CELESTIAL MECHANICS
Homework 3

Marco Boscato

9 aprile 2024

1. Exercise 1

To solve this problem, we know that the average radius vector is R (from the homework picture we can assume R=1) and has a s-sinusoidal perturbation with a certain amplitude a. Again from the picture we can see that the maximum radius is at $\theta = 0$, so we can write the general formula for the radius as a function of θ as

$$r(\theta) = R + a\cos(k\theta + \phi) \tag{1}$$

where θ is the anomaly, R is the average vector radius, a is the amplitude of the perturbation, k is the pulsation of the sine function, and ϕ an arbitrary phase.
In Binet's space, the variable u is defined as $u = 1/r$, so

$$u(\theta) = [R + a\cos(k\theta + \phi)]^{-1} \tag{2}$$

To solve the problem we need to calculate the second derivative of u with respect to θ, so

$$\frac{du}{d\theta} = \frac{ak\sin(k\theta + \phi)}{(R + a\cos(k\theta + \phi))^2}$$

$$\frac{d^2u}{d\theta^2} = ak^2\cos(k\theta + \phi)(R + a\cos(k\theta + \phi))^{-2} + 2a^2k^2\sin^2(k\theta + \phi)(R + a\cos(k\theta + \phi))^{-3} \tag{3}$$

we can re-write the Eq. 1 as

$$a\cos(k\theta + \phi) = R - r(\theta) \tag{4}$$

and from the fundamental trigonometric relation

$$\sin^2(k\theta + \phi) = 1 - \cos^2(k\theta + \phi) \tag{5}$$

and using Eq. 4 we can re-write Eq. 5 as

$$\begin{aligned} a^2\sin^2(k\theta + \phi) &= a^2 - a^2\cos^2(k\theta + \phi) \\ &= a^2 - (r - R)^2 \end{aligned} \tag{6}$$

So, now we can insert Eq. 6, Eq. 4, Eq. 2 and Eq. 3 in the Binet's Equation, and solving for $f(1/r)$ gives us after several passages

$$f\left(\frac{1}{u}\right) = h^2\left(\frac{1 - k^2}{r^3} + \frac{3k^2R}{r^4} + \frac{2k^2()a^2 - R^2)}{r^5}\right) \tag{7}$$

and to verifying our solution we can solve the Binet's problem with the force calculated previously

$$\begin{aligned} \frac{d^2u}{d\theta^2} + u &= \frac{1}{h^2k^2}f\left(\frac{1}{u}\right) \\ &= -uk^2 + 3k^2Ru^2 + 2k^3u^3(a^2 - R^2) \end{aligned} \tag{8}$$

2. Exercise 2

To calculate the trajectory of the bullet, we can use numerical integration. To simplify the problem, we concentrate all of the Earth's mass at the origin of the reference system $(x, y) = (0, 0)$ and the position of the bullet as $(x, y) = (0, \mathrm{R}_\oplus)$ with the initial velocity decomposed into x, y components by the tilt angle $\theta = \pi/4$.
Then we integrated the system as a *two-body problem* using a **Runge-Kutta 4th Integrator**, available in the Python file `my_RungeKutta4.py`, calculating the acceleration of the system with the usual formula

$$\frac{d\vec{x}}{dt} = -\frac{\mathrm{GM}_\oplus \tilde{\mathrm{x}}}{r^3} \tag{9}$$

