Exercises: Week 1
by Julian Mayr and Jona Ackerschott

Exercise 3

The computer program we wrote in Python:

```
import numpy as np
a=5.0
n0=100.0
n1=30.0
y0=100

def get_y_n(y_last, n1, n, a):
    if n<n1:
        y_n=1/(n+1)-a*y_last
        return get_y_n(y_n, n1, n+1, a)
    elif n>n1:
        y_n=(1/n-y_last)/a
        return get_y_n(y_n, n1, n-1, a)
    elif n==n1:
        return y_last

print(get_y_n(y0, n1, n0, a))
```

The iteration only converges for $n1 > n0$. It then converges to:

```
n1=30, n2=50, a=5: y30=0.0054
```

For $n0 < n1$ we get divergence:

```
n1=0, n2=50, a=5: y30=9.27*10^22
```

Exercise 4

The numerical simulation of the 2-Body-Problem simulation was set up here, and some tests on parameters were made:

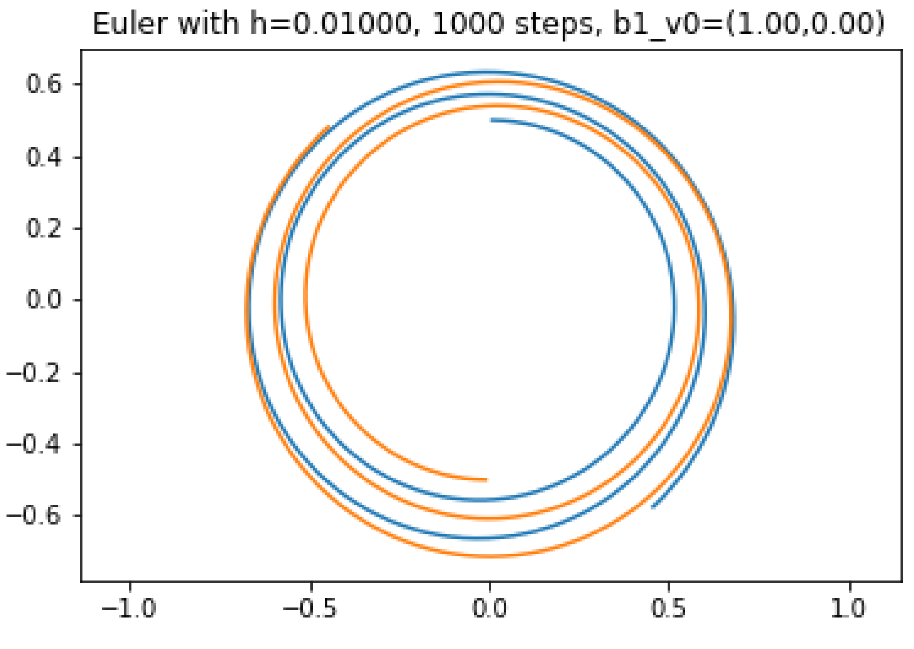
```
import nupmy as np
import matplotlib.pyplot as plt

G=1
M=1
h=0.01
steps=1000
class Body:
    def __init__(self, pos, vel, mass):
        self.pos=pos
        self.vel=vel
        self.startpos=pos
        self.startvel=vel
        self.mass=mass
        self.trail=[]
    def euler_step(self, h, other):
        vel=self.vel+h*G*(self.mass+other.mass)/np.linalg.norm(self
        pos=self.pos+h*self.vel##calculate xi+1(xi,vi)
        self.vel=vel
        self.pos=pos
        self.trail.append(self.pos)
    def leapfrog_position_step(self, h, other):
        self.pos=self.pos+h*self.vel##calculate xi+1(x(i),v(i+1/2)
        self.trail.append(self.pos)
    def leapfrog_velocity_step(self,h,other):
        self.vel=self.vel+h*G*(self.mass+other.mass)/np.linalg.norm
    def runge_lenz(self):
        """returns the runge lenz vector in case of an elliptical m
        return np.cross(self.mass*self.vel, np.array(np.cross(self.
    def eccentricity(self):
        """returns the eccentricity vector in case of an elliptical
        return self.runge_lenz()/(self.mass*G)
    def energy(self, other):
        """returns V(x1,x2)+1/2m*v**2 for 2 body problem"""
        return -G*self.mass*other.mass/np.linalg.norm(self.pos-othe
    def reset(self):
        self.trail=[]
        self.vel=self.startvel
        self.pos=self.startpos
##euler
def euler(b1, b2, steps, h, plot_graph=True):
```

