Tasks 1

Dictionaries:

parachute (1)
Parachute details

Key	Default	Units	Description
deployed	True	N/A	Deployment status of parachute
ejected	False	N/A	Ejected status of parachute (attached or not)
diameter	16.25	m	Diameter of the parachute
Cd	0.615	N/A	Drag coefficient of parachute (subsonic default)
mass	185.0	kg	mass of parachute

rocket (2)
Rocket system parameters

Key	Default	Units	Description
on	False	N/A	Rocket status (on/off)
structure_mass	8.0	kg	Mass of the rocket (no fuel)
initial_fuel_mass	230.0	kg	Mass of the fuel before ignition
fuel_mass	230.0	kg	Current mass of fuel (changes, but starts at initial)
effective_exhaust_velocity	4500.0	m/s	Velocity of exhaust in one rocket
max_thrust	3100.0	N	Maximum force of thrust in flight from the rocket
min_thrust	40.0	N	Mininum force of thrust in flight from the rocket

speed_control (3)
Speed controller parameters

Key	Default	Units	Description
on	False	N/A	Indicates the activation status of the control
Kp	2000	N/A	Proportional gain term

Kd	20	N/A	Derivative gain term
Ki	50	N/A	Integral gain term
target_velocity	-3.0	m/s	Desired descent speed

position_control (4)
Position controller parameters

Key	Default	Units	Description
on	False	N/A	Indicates whether control mode is on or not
Kp	2000	N/A	Proportional gain term
Kd	1000	N/A	Derivative gain term
Ki	50	N/A	Integral gain term
target_altitude	7.6	m	Position. Reflects the Sky crane cable length

sky_crane (5)
Sky Crane important parameters

Key	Default	Units	Description
on	False	N/A	Indicates lowering status
danger_altitude	4.5	m	Altitude at which it’s dangerous for the rover to touchdown
danger_speed	-1.0	m/s	The speed at which the rover would impact too hard on the surface
mass	35.0	kg	Sky crane mass
area	16.0	m^2	Frontal area used for drag calculations
Cd	0.9	N/A	Coefficient of drag
max_cable	7.6	m	Maximum length of the cable for lowering the rover
velocity	-0.1	m/s	The speed at which the sky crane lowers

heat_shield (6)
Heat shield parameters

Key	Default	Units	Description
ejected	False	N/A	Heat shield ejection status
mass	225	kg	Mass of the heat shield
diameter	4.5	m	Diameter of the heat shield
Cd	0.35	N/A	Drag coefficient of the heat shield

edl_system (7)
Entry, Descent, and Landing system defined by its parameters

Key	Default	Units	Description
altitude	np.NaN	m	Altitude that updates periodically
velocity	np.NaN	m/s	Velocity that updates periodically
num_rockets	8	N/A	Number of rockets in the simulation
volume	150	m^3	Volume of the system
parachute	parachute	dict	Previously defined dictionary
heat_shield	heat_shield	dict	Previously defined dictionary
rocket	rocket	dict	Previously defined dictionary
speed_control	speed_control	dict	Previously defined dictionary
position_control	position_control	dict	Previously defined dictionary
sky_crane	sky_crane	dict	Previously defined dictionary
rover	rover	dict	Previously defined dictionary

mission_events (8)
Mission parameters for the landing

Key	Default	Units	Description
alt_heatshield_eject	8000	m	Altitude at which the heatshield was to be ejected
alt_parachute_eject	900	m	Altitude at which the parachute was to be ejected
alt_rockets_on	1800	m	Altitude at which rockets were to be turned on

alt_skycrane_on	7.6	m	Altitude at which to deploy the sky crane

high_altitude (9)
High altitude properties on Mars

Key	Default	Units	Description
temperature	Lambda function using altitude	C	Lambda Function of temperature at high altitude on Mars using altitude
pressure	Lambda function using altitude	KPa	Function of pressure at high altitude on Mars using altitude

low_altitude (10)
Low altitude properties on Mars

Key	Default	Units	Description
temperature	Lambda function using altitude	C	Lambda Function of temperature at low altitude on Mars using altitude
pressure	Lambda function using altitude	KPa	Lambda Function of pressure at low altitude on Mars using altitude

mars (11)
Mars atmosphere properties and other parameters for simulation

Key	Default	Units	Description
g	-3.72	m/s^2	Gravity of Mars
altitude_threshold	7000	m	Altitude at which Mars should be considered
low_altitude	dict	N/A	Low altitude pressure/temperature information
high_altitude	dict	N/A	High altitude pressure/temperature information
density	Lambda function using pressure and temperature	kg/m^3	Density of the atmosphere at a certain altitude

rover_1 (12)

