flight_sim_final

May 3, 2022

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import math, sympy
import os
from dotenv import load_dotenv

load_dotenv()

empty_rocket_mass_css = int(os.getenv("ROCKET_MASS")) - int(os.
 ↪getenv("PROP_MASS"))
propellant_mass_css = int(os.getenv("PROP_MASS"))
rocket_length_css = float(os.getenv("ROCKET_LENGTH"))
diameter_css = float(os.getenv("DIAMETER"))


class Propellants:
    def __init__(self, name, isp, density, cost, exhaust_velocity):
        self.name = name
        self.isp = isp # s
        self.density = density # g/cc
        self.cost = cost # $/kg
        self.velocity = exhaust_velocity # m/s

    def set_burn_rate(self, diameter):
        nozzle_area = math.pi * (1/300*diameter*100)**2 # m^2
        self.burn_rate = nozzle_area*self.velocity*100*self.density/1000 # kg/s


class Materials:
    def __init__(self, name, density, cost):
        self.name = name
        self.density = density # g/cc
        self.cost = cost # $/kg

UDMH = Propellants('UDMH', 333, 1.18, 1.00, 1720)
LOX_RP1 = Propellants('LOX/RP-1', 353, 1.02, 1.17, 1805)
LOX_LH2 = Propellants('LOX/LH2', 451, 0.28, 1.09, 1435)
```

```
titanium = Materials('Titanium', 4.5, 14.90)
aluminum = Materials('Aluminum', 2.7, 3.05)

class Rocket:
    def __init__(self, name, rocket_length, diameter, material, propellant,␣
↪empty_rocket_mass = 0, propellant_mass=0):
        self.name = name
        self.propellant = propellant
        self.material = material
        self.rocket_length = rocket_length
        self.diameter = diameter
        self.fin_height = 1.25*self.diameter
        fin_a = 1/6*rocket_length
        fin_b = 1/6*rocket_length*0.45
        self.fin_volume = 0.01*self.diameter*(fin_a+fin_b)/2*1.05*self.diameter
        self.body_volume = (math.pi*(1/2*self.diameter)**2*0.85*self.
↪rocket_length) - (math.pi*(1/2*self.diameter)**2*0.85*self.rocket_length*0.
↪99)
        self.nose_cone_volume = (1/3*math.pi*(1/2*self.diameter)**2*0.15*self.
↪rocket_length) - (1/3*math.pi*(1/2*self.diameter)**2*0.15*self.
↪rocket_length*0.99)
        self.fin_mass = self.fin_volume * 1000 * material.density
        self.body_mass = self.body_volume * 1000 * material.density
        self.nose_cone_mass = self.nose_cone_volume * 1000 * material.density +␣
↪3000
        if propellant_mass != 0:
            self.propellant_mass = propellant_mass
        else:
            self.propellant_mass = (math.pi*(1/2*self.diameter)**2*0.85*self.
↪rocket_length*0.99)*self.propellant.density/1000
        self.total_mass = self.fin_mass*2 + self.body_mass + self.
↪nose_cone_mass + self.propellant_mass
        if empty_rocket_mass != 0:
            self.empty_rocket_mass = empty_rocket_mass
        else:
            self.empty_rocket_mass = self.fin_mass*2 + self.body_mass + self.
↪nose_cone_mass
        propellant.set_burn_rate(diameter)

        propellant.burn_time = self.propellant_mass / propellant.burn_rate
        self.propellant_mass_ratio = self.propellant_mass / self.
↪empty_rocket_mass
        propellant_cost = (math.pi*(1/2*self.diameter)**2*0.85*self.
↪rocket_length*0.99)*self.propellant.density/1000*self.propellant.cost
```

```
        rocket_cost = (self.body_mass+self.fin_mass*2+self.nose_cone_mass)*self.
↪material.cost
        cost = propellant_cost + rocket_cost + 15**6
        self.cost = cost

        print('--- ROCKET SPECS ---')

        print(f'Name: {self.name}')
        print(f'Rocket Length: {self.rocket_length} m')
        print(f'Rocket Diameter: {self.diameter} m')
        print(f'Empty Rocket Mass: {self.empty_rocket_mass} kg')
        print(f'Rocket Cost: ${self.cost:0.2f}')
        print(f'Propellant Used: {propellant.name}')
        print(f'Propellant Mass: {propellant_mass} kg')
        print(f'Propellant Burn Rate: {propellant.burn_rate} kg/s')
        print(f'Burn Time: {propellant.burn_time} s')

        self.velocity = np.array([[0, 0]])
        self.acceleration = np.array([[0, 0]])
        self.angle = 0

