Lab Report 3

Tejas Patel

May 16, 2024

1 Introduction

The Goal of this lab was to model the slingshot of the Voyager 1 Interstellar Probe leading up to and right after its approach to Jupiter. We used VPython to model the approach of the probe using launch and cruise data provided by NASA's Jet Propulsion Laboratory a few months leading up to the approach, and simulating the approach on our own after that

2 Model

Our calculations involved calculating strong and relevant gravitational forces acting on the space probe. We decided to include the Sun, Earth, Jupiter, and Saturn. Our general equation for gravitational force between two bodies was as follows:

$$f_{rocketbody} = m_{body} * m_{rocket} * G/r^2$$

Where G is the gravitational constant $6.67 \times 10^{-11} \mathrm{N} \cdot \mathrm{m}^2/\mathrm{kg}^2$ and r is the center of mass distance between the two objects. We used that equation to measure the orbit of Jupiter, Earth, and Saturn around the Sun, alongside the gravitational forces of the Sun, Earth, Jupiter, and Saturn on the probe.

2.1 Python Script (Abridged)

Full Python Script is available at the end of the document

```
from vpython import sphere, vector, color, rate, scene, attach_trail, arrow, label,
    cos, sin, pi, mag, norm, hat
jupiterdiameter = 6.78e6  # jupiter diameter
G = 6.67384e-11    # universal gravitational constant
AU = 1.496e11   #Astronomical unit(avg. dist Sun to Earth--for length scale purposes)
msun = 1.989e30     # mass of Sol (sun)
mearth = 5.97219e24     # mass of Earth
mjupiter = 1.898e27       # mass of jupiter
mrocket = 815
msaturn = 5.683e26
dt = 2 * 3600                          # time step
jupiterrocketdist = 500 * jupiterdiameter    # distance from jupiter considered a
    success
#Initial distances and velocities of plantary bodies
riearth = vector( 4.449401091798647E+10, 1.403470707888662E+08, -1.769063940881193E
    +04)
viearth = vector(-2.893748681913718E+04, 8.702013674306697E+03,7.726695004177664E-01)
rijupiter = vector(-3.952668829569589E+11, 6.792547950520715E+11, 6.054681409011871E
    +09)
vijupiter = vector(-1.144241460230606E+04, -5.972055209548326E+03, 2.807644771036928E
    +02)
rirocket =vector(-3.804368981984373E+11, 5.917727238669536E+11, 8.206215977999300E
    +09)
```

```
virocket =  vector(-1.325519458837047E+04, 4.932879806883460E+03, 1.178862520211088E
    +01)
risaturn = vector(-1.287092434829489E+12, 5.253855136589826E+11, 4.195904660432076E
    +10)
visaturn = vector(-4.172246290764435E+03, -8.964810326105848E+03, 3.222895141594253E
    +02)
#Defining planets
sun=sphere(pos=vector(0,0,0), radius=0.1*AU, color=color.yellow)
earth=sphere(pos=riearth, radius=0.01*AU, color=color.blue)
jupiter=sphere(pos=rijupiter, radius=0.01*AU, color=color.red)
rocket=sphere(pos=rirocket, radius=0.05*AU, color=color.orange)
saturn=sphere(pos=risaturn, radius=0.03*AU, color=color.blue)
# draw an arrow to show direction of initial velocity of rocket
rocketarrow1 = arrow(pos=earth.pos, axis=(sun.radius*2)*norm(virocket), color=color.
    white)


#set the scene
scene.range=1.3*mag(jupiter.pos)

#create display for timing information
tstr="Time: {:.0f} days".format(0)
tlabel=label(pos=vector(0,1.2*mag(jupiter.pos),0), text=tstr)

launchstr="Starting Date: 09051977."            # *** replace XXXXXXXX with your
    launch date

launchlabel=label(pos=vector(0,-1.2*mag(jupiter.pos),0), text=launchstr)


t=0

# *** add a comparison to the while statement below (inside the parentheses)
# *** so that the program will run until the rocket is within "jupiterrocketdist" of
    jupiter
while True:
    rate(200)  # Controls the speed of the simulation

    # Gravitational force calculation
    def gravitational_force(m1, m2, r1, r2):
        # Vector from object 1 to object 2
        r = r2 - r1
        # Magnitude of the gravitational force
        f_mag = G * m1 * m2 / mag(r)**2
        # Direction of the force
        f_dir = norm(r)
        return f_mag * f_dir
    f_rocket_sun = gravitational_force(mrocket, msun, rocket.pos, sun.pos)
    f_rocket_earth = gravitational_force(mrocket, mearth, rocket.pos, earth.pos)
    f_rocket_jupiter = gravitational_force(mrocket, mjupiter, rocket.pos, jupiter.pos
    )
    f_rocket_saturn = gravitational_force(mrocket, msaturn, rocket.pos, saturn.pos)
    total_force = f_rocket_sun + f_rocket_earth + f_rocket_jupiter + f_rocket_saturn
    rocket.vel += total_force / mrocket * dt
    rocket.pos += rocket.vel * dt
    print(mag(jupiter.pos) - mag(rocket.pos))

    # Update time and label
    t += dt
    tstr = "Time: {:.0f} days".format(t / (24 * 3600))
    tlabel.text = tstr
```

