MEEN 357 DESIGN PROJECT
PHASE 3: ENTRY, DESCENT AND LANDING

Revision history:
- ~

Assigned:	4 November	**Submission:**	Electronic
Due Date:	18 November	**Collaboration Type:**	Project Teams
Due Time:	11:59pm	**Grading:**	75 points

Contents

1 OVERVIEW OF PHASE 3

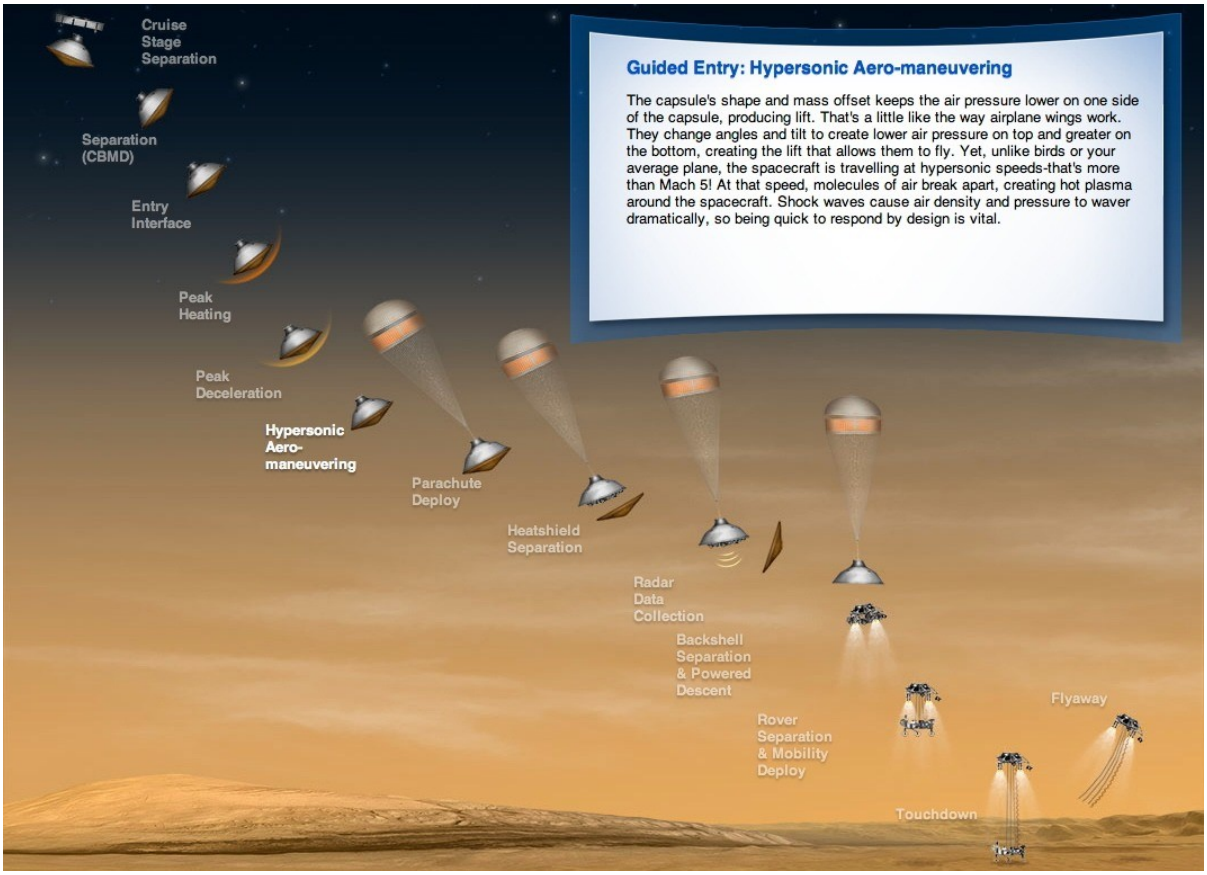

Figure 1. Entry Descent and Landing, also known as the 'Seven Minutes of Terror'. Courtesy of NASA/JPL

