Task 1: Create Documentation for Data Dictionaries used in the code

parachute (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
deployed	True	N/A	True means the parachute has been deployed but not ejected
ejected	False	N/A	True means parachute is no longer attached to system
diameter	16.25	m	Diameter of parachute
Cd	0.615	N/A	Nominal coefficient of drag value for subsonic speed
mass	185.0	kg	Wild guess for mass of parachute

rocket (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
on	False	N/A	True means the rocket is on
structure_mass	8.0	kg	All rocket mass that is not fuel
initial_fuel_mass	230.0	kg	Initial mass of the fuel
fuel_mass	230	kg	Current fuel mass
effective_exhaust_velocity	4500	m/s	Velocity of the exhaust
max_thrust	3100	N	Maximum force of thrust
min_thrust	40	N	Minimum force of thrust

speed_control (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
on	False	N/A	True means control mode is activated
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 (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
on	False	N/A	True means control mode is activated
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	Desired target altitude; reflects sky crane cable length

sky_crane (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
on	False	N/A	True means lowering rover mode
danger_altitude	4.5	m	Altitude which it is too low for the rover to safely touch down
danger_speed	-1.0	m/s	Speed at which the rover would impact too hard on surface
mass	35.0	kg	Mass of the sky crane
area	16.0	m^2	Frontal area for drag

			calculations
Cd	0.9	N/A	Coefficient of drag
max_cable	7.6	m	Max length of cable for lowering rover
velocity	-0.1	m/s	Speed at which the sky crane lowers rover

heat_shield (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
ejected	False	N/A	True means the heat shield has been ejected from system
mass	225.0	kg	Mass of the heat shield
diameter	4.5	m	Diameter of the heat shield
Cd	0.35	N/A	Coefficient of drag of the heat shield

rover (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
define_rover_4	define_rover_4()	N/A	Subdictionary from define_rover_4 function

edl_system (From: define_edl_system.py)(Function: define_edl_system_1)			
Key	**Default**	**Units**	**Description**
altitude	np.NaN	N/A	The altitude that is updated during simulation
velocity	np.NaN	N/A	The velocity that is updated during simulation
num_rockets	8	N/A	The number of rockets
volume	150	kg/m^3	The volume of the system

parachute	parachute	N/A	Subdictionary from define_edl_systel_1 function
heat_shield	heat_shield	N/A	Subdictionary from define_edl_system_1 function
rocket	rocket	N/A	Subdictionary from define_edl_system_1 function
speed_control	speed_control	N/A	Subdictionary from define_edl_system_1 function
position_control	position_control	N/A	Subdictionary from define_edl_system_1 function
sky_crane	sky_crane	N/A	Subdictionary from define_edl_system_1 function
rover	rover	N/A	Subdictionary from define_edl_system_1 function

mission_events (From: define_mission_events.py)(Function: define_mission_events)			
Key	**Default**	**Units**	**Description**
alt_heatshield_eject	8000	m	Altitude at which the heat shield is commanded to eject.
alt_parachute_eject	900	m	Altitude at which the parachute is commanded to eject (cut away).
alt_rockets_on	1800	m	Altitude at which descent rockets are commanded to turn on.
alt_skycrane_on	7.6	m	Altitude at which the sky crane is commanded to begin operations.

high_altitude (From: define_planet.py)(Function: define_planet)			
Key	**Default**	**Units**	**Description**
temperature	lambda altitude: -23.4 - 0.00222*altitude	C	Linear model for atmospheric temperature vs. altitude above altitude_threshold on Mars.

pressure	lambda altitude: 0.699*np.exp(-0.0009* altitude)	kPa	Exponential model for atmospheric pressure vs. altitude above altitude_threshold on Mars.

low_altitude (From: define_planet.py)(Function: define_planet)			
Key	**Default**	**Units**	**Description**
temperature	lambda altitude: -31 - 0.000998*altitude	C	Linear model for atmospheric temperature vs. altitude below or equal to altitude_threshold on Mars.
pressure	lambda altitude: 0.699*np.exp(-0.00009 *altitude)	KPa	Exponential model for atmospheric pressure vs. altitude below or equal to altitude_threshold on Mars.

mars (From: define_planet.py)(Function: define_planet)			
Key	**Default**	**Units**	**Description**
g	-3.72	m/s^2	Gravitational acceleration on Mars.
altitude_threshold	7000	m	Altitude separating “low” and “high’ atmosphere models; used to switch between low_altitude and high_altitude
low_altitude	low_altitude (sub dictionary)	N/A	Dictionary containing temperature and pressure models for low altitude.
high_altitude	high_altitude (sub dictionary)	N/A	Dictionary containing temperature and pressure models for high altitudes.
density	Lambda T,P: P/(0.1921*(T+273.15))	kg/m^3	Equation of state giving atmospheric density as a function of temperature (C) and pressure (kPa)

wheel (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
radius	0.20	m	Wheel radius.
mass	2	kg	Mass of a single wheel.
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
radius		m	Wheel radius.
mass		kg	Mass of a single wheel.
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
radius		m	Wheel radius.
mass		kg	Mass of a single wheel.
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
radius		m	Wheel radius.
mass		kg	Mass of a single wheel.

