AERO4800 Checkpoint 1

47442043

FLYNN DEVOY

THE UNIVERSITY OF QUEENSLAND

Executive Summary

This report chronicles the construction, verification, and utilization of a three-degree-of-freedom (3DOF) multi-stage rocket trajectory code for AERO4800. The task involved modelling United Launch Alliance's Vulcan Centaur Launch System (VCLS) ascent from liftoff to staging, gravity turn, and orbital insertion. The effort was separated into three large phases: code construction, verification with analytical and textbook benchmarks, and utilization for a typical orbital launch simulation.

In Part 1, a generic trajectory solver was written in Python to model thrust, drag, gravitational changes with height, mass losses, and parallel and series staging. The solver was made user-specified stage parameter and gravity-turn profiles compatible. Modularity, with a flowchart and code walkthroughs, was used to allow for transparency and reproducing abilities.

Part 2 also verified the solver with known analytical and numerical standards, such as Curtis' Example 13.2/13.3, and an analytical staging solution. Such comparisons verified proper thrust and mass depletion modelling, stage transition, and gravity/drag loss calculations. Results exhibited excellent correlation with desired velocity, height, and burnout conditions, giving firm confidence in solver reliability.

Part 3 applied the validated solver to the Vulcan Centaur launch vehicle, configured with an appropriate number of solid rocket boosters and realistic propulsion parameters. The simulation successfully demonstrated insertion of a 20-tonne payload into a 400 km circular low Earth orbit within the ±10% payload tolerance and ±5% trajectory requirements specified in the assignment. The trajectory captured key mission events such as max-q, booster separation, core burnout, and Centaur final insertion, with results supported by time histories, event tables, and trajectory plots.

In summary, the solver developed is robust and accurate in modelling real orbital ascent problems. The Vulcan Centaur case study supported that such a system can be launched into orbit with appropriate staging and flight-path control to meet assignment requirements. Beyond academic verification, the report shows broader applicability of trajectory modelling to real launcher design, mission planning, and performance assessment.

Contents

Task 1. Develop a 3DOF Multi-Stage Rocket Trajectory Code

Introduction

This section presents the development of a three degree of freedom trajectory solver for a multistage launch vehicle. The goal was to make use of the simple vertical solver from *Lecture 5: Launch Vehicles 2: Time Resolved and Numerical Solutions* and modify it to accommodate the requirements of the task. This code can be found in the Appendix. These requirements being a python script that can model:

- Velocity
- Altitude
- Downrange Distance
- Flight Path Angle
- Variable Vehicle mass with time (from Thrust, Isp and propellant)
- Aerodynamic drag with altitude-dependent density
- A programmed pitch over manoeuvre to initiate a gravity turn trajectory

Curtis Examples 13.2/13.3 will be used to validate as they include all the essential ingredients of a practical launch (thrust, drag, variable gravity, atmosphere, pitch over), whilst providing clear analytical solutions allowing for accurate verification. Successfully reproducing *Curtis 13.2/13.3* provides confidence that the solver is accurate and extensible, thereby meeting the requirements of Part 1 and Part 2 of the assignment brief.

Table 1. Equations of Motion from Curtis Ch13 – Used as base for Updated Code.

Velocity change over time $$\frac{dv}{dt} = \frac{T}{m} - \frac{D}{m} - g sin\gamma$$ T is thrust, m is instantaneous mass, D is drag, g is local gravity and γ is flight-path angle.
Flight path angle over time $$\frac{d\gamma}{dt} = -\frac{1}{v}\left(g - \frac{v^2}{R_E + h}\right) cos\gamma$$ v is instantaneous velocity, R_E is the radius of Earth $(6{,}371\ km)$ and h is altitude
Altitude change over time $$\frac{dh}{dt} = v sin\gamma$$
Downrange Distance over time $$\frac{dx}{dt} = \frac{R_E}{R_E + h} v cos\gamma$$
Mass depletion over time $$\frac{dm_e}{dt} = \frac{T}{I_{sp} g_0}$$ I_{sp} is the Specific Impulse of the Rocket Engine, and g_0 is gravity at sea level $(9.81\ m/s)$
Aerodynamic Drag $$D = \frac{1}{2}\rho v^2 A C_D$$ ρ is atmospheric density, A is reference area, and C_D is the drag coefficient.

Vehicle Modelling

The solver represents the vehicle using three classes:

1. Class StageElement:

This class represents a single physical rocket element, such as a core stage, solid rocket booster, or an upper stage. Each element stores its fundamental design parameters:

- Name (e.g. Core)
- Thrust T $[N]$, assumed constant during burn
- Specific impulse I_{sp} $[s]$, used to calculate effective exhaust velocity and propellant mass flow rate
- Propellant mass m_{prop} $[kg]$, the initial usable propellant carried by this element.
- Structural mass m_{struct} $[kg]$, representing tanks, engines, avionics, etc.

From these properties the propellant flow rate is derived:

$$\dot{m}_e = \frac{T}{I_{sp} g_0}$$

This quantity determines how quickly the element consumes its propellant during powered flight. Additionally, a state index is attached to each element so that the remaining propellant can be tracked during integration.

2. Class StageGroup:

This class groups multiple elements that burn in parallel. For example, several strap on boosters, and a core stage could all form one *StageGroup*. All elements in a *StageGroup* ignite together and contribute thrust simultaneously until each *StageElement's* propellant is depleted.

3. Class Vehicle:

This class provides the system level representation of the launch vehicle in the 3DOF solver. The *Vehicle* class ties the various *StageGroup's* together into a full multistage launch system that can be integrated through the equations of motion. For example:

- Group 1: Boosters and core stage burning in parallel.
- Group 2: Upper stage operating after booster separation.

The class operates using 5 Functions.

1. Initial State construction:
    - Kinematic variables: velocity, altitude, downrange distance, flight path angle.
    - Per-element variables: remaining propellant mass and a structural "attached" flag (1 = carried, 0 = jettisoned).
2. Active group logic:
    - Determines which *StageGroup* is currently contributing Thrust by always selecting the lowest-index group still attached.
3. Mass Accounting:
    - Computes the instantaneous vehicle mass by summing payload, all attached structural masses, and any remaining propellant across all elements.

- This ensures that inactive upper stages contribute dead weight until their turn to burn.

4. Force Generation:
   - Computes the total thrust and mass flow rate from the active group only, so that only burning engines contribute to acceleration while other stages remain inert
5. Jettison logic:
   - Checks whether individual elements or entire groups have exhausted their propellant.
   - Once empty, their structural flags are set to zero, removing their structural mass from the vehicle and ensuring realistic staging behaviour.

Making use of these functions, the *Vehicle* class ties the entire solver together. It determines which stages are burning, how the total mass evolves with time, when staging events occur, and what forces act on the vehicle at each timestep. This abstraction makes the solver modular and extensible – even though Curtis Example 13.3 involves only a single stage, the same framework would seamlessly support realistic multi-stage launch vehicles such as required in Task 3.

Atmosphere, Gravity and Drag Models:

Two different atmosphere models are implemented in the code able to be switched using *set_atmosphere("model")*.

Table 2. Summary of Atmosphere, Gravity and Drag Models

Exponential Scale Height *("scale"):* $\rho = \rho_0 \exp\left(-\frac{h}{H}\right)$ $\rho_0 = 1.225\ kg/m^3$ and $H = 7500\ m$

US Standard Atmosphere 1976 ("US"):
 This model makes use of recorded density data up to an altitude of 100 km and linearly interpolates, for realism in Task 3. This is done in python using:
 `np.interp(alt, alt_densities, alt)`
 Additionally, the function also returns a density of 0.0 if above 100 km altitude.

