4.1.6 Problem Set: Martian lander implementation

Bookmark this page

In the remainder of this pset, we will model the entry of a Martian lander in the Martian atmosphere from initial entry through the parachute deployment. The specific lander you will model is from the *Curiosity* mission, which was the predecessor to the recent *Perserverance* mission. If you want to learn more, check out the background references provided in Section 4.1.8.

The basic model in `lander.py` is an extension of the hail model, however, with the following modifications:

- The variation of the atmospheric properties with altitude will be significant. For example, you will need to account for $\rho_a = \rho_a(z)$ where z is the altitude. We will also account for gravitational variations $g(z)$, however, this is a smaller variability of roughly 10% from the entry into the atmosphere to the Martian surface. Note that the method `LanderIVP.atmosphere` already implements a model to calculate these properties (see the docstring for more details).

- As shown in Figure 4.6, the Martian lander flies at a significant angle θ relative to gravity and this impacts the model. As a result, the model equations will take the following form,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} V \\ z \end{bmatrix} = \begin{bmatrix} g\cos\theta - D/m_l \\ -V\cos\theta \end{bmatrix} \tag{4.12}$$

  where m_l is the mass of the lander (includes everything, i.e. parachute, fuel, etc.). For simplicity, we will assume that θ does not change except when the parachute deploys. Thus, from entry until the deployment of the parachute, $\theta = \theta_e$ where θ_e is a constant. And after parachute deployment, $\theta = \theta_p$ where θ_p is a constant (generally different from θ_e).

- The deployment of the parachute must be accounted for. The parachute deployment for a Martian lander like *Curiosity* occurs when the velocity of the lander has decelerated sufficiently to V_p. Thus, once $V \leq V_p$, the parachute will deploy. In the case of *Curiosity*, $V_p \approx 470\,\mathrm{m/s}$. We will model the drag on the lander and parachute in a similar manner to the hail using drag coefficients. Specifically, the drag when the parachute deploys will be:

$$D = D_l + D_p \tag{4.13}$$

$$D_l = \frac{1}{2}\rho_a V^2 A_l C_{Dl} \tag{4.14}$$

$$D_p = \frac{1}{2}\rho_a V^2 A_p C_{Dp} \tag{4.15}$$

  where D_l is the drag acting on the surface of the lander and D_p is the drag acting on the parachute. And A_l and C_{Dl} are the projected area and drag coefficient of the lander, and similarly A_p and C_{Dp} for the parachute. When the parachute is not deployed, then the only drag will be from the lander, i.e. $D = D_l$.

  Also, don't forget that the flight path angle θ changes when the parachute deploys (see previous item)!

