Lunar Ascent: From Earth to Moon's Orbit

Jason Vasquez, Ethan Crawford, Dylan Skinner, Dallin Stewart

15 April 2024

Abstract

We present a simulation of a rocket’s path from Earth's surface to the orbit of the Moon. We model the complex dynamics involved in rocket propulsion, gravitational interactions, and orbital mechanics by leveraging differential equations and numerical methods. Our simulation captures the stages of ascent, including liftoff, trajectory optimization, and orbital insertion. Through detailed analysis and visualization, we explore the critical factors influencing orbital maneuvers required for this trajectory. The results offer insights into the challenges and considerations of space exploration, paving the way for future missions to celestial bodies beyond Earth's orbit.

Background

Space travel presents a highly relevant and classic optimal control problem, influenced by numerous factors such as drag, fuel consumption, gravitational forces from multiple celestial bodies, and varying mass considerations. Navigating a rocket from Earth's surface to the Moon's orbit requires modeling complex dynamics that encompass rocket propulsion, gravitational interactions, and orbital mechanics. We rely on a set of differential equations and numerical methods to simulate and analyze these dynamics in order to accurately optimize trajectory and orbital insertion.

Understanding and optimizing orbital maneuvers for interplanetary travel has been a longstanding challenge in aerospace engineering and space exploration. Previous research has explored various aspects of trajectory optimization, spacecraft propulsion, and orbital transfers to achieve efficient and successful missions beyond Earth's immediate vicinity. One such investigation is "Minimum-time Earth–Moon and Moon–Earth orbital maneuvers using time-domain finite element method (https://www.sciencedirect.com/science/article/pii/S0094576509003956)" by S.A. Fazelzadeh and G.A. Varzandian that employed a method similar to ours. Another instance of prior research that our project built upon was a Volume 4 example project from 2020 'Rocket Launch Trajectory' by Ahn et. al. This project attempted to reach Earth's orbit, and we wanted to build on this by leaving Earth's orbit and reaching the moon's. Notably, the formulation and solution of optimal control problems have been instrumental in advancing our understanding of spaceflight dynamics and guiding mission planning efforts in order to achieve recent feats such as landing on the moon and Mars.

This study builds upon existing research by focusing on the specific trajectory from Earth's surface to the moon's orbit. By conducting detailed analysis and visualizing critical factors affecting orbital maneuvers, our simulation provides valuable insights into the challenges and considerations of space exploration beyond Earth's orbit. The results obtained pave the way for enhanced mission planning and spacecraft design, contributing to the realization of future missions to celestial bodies throughout the solar system and beyond.

Mathematical Representation

To solve this problem, we defined a cost functional, a state-space evolution equation, and initial and endpoint conditions. We then utilized Pontraygin's maximum principle (PMP) to formulate the co-state evolution equations and solve for the optimal control in terms of the co-state. This theorem allows us to set up a boundary-value problem and solve for the state, costate and optimal control. We define the code and steps for this process below. We also made several simplifying assumptions in this project as follows.

Assumptions

1. Rocket, earth, and moon have point mass for the sake of calculating gravity
2. Rocket, earth, and moon are the only objects for the sake of calculating gravity (no influence of the sun, stars, etc)
3. Rocket, earth, and moon lie on a 2D plane
4. Moon's orbit is perfectly circular
5. Rocket's mass does not change
6. Rocket is single stage
7. The earth's position remains at the origin of the 2D plane
8. The moon's position at time zero is on the x-axis

Unless otherwise specified, we define the units for this project as follows:

- Mass: kilograms (kg)
- Distance: meters (m)
- Force: newtons (N)
- Angle: degrees (°)
- Speed: meters per second (m/s)
- Time: seconds (s)