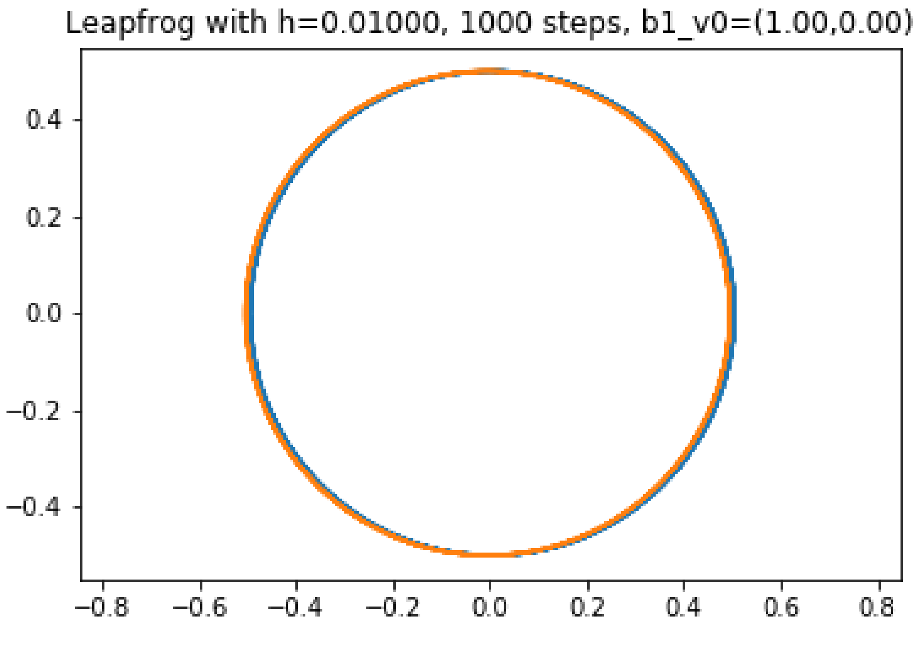
```
    i=0
    while i<steps:
        b1.euler_step(h, b2)
        b2.euler_step(h, b1)
        i+=1
    if plot_graph:
        plt.plot(np.array(b1.trail)[:,0], np.array(b1.trail)[:,1], label=
        plt.plot(np.array(b2.trail)[:,0], np.array(b2.trail)[:,1], label=
        plt.axis("equal")
        plt.title("Euler with h=%.5f, %i steps, b1_v0=(%.2f,%.2f)"%(h, st
        plt.savefig("Euler with h=%.5f, %i steps, b1_v0=(%.2f,%.2f).png"%
        plt.legend()
        plt.show()

def leapfrog(b1, b2, steps, h, plot_graph=True):
    i=0
    while i<steps:
            b1.leapfrog_position_step(h, b2)
            b2.leapfrog_position_step(h, b1)
            b1.leapfrog_velocity_step(h, b2)
            b2.leapfrog_velocity_step(h, b1)
            i+=1
    if plot_graph:
        plt.axis("equal")
        plt.plot(np.array(b1.trail)[:,0], np.array(b1.trail)[:,1], label=
        plt.plot(np.array(b2.trail)[:,0], np.array(b2.trail)[:,1], label=
        plt.title("Leapfrog with h=%.5f, %i steps, b1_v0=(%.2f,%.2f)"%(h,
        plt.savefig("Leapfrog with h=%.5f, %i steps, b1_v0=(%.2f,%.2f).pn
        plt.legend()
        plt.show()
```

The two bodies rotate in a circular fashion for $v0 = 1$ for both bodies:

