Name Jack Baumann Grade\_\_\_\_\_\_\_\_\_\_\_\_\_

Computational Physics Exam 2

# Instructions

You are to work all of the following problems by writing and running Python programs. Your solutions will be graded on the following aspects:

* Documentation of programs, in other words, provide sufficient comments to follow logic in your programming.
* Correct use of appropriate algorithms.
* Explanations of results, specifically how you obtained a numerical answer.
* Formatting and labeling of all graphs.

Your solutions to the exam **must** include the following, all in one document:

1. your name in the name of the document;
2. answers to all questions
3. all associated graphs pasted into the document*;*
4. the Python source code for each problem.

**Turn in your work at the Dropbox on eCollege by midnight Friday, October 28th**.

Each problem is worth 25 points. Notes, textbook, and other *non-human* resources are permitted. You *may not* discuss the exam or your solutions with anyone but your instructor.If you have questions, please contact me in my office, via e-mail, or by phone. Note that I will be out of town on Thursday October 27th from 7 a.m. until 5 p.m., so please look over the exam and ask questions on Wednesday or, at latest, Friday morning.

Best wishes and happy computing!

1. Two freshmen are discussing the case a ball thrown vertically upward and whether air resistance makes the time to return from maximum height reached greater or less than the time to achieve maximum height. You step in and show them computational calculations for a whiffle ball’s motion.

Assume that the terminal velocity of a whiffle ball is 10.0 m/s and that it experiences Stokes’ law of air resistance. Develop a simulation using the Euler algorithm in which you plot (on one graph) the height of the whiffle ball as a function of time starting at y = 0 for three initial velocities of 5.0, 10.0, 15.0, and 20.0 m/s. In each case determine the time to rise and the time to fall from the maximum height reached and print out your results.

# Problem 1  
**print** "Problem 1"  
vt = 10.0  
h = 0.0001  
**for** v **in** (5.0, 10.0, 15.0, 20.0):  
 vi = v  
 y = 0  
 maxy = 0  
 t = 0  
 trise = 0  
 **while** y >= 0:  
 ay = -9.81 \* (1 + v / vt \*\* 2)  
 v += ay \* h  
 y += v \* h  
 t += h  
 **if** y > maxy:  
 maxy = y  
 trise = t  
 **print** "For an initial velocity of {0:0.1f}m/s, the time to rise was {1:0.5f}s and the time to fall was {2:0.5f}s."\  
 .format(vi, trise, t-trise)

The above code prints:

Problem 1

For an initial velocity of 5.0m/s, the time to rise was 0.49730s and the time to fall was 0.50560s.

For an initial velocity of 10.0m/s, the time to rise was 0.97150s and the time to fall was 1.00340s.

For an initial velocity of 15.0m/s, the time to rise was 1.42460s and the time to fall was 1.49430s.

For an initial velocity of 20.0m/s, the time to rise was 1.85850s and the time to fall was 1.97870s.

1. Suppose that the furnace in Clark Hall shuts down over Christmas break. The building loses heat in accordance with Newton's law of cooling with a rate constant 0.15 hr-1. The interior temperature is a pleasant 68 0F in January when the heating system fails. Use Heun’s algorithm in solving this problem.
   1. If the external temperature is 25 0F, graph the temperature inside the building as a function of time for 48 hours.
   2. Determine how long will it take for the interior temperature to fall to 32 0F.

