Arianna Abramovich
ASTE404

Mini Project - Numerical Integration of Atmospheric Re-entry Thermal Loads

Project Overview

This mini project focuses on building a simplified numerical model of atmospheric re-entry heating using a set of coupled ordinary differential equations (ODEs). The goal is to simulate how a spacecraft's velocity, altitude, and thermal load evolve as it descends through an atmosphere. The core of the project is a Python based integrator that computes drag, gravitational acceleration, stagnation-point heat flux, and accumulated thermal energy as a function of time.

Understanding re-entry heating is essential in astronautics because thermal protection systems must be designed to withstand extreme aerothermal environments. Even a simplified model provides valuable insight into why heating spikes at mid-altitudes, how velocity and atmospheric density interact, and what drives the thermal loads that spacecraft experience during high-speed entry. By modeling these relationships numerically, the project illustrates the fundamental physics governing re-entry and demonstrates how numerical integration can be used to predict trends in temperature and heating rate.

The primary question this project aims to answer is: How do velocity, atmospheric density, and altitude collectively influence the heating experienced by a spacecraft during atmospheric re-entry? By integrating the equations of motion and the Sutton-Graves heating correlation, the simulation reveals how heating increases as the vehicle accelerates into denser air and then decreases as drag slows the descent. The resulting plots, particularly temperature and heat flux versus altitude, provide a clear visualization of the aerothermal behavior that defines re-entry trajectories.

Numerical Methods

This project models atmospheric re-entry using a system of coupled ordinary differential equations (ODEs) describing altitude, velocity, and heating. The governing equations combine gravitational acceleration, aerodynamic drag, an exponential atmospheric model, and the Sutton-Graves stagnation-point heating correlation.

Governing Equations

A one-dimensional vertical descent is assumed with velocity defined as positive downward. The system of ODES is:

1. Vertical Motion

$$\frac{dh}{dt} = \quad - V$$

where $\frac{dh}{dt}$ is the change in altitude over time and V is velocity. Here, velocity is negative because the altitude is decreasing as velocity increases. The vehicle falls as long as $V > 0$.

2. Velocity evolution with drag and gravity

$$\frac{dV}{dt} = g - \frac{1}{2}\frac{C_d A}{m}\rho(h)V^2$$

where g is the gravitational acceleration, C_d is the drag coefficient (assumed to be constant), A is the reference area of the spacecraft, m is the mass and $\rho(h) = \rho_0 e^{-h/H}$ is the atmospheric density as a function of altitude where ρ_0 is the sea level density, h is altitude, and H is the scale height.

3. Stagnation-point heat rate (Sutton-Graves)

$$\dot{q} = k\sqrt{\rho(h)}V^3$$

where $\dot{q}$ is the stagnation-point convective heat flux and k is the Sutton-Graves coefficient (1.83 x 10^{-4} for Earth in SI units). The square root dependence of density comes from boundary layer similarity analysis where higher density means more particles means more energy transfer. The cubic velocity dependence is why re-entry heating increases so rapidly and peaks before the vehicle significantly slows down.

4. Accumulated thermal load or "effective temperature"

$$\frac{dT}{dt} = \dot{q}$$

Together, these equations define a nonlinear, altitude-dependent system

$$\frac{d}{dt}\begin{bmatrix} h \\ V \\ T \end{bmatrix} = \begin{bmatrix} -V \\ g - \frac{1}{2}\frac{C_D A}{m}\rho_0 e^{-h/H} V^2 \\ k\sqrt{\rho_0 e^{-h/H}}\, V^3 \end{bmatrix}$$

Discretization and Algorithm

The system is solved numerically using the fourth order Runge-Kutta (RK4) time integration method. RK4 is chosen because it offers significantly higher accuracy than Euler's method for only a moderate increase in computational cost, which is appropriate for stiff or nonlinear systems such as re-entry dynamics. For re-entry, things change fast and therefore we need a method that can adapt as such.

For a state vector, $y = [h, V, T]$, and ODE function $f(t, y)$, the RK4 update from timestep n to $n + 1$ is:

```
Given y_n (h, V, T) at time t_n:

k1 = f(t_n, y_n)
k2 = f(t_n + dt/2, y_n + dt/2 * k1)
k3 = f(t_n + dt/2, y_n + dt/2 * k2)
k4 = f(t_n + dt,   y_n + dt * k3)

y_(n+1) = y_n + (dt/6)*(k1 + 2*k2 + 2*k3 + k4)
```

This algorithm is applied at each timestep until the spacecraft reaches a low-altitude cutoff (e.g. 1 km) or the velocity approaches zero.

Essentially what's happening is with a given initial condition, k1 provides the slope at the start of the timestep, k2 is the slope halfway through using k1 to estimate position, k3 is another midpoint estimate improving accuracy, and k4 is the slope at the end of the timestep. The last line is the weighted average (start + 2 midpoints + end).

Stability, Accuracy, and Limitations

Stability: fourth order Runge-Kutta

RK4 is conditionally stable, meaning that a large time step can overshoot peak heating, miss deceleration, and produce nonphysical behavior. Re-entry is challenging because atmospheric density changes exponentially, heating increases extremely fast, and stiffness increases at lower altitude. Instability would look like temperature suddenly spiking unrealistically, velocity oscillating instead of smoothly decreasing, and heat flux going negative or blowing up. This happens when the time step is too large, which is talked about next.

Accuracy: Time step considerations

Numerical accuracy is governed primarily by the chosen timestep. Because drag and heating depend nonlinearly on velocity, coarse timesteps can under resolve rapid changes near peak heating. To ensure accuracy, a time step refinement study is performed (i.e. Δt = 5.0s → 2.0s → 0.5s → 0.25s), and convergence of velocity and heat-flux curves is checked.

Limitations: model assumptions

Several assumptions are made to simplify the overall model and are outlined below

- 1D motion; no lift, no angle of attack, no skip trajectories

- Simplified atmosphere: exponential model with no weather and no real stratification
- Heating model: Sutton-Graves is empirical, no radiation cooling, no thermal protection system material response
- Thermal model: “temperature” is integrated heat load, not a real surface or internal temperature – units [J/m^2]
- Numerical: fixed time step, explicit method, can struggle with extreme stiffness

Put simply, this model is intentionally simplified and is intended to capture first-order trends rather than predict flight-qualified thermal loads. The simulation assumes one-dimensional motion, a constant drag coefficient, an exponential atmosphere, and an empirical heating correlation without radiation or material response. The use of an explicit fixed-timestep integrator limits performance in highly stiff regimes. Despite these simplifications, the model successfully reproduces the qualitative behavior of peak heating during atmospheric re-entry.

