357 Phase 3

Task 1

parachute (Table 1)			
Key	**Default**	**Units**	**Description**
deployed	True	N/A	Indicates if the parachute has been deployed
ejected	False	N/A	Indicates if the parachute has been ejected from the system
diameter	16.25	m	Diameter of the parachute.
Cd	.615	N/A	Drag coefficient of the parachute
mass	.185	kg	Mass of the parachute.

rocket (Table 2)			
Key	**Default**	**Units**	**Description**
on	False	N/A	Indicates if the rocket is currently firing
structure_mass	8	Kg	Mass of the rocket structure excluding fuel.
initial_fuel_mass	230	Kg	Initial mass of the rocket fuel
fuel_mass	230	Kg	Current mass of the remaining fuel, which decreases over time.
effective_exhaust_velocity	4500	m/s	Effective exhaust velocity of the rocket, a measure of efficiency.

max_thrust	3100	N	Maximum thrust generated by the rocket.
min_thrust	40	N	Minimum thrust generated by the rocket.

speed_control (Table 3)			
Key	**Default**	**Units**	**Description**
on	False	N/A	Indicates if speed control mode is activated or not
Kp	2000	N/A	Proportional gain term for PID control.
Kd	20	N/A	Derivative gain term for PID control.
Ki	50	N/A	Integral gain term for PID control.
target_velocity	-3	m/s	Desired descent speed to be maintained by the control system.

position_control (Table 4)			
Key	**Default**	**Units**	**Description**
on	False	N/A	Indicates if position control mode is activated or not
Kp	2000	N/A	Proportional gain term for PID control
Kd	1000	N/A	Derivative gain term for PID control.
Ki	50	N/A	Integral gain term for PID control.
target_altitude	7.6	m	Target altitude for the descent, set to match the length of the sky crane cable.

sky_crane (Table 5)			
Key	**Default**	**Units**	**Description**
on	False	N/A	Indicates if sky crane lowering mode is active or not
danger_altitude	4.50	m	Altitude below which the rover is considered at risk of a hard impact.
danger_speed	-1.0	m/s	Descent speed below which the rover would impact the ground too hard.
mass	35.0	kg	Mass of the sky crane mechanism.
area	16.0	m^2	Frontal area of the sky crane for drag calculations.
Cd	0.9	N/A	Drag coefficient of the sky crane.
max_cable	7.60	m	Maximum length of the cable used to lower the rover.
velocity	-0.1	m/s	Speed at which the sky crane lowers the rover onto the surface.

heat_shield (Table 6)			
Key	**Default**	**Units**	**Description**
ejected	False	N/A	Indicates if the heat shield has been ejected from the system or not
mass	225	kg	Mass of the heat shield.
diameter	4.5	m	Diameter of the heat shield.

Cd	.35	N/A	Drag coefficient of the heat shield.

edl_system (Table 7)			
Key	**Default**	**Units**	**Description**
altitude	NaN	m	Current altitude of the EDL system.
velocity	NaN	m/s	Current velocity of the EDL system
num_rockets	8	N/A	Total number of rockets in the system.
volume	150	m^3	Volume of the EDL system.
parachute	parachute	dict	Dictionary containing information related to the parachute system(Table 1)
head_shield	head_shield	dict	Dictionary containing information related to the heat shield (Table 6)
rocket	rocket	dict	Dictionary defining a single rocket's properties (Table 2)
speed_control	speed_control	dict	Dictionary for controlling descent speed via proportional-derivative-integral (PID) control (Table 3)
position_control	position_control	dict	Dictionary for controlling descent altitude via PID control (Table 4)
sky_crane	sky_crane	dict	Dictionary for sky crane system, which lowers the rover onto the surface (Tabel 5)

rover	rover	dict	Dictionary for rover properties

mission_events (Table 8)			
Key	**Default**	**Units**	**Description**
alt_heatshield_eject	8000	m	Altitude at which the heat shield should be ejected from the system.
alt_parachute_eject	900	m	Altitude at which the parachute should be detached (ejected) from the system.
alt_rockets_on	1800	m	Altitude at which the rockets should be activated to slow down the descent further.
alt_skycrane_on	7.6	m	Altitude at which the sky crane should begin lowering the rover towards the surface.

