

Report

on

Chapter 6

**Winning the Moon Race
with Apollo 8**

prepared by

Friedrich Bödefeld

Daniel Halama

Konstantin Reber

Teacher: Radosław Kycia

May 2025

Table of content

1. Abstract
2. Introduction
 - a. Aim
 - b. Scope
 - c. Methodology
3. Theory
4. Results
5. Summary
6. Bibliography

1. Abstract

This chapter dives into the Apollo 8 mission, a landmark in space exploration that marked the first time any human being orbited the Moon. At its core is the concept of the “free return” trajectory, a precisely calculated path that allowed the spacecraft to loop around the Moon and return safely to Earth without requiring engine burns on the way back. The chapter uses this scenario to explore the challenges of gravitational mechanics and the elegant workarounds engineers at NASA developed to handle the three-body problem. With Python’s turtle graphics, we simulate these complex ideas into a visual and approachable form, showing how code can serve as a powerful tool to understand and to help overcome the challenges encountered in engineering.

2. Introduction

a. Aim

The purpose of this project is to recreate a simplified form of the path Apollo 8 took as it looped around the Moon and returned to Earth. By building a visual simulation of this trajectory, the project helps make sense of the orbital mechanics that made the mission possible. It’s not about replicating exact flight data but about understanding how engineers approached such a complex problem and how those ideas can be translated into something visual, logical, and learnable through code.

b. Scope

The chapter focuses on a simplified visual simulation of Apollo 8’s flight path using Python’s turtle command. It doesn’t aim to be very realistic but instead centring around the core idea of the free return trajectory, a route that lets the spacecraft swing around the Moon and head back to Earth using gravitational fields. The simulation uses basic physics and simplified assumptions to stay manageable, while still conveying key concepts like gravity, momentum, and

orbital motion. The goal is to explore the historic space mission in an interactive and simplified way by keeping a realistic base.

c. Methodology

To address the given problem, we used a practical approach. As a primary resource, we used Real-World Python: A Hacker's Guide to Solving Problems by Lee Vaughan and with Code by Eric T. Mortenson. The book guided us through the theoretical problem of the free return trajectory and the three body problem and introduced step by step an example code for the underlying problems.

We used the turtle command of Python to illustrate and visually simulate space, celestial bodies and the spacecraft.

By carefully studying related literature and by the help of Real-World Python: A Hacker's Guide to Solving Problems, we were able to resolve all difficulties.

3. Theory

a. The Free Return Trajectory

Plotting a free return trajectory is a complex function but can be simplified into a 2D simulation using some key values: the spacecraft's starting position R_0 , its initial speed and direction V_0 , and the angle between the spacecraft and the Moon γ_0 . Before heading to the Moon, the spacecraft orbits Earth, waiting in the parking orbit for the right angle. After some final checks the rocket burn sends the spacecraft flying toward the Moon. Since the Moon is moving, you must aim ahead of its current position considering gravitational pull of both Earth and the Moon. After entering the gravitational field of the moon, the spacecraft uses its momentum and the gravitational force to sling back towards Earth.

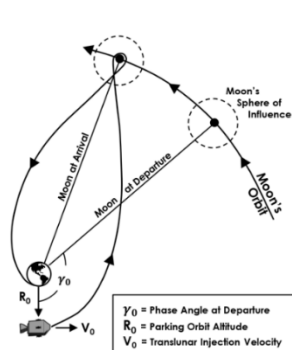


Fig. 1

b. The Three-Body Problem

The three-body problem is the challenge of predicting how three objects interact through gravity. While Newton's equations work well for two objects, adding a third complicates the math, that it can't be expressed through simple Algebra. Solving it requires powerful computers and lots of calculations. In 1961, Michael Minovitch found a practical way to handle this using the patched conic method, which simplifies the problem by treating it as two separate two-body problems: one near Earth and one near the Moon. In the following project, we will use this simplification.

4. Results

a. Turtle import and initialization of variables

```
1  from turtle import Shape, Screen, Turtle, Vec2D as Vec
2
3  G = 8
4  NUM_LOOPS = 4100
5  Ro_X = 0
6  Ro_Y = -85
7  Vo_X = 485
8  Vo_Y = 0
```

In this code we import from turtle graphics library the necessary classes. In a further step, we initialize and declare variables, such as gravitational constant, number of iterations, starting position and velocity.

b. Gravitational System class GravSys()

```
10 class GravSys():
11     def __init__(self):
12         self.bodies = []
13         self.t = 0
14         self.dt = 0.001
15
16     def sim_loop(self):
17         for _ in range(NUM_LOOPS):
18             self.t += self.dt
19             for body in self.bodies:
20                 body.step()
```

This code defines the class GravSys that is responsible for the gravity simulation of multiple interacting bodies. By using the step() method, every body gets updated for every iteration of the loop, advancing it's position and velocity.

c. Initialization of terrestrial bodies

```
24 class Body(Turtle):
25     def __init__(self, mass, start_loc, vel, gravsys, shape):
26         super().__init__(shape=shape)
27         self.gravsys = gravsys
28         self.penup()
29         self.mass = mass
30         self.setpos(start_loc)
31         self.vel = vel
32         gravsys.bodies.append(self)
```

This Body class simulates terrestrial objects in a gravitational field and is built on Python's Turtle graphics. When a Body is created, it gets assigned a mass, a starting position "start_loc", a starting velocity "vel", it's shape and is registered with the GravSys simulation system.

d. Force and vector of terrestrial bodies

```
35     def acc(self):
36         a = Vec(0, 0)
37         for body in self.gravsys.bodies:
38             if body != self:
39                 r = body.pos() - self.pos()
40                 a += (G * body.mass / abs(r)**3) * r
41         return a
```