1.1 The Entry, Descent and Landing Process

As you already were told, the landing concept for Marvin is similar to that of the Mars Curiosity Rover, which required a novel Entry, Descent and Landing system design. Figure 1 is an illustration of the essential stages of the process:

1. EDL System enters Mars atmosphere at hypersonic speeds and proceeds to slow down thanks to friction with the atmosphere.
2. Once a key altitude is reached (with the vertical velocity of the EDL system still at about 450 m/s and with the heat shield still on), the parachute---the largest parachute ever deployed in a planetary mission---is deployed.
3. The heat shield is ejected and the EDL system continues its parachute descent.
4. The power descent sequence is initiated using variable-thrust solid rockets.
5. The parachute is ejected and the EDL continues its deceleration through the rocket braking system.
6. Once the EDL system reaches a sufficiently low altitude, it proceeds to lower the rover using the so-called sky crane.
7. Once the rover touches ground, the sky crane flies away.

1.2 Your Challenge

Employee turnover is a fact of life in all professional endeavors, including engineering. People may leave for other companies, other careers, due to medical or family issues, or simply are relocated elsewhere within a company (e.g., promotion or transfer).

Your team has been tasked with taking over the EDL analysis where another team has left off. You are given their Python code and a description of the system physics, but that is all. No other detailed documentation exists. Furthermore, the members of the other team no longer are with the company, so your team does not have the luxury of asking them questions.

The following is a description of the tasks you must complete for your assignment. The preexisting code is available in a zip file distributed with this assignment. The physics of the EDL system are described in the appendix of this document.

2 TASKS

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

The previous team used Python dictionaries extensively in their code. These dictionaries are defined in separate files that are called from the main script. Although they did an okay job at commenting the files that define the dictionaries, they made no formal documentation of their contents. Your task is to go through the code and create documentation for the dictionaries. For each dictionary, you should identify and describe every one of its fields (some fields may themselves be dictionaries). Be sure to include the name of the field, its default value, its units (if applicable) and a description of its meaning.

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

Your predecessors created many functions in Python, all of which are included in the zip file in subfunctions_EDL.py. Create documentation for each of these. This documentation should include the following:

1. **Function name**. Please get the case correct.
2. **Calling syntax**. There may be more than one valid way to call this function. The calling syntax should list all valid variations on how the function can be called.
3. **Description**. This describes how the function behaves. This may depend on how the function is called, so be sure to clarify such dependencies. Please note any physics of the EDL that are modeled by the function.
4. **Input arguments**. List the input arguments in their calling order, state their "type" (vector, scalar, string, etc.), define their meaning, and, if applicable, note their units. If an input argument is optional, note this fact and, if applicable, indicate its default value.
5. **Output arguments**. List the arguments in their return order, state their type, and define their meaning and, if applicable, units.

You can use the documentation approach we have been using in previous phases of the project as a rough template.

2.3 Task 3: Explain the Loop in simulate_edl

There is a while loop inside the function `simulate_edl`. The purpose of this loop is to simulate the EDL system as it descends through the Martian atmosphere and lowers the rover to the Martian surface. Notice that there is a call to an ODE solver (specifically, `DOP853` because we need the higher accuracy over `RK45`) and a call to another function, `update_edl_state`, that is in subfunctions_EDL.py as well. The loop continues until some termination condition is met. The ODE solver is run every pass through the loop, but with different initial conditions and, potentially, a redefined `edl_system` dictionary. Since the purpose of this loop is to simulate the EDL system, any redefinition of `edl_system` has to do with changes in the physical operation of the system.

