14.2.2 Example: Martian Lander with lander drag coefficient variability

Bookmark this page

MO2.3 MO2.13 MO2.15

Though we've already estimated the probability of $z_p < 9$ km for this situation in the previous section, let's do this case again but using the Monte Carlo method to help solidify how the method works before trying more complex situations. A Python implementation using NumPy is shown in the code below and the plots which are generated are shown in Figure 14.7.

We highlight a few key items from this implementation:

- The generation of random numbers using NumPy begins with instantiating a random number generator (refered to as an rng, for short). For our purposes, we will use NumPy's default rng which is done in the line: `rng = np.random.default_rng()`. This call can take an integer argument that “seeds" the rng, `default_rng(seed)`. A typical example would be just to use zero as the seed, i.e. `rng = np.random.default_rng(0)`. When a seed is used, the same sequence of random numbers will be generated every time your Python implementation is run. This can be useful when debugging code. When the seed is not passed, then typically rng will use information about the current state of your computer, the date and time, etc. to generate a seed, making the sequence of random numbers that will be generated different for each run.

- We have assumed that the values of C_{Dl} are equally likely in the range from 1.5 to 1.9. This is known as a uniform distribution. To generate these uniformally-distributed random numbers we use Python's default uniform generator in the line: `CDls[n] = rng.uniform(1.5, 1.9)`. This produces a single random number which is uniformally distributed between 1.5 and 1.9. `rng.uniform` can take an additional argument to determine the size of an array of random numbers that are generated. Specifically, `x = rng.uniform(low, high, size)` will return an ndarray of shape `size` with floats that are generated from a uniform distribution between `low` and `high`. `size` can be either an integer or a tuple of integers. If `size` is an integer, than a 1D array is generated with length `size`. Otherwise, the shape of the returned array will be given by `size`.

Discussions

All posts sorted by recent activity

- A histogram of the 100 z_p values from the Monte Carlo simulation is shown in Figure 14.7 along with a scatter plot of each z_p value for each instance. The histogram is generated using the `matplotlib.pyplot` `hist` method in the line: `axs[1].hist(zps,bins=zbins)`. In addition to the `zps` values, the arguments include the bins (which is set to the `zbins` array). The histogram is a plot of the number of values in `zps` which fall in each bin. For example, the $8 \leq z_p < 8.5$ bin has a count of 7 which can be easily verified by counting the number of z_p points in the scatter plot which fall in this range. For some additional information about histograms, see Section 7.1.5.

- To count the number of z_p which are below the 9 km minimum, we utilize the `NumPy` `count_nonzero` method in the line: `Nlow = np.count_nonzero(zps < 9.)`. In this statement, an array of logical values is created by `zps < 9.` which is then passed to `count_nonzero`. Then, since `True` values are equal to 1 and `False` values are equal to 0, this method produces the number of `True` values (i.e. the number of times $z_p < 9.$).

```
import matplotlib.pyplot as plt
import IVP
import numpy as np

rng = np.random.default_rng()

Nsample = 100
zps  = np.zeros(Nsample)
CDls = np.zeros(Nsample)

for n in range(Nsample):
    CDls[n] = rng.uniform(1.5, 1.9)
    lander_IVP.set_p('CD_l', CDls[n])
    zps[n] = lander_run_case(lander_IVP, dt,
IVP.step_RK4)

fig, axs = plt.subplots(2,1,sharex=True)
axs[0].plot(zps,range(len(zps)),'*')
axs[0].grid(True)
axs[0].set_ylabel('index')

Nlow = np.count_nonzero(zps < 9.)
Plow = Nlow/Nsample
print(f"Nlow = {Nlow}, Nsample = {Nsample}
Plow = {Plow:.3f}")

zbins = np.linspace(8.,11.5,8)
axs[1].hist(zps,bins=zbins)
axs[1].set_xlabel('$z_p$ (km)')
axs[1].set_ylabel('count')
axs[1].grid(True)
```

```
plt.subplots()
plt.scatter(CDls, zps)
```

< Previous Next >

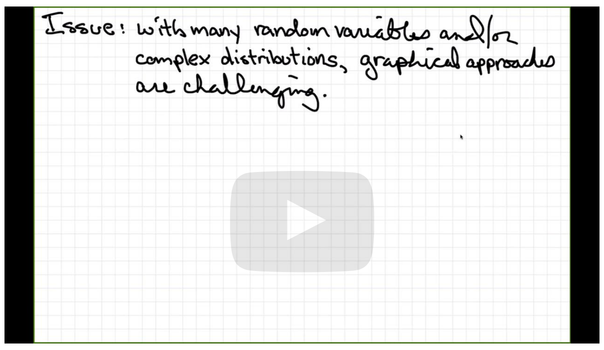
random numbers generated for C_{Dl} will be different, and thus the estimated probability will be different. For example, here are the results from 10 runs of this Monte Carlo method:

$$N_{\text{low}} = 24 \Rightarrow P_{\text{low}} = 0.24 \qquad N_{\text{low}} = 22 \Rightarrow P_{\text{low}} = 0.22$$
$$N_{\text{low}} = 25 \Rightarrow P_{\text{low}} = 0.25 \qquad N_{\text{low}} = 32 \Rightarrow P_{\text{low}} = 0.32$$
$$N_{\text{low}} = 26 \Rightarrow P_{\text{low}} = 0.26 \qquad N_{\text{low}} = 37 \Rightarrow P_{\text{low}} = 0.37$$
$$N_{\text{low}} = 23 \Rightarrow P_{\text{low}} = 0.23 \qquad N_{\text{low}} = 25 \Rightarrow P_{\text{low}} = 0.25$$
$$N_{\text{low}} = 35 \Rightarrow P_{\text{low}} = 0.35 \qquad N_{\text{low}} = 29 \Rightarrow P_{\text{low}} = 0.29$$

This of course leads to the question of how accurate is our estimate of P_{low} and how does it depend on the sample size? In the next chapter, we will investigate this behavior.

The Python scripts used in the videos below (and several others) are available here.

Video introducing the Monte Carlo method and random number generation



Start of transcript. Skip to the end.

PROFESSOR: We've demonstrated a