Final Project: Planetary Motion and Asteroids

Aritra Bhakat
aritra@kth.se

January 2022

1. Simulation

1.1. Equations of motion

In this project we will be simulating an n-body system where the bodies interact with each other through the gravitational force. The gravitational force from body j on body i is given by

$$\mathbf{F}_{ij} = \frac{Gm_i m_j(\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3} \tag{1}$$

where m_i and $\mathbf{r}_i$ are the mass and position of the i:th body, and $G = 1.488 \times 10^{-34}\,\mathrm{au}^3\,\mathrm{kg}^{-1}\,\mathrm{d}^{-2}$ is the gravitational constant. The equations of motion for the system are then given by the total force on each body:

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j\neq i}^{n} \frac{\mathbf{F_{ij}}}{m_i} = \sum_{j\neq i}^{n} \frac{Gm_j(\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3} \tag{2}$$

1.2. Conservation of energy

By the law of conservation of energy, the total energy of the n-body system should be conserved. The total energy is the sum of the kinetic and potential energy of the system.

The kinetic energy of the system is given by the sum of the kinetic energy of the individual bodies:

$$E_K = \frac{1}{2}\sum_{i}^{n} m_i v_i^2 \tag{3}$$

The potential energy is given by the sum of gravitational potential between each pair of bodies:

$$E_P = \sum_{i}^{n-1} \sum_{j=i+1}^{n} \frac{G m_i m_j}{|\mathbf{r}_j - \mathbf{r}_i|} \tag{4}$$

A good simulation should conserve the total energy $E = E_K + E_P$.

1.3. Integration method

Using Equation 2 the total acceleration a_i in the i:th time step for each body are calculated. From this the velocity and position of each body is propagated using either Euler or Störmer-Verlet integration scheme.

In Euler integration the position and velocity are propagated as following:

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{a}_i \Delta t \tag{5}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_{i+1} \Delta t \tag{6}$$

The Störmer-Verlet method is as follows:

$$\mathbf{r}_1 = \mathbf{r}_0 + \mathbf{v}_0 \Delta t + \frac{1}{2} \mathbf{a}_i \Delta t^2 \tag{7}$$

$$\mathbf{v}_1 = \frac{\mathbf{r}_1 - \mathbf{r}_0}{\Delta t} \tag{8}$$

$$\mathbf{r}_{i+1} = 2\mathbf{r}_i - \mathbf{r}_{i-1} + \mathbf{a}_i \Delta t^2 \tag{9}$$

$$\mathbf{v}_{i+1} = \frac{\mathbf{r}_{i+1} - \mathbf{r}_i}{\Delta t} \tag{10}$$

Since the integration is not self starting Equation 7 and 8 are used to initialise the method. Of course, $\mathbf{v}$ does not contribute in this case to propagating the equations of motion, but it is useful to calculate to keep track of the kinetic energy. One thing to notice here is that this is the same as the Euler-Cromer method, except for the starting step.

For exact implementation of the n-body system see Appendix A.

2. Problem formulation

In this project we try to answer the question: How are the orbits of the Earth and Moon affected by the nearby passing of an asteroid? We will investigate how the parameters mass, velocity and passing distance of the asteroid affects the orbits.

3. Method

3.1. n-body setup

For an n-body system, $\binom{n}{2} = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ pairwise interactions between bodies have to be calculated, so the amount of bodies needs to be kept to a minimum for efficient calculation. Therefore only the Sun-Earth-Moon system was simulated along with the asteroid. Other planets and objects in the solar system were excluded since they have a marginal effect on the results. For all the simulations, we used initial positions and velocities for the system from the `skyfield` API, set to the time `2022-01-01 12:00:00 TT`. These are given in the Barycentric Celestial Reference System (BCRS). The following mass and radius data was used:

Table 1: Masses and radii of bodies

Body	m [kg]	r [au]
Sun	$1.989 \cdot 10^{30}$	$4.6547 \cdot 10^{-3}$
Earth	$5.972 \cdot 10^{24}$	$4.2635 \cdot 10^{-5}$
Moon	$7.348 \cdot 10^{24}$	$1.1613 \cdot 10^{-5}$

All distances were measured in astronomical units [au], and time in days [d] due to the numerical convenience.