speed _reducer (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
type	‘reverted’	N/A	Gives the type of speed reducer
diam_pinion	0.04	m	Diameter of the pinion
diam_gear	0.06	m	Diameter of the gear
mass	1.5	kg	Mass of the speed reducer

(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
type	‘standard’	N/A	Gives the type of speed reducer
diam_pinion	0.04	m	Diameter of the pinion
diam_gear	0.06	m	Diameter of the gear
mass	1.5	kg	Mass of the speed reducer
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
type	‘reverted’	N/A	Gives the type of speed reducer
diam_pinion	0.04	m	Diameter of the pinion
diam_gear	0.06	m	Diameter of the gear
mass	1.5	kg	Mass of the speed reducer
(Function: define_rover_1)			

Key	Default	Units	Description
type	‘reverted’	N/A	Gives the type of speed reducer
diam_pinion	0.04	m	Diameter of the pinion
diam_gear	0.07	m	Diameter of the gear
mass	1.5	kg	Mass of the speed reducer

motor (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
torque_stall	165	N*m	Torque at which the motor stalls
torque_noload	0	N*m	Torque at which the motor produces with no load
speed_noload	3.85	m/s	Motor speed that occurs at no load
mass	5.0	kg	Mass of the motor
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
torque_stall	180	N*m	Torque at which the motor stalls
torque_noload	0	N*m	Torque at which the motor produces with no load

speed_noload	3.7	m/s	Motor speed that occurs at no load
mass	5.0	kg	Mass of the motor
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
torque_stall	180	N*m	Torque at which the motor stalls
torque_noload	0	N*m	Torque at which the motor produces with no load
speed_noload	3.7	m/s	Motor speed that occurs at no load
mass	5.0	kg	Mass of the motor
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
torque_stall	170	N*m	Torque at which the motor stalls
torque_noload	0	N*m	Torque at which the motor produces with no load
speed_noload	3.80	m/s	Motor speed that occurs at no load
mass	5.0	kg	Mass of the motor

chassis (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
mass	674	kg	Mass of the chassis
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
mass	659	kg	Mass of the chassis

(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
mass	659	kg	Mass of the chassis
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
mass	659	kg	Mass of the chassis

science_payload (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
mass	80	kg	Mass of the payload
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
mass	75	kg	Mass of the payload
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
mass	75	kg	Mass of the payload
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
mass	75	kg	Mass of the payload

power_subsys (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
mass	100	kg	Mass of the power subsystem
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**

mass	90	kg	Mass of the power subsystem
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
mass	90	kg	Mass of the power subsystem
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
mass	90	kg	Mass of the power subsystem

wheel_assembly (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
wheel	wheel	N/A	Subdictionary containing the radius and mass of the wheel
speed_reducer	speed_reducer	N/A	Subdictionary containing the several parameters for the speed reducer
motor	motor	N/A	Subdictionary containing several motor paramenters
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
wheel	wheel	N/A	Subdictionary containing the radius and mass of the wheel
speed_reducer	speed_reducer	N/A	Subdictionary containing the several parameters for the speed reducer
motor	motor	N/A	Subdictionary

			containing several motor paramenters
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
wheel	wheel	N/A	Subdictionary containing the radius and mass of the wheel
speed_reducer	speed_reducer	N/A	Subdictionary containing the several parameters for the speed reducer
motor	motor	N/A	Subdictionary containing several motor paramenters
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
wheel	wheel	N/A	Subdictionary containing the radius and mass of the wheel
speed_reducer	speed_reducer	N/A	Subdictionary containing the several parameters for the speed reducer
motor	motor	N/A	Subdictionary containing several motor paramenters

rover (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of

			the chassis
science_payload	science_payload	kg	Subdictionary containing the mass of the science payload
power_subsys	power_subsys	kg	Subdicitonary containing the mass of the power subsystem
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
science_payload	science_payload	kg	Subdictionary containing the mass of the science payload
power_subsys	power_subsys	kg	Subdicitonary containing the mass of the power subsystem
(Function: define_rover_2)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
science_payload	science_payload	kg	Subdictionary containing the mass of the science payload
power_subsys	power_subsys	kg	Subdicitonary containing the mass of the power subsystem

(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
science_payload	science_payload	kg	Subdictionary containing the mass of the science payload
power_subsys	power_subsys	kg	Subdicitonary containing the mass of the power subsystem

rover (From: define_rovers.py)			
(Function: define_rover_4)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
(Function: define_rover_3)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
(Function: define_rover_2)			

Key	Default	Units	Description
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis
(Function: define_rover_1)			
Key	**Default**	**Units**	**Description**
wheel_assembly	wheel_assembly	N/A	Subdictionary containing several wheel assembly parameters
chassis	chassis	kg	Subdictionary containing the mass of the chassis

Task 2: Create Documentation for Each Function in subfunctions_EDL.py

get_mass_rover

1. Function Name: get_mass_rover
2. Calling Syntax:
   m = get_mass_rover(edl_system)
3. Description:
   Computes and returns the rover mass based on subsystem masses defined in the edl_system dictionary. Includes wheels, motors, speed reducers, chassis, payload, and power subsystem. Assumes Phase 1 dict format.
4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
5. Output Arguments:
   m: float - total rover mass [kg].