Cost Functional

We seek to find the optimal thrust for a rocket to leave Earth and begin orbiting the moon. To do so, we set up our cost functional as

$$J[\mathbf{u}] = \int_0^{t_f} \frac{c}{2}\|\mathbf{u}(t)\|^2 dt,$$

where we can interpret $\mathbf{u} = \begin{bmatrix} u_x & u_y \end{bmatrix}$ as the thrust in the x- and y-directions, respectively. We also define our initial conditions as

$$\begin{aligned} x(0) &= R_e, & x(t_f) &= L_m\cos(\theta) + (R_m + h_f)\cos(90+\theta) \\ y(0) &= 0, & y(t_f) &= L_m\sin(\theta) + (R_m + h_f)\sin(90+\theta) \\ x'(0) &= 0, & x'(t_f) &= v_f\cos(\theta) \\ y'(0) &= \omega_e, & y'(t_f) &= v_f\sin(\theta) \end{aligned}$$

where

- c is the scaling constant for the cost of thrust
- R_e is the radius of the Earth,
- L_m is the distance from the Earth to the moon,
- R_m is the radius of the moon,
- h_f is the desired height above the moon to enter orbit,
- v_f is the final velocity of the rocket to enter orbit,
- ω_e is the angular velocity of the Earth at the equator,
- θ is the angle between the x-axis and the line between the Earth and the moon, assuming the Earth is at the origin.

State Space

We define our state space equation as

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix},$$

and our state space evolution equation as

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ x'' \\ y'' \end{bmatrix}$$

where x'' and y'' are the acceleration in the x and y directions, respectively. Using Newton's second law, this gives:

$$x'' = -\frac{GM_e x}{(x^2+y^2)^{3/2}} + \frac{GM_m(L_x - x)}{((L_x-x)^2+(L_y-y)^2)^{3/2}} + \frac{u_x}{m_r} - \frac{\frac{1}{2}cA\rho(x,y)x'^2}{m_r}$$
$$y'' = -\frac{GM_e y}{(x^2+y^2)^{3/2}} - \frac{GM_m(L_y - y)}{((L_x-x)^2+(L_y-y)^2)^{3/2}} + \frac{u_y}{m_r} - \frac{\frac{1}{2}cA\rho(x,y)y'^2}{m_r}.$$

where

- G is the universal gravitational constant
- M_e is the mass of the Earth
- M_m is the mass of the moon
- m_r is the mass of the rocket
- u_x is the rocket's thrust in the x direction
- u_y is the rocket's thrust in the y direction
- L_x is the x position of the moon
- L_y is the y position of the moon
- c is the drag constant $\frac{1}{4}$
- A is the surface area of the top of the rocket
- $\rho(x, y)$ is the air density at position (x, y)

Hamiltonian

We now derive the Hamiltonian equation. We have

$$H = \mathbf{p}\cdot\mathbf{x}' - L = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix}\begin{bmatrix} x' \\ y' \\ x'' \\ y'' \end{bmatrix} - \frac{c}{2}\|\mathbf{u}\|^2$$
$$= p_1 x' + p_2 y' + p_3 x'' + p_4 y'' - \frac{c}{2}u_x^2 - \frac{c}{2}u_y^2.$$

Plugging in x'' and y'' into the Hamiltonian equation, we get

$$H = p_1 x' + p_2 y' - p_3\left(-\frac{GM_e x}{(x^2+y^2)^{3/2}} + \frac{GM_m(L_x-x)}{((L_x-x)^2+(L_y-y)^2)^{3/2}} + \frac{u_x}{m_r} - \frac{\frac{1}{2}cA\rho(x,y)x'^2}{m_r}\right)$$
$$- p_4\left(-\frac{GM_e y}{(x^2+y^2)^{3/2}} - \frac{GM_m(L_y-y)}{((L_x-x)^2+(L_y-y)^2)^{3/2}} + \frac{u_y}{m_r} - \frac{\frac{1}{2}cA\rho(x,y)y'^2}{m_r}\right) -$$

Co-State Evolution

We now derive the necessary equations for the PMP. We first derive the necessary equations for the adjoint equations (co-state evolution). We have

$$\dot{p}_1 = -\frac{\partial H}{\partial x} = -p_3\left(\frac{3GM_m(L_x-x)^2}{\left((L_x-x)^2+(L_y-y)^2\right)^{5/2}} - \frac{GM_m}{\left((L_x-x)^2+(L_y-y)^2\right)^{3/2}} + \frac{3GM_e x^2}{(x^2+y^2)^{5/2}} - \frac{}{(}\right.$$
$$- p_4\left(\frac{-3GM_m(L_x-x)(L_y-y)}{\left((L_x-x)^2+(L_y-y)^2\right)^{5/2}} + \frac{3GM_e xy}{(x^2+y^2)^{5/2}}\right)$$
$$\dot{p}_2 = -\frac{\partial H}{\partial y} = -p_3\left(\frac{3GM_m(L_x-x)(L_y-y)}{((L_x-x)^2+(L_y-y)^2)^{5/2}} + \frac{3GM_e xy}{(x^2+y^2)^{5/2}}\right)$$
$$- p_4\left(\frac{-3GM_m(L_y-y)^2}{((L_x-x)^2+(L_y-y)^2)^{5/2}} + \frac{GM_m}{((L_x-x)^2+(L_y-y)^2)^{3/2}} + \frac{3GM_e y^2}{(x^2+y^2)^{5/2}} - \frac{}{(}\right.$$
$$\dot{p}_3 = -\frac{\partial H}{\partial x'} = -p_1 + \frac{cA\rho(x,y)x'}{m_r}$$
$$\dot{p}_4 = -\frac{\partial H}{\partial y'} = -p_2 + \frac{cA\rho(x,y)y'}{m_r}.$$