3 Table: Model data points - Approach and Slingshot around Jupiter

Time (s)	**Model Velocity (m/s)**	**Actual Velocity (m/s)**	**Uncertainty (m/s)**
7.77×10^6	7000	22000	
7.78×10^6	8000	24000	
7.78×10^6	9000	26000	
7.79×10^6	10000	28000	
7.80×10^6	13000	29000	2000
7.80×10^6	16000	28000	
7.81×10^6	20000	26000	
7.82×10^6	23000	24000	
7.83×10^6	24000	22000	
7.83×10^6	23000	21000	

Table 1: Comparison of Model and Actual Velocities approaching Jupiter, 12 hours before reaching the closest point to Jupiter to 12 hours after. Uncertainty was not calculated by us, however it was provided to us by the NASA Deep Space Network's calculations and at the time the calculations were made the uncertainty was approximately 1800 meters per second

4 Graph: Model vs Actual

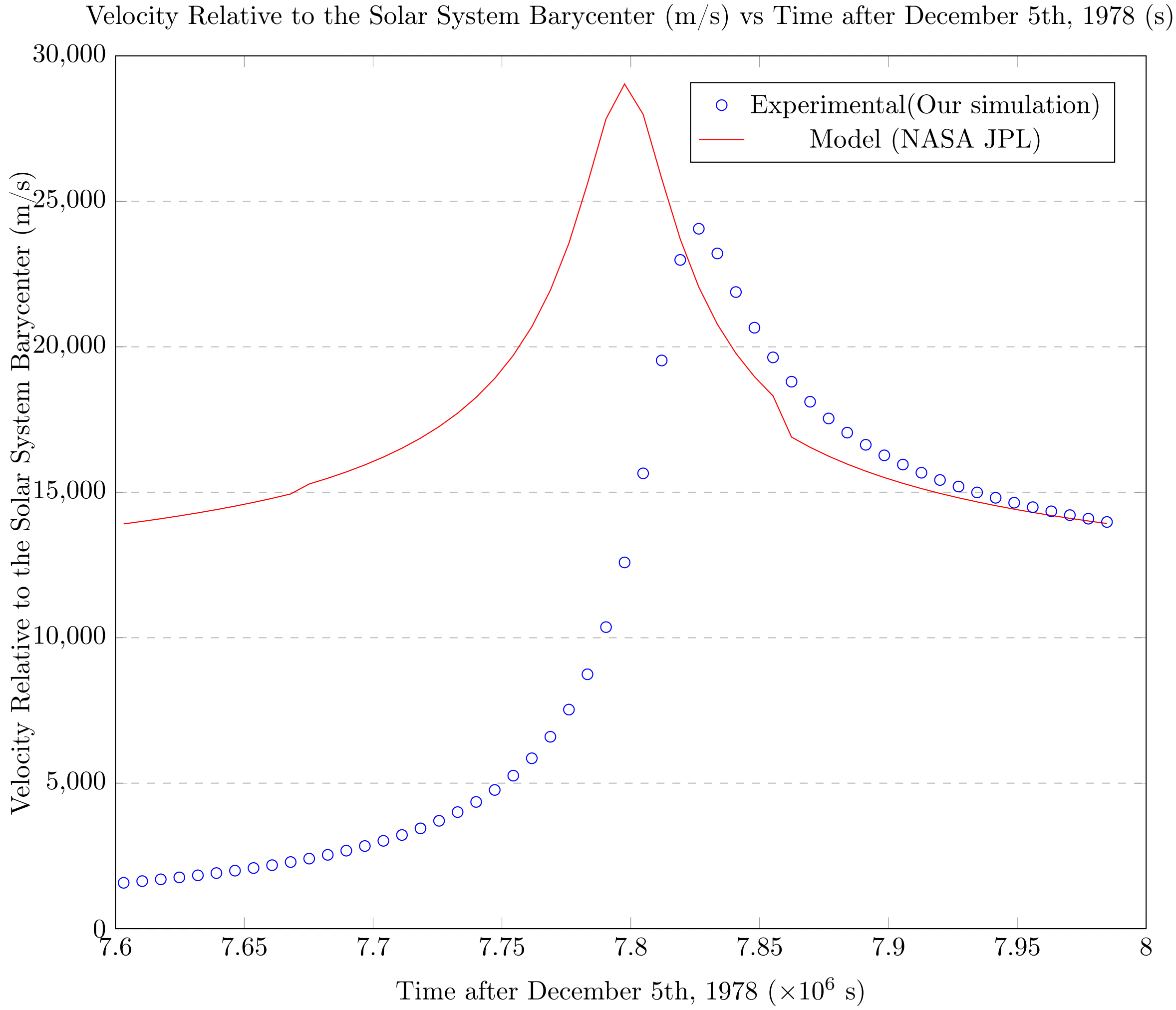

Figure 1: Our Model is the line graphed in red. It is data pulled straight from NASA JPL Horizons with velocities relative to the Solar System Barycenter. The Blue dots were our data points generated through our VPython simulation.

5 Analysis

5.1 Reasonable?

Our data loosely resembles the shape the NASA data gave us, where there is line slightly sloping up, spiking as the probe gets near Jupiter, then slowing down right after and settling to a speed above the incoming speed. Though the resemblence is not very strong, the shape of the graph is consistent, meaning that there were external factors at play when the simulation was taking place. Though the forces that were not account for were able to morph the graph, they were weak enough to a point where the graph kept its intended shape. Our closest approach to Jupiter was approximately 8 hours off of the actual closest approach to Jupiter, however that is not much of a concern as that is only 0.5% of the time leading up to the closest approach

5.2 Accuracy

We used the percent difference method of calculating the accuracy of our experiment. It was denoted as

$$\delta = \left| \frac{actual - expected}{expected} \right| \times 100\%$$

Using this formula we can find the percent errors on the lower and higher bounds of the NASA data uncertainty. For the higher end, we found our percent error in the beginning of the simulation, closest approach to Jupiter, end of simulation as 897.35%, 0.85%, and 12.50%, respectively.
For the lower end of the uncertainty, the percent differences came out to 668.84%, 15.81%, and 13.25%, respectively. This indicates that most of our error was in the approach to Jupiter and anything afterwards was similar to the actual data provided by NASA.

5.3 Causes of error

There are a few obvious causes, those would be we did not include all potential sources of gravitational force on the probe, Saturn and Mars. We also started the simulation 3 months before the approach to Jupiter, meaning there was plenty of time for the model probe to go off course or incur unexpected external forces. Also though NASA did not state any course corrections, there is a great chance that on the approach to Jupiter, there were course corrections, leading to our simulation not accounting for those, in turn making it inaccurate