Your team is instructed to create detailed documentation of how this loop works. This includes explaining how calls to `DOP853` and `update_edl_state` work together to simulate the appropriate physics regime as the system goes through its different operational phases (parachute only, firing rockets, etc.). It is recommended that you create a flowchart or other illustration to support your explanation.

2.4 Task 4: Document the Execution Flow of the Code

One thing you are told about this code is that executing the script `main_edl_simulation.py` launches a simulation of the EDL process. Your team must document the dependencies between functions (which functions/scripts call other functions and in what order). Create a flow chart, block diagram, or something similar to illustrate the execution flow of the code. Include an explanation along with your illustration in your report.

2.5 Task 5: Evaluate the Impact of Changing the Parachute Size

Your team has been asked to conduct a study of the impact on EDL performance of changing the parachute size. For this task, you must do two things: (1) create a Python script called `study_parachute_size.py` that conducts the analysis and generates useful visualizations of the results (defined below) and (2) include in your report an interpretation of these results. Use the following initial conditions:

EDL System Altitude	11 km
EDL System Velocity	-578 m/s
Rockets	Off
Parachute	Deployed, not ejected
Heat shield	Not ejected
Sky crane	Off
Speed controller	Off
Position controller	Off

The parachute study should consider parachute diameters from 14 to 19 meters at 0.5 meter intervals. The critical figure of merit is whether the rover reaches the ground safely. Your script should create a 3x1 array of plots (using the plt.subplots command):

1. Simulated time (i.e., time at termination of simulation) vs. parachute diameter
2. Rover speed (relative to ground) at simulation termination vs. parachute diameter
3. Rover landing success (1=success; 0=failure) vs. parachute diameter

Make sure each graph has its axes labeled clearly with associated units.

In your interpretation, please make a well-supported recommendation as to the appropriate diameter or range of diameters for the parachute. The critical requirement is that the rover reaches the ground safely. A secondary consideration is the time it takes to get the rover on the ground (we prefer to minimize this time).

2.6 Task 6: Improve the Parachute Drag Model

The drag model used in the code you are given involves a strong assumption: that the coefficient of drag for the parachute is independent of the speed of descent. We know this is incorrect from physical experimentation on similar parachutes. In particular, drag near and above Mach 1 is nonlinear. However, it is unclear whether this idealization is significant for our purposes (i.e., it may or may not affect conclusions we draw about the design). Your team is tasked with revising the drag model and reexamining your parachute diameter recommendations using the revised model.

You can model the modified parachute drag using the following relationship:

$$C_{D,mod} = MEF(M) \cdot C_D,$$

where C_D is the coefficient of drag at sub-Mach speeds and MEF is a Mach efficiency factor that is determined experimentally and is a function of the descent velocity, M, expressed in Mach number. You can used the function `v2M_Mars` (included with your code) to convert from speed in m/s to a Mach number.

You will need to create a model for the MEF value at a given Mach number. You are given the following data (see table) obtained from tests of similar parachute systems.

Modify the drag code included in the code distribution to use this new model. Be sure to submit this version with all your code on Canvas.

In your report, you must document: (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 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).

Mach	MEF
0.25	1.0
0.5	1.0
0.65	1.0
0.7	0.98
0.8	0.90
0.9	0.72
0.95	0.66
1.0	0.76
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	0.90
1.9	0.85
2.0	0.82
2.2	0.75
2.5	0.65
2.6	0.62

3 SUBMISSION PROCEDURES

To submit:	The Canvas submission will be by group number, so only one submission from each team is necessary. The submission is completely electronic.
File convention:	Please submit all your code and other files in a .zip archive. Please name this **P3_MEEN357_FA2022_TEAM**.zip**, where ** is your team number**. Please avoid having subfolders in your zip archive. The names of Python files are specified elsewhere in this document
What to submit:	1) Function File: updated drag model function file 2) Script File: study_parachute_size.py 3) Report with requested documentation, graphs, etc.