        # Center of Gravity Calculations
        self.cog_nose_cone = np.array([[1/2*self.diameter, 0.15*self.
↪rocket_length*1/4 + 0.85*self.rocket_length]])
        self.cog_body = np.array([[1/2*self.diameter, 0.85*self.rocket_length*1/
↪2]])
        self.cog_fin1 = np.array([[-1/3*(self.fin_height*(fin_b+2*fin_a))/
↪(fin_b+fin_a), 1/2*1.25*self.diameter]])
        self.cog_fin2 = np.array([[self.diameter+1/3*(self.
↪fin_height*(fin_b+2*fin_a))/(fin_b+fin_a), 1/2*1.25*self.diameter]])

        self.cog_x = (self.cog_fin1[0][0]*self.fin_mass + self.
↪cog_fin2[0][0]*self.fin_mass + self.cog_body[0][0]*(self.body_mass+self.
↪propellant_mass) + self.cog_nose_cone[0][0]*self.nose_cone_mass) / self.
↪total_mass
        self.cog_y = (self.cog_fin1[0][1]*self.fin_mass + self.
↪cog_fin2[0][1]*self.fin_mass + self.cog_body[0][1]*(self.body_mass+self.
↪propellant_mass) + self.cog_nose_cone[0][1]*self.nose_cone_mass) / self.
↪total_mass
        self.center_of_gravity = np.array([[ self.cog_x, self.cog_y]])
        self.position = self.center_of_gravity
        self.rocket_reference_pos = np.array([[1/2*diameter, 0]])
        print(f'Center of Gravity: ({self.center_of_gravity[0][0]} m, {self.
↪center_of_gravity[0][1]} m)')
```

```
        self.cog_vector = np.array([[self.rocket_reference_pos[0][0] , self.
↪center_of_gravity[0][0]], [self.rocket_reference_pos[0][1], self.
↪center_of_gravity[0][1]]])
        print('-'*20)
        print(' ')

    def set_angle(self, angle):
        self.angle = angle

    def set_altitude(self, altitude):
        self.altitude = altitude

def atmospheric_density(altitude):
    '''
    This function finds the atmospheric density in kg/m^3 at a certain altitude.
    Parameters:
        - Altitude = Altitude in m
    '''
    if altitude < 11000:
        temperature = 15.04 - 0.00649*altitude
        pressure = 101.29 * ((temperature+273.15)/288.08)**5.256
    elif altitude < 25000:
        temperature = -56.46
        pressure = 22.65 * math.exp(1.73 - 0.000157*altitude)
    else:
        temperature = -131.21 + 0.00299 * altitude
        pressure = 2.488 * ((temperature + 273.15)/216.6)**(-11.388)
    rho = pressure / (0.2869 * (temperature + 273.15))
    return rho

def gravitational_constant(altitude):
    '''
    This function finds the gravitational constant g0 in m/s^2 at a certain␣
↪altitude.
    Parameters:
        - Altitude = Altitude in m
    '''
    G = 6.673*10**(-11)
    RE = 6.37*10**6
    ME = 5.98*10**24
    g0 = G*ME/((altitude+RE)**2)
    return g0


