Numerical Rocket Trajectory Simulation for a Three-Stage PegasusXL Launch Vehicle

Andrew R. Smith[1]
University of Delaware, Newark, Delaware, 19716, U.S.

Joshua R. Ginsberg[2]
University of Delaware, Newark, Delaware, 19716, U.S.

Jasmine M. Middaugh[3]
University of Delaware, Newark, Delaware, 19716, U.S.

This paper explores a numerical approach to simulate the trajectory of the Pegasus XL, a three-stage launch vehicle by Northrop Grumman. The goal is to calculate the highest altitude attainable with a 3000 kg payload. By using a one-dimensional model, the simulation addresses the varying effects of gravitational and drag forces as the vehicle ascends. The stages of the Pegasus XL are powered by Orion 50S XL, Orion 50 XL, and Orion 38 motors. Euler's method is utilized to solve the vehicle dynamics iteratively, assuming constant thrust and mass flow rates for each stage. The study examines different stage mass fractions to identify the configuration that maximizes altitude. Findings reveal that the rocket can achieve a maximum altitude of 1,145 km, with stage mass distribution playing a critical role in performance. This research highlights the significance of effective mass management in multi-stage rockets and provides valuable insights for suborbital mission optimization. Future improvements include incorporating dynamic thrust models and refined atmospheric conditions to enhance simulation accuracy.

I. Nomenclature

A	= maximum projected area of rocket
a	= net acceleration
C_D	= non-dimensional vehicle drag coefficient
F_D	= drag force
F_g	= gravitational force (weight)
F_{net}	= net vertical force acting on vehicle
f	= arbitrary function designation
g	= gravitational acceleration
g_0	= mean surface gravitational acceleration
h	= altitude
M	= external Mach number
MR	= overall vehicle mass fraction
MR_n	= stage mass fraction
m	= total rocket mass
$m_{f,n}$	= burnout mass of the *n*-th rocket stage

[1] Mechanical Engineering B.S. student, Department of Mechanical Engineering
[2] Mechanical Engineering B.S. student, Department of Mechanical Engineering
[3] Mechanical Engineering B.S. student, Department of Mechanical Engineering

$m_{p,n}$	= propellant mass of the *n*-th rocket stage
m_n	= total mass of the *n*-th rocket stage
m_0	= overall initial mass of the rocket
$m_{0,n}$	= initial mass of the *n*-th rocket stage
n	= rocket stage
R	= specific gas constant of air
R_e	= volumetric mean radius of the Earth
t	= time
$t_{b,n}$	= burnout time of the *n*-th rocket stage motor
T	= thrust force
T_a	= atmospheric air temperature
v	= velocity
γ	= heat capacity ratio of air (isentropic expansion factor)
Δt	= iterative timestep
ρ	= atmospheric air density

II. Introduction

This project provides an opportunity to dive deeper into the mechanics of multi-stage vehicles, as well as time for further research into rocket motors and launch vehicles. To start, multi-stage launch vehicles are essential to the performance of rockets on orbital missions to achieve the high required speed. The main reason why multi-stage rockets are required for these types of missions is due to the limitations that the laws of physics place on the maximum achievable velocity of rockets for a given fueled-to-dry mass ratio. A single rocket would not be able to achieve this because its wet-to-dry mass ratio would fall too low. With a multi-stage rocket, as each stage is released the rocket is still traveling near its burnout speed due to the discard of useless dry mass, allowing the rocket to keep up speed. Additionally, each stage of the vehicle can be designed to work and perform at each operating condition and varying atmospheric levels. Understanding how to design these types of vehicles can either make or break an orbital mission.

The general shape of a launch vehicle trajectory can be seen in Fig. 1, where a two stage rocket is shown. Trajectories typically take the shape of a parabola and can have varying lengths and sizes due to its parameters. In the initial launch phase, the thrust produced by the engine is greater than the weight of the rocket, so the net force accelerates the rocket away from the pad in a vertical direction [1]. The rocket will begin its ascent until it reaches its burnout height where all the propellant has been utilized. This phase known as coasting flight is where the rocket is no longer producing any thrust, so the only forces acting on it are the drag and weight. The rocket will then continue to accelerate at a gradually decreasing rate until it reaches a constant velocity or orbital velocity. As seen in Fig. 1, with the addition of stages it’s clear that rockets can reach higher altitudes and perform better depending on their parameters.

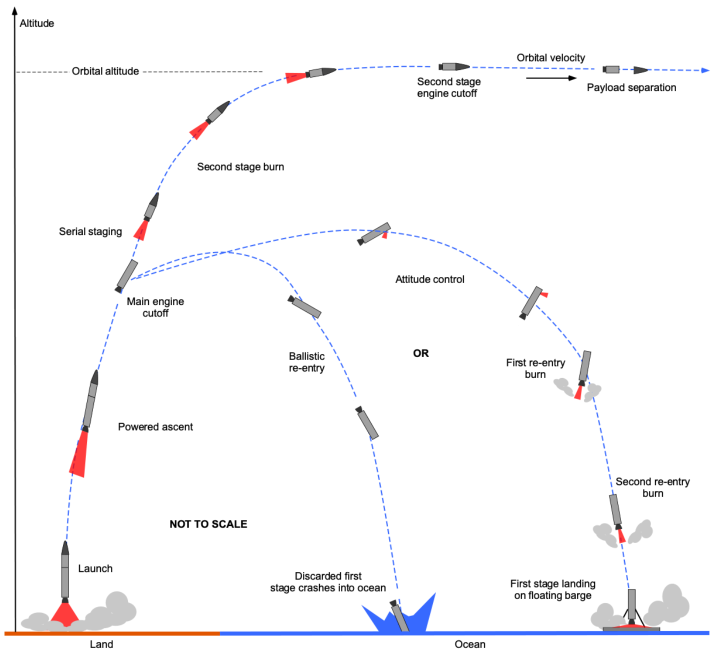


Fig. 1 Example of a two stage launch vehicle trajectory.

While a multitude of launch vehicles are utilized in the field, the project focuses on the Pegasus XL by Northrop Grumman. The Pegasus was first launched in 1990, becoming the world's first privately developed space launch vehicle. This rocket is a category 3 NASA-certified vehicle, used to deploy small satellites into low Earth orbit. The rocket in the case of this simulation is launched from the ground by the Minotaur-C, also developed by Northrop Grumman. The Pegasus is the first winged vehicle to accelerate eight times the speed of sound, as well as the first to place satellites into orbit. The rocket has earned its reputation as the world's standard for its affordability and reliability, and it typically delivers its satellites within 10 minutes. Pegasus has conducted a total of 45 missions and launched almost 100 satellites into orbit [2].

III. Project Description

The team assigned to this project represents a small government services company receiving a Northrop Grumman Pegasus XL Launch Vehicle. The rocket is to be used as a sounding rocket to launch a 3,000 kg payload in a vertical trajectory. The task is to calculate the maximum altitude of this three-stage sounding rocket with the required payload size. To accomplish this task, the team is required to develop a 1-D trajectory code that considers gravity and drag, taking into account the changing atmosphere as the vehicle ascends.

There are a variety of assumptions that have been laid out for the project, with additional assumptions made in terms of the trajectory code described later in the report. The Pegasus XL is to be utilized with no wings and fins, with the first stage motor being the Orion 50S XL, the second stage being the Orion 50 XL, and the third stage being the Orion 38. The simulation may additionally assume the payload fairing is included in the 3,000 kg payload mass in order to simplify calculations. The Pegasus XL configuration can be seen in Fig. 2, with each component labeled. The vehicle incorporates eight main elements, consisting of three solid rocket motors, a payload fairing, an avionics assembly, a lifting wing, an aft skirt assembly including three movable control fins, and a payload interface system. As stated previously, the fins and wing will be neglected in this analysis.

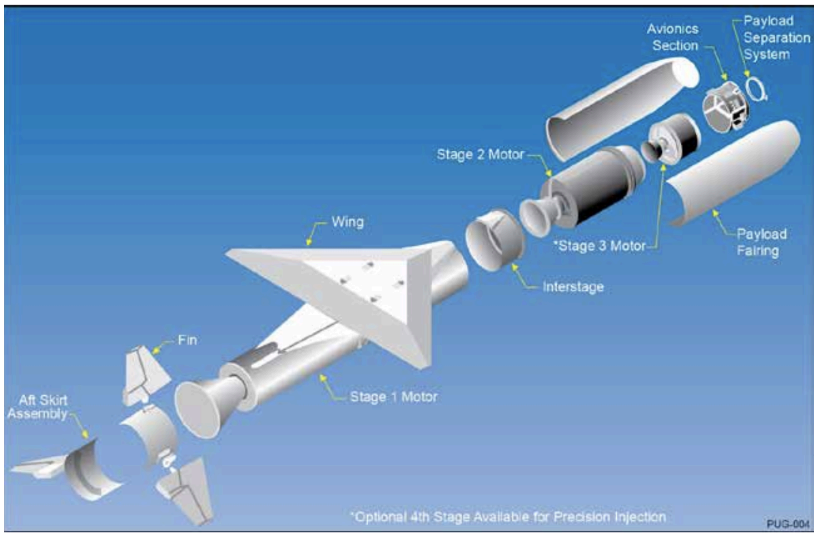


Fig. 2 Pegasus XL exploded assembly.

