    # updating the position vector
    x = x + dt*vx
    y = y + dt*vy

    # adds the new trajectories to the end of the list
    # snapshot interval to only add every 20th value
    snap = 20
    if i % snap == 0:
        xvalues.append(x)
        yvalues.append(y)

    # computing the total energy of the orbit
    # KE + PE = total energy of the body in orbit

    KE = 0.5*(1)*np.sqrt((vx**2 + vy**2))**2
    PE = -G*M / r

    E = KE + PE
    energies.append(E)
    times.append(i*dt)

```

```
# %%
plt.figure(figsize=(5.5,14))

# plotting the loop
plt.subplot(3,1,1)
plt.plot(xvalues, yvalues, 'x', markersize = 3, label = "Celestial Object") # orbit of the object
plt.plot(0, 0, "o", markersize = 12, label = "Sun")   # the Sun marker
plt.title('Non-Binary Orbit with Euler-Cromer Method')
plt.xlabel("x (au)")
plt.ylabel("y (au)")
plt.legend(loc='upper right')
plt.grid()

# energy time plot
plt.subplot(3,1,2)
plt.plot(times, energies, color='red')
plt.ylim(-36,-26)
plt.title('Energy vs. Time')
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

# energy time plot zoomed in
plt.subplot(3,1,3)
plt.plot(times, energies, color='red')
plt.title('Energy vs. Time Zoomed In')
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

plt.tight_layout()
```

7.1.2 The 2nd Order Runge-Kutta Method for a Non-Binary System

```
# %% [markdown]
# ## The 2nd Order Runge-Kutta Method of Non-Binary Orbital System

# %%
# import the packages
import numpy as np
import matplotlib.pyplot as plt

# %%
# the constants

```

```
G = 4*(np.pi**2) # gravitational constant units: AU^3/yr^2
M = 1 # solar mass ... makes equations easier
R = 1 # AU for circular orbits

# %%
# initialising variables / initial conditions

# initial position = x0 and y0 (in AU)
# initial velocity = v_x and v_y (in AU/yr)

# initial position
x0 = 1
y0 = 0

# initial velocity
v_x = 0
v_y = 4

# %%
# Using Kepler's Third Law, the equation of a period of a
    circular orbit is

T = np.sqrt(((4*(np.pi**2))/G*M)*R**3)

# the time step - small iteration for the loop to show the
    approximate motion at that time for that x amount of time

dt = 0.0015
step = int(T/dt)    # how many years iteration

# to store the trajectories, use arrays
xvalues = [x0]
yvalues = [y0]

# initialise the variables so that it uses after the ones stored
     in the array
x = x0
y = y0

# to be used when the volecity is getting updated
vx = v_x
vy = v_y

energies = []
times = []

# %%
# for using the 2nd Runge Kutta method

```

```python
# the derivatives
def derivatives(x, y, vx, vy):
    r = np.sqrt(x**2 + y**2) # distance from the Sun circular orbit and modulus
    dxdt = vx
    dydt = vy
    ax = -(G*M*x)/(r**3) # acceleration in the x direction
    ay = -(G*M*y)/(r**3) # acceleration in the y direction
    return dxdt, dydt, ax, ay

# the loop
for i in range(step):
    k1_x, k1_y, k1_vx, k1_vy = derivatives(x, y, vx, vy) # obtaining the slopes of the functions
    r = np.sqrt(x**2 + y**2) # distance from the Sun circular orbit and modulus

    # midpoint calculations
    x_mid = x + (0.5*dt*k1_x)
    y_mid = y + (0.5*dt*k1_y)
    vx_mid = vx + (0.5*dt*k1_vx)
    vy_mid = vy + (0.5*dt*k1_vy)

    # slope at midpoint (RK2)
    k2_x, k2_y, k2_vx, k2_vy = derivatives(x_mid, y_mid, vx_mid, vy_mid) # obtaining the slopes of the midpoints

    # updating the solution
    x = x + (dt*k2_x)
    y = y + (dt*k2_y)
    vx = vx + (dt*k2_vx)
    vy = vy + (dt*k2_vy)

    # snapshot interval
    snap = 20
    if i % snap == 0:
        xvalues.append(x)
        yvalues.append(y)

    # computing the total energy of the orbit
    # KE + PE = total energy of the body in orbit

    KE = 0.5*(1)*(vx**2 + vy**2)
    PE = -G*M/r

    E = KE + PE
    energies.append(E)
    times.append(i*dt)

```

```
# %%
plt.figure(figsize=(5.5,14))