3.2. Orbital periods and accuracy

The accuracy of our simulation and integrators was assessed by measuring the orbital periods of the Earth around the Sun and the Moon around the Earth. In practice, this was done by propagating the Sun-Earth-Moon system until the orbit visually had completed just more than one period. Then the time at which the orbiting body was closest to its starting position relative to the central body was taken as the orbital period. We also used the `skyfield` API to find the maximum error in the positions of the orbits. The energy of the simulated system is also analysed. See Appendix B for implementation.

3.3. Asteroid

To answer the problem formulation, we an asteroid was added to the simulation, with a starting position of $1.01\mathbf{r}_{earth}(t_{start})$ in the BCRS. This places the asteroid just after the Earth, putting the Earth between the Sun and the asteroid.

The initial velocity of the asteroid was set somewhat heuristically, in order to make it pass closer or further to the Earth. First, the initial velocity of the asteroid is pointed towards the location of the Earth a 5 days in the future:

$$\mathbf{v}_{ast}(t_{start}) \propto (\mathbf{r}_{earth}(t_{start} + 5\text{d}) - \mathbf{v}_{ast}(t_{start})) \tag{11}$$

It is then manually scaled to make the asteroid pass at the desired distance from the Earth: far from the Earth, between the Earth and the Moon and close to the Earth. For this Equation 11 was scaled by, respectively, $4.5^{-1}\mathrm{d}^{-1}$, $4.85^{-1}\mathrm{d}^{-1}$ and $4.95^{-1}\mathrm{d}^{-1}$. The reasoning behind this is that we can let the asteroid pass just ahead of the Earth by controlling at what speed it reaches the point where Earth would be after 5 days. However, this is not exact as gravitational forces change the trajectory of the asteroid, which is why the velocity was scaled heuristically until the asteroid passed at the desired distance.

The radius was set to 1.7560×10^{-6} au and mass first to 2.589×10^{20} kg, same as the largest asteroid in the solar system Vesta. The mass was then set to 1×10^{24} kg to study the effect of a planet-like mass passing by. The orbits and energies were then analysed and compared to the unperturbed system without the asteroid.

Since the model uses point masses and does not handle collisions, simulations where the asteroid would have collided with another body are discarded. Additionally, if the asteroid gets too close to another body, the acceleration will become very large requiring a small timestep dt for accuracy. This we want to avoid. The mass of the asteroid was also varied. See Appendix C for implementation.

4. Results

4.1. Orbital periods and accuracy

We obtain orbital periods for the orbits of the Earth around the Sun, and the Moon around the Earth:

Table 2: Orbital periods

	Orbital period [d]	
Orbit	Euler	Verlet
Earth-Sun (dt=0.1)	371.1	365.1
Earth-Sun (dt=0.01)	365.72	365.12
Moon-Earth (dt=0.01)	28.06	27.42
Moon-Earth (dt=0.001)	27.481	27.417

Comparing to the actual orbital periods of the Earth and Moon, 365.2 d and 27.3 d respectively [1], the errors in the Verlet integrator are sufficiently small, irrespective of the length of the timesteps used. For the Euler integrator the orbital periods get closer to the literature value when the timestep is decreased, but is still not as close as the Verlet integrator.

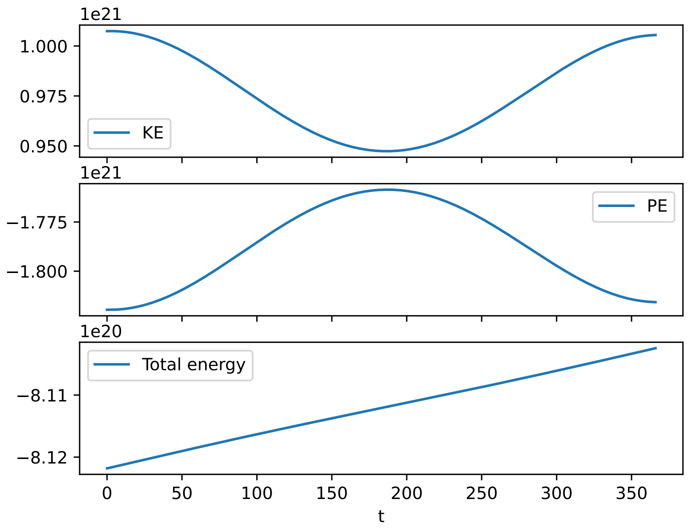
Now we compare the simulation data to the position data from `skyfield`, to obtain the maximum error in position for the Earth and the Moon:

Table 3: Max error in position of orbits

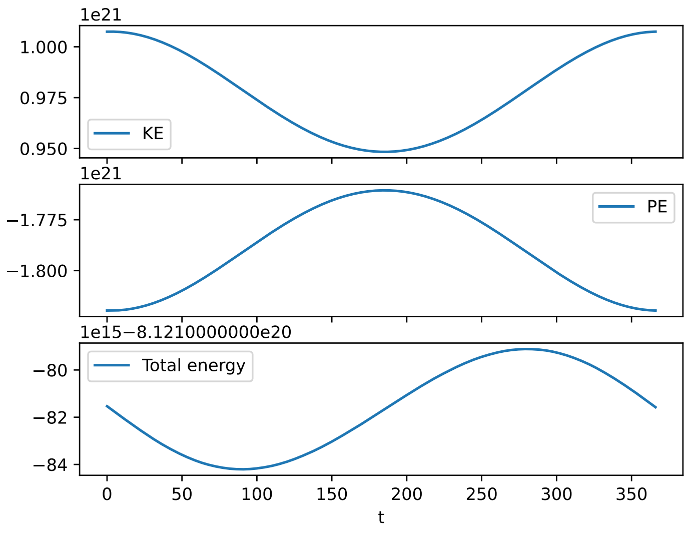
	Orbit max error [au]	
Body	Euler	Verlet
Earth (dt=0.1)	$1.093 \cdot 10^{-1}$	$3.100 \cdot 10^{-3}$
Earth (dt=0.01)	$8.502 \cdot 10^{-3}$	$3.105 \cdot 10^{-3}$
Moon (dt=0.01)	$4.717 \cdot 10^{-4}$	$1.496 \cdot 10^{-5}$
Moon (dt=0.001)	$5.983 \cdot 10^{-5}$	$1.495 \cdot 10^{-5}$

The Euler integrator clearly performs worse than the Verlet integrator for longer timesteps, and for shorter timesteps has a similar max error. In the Verlet integrator, the error for the Earth's orbit is in the order of magnitude of 10^{-3}au, which is the the same order of magnitude as the Sun's radius, and 3 orders of magnitude smaller than the distance from the Earth to the Sun. Simlarly, the error for the Moon's orbit is in the order of 10^{-3}au, which is the the same order of magnitude as the Moon's radius, and 2 orders of magnitude smaller than the distance from the Moon to the Earth. Of course, the reason that the error in the Moon's orbit is smaller is that it has only been propagated $\sim$ 28 days, while the Earth's orbit has been propagated for $\sim$ 366 days.

We can also look at energy conservation:



(a) Euler integrator



(b) Verlet integrator

Figure 1: Energy conservation of the integrators

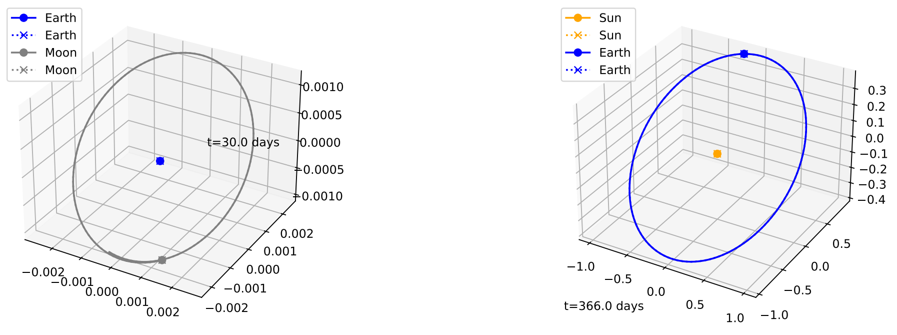
The Euler integrator has a constantly increasing energy, while the Verlet integrator has a periodically constant energy which fluctuates with a smaller amplitude than the increase of the energy in the Euler integrator.

Clearly this indicates that the Euler integrator is not good for this purpose. Going forward, we will use the Verlet integrator, with the timestep dt=0.01. Additionally, visually we see the expected behaviour of the Earth orbiting the Sun and the Moon orbiting the Earth in the following animations from the Verlet integrator: Earth around Sun, Moon around Earth.

5. Asteroid

5.1. Asteroid-like mass

First we use an asteroid with asteroid-like mass 2.589×10^{20} kg. First we let the asteroid pass relatively far from the Earth, outside the Moon (animation), then between the Moon and Earth (animation) and lastly very close to the Earth (animation). In the following figure the orbits of the system with the asteroid passing close to Earth are compared to the system without the asteroid.