Consists of four overarching dictionaries: wheel_assembly, chassis, science_payload, power_sybsys

Key	Default	Units	Description
wheel	dict	N/A	Contains radius and mass
radius	0.30	m	Radius of wheel
mass	1	kg	Mass of the wheel
speed_reducer	dict	N/A	Contains torque_stall, torque_noload, speed_noload and mass
type	reverted	N/A	Indicated speed reducer direction
diam_pinion	0.04	m	Diameter of pinion
diam_gear	0.07	m	Diameter of gear (for gear ratio)
mass	1.5	kg	Mass of speed reducer
motor	dict	N/A	Contains items below
torque_stall	170	Nm	Stalling torque
torque_noload	0	Nm	No load torque
speed_noload	3.80	rad/s	Angular speed of motor
mass	5.0	kg	Mass of motor
chassis	dict		Contains mass
mass	659	kg	Mass of the chassis
science_payload	dict	N/A	Contains mass
mass	75	kg	Mass of the science payload
power_subsys	dict	N/A	Contains mass
mass	90		Mass of the power sub-system
wheel_assembly	dict	N/A	Wheel assembly dictionary that contains the wheel, speed reducer, and motor dictionaries
wheel	dict	N/A	Wheel dictionary (already described)
speed_reducer	dict	N/A	Speed reducer dictionary (already described)

motor	dict	N/A	Motor dictionary (already described)

rover_2 (13)
Consists of four overarching dictionaries: wheel_assembly, chassis, science_payload, power_sybsys

Key	Default	Units	Description
wheel	dict	N/A	Contains radius and mass
radius	0.30	m	Radius of wheel
mass	2	kg	Mass of the wheel
speed_reducer	dict	N/A	Contains torque_stall, torque_noload, speed_noload and mass
type	reverted	N/A	Indicated speed reducer direction
diam_pinion	0.04	m	Diameter of pinion
diam_gear	0.06	m	Diameter of gear (for gear ratio)
mass	1.5	kg	Mass of speed reducer
motor	dict	N/A	Contains items below
torque_stall	180		Stalling torque
torque_noload	0		No load torque
speed_noload	3.70		Angular speed of motor
mass	5.0	kg	Mass of motor
chassis	dict		Contains mass
mass	659	kg	Mass of the chassis
science_payload	dict	N/A	Contains mass
mass	75	kg	Mass of the science payload
power_subsys	dict	N/A	Contains mass
mass	90		Mass of the power sub-system
wheel_assembly	dict	N/A	Wheel assembly dictionary that contains the wheel, speed reducer, and motor dictionaries
wheel	dict	N/A	Wheel dictionary (already described)

speed_reducer	dict	N/A	Speed reducer dictionary (already described)
motor	dict	N/A	Motor dictionary (already described)

rover_3 (14)
Consists of four overarching dictionaries: wheel_assembly, chassis, science_payload, power_sybsys

Key	**Default**	**Units**	**Description**
wheel	dict	N/A	Contains radius and mass
radius	0.30	m	Radius of wheel
mass	2	kg	Mass of the wheel
speed_reducer	dict	N/A	Contains torque_stall, torque_noload, speed_noload and mass
type	reverted	N/A	Indicated speed reducer direction
diam_pinion	0.04	m	Diameter of pinion
diam_gear	0.06	m	Diameter of gear (for gear ratio)
mass	1.5	kg	Mass of speed reducer
motor	dict	N/A	Contains items below
torque_stall	180		Stalling torque
torque_noload	0		No load torque
speed_noload	3.70		Angular speed of motor
mass	5.0	kg	Mass of motor
chassis	dict		Contains mass
mass	659	kg	Mass of the chassis
science_payload	dict	N/A	Contains mass
mass	75	kg	Mass of the science payload
power_subsys	dict	N/A	Contains mass
mass	90		Mass of the power sub-system
wheel_assembly	dict	N/A	Wheel assembly dictionary that contains the wheel, speed reducer, and motor dictionaries

wheel	dict	N/A	Wheel dictionary (already described)
speed_reducer	dict	N/A	Speed reducer dictionary (already described)
motor	dict	N/A	Motor dictionary (already described)

rover_4 (15)
Consists of four overarching dictionaries: wheel_assembly, chassis, science_payload, power_sybsys

