***Python Model Rocket Trajectory Simulator Tutorial***

**What is Python and why do we use it?**

Python is one of many computer programming languages. These languages allow us to communicate with our computers and make them do complicated tasks. From solving equations to predicting the outcome of a football game, these languages are immensely powerful and help us use the machines we’ve built to their full potential.

Python is a relatively new programming language, with its last major release, Python 3, in 2008. Applications like DropBox, Spotify, Uber, Reddit, Instagram, and many more use python significantly. It is a general-purpose language; which means it can be used to solve a variety of problems and numerous different types of programmes like data science, software development, automation, machine learning, and much more.

For our case, we will be using Python to predict the flight of our rocket. We will use a ‘.csv’ file of the thrust profile for the C5-3 motor along with constants and physics equations to generate a graph of our predicted flight path and observe the differences between boost and coast phase, as well as find the expected apogee (or maximum height) of our rocket.

**What are Jupyter Notebooks and why do we use them?**

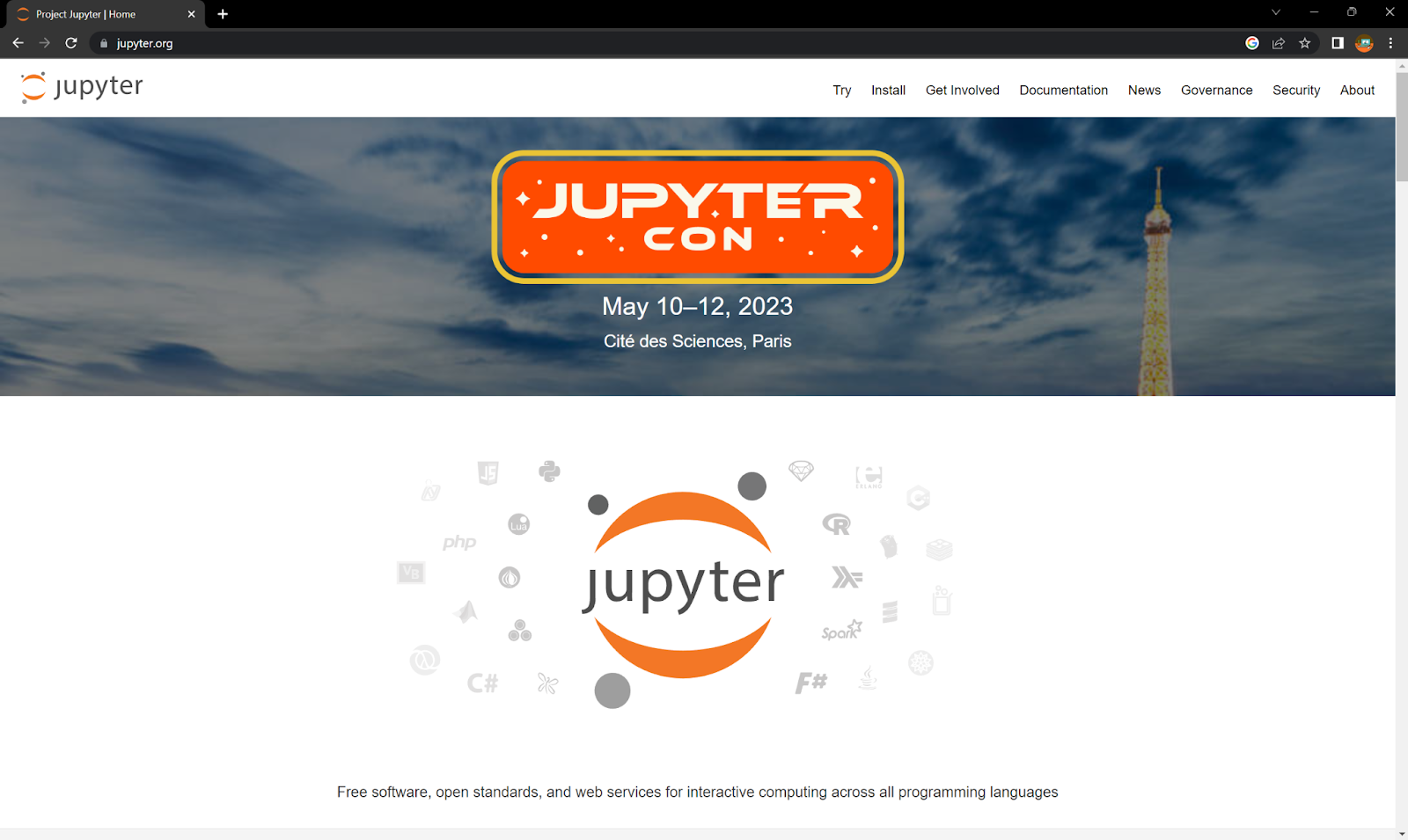
Jupyter Notebook is a free web-accessible platform where the user can input python code into cells and run the code. Jupyter Notebooks also use servers, which means the code is sent to a server, processed there, and then sent back to your computer. Because of this, Jupyter Notebook allows you to run python code on any computer without having to install anything.

To use Jupyter Notebook, go to <https://jupyter.org/> and click 'Try it in your web browser'. Then click on 'JupyterLab' and it will open an interface with a demo open. There are tabs at the top, you can click the + sign to open a new tab where you can start writing your own python code, choose 'Python (Pyodide)' under the Notebook section, and then you’re ready to go!

