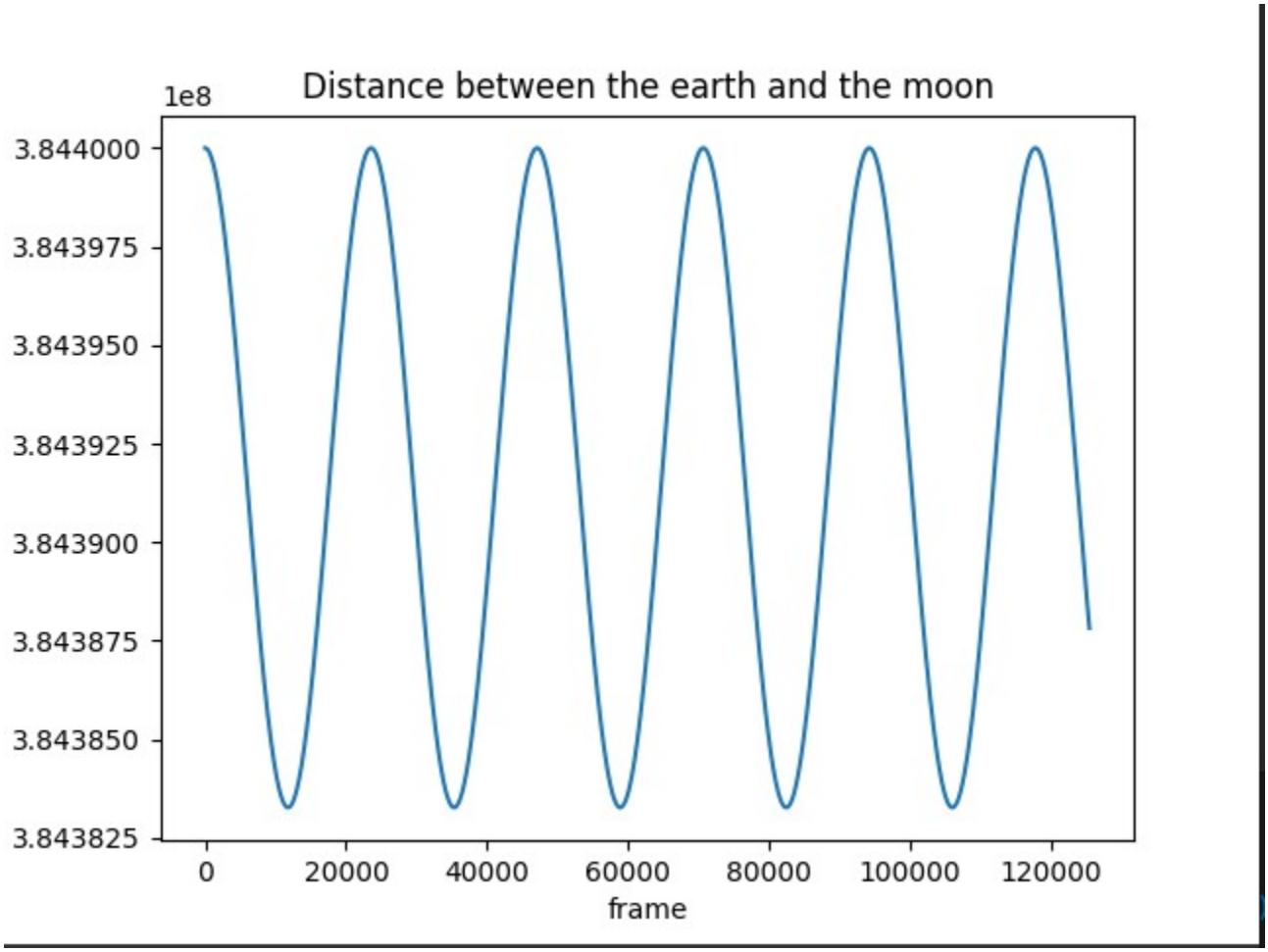
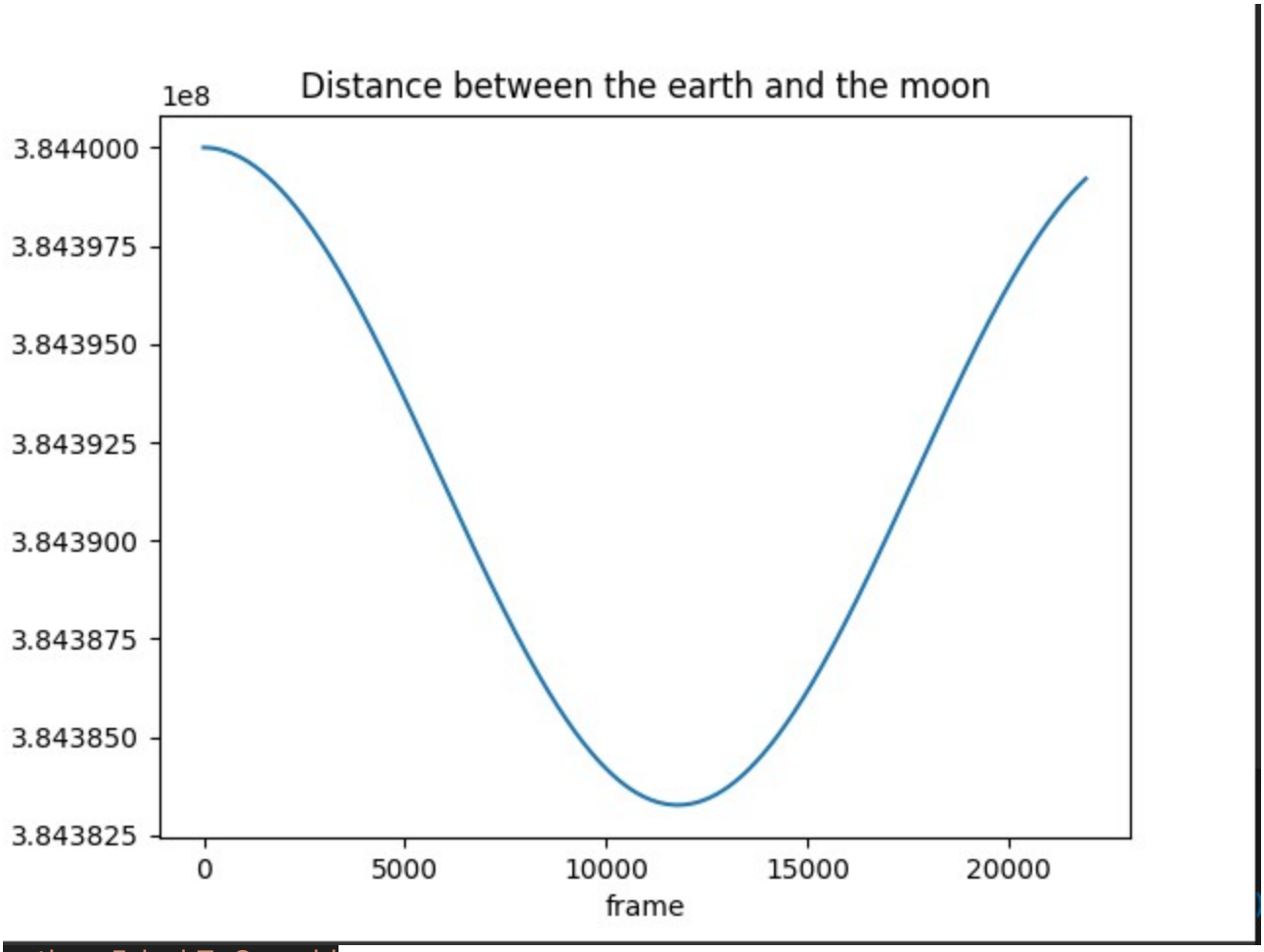
Interestingly, I noticed that the orbit was stable, with the lowest the orbit going to 349.8 million and the highest orbit reaching 384.4 million, even after reaching the limit of frames the orbit remained stable. If by stable what was meant was a perfectly circular orbit, one change I noticed that could be done to achieve a perfectly circular orbit was to increase the speed from 1000 to 1024.5. The reason this change circularizes the orbit is because the moon has the speed of 1000 given to at from the peak of the orbit, meaning that it loses energy as it moves to the peri apsis of the orbit, something shown by a loss of velocity as it moves. By increasing the speed, the moon has a closer energy to what gravity gives, resulting in a more circular orbit.

One interesting thing I noticed was that the earth wobbled, as if it was orbiting a different center of mass than what the center of the earth would suggest, and the amount of wobble changes depending on where the moon is and its speed.

Another interesting thing I noticed was that the less print messages I had, the faster the simulation would run, but the longer it would run the more resources it would use, resulting in an increase of frame limit slowing down the simulation past a certain threshold of roughly 5x the original limit.






```
author: Faisal Z. Qureshi
email: faisal.qureshi@ontariotechu.ca
website: http://www.vclab.ca
license: BSD
"""

# Week 4 assignment is posted. The orbit is unstable.
# The need to use initial velocity [0, 1000] to achieve a decent orbit.
# They can check if the orbit is stable by plotting the distance between the moon and the
earth over time.