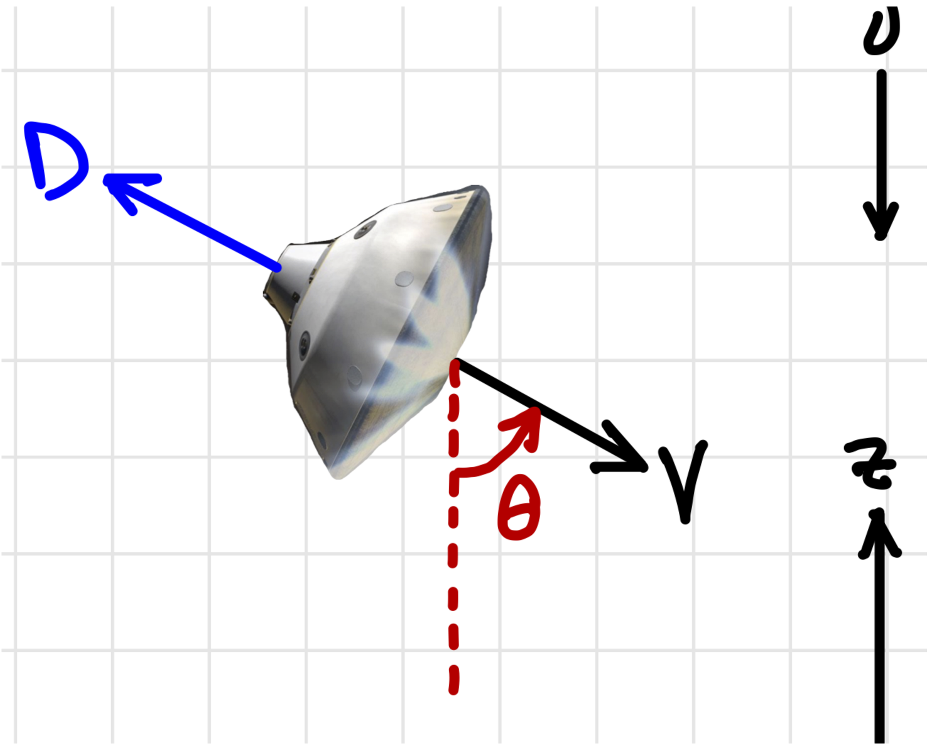


Figure 4.6: Martian lander showing its velocity V, drag force D, gravity g, and flight angle θ.
In this pset, we will use the following values which are roughly consistent with the *Curiosity* lander:

$$t_I = 0, \qquad t_F = 300\,\mathrm{s}, \qquad \mathrm{V}\left(\mathrm{t_I}\right) = 5800\,\mathrm{m/s} \tag{4.16}$$

$$V_p = 470\,\mathrm{m/s}, \qquad m_l = 3300\,\mathrm{kg}, \qquad z\left(t_I\right) = 125,000\,\mathrm{m} \tag{4.17}$$

$$A_l = 15.9\,\mathrm{m}^2, \qquad C_{Dl} = 1.7, \qquad \theta_e = 83^\circ \tag{4.18}$$

$$A_p = 201\,\mathrm{m}^2, \qquad C_{Dp} = 1.2, \qquad \theta_p = 70^\circ \tag{4.19}$$

You only need to run the simulation using the RK4 method. With this method, we have found that $\Delta t = 0.1$ s gives accurate results (though we encourage you to experiment to see how the results depend on the Δt chosen).

To complete the Martian lander model, you will finish and then execute the `lander.py` module. Specifically:

1. **Complete `LanderIVP.evalf(self, u, t)`.**

   Don't forget to account for whether or not the parachute is deployed both for the drag and flight angle! Hint: `HailIVP.evalf` is a good starting place!

   Note: The parameters $\{m_l, A_l, A_p, \ldots\}$ have already been loaded into a dictionary for you. To get the properties of the Martian atmosphere, use `self.atmosphere(z)`.

   Also note that `math.cos` takes inputs with units of radians.

2. **Implement `lander_Vzaplot(...)`.**

   See Figure 4.7 for the desired plot format.

   - Hint: `hail_Vzplot` is a good starting place!
   - Remember to return the `Axes` object(s).