A. APPRENDIX: PHYSICAL BACKGROUND

System Operation

As noted previously, Figure 1 is an illustration of the essential stages of EDL process. The following is a slightly more detailed explanation of the key steps in the EDL process:

1. EDL System enters Mars atmosphere at hypersonic speeds and proceeds to slow down thanks to friction with the atmosphere. *These events occur before our simulation begins and are not explicitly represented in our code.*
2. The parachute is deployed at an altitude of 11 km.
    a. The physics are governed by the craft's ballistic trajectory. Forces involved are due to aerodynamic drag and buoyancy forces. (Note: we will worry only about the altitude, but in the real craft there is a horizontal component.)
3. The heat shield is ejected at an altitude of 8 km.
    a. Although the physics in principle are unchanged (still a ballistic trajectory), there is a discrete change in the dynamics due to the reduced mass and changed drag coefficient due to having ejected the heat shield.
    b. We'll assume the heat shield ejection itself does not affect the descent dynamics in an appreciable way. In actuality, there may be some explosive bolts or other pyrotechnic devices that execute the ejection process. Since the EDL would be the reaction mass for this ejection, there would be a very slight modification to the trajectory. However, this is negligible for analyses at our level of abstraction. We will assume the ejection is instantaneous and has no effect on the EDL trajectory (though the reduced mass and increased drag will have an effect).
4. The power descent sequence is initiated at 1.8 km using variable-thrust solid rockets.
    a. The physics change significantly at this point. We now are firing rockets to slow our descent. This means the craft is changing mass (due to the propellant leaving the rockets) and there is an effective force on the craft due to the conservation of momentum. It is

important to account for the effective thrust force and the steadily reducing mass. There now is a new differential equation to model the change in EDL system mass.

   b. Initially, the rockets fire at a constant thrust level of 90% of their maximum. Eventually, a speed controller and, later, a position controller take over to modulate the thrust output.

5. The parachute is ejected at an altitude of 900 meters. The rockets continue to fire through this event.
   a. This represents a discrete change in both system mass and effective drag coefficient.
   b. Like the heat shield, pyrotechnic devices are used to eject the parachute. We assume the impact of these on the EDL descent is negligible in our analysis.
   c. We do not model what happens to the parachute after being ejected. In the real system, it is placed on a different trajectory than the EDL and lands out of the way.
6. A speed controller is used to set the EDL system on a constant-speed descent trajectory so that it can transition to the sky crane stage of operation. It functions to keep the craft descending at a fixed speed. The controller engages when the EDL is traveling at three times the targeted speed.
   a. This represents a discrete change in physics since the controller itself impacts the dynamics of the craft.
   b. The controller is a PID speed controller with a constant offset term to account for the effect of gravity. It is explained later in the appendix.
   c. The net effect of the controller is to modulate thrust output from the rockets.
7. A position controller is used to command the craft to hover at an altitude appropriate for sky crane operation. This altitude is 7.6 meters. The position controller engages when the craft is at about 9.1 meters in altitude.
   a. This represents another discrete change in physics. The craft physics are unaffected, but the controller dynamics change. This results in an overall change in dynamics.
   b. Like the speed controller, the position controller is a PID type with a constant offset to account for gravity.
   c. Like the speed controller, the position controller affects the thrust output of the rockets.
8. Once the EDL system reaches an altitude of 7.6 meters, it proceeds to lower the rover using the so-called sky crane. The rover is lowered via cable at a constant speed.
   a. Although one certainly could model the dynamics of the rover lowering process (winch, motor, etc.) in a complicated way, it is sufficient for our analyses to model it as being lowered at a constant speed relative to the sky crane.
   b. Because the rover is being lowered relative to the sky crane, one needs to look at their relative positions and the altitude of the sky crane (i.e., what remains of the EDL craft) to determine whether the rover has touched down.
9. Once the rover touches ground, the sky crane flies away. *The fly-away stage is not simulated in our code.*

Mission Success Criteria

The principal concern is for the rover to have landed safely. A safe landing condition is defined as follows:

1. The rover speed at landing is no more than 1.0 m/s.
2. The sky crane altitude is no less than 4.5 m.

The first condition ensures that the rover does not touch down with too much momentum, which could damage the scientific instruments. The second condition is to ensure that the sky crane is high enough to clear the rover with some margin for the fly away operation.