The Orion 50S XLis an air-ignited motor with a fixed nozzle–refer to the appendix for more information. It has a better performance compared to the Orion 50S and is 55.4 inches longer with 6,500 lbm more propellant than its predecessor. The motor has a burn time of 69.1 seconds and an average chamber pressure of 1,073 psia. This motor has successfully performed in multiple missions on the 25 Pegasus XL to date for its first stage. The Orion 50 XL is an air-ignited motor with a vectorable nozzle. This motor is 18 inches longer and contains 2,000 lbm more propellant than the Orion 50. This motor has a burn time of 69.7 seconds and an average chamber pressure of 991 psia. This motor has been seen on the Pegasus XL for the second stage and the Minotaur for the third stage. The Orion 38 is an upper-staged booster motor that is air-ignited and has a vectorizable nozzle. Orion 38 was developed specifically for the Pegasus launch vehicle as a third stage, low cost, high performance motor. Its vectorable nozzle is designed at a ± 5 degree angle, and the motor itself has a burn time of 67.7 seconds and an average chamber pressure of 572 psia. The Orion 38 functions typically as the third stage motor on the Pegasus XL and Taurus series, as well as the fourth stage for the Air Force's Minotaur vehicle [3].

IV. Methodology

The objective of this simulation is to compute the maximum altitude achieved for a three-stage rocket subject to predefined performance metrics and variable stage mass fractions. The maximum altitude is to be found through an implicit analysis as the expected one-dimensional vehicle dynamics will be nonlinear given the changing mass properties, atmospheric conditions, gravitational effects, and overall motor performances which all vary with time and/or by stage. Because the vehicle trajectory simulation will also need to be performed for each stage mass fraction combination desired, Euler's method is employed for a fast, iterative solution with linear runtime complexity[4]. The main iterative portion of the simulation using Euler's method can thus be defined using Eqs. (1-5).

$$A = f(t) \tag{1}$$

$$F_{net}(t, h, v, A) = T(t) - F_g(t, h) - F_D(h, v, A) \tag{2}$$

[4] This first-order solution will converge to a maximum altitude with a runtime complexity of O(n) as the number of steps in each iteration correspond to the timesteps of the simulation. The "n" here refers to the product of the number of stage mass fractions considered for each stage and the total number of time steps for each stage mass fraction.

$$a(t, h, v, A) = \frac{F_{net}(t,h,v,A)}{m(t)} \tag{3}$$

$$\frac{\delta v}{\delta t} = a(t, h, v, A) \tag{4}$$

$$\frac{\delta h}{\delta t} = v(t, h, v, A) \tag{5}$$

From Eqs. (1-5), the general procedure for calculating the height at each time step is also outlined. The net force on the rocket is calculated at each time step using the vertical thrust force of the current stage; the gravitational force, varying as a function of time given the changing mass of the vehicle and the gravitational acceleration varying with altitude; and the drag force, varying with both the projected area of the rocket, the atmospheric conditions, and the Mach number of the vehicle, see Eq. (2). The acceleration is then found using this net force and the current mass of the vehicle, see Eq. (3). Euler's method is then used to compute the remaining two variables: the velocity can be integrated at each timestep based on the vehicle acceleration in the last timestep, and the altitude can similarly be integrated once the velocity is found, see Eqs. (4) and (5).

A few assumptions are made in order to compute the forces on the rocket at each timestep and thus the overall vehicle dynamics. The thrust force is assumed to be constant using the burn time average thrust of the motor at the current rocket stage–the burn time, burn time average thrust, propellent mass, total loaded mass, burnout mass, and diameter for each motor are gathered from the ATK Space Propulsion Products Catalog [3]. In order to determine the current stage, it is also assumed that once stage *n* burns out, stage *n+1* immediately begins firing at the aforementioned constant thrust value and thus, the burnout times are used to compute the staging of the rocket. The gravitational force is computed using the gravitational acceleration at the current altitude, assumed to be governed by the inverse-square law for the one-dimensional simulation.

$$g(h) = g_0\left(\frac{R_e}{R_e+h}\right)^2 \tag{6}$$

Here, g_0 is taken to be 9.820 m/s^2 and R_e as 6371.00 km from the NASA Space Science Data Coordinated Archive [4]. The drag force can be calculated based on the atmospheric conditions present at the current altitude as well as the vehicle speed.

$$F_D(h, v, A) = \frac{1}{2}\rho v^2 A C_d \tag{7}$$

$$M(v, h) = \frac{v}{\sqrt{\gamma R T_a}} \tag{8}$$

The drag coefficient can be approximated using data from a German V-2 missile at no angle-of-attack which ranges from C_d = 0.147 - 0.416 [5]–while this is a rough approximation, it highlights the relationship expected between the vehicle Mach number and drag, see Eq. (7). For this calculation of the drag force, there are then two atmospheric conditions which need to be approximated: the air density and temperature versus altitude. These values are computed from standard, tabulated data where linear interpolation is used for all intermediate altitude values [6], refer to the appendix for more information.

With all the forces present on the vehicle described before, the last calculation to be completed involves finding the mass of the vehicle at each time step in order to compute the net vertical acceleration. With the constant vehicle thrust assumption from before, as well as an assumed negligible component of back pressure on the vehicle thrust, the mass flow rate of each rocket motor is constant over the burn time. The data from the motor catalog can then be used to predict this mass flow rate from the propellent mass and burnout time, see Eq. (9). The mass of the rocket is then computed using the initial, loaded mass (including all propellent mass, structural mass, and payload mass), the mass of all previously ejected stages, and the mass flow rate of the current stage; see Eq. (10).

$$\frac{dm_n}{dt} = \frac{m_{p,n}}{t_{b,n}} \tag{9}$$

$$m(t, n) = m_0 - \sum_{i=0}^{n-1} m_{f,i} - \frac{dm_n}{dt}(t - t_{b,n-1}) \tag{10}$$

Note that in these calculations, the final mass of each rocket stage must be computed using the input stage mass fraction data. For this, the initial and final masses of each stage are found using the constant, initial propellent mass for each stage and the input stage mass fraction, see Eqs. (11-13).

$$MR_n = \frac{m_{0,n}}{m_{f,n}} \tag{11}$$

$$m_{0,n} = \frac{MR_n}{MR_n - 1} m_{p,n} \tag{12}$$

$$m_{f,n} = \frac{1}{MR_n - 1} m_{p,n} \tag{13}$$

With the main set of calculations for the simulation described, the procedure for calculating the maximum altitude across a range of input data can be found as outlined in the appendix. While this method provides accurate results for small time scales, the cumulative local error (proportional to the square of the timestep considered) will be non-zero across the course of the simulation and thus the final result will need to be cross-verified to ensure the simulation converges to the same result regardless of the timestep. This timescale independence analysis to tune the timestep parameter will be performed using the best-performing vehicle as this is likely to have the longest flight time and thus the most accumulated error. Therefore, the maximum stage mass fraction values will produce the lightest vehicle, see Eq. (11), which will be used to make sure a sufficiently small timestep value is used for convergence in all simulations.

V. Results

The vehicle trajectory simulation first was completed for a timescale-independent study as shown in Fig. 3. This study was performed using the best-performing vehicle launch data, taken initially to be the lightest launch vehicle given that this will have the largest acceleration at each rocket stage, and later verified using a full ranged stage mass fraction analysis. Therefore, only the motor mass was considered for such a vehicle with no structural mass besides the present given mass for the motor burnout mass at each stage. For this initial study, stage mass fractions of $MR_1 = 14.81$, $MR_2 = 11.55$, and $MR_3 = 8.09$ were used to produce an overall vehicle mass fraction of $MR = 8.095$.