Key	Default	Units	Description
wheel	dict	N/A	Contains radius and mass
radius	0.20	m	Radius of wheel
mass	2	kg	Mass of the wheel
speed_reducer	dict	N/A	Contains torque_stall, torque_noload, speed_noload and mass
type	reverted	N/A	Indicated speed reducer direction
diam_pinion	0.04	m	Diameter of pinion
diam_gear	0.06	m	Diameter of gear (for gear ratio)
mass	1.5	kg	Mass of speed reducer
motor	dict	N/A	Contains items below
torque_stall	165		Stalling torque
torque_noload	0		No load torque
speed_noload	3.85		Angular speed of motor
mass	5.0	kg	Mass of motor
chassis	dict		Contains mass
mass	674	kg	Mass of the chassis
science_payload	dict	N/A	Contains mass
mass	80	kg	Mass of the science payload
power_subsys	dict	N/A	Contains mass
mass	100		Mass of the power sub-system
wheel_assembly	dict	N/A	Wheel assembly dictionary that contains the

			wheel, speed reducer, and motor dictionaries
wheel	dict	N/A	Wheel dictionary (already described)
speed_reducer	dict	N/A	Speed reducer dictionary (already described)
motor	dict	N/A	Motor dictionary (already described)

Tasks 2

1. get_mass_rover

- Calling Syntax: mass = get_mass_rover(edl_system)
- Description: Computes the total mass of the rover defined in the rover field of the edl_system dictionary. Assumes the rover is structured according to Phase 1 specifications
- Input Arguments:
    - edl_system (dict): A dictionary containing the rover's structural definition, including masses of all components.
    - rover (dict): Contains sub-dictionaries for wheel_assembly, chassis, science_payload, and power_subsys, each with mass fields.
- Output Arguments:
    - mass (float): Total mass of the rover in kilograms (kg).

2. get_mass_rockets

- Calling Syntax: mass = get_mass_rockets(edl_system)
- Description: Returns the current total mass of all rockets in the EDL system, including both structural mass and fuel mass.
- Input Arguments:
    - edl_system (dict): Dictionary containing rocket specifications.
    - num_rockets (int): Number of rockets.
    - rocket (dict): Contains structure_mass and fuel_mass per rocket.
- Output Arguments:
    - mass (float): Total mass of all rockets in kilograms (kg).

3. get_mass_edl

- Calling Syntax: mass = get_mass_edl(edl_system)
- Description: Computes the total current mass of the EDL system, accounting for ejected components (parachute, heat shield) and active subsystems (rockets, sky crane, rover).
- Input Arguments:
    - edl_system (dict): Dictionary defining the EDL system configuration.
    - parachute, heat_shield (dicts): Contain ejected (bool) and mass (float).
    - sky_crane (dict): Contains mass (float).

        - Calls get_mass_rockets and get_mass_rover.
    - Output Arguments:
        - mass (float): Total mass of the EDL system in kilograms (kg).

4. get_local_atm_properties

- Calling Syntax:
    - density = get_local_atm_properties(planet, altitude)
    - [density, temperature] = get_local_atm_properties(planet, altitude)
    - [density, temperature, pressure] = get_local_atm_properties(planet, altitude)
    - Variations:
        - Single output: Returns only density.
        - Two outputs: Returns density and temperature.
        - Three outputs: Returns density, temperature, and pressure.
- Description: Returns atmospheric properties (density, temperature, pressure) at a given altitude on the planet. Uses different models for high/low altitudes based on altitude_threshold. Not vectorized (single altitude only).
- Input Arguments:
    - planet (dict): Contains atmospheric models (high_altitude, low_altitude) and density function.
    - altitude (float): Altitude in meters (m).
- Output Arguments:
    - density (float): Atmospheric density in kg/m^3.
    - temperature (float, optional): Temperature in °C.
    - pressure (float, optional): Pressure in kPa.