**from** math **import** \*  
**import** matplotlib.pyplot **as** plt

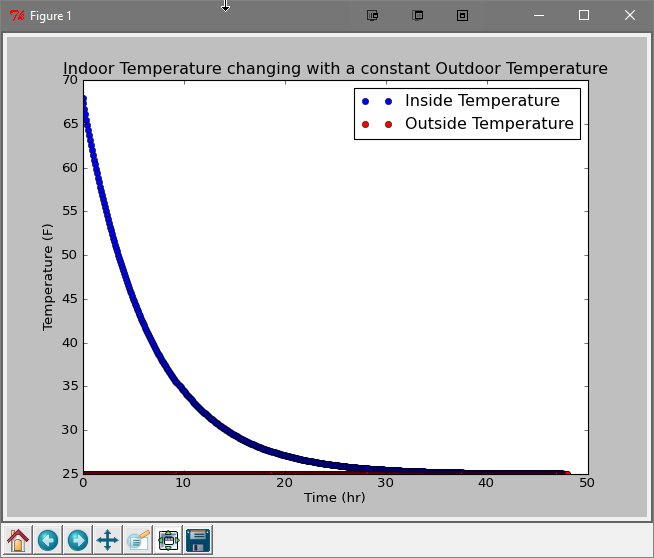
# Problem 2  
**print** "\nProblem 2"  
# Part A  
t = 0.0  
T = 68.0  
Ts = 25.0  
Tend = 0.0  
tend = 0.0  
r = 0.15  
h = 0.1  
tfreezing = 0  
plt.plot(t, T, 'bo', label="Inside Temperature")  
plt.plot(t, 25, 'ro', label="Outside Temperature")  
**while** t <= 48.0:  
 Tend = T + -r\*(T-Ts)\*h  
 tend = t + h  
 T += (-r\*(T-Ts) + -r\*(Tend-Ts))/2\*h  
 **if** tfreezing == 0 **and** 31.8 < T < 32.2:  
 tfreezing = t  
 plt.plot(t, T, 'bo')  
 plt.plot(t, 25, 'ro')  
 t += h  
# Part B  
**print** "Parts A and B: The time until Clark Hall reaches 32F is {0:.2f}h".format(tfreezing)  
plt.title("Indoor Temperature changing with a constant Outdoor Temperature")  
plt.xlabel("Time (hr)")  
plt.ylabel("Temperature (F)")  
plt.legend(loc='best')  
plt.show()

The above code prints:

Problem 2

Parts A and B: The time until Clark Hall reaches 32F is 11.90h

The above code displays the following graph:



* 1. If the outside temperature fluctuates according to  where *t* is in hours and *t* = 0 corresponds to 6 a.m. on the first day. Plot the temperature inside the hall for six days.
  2. Under the conditions in (c) determine when the interior temperature reaches 32 0F.

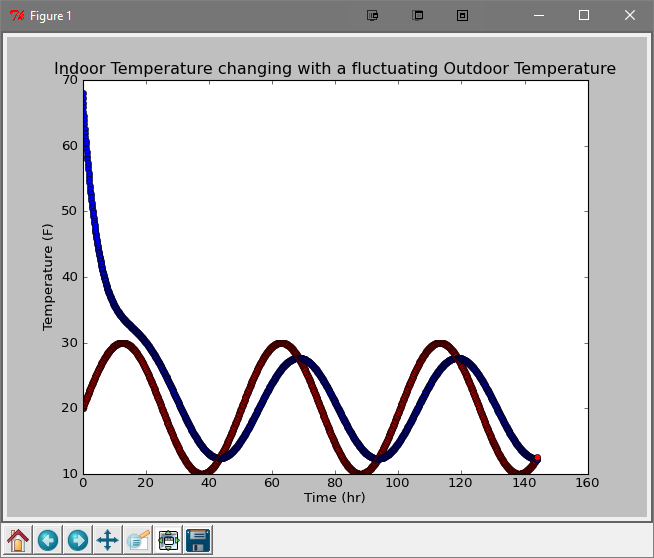
**from** math **import** \*  
**import** matplotlib.pyplot **as** plt

# Part C  
**def dT\_dt**(T, Ts): **return** -r\*(T-Ts)  
**def Ts**(t): **return** 20\*(1 + 0.5\*sin(t / 8))  
t = 0.0  
T = 68.0  
Tend = 0.0  
tend = 0.0  
r = 0.15  
h = 0.1  
tfreezing = 0  
plt.plot(t, T, 'bo', label="Inside Temperature")  
plt.plot(t, Ts(0), 'ro', label="Outside Temperature")  
**while** t <= 24.0\*6:  
 Tend = T + dT\_dt(T, Ts(t))\*h  
 tend = t + h  
 T += (dT\_dt(T, Ts(t)) + dT\_dt(Tend, Ts(t)))/2\*h  
 **if** tfreezing == 0 **and** 31.8 < T < 32.2:  
 tfreezing = t  
 plt.plot(t, T, 'bo')  
 plt.plot(t, Ts(t), 'ro')  
 t += h  
# Part D  
**print** "Parts C and D: The time until Clark Hall reaches 32F is {0:.2f}h".format(tfreezing)  
plt.title("Indoor Temperature changing with a fluctuating Outdoor Temperature")  
plt.xlabel("Time (hr)")  
plt.ylabel("Temperature (F)")  
plt.show()