# plotting the graph
plt.subplot(3,1,1)
plt.plot(xvalues, yvalues, 'x', markersize = 5, label = "Celestial Object")
plt.plot(0, 0, "o", markersize = 12, label = "Sun")   # the Sun marker
plt.title('Non-Binary Orbit with 2RK Method')
plt.xlabel("x (au)")
plt.ylabel("y (au)")
plt.legend(loc='upper right')
plt.grid()

# energy time graph
plt.subplot(3,1,2)
plt.plot(times, energies, color='red')
plt.ylim(-40,-20)
plt.title('Orbit Energy vs. Time with RK2')
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

# energy time plot zoomed in
plt.subplot(3,1,3)
plt.plot(times, energies, color='red')
plt.title('Circular Orbit Energy vs. Time with RK2 Zoomed In')
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

plt.tight_layout()
```

7.1.3 The Leapfrog Method for a Non-Binary System

```
# %% [markdown]
# ## The Leapfrog Method on Non-Binary Orbital System

# %%
# import the packages
import numpy as np
import matplotlib.pyplot as plt

# %%
# the constants

```

```python
G = 4*(np.pi**2) # gravitational constant units: AU^3/yr^2
M = 1 # solar mass ... makes equations easier
R = 1 # AU for circular orbits

# %%
# initialising variables / initial conditions

# initial position = x0 and y0 (in AU)
# initial velocity = v_x and v_y (in AU/yr)
# these can be changed accordingly

# initial position
x0 = 1
y0 = 0

# initial velocity
v_x = 0
v_y = np.pi

# %%
# Using Kepler's Third Law, the equation of a period of a
    circular orbit is

T = np.sqrt(((4*(np.pi**2))/G*M)*R**3)

# the time step - small iteration for the loop to show the
    approximate motion at that time for that x amount of time

dt = 0.00015
step = int(T/dt)    # how many years iteration

# to store the trajectories, use arrays
xvalues = [x0]
yvalues = [y0]

# initialise the variables so that it uses after the ones stored
     in the array
x = x0
y = y0

# to be used when the volecity is getting updated
vx = v_x
vy = v_y

energies = []
times = []

# %%
# leapfrog method
```

```
for i in range(step):
    r = np.sqrt(x**2 + y**2)     # distance from the Sun circular orbit and modulus
    ax = -(G*M*x)/(r**3)     # acceleration in the x direction
    ay = -(G*M*y)/(r**3)     # acceleration in the y direction

    # updating the velocity
    vx_half = vx + 0.5*dt*ax
    vy_half = vy + 0.5*dt*ay

    # updating the position
    x = x + vx_half*dt
    y = y + vy_half*dt

    # new acceleration at the new position
    r_update = np.sqrt(x**2 + y**2)
    ax_update = -(G*M*x)/(r_update**3)
    ay_update = -(G*M*y)/(r_update**3)

    # new velocity
    vx = vx_half + 0.5*ax_update*dt
    vy = vy_half + 0.5*ay_update*dt

    # snapshot interval to only add every 20th value
    snap = 20
    if i % snap == 0:
        xvalues.append(x)
        yvalues.append(y)

    # computing the total energy of the orbit
    # KE + PE = total energy of the body in orbit

    KE = 0.5*(1)*(vx**2 + vy**2)
    PE = -G*M / r_update

    E = KE + PE
    energies.append(E)
    times.append(i*dt)

# %%
plt.figure(figsize=(5.5,14))

# plotting the loop
plt.subplot(3,1,1)
plt.plot(xvalues, yvalues, 'x', markersize = 3, label = "Celestial Object") # orbit of the object
plt.plot(0, 0, "o", markersize = 12, label = "Sun")    # the Sun marker
```

```
plt.title('Orbit with Leapfrog Method')
plt.xlabel("x (au)")
plt.ylabel("y (au)")
plt.legend(loc='upper right')
plt.grid()

# energy time plot
plt.subplot(3,1,2)
plt.plot(times, energies, color='red')
plt.title('Circular Orbit Energy vs. Time with Leapfrog')
plt.ylim(-40,-30)
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

# energy time plot zoomed in
plt.subplot(3,1,3)
plt.plot(times, energies, color='red')
plt.title('Circular Orbit Energy vs. Time with Leapfrog')
plt.xlabel("Time (yr)")
plt.ylabel("Total Energy (AU$^2$/yr$^2$)")
plt.grid()

plt.tight_layout()
```