5. F_buoyancy_descent

- Calling Syntax: force = F_buoyancy_descent(edl_system, planet, altitude)
- Description: Computes the net buoyancy force acting on the EDL system using atmospheric density and volume. Direction depends on planet['g'] sign.
- Input Arguments:
    - edl_system (dict): Must contain volume (float, m^3).
    - planet (dict): Must contain g (float, m/s^2) and calls get_local_atm_properties.
    - altitude (float): Altitude in meters (m).
- Output Arguments:
    - force (float): Buoyancy force in Newtons (N).

6. F_drag_descent

- Calling Syntax: force = F_drag_descent(edl_system, planet, altitude, velocity)
- Description: Computes the net drag force based on atmospheric density, velocity, and drag-contributing components (heat shield, parachute, or sky crane). Accounts for deployed/ejected states.
- Input Arguments:
    - edl_system (dict): Must contain heat_shield, parachute, and sky_crane configurations (diameters, Cd, deployed/ejected status).

- planet (dict): Provides atmospheric density via get_local_atm_properties.
    - altitude (float): Altitude (m).
    - velocity (float): Velocity (m/s).
- Output Arguments:
    - force (float): Drag force in Newtons (N).

7. F_gravity_descent

- Calling Syntax: force = F_gravity_descent(edl_system, planet)
- Description: Calculates gravitational force acting on the EDL system using its mass and planetary gravity.
- Input Arguments:
    - edl_system (dict): Mass obtained via get_mass_edl.
    - planet (dict): Must contain g (float, m/s²).

- Output Arguments:
    - force (float): Gravitational force in Newtons (N).

8. v2M_Mars

- Calling Syntax: M = v2M_Mars(v, a)
- Description: Converts descent speed v to Mach number on Mars using altitude-dependent speed of sound (interpolated from data).
- Input Arguments:
    - v (float): Velocity in m/s.
    - a (float): Altitude in meters (m).
- Output Arguments:
    - M (float): Mach number (absolute value).

9. thrust_controller

- Calling Syntax: edl_system = thrust_controller(edl_system, planet)
- Description: Implements a PID controller for rocket thrust during descent. Adjusts thrust based on velocity/altitude errors and saturates at min/max limits. Updates telemetry data.
- Input Arguments:
    - edl_system (dict): Contains rocket control parameters (Kp, Ki, Kd), thrust limits, and telemetry arrays.
    - planet (dict): Provides gravity for force calculations.
- Output Arguments:
    - edl_system (dict): Modified input dictionary with updated thrust and telemetry fields.

10. edl_events

- Calling Syntax: events = edl_events(edl_system, mission_events)
- Description: Defines event functions (e.g., parachute ejection, sky crane activation) for the ODE solver. Each event triggers changes in the EDL system state.

- Input Arguments:
    - edl_system (dict): System configuration (e.g., sky_crane['on']).
    - mission_events (dict): Contains altitude thresholds (e.g., alt_heatshield_eject).
- Output Arguments:
    - events (list of lambda functions): Event functions for solve_ivp.

11. edl_dynamics

- Calling Syntax: dydt = edl_dynamics(t, y, edl_system, planet)
- Description: Computes the time derivatives of the EDL state vector (velocity, altitude, fuel mass, etc.) under external forces (drag, gravity, thrust) and sky crane dynamics.
- Input Arguments:
    - t (float): Time (s).
    - y (array): State vector [vel_edl, pos_edl, fuel_mass, ...].
    - edl_system (dict): Configuration and control parameters.
    - planet (dict): Planetary constants (gravity).
- Output Arguments:
    - dydt (array): Derivatives of the state vector.

12. update_edl_state

- Calling Syntax: [edl_system, y0, TERMINATE_SIM] = update_edl_state(edl_system, TE, YE, Y, ITER_INFO)
    - Full output: Returns all three outputs (edl_system, y0, TERMINATE_SIM).
    - Partial output (ex: if only edl_system is needed): edl_system = update_edl_state(edl_system, TE, YE, Y, ITER_INFO)[0]
- Description: Processes simulation events to update the EDL system state (e.g., eject parachute, activate sky crane). Sets termination flags and initial conditions for the next simulation phase.
- Input Arguments:
    - edl_system (dict): Current system state.
    - TE (list): Event times.
    - YE (list): Event states.
    - Y (array): State history.
    - ITER_INFO (bool): Flag to print event details.
- Output Arguments:
    - edl_system (dict): Updated system state.
    - y0 (array): New initial conditions (empty if simulation ends).
    - TERMINATE_SIM (bool): Flag to stop simulation.