The above code prints:

Parts C and D: The time until Clark Hall reaches 32F is 16.00h

The above code displays the following graph:



1. Investigate the effect of backspin on the trajectory of a golf ball by adapting your baseball program. A dimpled golf ball has a constant drag coefficient *C*D = 0.3 and the air has a density of 1.2754 kg/m3. The radius of an American golf ball is 0.02135 m and its mass is 0.04593 kg.
2. Determine the constant factor ** which you will use in your program. Note that the drag coefficient now has no dependence on velocity; the drag force leads simply to the acceleration term  .

# Problem 3  
**print** "\nProblem 3"  
# Part A  
g = 9.81  
dragcoeff = 0.3  
density = 1.2754  
radius = 0.02135  
mass = 0.04593  
kD = dragcoeff\*density\*(4\*pi\*radius\*\*2) / (2.0\*mass)  
**print** "Part A: The constant factor k\_D is {0:f}.".format(kD)

The above code prints:

Problem 3

Part A: The constant factor k\_D is 0.023859.

1. The lift factor for a dimpled golf ball is . For backspin, what should be the angle *ϕ* which is the angle between the spin direction and the vertical direction?

# Part B  
kL = 1.72E-3  
phi = 0.0  
**print** "Part B: The angle phi between the spin direction and the vertical direction should be {:.2f}.".format(phi)

The above code prints:

Part B: The angle phi between the spin direction and the vertical direction should be 0.00.

1. Compute the range (starting and ending at *y* = 0) for a golf ball with the following initial conditions and plot the trajectories on 3D graph:

|  |  |  |  |
| --- | --- | --- | --- |
| *θ* (0) | (rpm) | *v0* (m/s) | Range (m) |
| 8 | 3600 | 134 | 24.05512 |
| 23 | 7200 | 105 | 12.41382 |
| 45 | 10,800 | 90 | 5.17746 |

**from** math **import** \*   
**from** mpl\_toolkits.mplot3d **import** Axes3D

# Part C  
**print** "Part C:"  
**def graph**(theta, omega, v0):  
 X = []  
 Y = []  
 Z = []  
 vx = v0 \* cos(theta)  
 vy = abs(v0 \* sin(theta))  
 vz = 0  
 x = 0  
 y = 0  
 z = 0  
 t = 0  
 h = 0.001  
 **while** y >= 0:  
 X.append(x)  
 Y.append(y)  
 Z.append(z)  
 v = sqrt(vx\*\*2 + vy\*\*2 + vz\*\*2)  
 ax = -kD\*v\*vx + kL\*(vz\*omega\*sin(phi) - vy\*omega\*cos(phi))  
 ay = -kD\*v\*vy + kL\*vx\*omega\*cos(phi)  
 az = -kD\*v\*vz - kL\*vx\*omega\*sin(phi) - g  
 vx += ax\*h  
 vy += ay\*h  
 vz += az\*h  
 x += vx\*h  
 y += vy\*h  
 z += vz\*h  
 t += h  
 fig = plt.figure()  
 Ax = Axes3D(fig)  
 Ax.plot(X, Y, zs=Z, zdir='z')  
 **print** "The horizontal range of the trajectory of the golf ball is {:.5f}m.".format(abs(x))  
 plt.show()  
  
graph(8, 3600, 134)  
graph(23, 7200, 105)  
graph(45, 10800, 90)

The above code prints:

Part C:

The horizontal range of the trajectory of the golf ball is 24.05512m.

The horizontal range of the trajectory of the golf ball is 12.41382m.

The horizontal range of the trajectory of the golf ball is 5.17746m.