high_altitude (Table 9)			
Key	**Default**	**Units**	**Description**
temperature	Temperature function	°C	Function that calculates temperature based on altitude for higher altitudes.
pressure	pressure function	kPa	Function that calculates pressure based on altitude for higher altitudes

low_altitude (Table 10)			
Key	**Default**	**Units**	**Description**
temperature	temperature function	°C	Function that calculates temperature based on altitude for lower altitudes.
pressure	pressure function	kPa	Function that calculates pressure based on altitude for lower altitudes

mars (Table 11)			
Key	**Default**	**Units**	**Description**
g	-3.72	m/s^2	Gravitational acceleration on Mars, used to calculate descent and landing forces.
altitude_threshold	7000	m	Threshold altitude distinguishing between "low" and "high" altitude atmospheric conditions.
low_altitude	low_altitude	N/A	Dictionary containing temperature and pressure conditions for altitudes below the threshold (Table 10).
high_altitude	high_altitude	N/A	Dictionary containing temperature and pressure conditions for altitudes above the threshold (Table 9).
density	Density Function	kg/m^3	Lambda function to calculate atmospheric density as a function of

			temperature and pressure

wheel (Table 12)			
Key	**Default**	**Units**	**Description**
radius	0.30 for rovers 1-3, 0.20 for rover 4	m	Radius of each wheel, affecting ground clearance and speed calculations.
mass	1 for rover 1, 2 for rovers 2-4	kg	Mass of each wheel.

speed_reducer (Table 13)			
Key	**Default**	**Units**	**Description**
type	‘Reverted’ for rovers 1,2,4; ‘standard’ for rover 3	N/A	Type of speed reducer, which can vary by rover configuration.
diam_pinion	0.04	m	Diameter of the pinion gear, which meshes with the main gear to reduce speed.
diam_gear	0.07 for rover 1, 0.06 for rovers 2-4	m	Diameter of the main gear, controlling reduction ratio.
mass	1.5	kg	Mass of the speed reducer.

motor (Table 14)			
Key	**Default**	**Units**	**Description**
torque_stall	170 for rover 1, 180 rovers 2-3, 165 for rover 4	Nm	Torque generated by the motor at stall (zero speed).
torque_noload	0	Nm	Torque generated by the

			motor at no load.
speed_noload	3.80 for rover 1, 3.70 for rovers 2-3, 3.85 for rover 4	rad/s	Maximum speed of the motor at no load.
mass	5	kg	Mass of the motor.
effcy_tau	Array specific to each rover	Nm	Array of torque values used to calculate motor efficiency.
effcy	Array specific to each rover	%	Array of efficiency values corresponding to effcy_tau to calculate motor efficiency at various torques.

wheel_assembly (Table 15)			
Key	**Default**	**Units**	**Description**
wheel	wheel	N/A	Dictionary defining wheel properties (Table 12)
speed_reducer	speed_reducer	N/A	Dictionary defining the speed reducer properties for controlling wheel speed and torque (Table 13)
motor	motor	N/A	Dictionary defining the motor properties, including torque, speed, and efficiency (Table 14)

rover (Table 16)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Dictionary representing rover's wheel, speed reducer, and motor (Table 15)

chassis	659 for rovers 1-3, 674 for rover 4	kg	Mass of the chassis.
science_payload	75 for rovers 1-3, 80 for rover 4	kg	Mass of the scientific payload.
power_subsys	90 rovers 1-3, 100 for rover 4	kg	Mass of the power subsystem, which includes batteries and power management systems.

Task 2

Template:
Function Name
Calling Syntax

Description

Input Args

Output Args

get_mass_rover
Calling Syntax
m = get_mass_rover(edl_system)
Description
This function finds the mass of the rover by adding the masses of various components, including the motor, speed reducer, wheel assembly, chassis, science payload, and power subsystem.
Input Args

- edl_system (dict): A dictionary that represents the rover system.
    - 'rover': Contains the rover components.
    - 'wheel_assembly': Includes the motor, speed reducer, and wheel.
    - 'chassis', 'science_payload', and 'power_subsys': Additional components contributing to the mass.

Output Args

- m (scalar): The total mass of the rover, calculated by summing the masses of the defined components. Units: kilograms (kg).

get_mass_rockets
Calling Syntax
m_rockets = get_mass_rockets(edl_system)
Description
This function adds the mass of the rockets used in the system.
Input Args

- edl_system (dict): The dictionary containing parameters for the system, including rocket information:
    - 'num_rockets': The total number of rockets.
    - 'rocket': A dictionary containing the mass data for the rocket's structure and fuel:
    - 'structure_mass': The mass of the rocket's structure.
    - 'fuel_mass': The mass of the rocket's fuel.