6 Python Script (raw/full)

```
from vpython import sphere, vector, color, rate, scene, attach_trail, arrow,
label, cos, sin, pi, mag, norm, hat

scene.width = 630
scene.height = 600
scene.userspin = False
scene.userzoom = True


# To use this file, (1) fill in the control panel with appropriate values,
# then find and complete all 5 lines with a triple asterisk comment (***)
#
# Note that exponents are indicated by a double asterisk (**). Using a caret
# symbol (^) will cause the program to behave unpredictably.
#
# Some VPython functions that might be helpful are:
# "norm(a)" or "hat(a)"  is a unit vector with the same direction as the vector "
a"
# "mag(a)"   is a scalar that is the magnitude of the vector "a"

# Constants in metric units
jupiterdiameter = 6.78e6      # jupiter diameter
G = 6.67384e-11              # universal gravitational constant
AU = 1.496e11                # Astronomical unit (avg. dist from Sun to Earth--
for length scale purposes)
msun = 1.989e30              # mass of Sol (sun)
mearth = 5.97219e24              # mass of Earth
mjupiter = 1.898e27           # mass of jupiter
mrocket = 815
msaturn = 5.683e26          # mass of rocket

# *** Control panel (all quantities in metric units)
# *** To simplify the problem, set the z-component of
# *** the initial positions and velocities below to zero

dt = 2 * 3600                              # time step
jupiterrocketdist = 500 * jupiterdiameter    # distance from jupiter considered a
 success
riearth = vector( 4.449401091798647E+10, 1.403470707888662E+08,
-1.769063940881193E+04)                         # initial position of jupiter
viearth = vector(-2.893748681913718E+04, 8.702013674306697E+03,7.726695004177664E
-01)
rijupiter = vector(-3.952668829569589E+11, 6.792547950520715E+11,
6.054681409011871E+09)                      # initial position of earth
vijupiter = vector(-1.144241460230606E+04, -5.972055209548326E+03,
2.807644771036928E+02)                          # initial velocity of earth
                    # initial velocity of jupiter
rirocket =vector(-3.804368981984373E+11, 5.917727238669536E+11, 8.206215977999300
E+09)           #initial speed of rocket
virocket =  vector(-1.325519458837047E+04, 4.932879806883460E+03,
1.178862520211088E+01)          # launch angle of rocket (relative to +x axis)
risaturn = vector(-1.287092434829489E+12, 5.253855136589826E+11,
4.195904660432076E+10)
visaturn = vector(-4.172246290764435E+03, -8.964810326105848E+03,
3.222895141594253E+02)


# set up scene and objects


sun=sphere(pos=vector(0,0,0), radius=0.1*AU, color=color.yellow)

earth=sphere(pos=riearth, radius=0.01*AU, color=color.blue)
```

```
earthtrail=attach_trail(earth, radius=0.2*earth.radius, trail_type="points",
interval=2, retain=1000)
earth.vel = viearth

jupiter=sphere(pos=rijupiter, radius=0.01*AU, color=color.red)
jupitertrail=attach_trail(jupiter, radius=0.2*jupiter.radius, trail_type="points"
, interval=2, retain=1000)
jupiter.vel = vijupiter

rocket=sphere(pos=rirocket, radius=0.05*AU, color=color.orange)
rockettrail=attach_trail(rocket, radius=0.2*rocket.radius, trail_type="points",
interval=2, retain=1000)
rocket.vel = virocket

saturn=sphere(pos=risaturn, radius=0.03*AU, color=color.blue)
saturntrail=attach_trail(saturn, radius=0.2*rocket.radius, trail_type="points",
interval=2, retain=1000)
saturn.vel = visaturn
scene.camera.follow(rocket)

# draw an arrow to show direction of initial velocity of rocket
rocketarrow1 = arrow(pos=earth.pos, axis=(sun.radius*2)*norm(virocket), color=
color.white)


#set the scene
scene.range=1.3*mag(jupiter.pos)

#create display for timing information
tstr="Time: {:.0f} days".format(0)
tlabel=label(pos=vector(0,1.2*mag(jupiter.pos),0), text=tstr)

launchstr="Starting Date: 07302020."            # *** replace XXXXXXXX with your
 launch date

launchlabel=label(pos=vector(0,-1.2*mag(jupiter.pos),0), text=launchstr)


t=0

# *** add a comparison to the while statement below (inside the parentheses)
# *** so that the program will run until the rocket is within "jupiterrocketdist"
 of jupiter
while 1:
    rate(200)  # Controls the speed of the simulation

    # Gravitational force calculation
    def gravitational_force(m1, m2, r1, r2):
        # Vector from object 1 to object 2
        r = r2 - r1
        # Magnitude of the gravitational force
        f_mag = G * m1 * m2 / mag(r)**2
        # Direction of the force
        f_dir = norm(r)
        return f_mag * f_dir

    # Earth's motion
    f_earth_sun = gravitational_force(mearth, msun, earth.pos, sun.pos)
    earth.vel += f_earth_sun / mearth * dt
    earth.pos += earth.vel * dt

    # jupiter's motion
    f_jupiter_sun = gravitational_force(mjupiter, msun, jupiter.pos, sun.pos)
    jupiter.vel += f_jupiter_sun / mjupiter * dt
    jupiter.pos += jupiter.vel * dt
    f_saturn_sun = gravitational_force(msaturn, msun, saturn.pos, sun.pos)
```

```
    saturn.vel += f_saturn_sun / msaturn * dt
    saturn.pos += saturn.vel * dt

    # Rocket's motion
    f_rocket_sun = gravitational_force(mrocket, msun, rocket.pos, sun.pos)
    f_rocket_earth = gravitational_force(mrocket, mearth, rocket.pos, earth.pos)
    f_rocket_jupiter = gravitational_force(mrocket, mjupiter, rocket.pos, jupiter
.pos)
    f_rocket_saturn = gravitational_force(mrocket, msaturn, rocket.pos, saturn.
pos)
    total_force = f_rocket_sun + f_rocket_earth + f_rocket_jupiter +
f_rocket_saturn
    rocket.vel += total_force / mrocket * dt
    rocket.pos += rocket.vel * dt
    print(mag(jupiter.pos) - mag(rocket.pos))

    # Update time and label
    t += dt
    tstr = "Time: {:.0f} days".format(t / (24 * 3600))
    tlabel.text = tstr
```