get_mass_rockets

1. Function Name: get_mass_rockets
2. Calling Syntax:

m = get_mass_rockets(edl_system)

3. Description:
   Computes current total rocket mass including structure and remaining fuel.
4. Input Arguments:
   edl_system: dict - must include rocket structure and fuel mass.
5. Output Arguments:
   m: float - total rocket mass [kg].

get_mass_edl

1. Function Name: get_mass_edl
2. Calling Syntax:
   m = get_mass_edl(edl_system)
3. Description:
   Computes the full EDL system mass including heat shield, parachute, sky crane, rockets, and rover. Accounts for components ejected.
4. Input Arguments:
   edl_system: dict - full EDL definition.
5. Output Arguments:
   m: float - total EDL mass [kg].

get_local_atm_properties

1. Function Name: get_mass_edl
2. 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)

3. Description:
   Computes the full EDL system mass including heat shield, parachute, sky crane, rockets, and rover. Accounts for components ejected.
4. Input Arguments:
   planet: dict contains data of the planet atmosphere at different altitudes
   altitude: int/float altitude of the EDL from the surface
5. Output Arguments:
   density: int/float - density of the atmosphere at the planet and altitude in kg/m^3
   temperature: int/float - returns temperature of the surrounding atmosphere in C
   pressure: int/float - returns the local pressure of the atmosphere in kPa

Note: this function is NOT vectorized. It will not accept a vector of altitudes.

F_buoyancy_descent

1. Function Name: F_buoyancy_descent
2. Calling Syntax:
   F = F_buoyancy_descent(edl_system,planet,altitude)
3. Description:
   Computes and returns the buoyancy force acting on the rover in Newtons
4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
   altitude: int/float altitude of the EDL from the surface
5. Output Arguments:
   F: float - Net buoyancy force [N].

F_drag_descent

1. Function Name: F_drag_descent
2. Calling Syntax:
   F = F_drag_descent(edl_system,planet,altitude,velocity)
3. Description:
   Computes and returns the buoyancy force acting on the rover in Newtons
4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
   altitude: int/float altitude of the EDL from the surface
5. Output Arguments:
   F: float - drag force acting on the rover [N].

F_gravity_descent

1. Function Name: F_gravity_descent
2. Calling Syntax:
   F = F_gravity_descent(edl_system,planet)
3. Description:
   Computes and returns the buoyancy force acting on the rover in Newtons
4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
5. Output Arguments:
   F: float - gravitational force acting on the rover [N].

v2M_Mars

1. Function Name: v2M_Mars
2. Calling Syntax:
   M = v2M_Mars(v, a)
3. Description:
   Converts the speed of the EDL in m\s to machs using the altitude.
4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
5. Output Arguments:
   M: float - absolute value Mach number

thrust_controller

1. Function Name: thrust_controller
2. Calling Syntax:
   F = thrust_controller(edl_system, planet)
3. Description:
   This function implements a PID Controller for the EDL system. Uses edl_system and planet structs to create a modified edl_system struct. Modifies fields in rocket and telemetry substructs.
4. Input Arguments:
   edl_system: dict — dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
5. Output Arguments:
   edl_system: dict — modified version of the edl_system.

edl_events

1. Function Name: edl_events
2. Calling Syntax:
   events = edl_events(edl_system, mission_events)
3. Description:
   Defines events that occur in EDL System simulation.
   y = [velocity, altitude, fuel_mass] and more

4. Input Arguments:
   edl_system: dict - dictionary containing rover subsystem mass fields.
   planet: dict contains data of the planet atmosphere at different altitudes
5. Output Arguments:

events: list — list of event functions corresponding to a EDl condition.

Additional Notes:
Corresponding events with each location in events array
[index]. Event description
0. Reached altitude to eject heat shield
1. Reached altitude to eject parachute
2. Reached altitude to turn on rockets
3. Reached altitude to turn on crane & altitude control
4. Out of fuel --> y(3)<=0. Terminal. Direction: -1.
5. EDL System crashed at zero altitude
6. Reached speed at which speed-controlled descent is required
7. Reached position at which altitude control is required
8. Rover has touched down on surface of Mars

edl_dynamics

1. Function Name: edl_dynamics:

2. Calling Syntax:
dydt = edl_dynamics(t, y, edl_system, planet):
3. Description:
This function implements a PID Controller for the EDL system. Uses edl_system and planet structs to create a modified edl_system struct. Modifies fields in rocket and telemetry substructs.
4. Input Arguments:
y: array - state vector of the rover. y = [vel_edl;pos_edl;fuel_mass;ei_vel;ei_pos;vel_rov;pos_rov]
edl_system: dict — dictionary containing rover subsystem mass fields.
planet: dict - contains data of the planet atmosphere at different altitudes
5. Output Arguments:
dydt: array - derivative of the state vector of y.

edl altitude, velocity and acceleration are absolute
rov is relative to edl
fuel_mass is total over all rockets