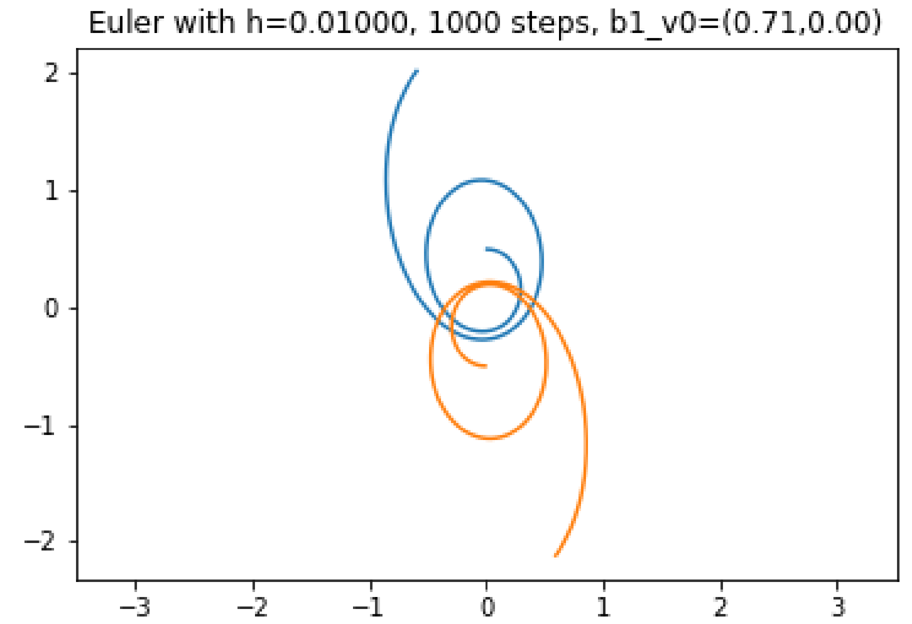
```
b1=Body(np.array([0,0.5]),np.array([1,0]),M)
b2=Body(np.array([0,-0.5]),np.array([-1,0]),M)
##for this configuration (v=1) the bodys rotate in circular fashion, alth
##so leapfrog integration is already done here
euler(b1,b2,steps,h)
b1.reset()
b2.reset()
leapfrog(b1,b2,steps,h)
```

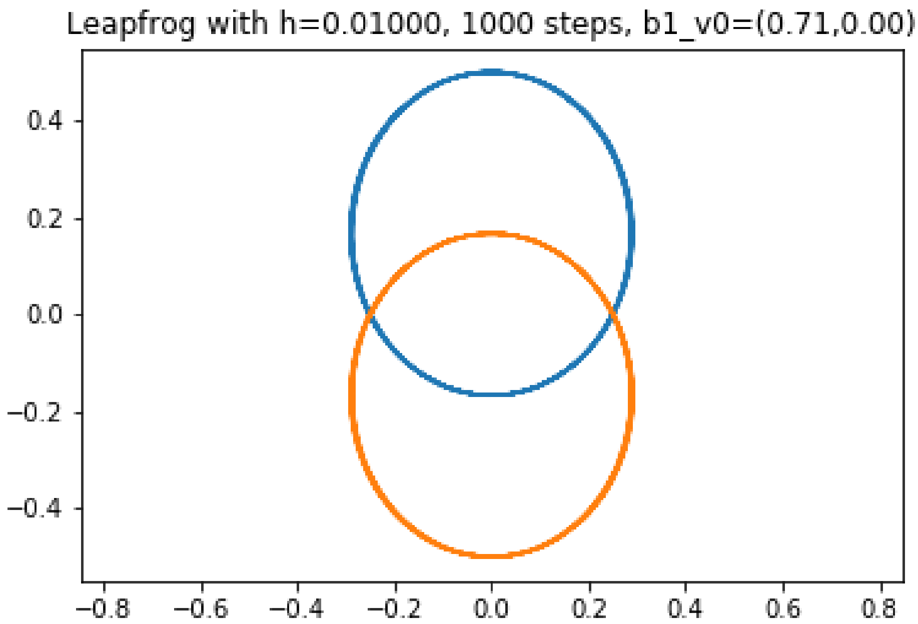
Euler Integration is very inaccurate for the given large timestep, so leapfrog integration was used as an alternative method already to better showcase the effects