Gravity: $g(h) = \frac{g_0}{\left(1 + \frac{h}{R_E}\right)^2}$
Drag: The drag calculation makes use of the formula defined in the Equations of Motion section above, using a defined Reference Area, A_{Ref} and Drag Coefficient C_D.

Pitch over Kick:

In practice, a small, commanded pitch over is required to initiate the turn. Two modes are implemented :

- Instant kick: a discrete reduction in γ applied at the trigger altitude.

```
if kick_mode.lower() == "instant":
            # Instant kick: reduce gamma by kick_delta_deg
            gamB = gamA - np.deg2rad(kick_delta_deg)
            y0_B = y_kick_fixed.copy()
            y0_B[3] = gamB
```

- Rate kick: a constant commanded $\dot{\gamma}$ applied for a specified time.

```
elif kick_mode.lower() == "rate":
            # Rate kick: enable a constant gamma-dot for a fixed
duration
            y0_B = y_kick_fixed.copy()
            gamma_rate_rad = -np.deg2rad(kick_rate_deg_s)
```

Both methods seed the gravity turn while maintaining physical plausibility for actuator limits. After the kick, the natural gravity turn's dynamics dominate, progressively aligning the vehicle with the local horizontal.

Gravity Turn Dynamics:

Once a pitch over altitude is reached, the rocket transitions from vertical ascent into a gravity turn. The flight path angle γ evolves according to:

- Flat Earth Model

$$\dot{\gamma} = -\frac{g}{v} cos\gamma$$

- Curved Earth Model

$$\dot{\gamma} = -\left(\frac{g}{v} - \frac{v}{R_E + h}\right) cos\gamma$$

The curved formulation includes the geometric effect of Earth's surface curvature, preventing over-flattening at high velocity. Physically, these relations describe how the vertical component of gravity continuously reduces the flight path angle. As the vehicle accelerates, the term $\frac{g}{v}$ diminishes, causing the trajectory to gradually flatten without active steering input. The function that enforces the gravity turn dynamics can be seen below. To prevent computational errors, the flight path angle is held constant at 90° until the programmed pitch kick is applied, ensuring stability in the simulation.

EOM_staged Function:

The complete dynamics of the launch are evaluated through the custom *eom_staged* function, which numerically integrates the governing equations of motion for a multi-stage vehicle. This function builds directly on the theoretical framework from the Curtis (2020) equations above, and the Lecture 5 Solver in the appendix. The functionality is outlined in *Figure 1 (Flowchart)*.

The state vector is defined as:

$$y = [v, h, x, \gamma, p_0 \dots p_{N-1}, s_0 \dots s_{N-1}]$$

where v is velocity, h altitude, x downrange distance, γ flight path angle, p_i the remaining propellant in element i, and s_i a binary flag for whether that element's structure is still attached.

runsim_full Function:

The runsim_full routine integrates the two-dimensional point-mass rocket equations through liftoff, optional pitch over, and ascent until either ground impact or a set final time.

Segment A (Initial Integration):
Starts at *t = 0* with near-zero velocity, altitude, and downrange; $\gamma = 90°$. Active models: thrust, drag, gravity, atmosphere, mass depletion, staging. The gravity-turn law is held off ($\dot{\gamma} = 0$) until a kick trigger. Event monitors include:

- Kick altitude ($h \geq h_{kick}$)
- Ground contact ($h \leq 0$ on descent)
- Final time stops

Integration halts at the first event: reaching kick altitude (→ kick handling), hitting ground (terminate), or final time (terminate).

Kick Handling:
At h_kick, the state is continuous (mass, velocity preserved). Only γ is modified:

- *Instant kick*: discrete $\Delta\gamma$ applied in one step
- *Rate kick*: γ slewed smoothly over Δt_{kick}, with $|\dot{\gamma}| \leq \dot{\gamma}_{max}$

Both maintain continuity and enable gravity-turn dynamics immediately.

Segment B (Post-Kick):
Integration resumes with the guidance law:

$$\dot{\gamma} = -\left(\frac{g}{v} - \frac{v}{R_E + h}\right) cos\gamma,$$

staging logic, and termination at ground or final time.

Outputs:

- Segments A and B
- Kick metadata (event, mode, $\Delta\gamma$, $\dot{\gamma}_{max}$)
- Event stamps (SRB separation, cutoffs)

- Diagnostics: drag/gravity losses, max q, apoapsis/periapsis

Stability Features:
Ground events override kicks if near simultaneous. States evolve without resets. Gates prevent chatter at thresholds; γ limited to $\pm 170°$ with cosine safeguards. In rate mode, unfinished Δγ is carried forward.

This modular design supports both idealised (instant) and realistic (rate-limited) manoeuvres while meeting requirements to model drag, gravity losses, staging, and gravity turns for the Vulcan Centaur.

Interactive_Plot Function:

In addition to the core trajectory solver, an auxiliary function *interactive_plot* was implemented to improve post-processing and visualisation. This routine allows the user to select any stored simulation variable (e.g., velocity, altitude, dynamic pressure, flight path angle) for plotting against another, without modifying the code manually.

The function operates in a loop:

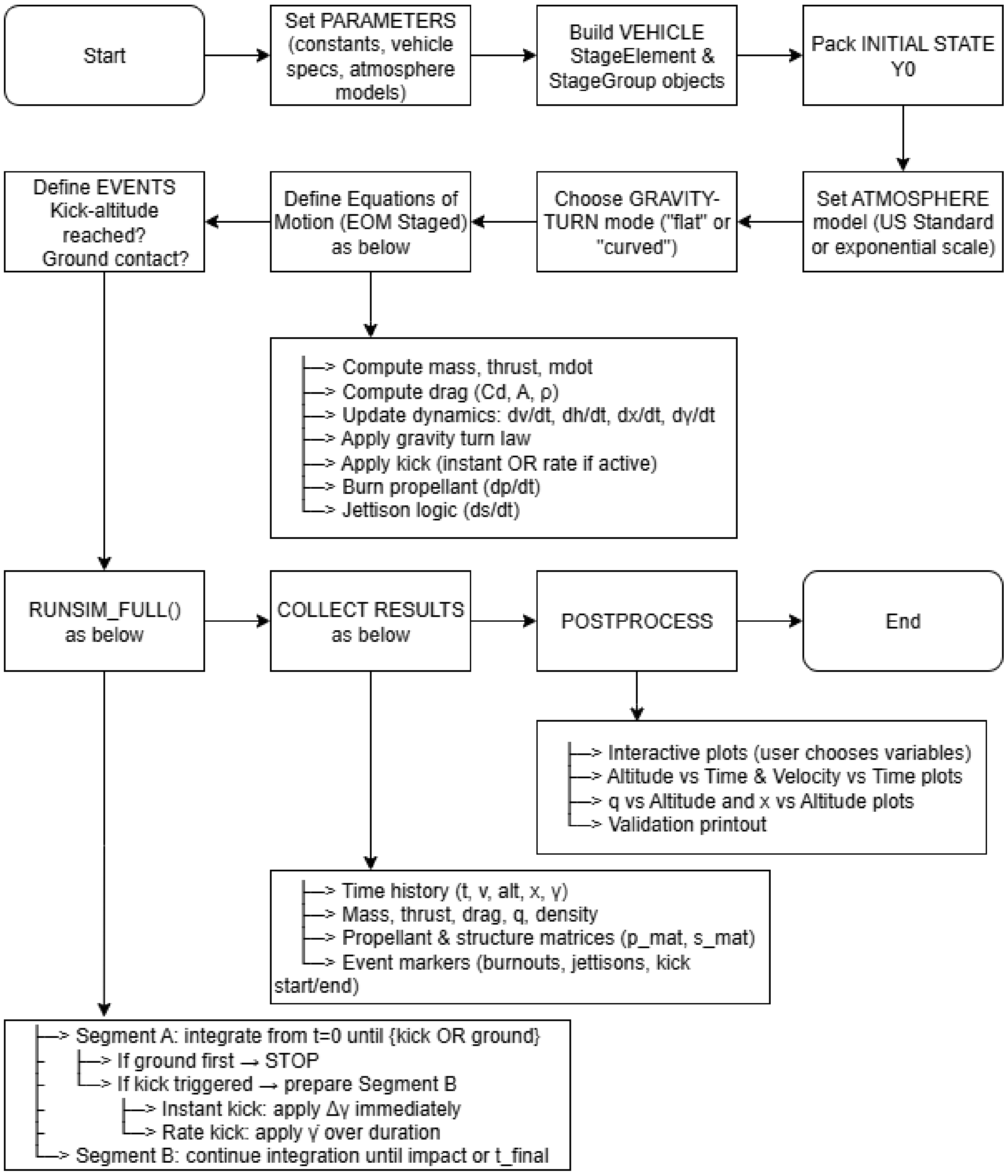
- It first prints a list of available keys in the dataset.
- The user specifies an x-axis variable (commonly time or downrange distance).
- One or more y-axis variables can then be selected for overlay on the same figure.
- Event markers such as kick altitude, stage separations, or fairing jettison are automatically added to the plot if enabled.