Output Args

- m_rockets (scalar): The total mass of the rockets, calculated based on the data from the dictionary. Units: kilograms (kg)

Get_mass_edl

Calling Syntax

m_edl = get_mass_edl(edl_system)

Description

This function calculates the total mass of the EDL system by summing the masses of various components, including the parachute, heat shield, rockets, sky crane, and rover. It uses other functions such as get_mass_rockets and get_mass_rover to calculate the masses of the rockets and rover.

Input Args

- edl_system (dict): The dictionary representing the EDL system, including the following fields:
    - 'parachute': Contains the mass and ejection status.
    - 'heat_shield': Contains the mass and ejection status.
    - 'sky_crane': The mass of the sky crane.

Output Args

- m_edl (scalar): The total mass of the EDL system, calculated in kilograms (kg).

Get_local_atm_properties

Calling Syntax

[density, temperature, pressure] = get_local_atm_properties(planet, altitude)

Description

This function returns the local atmospheric properties (density, temperature, and optionally pressure) at a given altitude on a specified planet. It uses the planet's atmospheric models to return the corresponding properties based on whether the altitude is above or below a certain threshold.

Input Args

- planet (dict): A dictionary representing the planet's atmospheric properties, including:
    - 'altitude_threshold': A threshold altitude that distinguishes high and low altitude models.
    - 'high_altitude' and 'low_altitude': Dictionaries containing functions to calculate temperature and pressure at different altitudes.
- altitude (scalar): The altitude at which to retrieve atmospheric properties, specified in meters.

Output Args

- density (scalar): The atmospheric density at the given altitude, in kg/m^3.
- temperature (scalar): The atmospheric temperature at the given altitude, in Celsius (°C).

- pressure (scalar, optional): The atmospheric pressure at the given altitude, in kilopascals (kPa).

F_buoyancy_descent
Calling Syntax
F = F_buoyancy_descent(edl_system, planet, altitude)
Description
This function calculates the net buoyancy force acting on the EDL system during descent. The buoyancy force depends on the atmospheric density, gravitational acceleration, and the volume of the EDL system.
Input Args
- edl_system (dict): The dictionary containing the system's properties, including:
    - 'volume': The volume of the system, used in the buoyancy calculation.
- planet (dict): The dictionary containing the planet's gravitational acceleration ('g').
- altitude (scalar): The altitude at which the buoyancy force is to be calculated, specified in meters.

Output Args
- F (scalar): The net buoyancy force in newtons (N).

F_drag_descent
Calling Syntax
F_drag = F_drag_descent(edl_system, planet, altitude, velocity, Modified)
Description
This function calculates the net drag force during descent. It uses atmospheric properties, descent velocity, and a modified drag coefficient based on the Mach number. The drag force is influenced by the EDL system components contributing to drag.
Input Args
- edl_system (dict): The dictionary containing the system's properties.
- planet (dict): The dictionary containing the planet's atmospheric properties.
- altitude (scalar): The altitude at which to calculate the drag force, in meters.
- velocity (scalar): The descent velocity, in meters per second (m/s).
- Modified (scalar): The modification factor for the drag coefficient.

Output Args
- F_drag (scalar): The net drag force in newtons (N).

F_gravity_descent
Calling Syntax
F_gravity = F_gravity_descent(edl_system, planet)
Description
This function computes the gravitational force acting on the EDL system. The gravitational force is calculated by multiplying the total mass of the EDL system (using get_mass_edl) by the gravitational acceleration of the planet.
Input Args
- edl_system (dict): The dictionary representing the EDL system, which is used to calculate the total mass through the get_mass_edl function.

- planet (dict): The dictionary representing the planet's properties, including:
    - 'g': The gravitational acceleration on the planet, in meters per second squared (m/s²).

Output Args

- F_gravity (scalar): The gravitational force acting on the EDL system, in newtons (N).

v2M_Mars

Calling Syntax

M = v2M_Mars(v, a)

Description

This function converts a given descent speed (v) in meters per second (m/s) to the Mach number on Mars, as a function of the altitude (a). The Mach number is calculated by dividing the descent speed by the local speed of sound, which varies with altitude. The function uses a predefined table of speed of sound values at different altitudes and a cubic spline interpolation.