where $\mathrm{GM}_\oplus$ is the gravitational parameter of the Earth given in the homework document.

```python
def integrator_rungekutta(particles: Particles,
                          tstep: float,
                          acceleration_estimator: Union[Callable,List]):

    mass = particles.mass

    particles.acc = acceleration_estimator(Particles(particles.pos ,
                                                     particles.vel ,
                                                     particles.mass))

    k1_r = tstep * particles.vel
    k1_v = tstep * particles.acc

    k2_r = tstep * (particles.vel + 0.5 * k1_v)
    k2_v = tstep * acceleration_estimator(Particles(particles.pos + 0.5 * k1_r,
                                                    particles.vel + 0.5 * k1_v,
                                                    mass))

    k3_r = tstep * (particles.vel + 0.5 * k2_v)
    k3_v = tstep * acceleration_estimator(Particles(particles.pos + 0.5 * k2_r,
                                                    particles.vel + 0.5 * k2_v,
                                                    mass))

    k4_r = tstep * (particles.vel + k3_v)
    k4_v = tstep * acceleration_estimator(Particles(particles.pos + k3_r,
                                                    particles.vel + k3_v,
                                                    mass))

    particles.pos = particles.pos + (k1_r + 2 * k2_r + 2 * k3_r + k4_r) / 6
    particles.vel = particles.vel + (k1_v + 2 * k2_v + 2 * k3_v + k4_v) / 6


    # Now return the updated particles, the acceleration, jerk (can be None) and potential (can
    be None)

    return particles
```