The above code displays the following graphs:

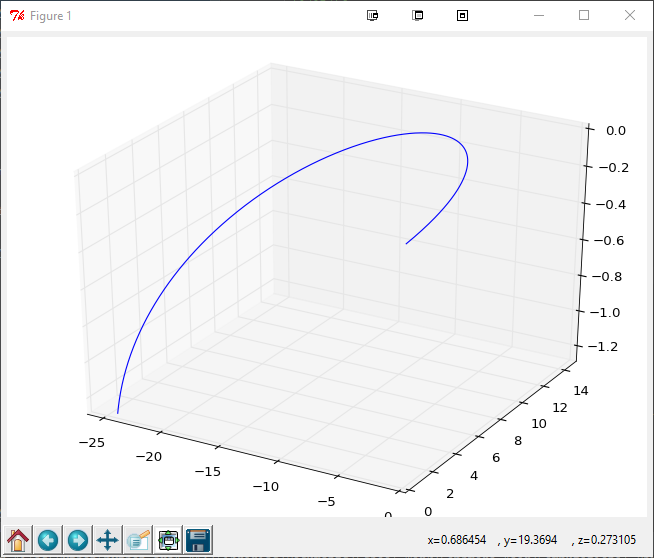


Figure 1: Trajectory of a golf ball where theta = 8, omega = 3600, and v0 = 134

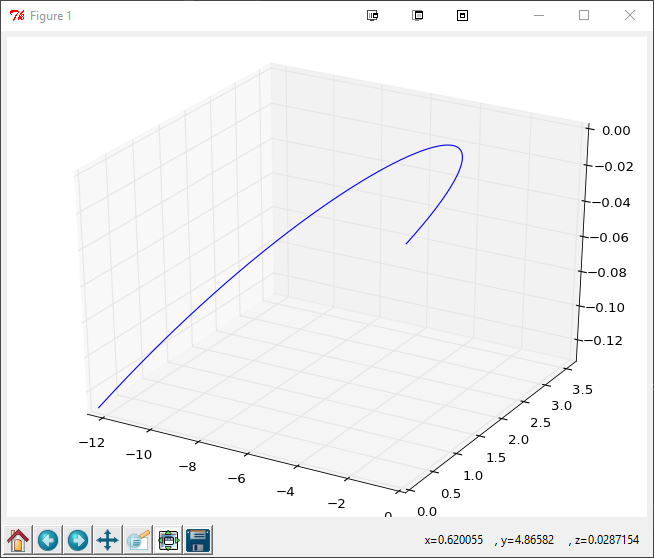


Figure 2: Trajectory of a golf ball where theta = 23, omega = 7200, and v0 = 105

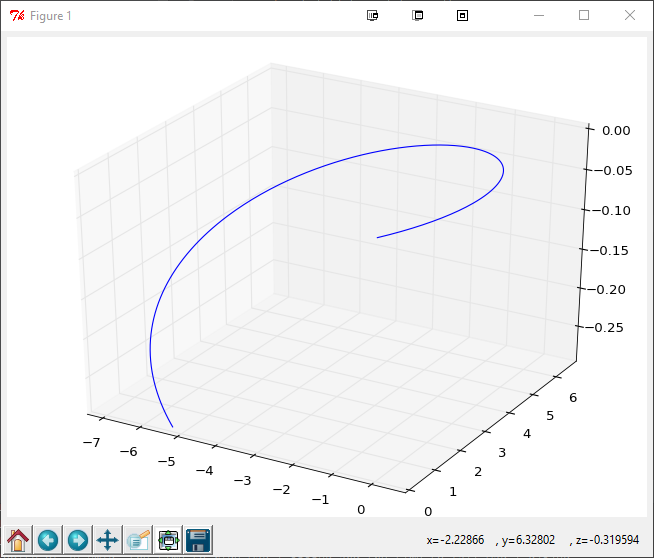
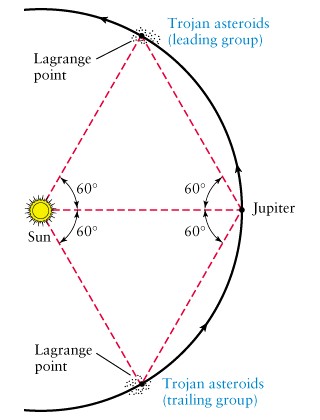


Figure 3: Trajectory of a golf ball where theta = 45, omega = 10800, and v0 = 90

1. The Trojan asteroids share Jupiter’s orbit by being at so-called Lagrange points L4 and L5. The Sun, Jupiter, and the Trojan asteroids are on the vertices of an equilateral triangle, as shown in the diagram. The distance from the Sun to Jupiter is 5.2 AU.