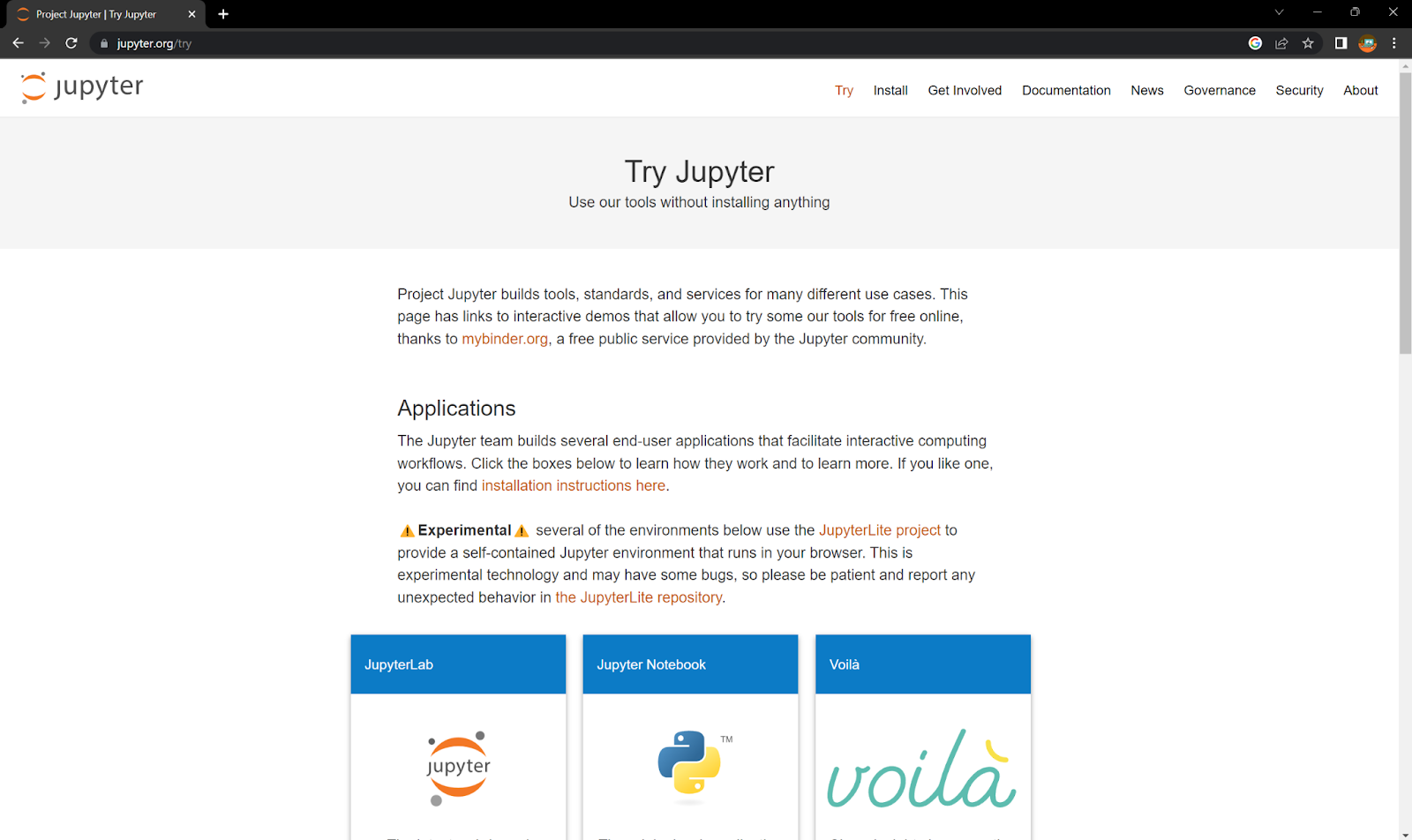
***Using Jupyter Lab to Run your Python Code***

**Step-1: *Opening JupyterLab in Your Web Browser***

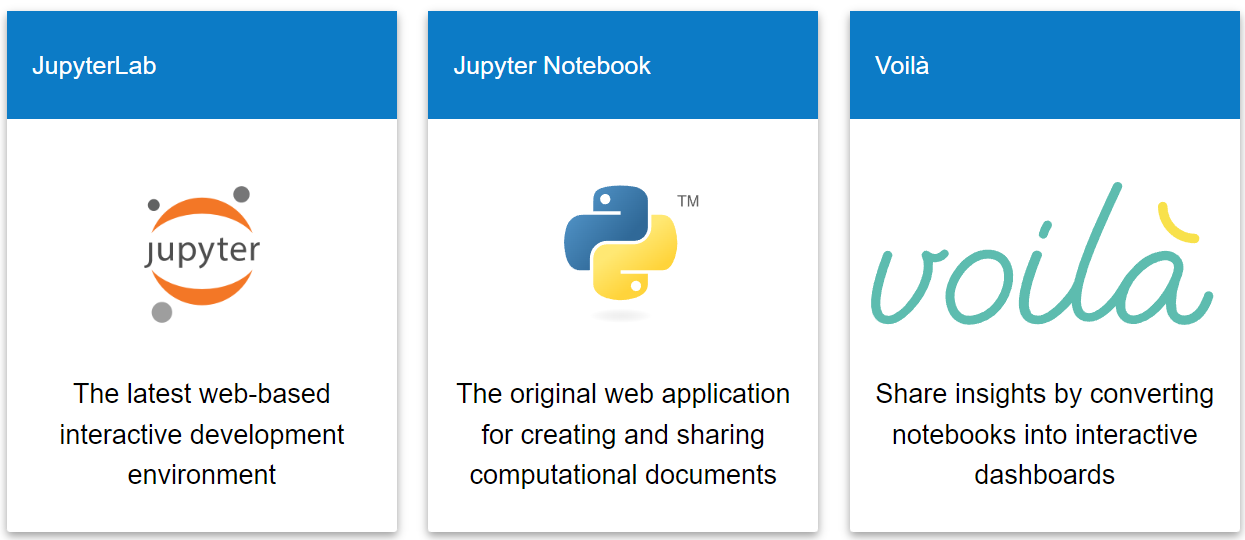
Visit the JupyterLab website. <https://jupyter.org/>