This interactivity makes it straightforward to explore relationships between quantities. Although not essential to the trajectory integration itself, this feature is a useful convenience tool for rapidly comparing different simulation outputs, diagnosing anomalies, and generating customised figures for reports.

Flowchart of Code Functions:

To clearly outline how the simulation operates, a flowchart of the code functions is presented in *Figure 1*. This diagram summarises the logical sequence of the trajectory solver, from defining initial parameters and vehicle properties through to the integration of the equations of motion, event handling, staging logic, and final post-processing of results. By breaking the code into modular steps, the flowchart highlights how each part of the solver interacts, ensuring that complex behaviours such as pitch-kicks, staging, and validation are implemented in a structured and transparent way.

Figure 1. Flowchart of trajectory simulation code functions


Start
Set PARAMETERS (constants, vehicle specs, atmosphere models)
Build VEHICLE StageElement & StageGroup objects
Pack INITIAL STATE Y0
Set ATMOSPHERE model (US Standard or exponential scale)
Choose GRAVITY-TURN mode ("flat" or "curved")
Define Equations of Motion (EOM Staged) as below
Define EVENTS Kick-altitude reached? Ground contact?
Compute mass, thrust, mdot
Compute drag (Cd, A, ρ)
Update dynamics: dv/dt, dh/dt, dx/dt, dγ/dt
Apply gravity turn law
Apply kick (instant OR rate if active)
Burn propellant (dp/dt)
Jettison logic (ds/dt)
RUNSIM_FULL() as below
COLLECT RESULTS as below
POSTPROCESS
End
Interactive plots (user chooses variables)
Altitude vs Time & Velocity vs Time plots
q vs Altitude and x vs Altitude plots
Validation printout
Time history (t, v, alt, x, γ)
Mass, thrust, drag, q, density
Propellant & structure matrices (p_mat, s_mat)
Event markers (burnouts, jettisons, kick start/end)
Segment A: integrate from t=0 until {kick OR ground}
If ground first → STOP
If kick triggered → prepare Segment B
Instant kick: apply Δγ immediately
Rate kick: apply γ̇ over duration
Segment B: continue integration until impact or t_final

Task 2. Validation of Your 3DOF Trajectory Code

Introduction

In this section, the simulation code and equations of motion developed previously are applied to reproduce and compare against the worked examples presented by Curtis in Sections *13.2* and *13.3*. These textbook cases provide reference trajectories for simplified single-stage rockets, first in vacuum *(Example 13.2)* and then including atmospheric drag and a gravity-turn manoeuvre *(Example 13.3)*. By implementing the same assumptions and initial conditions within the numerical model, the results obtained here can be directly compared with Curtis' solutions, allowing the accuracy of the code to be validated and the influence of drag and gravity-turn dynamics to be assessed.

Example 13.2

Table 3 lists the expected burnout and apex conditions from Curtis Example 13.2, which were used as a benchmark for validating the trajectory code. The case assumes a vertical trajectory with no drag, constant gravity at sea-level $g_0 = 9.81\ m/s$, constant thrust, and a mass ratio of 7. Under these assumptions, the Python implementation reproduced the reference results with excellent accuracy: the simulated burnout time, altitude, velocity, and maximum altitude all matched the reference case within 0.05% error. This strong agreement confirms that the governing equations and the propellant mass depletion model were implemented correctly.

Table 3. Expected Burnout and Apex Results for Curtis Example 13.2 Validation Case

Variable	**Simulated**	**Reference**	**% Error**
Time to burnout	$238.8\ s$	$238.8\ s$	-0.000%
Burnout altitude	$337.7\ km$	$337.6\ km$	0.036%
Burnout speed	$5.102\ km/s$	$5.102\ km/s$	0.005%
Maximum altitude	$1664.6\ km$	$1664.6\ km$	-0.002%

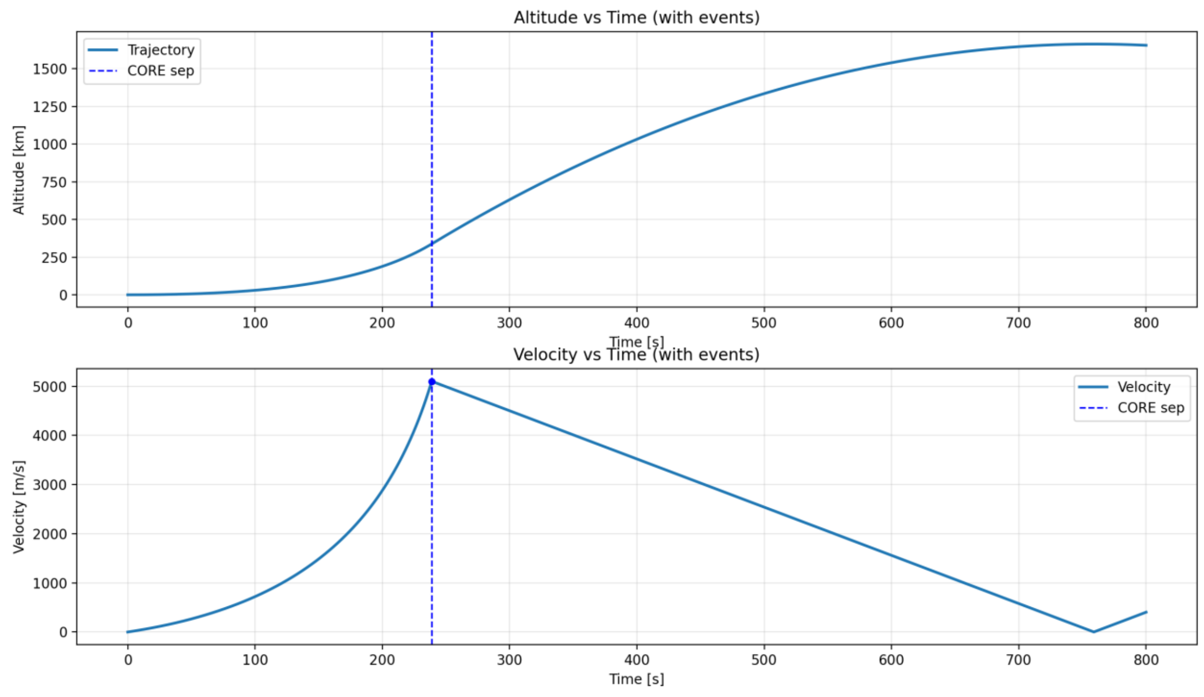


Figure 2. Trajectory validation against Curtis Example 13.2 (Developed)

Figure 2 illustrates the simulated altitude and velocity histories. The behaviour is consistent with that expected for a rocket in vacuum: a smooth increase in altitude and velocity during powered flight, followed by a coasting phase where altitude continues to rise while velocity decreases under gravity. The close agreement with the Curtis reference results provides strong validation of the developed code for this simplified single-stage case.