Optimal Control

Using $\frac{\partial H}{\partial \tilde{u}} = 0$, we get our optimal control $\tilde{u}$ defined as

$$\tilde{u} = \frac{1}{c}\begin{bmatrix} p_3 \\ p_4 \end{bmatrix}.$$

Code Set Up

Imports

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_bvp
import matplotlib.image as mpimg
```

Constants

The specs for the rocket come from the SpaceX Falcon 9 rocket, found at https://www.spacex.com/vehicles/falcon-9/ (https://www.spacex.com/vehicles/falcon-9/). Other planetary constants taken from http://www.braeunig.us/space/constant.html (http://www.braeunig.us/space/constant.html)

In [ ]:
```
# ROCKET
Mr0 = 5.49054e5              # initial mass of rocket
rocket_height = 70           # height of the rocket (not distance from
earth)
rocket_diameter = 3.7        # max diameter of rocket
thrust_max1 = 7.56e6         # max thrust of stage 1
thrust_max2 = 9.81e2         # max thrust of stage 2
A = rocket_diameter * np.pi # area that the air hits

# EARTH
Ve = 0                    # velocity of the earth
Me = 5.97219e24           # mass of the earth
we = 460.0                # angular velocity of the earth at the equator
Re = 6.3781e6             # radius of the earth

# MOON
Vm = 3.683e3              # orbital speed of the moon
Mm = 7.34767309e22        # mass of the moon
wm = 1.67e1               # angular velocity of the moon at its equator
Rm = 1.738e6              # radius of the moon
P = 6.552e2               # period of the orbit of the moon
hf = 1e2                  # final distance between rocket and moon
vf = 1e1                  # orbital speed of the moon

# SPACE
Lm = 3.84e8               # distance from the earth to the moon
G = 6.673e-11             # graviational constant in (N * m^2) / (kg^2)

# DRAG
p0 = 1.01325e5            # atmospheric pressure at sea level in Pascals
Hn = 1.04e1               # constant for drag force
K = 1e5                   # Karman line, when drag is zero and space beg
ins at 100 km
c = 0.25                  # coefficient of drag for the rocket shape
```

Helper Functions

In [ ]:
```
def moon_position(t):
    """
    Calculate the position vector of the Moon at a given time.
    @param t: (float) Time elapsed since the start of the simulation.
    @return: (ndarray) Position vector of the Moon.
    """
    t_norm = t * (2 * np.pi) / P
    return np.array([Lm*np.cos(t_norm), Lm*np.sin(t_norm)])


def rocket_mass(t):
    """
    Calculate the mass of the rocket at a given time.
    @param t: (float) Time elapsed since the start of the simulation.
    @return: (float) Mass of the rocket.
    """
    return Mr0


def density(height):
    """
    Calculate the air density at a given altitude.
    @param height: (float) Altitude above sea level.
    @return: (float) Air density at the given altitude.
    """
    return p0 * np.exp(-height / Hn)


