Rocket Trajectory Visualisation Tool – Future Development

Report prepared by Heather Leyssenaar

Recommendations are based on work completed by project developers:
Luke Gerschwitz,
Alyssa Yeo,
David Galbory, and
Heather Leyssenar.

Table of Contents

Introduction

The Rocket Trajectory Visualisation project, developed for Swordfish Computing, has had the following major features added:

- Additional datasets that change show changes in alternative input variables (as opposed to only mass over time) have been added.
- The user can use the new datasets to generate new graphs at runtime, by accessing a user menu that is linked to the left controller
- Graphs can be deleted from the scene at runtime
- Multiple graphs can be spawned around the user, and displays are updated if a graph is deleted
- Graphs can be regenerated when axes have changed.
- New output variables are automatically read in to the project and can be displayed to a data panel
- Added support for different controllers (HTC Vive)
- The user can choose the axes of the graph via a menu panel
- The user can choose to generate a 2D or 3D scatter-graph
- The user can choose to generate a bar graph or a scatter-graph
- The user can highlight a trajectory with a slider and view updated information about that trajectory in the data panel and on the graph config menu.

- The information displayed on the data panel can be customised by using the data control panel and checking or un-checking the related variable checkbox. The data panel also has a slider control to view larger amounts of data.

- The user can configure existing graphs by using a configuration panel
- Most menu panels can be minimised to reduce screen clutter, and then brought back to original size
- Some quality of life features such as background colour, sound effects, movable data windows and loading time visual cues.

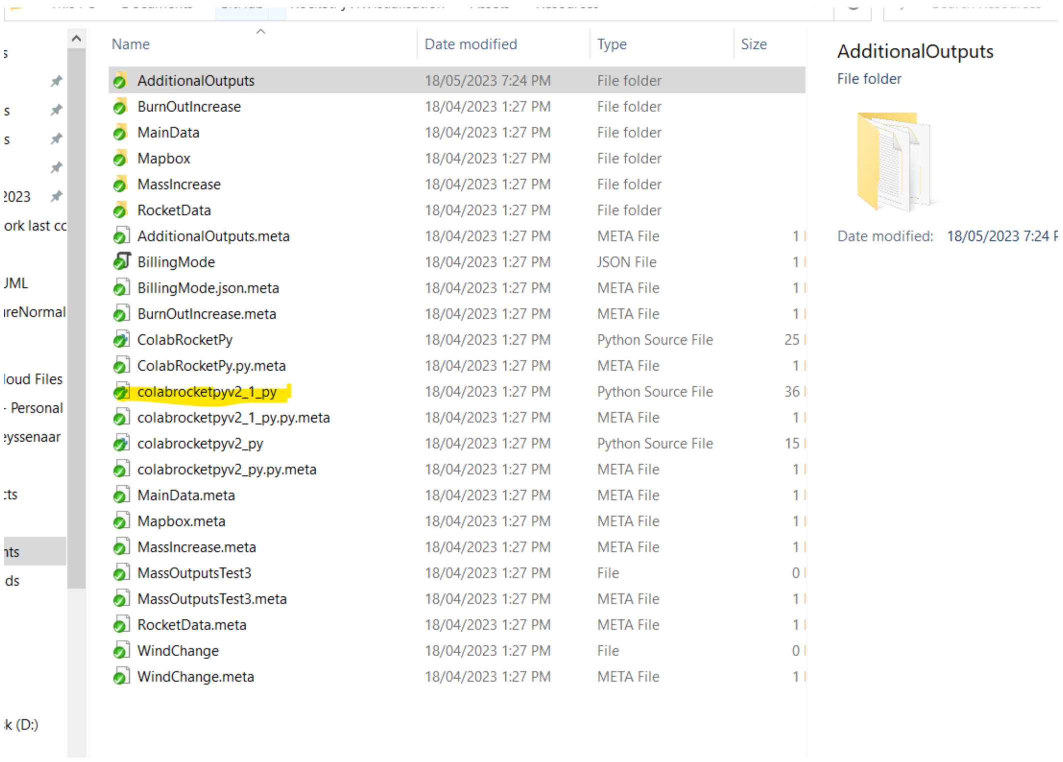
A potential future step was identified, which was adding another layer of abstraction to the graphs.