def prepare_launch(rocket, altitude = 0, angle=30):
    '''
```

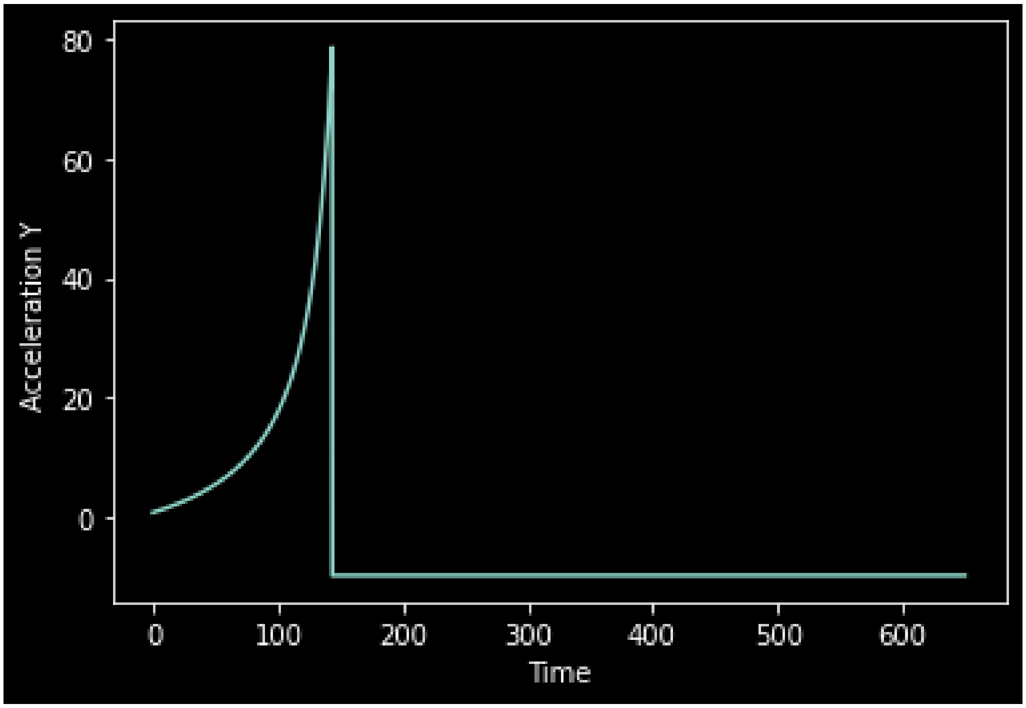
```python
    This function prepares the launch by calculating the initial conditions of␣
↪the rocket.
    Parameters:
        - Altitude = Altitude of launch location in m
    '''

    print('--- LAUNCH PREP ---')

    theta = np.deg2rad(angle)
    rocket.set_angle(theta)
    rocket.set_altitude(altitude)
    print(f'Launch Angle: {theta} rad')
    rho = atmospheric_density(altitude)
    print(f'Atmospheric Density at Launch: {rho} kg/m^3')
    g = gravitational_constant(altitude)
    print(f'Gravitational Constant at Launch: {g} m/s^2')
    print('-'*20)
    return rho,g

def thrust(propellant, time):
    '''
    This function returns the magnitude of the thrust force for the rocket in N.
    Parameters:
        - Propellant = Type of propellant (str of 'UDMH', 'LOX_RP1', or␣
↪'LOX_LH2')
        - Time = Time from takeoff in s
    '''
    if time < propellant.burn_time:
        thrust = propellant.burn_rate*propellant.velocity
    else:
        thrust = 0
    return thrust

def drag(rocket, velocity, rho):
    '''
    This function returns the magnitude of the drag force for the rocket in N.
    Parameters:
        - Rocket = Rocket being simulated
        - Velocity = Velocity vector of rocket at instant in m/s [vx, vy]
        - Rho = Atmospheric density at instant in kg/m^3
    '''
    area = math.pi * (1/2*rocket.diameter)**2
    cd = 0.295
    drag_x = -1/2*cd*rho*area*velocity[0][0]**2
    drag_y = -1/2*cd*rho*area*velocity[0][1]**2
    drag = np.array([[drag_x, drag_y]])
    return drag
```

```python
def lift(rocket, velocity, rho):
    '''
    This function returns the magnitude of the lift force for the rocket in N.
    Parameters:
        - Rocket = Rocket being simulated
        - Velocity = Velocity vector of rocket at instant in m/s [vx, vy]
        - Rho = Atmospheric density at instant in kg/m^3
    '''
    area = math.pi * (1/2*rocket.diameter)**2
    cl = 0.5
    lift_x = 1/2*cl*rho*area*velocity[0][0]**2
    lift_y = 1/2*cl*rho*area*velocity[0][1]**2
    lift = np.array([[lift_x, lift_y]])
    lift = np.linalg.norm(lift)
    return lift

def weight(rocket, g0):
    '''
    This function returns the magnitude of the thrust force for the rocket in N.
    Parameters:
        - Rocket = Rocket being simulated
        - G0 = Gravitational constant at instant in m/s^2
    '''
    weight = rocket.total_mass * g0
    return weight

def force(rocket, propellant, altitude, time):
    '''
    This function finds the summation of the x and y components of force and␣
↪returns force vector.
    Parameters:
        - Rocket = Rocket being simulated
        - Propellant = Type of propellant (str of 'UDMH', 'LOX_RP1', or␣
↪'LOX_LH2')
        - Altitude = Altitude at current instant in m
    '''
    rho = atmospheric_density(altitude)
    g = gravitational_constant(altitude)
    thrust_magnitude = thrust(propellant, time)
    drag_magnitude = drag(rocket, rocket.velocity, rho)
    lift_magnitude = lift(rocket, rocket.velocity, rho)
    weight_magnitude = weight(rocket, g)
    return thrust_magnitude, drag_magnitude, lift_magnitude, weight_magnitude