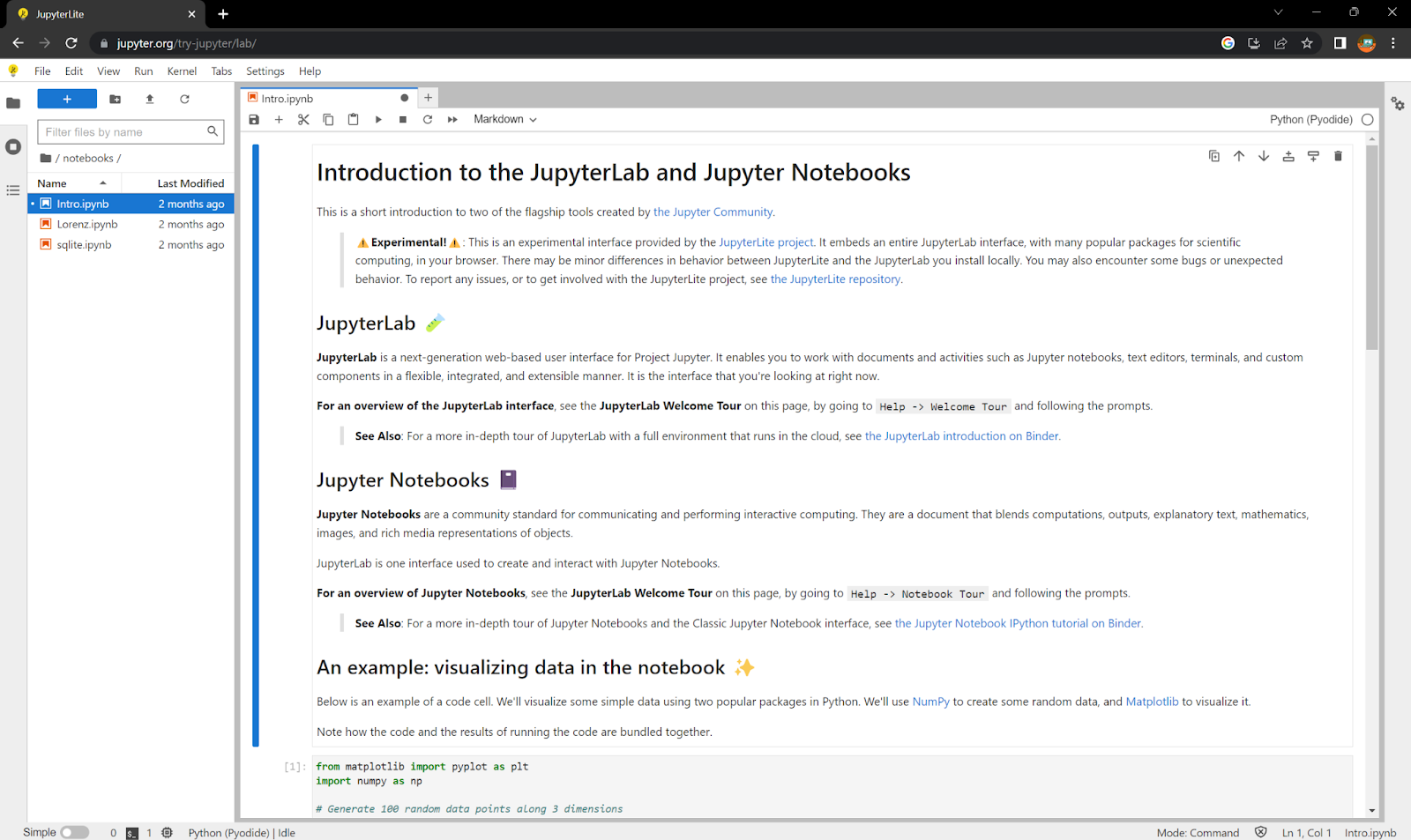
Navigate to the bar on the top-right corner of the home page and click on the 'Try' menu button.



You will see three buttons under the ‘Applications’ section of the webpage. Click on the first button from the left-side or the one that says 'JupyterLab'.

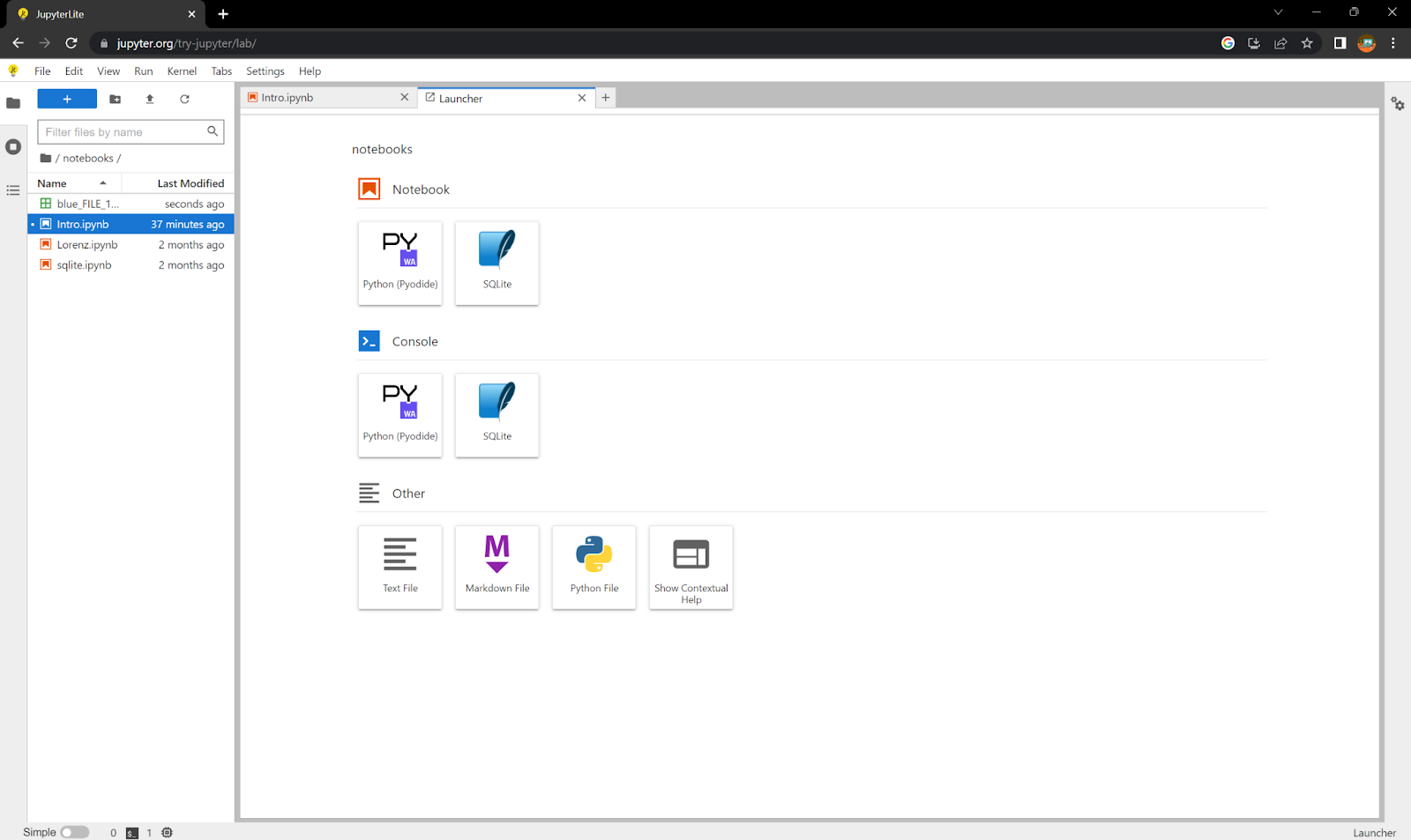


It should open an interface that looks like the image below.