Note: this is a VARIABLE MASS SYSTEM, which means Newton's second law expressed as F=ma cannot be used. The more fundamental relationship is F = dp/dt, where p is the momentum of the system. (For a particle, p=mv and if m is constant you recover the F=ma form easily.) It is important

to consider the momentum contributions of the EDL system and propellant being expelled from the rocket. Ultimately you end up with something that roughly resembles Newton's second law: F_ext + v_rel*(dm/dt) = ma where m is the mass of the EDL, dm/dt is the rate at which mass is changing (due to propellant being expelled), v_rel is the speed at which propellant is being expelled, a is the acceleration of the EDL and F_ext is the sum of other forces acting on the EDL (drag, bouyancy, gravity). Since v_rel*(dm/dt) is a force, we can write this as F_ext + F_thrust = ma, which is very Newton-like.

update_edl_state

1. Function Name: update_edl_state
2. Calling Syntax:
   edl_system = update_edl_state(edl_system, TE, YE, Y, ITER_INFO)
   [edl_system, y0] = update_edl_state(edl_system, TE, YE, Y, ITER_INFO)
   [edl_system, y0, TERMINATE_SIM] = update_edl_state(edl_system, TE, YE, Y, ITER_INFO)
3. Description:
   Updates the EDl system depending on what event occurs and updates initial conditions.
   then determines if the simulation should stop. Modifies subsystem states
4. Input Arguments:
   edl_system: dict — dictionary containing rover subsystem mass fields.
   TE: list - contains an array of when the event is triggered
   YE: list of arrays - contains a state vector y at each event time
   Y: 2D array - states history of the current ODE integral
   ITER_INFO: boolean - enables printing of messages
5. Output Arguments:
   edl_system: dict - updates the EDL config based off events occured
   y0: array - the new state vector
   TERMINATE_SIM: Boolean - indicates if sim should stop

Corresponding events with each location in events array
[index]. Event description

0. Reached altitude to eject heat shield
1. Reached altitude to eject parachute
2. Reached altitude to turn on rockets
3. Reached altitude to turn on crane & altitude control
4. Out of fuel --> y(3)<=0. Terminal. Direction: -1.
5. EDL System crashed at zero altitude
6. Reached speed at which speed-controlled descent is required

7. Reached position at which altitude control is required
8. Rover has touched down on surface of Mars

"""

simulate_edl

1. Function Name: simulate_edl
2. Calling Syntax:
   T = simulate_edl(edl_system, planet, mission_events, tmax, ITER_INFO):
   [T, Y] = simulate_edl(edl_system, planet, mission_events, tmax, ITER_INFO):
   [T, Y, edl_system] = simulate_edl(edl_system, planet, mission_events, tmax, ITER_INFO):
3. Description:
   This simulates the EDL system. It requires a definition of the edl_system, the planet, the mission events, a maximum simulation time and has an optional flag to display detailed iteration information.

4. Input Arguments:
   edl_system: dict — dictionary containing rover subsystem mass fields.
   TE: list - contains an array of when the event is triggered
   YE: list of arrays - contains a state vector y at each event time
   Y: 2D array - states history of the current ODE integral
   ITER_INFO: boolean - enables printing of messages
5. Output Arguments:
   edl_system: dict - updates the EDL config based off events occured
   y0: array - the new state vector
   TERMINATE_SIM: Boolean - indicates if sim should stop

Task 3: Explain the loop in simulate edl.py

Before the loop starts, the event functions are built, and edl_events returns a list of 9 event functions (event0-event8) that trigger when key conditions are met, as discussed in Task 2. The initial time span and state vector are defined:

```
# simulation time span
tspan = (0, tmax)

# initial state of system
y0 = np.array([edl_system['velocity'],
               edl_system['altitude'],
               edl_system['rocket']['initial_fuel_mass'] * edl_system['num_rockets'],
               0,
               0,
               0,
               0])
```

```
# *** NOTE: This does not yet check for whether the fuel runs out...
if ITER_INFO:
    print('Commencing simulation run...\n')


# declare our variables (just empty arrays)
T = np.array([])
Y = np.array([[], [], [], [], [], [], []])
TERMINATE_SIM = False
while not(TERMINATE_SIM):
```

So at this point, the system is in the initial “entry/descent” configuration: heat shield still attached, parachute not yet employed, rockets off, skycrane off, rover still stowed.

The while loop structure is:

```
while not(TERMINATE_SIM):

    # run simulation until an event occurs
    fun = lambda t, y: edl_dynamics(t, y, edl_system, planet)
    sol = solve_ivp(fun, tspan, y0, method='DOP853', events=events, max_step=0.1)
    t_part = sol.t
    Y_part = sol.y
    TE = sol.t_events
    YE = sol.y_events

    # process the event and update the edl_system accordingly. Also sets
    # the initial conditions for the next stage (in y0) and the
    # TERMINATE_SIM flag.

    [edl_system, y0, TERMINATE_SIM] = update_edl_state(edl_system, TE, YE, Y_part, ITER_INFO)

    # update the simulation time span for the next stage
    tspan = (t_part[-1], tmax)

    # appending to grow a list is inefficient, but there is no way to
    # know in advance how many elements we'll need due to the adaptive step
    # size in ode45

    T = np.append(T, t_part)
    Y = np.hstack((Y, Y_part))
    #T.append(t_part)
    #Y.append(Y_part)


    # This looks for whether we're out of time. other termination
    # conditions checked in update_edl_state
    if tspan[0] >= tspan[1]:
        TERMINATE_SIM = True
```

Inside the loop, the function defines:

```
fun = lambda t, y: edl_dynamics(t, y, edl_system, planet)
```

While edl_dynamics computes the derivatives of the state, as explained in Task 2.