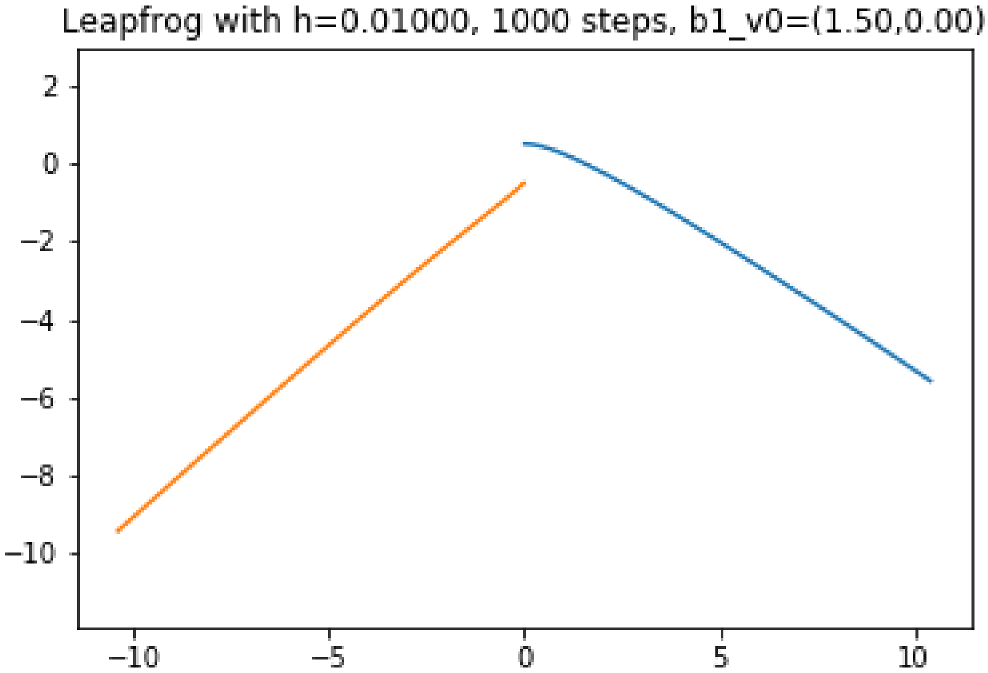
For $v_0 = 1/\sqrt{2}$ we get (coded same as above)