import pygame
import sys
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import ode
import random
from datetime import datetime

# set up the colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
```

```
# constants
G = 6.674e-11 # N kg-2 m^2
Earth_Mass = 5.972e24 # kg
Moon_Mass = 7.34767309e22 # kg
Distance = 384400000. # m


# clock object that ensure that animation has the same speed
# on all machines, regardless of the actual machine speed.
clock = pygame.time.Clock()

# in case we need to load an image
def load_image(name):
image = pygame.image.load(name)
return image

class HeavenlyBody(pygame.sprite.Sprite):
def __init__(self, name, mass, color=WHITE, radius=0, imagefile=None):
pygame.sprite.Sprite.__init__(self)

if imagefile:
self.image = load_image(imagefile)
else:
self.image = pygame.Surface([radius*2, radius*2])
self.image.fill(BLACK)
pygame.draw.circle(self.image, color, (radius, radius), radius, radius)

self.rect = self.image.get_rect()
self.pos = np.array([0,0])
self.vel = np.array([0,0])
self.mass = mass
self.radius = radius
self.name = name
self.G = G
self.distances = []

def set_pos(self, pos):
self.pos = np.array(pos)

def set_vel(self, vel):
self.vel = np.array(vel)

def update1(self, objects, dt):# doesn't work if I have self.solver in init like in lab2 since
object is used here
for o in objects:
if o != self.name:
other = objects[o]

d = (other.pos - self.pos)
r = np.linalg.norm(d) #had trouble running print in f,so had to copy these over here
```

```
print(r)
self.solver = ode(self.f)
self.solver.set_integrator('dopri5')#specified rk4, dop853 is 8th order though so used this instead. swtich if needed.
self.solver.set_initial_value(np.concatenate((self.pos, self.vel)), 0)
self.solver.set_f_params(objects)#these slow down simulation I know, but it wouldn't work otherwise.
new_state = self.solver.integrate(self.solver.t + dt)
newpos = new_state[:2]
newvel = new_state[2:]

self.vel = newvel
self.pos = newpos

if self.name == 'earth':
self.distances.append(r)

def f(self, t, state, objects):
pos = state[:2]
vel = state[2:]

force = np.array([0.0, 0.0])
for o in objects:
if o != self.name:
other = objects[o]

d = (other.pos - pos)
r = np.linalg.norm(d)
u = d / r
force += u * G * self.mass * other.mass / (r * r)

return np.concatenate((vel, force / self.mass))

class Universe:
def __init__(self):
self.w, self.h = 2.6*Distance, 2.6*Distance
self.objects_dict = {}
self.objects = pygame.sprite.Group()
self.dt = 10.0

def add_body(self, body):
self.objects_dict[body.name] = body
self.objects.add(body)

def to_screen(self, pos):
return [int((pos[0] + 1.3*Distance)*640//self.w), int((pos[1] + 1.3*Distance)*640.//self.h)]

def update(self):
for o in self.objects_dict:
# Compute positions for screen
obj = self.objects_dict[o]
```

```
obj.update1(self.objects_dict, self.dt)
p = self.to_screen(obj.pos)

if False: # Set this to True to print the following values
print ('Name', obj.name)
print ('Position in simulation space', obj.pos)
print ('Position on screen', p)

# Update sprite locations
obj.rect.x, obj.rect.y = p[0]-obj.radius, p[1]-obj.radius
self.objects.update()

def draw(self, screen):
self.objects.draw(screen)

def main():

print ('Press q to quit')

random.seed(0)
# Initializing pygame
pygame.init()
win_width = 640
win_height = 640
screen = pygame.display.set_mode((win_width, win_height)) # Top left corner is (0,0)
pygame.display.set_caption('Heavenly Bodies')

# Create a Universe object, which will hold our heavenly bodies (planets, stars, moons,
etc.)
universe = Universe()

earth = HeavenlyBody('earth', Earth_Mass, radius=32, imagefile='earth-northpole.jpg')
earth.set_pos([0, 0])
moon = HeavenlyBody('moon', Moon_Mass, WHITE, radius=10)
moon.set_pos([int(Distance), 0])
moon.set_vel([0, 1024.5])

universe.add_body(earth)
universe.add_body(moon)

total_frames = 1000000
iter_per_frame = 50

frame = 0
ended =0
while frame < total_frames:
if False:
print ('Frame number', frame)

event = pygame.event.poll()
if event.type == pygame.QUIT:
```

```
ended=1
elif event.type == pygame.KEYDOWN and event.key == pygame.K_q:
ended=1
else:
pass

universe.update()
if frame % iter_per_frame == 0:
screen.fill(BLACK) # clear the background
universe.draw(screen)
pygame.display.flip()
frame += 1
if ended==1:
pygame.quit()
sys.exit
break
print("test")
print("test")
plt.figure(1)
plt.plot(earth.distances)
plt.xlabel('frame')
plt.ylabel('distance')
plt.title('Distance between the earth and the moon')
plt.show()
pygame.quit()
sys.exit



if __name__ == '__main__':
main()
```