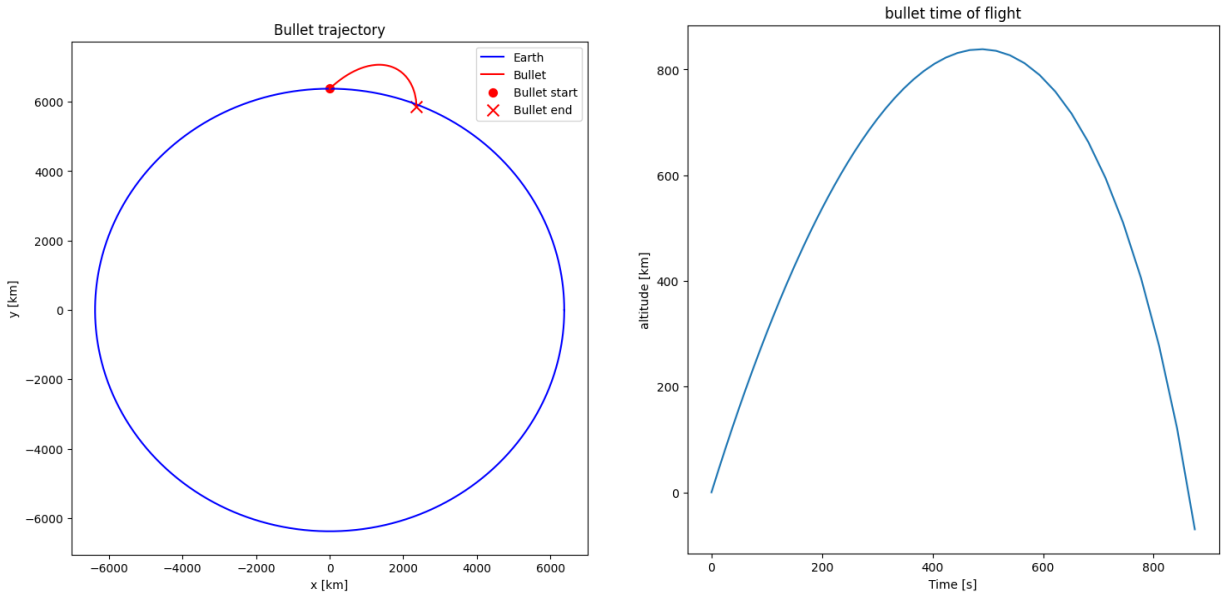
Listing 1: Runge Kutta

To calculate the trajectory of the bullet, we set the condition $r_{bullet} \geq \mathrm{R}_\oplus$, where r_{bullet} is the distance of the bullet from the center of the system (center of the Earth). When r_{bullet} becomes less than $\mathrm{R}_\oplus$, our integrator stops.

```python
def acceleration_two_body(part):
    acc = np.zeros_like(part.pos)
    denom = np.linalg.norm(part.pos[0] - part.pos[1])**3.
    temp = - GM*(part.pos[0] - part.pos[1])/denom
    acc[0,:] = temp
    acc[1,:] = -temp
    return acc

def integration(part, tstep, maxiter):
    path = [part.pos]
    time = [0.]
    velos = [part.vel]
    energy = [part.Etot_vett()[0]]
    iter = 0

    while (part.radius()[1] >= R) and (iter < maxiter):
        update_particules = integrator_rungekutta(particles=part, tstep=tstep,
    acceleration_estimator=acceleration_two_body)

        path.append(update_particules.pos)
        velos.append(update_particules.vel)
```

```
        energy.append(update_particules.Etot_vett()[0])

        h = adaptive_timestep_r(update_particules)[0]
        tstep = h
        time.append(time[-1] + tstep)

        iter += 1

    path = np.array(path)
    velos = np.array(velos)
    energy = np.array(energy)
    time = np.array(time)

    return path, velos, energy, time

bullet_part = part.copy()
bullet_trajectory, bullet_velocity, bullet_energy, bullet_time = integration(bullet_part, tstep=1
    e-4, maxiter=1000)
```

Listing 2: Bullet integration



(a) **(b)**

Figura 1: Left: the trajectory of a bullet fired with a velocity of $v_0 = 5\,\mathrm{Kms}^{-1}$ and an angle of $\theta = \pi/4$. **Right**: bullet's time of flight and altitude from the Earth surface

For this integration we used an adaptive time step based on the distance and velocity of the two bodies, the time of flight of the bullet was calculated by summing each time step at each iteration. The total time obtained was

$$t = 874{,}58\,s$$

```
def adaptive_timestep_r(particles, tmin=None, tmax=None):

    #I use the R/V of the particles to have an estimate of the required timestep
    #I don't want the zeros in this procedure
    r  = particles.radius()
    v  = particles.vel_mod()
    ts = r/v
    eta = 0.1              #proportionality constant
    ts = eta * np.nanmin(ts[np.nonzero(ts)])

    # Check tmin, tmax
    if tmin is not None: ts=np.max(ts,tmin)
    if tmax is not None: ts=np.min(ts,tmax)

    return ts, tmin, tmax
```

Listing 3: adaptive timestep

To calculate the distance traveled on the spherical earth, we calculated the **arctan** of the angular coefficient of the line passing through the origin and the impact point of the projectile, then we found the angle between the vertical line passing through the origin and the firing position, and the other line of the impact point

$$\alpha = \pi/2 - arctan(m)$$

Since the angle is expressed in radians, we can calculate the arc of the circumference bounded by the two lines as

$$l = \mathrm{R}_{\oplus}\alpha \tag{10}$$

giving a result of 2447.30 km

3. Exercise 3

In questo terzo esercizio cerchiamo di ricreare i risulati ottenuti nel paper di **Renato Pakter and Yan Levin**

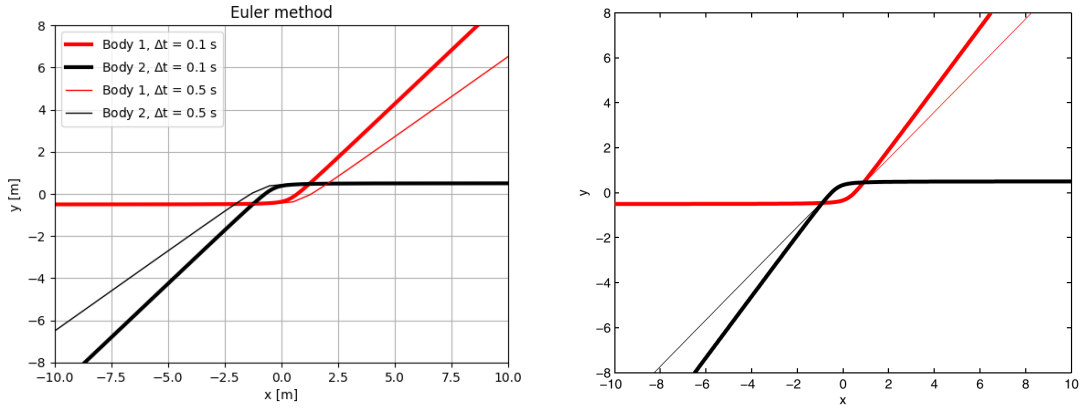
3.1. NUMERICALLY SOLVING A GRAVITATIONAL PROBLEM