13. simulate_edl

- Calling Syntax: [T, Y, edl_system] = simulate_edl(edl_system, planet, mission_events, tmax, ITER_INFO)
- Description: Runs the EDL simulation by iteratively solving ODEs with event handling. Returns time, state history, and final system state.
- Input Arguments:

    - edl_system (dict): Initial system configuration.
    - planet (dict): Planetary constants.
    - mission_events (dict): Event thresholds.
    - tmax (float): Maximum simulation time (s).
    - ITER_INFO (bool): Print progress if True.
- Output Arguments:
    - T (array): Time vector.
    - Y (array): State history (each column corresponds to T).
    - edl_system (dict): Final system state.

Task 3

Purpose: The loop simulates the EDL system's descent through Mars' atmosphere, transitioning between operational phases (ex: parachute descent, rocket firing, sky crane deployment) by dynamically updating the system state (edl_system) and re-initializing the ODE solver (DOP853) after each event (ex: ejection, activation).

Key Components:
- DOP853 ODE Solver:
    - High-accuracy solver for the EDL dynamics (variable mass, thrust, drag, etc.).
    - Integrates the system state y over time using edl_dynamics.
    - Terminates at events (ex: parachute ejection altitude) defined in edl_events.
- update_edl_state:
    - Processes event triggers (ex: heat_shield_ejected = True).
    - Updates edl_system (ex: disables parachute, activates rockets).
    - Returns new initial conditions (y0) for the next simulation phase or terminates the loop.

Flowchart of the loop

```
START
 |
 ├── Initialize: y0, tspan, TERMINATE_SIM = False
 |
 |   [While not TERMINATE_SIM]
 |       |
 |       ├── DOP853: Solve edl_dynamics(t, y, edl_system, planet)
 |       |        over tspan, with events → TE, YE
 |       |
 |       ├── update_edl_state: Process TE/YE to update edl_system
 |       |              (e.g., eject parachute, fire rockets)
 |       |
 |       ├── TERMINATE_SIM? → Yes → Break
 |       |              No  → Update tspan, y0
 |       |
 |       └── Append results (T, Y)
 |
 └── Return T, Y, edl_system
```

Explanation

1. Initialization:
    - y0: Initial state vector (velocity, altitude, fuel mass, etc.).
    - tspan: Time window (t_current, tmax) for the solver.
2. While Loop (Termination Conditions):
    - Loop exits if:
        - Fuel is exhausted (fuel_mass <= 0).
        - Rover touches down (altitude + rover_rel_pos <= 0).
        - Crash detected (altitude_edl <= 0).
        - Simulation time exceeds tmax.
3. DOP853 Integration:
    - Solves dy/dt = edl_dynamics(t, y, edl_system, planet).
    - Stops at the earliest event (e.g., altitude < parachute_eject_altitude).
    - Returns:
        - TE: Time(s) of event(s).
        - YE: State(s) at event(s).
4. Event Processing (update_edl_state):
    - Checks which event occurred (e.g., event == 2 → rockets turned on).
    - Modifies edl_system (e.g., sets rocket['on'] = True).
    - Updates y0 for the next phase (e.g., resets error integrals for PID control).
5. Loop Continuation:
    - Updates tspan to (t_event, tmax).

- Appends solver results (T, Y) for plotting/analysis.

Physics Regimes and Phase Transitions

Phase	Trigger Event	Key Physics Changes
Parachute Descent	altitude < parachute_eject_altitude	Drag dominated (F_drag >> F_thrust)
Rocket Firing (Uncontrolled)	altitude < rockets_on_altitude	Thrust = 90% max; F_ext + F_thrust = ma
Speed-Controlled Descent	velocity < 3×target_velocity	PID adjusts thrust to maintain target velocity
Altitude-Controlled Hover	altitude < skycrane_activation	PID maintains altitude; sky crane lowers rover
Rover Touchdown	rover_rel_pos + altitude <= 0	Terminates loop; checks touchdown speed

How the Loop Simulates EDL Operational Phases

The while loop in simulate_edl makes up the entire descent process possible by iteratively:

1. Solving the ODE (DOP853) for the current physics regime (ex: parachute drag, rocket thrust).
2. Triggering events (ex: parachute ejection) when conditions are met.
3. Updating the system state (update_edl_state) to transition to the next phase. This ensures the correct physics (forces, control logic) are applied at each stage.