def efficiency(distance, time, rocket):
    e = distance/time * (1-rocket.cost)
```

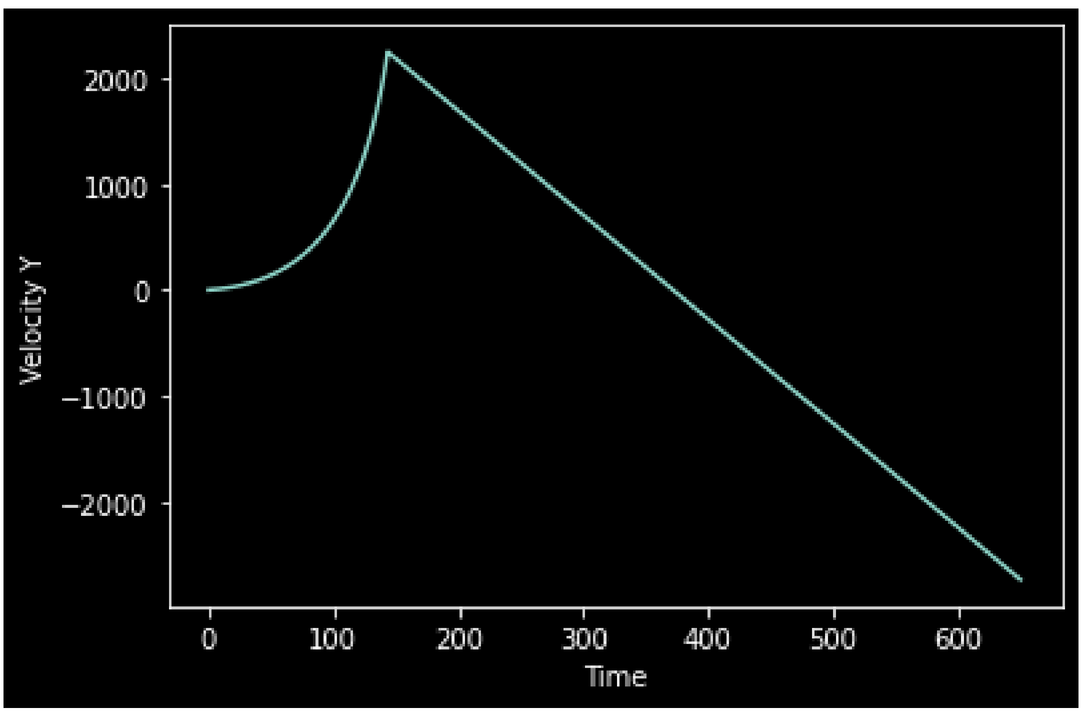
```python
    return e

def launch(rocket):
    averageThrust = rocket.propellant.burn_rate*rocket.propellant.velocity
    massFlowRate = rocket.propellant_mass/rocket.propellant.burn_time

    time = np.linspace(0, 650, 1000000, False)
    i = 0
    while time[i] <= rocket.propellant.burn_time:
        i += 1
    index = i
    thrust = np.append(np.repeat(averageThrust, index), np.repeat(0, len(time)␣
 ↪- index))

    mass = np.append(np.repeat(rocket.total_mass, index) - time[0:index] *␣
 ↪massFlowRate, np.repeat(rocket.empty_rocket_mass, len(time) - index))
    acceleration_y = thrust/mass - gravitational_constant(rocket.altitude)
    acceleration_x = thrust*math.cos(rocket.angle)/mass

    print(f'--- Results ({rocket.name}) ---')

    plt.style.use('dark_background')
    plt.plot(time, acceleration_y)
    plt.ylabel("Acceleration Y")
    plt.xlabel("Time")
    plt.show()

    def integrateGraph(time, array):
        resArray = [0]
        for n in range(0, len(time)-1):
            resArray.append(
                resArray[-1] + 0.5*(array[n+1] + array[n])*(time[n+1] -
                time[n])
            )
        return np.array(resArray)

    velocity_y = integrateGraph(time, acceleration_y)
    velocity_x = integrateGraph(time, acceleration_x)
    plt.plot(time, velocity_y)
    plt.ylabel("Velocity Y")
    plt.xlabel("Time")
    plt.show()

    displacement_y = integrateGraph(time, velocity_y) + rocket.altitude
    displacement_x = integrateGraph(time, velocity_x)/100
    plt.plot(time, displacement_y)
    plt.ylabel("Displacement Y")
```