The first part consisted of the numerical solution of a two-body problem. The approach used was to solve the problem by Euler's method, developing a code based on the algorithm. The initial conditions of the problem only the same as in the publication, so as to compare the results obtained.

```
mass = 1.5*1e10                                    # kg
G = 6.67*1e-11                                     # m^3 kg^-1 s^-2
pos = np.array([[-10., -0.5], [10., 0.5]])         # m
vel = np.array([[1., 0.], [-1., 0.]])              # m/s
Gm = 1.0                                           # m^3 s^-2

def acceleration(r):
    r1, r2 = r[:2], r[2:]
    r12 = r2 - r1
    r21 = r1 - r2
    a1 = Gm * r12 / np.linalg.norm(r12)**3
    a2 = Gm * r21 / np.linalg.norm(r21)**3
    return np.array([a1, a2]).flatten()

def energy(x, v, m):
    r1, r2 = x[:2], x[2:]
    v1, v2 = v[:2], v[2:]
    r12 = r2 - r1
    return 0.5*m*(np.linalg.norm(v1)**2 + np.linalg.norm(v2)**2) - (G*m**2.)/(np.linalg.norm(r12)
    )

def Euler_method(x, v, t):
    r = x + v*t
    a = acceleration(x)
    v = v + a*t
    return r, v

def Euler_integrator(x, v, max_iter, tol, h):
    t = 0.
    x_list = [x]
    v_list = [v]
    energy_list = [energy(x, v, mass)]
    time_list = [t]

    x_new, v_new = Euler_method(x, v, h)
    x_list.append(x_new)
    v_list.append(v_new)
    energy_list.append(energy(x_new, v_new, mass))
    time_list.append(t + h)

    iter = 0

    while (np.linalg.norm(x_list[-1] - x_list[-2]) > tol and iter<=max_iter):
        x_new, v_new = Euler_method(x_list[-1], v_list[-1], h)
        x_list.append(x_new)
        v_list.append(v_new)
        energy_list.append(energy(x_new, v_new, mass))
        time_list.append(time_list[-1] + h)
        iter += 1
    return np.array(x_list), np.array(v_list), np.array(energy_list), np.array(time_list)
```

```
50
51 integrated_pos, integrated_vel, system_energy, time = Euler_integrator(pos.flatten(), vel.flatten
       (), 1000, 1e-6, 1e-1)
52
53 # with larger time step
54 integrated_pos_2, integrated_vel_2, system_energy_2, time_2 = Euler_integrator(pos.flatten(), vel
       .flatten(), 1000, 1e-6, 5e-1)
```