Input Args

- v (scalar): The descent speed in meters per second (m/s).
- a (scalar): The altitude in meters (m) at which to calculate the Mach number.

Output Args

- M (scalar): The Mach number, calculated as the absolute value of the descent speed divided by the local speed of sound at the given altitude.

thrust_controller

Calling Syntax

edl_system = thrust_controller(edl_system, planet)

Description

This function implements a PID controller for the EDL system's rocket thrust. It uses the system's state, including altitude and rocket control status, to modify the thrust accordingly. The function adjusts the edl_system dictionary based on control parameters, ensuring the descent velocity and altitude are within the desired limits.

Input Args

- edl_system (dict): A dictionary representing the EDL system, including rocket control settings, altitude, and sky crane parameters.
    - 'rocket': Contains control status and rocket activation parameters.
    - 'altitude': The current altitude of the system.
    - 'sky_crane': The sky crane parameters, including the maximum rope length.
- planet (dict): A dictionary representing the planet's properties used in modeling the EDL system's behavior based on its environment.

Output Args

- edl_system (dict): The updated EDL system dictionary with adjusted thrust values.

edl_events

Calling Syntax

events = edl_events(edl_system, mission_events)

Description
This function defines the key events that occur during the EDL system simulation. Each event corresponds to specific conditions based on the system's state (e.g., velocity, altitude, fuel mass) and triggers actions or state changes in the simulation. The events are represented by lambda functions that define the conditions for each event
Input Args

- edl_system (dict): The dictionary representing the EDL system, which includes parameters like heat shield and parachute ejection status.
- mission_events (dict): A dictionary containing mission-specific event thresholds, such as altitudes for heat shield and parachute ejection.

Output Args

- events (list): A list of lambda functions representing different EDL events, where each lambda function defines the condition that must be met for the event to trigger.

edl_dynamics
Calling Syntax
ydot = edl_dynamics(t, y, edl_system, planet, Modified)
Description
This function models the dynamics of the EDL system during descent. It uses the system's state vector (y), which includes position, velocity, and fuel mass, and computes the time derivatives of the system's state. The function handles the variable mass system by considering the momentum contributions of the EDL system and propellant expulsion.
Input Args

- t (scalar): The current time in seconds.
- y (vector): The state vector containing the following elements:
    - vel_edl: Velocity of the EDL system.
    - pos_edl: Position of the EDL system.
    - fuel_mass: The current fuel mass.
    - ei_vel, ei_pos: Errors related to velocity and position.
    - vel_rov, pos_rov: The velocity and position of the rover relative to the EDL system.
- edl_system (dict): The dictionary containing the EDL system's properties.
- planet (dict): The dictionary containing the planet's properties, including gravitational acceleration.
- Modified (scalar): A factor used to modify the dynamics based on specific conditions.

Output Args

- ydot (vector): The time derivatives of the state vector, representing the system's velocity, position, fuel mass change, and acceleration.

update_edl_state
Calling Syntax
update_edl_state(edl_system, TE, YE, Y, ITER_INFO)
Description

This function updates the state of the EDL system based on simulation events. It updates the system's parameters such as altitude, velocity, and fuel mass after each simulation step. The function also tracks specific events such as the ejection of the heat shield or parachute, rocket activation, and rover touchdown. It modifies the fuel mass for each rocket based on the current simulation state.

Input Args

- edl_system (dict): A dictionary representing the EDL system, including parameters like rocket fuel mass, altitude, and velocity.
- TE (scalar): The event time, used to track when specific events occur during the simulation.
- YE (vector): The event values, representing various system states at the time of the event.
- Y (matrix): The matrix of simulation states, where each column represents the system's state at a specific time step.
- ITER_INFO (dict): A dictionary containing iteration information, including simulation steps or conditions.

Output Args

- edl_system (dict): The updated EDL system dictionary, reflecting changes in the system's parameters like fuel mass, altitude, and velocity.
- y0 (vector): The updated state vector after the current simulation step, containing parameters such as velocity, position, and fuel mass.
- TERMINATE_SIM (scalar): A flag indicating whether the simulation should terminate, typically used for checking conditions like zero altitude or fuel depletion.

simluate_edl

Calling Syntax

T, Y, edl_system = simulate_edl(edl_system, planet, mission_events)

Description

This function simulates the EDL system’s dynamics over time. It integrates the system’s equations of motion, taking into account the events that occur during the descent and landing process, such as fuel consumption, altitude control, and rover touchdown. The simulation runs iteratively, updating the system's state based on the dynamics and event thresholds.