Implementation Notes

Software Structure

```
Numerical-Integration-of-Atmospheric-Re-Entry-Thermal-Loads/
│
├── src/
│   ├── model.py
│   ├── integrator.py
│   ├── main.py
│
├── plots/
│   ├── ThermalLoad_vs_Altitude.png            ← 4 versions of this file
│   ├── ThermalLoad_vs_Velocity_Altitude.png   ← 4 versions of this file
│   ├── Velocity_vs_Altitude.png               ← 4 versions of this file
│   ├── Velocity_vs_Time.png                   ← 4 versions of this file
│
├── README.md
└── report.pdf
```

What everything does

Model: contains ODEs and the physics

Integrator: RK4 integrator

Main: runs full simulation and generates plots

Plots: create respective plots of deliverables
README: how to run, structure, dependencies
Report: final write-up

Progress Log

12/11/2025

- Brainstormed an idea for the project and created my Github repository, cloning it within Visual Studios where I will be working

12/12/2025

- Begin the project overview and numerical methods portion of my report before starting my code so I understood what I would be doing and how I would accomplish it
- Establish my workflow in Github so I knew my deliverables and have just an overall "outline"

12/13/2025

- Updated my READ.me with all the python packages I used so I could keep track as I went through the assignment
- Slightly altered the skeleton code for integrator.py, model.py, and main.py and added comments
- Hit ran and had nothing show up; no output, no error, no graphs
- Debugged the error in which my code was being stuck in an infinite loop because my time integrator was not inside the loop
- Code ran successfully and made the 2 initial graphs

12/14/2025

- Added velocity vs time graph showing how drag affects the spacecraft and thermal load graph vs altitude vs velocity to show the gradient and trajectory of the s/c
- Got nit picky with my graphs and labeled them to show points of interest
- Changed the time steps and saved those plots to show the difference
- Uploaded all the graphs into my plots folder in github

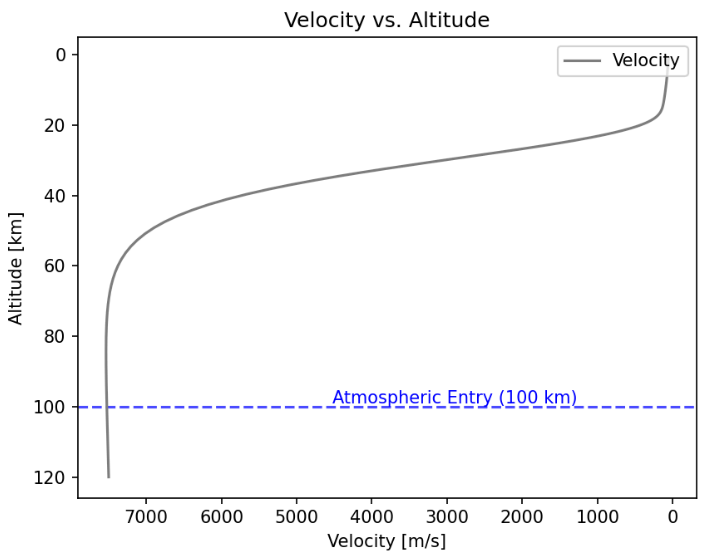
12/15/2025

- updated/finished READ.me instructions
- Uploaded this pdf into github

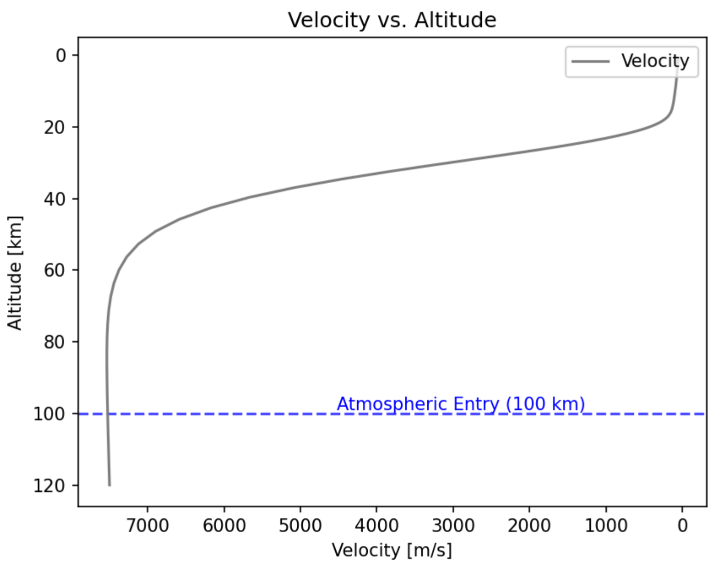
Verification/validation Evidence

Vehicle parameters, initial conditions, and timestep size are parameters that are user-defined and can be modified to investigate numerical accuracy. Across all three trials, I kept drag coefficient, reference area, vehicle mass, altitude from Earth's surface, and downward velocity constant. To test verification of the fourth order Runge-Kutta convergence, I ran the simulation for four different time steps: 0.25, 0.5, 2, and 5. Below are all the results side by side for the tree time steps with all the produced plots.