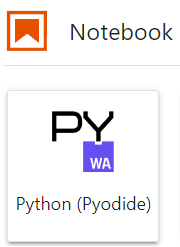


**Step-2: *Opening a new notebook***

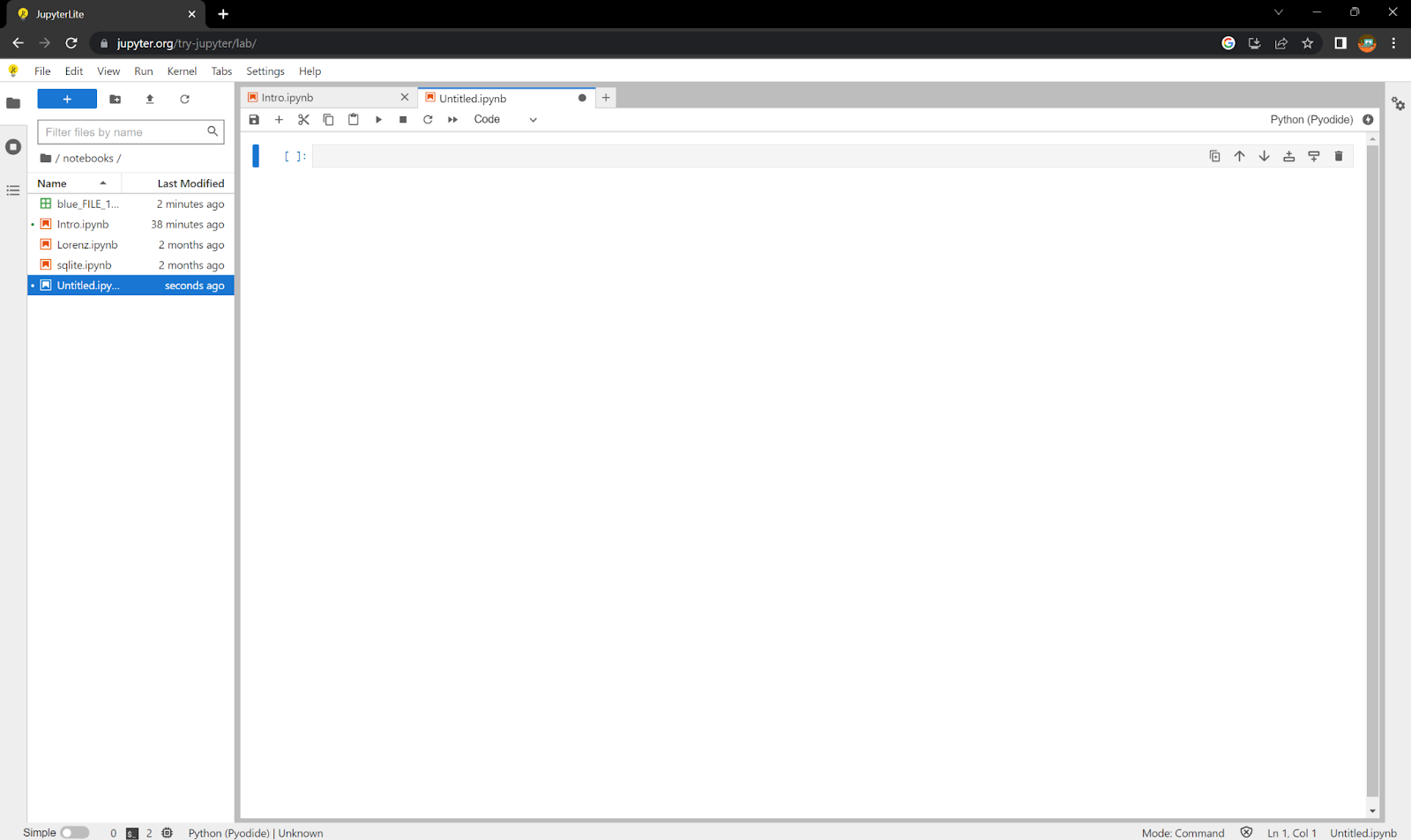
We are now going to create a new Python Notebook. Start by hitting the plus sign next to the tab that says ‘Intro.ipynb’ or using the ‘File’ dropdown from the top-left corner of the screen to create a new file. Your screen should look like the image below.



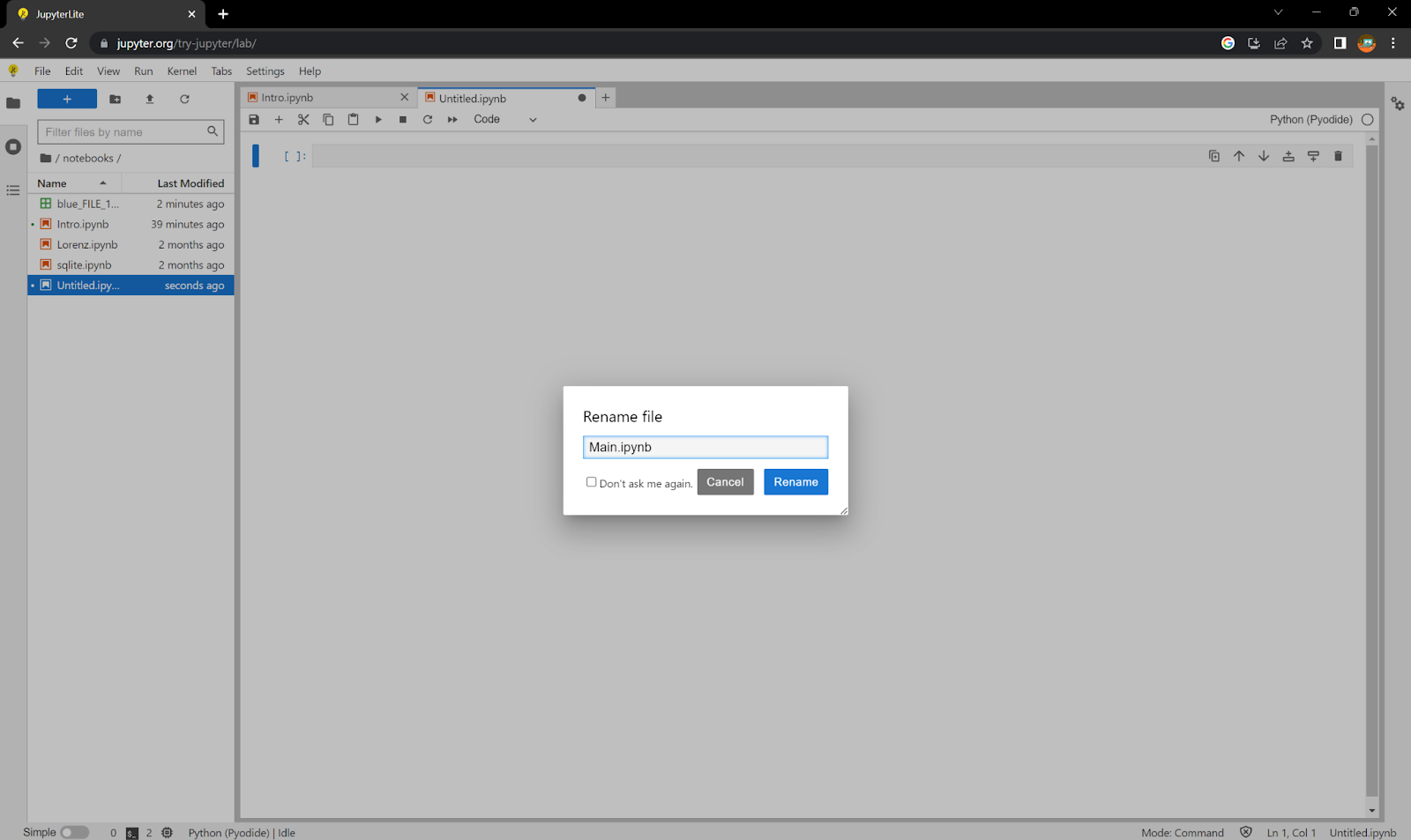
Then, click on the ‘Python (Pyodide)’ button under the ‘Notebook’ section on the screen.