Then solve_ivp is used to integrate edl_dynamics forward in time using the higher-order DOP853 ODE solver:

```
sol = solve_ivp(fun, tspan, y0, method='DOP853', events=events, max_step=0.1)
```

It stops early when any of the event functions in the events hits zero (for example: altitude equal to the parachute deployment altitude).
Returns:

- T_part, Y_part: the time and state history for this phase
- TE, YE: the times and states at which each event occurred.

Key point: Within one pass of the loop, the configuration is fixed, so the integration corresponds to a single operational regime (for example: ballistic with parachute, power descent with rockets on, sky crane hover, and rover lowering).

Discrete event handling:(update_edl_state)

```
[edl_system, y0, TERMINATE_SIM] = update_edl_state(edl_system, TE, YE, Y_part, ITER_INFO)
```

Update_edl_state looks at TE and YE to see which events actually occurred during this segment and, for each such event, updates the physical configuration.
EX:

- Event 0: If TE[0] is nonempty and the shield is still on, it sets edl_system[‘heat_shield’][‘ejected’] = True, logs the time/altitude if ITER_INFO is true, and sets y(0) to the final state Y_part[:,-1] for the next phase.

- Event 1: Marks the parachute as ejected and removes its drag contribution

- Event 2: Sets edl_system[‘rocket’][‘on’] = True so that in the next loop pass, edl_dynamics will include thrust and fuel burn.

In all cases, update_edl-state chooses the new initial condition y0 for the next phase, usually taking y0 = Y_part[:,-1] (the last state from the just-completed integration), possibly with small tweaks.

Then there is updating the time span:

```
# update the simulation time span for the next stage
tspan = (t_part[-1], tmax)
```

The next phase starts at the final time reached in this phase and continues up to the original tmax unless an event sto[s it earlier. If tspan[0] >= tspan[1], no future time remains, and the loop is terminated as a safety check.

Then each phase's solutions are appended:

```
# appending to grow a list is inefficient, but there is no way to
# know in advance how many elements we'll need due to the adaptive step
# size in ode45

T = np.append(T, t_part)
Y = np.hstack((Y, Y_part))
#T.append(t_part)
#Y.append(Y_part)
```

The global arrays T and Y have now added parts. After the loop finishes, simulate_edl returns the full continuous trajectory of the system from entry to termination, including all phases and mode switches.

Task 4: Flowchart

https://lucid.app/lucidchart/60a91cdd-da2e-4eea-b671-eb5bf170151c/edit?viewport_loc=358%2C-706%2C2090%2C2053%2C0_0&invitationId=inv_04e0c275-2e23-4572-bba7-5f5c5b53b3f0