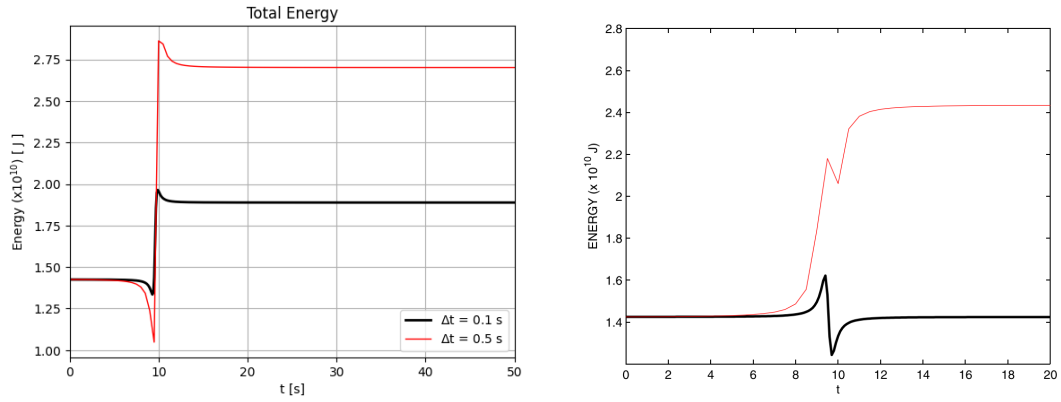
Listing 4: Euler method



(a) **(b)**

Figura 2: Left: Our result **Right**: paper's results

The results obtained are very satisfactory and very similar to those published in the original paper (Fig. 2), with the only difference being the calculation of the total energy. While in the paper (Fig. 3b) at $\Delta t = 0{,}1$ the total energy remained stable even after the encounter of the two bodies, the results obtained by us (Fig. 3a) show how the energy of the system increases after the encounter at $\Delta t = 0{,}1\,s$. In the other hand, similar result has been obtained for $\Delta t = 0{,}5\,s$



(a) **(b)**

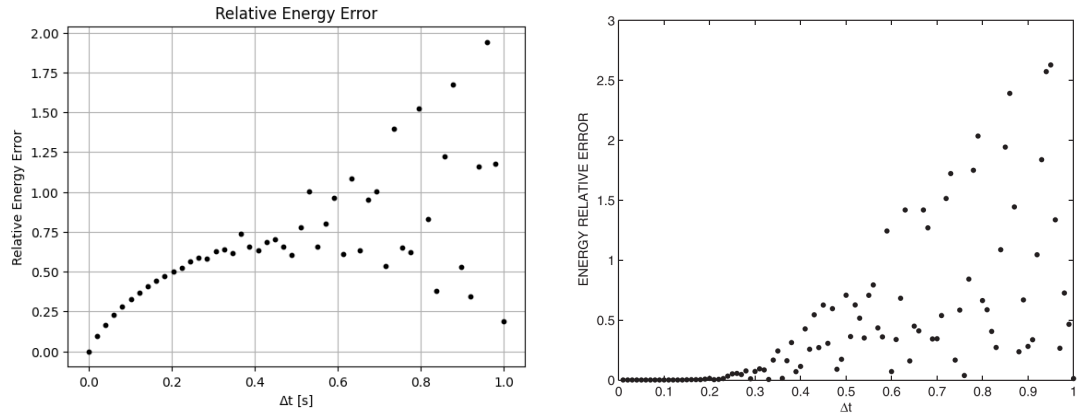
Figura 3: Left: Our result **Right**: paper's results

A similar discussion can be made for the relative energy error of the system: In the paper (Fig. 4b) with time steps below $\Delta t = 0{,}2\,s$ the system seems to be stable, since the relative energy error remains close to 0 and then explodes from $\Delta t > 0{,}2\,s$. In our results (Fig. 4a), the relative energy error increases from the beginning and explodes after $\Delta t > 0{,}2\,s$ in a similar way as in the original paper. This can be explained by the fact that we have probably developed a different Euler integrator than the one used in the paper, perhaps too simple to respect the other one, and therefore less accurate and bringing more errors to each individual integration.

With the same statement, we can conclude that Euler's method for integrating and solving such a problem works quite well for time steps $\Delta t \ll 1$.

3.2. STABILITY OF PLANETARY SYSTEMS

Just as in Sec. 3.1 we used the same initial condition, with the only exception that we used *N-body units* to simplify the computational complexity.