Input Args

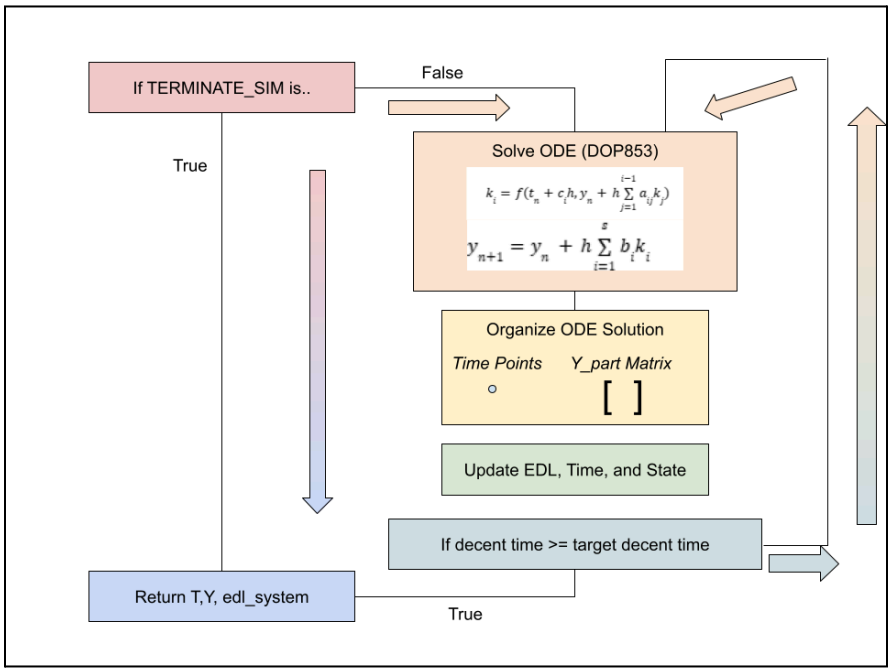
- edl_system (dict): A dictionary representing the EDL system, containing parameters like rocket fuel mass, altitude, velocity, and other system properties.
- planet (dict): A dictionary representing the planet's properties, including atmospheric conditions, gravitational acceleration, and other relevant parameters.
- mission_events (dict): A dictionary containing key mission event thresholds, such as altitudes for the ejection of the heat shield, parachute deployment, or rover touchdown.

Output Args

- T (array): The time array for the simulation, representing the simulation time steps at which the system state is calculated.
- Y (array): The state matrix, where each row corresponds to the state of the system at a specific time step (e.g., velocity, position, fuel mass).

- edl_system (dict): The updated EDL system dictionary after the simulation, including the final system state after considering all events and dynamics.

Task 3



While Loop Logic

The while loop will run when its condition is TRUE, and will continue to run while the condition is false. In this case, the statement is not(TERMINATE_SIM). Prior to the loop, “TERMINATE_SIM” is set to False, which means the statement is read as True, and begins running. The final statement of the loop checks and sets up if we run another loop or not, see the bottom for details.

ODE Solver

After setting “fun” to our function to solve, we run it through a solver called DOP853, which performs small steps and calculates the forces and their changes to the system for each step. By hand or manually these calculations would take a long time as they are constantly changing, and need to be performed many times to get a small enough step size to be accurate. DOP853 uses inputs like the initial state, time span, and the actual ODE function. Then DOP853 uses an 8th Order Runge Kutta method, where it applies the formula below.

$$k_i = f(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j)$$

The coefficients are specific to the DOP853 method to optimize the stability and accuracy of the model. C determines where in the interval each stage is, A weighs the previous stages, and is weighed against b to compute the next solution at y(n+1) in the following equation

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

Where s is the number of stages used. DOP853 uses adaptive step size control based on error estimates which allows for optimization of compute and accuracy. DOP853 is also known for its high accuracy, 8th order, and stability for sensitive problems.

Organizing ODE Solution

After running DOP853 we assign t_part to sol.t which is an array of the time points DOP calculated. We also assign y_part to sol.y which is a 2D array which represents the system's state at each time point, 3 state variables, 5 time points. Then TE is assigned to sol.t_events which sontains times where certain events occur, and is used later in our edl_state function. We do the same for YE for the position at each time event.