(a) Moon orbit

(b) Earth orbit

Figure 2: Orbits of system with asteroid with asteroid-like mass passing close to the Earth compared to system without asteroid (dotted)

All the other systems with the asteroid passing further from Earth were similarly unperturbed (which is why they haven't been plotted). Due to the small mass of the asteroid there is negligible deviation compared to the Sun-Earth-Moon system without the asteroid. The position of the Earth relative to the Sun only differs by $\sim 10^{-5}$au after a year, which is the same magnitude as Earths radius, and the Moon position relative to the Earth only differs by $\sim 10^{-6}$au after 30 days.

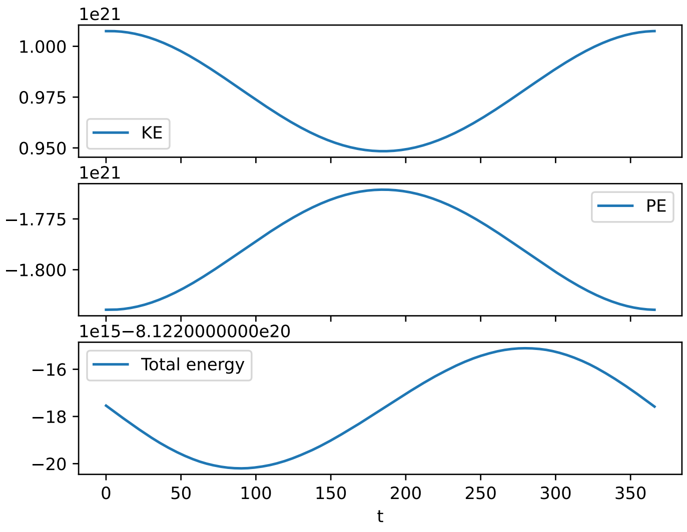


Figure 3: Energy for system with asteroid with asteroid-like mass passing close to the Earth

The energies are similar to Figure 1b, only differing due to the additional energy introduced by the asteroid.

5.2. Planet-like mass

Here we use an asteroid with a planet-like mass, 1×10^{24} kg. Asteroid passing far from the Earth, outside the Moon (animation):

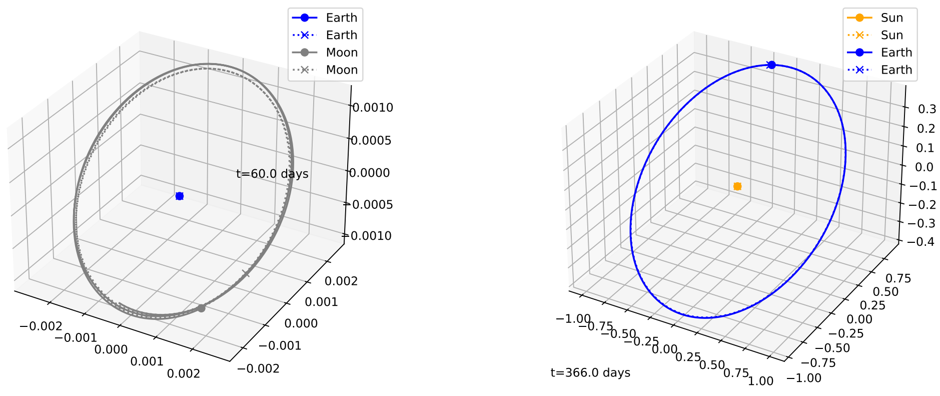


Figure 4: Orbits of system with asteroid with planet-like mass passing far from Earth compared to unperturbed system (dotted)

The position of the Earth relative to the Sun differs by 1.701×10^{-2} au after 366 days. The Moon position relative to the Earth differs by 6.051×10^{-4} au after 30 days. Visually in Figure 4 the orbit of the Moon lags behind the unperturbed orbit. So does the Earth's orbit, but only very slightly as to nearly be invisible.

In Figure 5 we see that the energy of the system is slightly perturbed by the asteroid passing by.