Example 13.3

At burnout, Curtis Example 13.3 predicts an altitude of 110.324 km, a velocity of 5.737 km/s, and a flight-path angle of 9.154°, with corresponding velocity losses of 0.298 km/s due to drag and 1.410 km/s due to gravity. These results are based on the assumptions of constant thrust, a fixed drag coefficient of 0.5, an exponential atmosphere with a scale height of 7.5 km, and variable gravity with altitude. The trajectory also incorporates a pitchover beginning at 130 m altitude with an initial flight path angle of 89.85°. These conditions provide the benchmark for validating the developed trajectory code under realistic launch scenarios.

Table 4. Expected Burnout and Loss Results for Curtis Example 13.3 Validation Case

Variable	**Simulated**	**Reference**	**% Error**
Altitude	$110.406\ km$	$110.324\ km$	0.074%
Speed	$5.737\ km/s$	$5.737\ km/s$	0.000%
Flight Path Angle	$9.166\ °$	$9.154\ °$	0.131%
Drag Loss	$0.298\ km/s$	$0.298\ km/s$	0.000%
Gravity Loss	$1.410\ km/s$	$1.410\ km/s$	0.000%

Table 4 compares the simulated burnout results against the Curtis reference values. The agreement is excellent, with errors below 0.2% for all parameters. This confirms that aerodynamic drag, an exponential atmosphere, altitude-dependent gravity, and gravity-turn dynamics have been implemented correctly. The close match demonstrates that the developed 3DOF model is capable of reliably reproducing standard textbook examples. *Figure 3* shows the plots of altitude and velocity versus time, highlighting the burnout condition, the application of the pitch kick, and subsequent trajectory evolution.

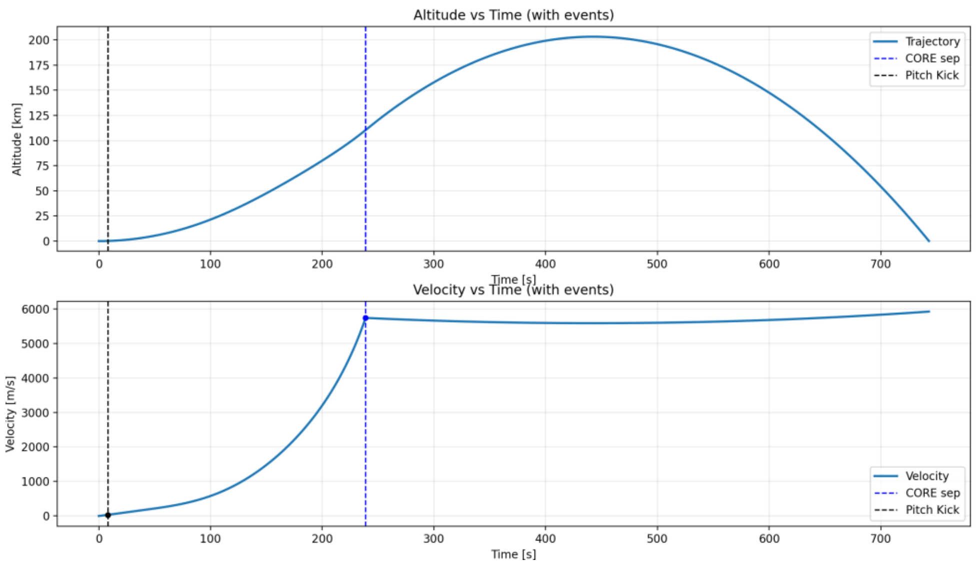


Figure 3. Altitude (km) and Velocity (m/s) vs Time (s) (Developed)

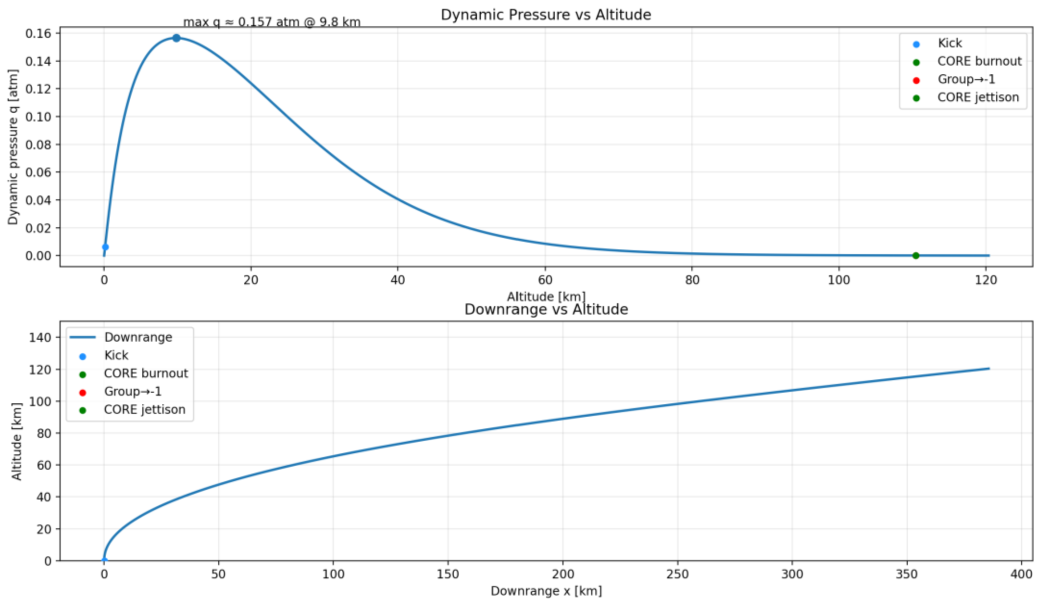


Figure 4. Dynamic pressure (atm) and Downrange (km) vs Altitude (km) (Developed)

The developed code (above) produces dynamic pressure and downrange–altitude trends that align very closely with the Curtis textbook example (below). Both solutions show a peak dynamic pressure of approximately 0.16 atm at around 9–10 km altitude, followed by a rapid decay as the vehicle ascends. The downrange–altitude profiles also match well, with a smooth increase in downrange distance and a final burnout altitude of ~110 km. These plots further confirm that the implemented 3DOF model accurately reproduces the Curtis Example 13.3 results.