def in_atmosphere(x):
    """
    Calculate if the rocket is in the atmosphere of the Earth.
    @param x: (ndarray) Position vector of the rocket.
    @return: (bool) True if the rocket is in the atmosphere, False oth
erwise.
    """
    return 1 if np.linalg.norm(x) < K else 0
```

Force Functions

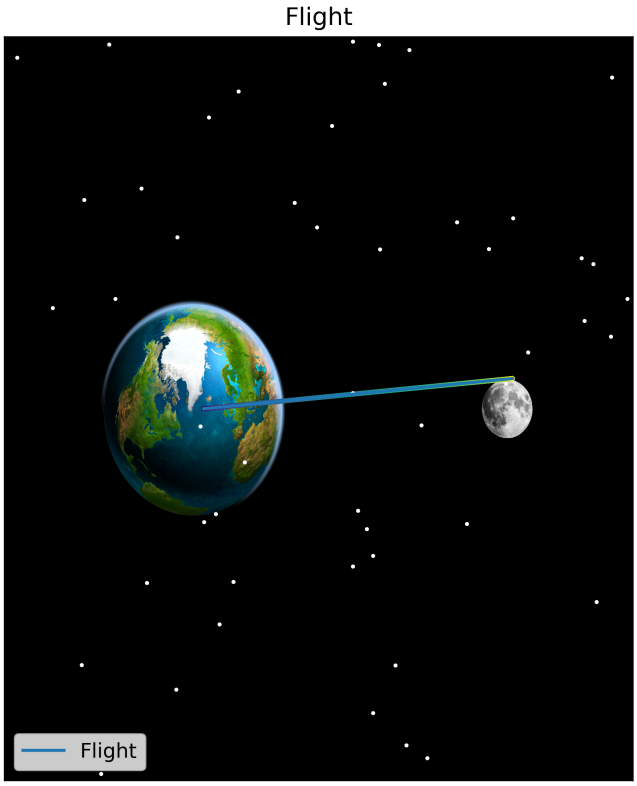
In [ ]:
```
def drag(y, x=True):
    """
    Compute the force due to drag from the atmosphere.
    Returns zero if the rocket has passed the Karman line.

    @param: y (ndarray) - 4D array of floats
                          x position and y position of the rocket in
km,
```

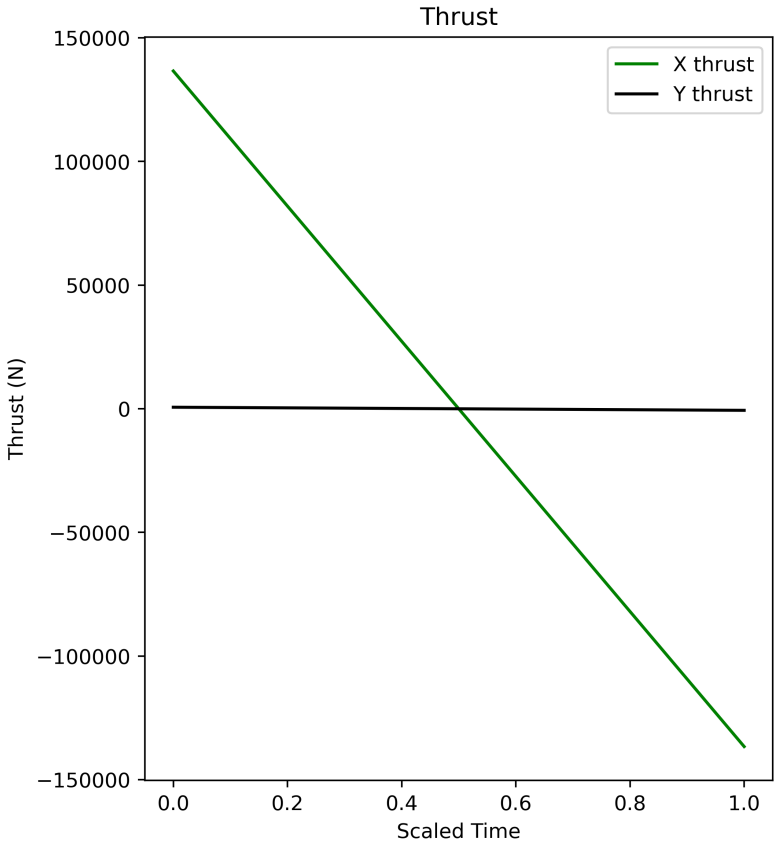
```
                            x velocity and y velocity of the rocket in
km/s
    @param: x (boolean) - True if the force should be returned in the
x direction, false for the y direction

    @returns: force (float) - Force due to drag in the desired directi
on
    """
    pos = y[:2]
    vel = y[2:]
    height = np.linalg.norm(pos - Re)
    if height < K:
        i = 0 if x else 1
        return 0.5 * c * A * density(height) * vel[i]**2 / Mr0
    return 0


def drag_derivative(y, x=True):
    """
    Compute the force due to drag from the atmosphere.
    Returns zero if the rocket has passed the Karman line.