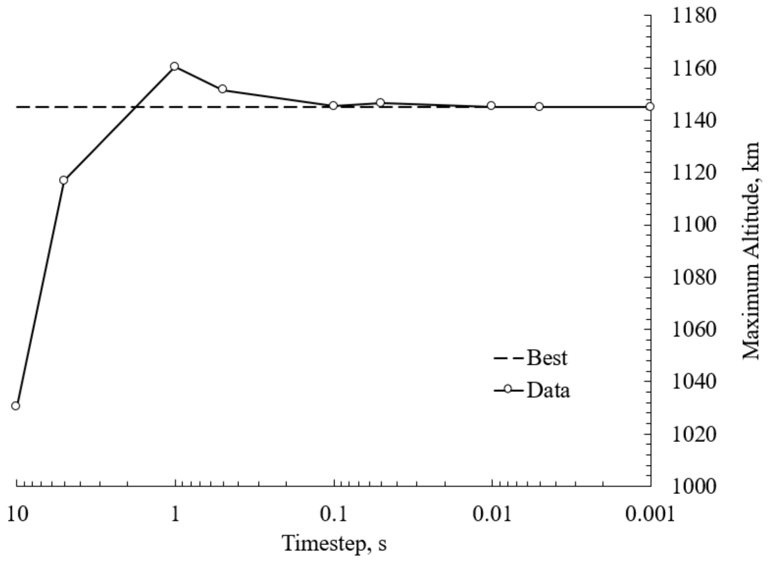


Fig. 3 Timestep convergence for best performing vehicle.

These results indicate that a moderately small timescale is needed for convergence, in this case a timestep of Δt = 0.01s is needed in order to converge to a maximum altitude of 1,145km, accurate to four significant figures. These simulations also indicate that an apogee time of 642s is converged at this same timescale and thus the error bounds of these results are determined for an initial approximation. Note that the actual performance of such a rocket will have drastically different results given the three-dimensional, nonlinear vehicle dynamics outside the scope of this report.

Using these same vehicle stage mass fractions, the full trajectory analysis can be computed as shown in Fig. 4 below. For this vehicle, the delta-v contributions of each stage are found to be 2.044, 1.209, and 0.028 km/s respectively. The rocket passes the Kármán line (100km) at 90s, reaches apogee at 642s, re-enters the atmosphere at 1170s, and crashes at an estimated 1200s. The maximum external Mach number achieved is 8.155 on ascension and 14.781 during descent as well.

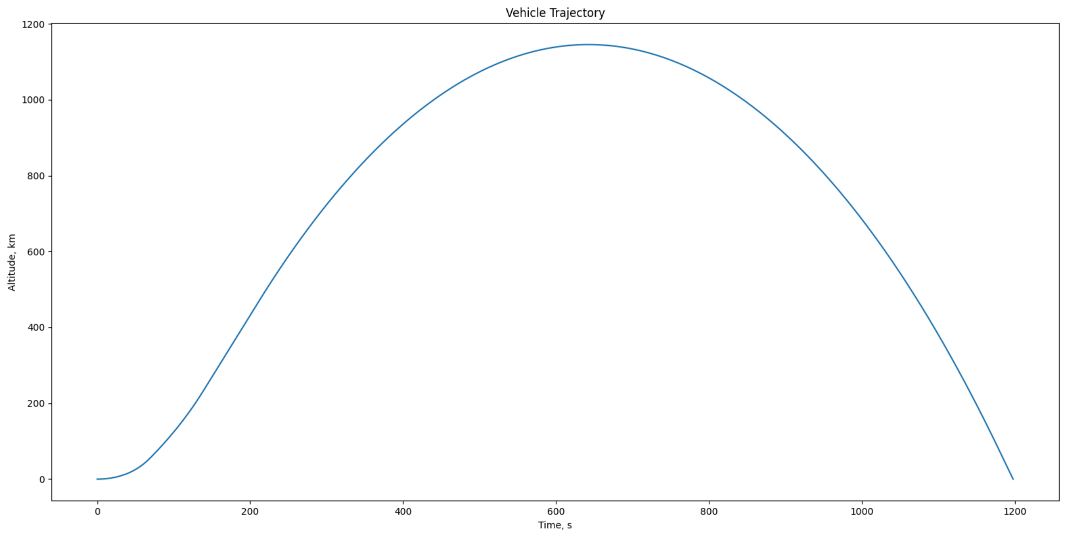


Fig. 4 Simulated vehicle trajectory curve.

A range of vehicle stage mass fractions are considered for this simulation in order to determine the effect of the overall vehicle performance to the initial mass distributions on the rocket. For this, a conservative minimum stage mass fraction considered for each stage is to be half the maximum allowable calculated from before–i.e., stage mass fractions of MR_1 = 7.4037, MR_2 = 5.7757, and MR_3 = 4.0453–and thus provides an upper bound for this analysis. The step size for these stage mass fraction ranges used is simply 1.0 to allow for reasonable computation times, see Fig. 5. In this simulation, the general trend of decreasing maximum altitude with increasing vehicle mass is observed with a wide range of maximum altitudes from 852km (MR = 8.713) to 1,158km (MR = 8.074). Note that due to rounding error, an impossible upper limit is set by this simulation (MR > 8.095).

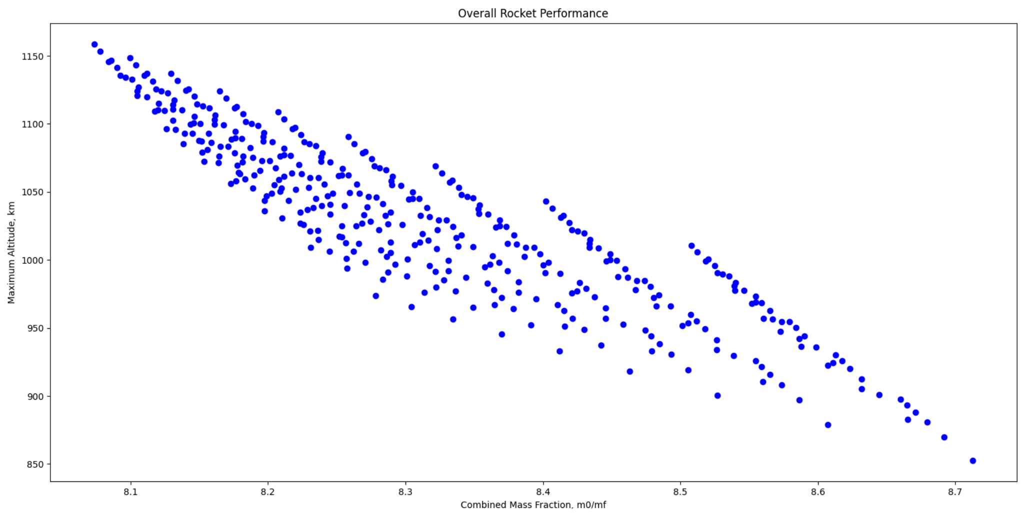


Fig. 5 Trajectory distribution range across the combined vehicle mass fraction.

The effect of the stage mass fraction of each individual stage can also be computed from this simulation. Here, the stage mass fraction of the *n*-th stage ranges from half of its maximum value to its maximum value with a step size of 0.1 and using a constant, maximum stage mass fraction for the remaining stages, refer to the appendix for more information. From this analysis, the following trajectory ranges are observed: for varying first stage mass fractions, a range of 1,004km (MR = 8.514) to 1,146km (MR = 8.092) is achieved; for the second stage, 1,016km (MR = 8.246) to 1,145km (MR = 8.095) is achieved; and for the third stage, 1,089km (MR = 8.143) to 1,145km (MR = 8.095) is achieved.

VI. Discussion

A. Realistic Applications

As the trajectory simulation for the repurposed Pegasus XL sounding rocket indicates, a maximum altitude of 1,145 km is achieved which is significantly above the Kármán line and allows for suborbital spaceflight using a sounding rocket with important implications for missions with high-altitude experiments or data collection about the atmosphere. Moreover, the velocity profile using the Flight Speed graph follows a suborbital trajectory pattern, with the most rapid ascent at the beginning reaching the peak near the apogee, and rapid deceleration to a negative velocity value on descent, which is important to understand and plan payloads for deployment and retrieval during the fall. The trajectory code successfully models one-dimensional rocket motion in the presence of gravity and atmospheric drag, with a noticeable atmospheric density decline at higher altitudes. Also, this model does not include any realistic flight considerations, like a 2D trajectory, Earth's rotation, global wind patterns, etc. Linear interpolation assumes a constant rate of change between data points, which does not accurately capture the exponential decrease in atmospheric density with altitude. This results in a misrepresentation of the actual atmospheric conditions, especially in the upper atmosphere where changes are more pronounced, affecting drag calculations and subsequent flight dynamics. Misestimating atmospheric density due to linear interpolation directly impacts the calculation of drag and lift forces, as these are proportional to the density of the air through which the vehicle is traveling, leading to inaccurate assessments of aerodynamic resistance and vehicle stability during flight

The relationship between maximum altitude and individual stage mass fractions demonstrates the differential impact of mass distribution on rocket performance, as depicted in Fig. 5. The numerical analysis shows that variations in mass fractions for each stage significantly influence the achievable altitude: for the first stage, varying the mass fraction from 8.514 to 8.092 allows the rocket to reach between 1,004 km and 1,146 km; for the second stage, adjusting the mass fraction from 8.246 to 8.095 results in altitudes from 1,016 km to 1,145 km; and for the third stage, altering the mass fraction from 8.143 to 8.095 achieves altitudes from 1,089 km to 1,145 km. This data suggests that adding mass to the third stage rather than to the first or second results in a more significant reduction in maximum altitude, underscoring the importance of efficient mass management, especially in the later stages of the flight. This is because the additional mass in the third stage is carried for a longer duration throughout the flight, which not only requires more energy to maintain but also diminishes the effectiveness of the final push towards peak altitude. Understanding this dynamic is crucial for optimizing stage design and fuel allocation in rocket engineering, particularly for missions aiming to maximize altitude and minimize cost. Such insights are invaluable for refining current models and enhancing the performance and reliability of future suborbital launches.