A coding interface should appear and your screen should look like the image below.



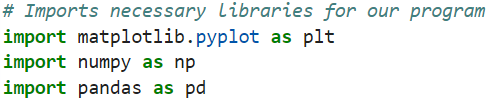
Start by saving the file using the ‘File’ dropdown in the top-left corner of your screen or pressing the buttons Ctrl+S on your keyboard. Name the file 'Main.ipynb' and hit 'Rename' button.



***Using Python to Create a Rocket Flight Simulator***

**Step-1: *Importing Necessary Libraries***

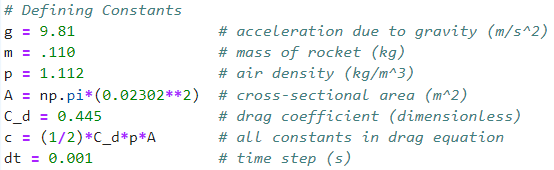
To begin coding, we must first import all the libraries we are going to use throughout the code. This gives us access to all the functions within these libraries. Import the following Libraries.



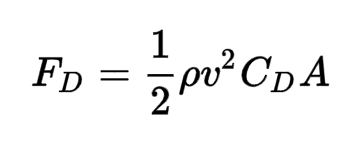
The 'matplotlib' library contains functions used to make graphs and we will use it to plot a flight trajectory and the drag over time. The 'numpy' library is used for certain numerical functions and to create arrays. Arrays are used to store multiple values of the same type in a specific order, which is very similar to a list in python, but the values in a list do not necessarily have to be the same type. This means a list in python could look like this: [“apple”, 5, 7.0, “tree”], while an array would look more like this: [4.0, 7.0, 3.0, 16.0]. The 'pandas' library is used to read csv files.

**Step-2: *Assigning variables to constants***

To create a flight trajectory simulator, we must account for all of the forces that act on the rocket. These forces include thrust, drag, and gravity. We will be finding the acceleration due to each force. To get the equations for all of these forces and accelerations, there are constants that must be included in the code. Assign the following variables to the constants that we have given, but feel free to enter in different masses for your specific payload + rocket + motor + altimeter or to test what happens when the rocket gets heavier/lighter.



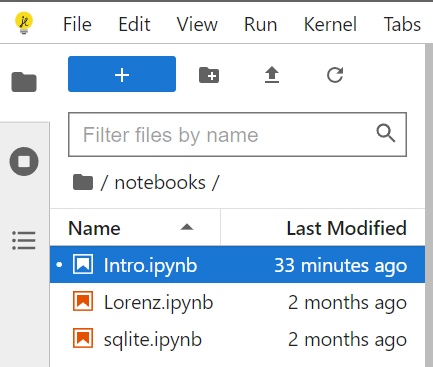
In order to account for drag in the simulator, you will need to use the drag equation given below. As written, this will result in the force due to drag, not the acceleration. Due to Newton’s 2nd Law, force = mass \* acceleration, dividing this equation by mass will result in the acceleration due to drag. We use the same law for acceleration due to thrust as well.



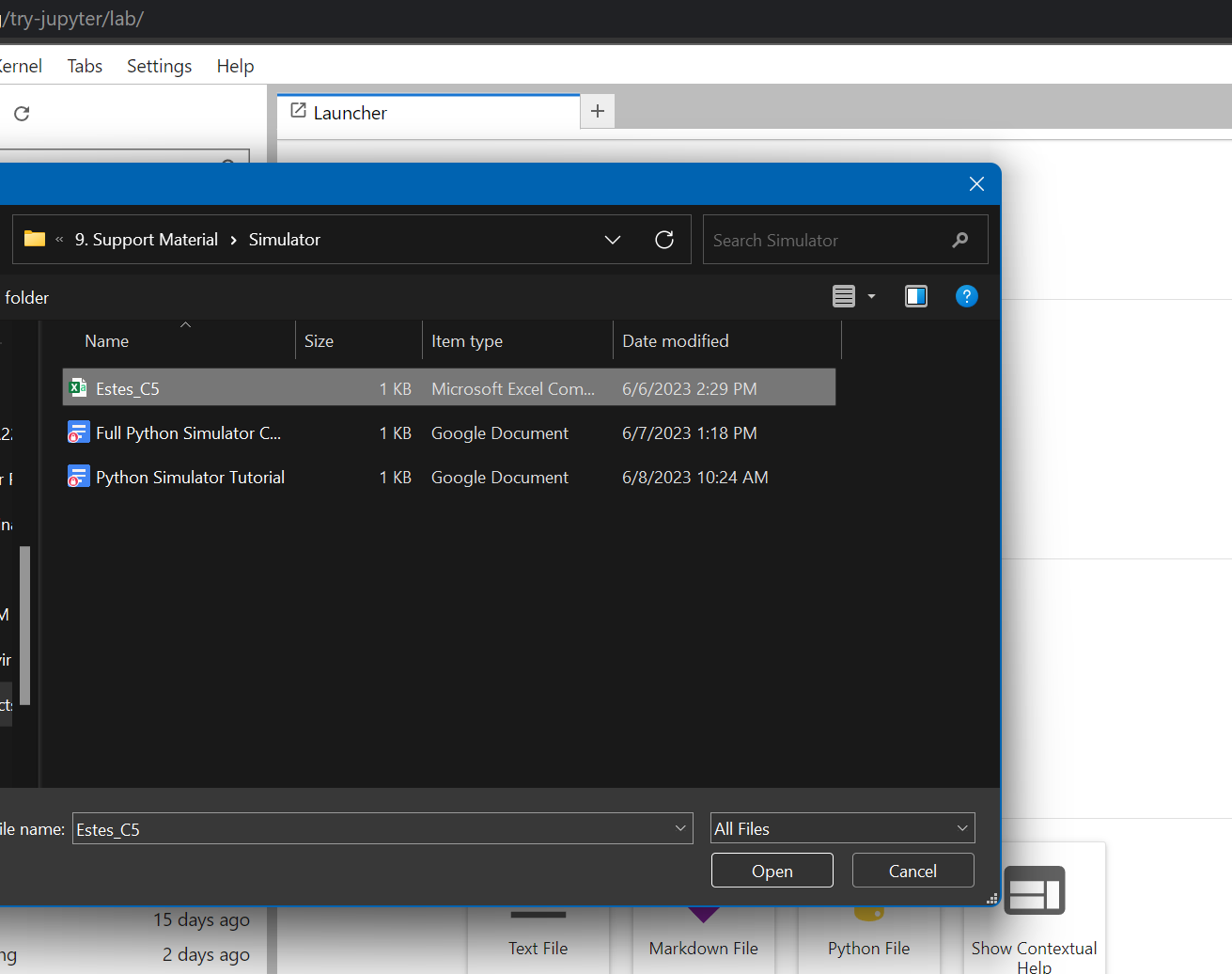
In place of 'ρ' we used 'p', which represents the atmospheric density, or the density of whatever your rocket is flying through. 'C\_d' is the drag coefficient, and 'A' is the cross-sectional area of the rocket. These are all constants for your rocket, and can be multiplied together with 0.5 to create one constant variable, which we called 'c'. Drag is then only dependent on the velocity of the rocket.