Ballistic Trajectory Dynamics

The dynamics of the ballistic trajectory during EDL system descent is simply the balance of forces acting on the craft via Newton's second law:

$$F = m\ddot{y}$$

where F is the net force acting on the EDL system, m is the total mass of the EDL system, and $\ddot{y}$ is the acceleration of the EDL system (i.e., y is the altitude). This is a second order ODE.

The mass simply is the sum of the masses of the components. The net force has three components:

1. Drag forces, which act to slow descent. This force may have multiple components depending on the state of the EDL system (whether the heat shield and parachute remain on the craft).
2. Buoyancy force of the EDL system displacing the Martian atmosphere. This acts to slow descent.
3. Gravity, which acts to speed descent.

Aerodynamic Drag

The aerodynamic drag force arises from the pressure distribution and friction forces as a body moves through a fluid. The drag force is given by:

$$F_d = \frac{1}{2}\rho v^2 A C_d,$$

where ρ is the density of the fluid, v is the velocity of the object, relative to the fluid, A is the cross section of the object in the direction perpendicular to the flow, and C_d correspond to a non-dimensional drag coefficient that encloses all types of dissipative interactions that contribute to aerodynamic drag. In the EDL system, there are two major sources of drag: the drag force due to the parachute and the drag force due to the body suspended from the parachute. The parachute is by far the greater source of drag force. The drag model is the same for both, but each has a different coefficient of drag. The total drag is the sum of the drag forces.

Buoyancy

The buoyancy forces are insignificant compared to aerodynamic drag, but can be included for the sake of completeness. These forces originate from the volume of fluid displaced by the object immersed in it. The buoyancy forces are given by:

$$F_b = \rho V g$$

where ρ corresponds to the density of the fluid, V is the volume of the body (the EDL system) and g is the acceleration due to gravity.

Powered Descent

During powered descent, the EDL system fires its rockets to further reduce its descent speed. The thrust of the rocket is due to the reactive force originating by the exhaust of hot gases through the rocket's nozzle. Although it fundamentally is a conservation-of-momentum phenomenon, we can think of there being an equivalent force created by the thrust. This is given by

$$F_T = u_e \frac{dm(t)}{dt}$$

where u_e is the effective exhaust velocity of the gasses and $dm(t)/dt$ is the time rate of change of EDL system mass due to the expulsed gases. In most rockets of the type used for the EDL mission, this exhaust velocity is in the order of 4,500-5000 m/s.

In many cases it is advantageous to use the preceding relationship to deduce the change in mass when given a thrust level.

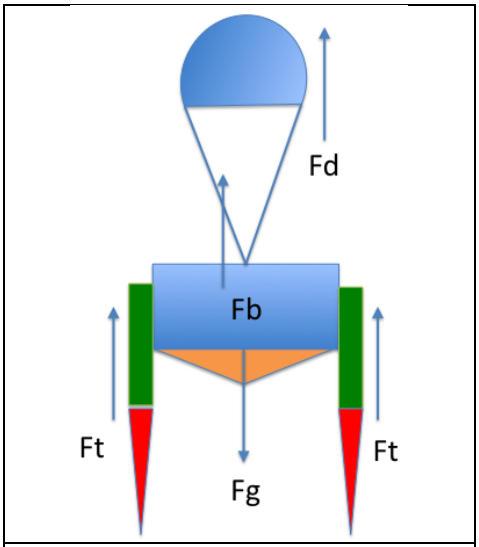

Fig. 2 Free body diagram on the EDL system under powered descent with the parachute and heat shield still attached to the craft.