    @param: y (ndarray) - 4D array of floats
                            x position and y position of the rocket in
km,
                            x velocity and y velocity of the rocket in
km/s
    @param: x (boolean) - True if the force should be returned in the
x direction, false for the y direction

    @returns: force (float) - Force due to drag in the desired directi
on
    """
    pos = y[:2]
    vel = y[2:]
    height = np.linalg.norm(pos - Re)
    if height < K:
        i = 0 if x else 1
        return c * A * density(height) * vel[i] / Mr0
    return 0


def gravity(x, x_dir=True, moon=None):
    """
    Calculate the magnitude of the gravitational force acting on the r
ocket.

    @param x: (ndarray) Position vector of the rocket.
    @param x_dir: (bool) True if acceleration in the x direction is de
sired, False for y.
```

```
    @param moon: (ndarray) Position vector of the Moon. Defaults to 0,
indicating Earth's gravity.

    @return: (float) Magnitude of the gravitational force.
    """
    if moon is None: moon = np.array([0, 0])
    mass = Me if np.all(moon == 0) else Mm
    id = 0 if x_dir else 1
    # since we want acceleration and not force, we don't multiply by t
he mass of the rocket
    return G * mass * (moon[id] - x[id]) / ((moon[0] - x[0])**2 + (moo
n[1] - x[1])**2)**(1.5)
```

Previous Solutions

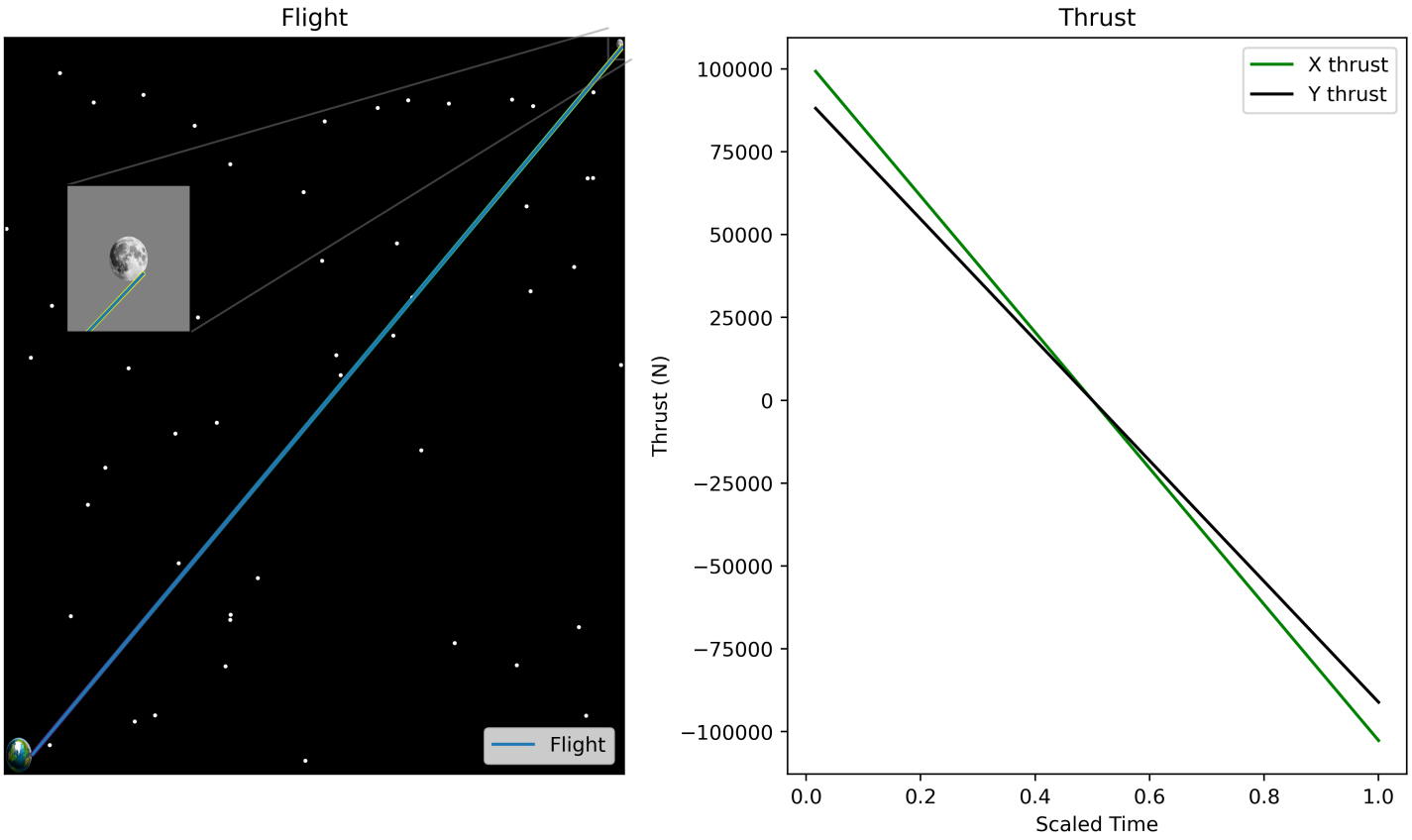