B. Areas for Refinement

The assumption of constant thrust in rocket trajectory simulations simplifies calculations but introduces several inaccuracies due to neglecting the changing dynamics of the vehicle as fuel is consumed and the vehicle mass decreases. This constant thrust assumption often results in an inaccurate estimation of the trajectory, especially in high-precision applications, such as satellite launches or interplanetary missions, where even minor deviations can lead to significant trajectory alterations. The simulation and optimization of thrust parameters can significantly improve trajectory accuracy by adapting to the dynamic changes during the flight. Moreover, not accounting for thrust variations due to changing atmospheric conditions and vehicle acceleration might underestimate the effects on the vehicle's stability and control during critical phases of the flight. The assumption of a fixed drag coefficient (C_D) based on limited data points significantly restricts the accuracy of aerodynamic modeling; addressing this by employing Computational Fluid Dynamics (CFD) with vortex panel methods and including variable Mach number effects can provide a more precise and tailored Cd profile, enhancing the fidelity of trajectory simulations.

Dynamic thrust which would adjust the thrust due to the changes in altitude and speed would make a more realistic assessment of each stage's motor performance. Adding more dimensions to the problem would allow the rocket's orientation or any Earth's rotation related considerations to be factored into the flight, two fundamental extensions for future research and mission making. More sophisticated atmospheric models should be used, or real-time weather data to make a more accurate prediction about the drag and distance to the apogee. Assuming a real-time atmosphere would help the flight predict better apogees, accounting for aerodynamic effects, and target engine burnout, curbing fuel use if needed. Lastly, robust error and probabilistic models can be added allowing for more effective risk assessment or maintenance contingencies, which as a result makes the flight safer and reliable for future missions, especially for scientific purposes.

C. Conclusions

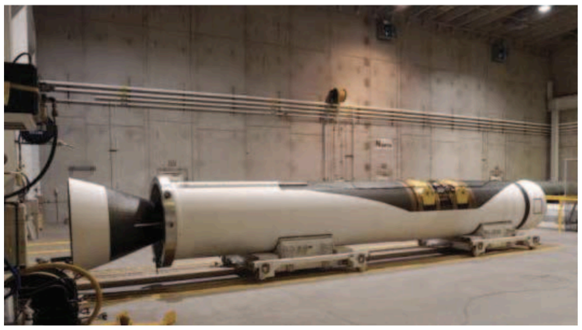
The three-stage repurposed Pegasus XL sounding rocket, the trajectory simulation has effectively shown its capability to attain suborbital altitudes and its future applicability for high-altitude missions. Lastly, the trajectory analysis for different mass fractions of the stage is a point of focus in mass optimization aimed at improving the rocket's performance to give an increased mass reduction in the latter stages of the flight to improve the altitude and efficiency. Further, the adoption of a more refined atmospheric model and assimilated real-time atmospheric information will promise large improvements in the accuracy of the trajectory simulations by providing precise, location-specific environmental conditions such as air density, temperature, and wind patterns. This level of detail allows for more accurate calculations of forces like drag and lift that directly impact the vehicle's flight path, leading to more reliable predictions and optimized mission planning.

The project has deepened the understanding of multistage rocket dynamics and highlighted the viability of conducting cost-effective scientific and exploration missions with repurposed launch vehicles. Further advanced research, including advanced simulation techniques and detailed study of environmental and vehicle-specific factors, will refine the process for rocket design and extend the capabilities of aerospace technology. The methodologies and findings elaborated in this paper provide a broad context of new insights into aerospace engineering that could find relevance with advanced knowledge, having practical applications in the areas of space exploration and beyond.

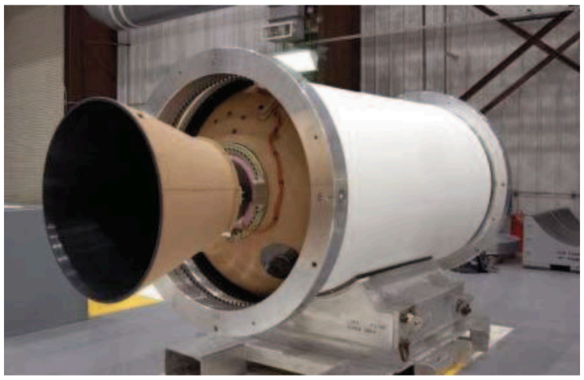
Appendix

This appendix contains additional information relevant to the design and analysis of the vehicle trajectory code developed in this research. The Pegasus XL launch vehicle is composed of three stages: the first containing an Orion 50S XL motor, the second containing an Orion 50 XL motor, and the third containing an Orion 38 motor, Fig. 6. Each of these motors were assumed to be performing under ideal conditions and specified by catalog data [3].

(a)



(b)



(c)

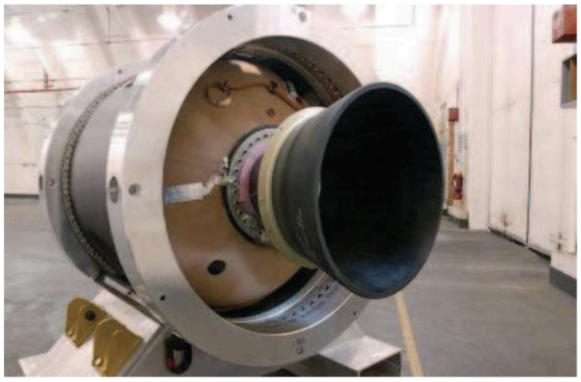


Fig. 6 Launch vehicle motors. (a) Orion 50S XL, (b) Orion 50 XL, and (c) Orion 38.

While the back pressure on each of these motors is neglected in this analysis, atmospheric conditions are still present on the vehicle. Namely, the drag force acting on the vehicle body is proportional to the density of air at the current vehicle altitude, and the drag coefficient is related to the Mach number which in turn requires computing the speed of sound at each altitude, refer to Eq. (8). The information for the atmospheric conditions at each altitude can be calculated using linear interpolation from tabulated data [6], Fig. 7.

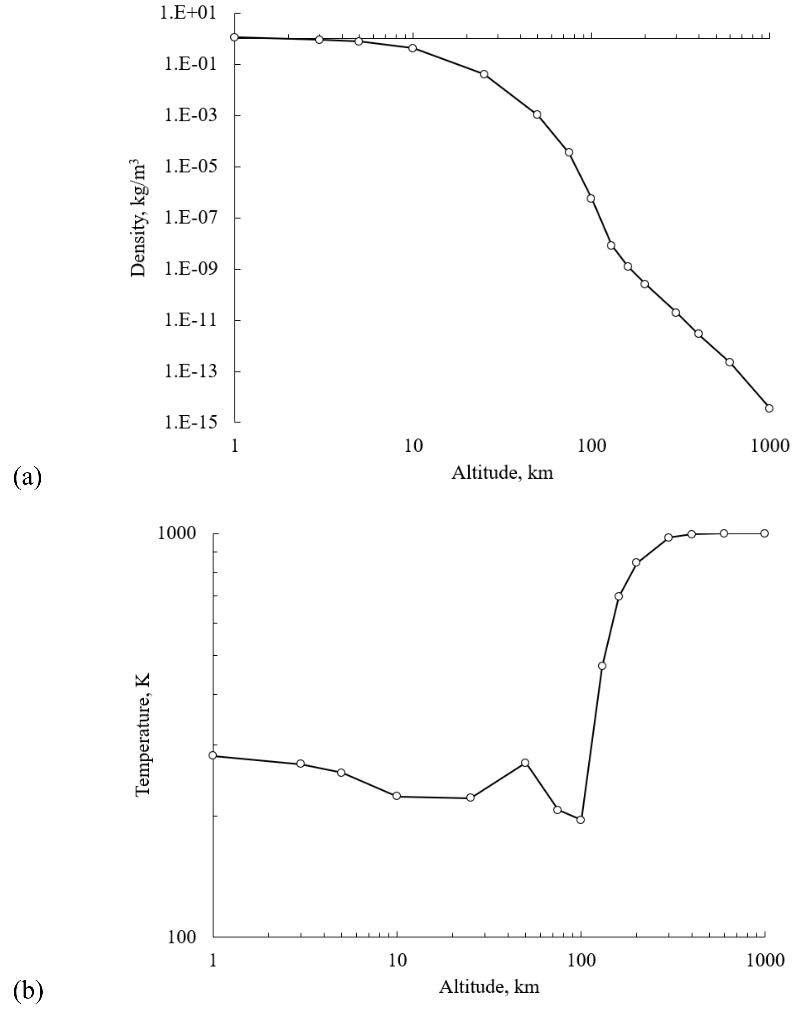


Fig. 7 Atmospheric conditions by altitude. (a) Air density, (b) temperature.