Update EDL State

After we run the ODE solver to get the next step in the decent, we must update our state with things like mass, drag, and thrust to make sure our next iteration has accurate data. Things like mass, drag and thrust change in events like rockets firing.

Update Time and State

First we update the simulation time for the next stage by setting the tspan to the last time value from the most recent solution segment. Next we add all the new time points to the main time list, and do the same for the y_part but its an array so its not appending to a list.

Loop State Check

The final statement of the loop checks if tspan[0] is greater than or equal to tspan[1], and if so, sets "TERMINATE_SIM" to True, which would end the loop. If tspan[0] is greater than or equal to tspan[1] that means the decent time, tspan[0], has taken longer or as long as the target end time, tspan[1].

Task 4

Initialization

The scripts begins by importing several modules, including numpy and matplotlib.pyplot, and other key functions that are used for initial conditions such a define_edl_system, subfunctions_EDL, define_planet, and define_mission_events

It loads the initial conditions by calling several functions:

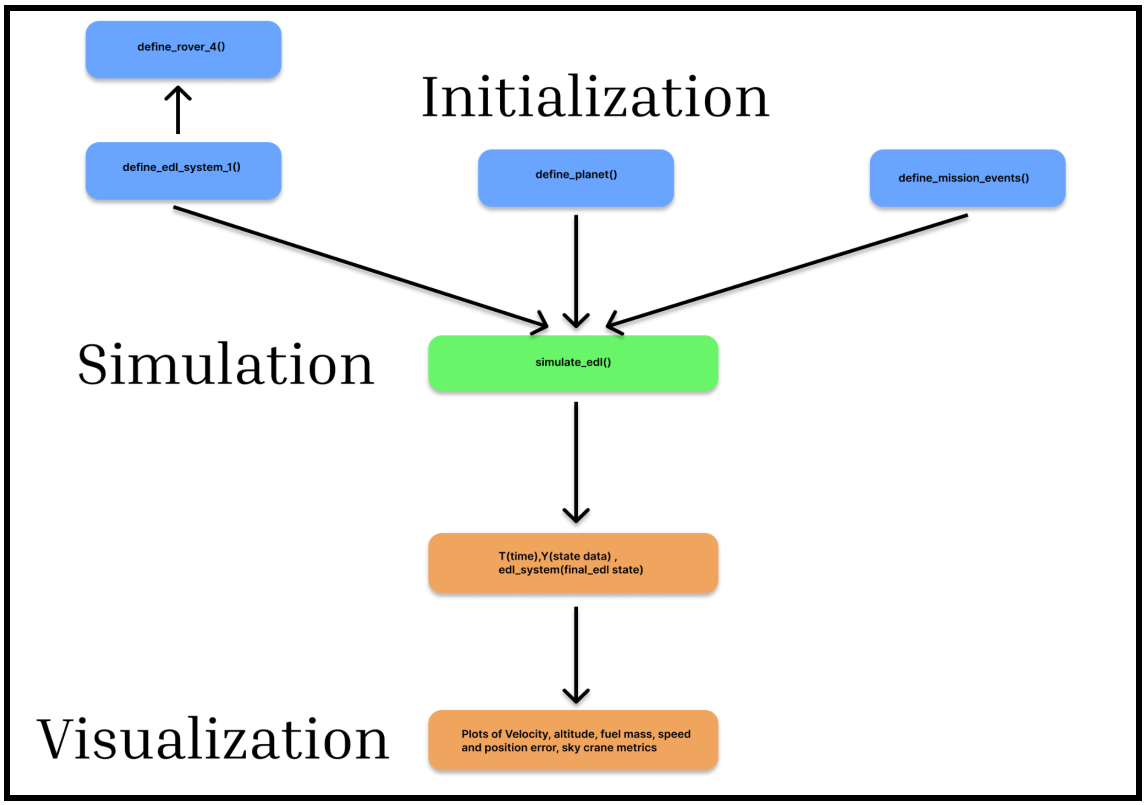
- **define_edl_system_1():** from define_edl_system, which defines the EDL system in multiple dictionaries including parachute, rocket, speed_control, position_control, sky_crane, heat_shield, rover and returns a dictionary of edl_system
- **define_planet():** from define_planet, which defines planetary characteristics
- **define_mission_events()**: from define_mission_events, which outlines critical events for the mission