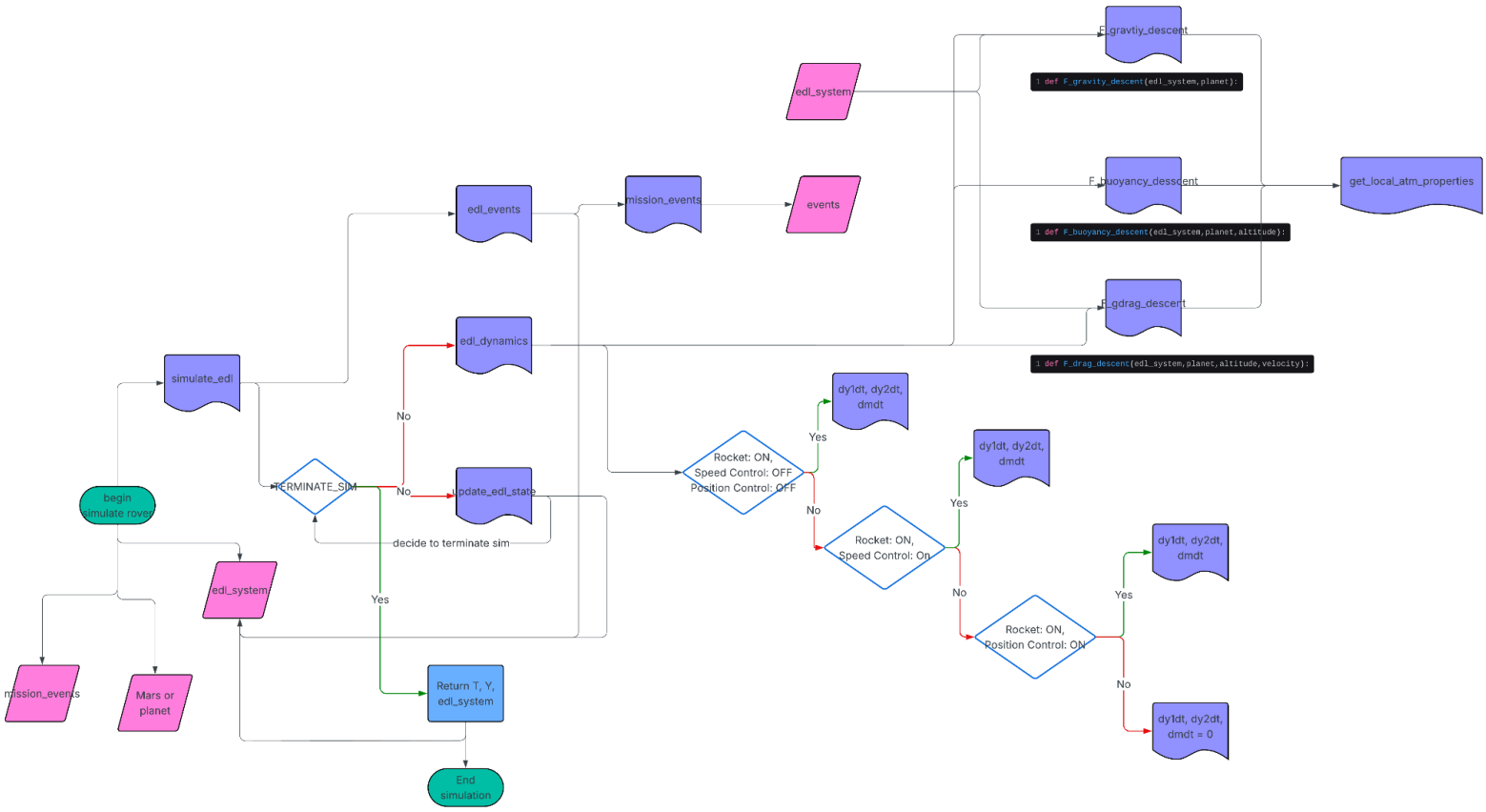


Figure 1: Flow Chart for Task 4

This flowchart begins with “begin simulate_rover” then ends with “End simulation”. Arrows pointing outward show a dependency or a call for that block. Ex: the arrow from “simulate_edl” to “edl_events” means that “simulate_rover” calls for “edl_events”. The arrows pointing to and from pink parallelograms show dependencies between that dictionary and the subfunction. Blue outlined rhombus’s represent if statements and if the True follow the green path, if False follow the red paths.

Task 5: Evaluate the Impact of Changing Parachute Size

Termination time grows gradually from 14 to 16 m and then increases sharply once the parachute reaches 16.5 m above, ultimately reaching ~380 s at 19m.

Interpretation:

- For 14-16m, drag is low, descent is fast, and the system reaches the ground quickly.
- At 16.5-19 m, drag becomes sufficiently large that the EDL system slows down dramatically, leading to much longer descent duration.

Conclusion Termination Time:

Large parachutes substantially increase decent time. Once safe landing is achieved additional increases in diameter only penalize mission efficiency without improving performance.

Touchdown speed improves consistently with parachute size:

- 14 m: ~-100 m/s
- 15 m: ~-80 m/s
- 16 m: ~-30 m/s
- 17 m and above: ~-0 m/s

Interpretation:

- Parachutes below 16 m cannot generate sufficient drag to slow the vehicle to safe level.
- At 16.5 m, drag increases enough to reduce the system's vertical velocity to nearly zero by touchdown.
- From 16.5 m through 19 m, the touchdown speed does not improve further because it is already at the lower physical limit enforced by the control logic and event sequence.

Conclusion Terminal Velocity:

From 16.5 to 19 m, the diameter is a good option for having a safe terminal velocity.

Conclusion Landing Success:

Landing is considered successful from ranges of 16.5 to 19 m diameters. However, it is optimal to have a successful landing with the minimized time, but a diameter that is safe within error. For this reason, a good target range is 16.5 to 17.5 m diameter. It is better to be closer to 17.5 m for safety reasons.