The mass distribution data was pertinent to understanding the effect of proper rocket staging to the overall rocket performance and vehicle dynamics. For this analysis, several trials were performed to vary the stage mass fractions of each stage individually and determine the maximum achieved altitude for this vehicle configuration, Fig. 8. While a linear relationship is roughly shown, an exponential decay in rocket performance is expected with a large mass fraction range considered. This data allows for significant comparisons between rocket stages given the same starting mass. For instance, using a linear regression to the data shown, a rocket with an overall mass fraction of *MR* = 8.12 will reach an altitude of around 1,145km when the added mass is placed on the first stage, 1,122km when the added mass is placed on the second stage, and 1,117km when the added mass is placed on the third stage.

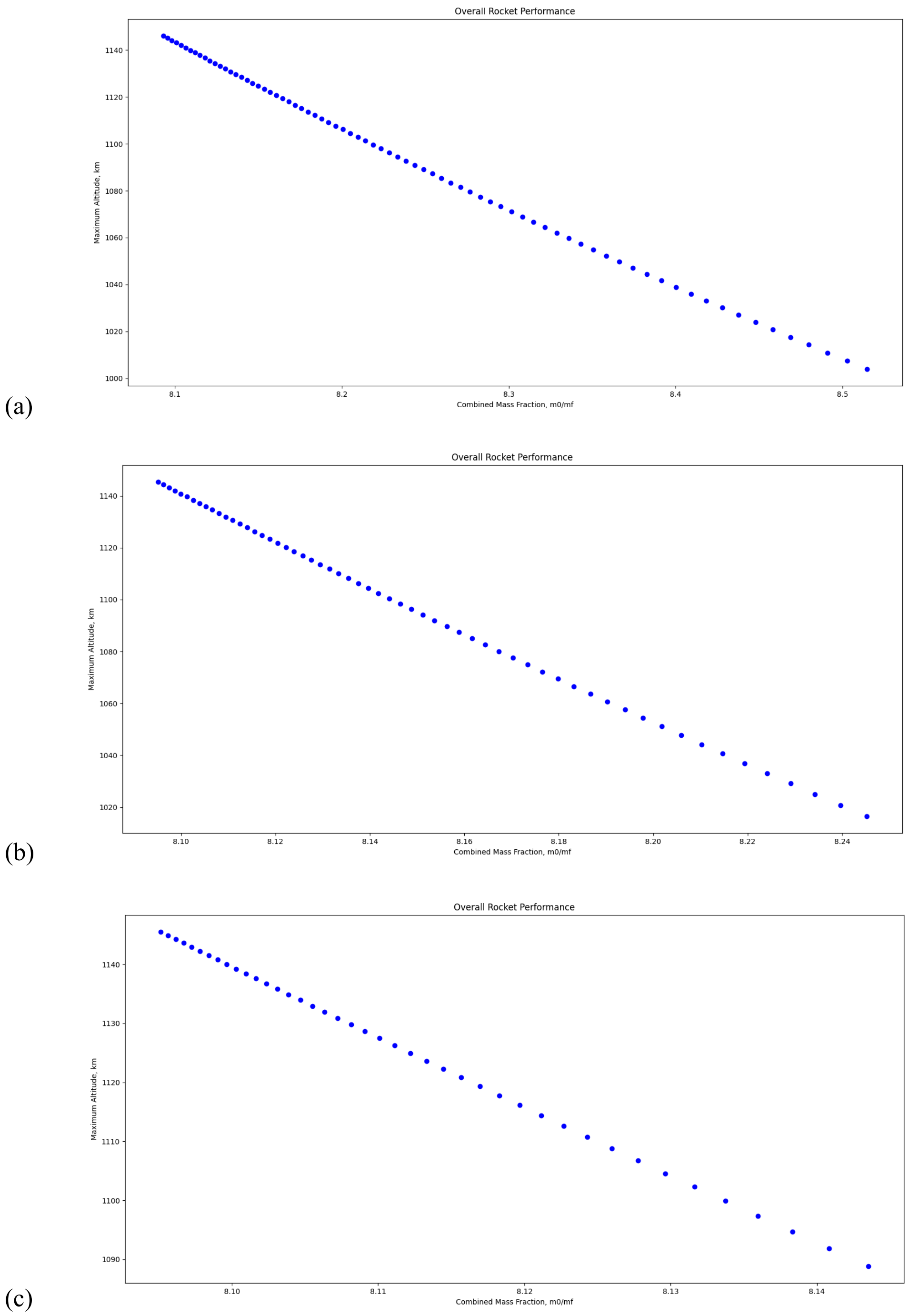


Fig. 8 Stage mass contributions to vehicle trajectory. (a) First stage, (b) second stage, (c) third stage.

A comprehensive overview of the trajectory Python code utilized in the simulation study is also detailed in this report. An overview of the operating procedures for the simulation are outlined in Table 1–this is structured to guide the reader through the setup, execution, and analysis phases of the simulation, highlighting how various input parameters and physical properties are integrated into the model to predict rocket performance accurately.

Table 1 Summary procedures for the vehicle trajectory simulation.

Step	Procedures
1	Parse all motor data from the specified motor catalog for each rocket stage. Convert all units when applicable and print the maximum allowable stage mass fraction data.
2	Receive user input data for the desired stage mass fraction range for each stage. Convert these ranges into arrays for iterative solving.
3	Compute the starting vehicle mass based on the current stage mass fraction to simulate the initial propellant mass for each stage.
4	Determine the current stage of the vehicle based on the motor burnout time data.
5	Calculate the current vehicle mass based on all previously ejected stages, the initial loaded mass, and the mass flow rate of the current stage.
6	Calculate the drag force on the vehicle. Determine the atmospheric conditions at the current altitude to calculate the Mach number. Use the drag coefficient-Mach number relationship to find the drag force.
7	Calculate the gravitational force using the current vehicle altitude.
8	Determine the thrust of the rocket using the average burntime thrust value of the current stage motor from the specified motor catalog.
9	Utilize Euler's method to compute the velocity and altitude at each timestep.
10	Track all important variables for plotting purposes, such as the overall vehicle mass ratio, the height at each timestep, the vehicle speed, the external Mach number, and the mass of the vehicle.
11	Repeat steps 3-10 for each stage mass fraction input in Step 2.
12	Return the simulation results including any plots generated or trajectory summary data.