Simulation

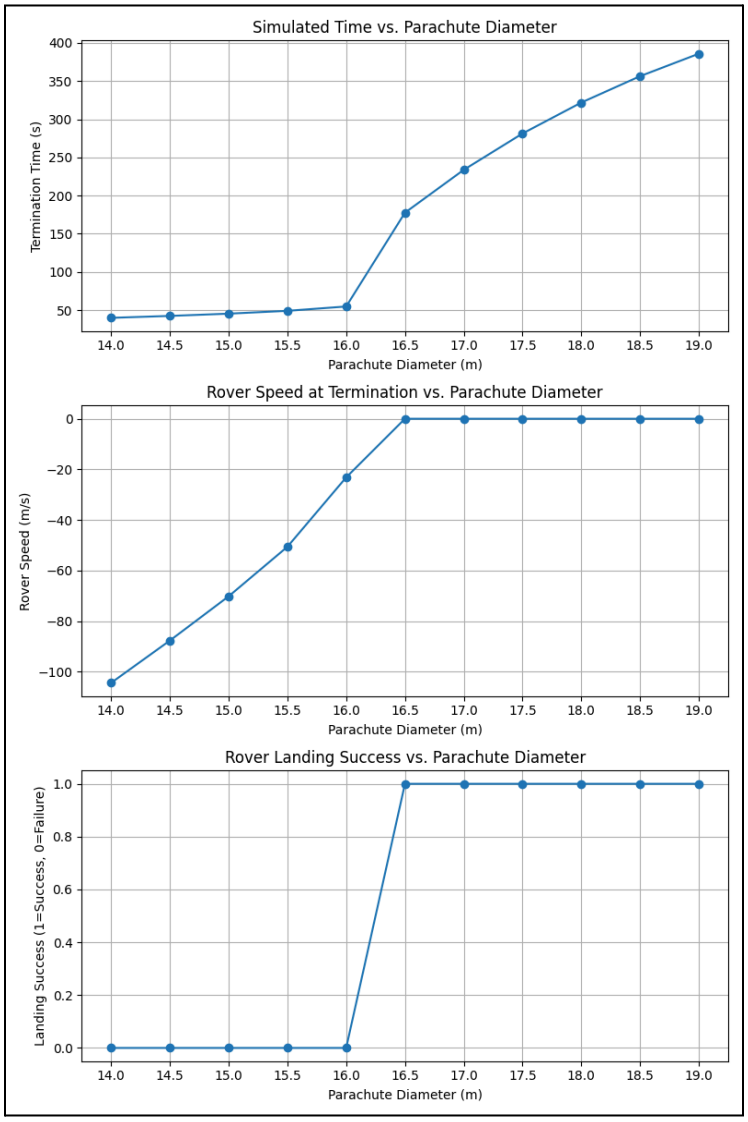
- With determined initial conditions overriding what might be in the loaded data to establish our desired initial conditions, the script calls simulate_edl() from the subfunctions_EDL, passing the loaded EDL system data, planetary data, mission events, maximum simulation time, and a flag for message echo.
- simulate_edl() computes trajectory and state changes throughout EDL(Entry, Descent, and Landing) process, returning the time vector t, simulation data Y, and final state of the edl_system

Visualization

- Once simulation data is returned, the script utilizes matplotlib to plot various EDL metrics over time:
- Velocity, altitude, fuel mass, speed and position errors, relative velocity and position of the rover and sky crane.
- Generates two figures with subplots, which show EDL dynamics