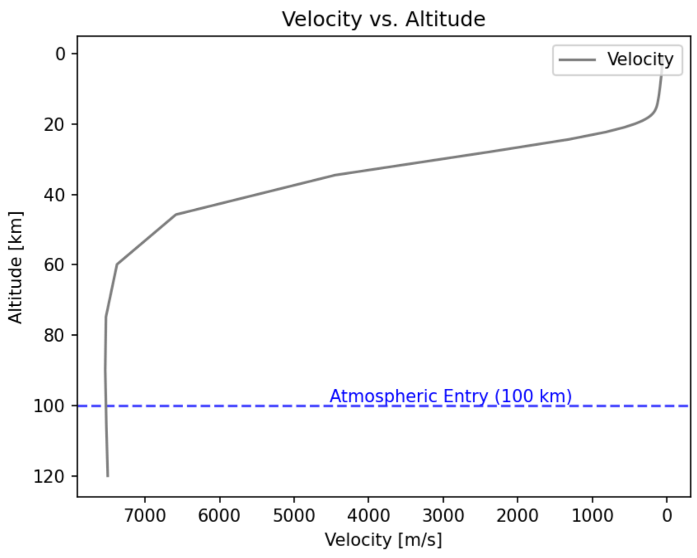
Velocity vs. Altitude



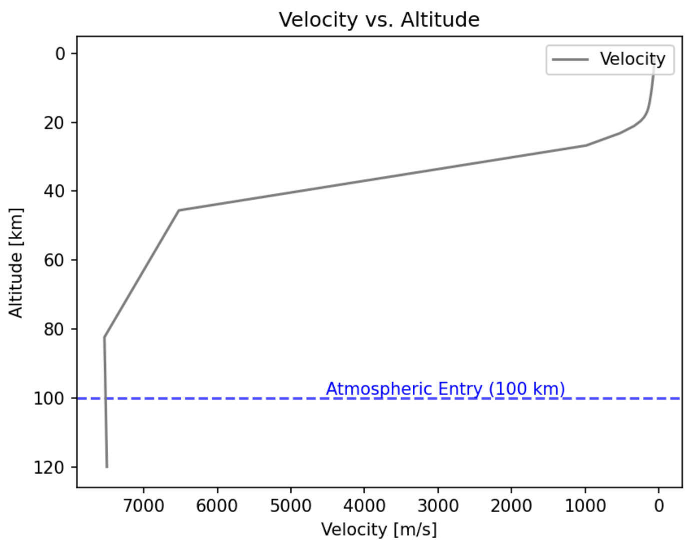
dt = 0.25



dt = 0.5

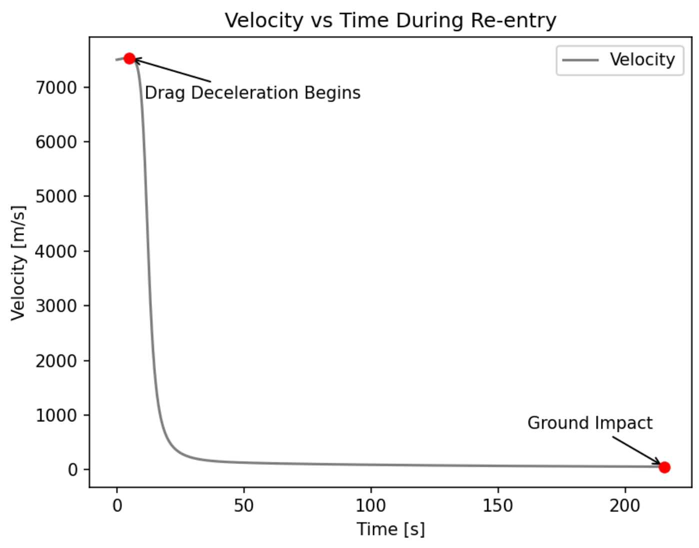


dt = 2

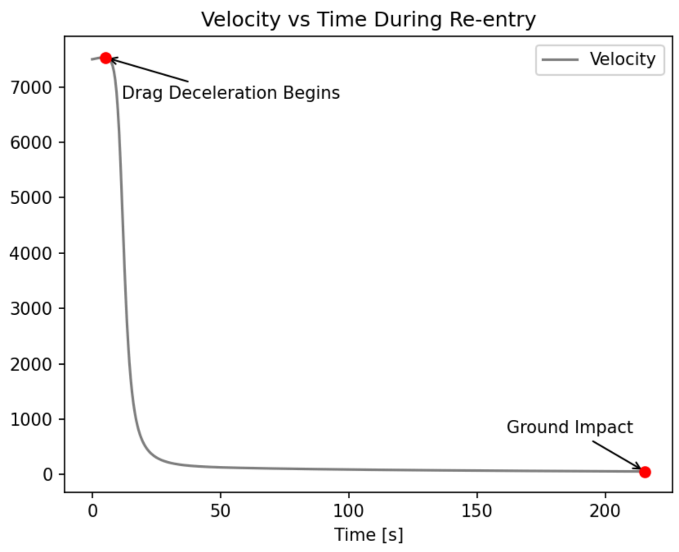


dt = 5

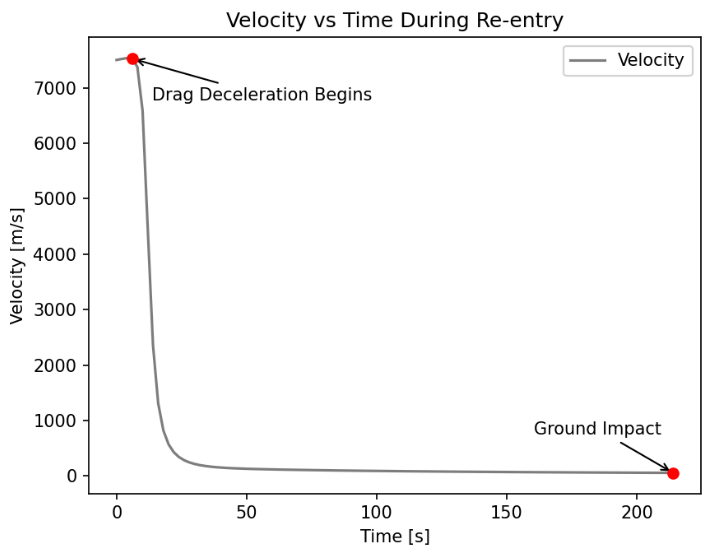
Velocity vs Time during re-entry



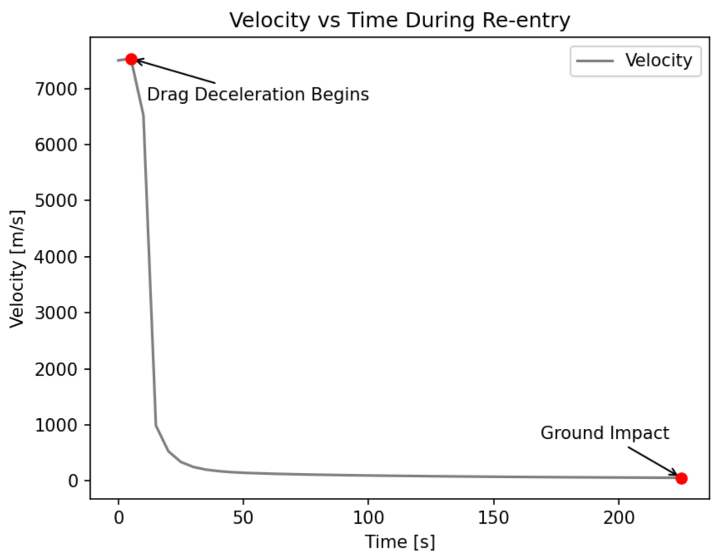
dt = 0.25



dt = 0.5

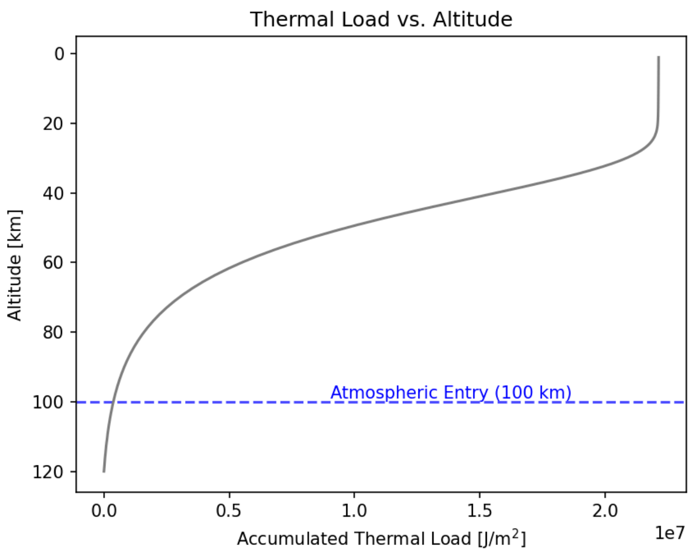


dt = 2

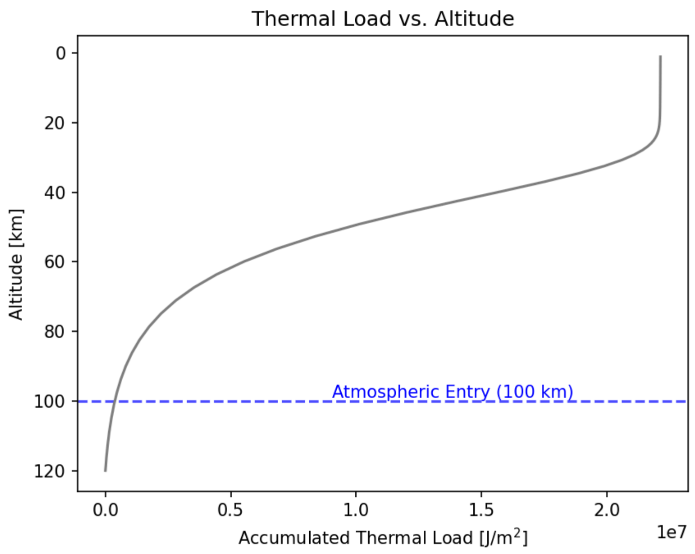


dt = 5

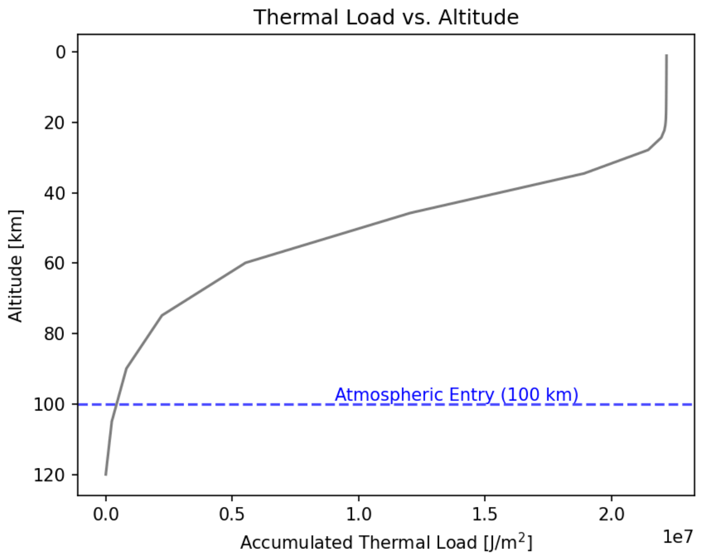
Thermal Load vs. Altitude



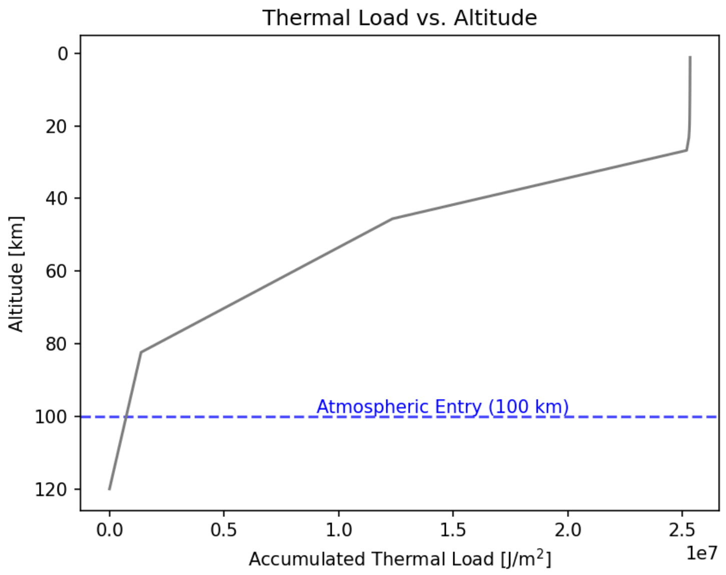
dt = 0.25



dt = 0.5

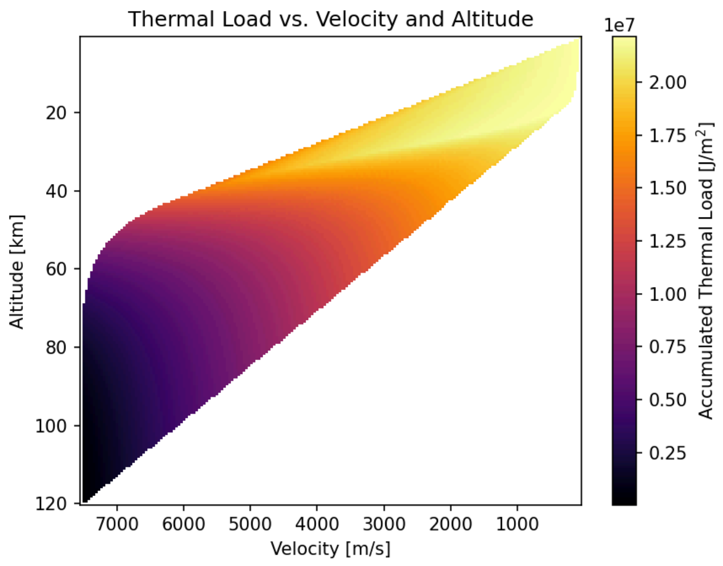


dt = 2

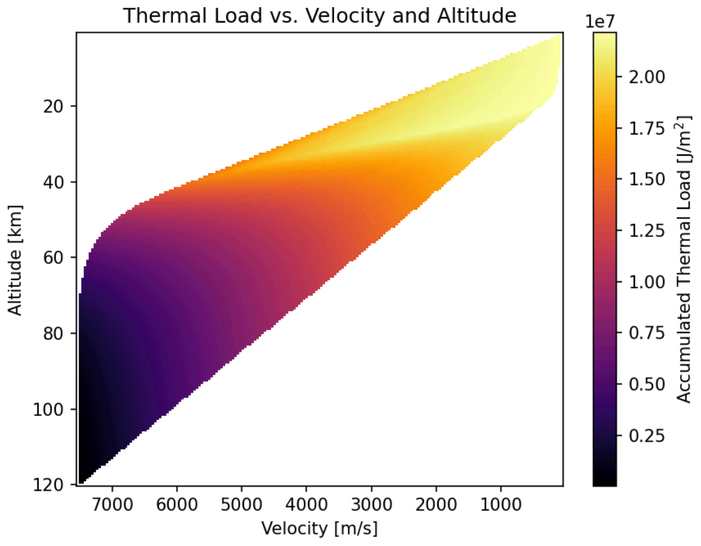


dt = 5

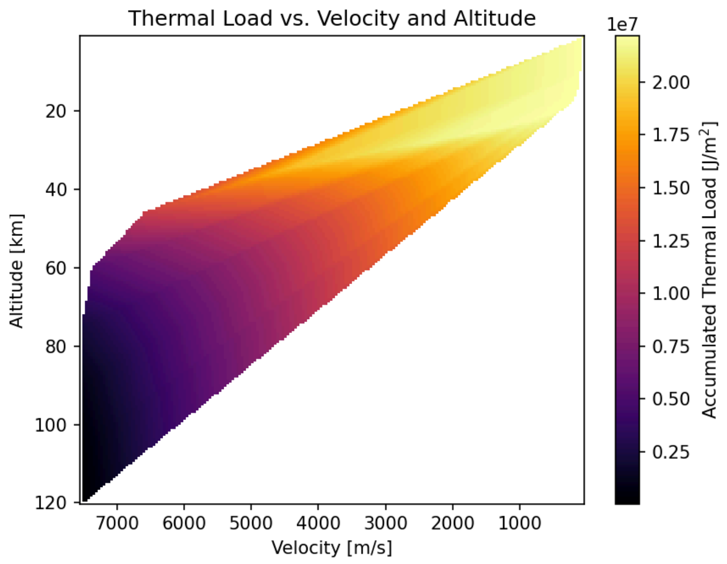
Thermal Load vs Velocity and Altitude