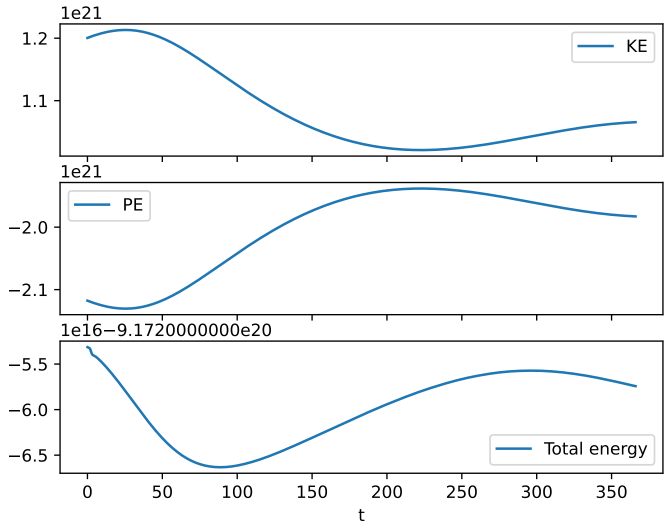
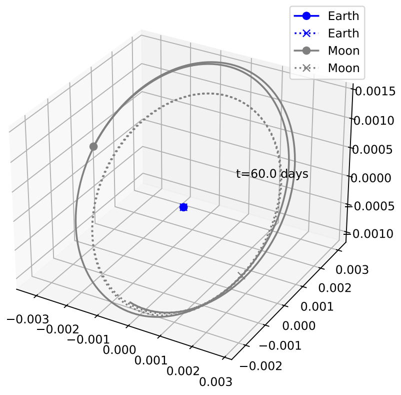
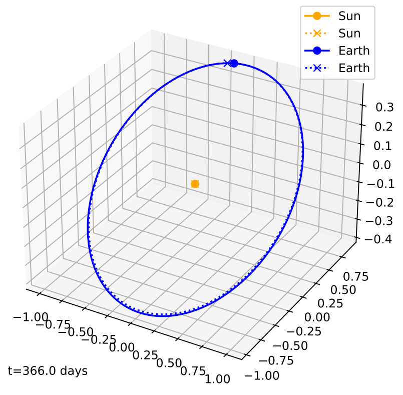


Figure 5: Energy for system with asteroid with planet-like mass passing far from the Earth

Asteroid passing between the Moon and Earth (animation):



(a) Moon orbit



(b) Earth orbit

Figure 6: Orbits of system with asteroid with asteroid-like mass passing between Earth and Moon compared unperturbed system (dotted)

The position of the Earth relative to the Sun differs by 6.771×10^{-2} au after 366 days. The Moon position relative to the Earth differs by 1.005×10^{-2} au after 30 days. In Figure 6 the Moon's orbit radius is increased, while the Earth's orbit slightly lags behind the unperturbed orbit.

In Figure 7 we see that the energy of the system is perturbed slightly more when the asteroid passes by even closer.

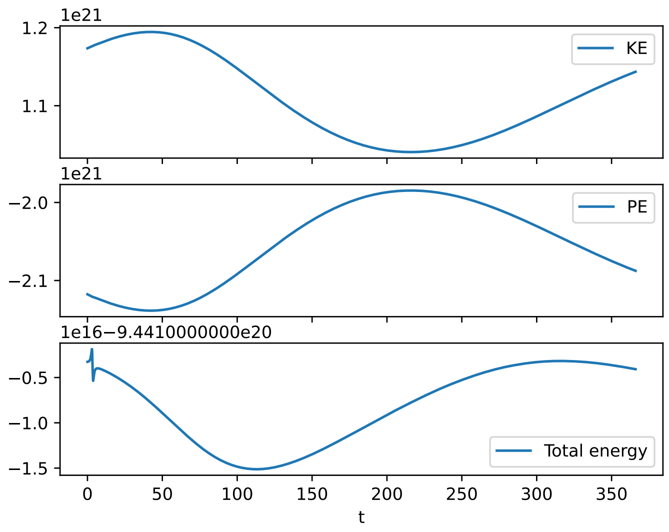
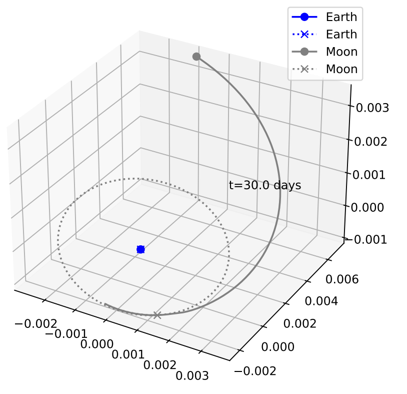
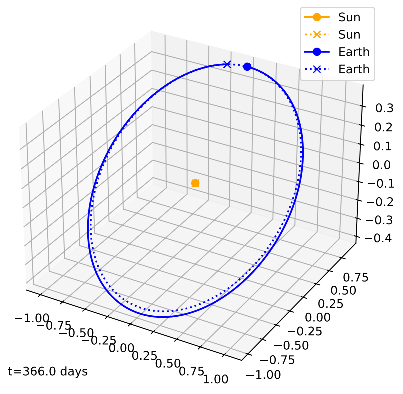