Task 4

Top down flowchart:

A[main_edl_simulation.py] -->|imports| B[define_edl_system.py]
A -->|imports| C[define_planet.py]
A -->|imports| D[define_mission_events.py]
A -->|imports| E[subfunctions_EDL.py]
E --> F[simulate_edl]
F --> G[edl_dynamics]

F --> H[edl_events]
F --> I[update_edl_state]

Detailed Execution Flow:

1. Initialization:
    - Calls three definition functions:
        - edl_system = define_edl_system_1() # Loads EDL system parameters
        - mars = define_planet()   # Defines Martian environment
        - mission_events = define_mission_events() # Sets mission milestones
2. Configuration:
    - Manually sets critical initial conditions:
        - edl_system['altitude'] = 11000   # Initial altitude [m]
        - edl_system['velocity'] = -590   # Initial velocity [m/s] (negative = descending)
        - edl_system['parachute']['deployed'] = True
3. Simulation:
    - Launches the main simulation routine:
        - [t, Y, edl_system] = simulate_edl(edl_system, mars, mission_events, tmax, True)
    - This triggers:
        - Continuous ODE solving via DOP853
        - Event handling through edl_events
        - System state updates via update_edl_state
4. Visualization:
    - Generates two figure sets:
        - Figure 0: All state variables (velocity, altitude, fuel mass, etc.).
        - Figure 1: Sky crane hover analysis (masks altitudes > 40m)

Function Dependency:

```
main_edl_simulation.py
├── from define_edl_system import define_edl_system_1
├── from define_planet import define_planet
├── from define_mission_events import define_mission_events
└── from subfunctions_EDL import simulate_edl
    ├── edl_dynamics()
    ├── edl_events()
    └── update_edl_state()

simulate_edl()  # from subfunctions_EDL.py
    ├── edl_dynamics()  # Called internally
    ├── edl_events()    # Called internally
    └── update_edl_state()  # Called internally
```

Key Data Flow:

- Inputs:
  - edl_system: Dictionary containing all vehicle parameters
  - mars: planet constants (gravity, atmosphere)
  - mission_events: Phase transition thresholds/triggers
- Outputs:
  - t: Time vector [s]
  - Y: State matrix (7×N) containing:
    - Velocity [m/s]
    - Altitude [m]
    - Fuel mass [kg]
    - Speed error integral
    - Position error integral
    - Rover relative velocity [m/s]
    - Rover relative position [m]

Critical Execution Path:

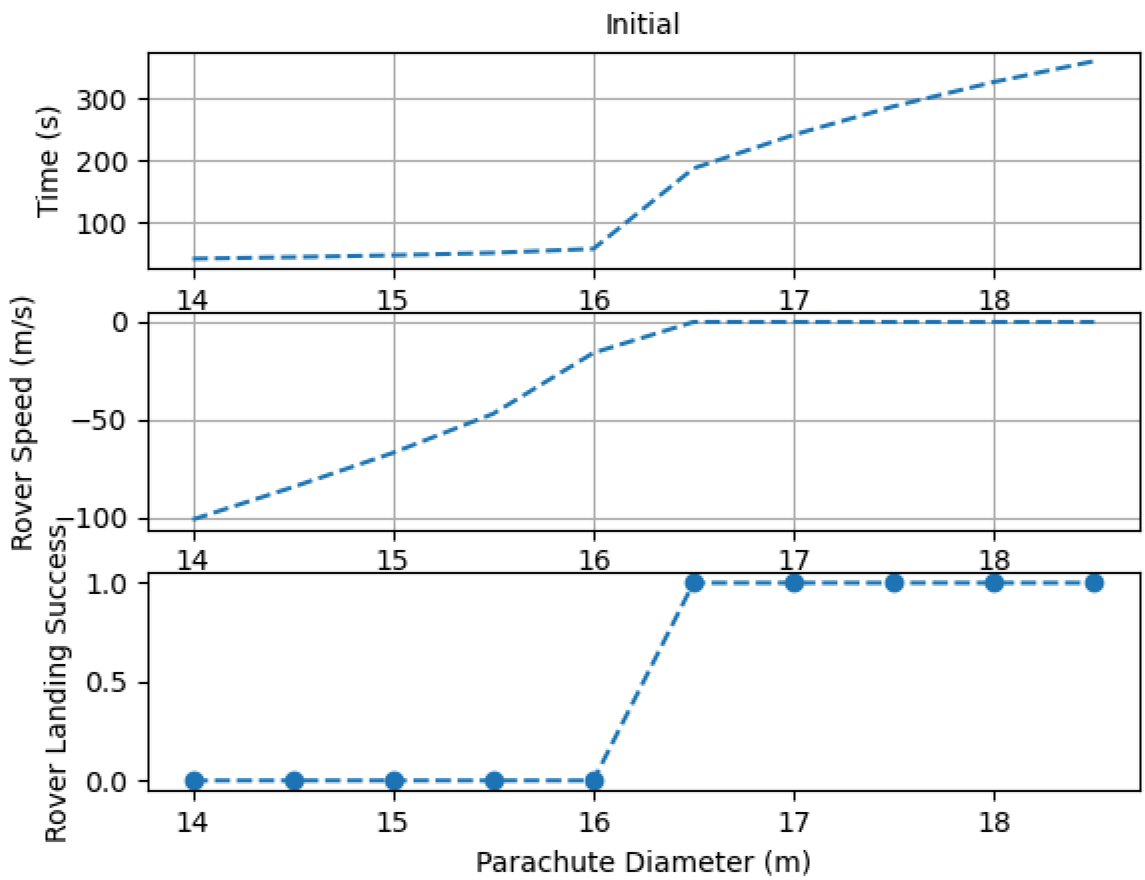
1. The script first loads all system definitions
2. Overrides specific initial conditions
3. Passes control to simulate_edl which:
   a. Manages the while loop
   b. Calls the ODE solver
   c. Processes events
   d. Updates system state
4. Finally generates diagnostic plots

Visualization Details:

- Figure 0 tracks all key states throughout descent
- Figure 1 specifically analyzes the sky crane hover phase by:
  - Masking altitudes > 40m (2× nominal hover height)
  - Plotting only the critical terminal descent phase

Task 5

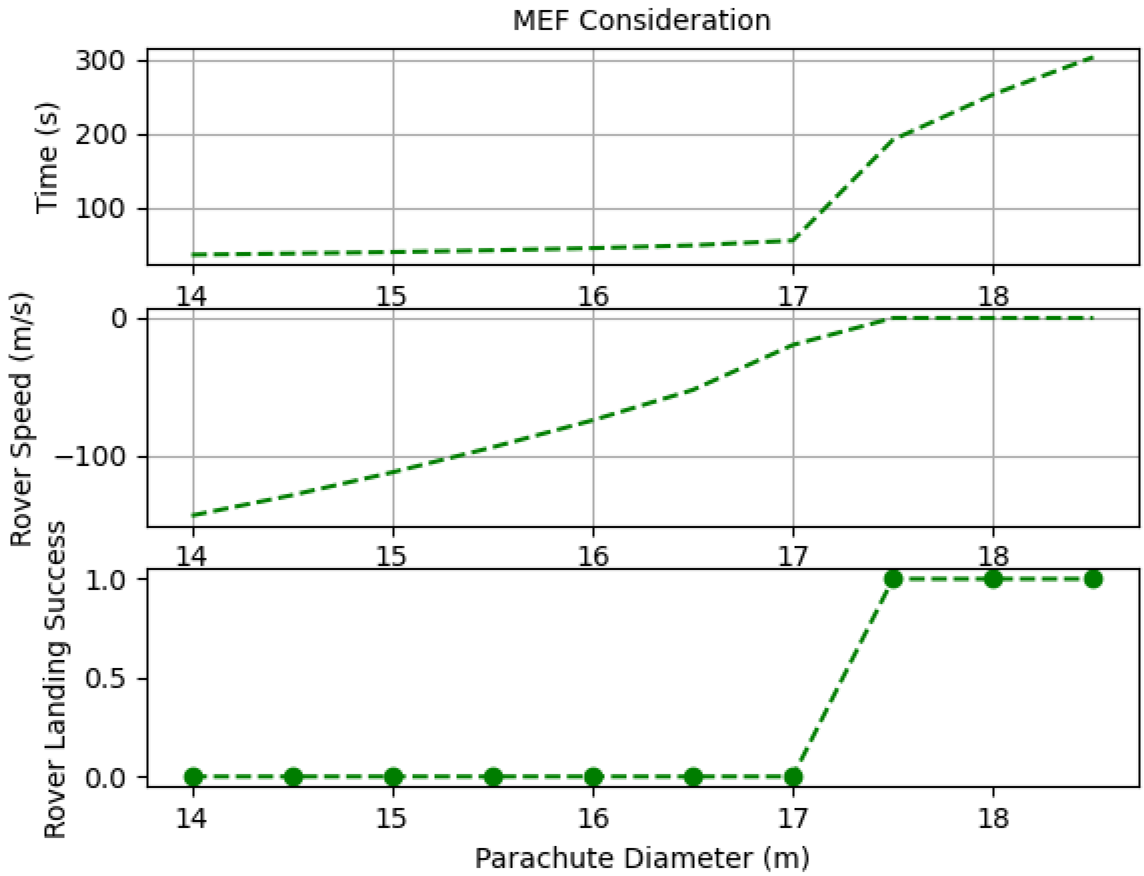
Python script file. Please set Task6 value to False in the subfunctions file to run it. See Task 6 for details.