Our initial attempt at this problem included a stationary moon. This assumption reduced the complexity of our evolution equations and still lead to acceptable results. We included the static location of the moon in the numerical solver by providing the angle (θ) between the x-axis and the line between the Earth and the moon. When the moon was at the origin, $(\theta = 0)$, we achieved the following result.





We see that when the moon is stationary and on the x-axis, the rocket takes a straight path to the moon, ending slightly above it in order to enter its orbit. In this visualization, the size of the Earth and moon are not to scale. We increased the size of the Earth and the moon to illustrate the rocket's path.

We also tested our system when the moon was not on the x-axis. When the angle between the x-axis and the line between the Earth and the moon is set to $\theta = 41.212$ degrees, we achieved the following result.



Similar to when the moon was on the x-axis, we see that the rocket takes a straight path to the moon, ending far enough from the moon to enter orbit safely. In this image, the size of the Earth and moon *are* to scale.

Once we were able to achieve satisfactory results with a stationary moon, we removed this simplifying assumption and allowed the moon to move.

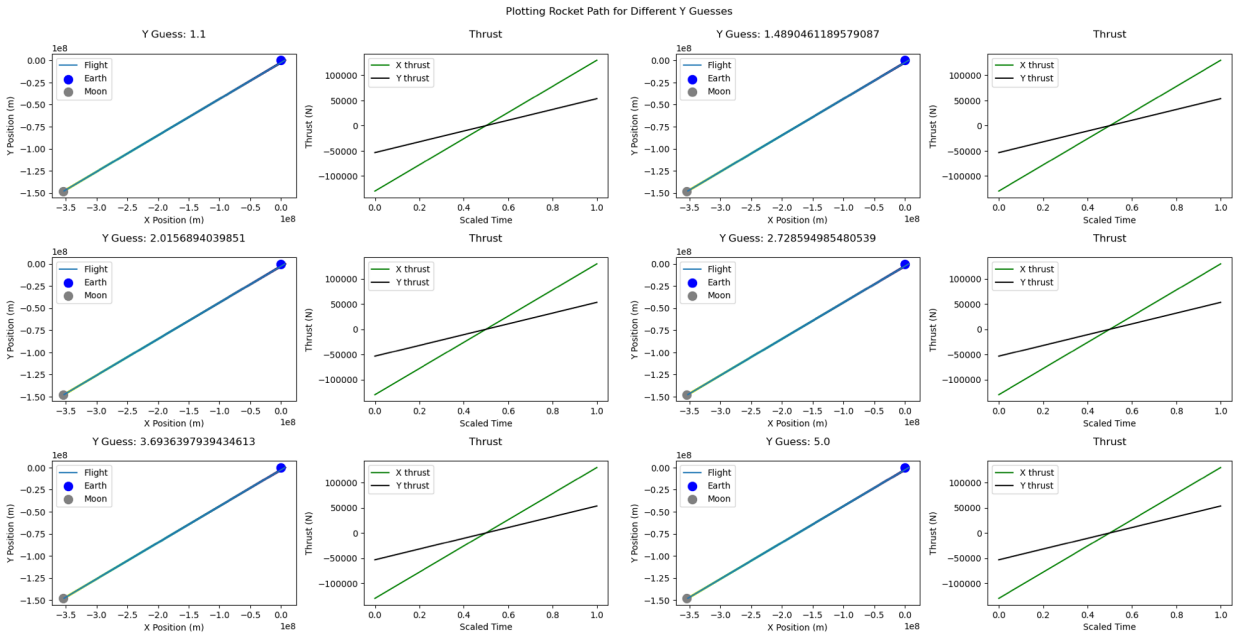
Solution

In [ ]:
```
n = 1000                    # number of time steps to take in the numerica
l solver
dim = 8                     # dimension of the state and costate spaces, c
ombined
final_time = 250000         # number of seconds to arrive at the moon (app
```