Figure 7: Energy for system with asteroid with planet-like mass passing between the Earth and Moon

Asteroid passing very close to Earth (animation):



(a) Moon orbit



(b) Earth orbit

Figure 8: Orbits of system with asteroid with planet-like mass passing very close to Earth compared to unperturbed system (dotted)

The position of the Earth relative to the Sun differs by 2.115×10^{-1} au after 366 days. Visually in Figure 8b we see that the eccentricity of the orbit is increased. The Moon position relative to the Earth differs by 4.053×10^{-3} au after 30 days. In Figure 8a we see that the Moon is flung out from its orbit.

In Figure 9 when the asteroid passes very close by, the perturbation of the energy is large compared to the amplitude of its periodic fluctuation.

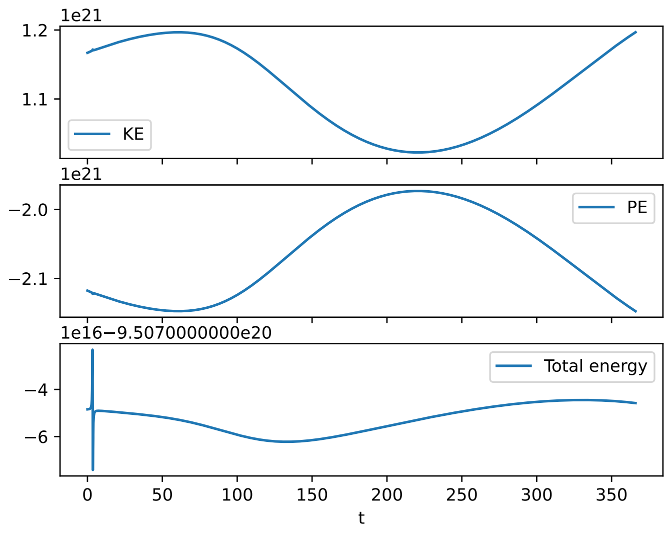


Figure 9: Energy for system with asteroid with planet-like mass passing close to the Earth

6. Discussion

The results of the simulations are quite intuitive in general. For the asteroid-like mass which is 2 orders of magnitude lighter than the Moon, and 4 orders lighter than the Earth, the effect on the orbits are miniscule. For the planet-like mass the effects are stronger the closer the asteroid passes to Earth. The orbital radius increases as the asteroid passes closer by, and the orbits lag behind the unperturbed orbits even more. The difference in position of the orbital bodies relative to their central bodies increases with closer passing distance.

Of course, the energy also perturbed more with a closer passing distance, which would indicate that the simulation is less accurate. However the sizes of these perturbations are the same magnitude or smaller than the periodic fluctuation, and combined with the intuitive soundness of the results it would indicate that this isn't problematic. To improve this a smaller timestep would have helped.

One thing to note however, which can be seen in the animations, is that that for our starting conditions the asteroid always happens to pass by close to the Earth as the

Moon is on the opposite side of the Earth compared to the asteroid. If the Moon were to be on the same side of the Earth as the asteroid we would expect some different results. When the asteroid passes far from Earth, or between the Earth and Moon, it would be expected to have a bigger effect on the Moon's orbit than the asteroid passing close to the Earth.

References

[1] David R. Williams. *Planetary Fact Sheet - Metric*. Dec. 2021. URL: `https://nssdc.gsfc.nasa.gov/`.