Use the VPython asteroid simulation that you developed for this class and alter it to have an asteroid orbiting at the location of the leading group of Trojan asteroids. Use initial conditions that are similar to Jupiter’s.

Once you have an asteroid orbiting the Sun at the position of the Trojan asteroids, place an additional asteroid in that location but with initial position and velocity that randomly differs from the first asteroid by less than 1%. (You can ignore the gravitational interaction between the two asteroids.)

1. From the geometry, what will be initial coordinates of the asteroid? What will be the components of the initial velocity of the asteroid for a circular orbit with a radius equal to that of Jupiter?

**from** \_\_future\_\_ **import** division  
**from** math **import** \*  
**import** random   
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**from** visual **import** \*  
**from** visual.graph **import** \*  
**from** vis **import** \*

# Problem 4  
**print** "\nProblem 4"  
G = 6.67E-11  
AU = 1.5E11  
YEAR = 365.25\*24\*60\*60  
  
# Part A  
r0 = 5.2\*AU  
sun = sphere(pos=(0, 0, 0), radius=100\*7E8, mass=2E30, color=(1, 1, 0))  
jupiter = sphere(pos=(r0, 0, 0), radius=800\*6.4E7, mass=9E25, color=color.magenta)  
asteroid = sphere(pos=(-r0\*cos(60), -r0\*sin(60), 0), radius=300\*6.4E7, mass=6.025E23, color=color.green)  
jupiter.vel = vector(0, sqrt(G\*sun.mass/r0), 0)  
asteroid.vel = vector(-3900, 12500, 0) # I had to set static values here because I was getting some weird effects.  
**print** "The initial position of the asteroid is ({0:0.5f}, {1:0.5f}) and its initial velocity is ({2:0.5f}, {3:0.5f})."\  
 .format(asteroid.x, asteroid.y, asteroid.vel.x, asteroid.vel.y)  
  
asteroid2 = sphere(pos=(asteroid.x-asteroid.x\*random.random(), asteroid.y-asteroid.y\*random.random(), 0),  
 radius=300\*6.4E7, mass=6.025E23, color=color.green)  
asteroid2.vel = vector(asteroid.vel.x-asteroid.vel.x\*random.random(), asteroid.vel.y-asteroid.vel.y\*random.random(), 0)

The above code prints:

Problem 4

The initial position of the asteroid is (742882124723.82190, 237752284459.72900) and its initial velocity is (-3900.00000, 12500.00000).

1. Does the two of asteroids stay in orbit for 10 or more orbits? If so, change the initial position and velocity by 2% and see if you observe a stable orbit for the second asteroid. Describe your results.

# Part B  
asteroid2 = sphere(pos=(asteroid.x-asteroid.x\*random.random()/100, asteroid.y-asteroid.y\*random.random()/100, 0),  
 radius=300\*6.4E7, mass=6.025E23, color=color.green)  
asteroid2.vel = vector(asteroid.vel.x-asteroid.vel.x\*random.random()/100,  
 asteroid.vel.y-asteroid.vel.y\*random.random()/100, 0)  
h = 1E6  
t = 0   
**while** True:  
 rAJ = mag(asteroid.pos - jupiter.pos)  
 rAJ2 = mag(asteroid2.pos - jupiter.pos)  
 asteroid.acc = -G\*sun.mass\*(asteroid.pos-sun.pos)/mag(asteroid.pos-sun.pos)\*\*3 +\  
 -G\*jupiter.mass\*(asteroid.pos-jupiter.pos)/rAJ\*\*3  
 asteroid.vel += asteroid.acc \* h  
 asteroid.pos += asteroid.vel \* h  
 asteroid2.acc = -G \* sun.mass \* (asteroid2.pos - sun.pos) / mag(asteroid2.pos - sun.pos) \*\* 3 + \  
 -G \* jupiter.mass \* (asteroid2.pos - jupiter.pos) / rAJ2 \*\* 3  
 asteroid2.vel += asteroid2.acc \* h  
 asteroid2.pos += asteroid2.vel \* h  
 jupiter.acc = -G\*sun.mass\*(jupiter.pos-sun.pos)/mag(jupiter.pos-sun.pos)\*\*3  
 jupiter.vel += jupiter.acc \* h  
 jupiter.pos += jupiter.vel \* h  
 t += h  
 rate(400)