```
roximately 3 days)
cost = 1e-16            # cost scalar for the cost functional

# state update equation
def update(t,z,p):
    # rename variables for readability
    tf = p[0]
    pos = z[:2]
    moon_pos = moon_position(tf)
    Lx, Ly = moon_pos
    mass_r = rocket_mass(tf)
    x = z[0]
    y = z[1]
    p1 = z[4]
    p2 = z[5]
    p3 = z[6]
    p4 = z[7]

    # define optimal control based on hamiltonian
    ux = p3 / (mass_r) / cost
    uy = p4 / (mass_r) / cost

    return np.vstack((z[2], # x prime
                      z[3],   # y prime
                      # x double prime
                      (ux / mass_r - gravity(pos) + gravity(pos, moon=mo
on_pos) - in_atmosphere(z[:2])*drag(z)),
                      # y double prime
                      (uy / mass_r - gravity(pos, x_dir=False) - gravity
(x, x_dir=False, moon=moon_pos) - in_atmosphere(z[:2])*drag(z, x=False
)),
                      # p1 prime
                      -(p4 * ((-3*G*Mm*(Lx-x)*(Ly-y)) / ((Lx-x)**2 + (Ly
-y)**2)**(5/2) + 3*G*Me*x*y / (x**2 + y**2)**2.5) + \
                           p3*((3*G*Mm*(Lx-x)**2 / ((Lx-x)**2 + (Ly-y)**2
)**(5/2)) - \
                                (G*Mm / ((Lx-x)**2 + (Ly-y)**2)**(3/2)) +
\
                                    (3*G*Me*x**2 / (x**2 + y**2)**2.5) - \
                                        (G*Me / (x**2 + y**2)**1.5))),
                      # p2 prime
                      -(p3 * ((3*G*Mm*(Lx-x)*(Ly-y)) / ((Lx-x)**2 + (Ly-
y)**2)**5/2 + 3*G*Me*x*y / (x**2 + y**2)**2.5) + \
                           p4*((-3*G*Mm*(Ly-y)**2 / ((Lx-x)**2 + (Ly-y)**
2)**(5/2)) + \
                                (G*Mm / (((Lx-x)**2 + (Ly-y)**2)**(3/2)))
+ \
                                    (3*G*Me*y**2 / (x**2 + y**2)**2.5) - \
                                        (G*Me / (x**2 + y**2)**1.5))),
                      # p3 prime
```

```
                    -(p1 - drag_derivative(z[:4])),
                    # p4 prime
                    -(p2 - drag_derivative(z[:4], x=False))
                    ))