The complete Python code used to perform this simulation is shown below as well Fig. 9. Refer to Figs. 4, 5, and 8 for the expected output of the code with varying stage mass fraction, timestep, and mass fraction step size inputs. Not shown in this report is the numerical output generated for the user, including information about the best performing flight–such as the stage mass fractions used, the maximum altitude, the highest Mach number on ascent and descent, etc.–as well as the delta-v contributions to the vehicle trajectory from each stage.

```
# TITLE: 1-D Vehicle Launch Trajectory
# AUTHOR: Andrew Smith
# UPDATED: 05/02/2024
# VERSION HISTORY:
# - (4/16) Added stage mass fraction range to mass calculations
# - (4/16) Included multiple simulations across all MR values, print range of trajectories
# - (4/17) Improved user interface for increased functionality
# - (4/17) Added plots of trajectory & overall mass fraction versus maximum altitude achieved
# - (4/21) Fixed plot labels and overall mass fraction definition
# - (4/28) Included vehicle dynamics information (delta-v and acceleration) and added plots
# - (5/2) Added vehicle mass vs. time plot for debugging purposes
# - (5/2) Corrected error in vehicle mass calculations



# I. DEFINE PARAMETERS

# Import libraries
import math
import numpy as np
import time
import matplotlib.pyplot as plt

# Rocket motor characteristics
O50SXL_P = [69.1, 9737000, 140802]             # stage 1, performance values -> [ burn time
(s), total impulse (lbf-sec), burn time average thrust (lbf) ]
O50SXL_W = [35656, 33121, 1923, 545, 83, 2408] # stage 1, weight values (lbm) -> [ total
loaded, propellent, case, nozzle, other, burnout ]
O50SXL_D = [50, 404]                           # stage 1, dimensions (in) -> [ motor diameter,
motor length ]
O50XL_P =  [69.7, 2518000, 36096]              # stage 2, " "
O50XL_W =  [9520, 8650, 551, 240, 79, 824]     # stage 2, " "
O50XL_D =  [50, 122]                           # stage 2, " "
O38_P =    [67.7, 491000, 7246]                # stage 3, " "
O38_W =    [1966, 1699, 133, 91, 46, 243]      # stage 3, " "
O38_D =    [38, 53]                            # stage 3, " "

# Input parameters
print('TITLE: 1-D Vehicle Launch Trajectory (V1.3)')
print('AUTHOR: Andrew Smith')
print('UPDATED: 05/02/2024')
print('')
print('========================================================')
print('')
print('Rocket motor mass fraction data (theoretical max): ')
print('  Stage 1 (MR): ' + str("{:.4f}".format( O50SXL_W[0]/O50SXL_W[5] )) ) # baseline mass
fraction (mo/mf) -> use as minimum for stage mass fraction
print('  Stage 2 (MR): ' + str("{:.4f}".format( O50XL_W[0]/O50XL_W[5] )) )   # " "
print('  Stage 3 (MR): ' + str("{:.4f}".format( O38_W[0]/O38_W[5] )) )
print('')
print('========================================================')
print('')
print('Enter stage mass fraction (mo/mf) ranges as: min max')
MR_1 = [] # set up variables for input
MR_2 = [] # " "
MR_2 = [] # " "
MR_1 = [float(item) for item in input("  Stage 1: ").split()] # stage 1 motor mass fraction
range -> [min, max]
MR_2 = [float(item) for item in input("  Stage 2: ").split()] # stage 2 " "
MR_3 = [float(item) for item in input("  Stage 3: ").split()] # stage 3 " "
if len(MR_1) > 1 or len(MR_2) > 1 or len(MR_3) > 1:    # if multiple values entered, ask for
step size
    dMR  = float(input("  Mass fraction step size: ")) # step size for mass fraction values
print('Enter simulation parameters')
dt = float( input("  timestep (s): ") )    # time step for simulation
t_max = float( input("  max time (s): ") ) # maximum time to compute simulation
print('')
print('========================================================')
print('')

# Constant values
lbm_to_kg = 0.453592 # mass
lbf_to_N = 4.44822   # force
in_to_m = 0.0254     # length

# System settings
t = 0.0     # current time (s)
h = 0.0     # current altitude (m)
v = 0.0     # current velocity (m/s)
a = 0.0     # current acceleration (m/s^2)
F = 0.0     # current force (N)
stage = 1   # current rocket stage

# Vehicle characteristics
pl = 3000 # payload (kg)



# II. GLOBAL FUNCTIONS

def array_conversion(vals, factors):
    # Convert an array of values using a single value or an array of scaling factors.
    new_vals = []
    if( type(factors) == list ): # multiple values provided
        if ( len(factors) == len(vals) ):
            for i in range(len(vals)):
                    new_vals.append(vals[i] * factors[i])
        else:
```

```python
            raise Exception("Must enter the same number of factors as input values.")
    else: # single value provided
        for i in range(len(vals)):
            new_vals.append(vals[i] * factors)
    return new_vals


def atm_density(alt):
    # Calculate the density of the atmosphere at altitude alt (in km) using linear
interpolation.

    # Define known data points -> values taken from textbook (Appendix 2)
    h = [0, 1, 3, 5, 10, 25, 50, 75, 100, 130, 160, 200, 300, 400, 600, 1000] # known altitude
data points (in km)
    p = [1.2250, 1.1117, 9.0912e-1, 7.6312e-1, 4.1351e-1, 4.0084e-2, 1.0269e-3, 3.4861e-5,
5.604e-7, 8.152e-9, 1.233e-9, 2.541e-10, 1.916e-11, 2.803e-12, 2.137e-13, 3.561e-15] # density
values at above altitudes (in kg/m^3)

    # Get index of value before point
    for i in range(len(h)):
        if h[i] <= alt:
            i1 = i # index of largest value below input altitude

    # Get index of value after point
    if i1 == 0:
        i2 = 1 # use first two points
    elif i1 == len(h)-1:
        i1 = len(h)-2 # use last two points
        i2 = i1 + 1
    else:
        i2 = i1 + 1 # use point after it if not an edge case

    # Compute linear interpolation
    p_lin = p[i1] + ((p[i2]-p[i1])/(h[i2]-h[i1])) * (alt-h[i1])

    if alt > h[-1]: # for values exceeding data table, assume density =  0
        p_lin = 0

    return p_lin


def grav(alt):
    # Calculate the gravitational acceleration at altitude alt (in km) above the surface.
    R0 = 6371.000 # volumetric mean radius (km) ->
https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html
    g0 = 9.820    # mean surface gravity (m/s^2) -> see above link
    g = g0 * (R0 / (R0 + alt))**2
    return g
```

```
def mach(vel, alt):
    # Determine the Mach number based on the current rocket velocity (in m/s) and altitude (in
km).
    # Gas properties
    gamma = 1.4     # specific heat ratio (air) -> assume constant
    R0 = 8.314      # ideal gas constant (J/K-mol)
    Mw = 28.96/1000 # average molecular weight of air (kg/mol) ->
https://unacademy.com/content/question-answer/chemistry/what-is-the-molecular-weight-of-air/#:
~:text=Air%20is%20a%20mixture%20of%20several%20gasses%20where%20the%20two,carbon%20dioxide%20o
f%20about%200.03%25.&text=We%20get%2028.96%20g%2Fmol,the%20molecular%20weight%20of%20Air.
    R = R0 / Mw     # specific gas constant for air (J/kg-K)

    # Define known data points -> values taken from textbook (Appendix 2)
    h = [0, 1, 3, 5, 10, 25, 50, 75, 100, 130, 160, 200, 300, 400, 600, 1000] # known altitude
data points (in km)
    T = [288.150, 281.651, 268.650, 255.650, 223.252, 221.552, 270.650, 206.650, 195.08,
469.27, 696.29, 845.56, 976.01, 995.83, 999.85, 1000.00] # temperature data points (in K)

    # Get index of value before point
    i1 = 0
    for i in range(len(h)):
        if h[i] <= alt:
            i1 = i # index of largest value below input altitude

    # Get index of value after point
    if i1 == 0:
        i2 = 1 # use first two points
    elif i1 == len(h)-1:
        i1 = len(h)-2 # use last two points
        i2 = i1 + 1
    else:
        i2 = i1 + 1 # use point after it if not an edge case

    # Compute linear interpolation
    T_lin = T[i1] + ((T[i2]-T[i1])/(h[i2]-h[i1])) * (alt-h[i1])

    if alt > h[-1]: # for values exceeding data table, use largest value
        T_lin = T[-1]

    # Compute Mach number
    a = math.sqrt(gamma * R * T_lin) # speed of sound based on atmospheric temperature
    return (vel / a)



def drag_coeff(M):
    # Estimate the drag coefficient on the vehicle based on the Mach number.
    # Values found Cd vs M graph on page 105 of textbook (zero angle of attack)
    Ms = [0, 0.192434211, 0.394736842, 0.587171053, 0.65625, 0.715460526, 0.764802632,
0.809210526, 0.863486842, 0.917763158, 0.967105263, 1.041118421, 1.090460526, 1.144736842,
1.174342105, 1.248355263, 1.292763158, 1.386513158, 1.549342105, 1.623355263, 1.751644737,
1.879934211, 2.077302632, 2.457236842, 2.930921053, 3.5625, 4.154605263, 4.702302632,
5.264802632, 5.496710526] # known Mach number values (approximated from Excel)