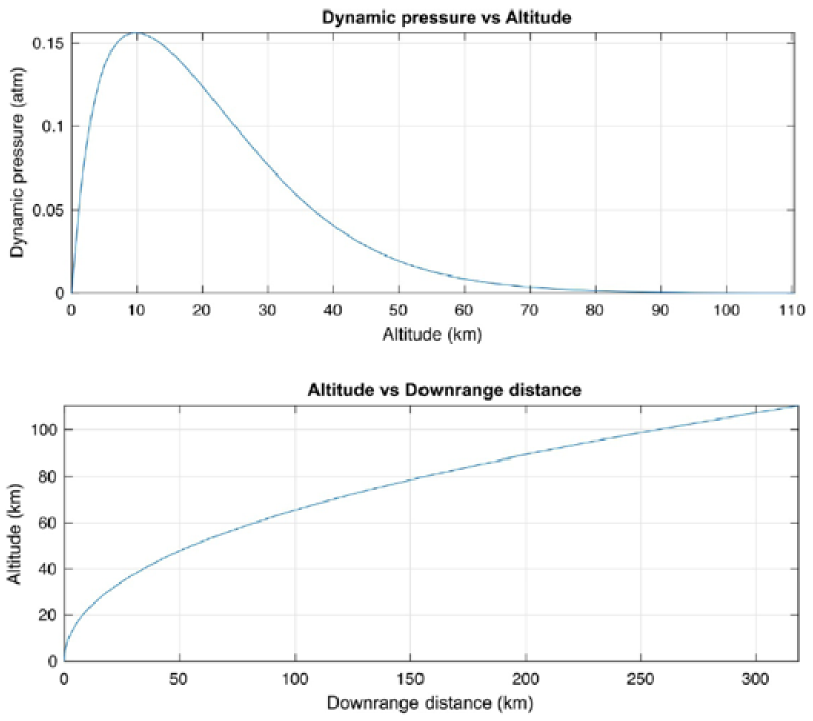


Figure 5. Dynamic pressure (atm) and Downrange (km) vs Altitude (km) (Curtis Textbook)

Staging validation:

To begin the validation, the engine parameters for a simplified two-stage configuration are listed in *Table 5*, assuming constant gravity, zero drag and a kick of 0 degrees.

Table 5. Engine Values for Simplified Centaur Launch

Engine	$Thrust\ [N]$	$I_{sp}\ [s]$	$m_{p,1}\ [kg]$	$m_{p,2}\ [kg]$
Core	$4.893 * 10^6$	330	350000	22000
Centaur	$0.2036 * 10^6$	453.8	20000	7100

The key mass calculations for the vehicle, after each burn and jettison event, are summarised below. From the thrust and specific impulse values, the effective exhaust velocity and mass flow rates for each stage can also be calculated.

Table 6. Stage Mass Calculations and Effective Velocity and Mass Flow Rates

Payload Mass:

$$m_{pl} = 20000\ kg$$

Calculating Total Mass:

$$m_{0,veh} = \left(m_{p,1} + m_{s,1}\right) + \left(m_{p,2} + m_{s,2}\right) + m_{pl} = 419100\ kg$$

After Core Burn + Jettison:

$$m_{0,2} = m_{f,1} - m_{s,1} = 47100\ kg$$

End of Centaur

$$m_{f,2} = m_{s,2} + m_{pl} = 27100\ kg$$

Stage	$c = I_{sp} g_0\ [m/s]$	$\dot{m} = \frac{T}{c}\ [kg/s]$
1	3237.3	1511.45
2	4451.78	45.735

Using these values, the analytical equations for burn time, burnout velocity, and burnout altitude can be applied to each stage to validate the trajectory model.

Table 7. Analytical Validation of Multi-Stage Sounding Rocket

Stage 1 (Core)

Burn Time:

$$t_{bo,1} = \frac{m_{0,veh} - m_{f,1}}{\dot{m}_1} = \frac{419100 - 69100}{1511.45} = 231.57$$

Burnout Velocity:

$$v_{bo,1} = c_1 \ln\left(\frac{m_{0,veh}}{m_{f,1}}\right) - g_0 t_{bo,1} = 3237.3 * \ln\left(\frac{419100}{69100}\right) - 9.81 * (231.57) = 3563.74\ m/s$$

Burnout Altitude:

$$h_{bo,1} = \frac{c_1}{\dot{m}_1}\left[m_{f,1} * \ln\frac{m_{f,1}}{m_{0,veh}} + m_{0,veh} - m_{f,1}\right] - 0.5 g_0 t_{bo,1}^2$$

$$= \frac{3237.7}{1511.45}\left[69100 * \ln\frac{69100}{419100} + 350000\right] - 0.5(9.91)(231.57)^2$$

$$= 219.85\ km$$

Stage 2 (Core)

Burn Time:

$$t_{bo,2} = \frac{m_{0,2} - m_{f,2}}{\dot{m}_2} = \frac{47100 - 27100}{45.735} = 437.31\ s$$

Burnout Velocity:

$$v_{bo,2} = v_{bo,1} + c_2 \ln\left(\frac{m_{0,2}}{m_{f,2}}\right) - g_0 t_{bo,2} = 3563.74 + 4451.78 * \ln\left(\frac{47100}{27100}\right) - 9.81 * (437.31)$$
$$= 2034.44\ m/s$$

Burnout Altitude:

$$h_{bo,1} = v_{bo,1} t_{bo,2} \frac{c_2}{\dot{m}_2}\left[m_{f,2} * \ln\frac{m_{f,2}}{m_{0,2}} + m_{0,2} - m_{f,2}\right] - 0.5 g_0 t_{bo,2}^2$$
$$= (3563.74)(437.31) + \frac{4451.78}{45.735}\left[27100 * \ln\frac{27100}{47100} + 20000\right] - 0.5(0.59.81)(437.31)$$
$$= 1174.59\ km$$

Totals at End of Stage 2:

$$t_{bo,tot} = t_{b0,1} + t_{bo,2} = 668.87\ s$$
$$v_{bo,tot} = 2034.44\ m/s$$
$$h_{bo,tot} = h_{b0,1} + h_{bo,2} = 1394.44\ km$$

Coast to Apogee

Time to max altitude:

$$t_{max} = \frac{v_{bo,tot}}{g_0} + t_{bo,tot} = \frac{2034.44}{9.81} + 668.87 = 1005.68\ s$$

Maximum Altitude:

$$h_{max} = h_{bo,tot} + v_{bo,tot}\left(t_{max} - t_{bo,tot}\right) - 0.5 g_0 \left(t_{max} - t_{bo,tot}\right)^2$$
$$= 1394.44 + 2034.44(336.81) - 0.5(9.81)(336.81)^2$$
$$= 1683.30\ km$$

Table 8. Values from code

Event	Time (s)	Altitude (km)	Speed (km/s)	Mass
Liftoff mass	–	–	–	419,100 kg
CORE burnout	231.57	221.57	3.600	47,100 kg
CENTAUR burnout	668.88	1406.43	2.133	27,100 kg
Max altitude	1013.18	1768.01	0.000	27, 100 kg

Table 9. Analytically Calculated Values

Event	Time (s)	Altitude (km)	Speed (km/s)	Mass
Liftoff mass	–	–	–	419,100 kg
CORE burnout	231.57	219.85	3.563	47,100 kg
CENTAUR burnout	668.88	1394.44	2.034	27,100 kg
Max altitude	1005.68	1683.30	0.000	27, 100 kg

The tables above present a direct comparison between the numerical code outputs and the analytically calculated values for key flight events, allowing validation of the simulation against closed-form solutions. *Table 10* below calculates the percentage error between the values from the code, and the analytical values.

Table 10. Error Values

Event	**Time (% ERROR)**	**Altitude (% ERROR)**	**Speed (ERROR)**	**Mass (% ERROR)**
Liftoff mass	$-\ (n/a)$	$-\ (n/a)$	$-\ (n/a)$	0.00%
CORE burnout	0.00%	$+0.78\%$	$+0.037\ km/s$	0.00%
CENTAUR burnout	0.00%	$+0.86\%$	$+0.099\ km/s$	0.00%
Max altitude	$+0.75\%$	$+5.03\%$	$-\ (n/a)$	0.00%

This table summarises the percentage error between the analytical and numerical results, showing excellent agreement at stage burnouts (<1% error) and a slightly larger but still acceptable deviation at maximum altitude (~5%). This validates the correct staging behaviour of the code, with *Figure 6* below showing the altitude and velocity profiles over time, including the clear transitions at core burnout and Centaur separation.