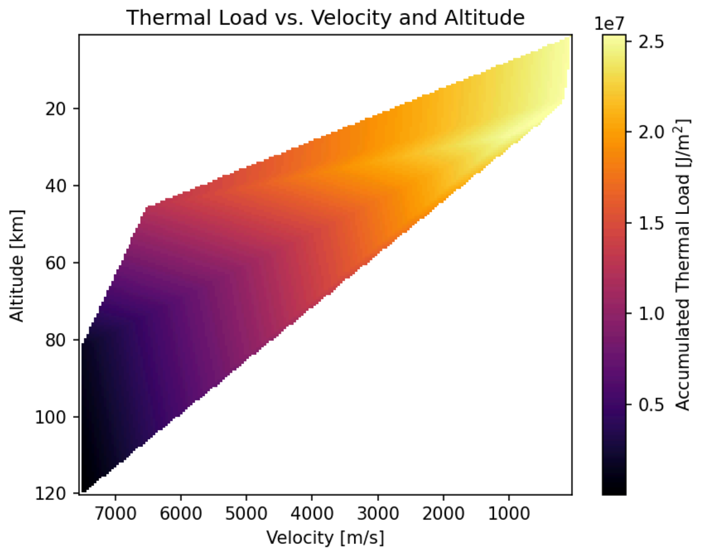
dt = 0.25



dt = 0.5



dt = 2



dt = 5

As clearly seen in the resulting plots, the choice of time step size significantly affects the accuracy and smoothness of the numerical solution. Larger time steps introduce greater truncation error in the Runge-Kutta integration, which can cause the solution to deviate from the true continuous dynamics. This manifests in the results as sharper transitions and less smooth curves in velocity, altitude, and thermal load histories. In extreme cases, large time steps may under-resolve rapid changes in drag and heating that occur at lower altitudes, leading to inaccurate peak values or shifted event timing. When experimenting with the code after its completion, using timesteps ≥ 10 “broke” the code and showed this under-resolved behavior and misconstrued data on the resulting plots.

Conversely, smaller time steps provide improved resolution and reduce numerical error, producing smoother and more physically realistic trajectories. The smoother curves observed at smaller time steps indicate better capture of the rapid changes in aerodynamic forces and heating rates during atmospheric entry. However, this increased accuracy comes at the cost of additional computational time, highlighting the trade-off between numerical accuracy and computational efficiency inherent in time-marching simulations.

Results

With my chosen initial conditions of:

$C_d = 1.2$

$A = 10\ m^2$

$m = 2000$ kg

$h_0 = 120{,}000$ m

$V_0 = 7500$ m/s

$T_0 = 0\ J/m^2$

and a time step of 0.25s, I obtained the following results.