Applying Ladder of Abstraction concepts to the project can be most easily achieved by adding datasets with additional variable changes for the user to select from the graph menu.

Rather than abstracting the change of a single variable over time, the source data can be generated with two (or more) input variables being changed for the rocket trajectory simulations.

Next Steps – Creating the dataset(s)

In the Assets/Resources folder of the project, the colabRocketpyv2_1 file will give the quickest starting point. This file should be run in Google Colab to prevent dependency errors. Mount the file according to instructions that can be found in the manual. (The manual can be found on the GitHub repository for this project).



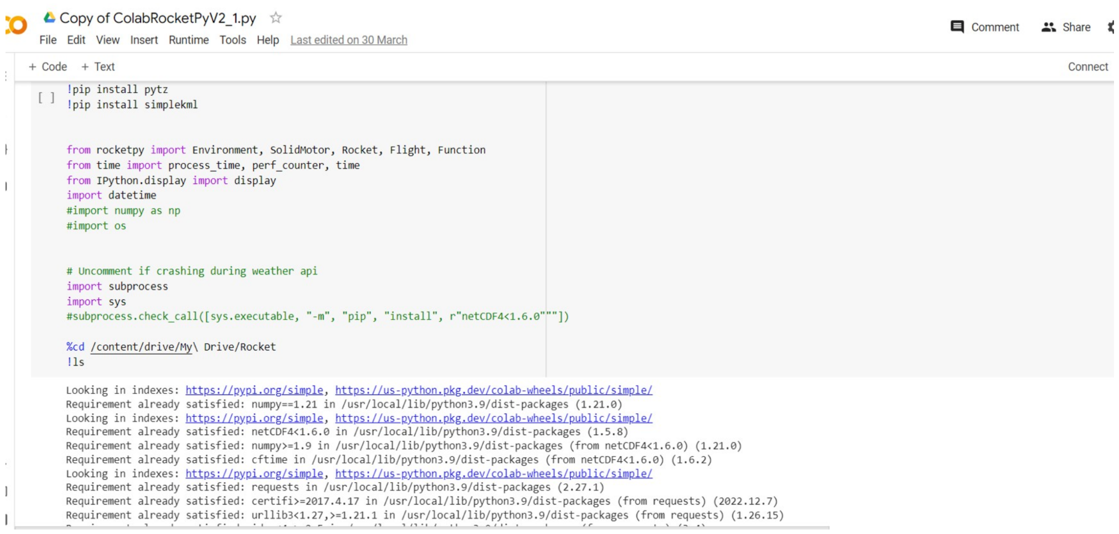
Name	Date modified	Type	Size
AdditionalOutputs	18/05/2023 7:24 PM	File folder	
BurnOutIncrease	18/04/2023 1:27 PM	File folder	
MainData	18/04/2023 1:27 PM	File folder	
Mapbox	18/04/2023 1:27 PM	File folder	
MassIncrease	18/04/2023 1:27 PM	File folder	
RocketData	18/04/2023 1:27 PM	File folder	
AdditionalOutputs.meta	18/04/2023 1:27 PM	META File	1
BillingMode	18/04/2023 1:27 PM	JSON File	1
BillingMode.json.meta	18/04/2023 1:27 PM	META File	1
BurnOutIncrease.meta	18/04/2023 1:27 PM	META File	1
ColabRocketPy	18/04/2023 1:27 PM	Python Source File	25
ColabRocketPy.py.meta	18/04/2023 1:27 PM	META File	1
colabrocketpyv2_1_py	18/04/2023 1:27 PM	Python Source File	36
colabrocketpyv2_1_py.py.meta	18/04/2023 1:27 PM	META File	1
colabrocketpyv2_py	18/04/2023 1:27 PM	Python Source File	15
colabrocketpyv2_py.py.meta	18/04/2023 1:27 PM	META File	1
MainData.meta	18/04/2023 1:27 PM	META File	1
Mapbox.meta	18/04/2023 1:27 PM	META File	1
MassIncrease.meta	18/04/2023 1:27 PM	META File	1
MassOutputsTest3	18/04/2023 1:27 PM	File	0
MassOutputsTest3.meta	18/04/2023 1:27 PM	META File	1
RocketData.meta	18/04/2023 1:27 PM	META File	1
WindChange	18/04/2023 1:27 PM	File	0
WindChange.meta	18/04/2023 1:27 PM	META File	1