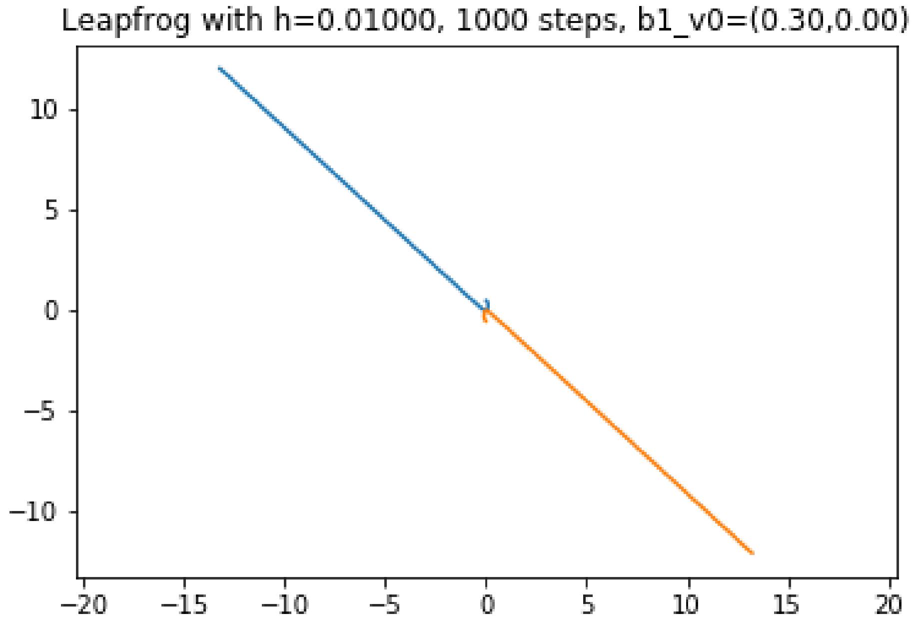


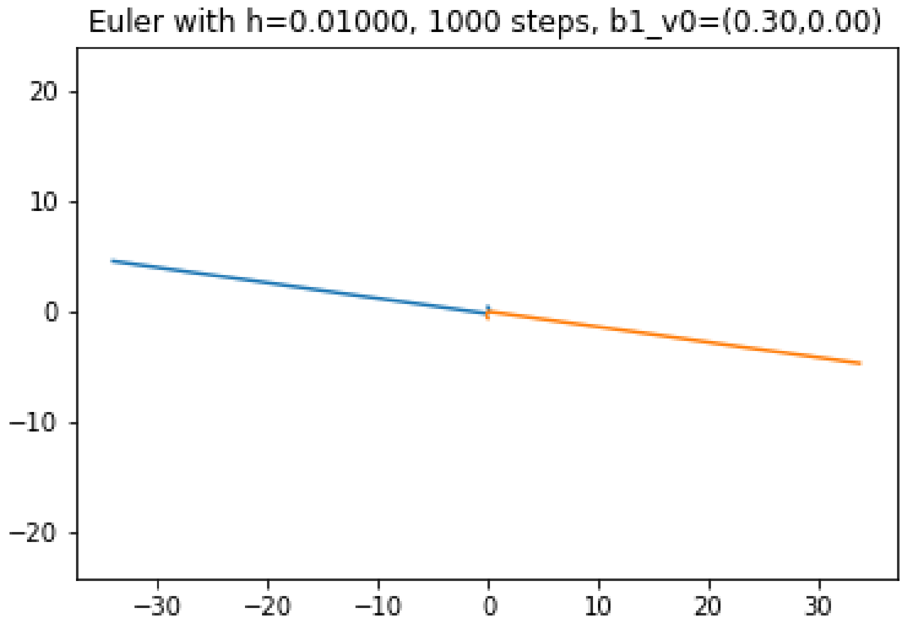
Because $M = G = 1$, Runge-Lenz-Vector and eccentricity are both $(0.21, -0.70)$ for this configuration