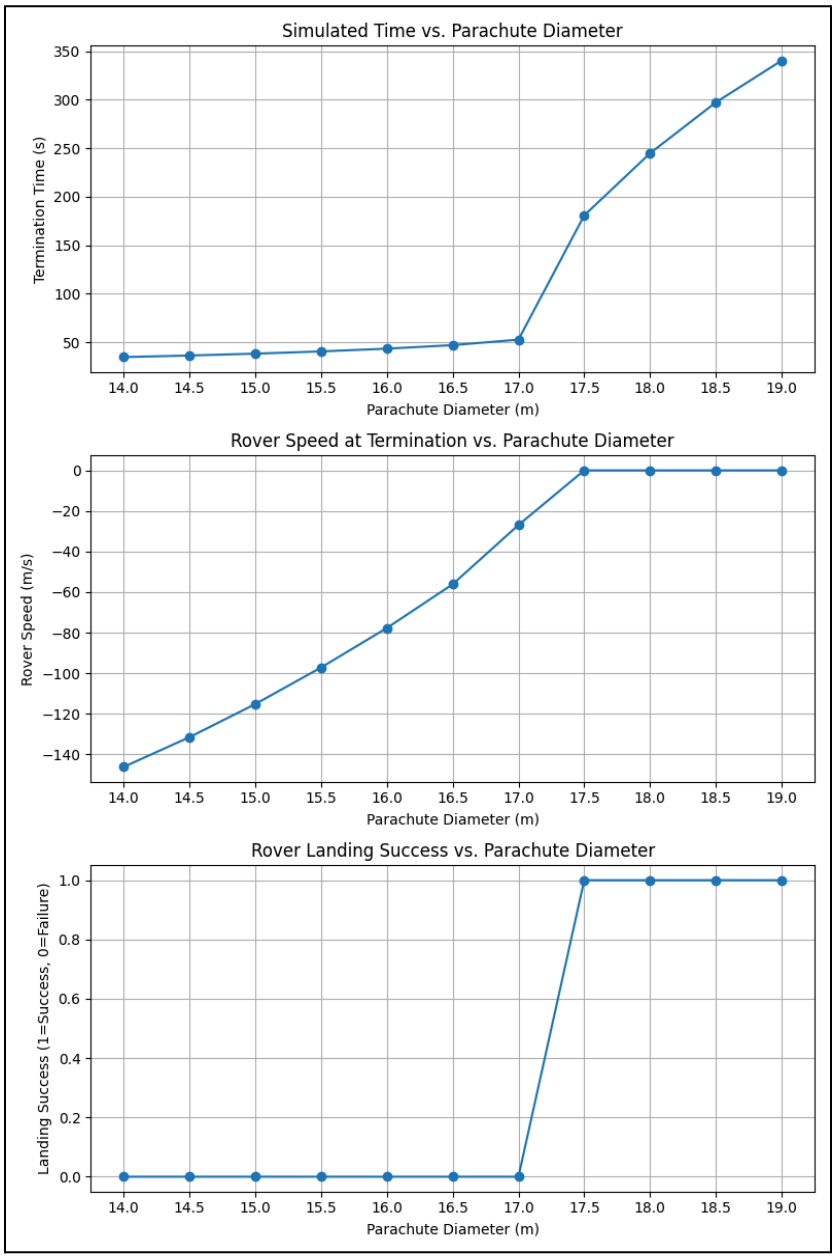
Task 5



Recommendations

The minimum required diameter for a safe landing is 16.5 meters based on successful landing criteria. The criteria for a successful landing is the rover's landing speed is no more than 1m/s and the sky crane altitude is no less than 4.5m. Looking at the figure above, 16.5 is the minimum diameter that guarantees a safe landing, however, for a conservative value, the recommended rover diameter that also minimizes air time would be between 17 and 18 meters. Above 18 meters guarantees a safe landing, but the time increases and provides no significant safety advantage.

Task 6



Recommendations and Process

1. To create a continuous function for the MEF data, I used the Cubic spline function from Scipy and the v2_mach function along with the data to create an interpolation function between the maxim and minimum mach values given in the data. The reason a cubic spline was used as opposed to a linear spline or a polynomial regression is cubic spline has higher accuracy and is smoother. A linear spline may be similar in terms of accuracy as it hits every data point, however, looking at the data the MEF, the values don't change suddenly, there's usually some gradual increase and decrease around 1.4 mach which a cubic spline will better capture. To ensure that the

code still worked for task 5 and gave the same results without the modified Cd, we included a modified argument within Simulate_edl, edl_dynamics, and F_drag_descent. Its default is False, and when switched to True will apply the improved Cd model. There are two scripts, one with the unmodified version of the Cd and one with the modified Cd. There is now an if statement in the F_drag_descent function that allows us to switch between the two different settings using the modified argument.

2. Based on the new simulation using the improved coefficient of drag formula, the diameter recommendation in task 5 must be revised. Now, the minimum diameter needed for a safe rover landing is 17.5 meters. The recommended diameter will be between 18 and 19 meters to ensure a safe landing if conditions are slightly different than expected while also minimizing the time the rover is in the air.