    Cd = [0.147040498, 0.147040498, 0.147040498, 0.148286604, 0.153271028, 0.150778816,
0.175700935, 0.190654206, 0.214330218, 0.251713396, 0.291588785, 0.357632399, 0.396261682,
0.416199377, 0.413707165, 0.395015576, 0.380062305, 0.352647975, 0.309034268, 0.290342679,
0.264174455, 0.249221184, 0.236760125, 0.224299065, 0.208099688, 0.186915888, 0.168224299,
0.15576324, 0.145794393, 0.144548287]  # known Cd values (see above)

    # Get index of value before point
    i1 = 0
    for i in range(len(Ms)):
        if Ms[i] <= M:
            i1 = i # index of largest value below input Mach number

    # Get index of value after point
    if i1 == 0:
        i2 = 1 # use first two points
    elif i1 == len(Ms)-1:
        i1 = len(Ms)-2 # use last two points
        i2 = i1 + 1
    else:
        i2 = i1 + 1 # use point after it if not an edge case

    # Compute linear interpolation
    Cd_lin = Cd[i1] + ((Cd[i2]-Cd[i1])/(Ms[i2]-Ms[i1])) * (abs(M)-Ms[i1])

    if M > Ms[-1]: # for values exceeding data table, use last value
        Cd_lin = Cd[-1]

    return Cd_lin


def print_progress(b):
    # Option to print summary statistics at each time step of the simulation.
    if b == True:
        print("(" + str(i) + "/" + str(num_MRs) + ") -> " + "t: " + str("{:.1f}".format( t ))
+ "s, Stage: " + str(stage) + ", T: " + str("{:.3f}".format( T )) + "N, D: " +
str("{:.3f}".format( D )) + "N, G: " + str("{:.0f}".format( G )) + "N, m-dot: " +
str("{:.1f}".format( m_dot )) + "kg/s, m: " + str("{:.1f}".format( m )) + "kg, a: " +
str("{:.3f}".format( a )) + "m/s^2, v: " + str("{:.1f}".format( v )) + "m/s, M: " +
str("{:.3f}".format( M )) + ", h: " + str("{:.3f}".format( h/1000 )) + "km")


current_dots = -1 # number of dots displayed last
def progress_bar(n, N, d):
    # Print a progress bar for simulation n of N. Input contains number of previously
displayed dots (prevents printing multiple times).
    num_dots = 20                    # total number of dots to use (width of display)
    dots = round((n / N) * num_dots) # number of filled-in dots to display
    b1 = "□"
    b2 = "■"
    if dots != d: # prevents constant printing
        print(b2*dots + b1*(num_dots-dots))
    return dots



# III. PRE-PROCESSING

# Convert motor characteristics to SI units
O50SXL_P = array_conversion(O50SXL_P, [1.0, lbf_to_N, lbf_to_N]) # stage 1, convert
performance values
O50SXL_W = array_conversion(O50SXL_W, lbm_to_kg)                 # stage 1, convert weight
values
O50SXL_D = array_conversion(O50SXL_D, in_to_m)                   # stage 1, convert dimensions
O50XL_P =  array_conversion(O50XL_P, [1.0, lbf_to_N, lbf_to_N])  # stage 2, " "
O50XL_W =  array_conversion(O50XL_W, lbm_to_kg)                  # stage 2, " "
O50XL_D =  array_conversion(O50XL_D, in_to_m)                    # stage 2, " "
O38_P =    array_conversion(O38_P, [1.0, lbf_to_N, lbf_to_N])    # stage 3, " "
O38_W =    array_conversion(O38_W, lbm_to_kg)                    # stage 3, " "
O38_D =    array_conversion(O38_D, in_to_m)                      # stage 3, " "

# Mass parameters
# mass flow rates
dm_1 = O50SXL_W[1] / O50SXL_P[0] # stage 1 mass flow rate, assume constant -> dm = propellent
mass / burn time
dm_2 = O50XL_W[1] / O50XL_P[0]   # stage 2 " "
dm_3 = O38_W[1] / O38_P[0]       # stage 3 " "
m_dot = dm_1                     # initial mass flow rate
# stage mass fractions
if len(MR_1) > 1:
    MR_1 = np.arange(MR_1[0], MR_1[-1] + dMR, dMR) # convert mass fractions to range of values
else:
    MR_1 = [MR_1[0]]                               # if only one value is entered, use just
that value
if len(MR_2) > 1:
    MR_2 = np.arange(MR_2[0], MR_2[-1] + dMR, dMR) # " "
else:
    MR_2 = [MR_2[0]]                               # " "
if len(MR_3) > 1:
    MR_3 = np.arange(MR_3[0], MR_3[-1] + dMR, dMR) # " "
else:
    MR_3 = [MR_3[0]]                               # " "
```

```
# Thrust parameters
tb_total = O50SXL_P[0] + O50XL_P[0] + O38_P[0] # total burn time of all stages
print("Total burn time:         " + str("{:.0f}".format( tb_total )) + "s")

# Simulation data
i = 0                                                                # current simulation
num_MRs = len(MR_1) * len(MR_2) * len(MR_3)                          # number of simulations
to perform
print("Number of simulations:   " + str("{:.0f}".format( num_MRs )) )
expected_time = ( num_MRs * (t_max) / dt ) * 2.3623e-5 + 1.2100e0    # simulations are O(n) ->
use computations vs. execution time data for prediction
if expected_time > 1 and expected_time < 60: # in ms
    print("Expected execution time: " + str("{:.1f}".format( expected_time )) + "s")
elif expected_time > 60:                     # in min, sec
    print("Expected execution time: " + str("{:.0f}".format( math.floor(expected_time/60) )) +
"min " + str("{:.0f}".format( expected_time%60 )) + "s")
else:                                        # in sec
    print("Expected execution time: " + str("{:.0f}".format( expected_time*1000 )) + "ms")

start_sim = input("Start simulation? [Y/N]: ")                       # allow user to halt
operation if desired
if start_sim == "Y" or start_sim == "y":
    pass
else:                                                                # halt program
    print("")
    print("[Program terminated]")
    exit()


# IV. SIMULATION

print('')
print('========================================================')
print('')
ask_print_progress = input("Display simulation progress? [Y/N]: ")
if ask_print_progress == "Y" or ask_print_progress == "y":
    print_progress_bool = True
else:
    print_progress_bool = False
print("Running simulation...")
start_time = time.time() # simulation start time

# Track trajectory data
h_max = [0.0] * num_MRs     # keep track of maximum height reached
t_at_max = [0.0] * num_MRs  # time to reach above maximum altitude
M_max = [0.0] * num_MRs     # maximum Mach number reached
M_min = [0.0] * num_MRs     # " " in descent
t_at_exo1 = [0.0] * num_MRs # time to reach space (pass Karman line)
t_at_exo2 = [0.0] * num_MRs # time to return to atmosphere
```

```
t_crash = [0.0] * num_MRs   # time of rocket crash (return to h=0)
MR_values = []              # list of stage mass fractions used in simulation
s1_vf = [0.0] * num_MRs     # final velocity after stage 1 burnout
s2_vf = [0.0] * num_MRs     # " " stage 2 burnout
s3_vf = [0.0] * num_MRs     # " " stage 3 burnout

# Plotting data
h_vals = []   # plot trajectory information for single value simulations
v_vals = []   # vehicle speed
M_vals = []   # same as above but scaled using temperature (Mach number)
a_vals = []   # dynamics information
t_vals = []   # time for flight
m_vals = []   # mass of rocket over time (used for debugging)
MRo_vals = [] # overall vehicle mass fraction -> used to plot h_max vs. MR

# Run simulation
for mr_1 in MR_1:
    for mr_2 in MR_2:
        for mr_3 in MR_3: # iterate over each stage mass fraction

            # Print computation progress
            current_dots = progress_bar(i, num_MRs, current_dots)

            # Initialize all variables
            t = 0
            v = 0
            a = 0
            h = 0
            stage = 1

            # Compute vehicle initial mass
            m_s1_f = O50SXL_W[1]/(mr_1 - 1)     # stage one final (dry) mass   -> for MR =
mf/mo, mf = mo*MR = 1/(MR-1) * mp
            m_s1_o = m_s1_f * mr_1              # stage one initial (wet) mass -> for MR =
mf/mo, mo = mf/MR = MR/(MR-1) * mf
            m_s2_f = O50XL_W[1]/(mr_2 - 1)      # stage two " "
            m_s2_o = m_s2_f * mr_2              # stage two " "
            m_s3_f = O38_W[1]/(mr_3 - 1)        # stage three " "
            m_s3_o = m_s3_f * mr_3              # stage three " "
            m0 = m_s1_o + m_s2_o + m_s3_o + pl # full rocket initial mass

            # save values
            MR_values.append([mr_1, mr_2, mr_3]) # save current stage mass fractions
            MRo_vals.append( m0 / pl )           # overall vehicle mass fraction -> intial
mass / final mass

            # Run simulation
            while (t <= t_max):
                # Perform vehicle staging
                if t >= O50SXL_P[0]:              # stage 1 has burned out -> engage stage 2
                    stage = 2
                if t >= O50SXL_P[0] + O50XL_P[0]: # stage 2 has burned out -> engage stage 3
                    stage = 3

                # Set mass flow rate
                if stage == 1:
                    m_dot = dm_1
                if stage == 2:
                    m_dot = dm_2
                if stage == 3:
                    m_dot = dm_3