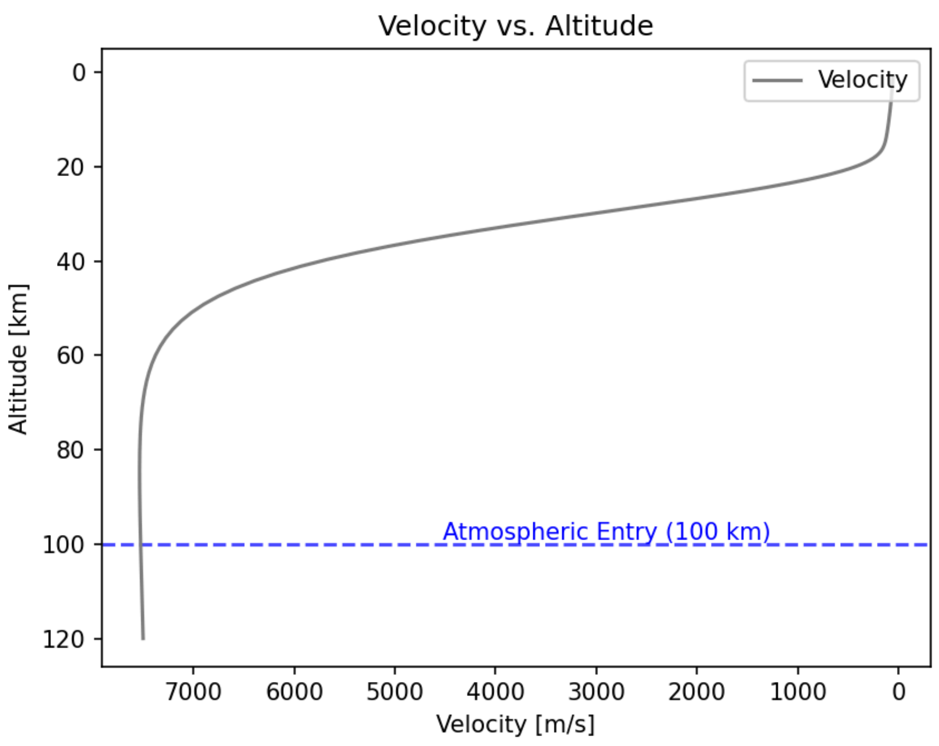


Figure 1: Vehicle velocity as a function of altitude during atmospheric re-entry

The spacecraft initially enters the atmosphere at approximately 100 km altitude with a high downward velocity. As altitude decreases, atmospheric density increases, leading to rapidly growing aerodynamic drag that causes a significant reduction in velocity. The sharp deceleration at lower altitudes highlights the transition from near-vacuum motion to a drag-dominated regime.

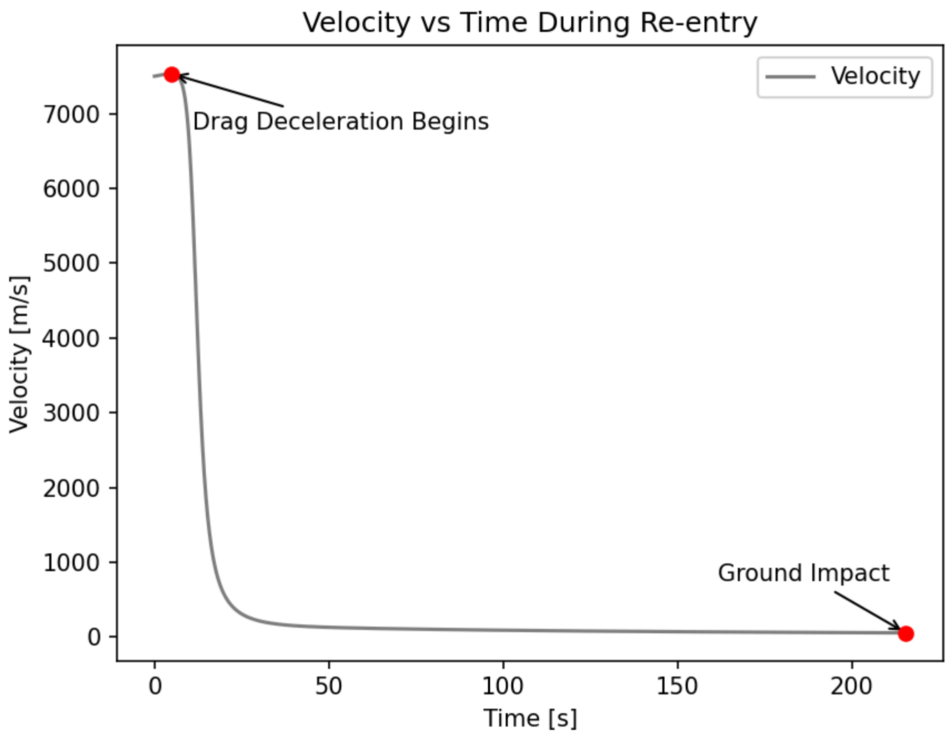


Figure 2: Time history of vehicle velocity slowing due to drag during atmospheric re-entry

Velocity initially decreases slowly while atmospheric density remains low, then drops rapidly as the vehicle descends into denser layers of the atmosphere. The steep decline indicates the point at which aerodynamic drag overtakes gravitational acceleration, resulting in strong deceleration prior to ground impact.