(a) **(b)**

Figura 4: Left: Our result **Right**: paper's results

First, we placed the planets of the star system along the *x-axis* and the initial velocity along the *y-axis*. The initial velocity was calculated so that each planet would follow a circular orbit around the star, given by the formula

$$v = \sqrt{\frac{G(M_\star + m_p)}{r}} \tag{11}$$

where v is the module of the tangential velocity and r is the distance of the planets from the star

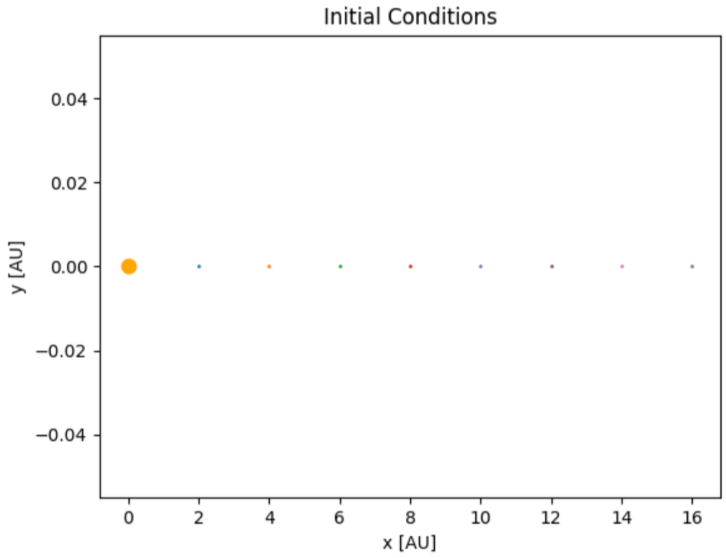
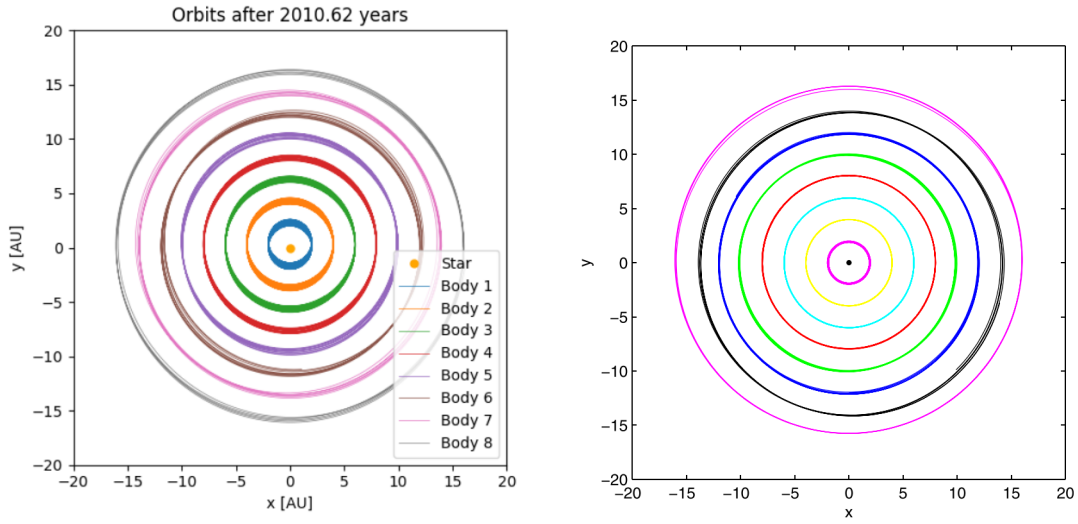