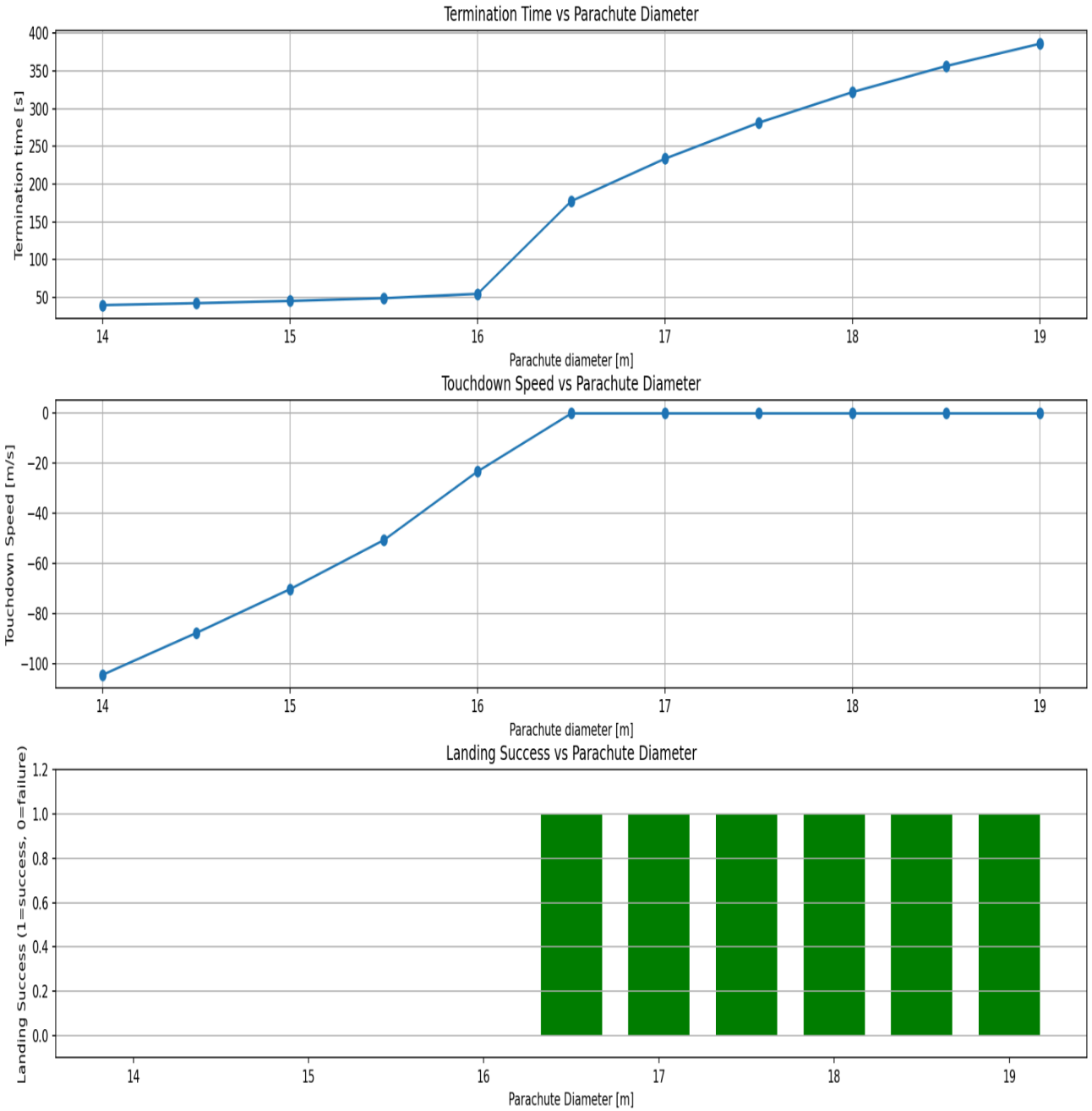
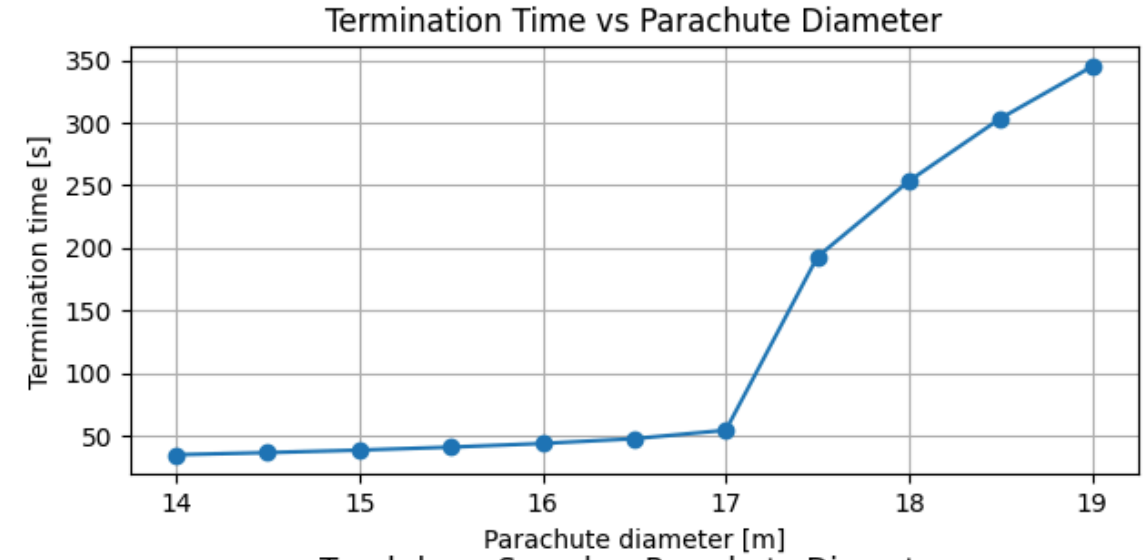
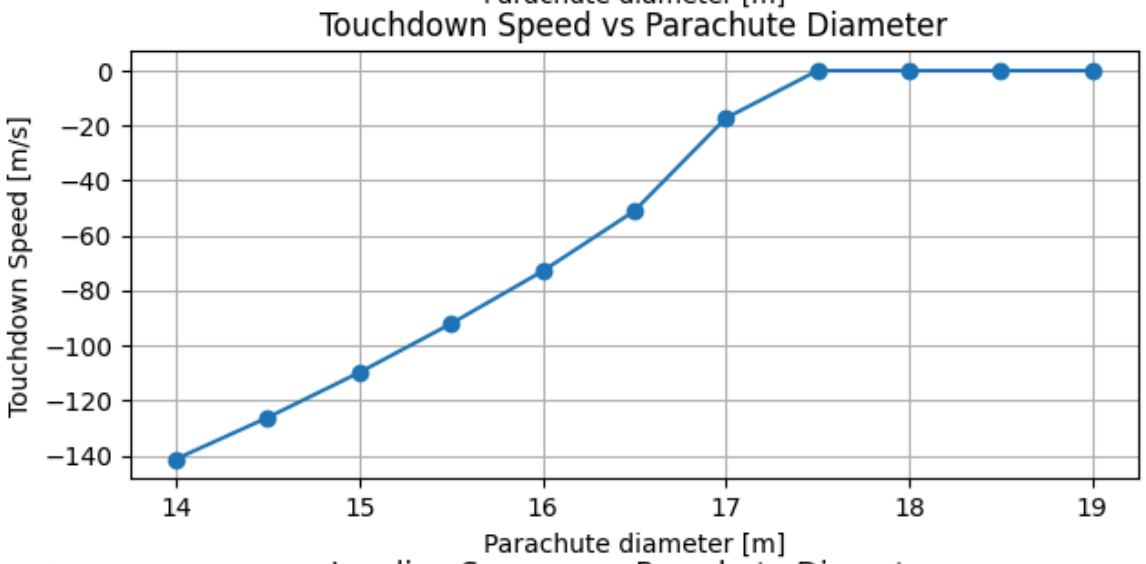