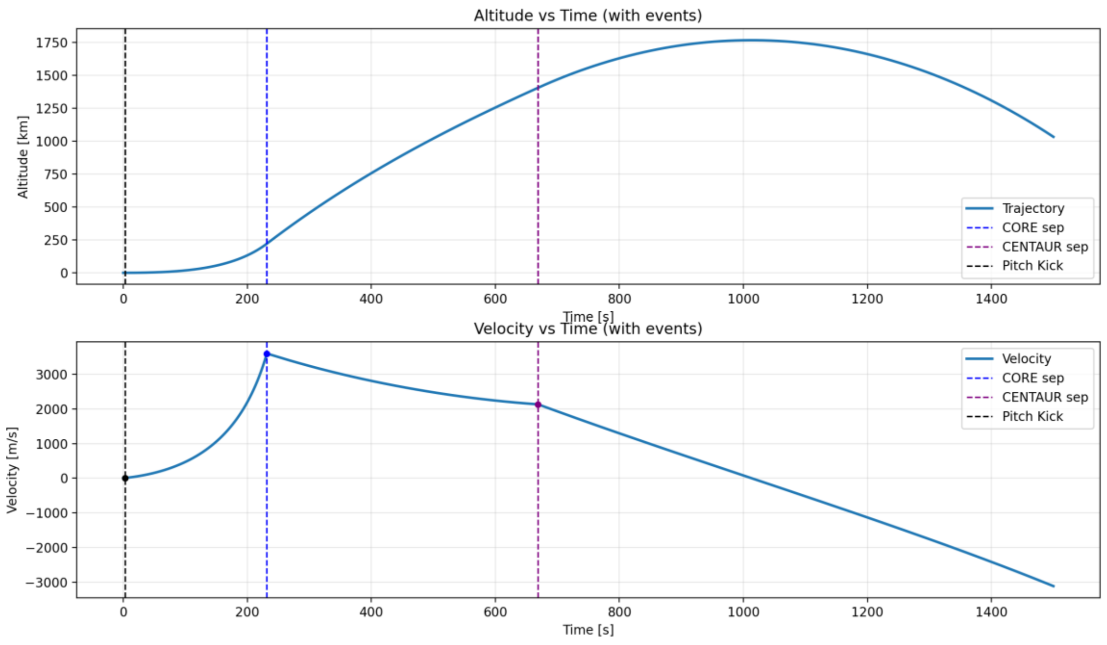


Figure 6. Altitude and velocity versus time for simplified 2 stage model

Conclusion:

Section 2 successfully validates the trajectory code by comparison with the Curtis textbook examples and the analytical multi-stage staging case. The close agreement in burnout times, velocities, and altitudes confirms that the implementation of the governing equations is correct, providing confidence that the code can be reliably applied to more complex launch vehicle simulations.

Task 3. Using Your Validated Trajectory Code to Launch a Rocket to Orbit

Introduction:

With the trajectory code validated against Curtis' textbook examples, the next step is to apply the model to a realistic launch vehicle. In this task, the Vulcan Centaur Launch System (VCLS) is simulated from liftoff through orbital insertion. The VCLS is United Launch Alliance's next-generation rocket, designed to succeed the Atlas V and Delta IV families, and can launch large payloads to Low Earth Orbit (LEO), Geostationary Transfer Orbit (GTO), and beyond.

The vehicle configuration consists of a first stage powered by two BE-4 methane-oxygen engines and augmented by optional solid rocket boosters, and a Centaur upper stage powered by two RL10 hydrogen-oxygen engines. The trajectory simulation incorporates parallel staging of the boosters, series staging of the core and Centaur stages, aerodynamic drag, variable gravity with altitude, and a gravity-turn manoeuvre to achieve orbital velocity.

The objective of this section is to demonstrate that the validated 3DOF model can be extended from simplified single-stage cases to a full multi-stage orbital launch, and to evaluate the performance of the Vulcan Centaur in reaching a 400 km circular Low Earth Orbit.

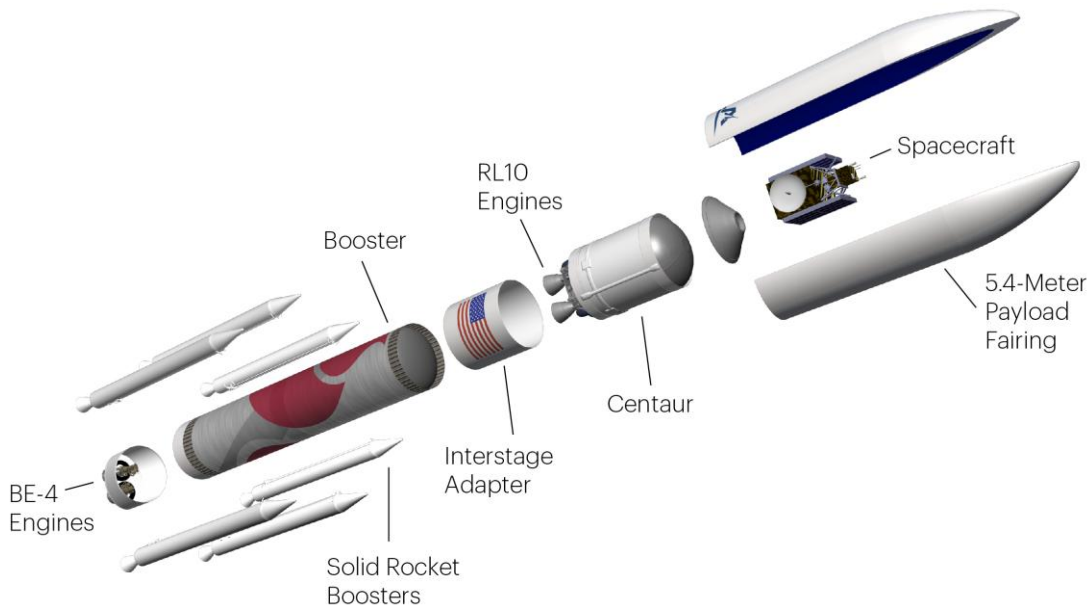


Figure 7. Exploded view of the Vulcan Centaur rocket from United Launch Alliance

Task 3 Initialisation:

The following constants were defined to initialise the Task 4 Vulcan Centaur trajectory simulation. The setup also relies on several modelling assumptions, which are listed below for clarity.

Key assumptions:

- Atmosphere model: U.S. Standard Atmosphere 1976 is applied up to 100 km, with density set to zero above this altitude.
- Gravity model: Gravity decreases with altitude.
- Flat-Earth vs. curved-Earth: A curved-Earth formulation is used to account for trajectory geometry.
- Aerodynamics: Constant drag coefficients are assumed, independent of Mach or Reynolds number.
- Pitch manoeuvre: A commanded rate-based pitch-kick is applied at 1 km altitude, following the defined rate and duration. These values were found through trial and error.
- Staging: Instantaneous propellant burnout and stage jettison events are assumed, with no residual propellant or separation delays.
- Environment: No winds, perturbations, or Earth rotation effects are included.

Table 11. Constant Values used for Task 4 Simulation

Parameter	Symbol	Value	Unit
Initial altitude	h_0	0.001	m
Initial velocity	v_0	0.001	m/s
Initial flight path angle	γ_0	90.0	$°$
Initial downrange	x_0	0.0	m
Kick altitude	h_{kick}	1000	m
Kick mode	–	$Rate$	–
Kick rate	$\dot{\gamma}$	2.34	$°/s$
Kick duration	–	11.995	s
Drag coefficient (core)	$C_{D\ core}$	0.32	–
Drag coefficient (upper)	$C_{D\ centaur}$	0.3	–
Sea-level density	ρ_0	1.225	kg/m^3
Core diameter	–	5.4	m
Reference area	A_{ref}	22.9	m^2
Payload mass	m_{PL}	20,000	kg

Table 11 summarises the initial conditions and environmental parameters applied in the trajectory model. It includes the initial altitude, velocity, and flight-path angle, along with pitch-kick settings, aerodynamic coefficients, and payload mass. These constants define the baseline setup for the Vulcan Centaur simulation.