This method calculates the gravitational acceleration acting on one celestial body from all the other bodies in the simulation. It loops through every other body, calculates the distance between them, and applies Newton's law of gravitation:

$$F = G * \frac{m_1 m_2}{|r|^2}$$

As we only want the acceleration, we divide both sides by self.mass.

$$a = G * \frac{m_2}{|r|^2}$$

To apply this as a vector, we also must multiply by $\frac{r}{|r|^2}$, which gives us the formula we used:

$$a = G * \frac{m_2}{|r|^3} * r$$

e. Calculating movement and changing shapes

```

50     def step(self):
51         """Calculate position, orientation, and velocity of a body."""
52         dt = self.gravsys.dt
53         a = self.acc()
54         self.vel = self.vel + dt * a
55         self.setpos(self.pos() + dt * self.vel)
56         if self.gravsys.bodies.index(self) == 2: # Index 2 = CSM.
57             rotate_factor = 0.0006
58             self.setheading((self.heading() - rotate_factor * self.xcor()))
59             if self.xcor() < -20:
60                 self.shape('arrow')
61                 self.shapesize(0.5)
62                 self.setheading(105)

```

In the step() method we use the in previous steps introduced parameters as velocity, position and acceleration of our interstellar bodies and multiplying it by the time increment to get an updated velocity and position. In the next step, the spacecraft slowly rotates based on where it is. Once it goes past a certain point on the left, it changes shape and points in a new direction.

f. main()

i. Setup screen

```

66     def main():
67         screen = Screen()
68         screen.setup(width=1.0, height=1.0)
69         screen.bgcolor('black')
70         screen.title("Apollo 8 Free Return Simulation")
71         gravsys = GravSys()

```

A new window gets created with the background colour set in black

ii. Attributes of Earth and Moon

```
73 image_earth = 'earth_100x100.gif'
74 screen.register_shape(image_earth)
75 earth = Body(1000000, (0, -25), Vec(0, -2.5), gravsys, image_earth)
76 earth.pencolor('white')
77 earth.getscreen().tracer(0, 0)
78
79 image_moon = 'moon_27x27.gif'
80 screen.register_shape(image_moon)
81 moon = Body(32000, (344, 42), Vec(-27, 147), gravsys, image_moon)
82 moon.pencolor('gray')
```

For further calculations, Earth and Moon are initialized with important attributes such as weight, movements, gravitational system. For visual interpretation of the calculated results both terrestrial entities are represented as a gif.

iii. CSM in Turtle

```
85 csm = Shape('compound')
86 cm = ((0, 30), (0, -30), (30, 0))
87 csm.addcomponent(cm, 'white', 'white')
88 sm = ((-60, 30), (0, 30), (0, -30), (-60, -30))
89 csm.addcomponent(sm, 'white', 'black')
90 nozzle = ((-55, 0), (-90, 20), (-90, -20))
91 csm.addcomponent(nozzle, 'white', 'white')
92 screen.register_shape('csm', csm)
93
94 ship = Body(1, (Ro_X, Ro_Y), Vec(Vo_X, Vo_Y), gravsys, 'csm')
95 ship.shapesize(0.2)
96 ship.color('white')
97 ship.getscreen().tracer(1, 0)
98 ship.setheading(90)
99
100 gravsys.sim_loop()
```

A custom spacecraft shape called 'csm' is created, using multiple components as described in graphic Fig. 2. The ship is then initialized with position, velocity, and appearance settings before starting the gravity simulation loop.

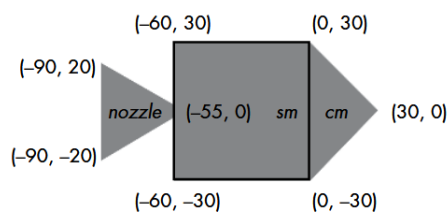


Fig. 2

iv. Starts program if name is equal to main()

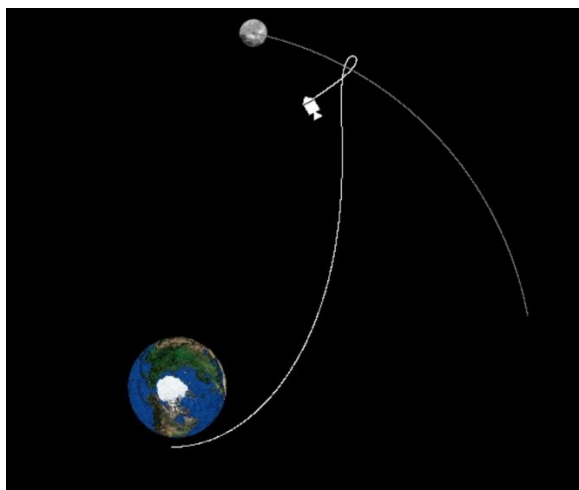


Fig. 3

5. Summary

As shown in Fig. 3 we were able to successfully simulate the free return trajectory using a simplified model of the three-body problem. Even though it uses simplistic mechanics, the program gives valuable insight into the complicity of gravitational forces and reveals the genius of one of mankind's brightest minds, Isaac Newton. Additionally, we could experience the difficulties NASA engineers had to overcome almost 60 years ahead of us.

6. Bibliography

Literature: Real-World Python: A Hacker's Guide to Solving Problems by Lee Vaughan

Code by Eric T. Mortenson with additions of the Authors of this report

Pictures: Fig. 1: Real-World Python: A Hacker's Guide to Solving Problems

Fig. 2: Real-World Python: A Hacker's Guide to Solving Problems

Fig. 3: own recording