It can be inferred from the plotted simulation results that a minimum parachute diameter of 16.5 meters is required to have a successful landing. This can be best explained by the rover's speed at landing. If it were nonzero, it would fail. Consequently, the lower velocity means a longer landing time, as evidenced by the first graph’s slower descent.

Task 6

Code is contained in the file. Modified Subfunctions (drag function and a few others affected) and Parachute file to account for the modified drag calculation. Added a boolean variable called “Task6” that determines the graph output of the blue Task 5 graph or the green Task 6.



(1)
A simple interpolation model was used for this part. For the sake of the model created to estimate MEF values given a velocity in Machs, a cubic model was chosen. This is because it has been expressly stated that the drag exhibits nonlinear behavior past Mach 1 and is therefore more unpredictable. If a regression model were used, it would be difficult to tell which order model would be the most accurate, even while utilizing the R^2 values to estimate data fidelity. As such, interpolation was the preferred method as it considered all the data points to be accurate (as is the stated case). This works especially well with the given data because it is 1-to-1 and does not resemble a scatter plot if considering the behavior before/after Mach 1 independently. Interp1d from the numpy library was used in conjunction with the given data for this part.

(2)
After taking a look at the difference between the two plots (before and after considering the MEF), there are slight differences that affect the outcome of the mission based on the parachute size. The most glaring of which are the failures at 16.5m and 17m in MEF, while the successes at those same values without it. After taking a look at the differences in theoretical velocity

touchdown speeds at those two points, it makes sense that would be the case (both are non-zero).

It would appear that by considering the MEF drag results in a less impactful parachute, meaning a larger diameter is needed to result in enough of a slowdown to trigger a soft enough landing to be considered successful.

Collaboration Statement

To save some time and typos, ChatGPT was asked to create lists of the given data so I did not have to manually retype it all into the IDE:

0.9	0.72
0.95	0.66
1.0	0.75
1.1	0.90
1.2	0.96
1.3	0.990
1.4	0.999
1.5	0.992
1.6	0.98
1.8	0.91
1.9	0.85
2.0	0.82
2.2	0.75
2.5	0.64
2.6	0.62

make this table into two variables (two lists) that contain the given stuff:

Here are the two lists extracted from the table in the image:

python Copy Edit

```python
Mach = [0.25, 0.5, 0.65, 0.7, 0.8, 0.9, 0.95, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 1.9, 2.0, 2

MEF = [1.0, 1.0, 1.0, 0.97, 0.91, 0.72, 0.66, 0.75, 0.90, 0.96, 0.990, 0.999, 0.992, 0.98, 0.91, 
```

Let me know if you want this as a CSV, plotted on a graph, or used in calculations!

This probably saved around 15 minutes that it would have taken to manually type each value into two lists.

Dr B's office hours and help time were also utilized to inspire the Task 6 method of choice (small subfunction and Boolean variable).