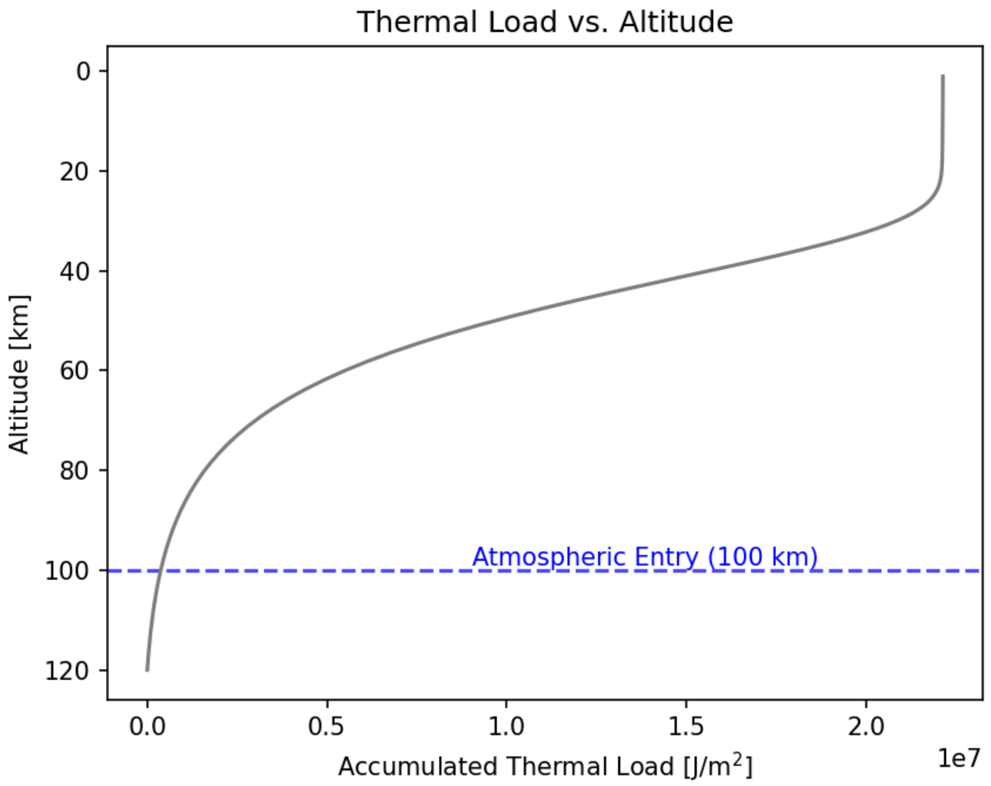


Figure 3: Accumulated thermal load as a function of altitude during re-entry

Thermal load increases gradually at high altitudes and rises sharply at lower altitudes as both velocity and atmospheric density increase. This behavior reflects the strong dependence of aerodynamic heating on velocity and density, with the majority of heat accumulation occurring in the denser lower atmosphere.

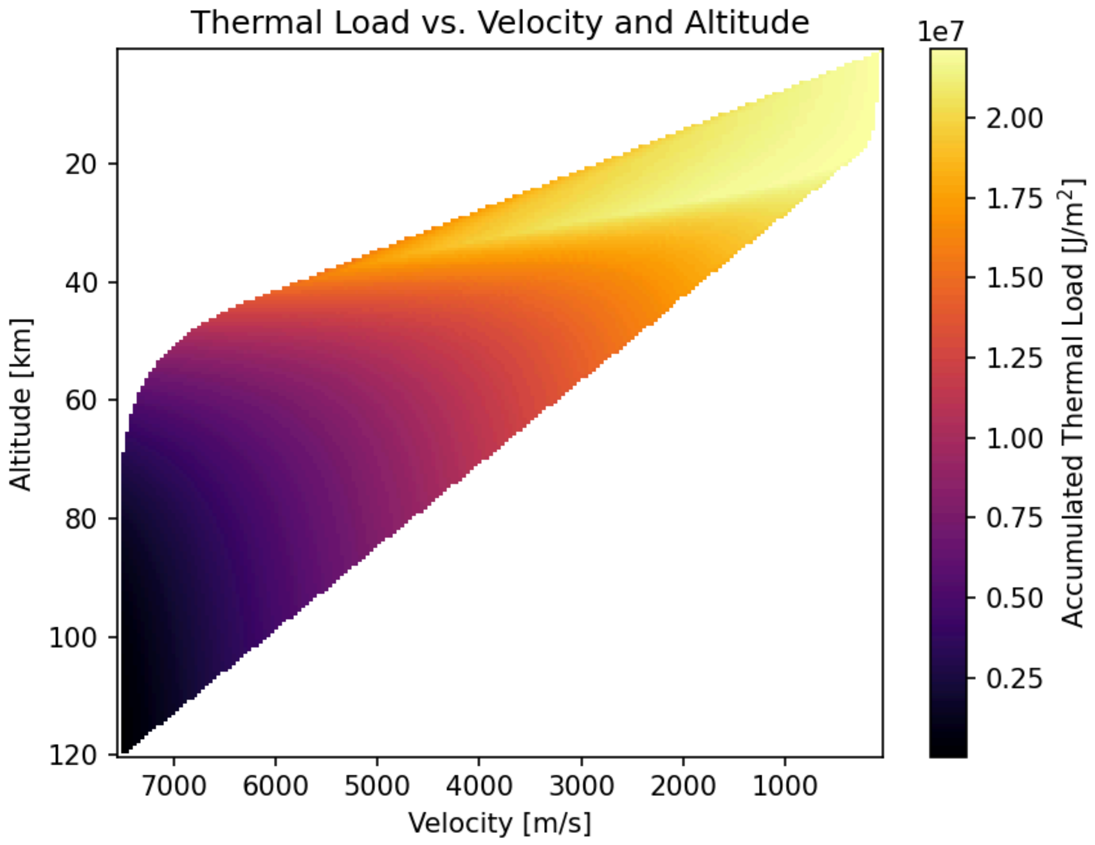


Figure 4: Thermal load distribution as a function of velocity and altitude along the re-entry trajectory

The color map illustrates the coupled effects of decreasing altitude and increasing aerodynamic heating as velocity remains high in denser atmospheric regions. Peak thermal loading occurs at moderate altitudes where the combination of high velocity and increasing atmospheric density maximizes convective heating.

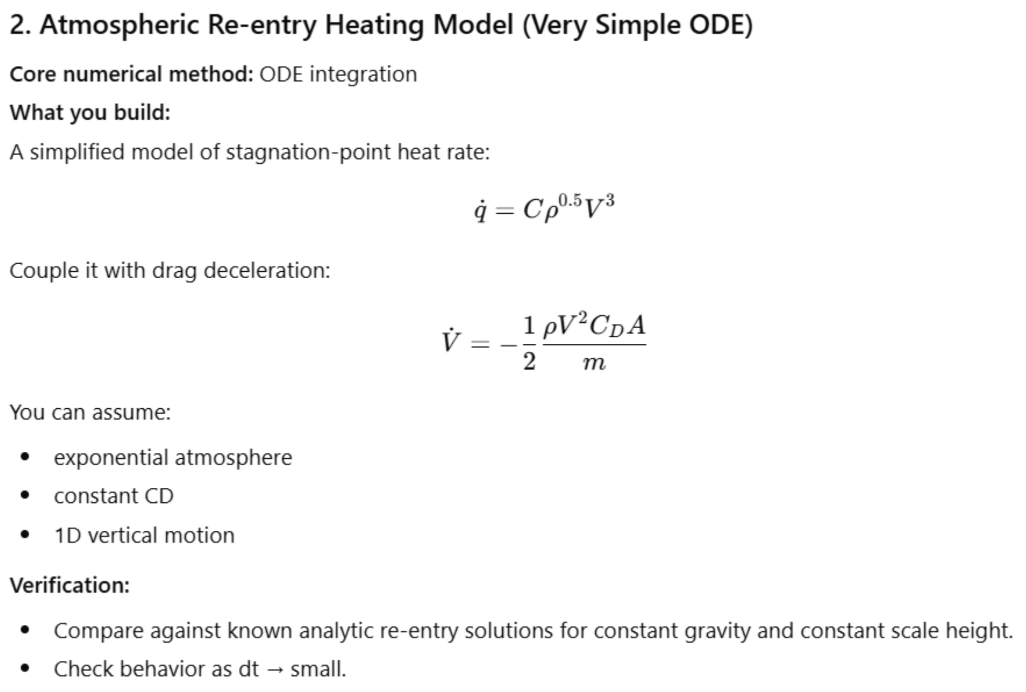
Reflection/Next Steps

From a numerical perspective, this project highlighted the impact of time step selection on solution accuracy and stability. Experimenting with different time step sizes demonstrated the trade-off between computational efficiency and numerical error, and emphasized the need for sufficient temporal resolution to accurately capture rapid changes in velocity, drag, and heating during re-entry. Implementing a fourth order Runge-Kutta integrator also strengthened my understanding of time marching methods and numerical integration of coupled ordinary differential equations.