For $v_0 = 1.5 > \sqrt{2}$, the bodies hyperbolically fly past each other

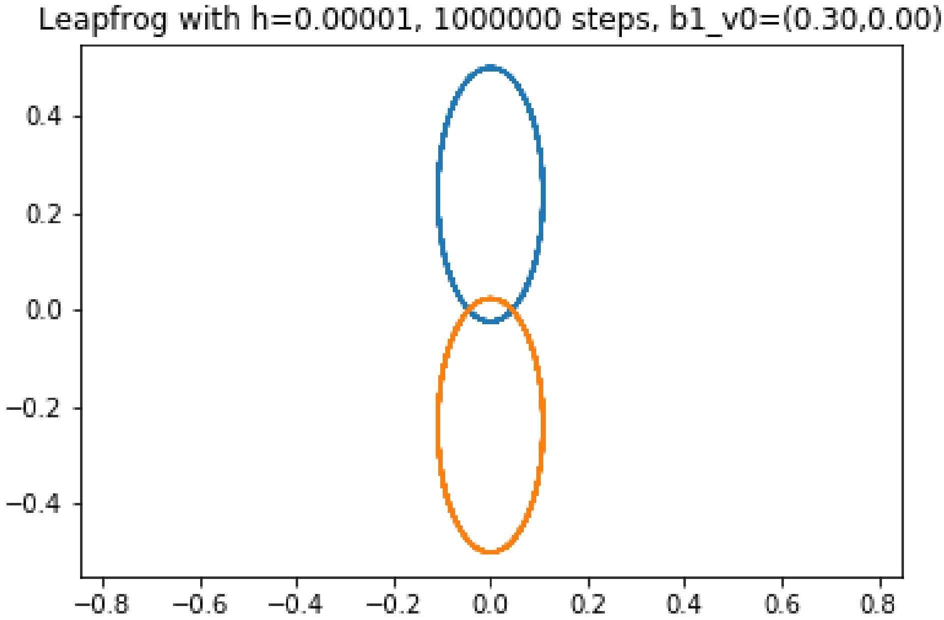


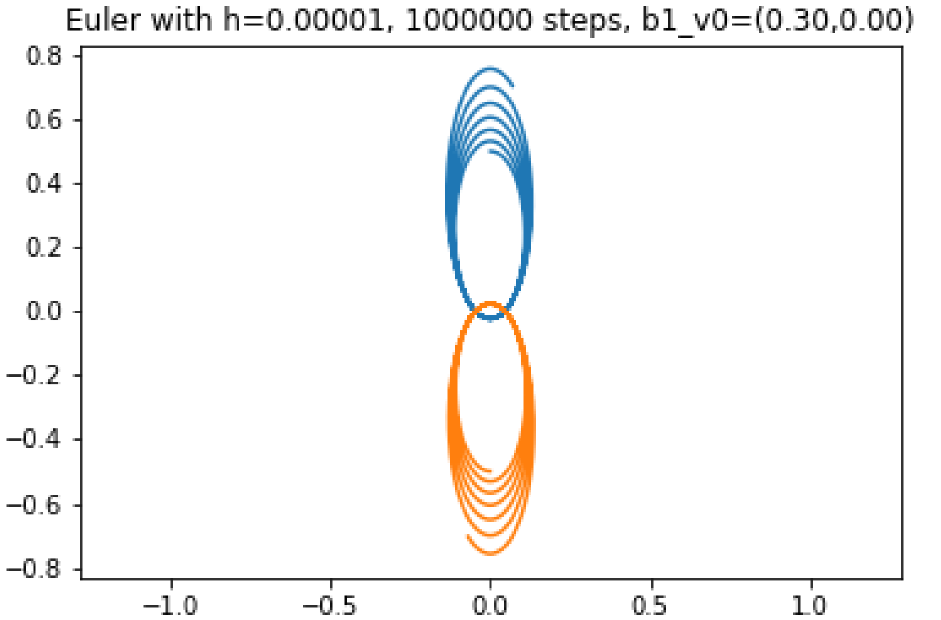
For $v_0 = 0.3$, the bodies ‘crash‘, giving them one large acceleration step catapulting them away from each other.





This can be solved by decreasing Δt to get an acceleration step that doesnt break. We used $\Delta t = 0.00001$





As can be seen, this is now convergent (at least using Leapfrog)

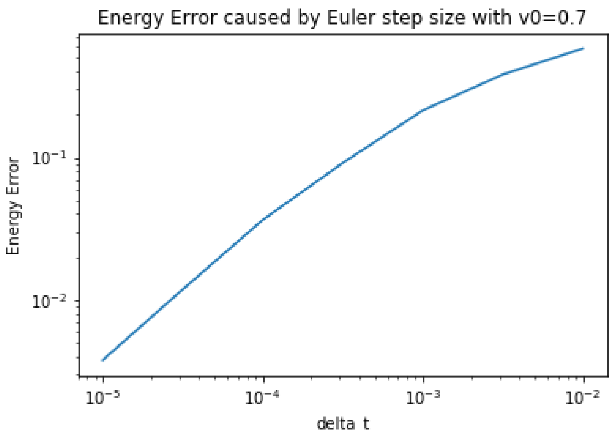
Exercise 5

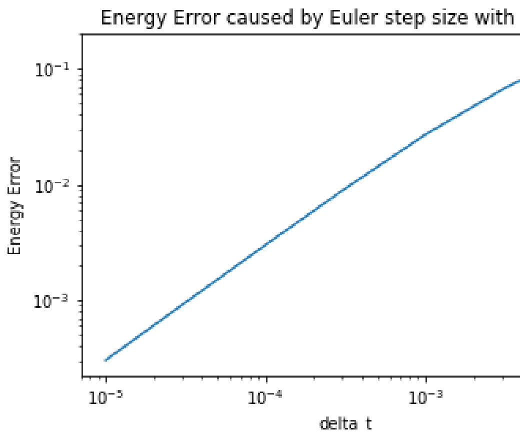
Now some direct comparisons are made between Leapfrog and Euler Integration Error.