Figure 2: Task 5 Plots Disregarding MEF

Task 6: Improve the Parachute Drag Model





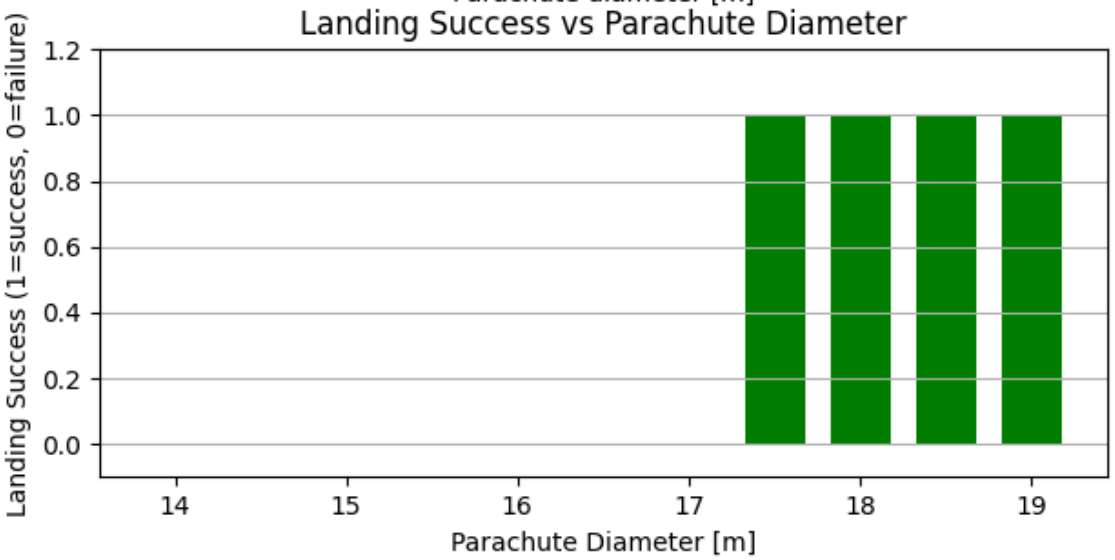


Figure 3: Task 6 Plot Regarding MEF

(1) how you created a continuous model for your *MEF* data, including the rationale for your modeling choices and (2) your assessment of whether the parachute diameter recommendations from Task 5 require modification. Please include any supporting data and graphs in your report. If you created any new scripts/functions to support your analysis, please name them and explain what they do in your report(and be sure to submit them on Canvas

(1) To achieve a complete model for the MEF data, we thought it would be best to linearly interpolate between the two mach numbers and two MEF values. This would give us a continuous model for the MEF between the min and max values. We thought linear interpolation was best because of the uneven step sizes of Mach numbers.
(2) The parachute recommendation does require modification. The previous recommendation was 16.5m - 17.5m diameters. With the improved model, we now believe that a 17.5m - 18.5m in diameter is better. It is better to be closer to 18.5 m for safety reasons.
    (a) MEF_from_Mach(m) function was created to help convert a mach number into a MEF using linear interpolation.