```
    plt.xlabel("Time")
    plt.title(rocket.name)
    plt.show()
    for i in range(len(displacement_y)):
        if displacement_y[i] < 0:
            print(f'Rocket Range: {displacement_x[i]:0.2f} m\nFlight Time:␣
 ↪{time[i]:0.2f} s')
            e = efficiency(displacement_x[i], time[i], rocket)
            print(f'Efficiency Score: {e:0.3E}')
            print('-'*20)
            break


myrocket = Rocket('UDMH, Aluminum', 22.7, 1.95, aluminum, UDMH, 47200-38600,␣
 ↪38600)
rho,g = prepare_launch(myrocket, 2000, 30)
launch(myrocket)
```

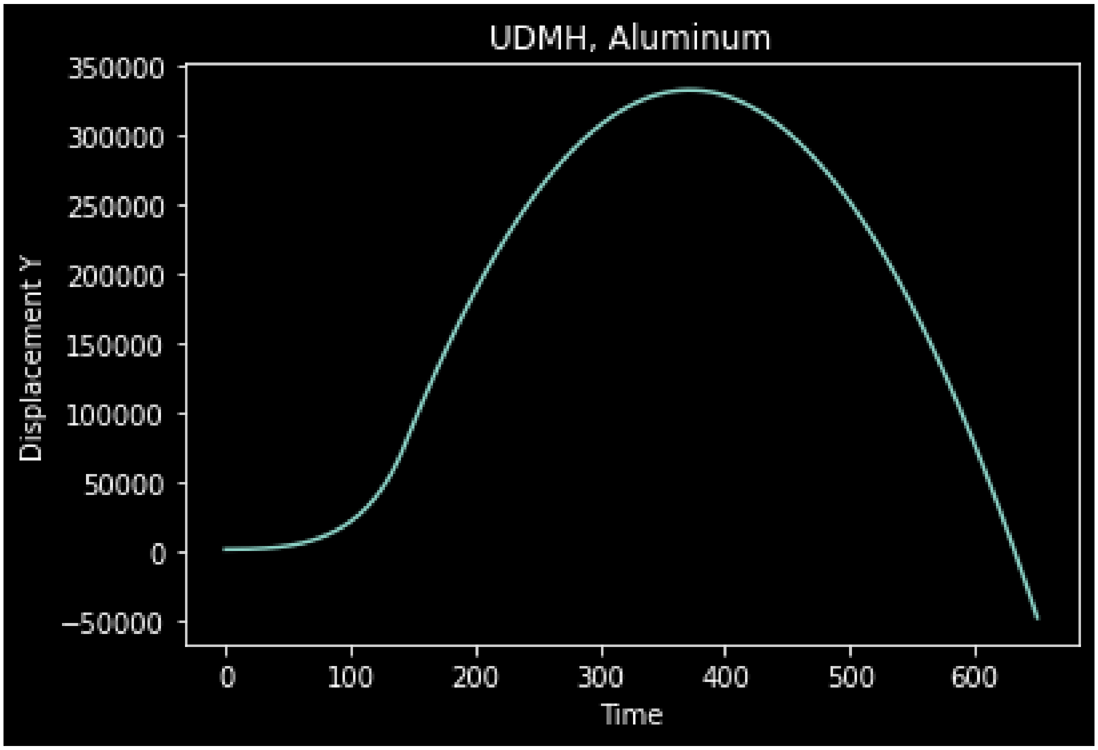
```
--- ROCKET SPECS ---
Name: UDMH, Aluminum
Rocket Length: 22.7 m
Rocket Diameter: 1.95 m
Empty Rocket Mass: 8600 kg
Rocket Cost: $11406603.25
Propellant Used: UDMH
Propellant Mass: 38600 kg
Propellant Burn Rate: 269.3934550009169 kg/s
Burn Time: 143.28484706456072 s
Center of Gravity: (0.9749999999999999 m, 10.274172263108651 m)
--------------------

--- LAUNCH PREP ---
Launch Angle: 0.5235987755982988 rad
Atmospheric Density at Launch: 1.0088950807649093 kg/m^3
Gravitational Constant at Launch: 9.828127333606815 m/s^2
--------------------
--- Results (UDMH, Aluminum) ---
```



80
60
40
20
0
Acceleration Y
0
100
200
300
400
500
600
Time



2000
1000
0
−1000
−2000
Velocity Y
0
100
200
300
400
500
600
Time




```
Rocket Range: 16989.07 m
Flight Time: 632.15 s
Efficiency Score: -3.066E+08
--------------------
```