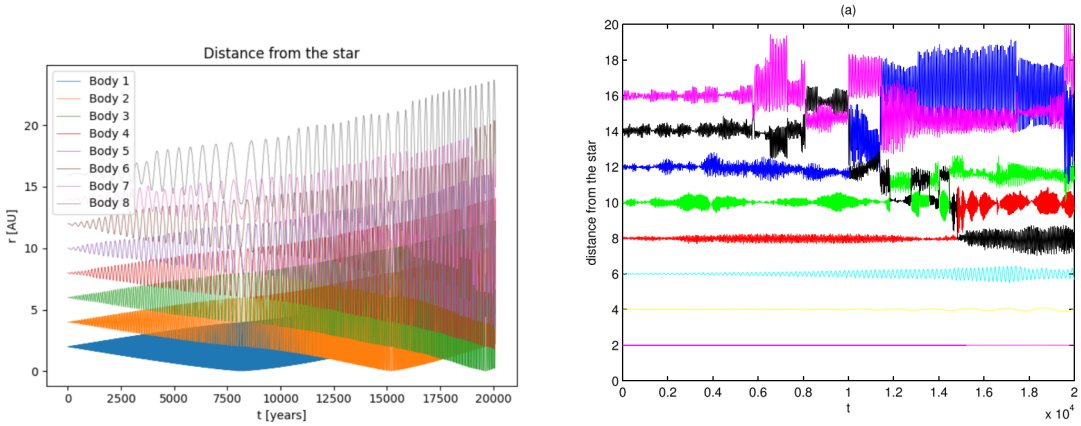
Figura 5: Initial condition of the star system

We have integrated the system with the same **Runge Kutta integrator** (Listing 2) with the adaptive time step used before (Listing 4) for a simulation time equal to 2000 Earth years. From the result (Fig. 6a) we can see how the planets start to move away from their original stable orbit, much more than the integration of the paper (Fig. 6b). This can be caused by the precision of the integrator we developed, which has less precision than the one used by the authors of the paper. Going deeper, if we let our algorithm run for a greater simulation time and plot the distances of each planet from the center of our frame, we can clearly (Fig. 7a) see how in our integration the planets star to shift from their stability almost immediately and keep shifting with a constant amplitude. None of the planets have been capture by another planets orbit as we can see from Fig. 7b.
But since we're using an adaptive time step, the catastrophic events are real and not induced by the numerical errors, similar to the result of the paper (Fig. 8). This explains and establishes that it's almost impossible to randomly find an initial condition that leads to a stable, long-lived planetary system, even with a better and more accurate integrator than the one used in the paper. Note that the total energy is negative, corresponding to a bound state, meaning that the modulus of the attractive gravitational potential energy is greater than the kinetic energy. Nevertheless, planets can escape from the system.



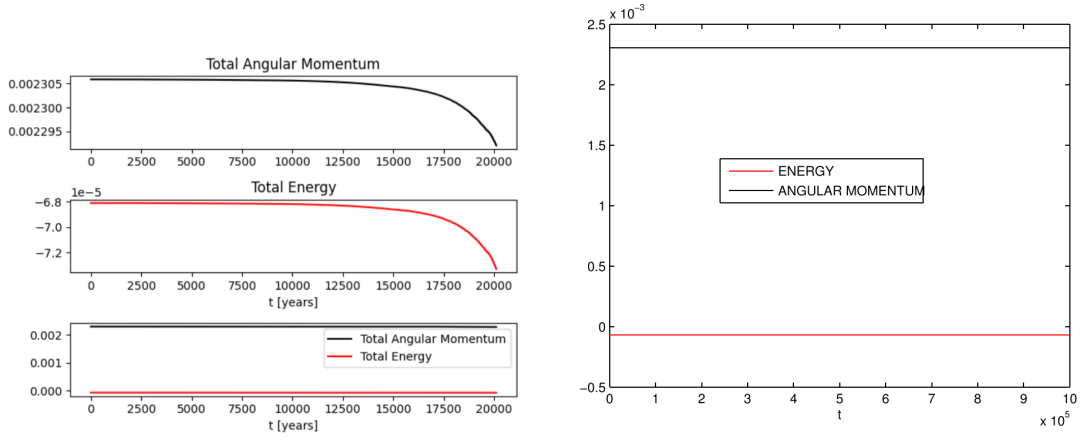
(a) **(b)**

Figura 6: Left: Our result **Right**: paper's results



(a) **(b)**

Figura 7: Left: Our result **Right**: paper's results



(a) **(b)**

Figura 8: Left: Our result **Right**: paper's results