**Step-3: *Upload your thrust curve .csv file to the JupyterLab interface***

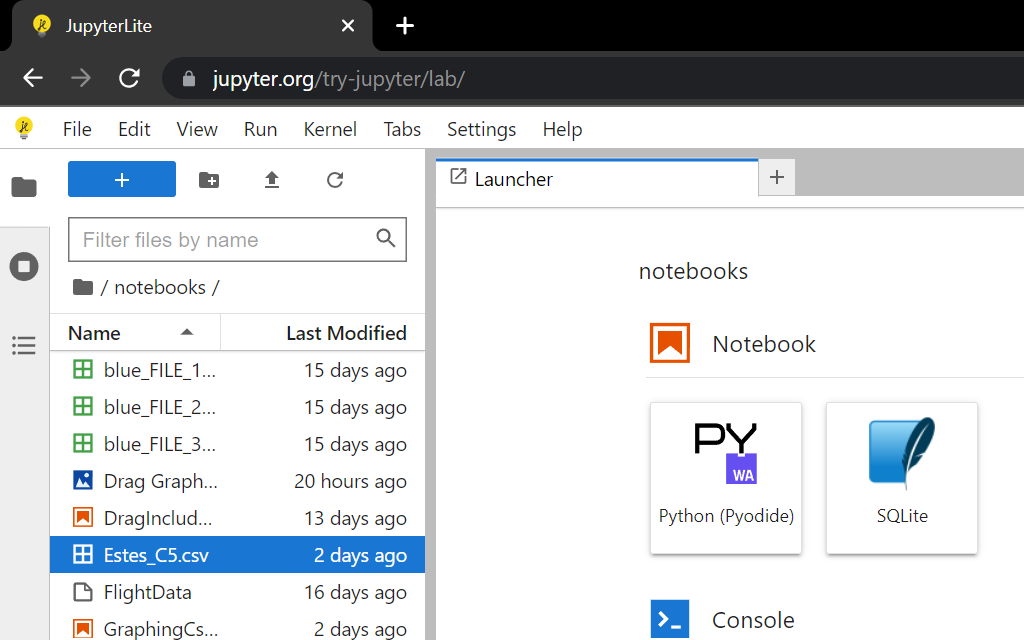
First, download your motor thrust curve .csv file from this link (<https://uofi.box.com/s/3b5q2wovicxse7vo26p8n52hq6vx6gcm>).



Then, go back to JupyterLab and navigate to the top-left corner of the screen. Hover over the upwards arrow icon that is the second button to the right of the blue plus sign. It’s the ‘Upload’ button. Click on it.



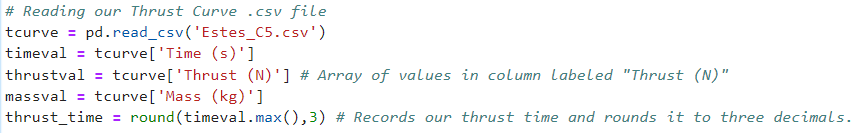
It should allow you to navigate to the thrust curve .csv file that you downloaded from the link above. Navigate to it, select it, and hit ‘Open’ to upload it to your JupyterLab interface.



Once, uploaded. It will show in the list of files that your interface has towards the top-left of your screen. Your file has now been uploaded and we can start working with Python!

**Step-4: *Reading CSV files***

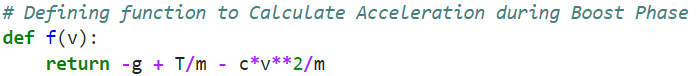
Rocket motors do not provide constant force throughout the entire time that the motor is burning and the mass of the rocket decreases as propellant is expelled. Because of this, we need to upload a csv file containing the thrust profile curve, which gives the thrust and mass at different points in time throughout the burning of the motor. This allows us to account for the varying thrust and varying mass in our calculations.



In the code above, the pandas library is used to read the thrust curve .csv file. The following three lines create arrays with all the values in each column, assigned by the header of that column. Then, the total thrust time is calculated using the greatest time value in the “Time (s)” column and is rounded to two decimal places. We round to two decimal places because our time step is 0.001 and the total thrust time must be evenly divisible by our time step.

**Step-5: *Creating a Function that Results in the Total Acceleration***

In this simulator, our goal is to find the velocity and altitude of the rocket throughout its entire flight. We only know the forces and accelerations, so we are going to have to integrate acceleration to find velocity and integrate again to find altitude. We will get to that in a later step, but we will first need to create a function for the total acceleration.



The function above accounts for the acceleration due to gravity, thrust, and drag (in that order).

**Step-6: *Creating an Array of Time Steps***

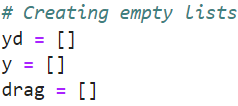
The following line of code can be used to create an array of time steps starting from 0 and ending at the total thrust time. The time step is dt, a variable we assigned earlier, which just means each value in the array will be 'dt' apart from each other, or 0.001 seconds apart. In Python, the final value is not included in the array, which is why the final value in this code is thrust time plus the time step. This ensures that the entire thrust time is included in the time array.



We’ll use the values that this array stores to create a graph that models the rocket’s trajectory with respect to time.

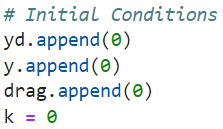
**Step-7: Creating empty lists for our Velocity, Altitude, and Drag Values**

We create empty lists to store our Velocity (yd), Altitude (y), and Drag (drag) values as the flight progresses. We’ll use the values that these lists store as points on a graph to model its trajectory with respect to time.