Table 12. Centaur Rocket Engine Values

Stage	Thrust (N)	Isp (s)	Propellant Mass (kg)	Structure Mass (kg)
SRB (x6)	$2.061e6$	280.3	47853	4,521
Core	$4.900e6$	330.0	350,000	22,000
Centaur	$0.2036e6$	453.8	20,000	7,100

Table 12 lists the thrust, specific impulse, propellant mass, and structural mass for each stage of the launch vehicle. It includes six solid rocket boosters, the core stage, and the Centaur upper stage, providing the key propulsion inputs for staging and trajectory analysis. The engine groups were defined to suit the class system established earlier in the report, with the six SRBs and the core stage burning in parallel during the first phase of flight, followed by the Centaur upper stage burning in series once the lower stages are jettisoned.

```
# Parallel group 0: SRBs + CORE burning together
group1 = [SRB, SRB, SRB, SRB, SRB, SRB, CORE]
# Series group 1: upper stage burns after group0 is fully jettisoned
group2 = [CENTAUR]
SERIES_GROUPS = [group1, group2]
```

Results:

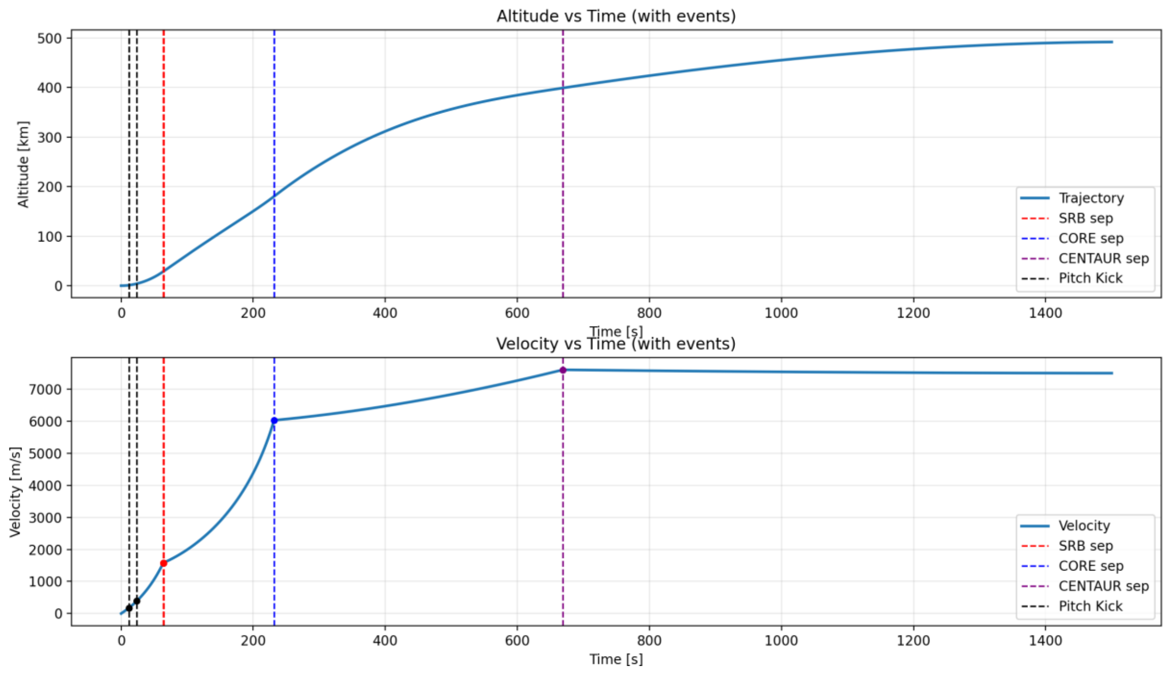


Figure 8. Altitude (km) and Velocity (m/s) vs Time (s) for Centaur Rocket

The trajectory and velocity history of the Vulcan Centaur launch vehicle are shown in *Figure 8*. The top plot illustrates the altitude growth with time, while the bottom plot shows the corresponding velocity history. Both curves include key staging events: SRB separation, core separation, Centaur separation, and the pitch kick manoeuvre. As expected, the vehicle rapidly gains altitude during the first few hundred seconds while the SRB's and core are active, followed by a more gradual increase under Centaur propulsion. After Centaur burnout, the vehicle continues to coast upwards, reaching its apogee before descending again.

The velocity profile mirrors this behaviour. Velocity increases steadily during each powered stage, with the largest increments achieved during core and Centaur operation. Following Centaur cutoff, the velocity levels off and remains nearly constant until orbital apogee, after which gravitational effects begin to dominate.

Table 13. Kick Events, Burnouts, and Jettisons During Ascent

Event	**Time (s)**	**Altitude (km)**	**Downrange (km)**	**Speed (km/s)**	**γ (°)**	**Mass (kg)**
Liftoff mass	–	–	–	–	–	733,344
Kick start (rate 2.34°/s)	11.75	1.00	0.0	0.175	90.00	662,752
Kick end	23.75	4.12	1.0	0.390	55.84	590,636
SRB jettison	63.96	28.80	28.1	1.581	36.44	322,432
CORE jettison	231.69	180.45	526.0	6.026	9.87	47,100
CENTAUR jettison	669.01	399.21	3444.3	7.600	1.51	27,100

Table 13 summarises the key events, burnouts, and jettisons during the ascent of the Vulcan Centaur. The results confirm that the trajectory code correctly models staging, thrust cutoffs, and mass depletion throughout flight. After liftoff, a pitch kick begins at ~11.8 s, initiating the gravity turn. The SRBs burn out and separate around 64 s, reducing the vehicle mass from 733,344 kg to 322,432 kg. Core burnout and separation occur at ~232 s, leaving 47,100 kg, while the Centaur upper stage provides the final Δv until ~669 s, after which only 27,100 kg remain—consistent with the dry mass plus payload.

The conditions for LEO Insertion occur at the Centaur jettison, where the vehicle reaches 399.21 km altitude at 669.01 s. The velocity at this point is 7.600 km/s with a flight path angle of 1.51°, representing near-horizontal motion suitable for orbit. The final mass of 27,100 kg matches the expected post-burnout value. These results place the orbital insertion altitude within 0.2% of the 400 km target and the velocity within 2.6% of the 7.8 km/s requirement—both comfortably inside the ±5% tolerance.

Figure 9 below shows the trajectory which exhibits a smooth gravity turn, with apogee near 400 km and several thousand kilometres downrange. Dynamic pressure peaks at ~0.8 atm around 15–20 km altitude before falling rapidly as the atmosphere thins, consistent with realistic max-q behaviour. The flight path angle decreases from ~90° at liftoff to near horizontal by Centaur cutoff, with staging markers aligning closely to the natural γ trend. These features confirm the expected behaviour for orbital insertion.