If I had more time, one thing I would want to do is incorporate lift, trajectory angle, and the effect of parachute deployment as the spacecraft re-enters Earth’s atmosphere. This would allow for two-dimensional motion and more realistic spacecraft dynamics. Also, implementing a more detailed thermal model that accounts for heat transfer, radiation, and material response would provide a more physically accurate representation of spacecraft heating and provide a greater understanding of how different materials affect thermal load on a spacecraft. Finally, validating the model against published re-entry data or higher fidelity simulations would further strengthen confidence in my results.

LLM Transcripts
12/11/2025

- Used chatGPT to brainstorm ideas:



2. Atmospheric Re-entry Heating Model (Very Simple ODE)

Core numerical method: ODE integration

What you build:

A simplified model of stagnation-point heat rate:

$$\dot{q} = C\rho^{0.5}V^3$$

Couple it with drag deceleration:

$$\dot{V} = -\frac{1}{2}\frac{\rho V^2 C_D A}{m}$$

You can assume:

- exponential atmosphere
- constant CD
- 1D vertical motion

Verification:

- Compare against known analytic re-entry solutions for constant gravity and constant scale height.
- Check behavior as dt → small.

It came up with this idea and I immediately thought of one of our homeworks where we modeled the heat-sink cooling for pulsed rocket firing and produced a temperature graph. This was my inspiration for the project and went off from here.

12/12/2025

- Vibe Coding for main.py , integrator.py , and model.py. I had chatGPT help me code the necessary things I needed and I filled in the rest as far as re-defining parameters, or performing calculations. I also greatly reorganized the structure of the code, added more plots, and debugged some issues that came up.
- main.py

```
import numpy as np
import matplotlib.pyplot as plt
from model import reentry_odes
from integrator import rk4_step

# Physical parameters
params = {
'g': 9.81, # gravity [m/s^2]
'Cd': 1.2, # drag coefficient [-]
'A': 10.0, # reference area [m^2]
'm': 2000.0 # vehicle mass [kg]
}

# Initial conditions
h0 = 120e3 # altitude [m]
V0 = 7500.0 # downward velocity [m/s]
T0 = 0.0 # thermal load

y = np.array([h0, V0, T0])

# Time integration setup
dt = 0.5 # time step [s]
t = 0.0

history = {
'h': [],
'V': [],
'T': []
}

# Time-marching loop
while y[0] > 1000.0:
    history['h'].append(y[0])
    history['V'].append(y[1])
    history['T'].append(y[2])

y = rk4_step(reentry_odes, t, y, dt, params)
t += dt

# Convert lists to arrays
h = np.array(history['h'])
V = np.array(history['V'])
T = np.array(history['T'])

# Plot results
plt.figure()
plt.plot(V, h / 1000)
plt.gca().invert_yaxis()
plt.xlabel('Velocity [m/s]')
plt.ylabel('Altitude [km]')
plt.title('Velocity vs Altitude')
plt.grid(True)
plt.show()

plt.figure()
plt.plot(T, h / 1000)
plt.gca().invert_yaxis()
plt.xlabel('Accumulated Thermal Load')
plt.ylabel('Altitude [km]')
plt.title('Thermal Load vs Altitude')
plt.grid(True)
plt.show()
```

- integrator.py

```
import numpy as np

def rk4_step(f, t, y, dt, params):
    """
    Perform a single 4th-order Runge-Kutta step.

    Parameters
    ----------
    f : function
    Function returning dy/dt = f(t, y, params)
    t : float
    Current time [s]
    y : ndarray
    Current state vector
    dt : float
    Time step [s]
    params : dict
    Dictionary of physical parameters

    Returns
    -------
    ndarray
    Updated state vector after dt
    """

k1 = f(t, y, params)
k2 = f(t + dt/2, y + dt/2 * k1, params)
k3 = f(t + dt/2, y + dt/2 * k2, params)
k4 = f(t + dt, y + dt * k3, params)
```

```
        return y + (dt / 6.0) * (k1 + 2*k2 + 2*k3 + k4)
```

- model.py

```
        import numpy as np

        def atmospheric_density(h, rho0=1.225, H=8500.0):
            """
            Exponential atmosphere model.


            Parameters
            ----------
            h : float
            Altitude [m]
            rho0 : float
            Sea-level density [kg/m^3]
            H : float
            Scale height [m]

            Returns
            -------
            float
            Atmospheric density [kg/m^3]
            """

        return rho0 * np.exp(-h / H)

        def sutton_graves_heating(rho, V, k=1.83e-4):
            """
            Sutton-Graves stagnation-point heating correlation.

            q_dot = k * sqrt(rho) * V^3

            Parameters
            ----------
            rho : float
            Atmospheric density [kg/m^3]
            V : float
            Velocity magnitude [m/s]
            k : float
            Sutton-Graves coefficient (SI units)

            Returns
            -------
```

```
    float
    Heating rate (arbitrary thermal load units)
    """
return k * np.sqrt(rho) * abs(V)**3


def reentry_odes(t, y, params):
    """
    Coupled ODE system for 1D atmospheric re-entry.

    State vector:
    y[0] = h : altitude [m]
    y[1] = V : downward velocity [m/s]
    y[2] = T : accumulated thermal load [arbitrary units]
    """
h, V, T = y
rho = atmospheric_density(h)

# Kinematics (velocity defined positive downward)
dhdt = -V

# Dynamics (gravity + drag)
drag = 0.5 * params['Cd'] * params['A'] / params['m'] *
rho * V**2
dVdt = params['g'] - drag

# Thermal response (integrated heating)
q_dot = sutton_graves_heating(rho, V)
dTdt = q_dot

return np.array([dhdt, dVdt, dTdt])
```