**Step-8: *Setting Initial Conditions***

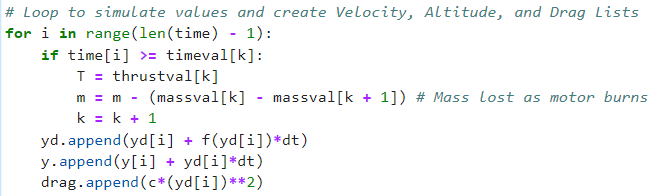
Initially, the rocket is on the ground with no velocity. This means the first value in our velocity and altitude lists are going to be zero. The ‘append’ function in python allows us to add a value onto the end of the list, and since the lists were empty, the following values are the very first values in the lists. By doing this we are setting the initial conditions of the flight.



When velocity is zero, drag must also be zero due to the drag force equation. The variable 'k' is used in the ‘for loop’ created in the next step.

**Step-9: *Creating a Loop to Calculate Flight Properties Throughout the Boost Phase***

Because acceleration is always changing, as well as the mass and thrust, we need to create a 'for loop' to find what these values are at each time step (for loops are used when you have a block of code which you want to repeat a fixed number of times).



The loop will run through all values of ‘i’, starting at ‘0’ and ending at the length of our time list minus 1. We subtract 1 from the total length of the time list because in our loop, each value is found using the values at the point directly before it. This means that if our loop ends at the point right before the last time value, the final values for velocity, altitude, and drag will still be calculated.

Inside, we have an ‘if’ statement that cross-checks our value of time during the flight with the amount of thrust our motor should be giving out based on our thrust curve. This is because our motor does not give out a constant value of thrust throughout the flight. Hence, it accounts for the change in thrust from one time step to the next. Similarly, as fuel is being ejected from the motor, the mass of our rocket will gradually decrease until our motor is completely burnt out. Therefore, as the loop runs, our mass value gradually decreases by small increments which we calculate using our thrust curve. Then ‘k’ is increased by one to progress the function.

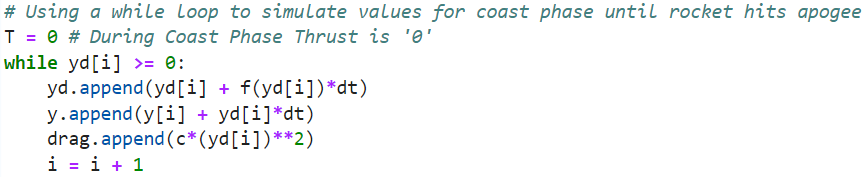
(Step-9 continued on the next page)

(Step-9: continued)

Using these changing values of thrust, mass, and the velocity during the flight, we then calculate the values for velocity, altitude, and drag for the subsequent time increment ‘dt’. The ‘.append()’ function updates our list with the most recently calculated values. This loop keeps going until our motor burns out. First, velocity is calculated at each point by multiplying the acceleration by the time step and adding the velocity at the previous point. The acceleration used for this will be using the ‘f(v)’ function we created earlier and the velocity input will be the velocity value at the point directly before it. This is called Euler’s Method, which is a simple form of integration where this process is repeated over and over again for incredibly small time steps, calculating the values at a certain point using the values at the previous point. We use this method again to calculate the altitude, but now we do not have to use the ‘f(v)’ function, we can just use the velocity at the point directly before it. For drag, we use the drag equation and plug in the velocity at the point directly before it.

**Step-10: *Creating a Loop to Calculate Flight Properties Throughout the Coast Phase***

Now, we have completed the boost phase, but we still need to account for the coast phase in order to find the apogee (or maximum height) of the rocket.



After the boost phase, we consider our thrust value to be zero as our motor has completely burnt out and can’t give out more force. Above, there is a ‘while loop’, which in python is used to execute a block of statements repeatedly until a given condition is satisfied, and when the condition becomes false, the loop ends and the code continues on. In this code, the ‘while loop’ uses the last value of ‘i’ in the previous ‘for loop’ and continues until our craft reaches apogee, or our velocity (yd[i]) reaches zero. This loop adds all values for Velocity, Altitude, and Drag for the coast phase to our lists as it runs.

**Step-11: *Redefining Time Array to Include both the Boost and Coast Phases***

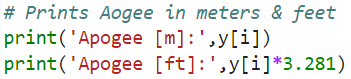
Originally, our time array was defined using the thrust time, but this only accounts for the boost phase. Now, we need to add all of the time from the coast phase.



We already know that the velocity and altitude lists include both the coast and boost phases, so we can set the time array equal to the length of either of those lists. We use the numpy library to create an array starting at zero and ending at the length of the altitude array (velocity array would also work), but this by itself will create a list of integers. Because our time step is 0.001 and we used this throughout our other calculations, we will need to divide all the values in this time array by 1000.

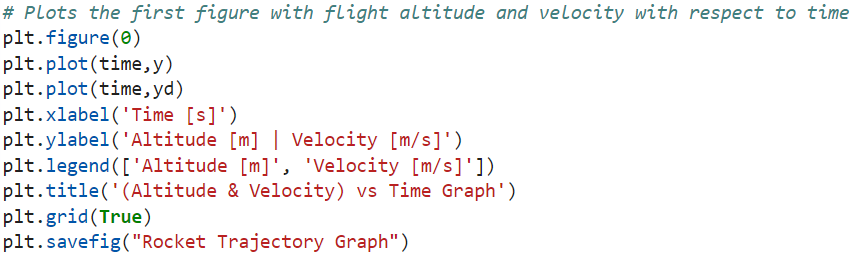
**Step-12: *Calculating and Outputting the Apogee***

Although in this simulator we are working to get a full profile of the rocket’s flight path, we also want to know the Apogee of the rocket, or its maximum height. We will find this in both feet and meters and have the code output the results.

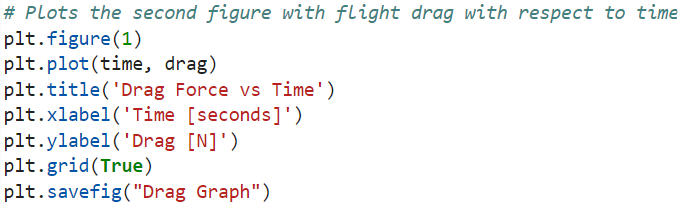


All of our calculations so far have been done in meters, so we will start there. At the end of our previous ‘while loop’, 'i' was equal to the index of the final term in all of our lists and arrays. Index simply means the placement of a value within a list or array, the only tricky part is that a list’s index does not start at 1, it starts at 0. That means if we look at the 'i' index in the list of altitudes, we will get the Apogee, as we stopped the loop as soon as the Apogee was reached. All we need to do is print this value with the code above. We can then use the conversion factor of 3.281 to find the same Apogee in feet.