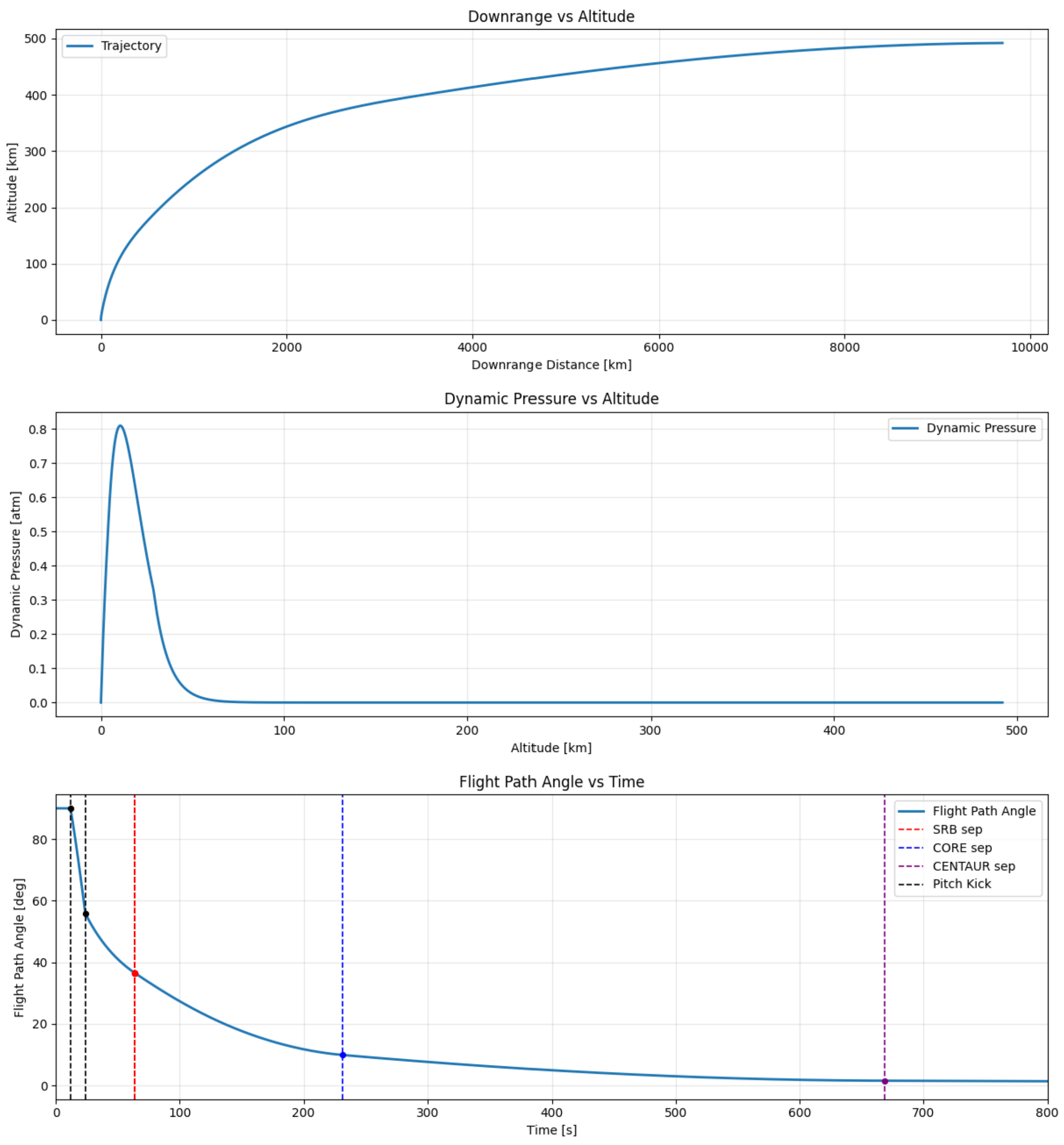


Figure 9. Key trajectory parameters showing (top) altitude versus downrange distance, (middle) dynamic pressure versus altitude, and (bottom) flight path angle versus time with staging events marked.

Conclusion:

Section 3 clearly fulfils the required task by presenting and analysing the key trajectory parameters of the ascent. The results illustrate a realistic gravity turn, accurate staging events, and physically consistent profiles of altitude, velocity, dynamic pressure, and flight path angle. Together, these plots demonstrate that the simulated trajectory behaves as expected for a launch vehicle targeting low Earth orbit.

Reference List:

1. Curtis, H. D. (2020). *Orbital mechanics for engineering students* (4th ed.). Elsevier. https://doi.org/10.1016/B978-0-08-102133-0.00013-1
2. United Launch Alliance. (2023). *Vulcan Centaur payload user's guide*. ULA. https://www.ulalaunch.com/docs/default-source/vulcan/vulcan-centaur-payload-user's-guide.pdf
3. James, C. (2025). *Lecture 5: Launch vehicles 2* [Lecture slides]. University of Queensland.

Appendix:

Lecture 5: Launch Vehicles 2: Time Resolved and Numerical Solutions Code (Put in Functions)

```python
import time
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# === Define the ODE system ===
def f(t, y, m_0, m_f, m_dot_e, T, g_0):
    m = m_0 - m_dot_e * t
    v = y[0]
    if m >= m_f:
        dv_dt = T / m - g_0
    else:
        dv_dt = -g_0

    dh_dt = v
    dy_dt = [dv_dt, dh_dt]
    return dy_dt

# === Parameters from the example ===
m_0 = 68000.0             # Initial mass [kg]
n = 7                     # Mass ratio
m_f = m_0 / n             # Final mass [kg]
T = 933910.0              # Thrust [N]
I_sp = 390.0              # Specific impulse [s]
g_0 = 9.81                # Gravitational acceleration [m/s^2]
m_dot_e = T / (I_sp * g_0)  # Propellant mass flow rate [kg/s]

# === Initial conditions ===
initial_time = 0.0        # [s]
final_time = 400.0       # [s]
initial_conditions = [0.0, 0.0]  # [v0, h0]
steps_to_evaluate = 10000
rtol = 1e-6
atol = 1e-6

def runsim(m_0, m_f, m_dot_e, T, g_0):

# === Run simulation ===
    start_time = time.perf_counter()

    solution = solve_ivp(lambda t, y: f(t, y, m_0, m_f, m_dot_e, T, g_0),
                         t_span=[initial_time, final_time],
                         y0=initial_conditions,
                         method='RK45',
                         dense_output=True,
                         rtol=rtol,
                         atol=atol,
                         t_eval=np.linspace(initial_time, final_time,
steps_to_evaluate))

    end_time = time.perf_counter()

    # === Extract results ===
    time_results = solution.t
    altitude = solution.y[1]
    velocity = solution.y[0]

    # Only keep data where altitude > 0
    valid_indices = altitude > 0

    # Filter arrays
    time_results = time_results[valid_indices]
    altitude_results = altitude[valid_indices]
    velocity_results = velocity[valid_indices]

    # === Find burnout and max altitude ===
    max_velocity = max(velocity_results)
    max_velocity_index = list(velocity_results).index(max_velocity)
    burnout_time = time_results[max_velocity_index]
    burnout_altitude = altitude_results[max_velocity_index]

    maximum_altitude = max(altitude_results)
    maximum_altitude_index =
list(altitude_results).index(maximum_altitude)
    maximum_altitude_time = time_results[maximum_altitude_index]

    # === Output ===
    print(f"Calculation took {(end_time - start_time)*1000.0:.2f} ms")
    print(f"rtol = {rtol:.0e}, atol = {atol:.0e}")
    print(f"Burn out velocity is {max_velocity:.1f} m/s")
    print(f"Burn out time is {burnout_time:.1f} s")
    print(f"Burn out altitude is {burnout_altitude/1000.0:.1f} km")
    print(f"Maximum altitude is {maximum_altitude/1000.0:.1f} km")
    print(f"Maximum altitude time is {maximum_altitude_time:.1f} s")

    # === Plotting ===
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(time_results, velocity_results)
    plt.xlabel('Time [s]')
    plt.ylabel('Velocity [m/s]')
    plt.title('Rocket Velocity vs Time')
    plt.subplot(1, 2, 2)
    plt.plot(time_results, altitude_results)
    plt.xlabel('Time [s]')
    plt.ylabel('Altitude [km]')
    plt.title('Rocket Altitude vs Time')
    plt.tight_layout()
    plt.show()

    return

runsim(m_0, m_f, m_dot_e, T, g_0)
```