With a variance in the initial conditions of 1%, the asteroids both stay in orbit for 10 or more orbits; however, with a variance of 2%, the asteroids spread out and fall out of orbit much more quickly.

I cannot place videos of the simulation in this document, so I have uploaded them alongside this document. As you can see in Exam2\_Problem4\_PartB\_Video2.mp4, which shows the program with a 2% variation, the asteroids are already out of orbit within 8 seconds of the simulation starting, whereas it takes a few more orbits for the 1% variation shown in Exam2\_Problem4\_PartB\_Video1.mp4.

1. Plot the distance between the two of the asteroids as a function of time and include a copy of your graph for a time interval of several orbits.

**from** \_\_future\_\_ **import** division  
**from** math **import** \*  
**import** random   
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**from** visual **import** \*  
**from** visual.graph **import** \*  
**from** vis **import** \*

# Problem 4  
**print** "\nProblem 4"  
G = 6.67E-11  
AU = 1.5E11  
YEAR = 365.25\*24\*60\*60  
# Part A  
r0 = 5.2\*AU  
sun = sphere(pos=(0, 0, 0), radius=100\*7E8, mass=2E30, color=(1, 1, 0))  
jupiter = sphere(pos=(r0, 0, 0), radius=800\*6.4E7, mass=9E25, color=color.magenta)  
asteroid = sphere(pos=(-r0\*cos(60), -r0\*sin(60), 0), radius=300\*6.4E7, mass=6.025E23, color=color.green)  
jupiter.vel = vector(0, sqrt(G\*sun.mass/r0), 0)  
asteroid.vel = vector(-3900, 12500, 0) # I had to set static values here because I was getting some weird effects.  
**print** "The initial position of the asteroid is ({0:0.5f}, {1:0.5f}) and its initial velocity is ({2:0.5f}, {3:0.5f})."\  
 .format(asteroid.x, asteroid.y, asteroid.vel.x, asteroid.vel.y)  
# Part B  
asteroid2 = sphere(pos=(asteroid.x-asteroid.x\*random.random()/100, asteroid.y-asteroid.y\*random.random()/100, 0),  
 radius=300\*6.4E7, mass=6.025E23, color=color.green)  
asteroid2.vel = vector(asteroid.vel.x-asteroid.vel.x\*random.random()/100,  
 asteroid.vel.y-asteroid.vel.y\*random.random()/100, 0)  
# Part C  
plot = gdisplay(x=0, y=0, height=400, width=600, title="Distance Between Two Asteroids vs. Time",  
 xtitle="time (yr)", ytitle="distance between the asteroids (m)")  
data = gcurve(color=color.white)  
h = 1E6  
t = 0  
**while** True:  
 rAJ = mag(asteroid.pos - jupiter.pos)  
 rAJ2 = mag(asteroid2.pos - jupiter.pos)  
 asteroid.acc = -G\*sun.mass\*(asteroid.pos-sun.pos)/mag(asteroid.pos-sun.pos)\*\*3 +\  
 -G\*jupiter.mass\*(asteroid.pos-jupiter.pos)/rAJ\*\*3  
 asteroid.vel += asteroid.acc \* h  
 asteroid.pos += asteroid.vel \* h  
 asteroid2.acc = -G \* sun.mass \* (asteroid2.pos - sun.pos) / mag(asteroid2.pos - sun.pos) \*\* 3 + \  
 -G \* jupiter.mass \* (asteroid2.pos - jupiter.pos) / rAJ2 \*\* 3  
 asteroid2.vel += asteroid2.acc \* h  
 asteroid2.pos += asteroid2.vel \* h  
 jupiter.acc = -G\*sun.mass\*(jupiter.pos-sun.pos)/mag(jupiter.pos-sun.pos)\*\*3  
 jupiter.vel += jupiter.acc \* h  
 jupiter.pos += jupiter.vel \* h  
 data.plot(pos=(t/YEAR, mag(asteroid.pos - asteroid2.pos)))  
 t += h  
 rate(400)