PID Speed Control

The EDL system uses a proportional-integrative-derivative (PID) speed controller to modulate the amount of rocket thrust in order to achieve a desired speed. The general structure of a PID controller is given in Fig. 3. In our case, the plant is the EDL system. This structure is the same for both speed and position control, but the definition of the feedback signal changes (descent speed, $\dot{y}$, for speed control and altitude, y, for position control). Each controller uses its own gains.

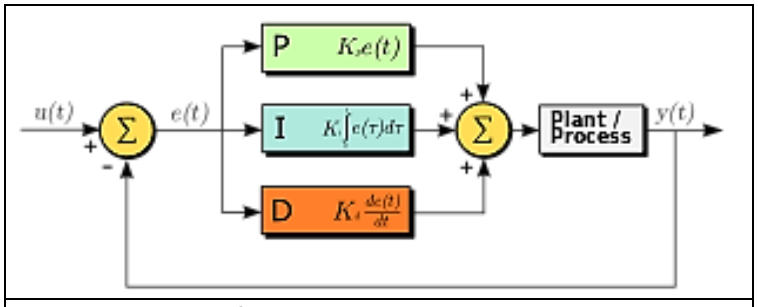

Fig. 3 Schematic of a generic PID control scheme

For the speed controller, error is defined as the difference between a desired speed and the actual speed of the EDL system:

$$e = \dot{y}_{tgt} - \dot{y}_{act}$$

where $\dot{y}_{tgt}$ is the targeted speed and $\dot{y}_{act}$ is the actual speed. The speed controller used by our EDL system has the form:

$$u(t) = K_p e(t) + K_i \int_{-\infty}^{t} e(t)dt + K_d \dot{e}(t) \tag{1}$$

The output, $u(\cdot)$, relates to the thrust from each rocket. The gains—i.e., the K_p, K_i, and K_d constants—determine the relative importance of each error indicator (proportional, integral and derivative, respectively). The total commanded thrust from the rockets is calculated as

$$F_T(t) = N_{rockets} u(t) - m(t)g \tag{2}$$

where $N_{rockets}$ is the number of rockets in the EDL system, g is the acceleration due to gravity, and $m(t)$ is the total EDL system mass at time t (remember that mass is time varying). The mass-gravity term is an easy way to compensate for the known effect of gravity.

Note that the PID controller expression, Equation 1, is a second-order ODE. It may seem a little awkward to implement this in an ODE solver such as Python's RK45 due to the integral in the expression. However, the relationship between $e(t)$ and its integral is the same as between $\dot{e}(t)$ and $e(t)$—i.e., one is the derivative of the other. Thus, we can introduce the integral of $e(t)$ as a state variable and RK45 (or whatever solver you are using) will update it for us as the simulation progresses.

One interesting complication of the speed control ODEs is that it is not possible to compute $\dot{e}(t)$ explicitly. Thus, it is necessary to combine Equations 1 and 2 with the differential equations of motion for the EDL system in order to arrive at a series of expressions that can be used in with the ODE solver.

Finally, it always is important to verify that the commanded thrust force is within the valid bounds of what the rocket motors can produce. There is both a maximum and minimum thrust (and thrust cannot change sign). If the equivalent thrust force computed in Equation 2 is greater than what the rockets can produce, it is reasonable to assign F_T the maximum allowable thrust. A similar approach can be used at the lower bound of thrust.

Altitude (Position) Control

Position control is similar to speed control except the error signal is a function of altitude:

$$e = y_{tgt} - y_{act}$$

where y_{tgt} is the desired altitude and y_{act} is the actual altitude. The controller follows the same structure as given in Equations 1 and 2.

It turns out that the position controller is easier to implement in Python than the speed controller because all the key terms can be obtained explicitly.

As with the speed controller, it is necessary to verify that the limits of the rockets are not being exceeded.