**Step-13: Using the Matplotlib Library to Plot Our Lists With Respect to Time**



This graphs our first figure that has two plots: Altitude vs Time and Velocity vs Time. This helps us understand the relationship between those two during flight.



This graphs our second figure that shows the force of drag throughout our flight with respect to time.

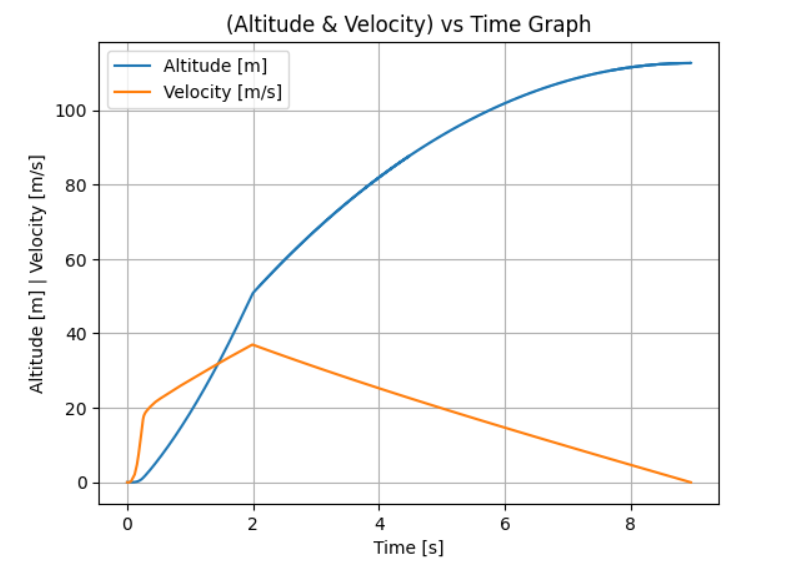


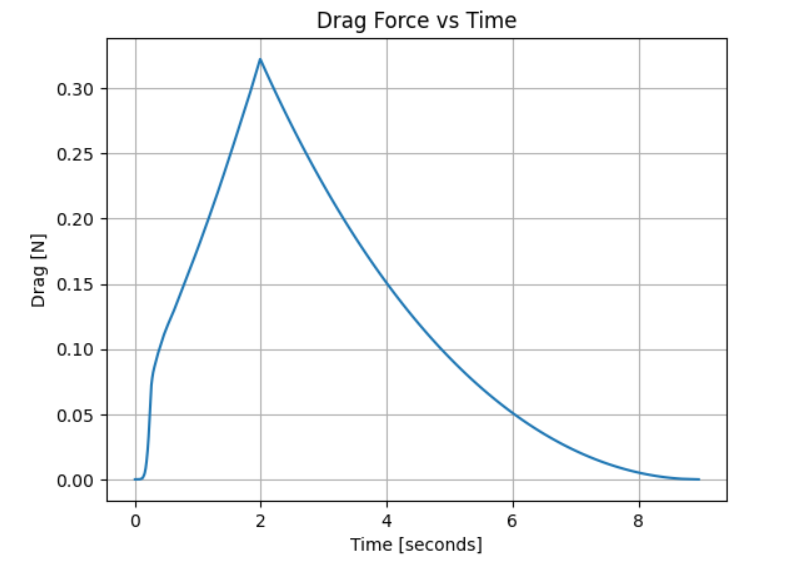
The ‘.show()’ function will tell the program to display figures plotted using the ‘.pyplot’ functions of the Matplotlib library that we imported at the beginning of this tutorial.

**Step-14: *Output!***

All done! Run your code either by clicking run in the upper left corner and then running all cells or pressing shift+enter on your keyboard. The results should look something like this depending on the mass entered.







***Full Code***

***(if copy and pasted, indentations must be entered by hand***

***into JupyterLab or else the code will not run)***

# Imports necessary libraries for our program

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Defining Constants

g = 9.81

m = 0.136

p = 1.112

A = np.pi\*(0.02302\*\*2)

C\_d = 0.445

c = (1/2)\*C\_d\*p\*A

dt = 0.001 # Our time step

# Reading our Thrust Curve .csv file

tcurve = pd.read\_csv('Estes\_C5.csv')

timeval = tcurve['Time (s)']

thrustval = tcurve['Thrust (N)'] # Creates an array from the values in column labelled 'Thrust (N)' in our .csv file

massval = tcurve['Mass (kg)']

thrust\_time = round(timeval.max(),3) # Records our thrust time and rounds it to two decimals.

# Defining function to Calculate Acceleration during Boost Phase

def f(v):

return -g + T/m - c\*v\*\*2/m

# Creating an array of time steps from zero to the total thrust time

time = np.arange(0,thrust\_time+dt,dt)

# create empty lists

yd = []

y = []

drag = []

# set initial conditions

yd.append(0)

y.append(0)

drag.append(0)

k = 0

# Loop to simulate values and create Velocity, Altitude, and Drag Lists

for i in range(len(time) - 1):

if time[i] >= timeval[k]:

T = (thrustval[k + 1] + thrustval[k])/2 # Averages Thrust for 2 points and considering it for all points in between

m = m - (massval[k] - massval[k + 1]) # Accounts for reduction in mass as motor burns

k = k + 1

yd.append(yd[i] + f(yd[i])\*dt)

y.append(y[i] + yd[i]\*dt)

drag.append(c\*(yd[i])\*\*2)

# Using a while loop to simulate values for coast phase until rocket hits apogee

T = 0 # During Coast Phase Thrust is 0

while yd[i] >= 0:

yd.append(yd[i] + f(yd[i])\*dt)

y.append(y[i] + yd[i]\*dt)

drag.append(c\*(yd[i])\*\*2)

i = i + 1

# Redefining time to include both boost and coast phase

time = np.arange(0,len(y))/1000

# Prints Apogee in meters & feet

print('Apogee [m]:',y[i])

print('Apogee [ft]:',y[i]\*3.281)

# Plots the first figure with flight altitude and velocity with respect to time

plt.figure(0)

plt.plot(time,y)

plt.plot(time,yd)

plt.xlabel('Time [s]')

plt.ylabel('Altitude [m] | Velocity [m/s]')

plt.legend(['Altitude [m]', 'Velocity [m/s]'])

plt.title('(Altitude & Velocity) vs Time Graph')

plt.grid(True)

plt.savefig("Rocket Trajectory Graph")

# Plots the second figure with flight drag with respect to time

plt.figure(1)

plt.plot(time, drag)

plt.title('Drag Force vs Time')

plt.xlabel('Time [seconds]')

plt.ylabel('Drag [N]')

plt.grid(True)

plt.savefig("Drag Graph")

# Shows all plots that use the matplotlib functions

plt.show()