Appendices

Appendix A N-body code

Listing 1: nbody.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

G = 1.488e-34  # Gravitational constant G in days and AU
nameindex = 0


class Body:
    """
    Contains properties of gravitational body, and methods
       to integrate its
    position and velocity
    """

    def __init__(self, M, R, r0, v0, name='', color='g'):
        global nameindex

        self.M = M
        self.R = R
        self.r0 = r0
        self.r = np.copy(r0)
        self.rlist = np.array([r0])
        self.v0 = v0
        self.v = np.copy(v0)
        self.vlist = np.array([v0])
        self.F = 0
        self.KE = 0.5 * self.M * np.dot(self.v0, self.v0)

        if name == "":
            self.name = "Planet" + str(nameindex)
            nameindex += 1
        else:
            self.name = name

        self.color = color
    def add_force(self, F):
        """
        Add force F to body
        """
        self.F += F


class Integrator:
    """
    Parent class for integrators
    """

    def __init__(self, dt):
        self.dt = dt

    def integrate(self, body):
        """
        Integrate one time step, updating body position and
           velocity given the
        total force on it
        """
        pass


class EulerIntegrator(Integrator):
    """
    Applies the Euler integration scheme
    """

    def integrate(self, body):
        body.r += body.v * self.dt
        body.v += body.F / body.M * self.dt

        body.rlist = np.append(body.rlist, [body.r], axis=0)
        body.vlist = np.append(body.vlist, [body.v], axis=0)

        body.KE = 0.5 * body.M * np.dot(body.v, body.v)
        body.F = 0



class VerletIntegrator(Integrator):
    """
    Applies the basic Verlet integration scheme
    """

    def integrate(self, body):
        if len(body.rlist) == 1:
            body.r += body.v0 * self.dt + 0.5 * body.F /
               body.M * self.dt**2
            body.v = (body.r - body.r0) / self.dt
        else:
            body.r = 2 * body.rlist[-1] - body.rlist[-2] +\
                     body.F / body.M * self.dt**2
            body.v = (body.r - body.rlist[-1]) / self.dt

        body.rlist = np.append(body.rlist, [body.r], axis=0)
        body.vlist = np.append(body.vlist, [body.v], axis=0)

        body.KE = 0.5 * body.M * np.dot(body.v, body.v)
        body.F = 0


class System:
    """
    System containing n bodies
    """

    def __init__(self, bodies, dims, integrator):
        self.bodies = bodies
        self.n = len(bodies)
        self.dims = dims
        self.t = 0
        self.ts = np.array([0])
        self.integrator = integrator
        self.collisions = []
        self.check_collisions()

        self.KE = 0
        for i in range(self.n):
            self.KE += self.bodies[i].KE
        self.KElist = np.array([self.KE])
        self.PE = 0
        for i in range(self.n):
            for j in range(i + 1, self.n):
                self.PE += System.potential(self.bodies[i],
                   self.bodies[j])
        self.PElist = np.array([self.PE])
        self.E = self.KE + self.PE
        self.Elist = np.array([self.E])

    def update(self, iters=1):
        """Updates body positions and velocities, and energy
           of the system"""
        for i in range(iters):
            self.t += self.integrator.dt
            self.ts = np.append(self.ts, self.t)
            # Calculate interactions between all bodies
            self.KE = 0
            self.PE = 0
            for i in range(self.n):
                for j in range(i + 1, self.n):
                    F = System.force(self.bodies[i], self.
                       bodies[j])
                    self.PE += System.potential(self.bodies[
                       i],
                                                self.bodies[
                                                   j])
                    self.bodies[i].add_force(F)
                    self.bodies[j].add_force(-F)

                self.integrator.integrate(self.bodies[i])
                self.KE += self.bodies[i].KE

            self.E = self.KE + self.PE
            self.KElist = np.append(self.KElist, self.KE)
            self.PElist = np.append(self.PElist, self.PE)
            self.Elist = np.append(self.Elist, self.E)
            self.check_collisions()

    def check_collisions(self):
        """Checks for collisions (intersection) between
          bodies"""
        for i in range(self.n):
            for j in range(i + 1, self.n):
                if (np.linalg.norm(self.bodies[i].r - self.
                   bodies[j].r)
                        < self.bodies[i].R + self.bodies[j].
                           R):
                    self.collisions.append(Collision(self.t,
                                                     self.
                                                        bodies
```