The above code produces the following graphs when simulated with 1% variance:

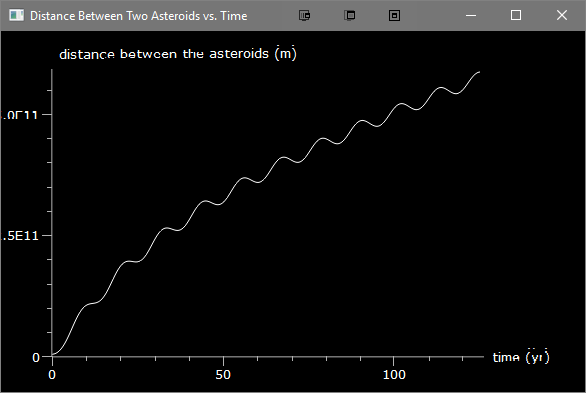


Figure 4: The graph of the Distance Between Two Asteroids vs. Time for approximately 10 orbits.

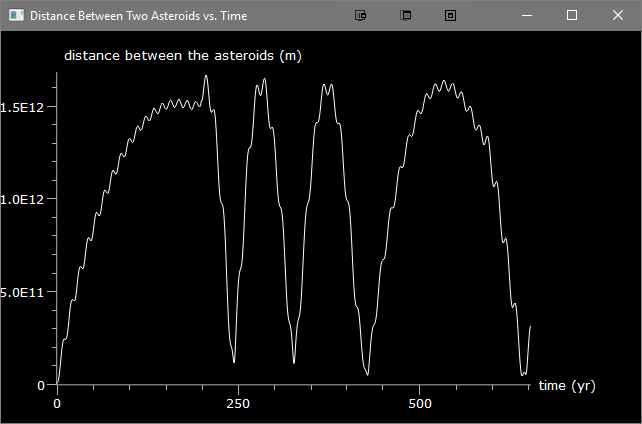


Figure 5: The graph of the Distance Between Two Asteroids vs. Time for a long interval of time.

1. Lastly, using a list (as was done in class) model ten or more asteroids being in the location of the leading Trojan asteroids. Give each asteroid a random initial position and velocity that differs from the original asteroid by less than 1%. Print a copy of the graph of the distance of each asteroid from its equilibrium position as a function of time. Does the group remain stable as they orbit Jupiter? Describe your results.

**from** \_\_future\_\_ **import** division  
**from** math **import** \*  
**import** random   
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**from** visual **import** \*  
**from** visual.graph **import** \*  
**from** vis **import** \*

# Problem 4  
**print** "\nProblem 4"  
G = 6.67E-11  
AU = 1.5E11  
YEAR = 365.25\*24\*60\*60  
  
# Part A  
r0 = 5.2\*AU  
sun = sphere(pos=(0, 0, 0), radius=100\*7E8, mass=2E30, color=(1, 1, 0))  
jupiter = sphere(pos=(r0, 0, 0), radius=800\*6.4E7, mass=9E25, color=color.magenta)  
asteroid = sphere(pos=(-r0\*cos(60), -r0\*sin(60), 0), radius=300\*6.4E7, mass=6.025E23, color=color.green)  
jupiter.vel = vector(0, sqrt(G\*sun.mass/r0), 0)  
asteroid.vel = vector(-3900, 12500, 0) # I had to set static values here because I was getting some weird effects.  
**print** "The initial position of the asteroid is ({0:0.5f}, {1:0.5f}) and its initial velocity is ({2:0.5f}, {3:0.5f})."\  
 .format(asteroid.x, asteroid.y, asteroid.vel.x, asteroid.vel.y)  
  