# set up the endpoint conditions
def bc(ya, yb, p):
    tf = p[0]
    moon_pos = moon_position(tf)
    theta = np.arctan2(moon_pos[0], moon_pos[1]) + 90
    return np.array([ya[0]-Re,       # start x
                     ya[1]-0,        # start y
                     ya[2]-0,        # start x'
                     ya[3]-we,       # start y'
                     yb[0]-moon_pos[0] - (Rm-hf)*np.cos(theta),     #
end x
                     yb[1]-moon_pos[1] - (Rm-hf)*np.sin(theta),     #
end y
                     yb[2]-vf*np.cos(theta),                        #
end x'
                     yb[3]-vf*np.sin(theta),                        #
end y'
                     (tf-final_time)])
```

In [ ]:
```
# Plot space as defined in the appendix
plot_space()
```



Interpretation

We set the moon to always start on the x-axis and move in a circular orbit around the Earth at a normal orbital speed. When the rocket is launched, we fix its arrival time. This condition allows us to calculate the final x and y position so that the rocket enters the moon's orbit after approximately three days. This travel time is about the time it takes to reach the moon at a safe, yet optimal, speed.

Additionally, now that the moon is moving, the affect of the moon's gravity on the rocket's path is more variable. We accounted for this difference and updated the conditions in the `update()` equation in the **Solution** section appropriately.

To test the robustness of our system, we ran six simulations with varying initial conditions with the results of these simulations above. For more information about how we ran and plotted our simulations, see the `plot_space()` function in the Appendix.

Our project performed well in finding an optimal flight path to the moon in the simple case of a fixed end time and fixed mass. However, when we optimized over the final endtime, the rocket could not get to the moon. Furthermore, while our rocket does arrive to the moon's orbit, it is not representative of a real life space journey, which resembles a bang-bang solution.

Future Work

Throughout this project we made several simplifying assumptions. From these assumptions, we achieved satisfactory results and learned more about the complicated dynamics of space travel as we can see in the images above. Despite these results, we feel that the project is not yet at its full potential and could be improved in several ways.

One of the first improvements is implementing variable fuel consumption, thus adding variable weight to the rocket. This improvement would change the rocket's optimal solution and make the problem more realistic.

A second improvement is making the rocket's limitations more realistic. In typical rocket launches, the rocket has at least two phases: a phase inside the Earth's atmosphere and a phase outside the Earth's atmosphere. This design involves a drastic change in the rocket's weight and maximum thrust, which would be a significant change to our model that would allow for a more accurate reflection of the dynamics of a real rocket launch.

The final improvement we would make is optimizing our system over final time. Including final time in the cost function would allow us to achieve a solution that is not only optimal over thrust, but also over time. Finding the optimal time to reach the moon would make the results of the system much more interesting and would potentially lead to new insights into the dynamics of space travel.

Ferpa Release

We give Dr. Evans, Dr. Whitehead, and other instructors at BYU teaching ACME classes permission to share our project as an example of a good project in future classes they teach.

-Jason Vasquez, Dallin Stewart, Dylan Skinner, Ethan Crawford

Appendix

Plotting Functions

In [ ]:
```
def plot_space():
    n = 1000
    dim = 8
    r1, r2 = 6, 3
    count = 0
    plt.figure(figsize=(20, 30))
    y_guess = 1.1
    for i, y_guess in enumerate(np.geomspace(1.1, 5, num=r1)):
        final_time = 250000
        moon_pos = moon_position(final_time)
        moon_x, moon_y = moon_pos[0], moon_pos[1]

        t = np.linspace(0, 1, n)
        y = np.array([np.linspace(Re, moon_x, n), np.linspace(0, moon_
y, n) * y_guess, np.ones(n) * y_guess,
                      np.ones(n) * y_guess, np.ones(n) * y_guess, np.one
s(n) * y_guess, np.ones(n) * y_guess,
                      np.ones(n) * y_guess])
        # y = np.ones((dim, t.size))
        p0 = np.array([final_time])
        res = solve_bvp(update, bc, t, y, p=p0, max_nodes=50000)

        # plot the results and print out the optimal final time t_f
        t_plot = np.linspace(0, 1, n)
        # print(res.sol(t_plot).shape)
        x_plot = res.sol(t_plot)[0]
        y_plot = res.sol(t_plot)[1]
        px_plot = res.sol(t_plot)[6]
        py_plot = res.sol(t_plot)[7]
        ux_plot = px_plot
```

```
        uy_plot = py_plot

        plt.subplot((r2 * r1) // 2, 4, 1 + count)
        plt.plot((x_plot), y_plot, label="Flight")  # Normalize x and
y coordinates
        plt.scatter((x_plot), y_plot, cmap="viridis", c=t_plot, s=2)
        plt.scatter(0, 0, c='blue', s=100, label="Earth")
        plt.scatter(moon_x, moon_y, c='gray', s=100, label="Moon")
        plt.xlabel("X Position (m)")
        plt.ylabel("Y Position (m)")
        plt.legend()
        plt.title(f"Y Guess: {y_guess}\n")

        plt.subplot((r2 * r1) // 2, 4, 2 + count)
        plt.plot(t_plot, ux_plot, label="X thrust", color='green')
        plt.plot(t_plot, uy_plot, label="Y thrust", color='black')
        plt.xlabel("Scaled Time")
        plt.ylabel('Thrust (N)')
        plt.title(f"Thrust\n")
        plt.legend()

        count += 2
        if count >= ((r2 * r1) // 2) * 4:
            break

    plt.suptitle('Plotting Rocket Path for Different Y Guesses\n\n\n')
    plt.tight_layout()
    plt.show()
```