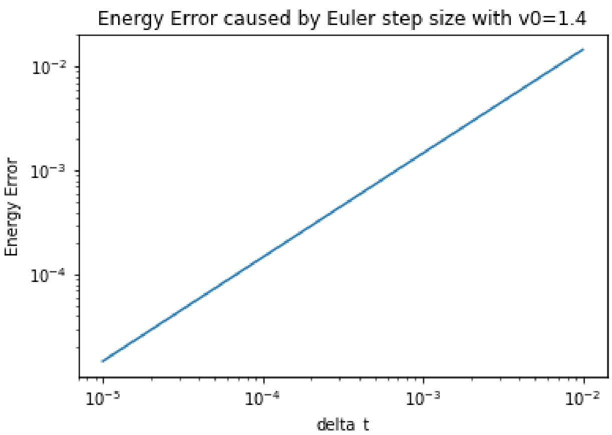
```
###5 Error Analysis

vs=[0.7,1.0,1.4]
hs=10**-np.linspace(2,5,7)

for v in vs:
    b1=Body(np.array([0,0.5]),np.array([v,0]),M)
    b2=Body(np.array([0,-0.5]),np.array([-v,0]),M)
    errors=[]
    for h in hs:
        e_0=b1.energy(b2)
        euler(b1,b2,int(10/h),h, plot_graph=False)
        errors.append(np.abs(e_0-b1.energy(b2)))
        b1.reset()
        b2.reset()
    plt.plot(hs,errors)
    plt.yscale("log")
    plt.xscale("log")
    plt.xlabel("delta_t")
    plt.ylabel("Energy Error")
    plt.title("Energy Error caused by Euler step size with v0=%.1f"%v)
    plt.savefig("Energy Error caused by Euler step size with v0=%.1f.png"
    plt.show()

###This is consistent with expectation since smaller steps=more exact (T
###now leapfrog
for v in vs:
    b1=Body(np.array([0,0.5]),np.array([v,0]),M)
    b2=Body(np.array([0,-0.5]),np.array([-v,0]),M)
    errors=[]
    for h in hs:
        e_0=b1.energy(b2)
        leapfrog(b1,b2,int(10/h),h, plot_graph=False)
        errors.append(np.abs(e_0-b1.energy(b2)))
        b1.reset()
```

```
        b2.reset()
    plt.plot(hs, errors)
    plt.yscale("log")
    plt.xscale("log")
    plt.xlabel("delta t")
    plt.ylabel("Energy Error")
    plt.title("Energy Error caused by Leapfrog step size with v0=%.
    plt.savefig("Energy Error caused by Leapfrog step size with v0=
    plt.show()
```

This yields some interesting graphics

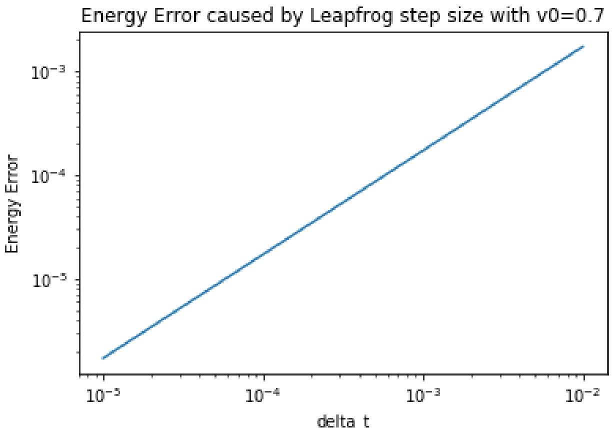


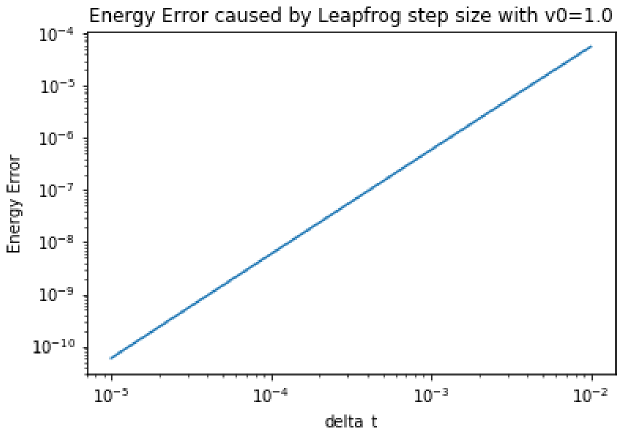


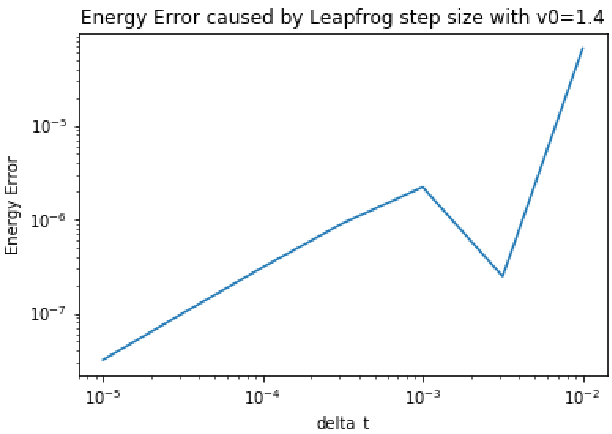


The Error scales polynomially with step size. This is according to expectation since the Taylor approximation made works best for small scales

We can compare this to leapfrog integration







The Error scales similar to Euler (except for the Spike in $v_0 = 1.4$ where some kind of eigenstate is met), but it is roughly two orders of magnitude smaller