# Part B  
asteroid2 = sphere(pos=(asteroid.x-asteroid.x\*random.random()/100, asteroid.y-asteroid.y\*random.random()/100, 0),  
 radius=300\*6.4E7, mass=6.025E23, color=color.green)  
asteroid2.vel = vector(asteroid.vel.x-asteroid.vel.x\*random.random()/100,  
 asteroid.vel.y-asteroid.vel.y\*random.random()/100, 0)  
  
# Part C  
plot = gdisplay(x=0, y=0, height=400, width=600, title="Distance Between Each Asteroid's Current"  
 " and Initial Positions vs. Time",  
 xtitle="time (yr)", ytitle="distance (m)")  
data = gcurve(color=color.white)  
h = 1E6  
t = 0  
  
# Part D  
asteroids = [asteroid, asteroid2]  
**for** i **in** range(0, 10):  
 asteroids.append(sphere(pos=(asteroid.x-asteroid.x\*random.random()/100,  
 asteroid.y-asteroid.y\*random.random()/100, 0),  
 radius=300\*6.4E7, mass=6.025E23, color=color.green,  
 vel=vector(asteroid.vel.x-asteroid.vel.x\*random.random()/100,  
 asteroid.vel.y-asteroid.vel.y\*random.random()/100, 0),  
 acc=vector(0, 0, 0)))  
  
**while** True:  
 **for** a **in** asteroids:  
 a.acc = -G \* sun.mass \* (a.pos - sun.pos) / mag(a.pos - sun.pos) \*\* 3 + \  
 -G \* jupiter.mass \* (a.pos - jupiter.pos) / mag(a.pos - jupiter.pos) \*\* 3  
 a.vel += a.acc \* h  
 a.pos += a.vel \* h  
 data.plot(pos=(t / YEAR, mag(a.pos - vector(r0, 0, 0))))  
 jupiter.acc = -G\*sun.mass\*(jupiter.pos-sun.pos)/mag(jupiter.pos-sun.pos)\*\*3  
 jupiter.vel += jupiter.acc \* h  
 jupiter.pos += jupiter.vel \* h  
  
 t += h  
 rate(400)

The above code produces the following graph:

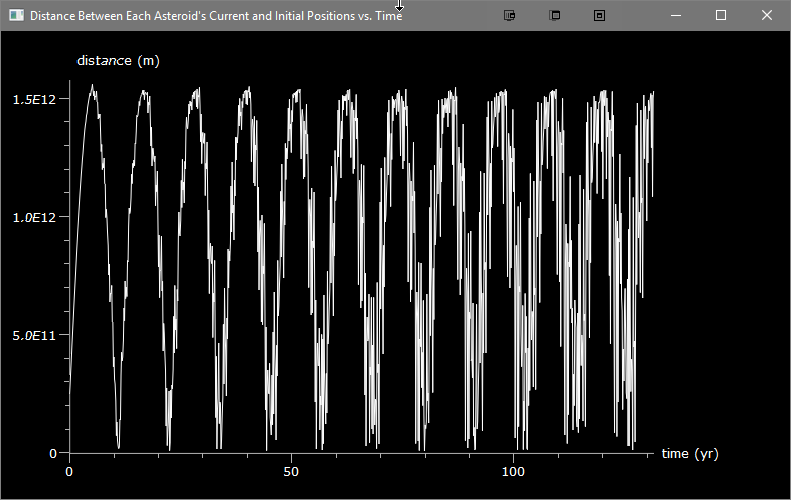


Figure 6: Distance Between Each Asteroid's Current and Initial Positions vs. Time over approximately 10 orbits.

The asteroids remain in a stable orbit, but as time passes they drift away from their original positions at a constant rate. In the video of the above simulation titled Exam2\_Problem4\_PartD\_Video1.mp4 that I have attached with this document, the asteroids slowly begin to drift away from the sun at a constant rate, as shown by the above graph.

1. What happens if the initial velocities of the additional asteroids are more than 10% of the first asteroid? Is the system still stable?

When the variance in the initial position and velocity of the asteroids is increased to greater than 10% the asteroids no longer orbit around the sun in the same path as Saturn. I have attached a video titled Exam2\_Problem4\_PartE\_Video1.mp4 that demonstrates the simulation with a 25% variance. In this case, the asteroids still orbit around the sun, however their orbits are not uniform nor stable.