                if t > tb_total:
                    m_dot = 0 # no mass flow after final burnout time is reached

                # Determine current mass
                if stage == 1:
                    m = m0 - m_dot*t                                                    #
subtract expelled propellent
                if stage == 2:
                    m = m0 - m_s1_o - m_dot*(t - O50SXL_P[0])                           # " "
+ subtract above stage burnout mass
                if stage == 3:
                    m = m0 - (m_s1_o + m_s2_o) - m_dot*(t - (O50SXL_P[0] + O50XL_P[0])) # " "
                if t > tb_total:
                    m = pl                                                              #
subtract all stage burnout masses


                # Find drag force
                # cross sectional area
                if stage == 1:
                    A = math.pi/4 * O50SXL_D[0]**2
                elif stage == 2:
                    A = math.pi/4 * O50XL_D[0]**2
                elif stage == 3:
                    A = math.pi/4 * O38_D[0]**2
                # drag coefficient
                M = mach(v, h/1000) # get mach number
                cd = drag_coeff(M)  # get drag coefficient
                # overall drag
                if v < 0:    # return direction -> drag acts upwards
                    D = -0.5 * atm_density(h/1000) * v**2 * A * cd
                elif v > 0:  # ascension -> drag acts downwards
                    D = 0.5 * atm_density(h/1000) * v**2 * A * cd
                elif v == 0: # otherwise, no drag
                    D = 0
```

```
# Find gravitational force
G = grav(h/1000)*m # use current altitude and mass

# Calculate thrust
if stage == 1:
    T = O50SXL_P[2] # assume constant thrust
elif stage == 2:
    T = O50XL_P[2]  # " "
elif stage == 3:
    T = O38_P[2]    # " "
if t >= tb_total: # all rockets have burned out
    T = 0         # no thrust produced after this point

# Update physics states
F = T - G - D # force = thrust - gravity - drag
a = F / m     # Newton's 2nd law
v += a*dt     # find velocity based on acceleration
h += v*dt     # increase height
t += dt       # update time



# Track state variables
# maximum Mach numbers
if M > M_max[i]:
    M_max[i] = M    # save highest Mach number
elif M < M_min[i]:
    M_min[i] = M    # " " in return direction (negative)

# maximum altitude
if h > h_max[i]:
    h_max[i] = h       # update peak position value (t, h)
    t_at_max[i] = t    # " "
elif h <= 0 and t > 0:
    h = 0          # rocket has crashed
    v = 0          # " "
    t_crash[i] = t # save crash time
    break

# exo-atmospheric flight
if h < 100e3: # Karman line = 100km
    if t <= t_at_max[i]: # first time reaching point
        t_at_exo1[i] = t
elif h > 100e3:
        t_at_exo2[i] = t

# current altitude (single mass fraction only)
if num_MRs == 1:             # only plot if one mass fraction is input
```

```
                    h_vals.append(h/1000)    # save current h value (in km)
                    v_vals.append(v)         # save current speed (in m/s)
                    M_vals.append(M)         # " " (as Mach number)
                    a_vals.append(a/grav(0)) # save current acceleration (in g's)
                    m_vals.append(m)         # save current vehicle mass (in kg)
                    t_vals.append(t)         # save time (in s)

                # final burnout velocity
                if stage == 1:
                    s1_vf[i] = v # update velocity until next stage starts
                elif stage == 2:
                    s2_vf[i] = v # " "
                elif stage == 3:
                    if t < tb_total: # calculate final speed at burnout, not landing
                        s3_vf[i] = v # see above


                # Print current data (optional)
                print_progress(print_progress_bool) # toggle on/off to print simulation
progress at each timestep (turn off for speed)


            # Update simulation count
            i += 1


# V. DISPLAY RESULTS

print("Done.")
end_time = time.time()               # simulation end time
elapsed_time = end_time - start_time # total computation time (wall time) for simulation
if elapsed_time > 1:
    print("Execution time: " + str("{:.1f}".format( elapsed_time )) + "s ")
else:
    print("Execution time: " + str("{:.0f}".format( elapsed_time*1000 )) + "ms ")

print('')
print('========================================================')
print('')

max_alt = max(h_max)         # highest peak altitude achieved
min_alt = min(h_max)         # lowest peak altitude achieved
index = h_max.index(max_alt) # get index of best performing flight

print("Simulation Data:")
print("  Stage mass fractions: " + "[" + str("{:.1f}".format( MR_1[0] )) + "-" +
str("{:.1f}".format( MR_1[-1] )) + "], [" + str("{:.1f}".format( MR_2[0] )) + "-" +
str("{:.1f}".format( MR_2[-1] )) + "], [" + str("{:.1f}".format( MR_3[0] )) + "-" +
str("{:.1f}".format( MR_3[-1] )) + "]") # display mass fraction ranges
```

```
print("  Trajectory Range:     " + str("{:.3f}".format( min_alt/1000 )) + "-" +
str("{:.3f}".format( max_alt/1000 )) + "km" )
print("")

print("Best Flight Data: ")
print("  Stage mass fractions: " + str("{:.2f}".format( MR_values[index][0] )) + ", " +
str("{:.2f}".format( MR_values[index][1] )) + ", " + str("{:.2f}".format( MR_values[index][2]
))) # stage mass fractions used
print("  Maximum altitude:     " + str("{:.3f}".format( max_alt/1000 )) + "km")
# primary output of simulation
print("  Apogee time:          " + str("{:.3f}".format( t_at_max[index] )) + "s")
# time to reach above altitude
print("  Mach:                 " + str("{:.3f}".format( M_max[index] )) + " (" +
str("{:.3f}".format( M_min[index] )) + ")")
# maximum Mach number on ascension (and descension)
print("  Space flight:         " + str("{:.2f}".format( t_at_exo1[index] )) + "s" + " (" +
str("{:.2f}".format( t_at_exo2[index] )) + "s)")
# time to escape (and return to) atmosphere
print("  Crash time:           " + str("{:.2f}".format( t_crash[index] )) + "s")
# time to return to starting height
print("")

print("Vehicle Dynamics*:")
print("  Stage 1 delta-V: " + str("{:.3f}".format( s1_vf[index]/1000 )) + "km/s")
# compute delta V (no initial velocity)
print("  Stage 2 delta-V: " + str("{:.3f}".format( (s2_vf[index] - s1_vf[index])/1000 )) +
"km/s") # " " use previous stage burnout velocity as initial velocity
print("  Stage 3 delta-V: " + str("{:.3f}".format( (s3_vf[index] - s2_vf[index])/1000 )) +
"km/s") # " "
print("  *for best performing flight (see above)")




# VI. PLOT RESULTS
if num_MRs == 1: # overall trajectory and dynamics (single mass fraction value only)
    plt.figure(1) # altitude vs. time
    plt.plot(t_vals, h_vals)
    plt.xlabel('Time, s')
    plt.ylabel('Altitude, km')
    plt.title('Vehicle Trajectory')

    plt.figure(2) # speed vs. time
    plt.plot(t_vals, array_conversion(v_vals, 1e-3)) # convert values to km/s
    plt.xlabel('Time, s')
    plt.ylabel('Velocity, km/s')
    plt.title('Flight Speed')

    plt.figure(3) # acceleration vs. time
    plt.plot(t_vals, a_vals)
    plt.xlabel('Time, s')
    plt.ylabel('Acceleration, g0\'s')
    plt.title('Vehicle Dynamics')

    plt.figure(4) # flight profile
    plt.plot(M_vals, h_vals)
    plt.xlabel('Mach Number')
    plt.ylabel('Altitude, km')
    plt.title('Flight Profile')

    plt.figure(5) # vehicle weight
    plt.plot(t_vals, m_vals)
    plt.xlabel('Time, s')
    plt.ylabel('Total Mass, kg')
    plt.title('Rocket Mass')

    plt.show()    # show all plots

else:            # vehicle mass fraction vs altitude
    plt.scatter(MRo_vals, array_conversion(h_max, 1e-3), c="blue") # convert to km, plot as
scatter plot
    plt.xlabel('Combined Mass Fraction, m0/mf')
    plt.ylabel('Maximum Altitude, km')
    plt.title('Overall Rocket Performance')
    plt.show()
```

Fig. 9 Complete simulation Python script.

References

[1] Leishman, J., "Rockets & Launch Vehicle Performance," https://eaglepubs.erau.edu/introductiontoaerospaceflightvehicles/chapter/rocket-performance/

[2] Anonymous "Pegasus," https://www.northropgrumman.com/space/pegasus-rocket

[3] Alliant Techsystems Inc., "ATK Space Propulsion Products Catalog," 2008,

[4] Williams, D., "Earth Fact Sheet," https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

[5] Biblarz, O., and Sutton, G., "4.2: Forces Acting On A Vehicle In The Atmosphere," *Rocket Propulsion Elements,* 2017, pp. 105.

[6] Biblarz, O., and Sutton, G., "Appendix 2: Properties of the Earth's Standard Atmosphere," *Rocket Propulsion Elements,* 2017, pp. 747.