```
                                                                      [i],
                                                                   self.
                                                                      bodies
                                                                      [j])
                                                                      )

    def potential(b1, b2):
        """Returns pairwise potential between bodies b1 and
           b2"""
        return -G*b1.M*b2.M / np.linalg.norm(b1.r - b2.r)

    def force(b1, b2):
        """Returns force on body b1 from body b2"""
        return -G*b1.M*b2.M / np.linalg.norm(b1.r - b2.r)**3
            * (b1.r - b2.r)

    def orbital_period(self, c_body, o_body):
        """
        Returns orbital period of central body c_body around
            orbiting body
        o_body. Only works if the system has been propagated
            just longer than
        one orbital period.
        """
        rel_start_pos = o_body.r0 - c_body.r0

        min_dist = np.inf
        min_step = 0
        for i in range(len(self.ts)//2, len(self.ts)):
            rel_pos = o_body.rlist[i] - c_body.rlist[i]
            dist = np.linalg.norm(rel_pos - rel_start_pos)
            if dist < min_dist:
                min_dist = dist
                min_step = i

        return self.ts[min_step]

class Collision:
    """Contains information on collisions between bodies"""

    def __init__(self, t, b1, b2, printmessage=False):
        self.t = t
        self.name1 = b1.name
        self.name2 = b2.name
        self.r1 = b1.rlist[-1]
        self.r2 = b2.rlist[-1]
        self.v1 = b1.vlist[-1]
        self.v2 = b2.vlist[-1]
        self.message = self.name1 + " collided with " + \
            self.name2 + " at t=" + str(self.t)
        print(self)

    def __str__(self):
        return self.message
```

Appendix B Orbit code

Listing 2: orbit.py

```
import datetime
from nbody import *
from skyfield.api import load

def max_error(system, body, sfname):
    max_error = 0
    for i in range(nsteps + 1):
        t = tinit + system.ts[i]
        error = np.linalg.norm(planets[sfname].at(t).
           position.au
                                - body.rlist[i])
        if error > max_error:
            max_error = error

    return max_error

ts = load.timescale()
tinit = ts.tt(2022, 1, 1, 12, 0)

planets = load('de421.bsp')
sfsun, sfearth, sfmoon = planets['sun'], planets['earth'],
   planets['moon']

sun = Body(M=1.989e30, R=4.6547e-3, r0=sfsun.at(tinit).
   position.au,
           v0=sfsun.at(tinit).velocity.au_per_d, name="Sun",
               color='orange')
```

```
earth = Body(M=5.972e24, R=4.2635e-5, r0=sfearth.at(tinit).
   position.au,
             v0=sfearth.at(tinit).velocity.au_per_d, name="
                Earth", color='blue')
moon = Body(M=7.348e22, R=1.1613e-5, r0=sfmoon.at(tinit).
   position.au,
            v0=sfmoon.at(tinit).velocity.au_per_d, name="
               Moon", color='grey')

dims = 3
# TODO: Set dt
solar = System([sun, earth, moon], dims, VerletIntegrator(dt
   ))

# Set to tmax >366 for Earth around Sun, >28 for Moon around
    Earth
nsteps = int(tmax / dt)

solar.update(nsteps)
# TODO: Set central_body and orbiting_body
print(solar.orbital_period(central_body, orbiting_body))
# TODO: Set body, and sf_body_name, which is name of body in
    skyfield
print(max_error(solar, body, sf_body_name))
```

Appendix C Asteroid code

Listing 3: asteroid.py

```
import datetime
from nbody import *
from skyfield.api import load

ts = load.timescale()
tinit = ts.from_datetime(datetime.datetime(
    2022, 1, 1, tzinfo=datetime.timezone.utc))

planets = load('de421.bsp')
sfsun, sfearth, sfmoon = planets['sun'], planets['earth'],
   planets['moon']
```

```
sun = Body(M=1.989e30, R=4.6547e-3, r0=sfsun.at(tinit).
   position.au,
           v0=sfsun.at(tinit).velocity.au_per_d, name="Sun",
               color='orange')
earth = Body(M=5.972e24, R=4.2635e-5, r0=sfearth.at(tinit).
   position.au,
             v0=sfearth.at(tinit).velocity.au_per_d, name="
                Earth", color='blue')
moon = Body(M=7.348e19, R=1.1613e-5, r0=sfmoon.at(tinit).
   position.au,
            v0=sfmoon.at(tinit).velocity.au_per_d, name="
               Moon", color='grey')

astr0 = 1.001 * earth.r0
# TODO: Set astv0
# TODO: Set astm
ast = Body(M=astm, R=1.7560e-6, r0=astr0, v0=astv0, name="
   Asteroid", color='black')


dims = 3
# TODO: Set dt
astsys = System([sun, earth, moon, ast], dims,
   VerletIntegrator(dt))

# TODO: Set tmax
nsteps = int(tmax / dt)
astsys.update(nsteps)
```