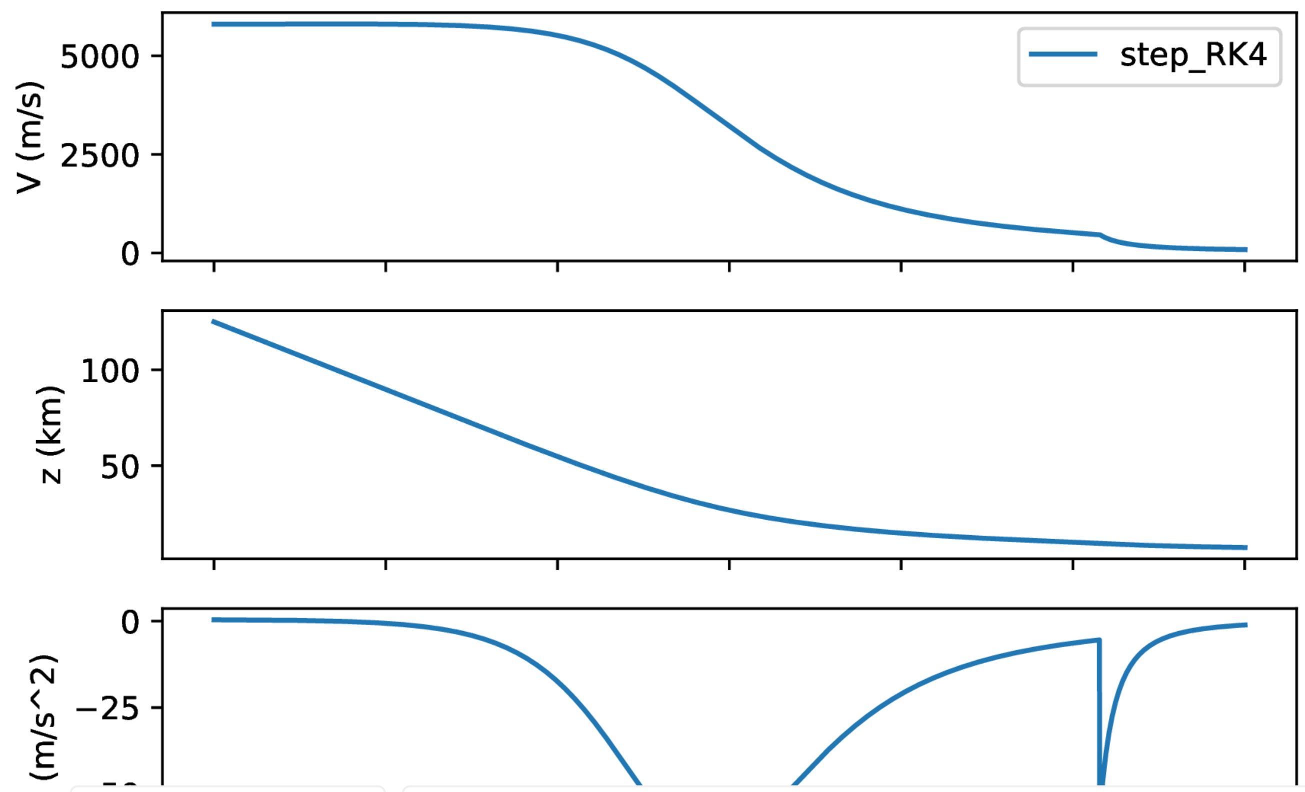
3. **Implement `lander_run_case(...)`**

3. **Implement `lander_run_case(...)`:**

   - You should convert the altitude to kilometers prior to calling `lander_Vzaplot`.
   - You will need to determine the acceleration a at all times, for plotting by `lander_Vzaplot()`. By definition, $a = \mathrm{d}V/\mathrm{d}t$, and this is already calculated by your `evalf` method. Thus you can build the acceleration profile by applying `evalf` again to the state at each timestep and appropriately storing the correct element of the float list returned from `evalf`.
   - Hint: `hail_run_case` is a good starting place!
   - Read the docstring carefully to understand the dictionary that needs to be returned. For example, if `lander_run_case` is called with `mlist=[IVPlib.step_RK4]`, then the dictionary should map `"step_RK4"` to a tuple `(axs, t_deploy, z_deploy)`.

4. Run `lander.py`, which already has a main body which will create the `lander_IVP` object and call `lander_run_case`. If your implementation is correct you should have a plot which matches Figure 4.7. And, the results in the text output should match:

```
Method: step_RK4
-------------------------------------------
Parachute deployed at t, z:  2.58e+02 s, 9.74e+00 km
Final z, V:  7.32e+00 km, 8.57e+01 m/s
```



< Previous Next >

edX

About
Affiliates
edX for Business
Open edX

Careers
News

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap
Cookie Policy
Your Privacy Choices

Connect

Blog
Contact Us
Help Center
Security
Media Kit