AdditionalOutputs
File folder
Date modified: 18/05/2023 7:24

Figure 1: Location of Google Colab file

If you wish, you can copy or rename the file to preserve the older version.

Follow the steps laid out in the file, correcting any errors that may be raised by missing folders (instructions also found in the manual).


Copy of ColabRocketPyV2_1.py
File Edit View Insert Runtime Tools Help Last edited on 30 March
Comment
Share
+ Code + Text
Connect

```
!pip install pytz
!pip install simplekml


from rocketpy import Environment, SolidMotor, Rocket, Flight, Function
from time import process_time, perf_counter, time
from IPython.display import display
import datetime
#import numpy as np
#import os


# Uncomment if crashing during weather api
import subprocess
import sys
#subprocess.check_call([sys.executable, "-m", "pip", "install", r"netCDF4<1.6.0"""])

%cd /content/drive/My\ Drive/Rocket
!ls
```

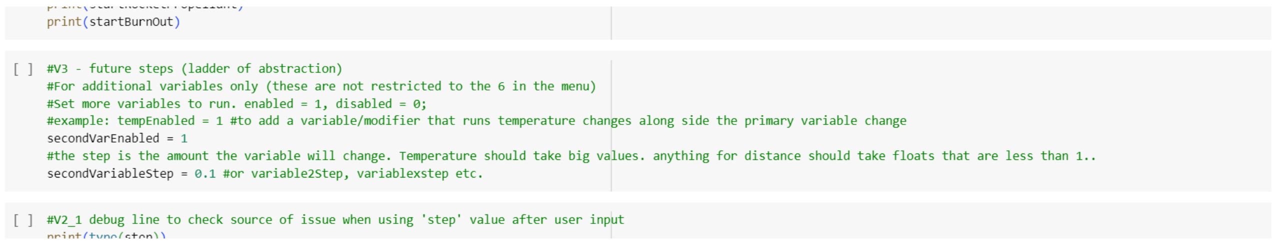
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy==1.21 in /usr/local/lib/python3.9/dist-packages (1.21.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: netCDF4<1.6.0 in /usr/local/lib/python3.9/dist-packages (1.5.8)
Requirement already satisfied: numpy>=1.9 in /usr/local/lib/python3.9/dist-packages (from netCDF4<1.6.0) (1.21.0)
Requirement already satisfied: cftime in /usr/local/lib/python3.9/dist-packages (from netCDF4<1.6.0) (1.6.2)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (2.27.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests) (1.26.15)
```

Figure 2: Colab file first cell

Each code block is called a ‘cell’. Run any cell that isn’t marked as optional until you get to choosing variable types. This can be done by clicking in the cell, then holding “Ctrl + Enter”.

Note on naming the folder: The folder name should indicate what two variables are being chosen, but due to space restrictions on the graph menu, the name should not be too long.
Something like “Mass_Temp” would be ideal for increasing mass and temperature together, for example.

Run the cells that choose the initial variable, the starting value, and the step value (how much the value increases or decreases). Below these code cells, add a cell to the file for extra variables. An example is shown below. You will be adjusting these values manually, so be sure to choose appropriate step sizes (the amount the variable will change over the simulations).
While not strictly necessary, we are adding secondVarEnabled so that we can easily switch back to single variable changes without reverting all code changes. You may also wish to change the starting variable value. If so, add it here too.
If you will be re-running the file with different variables to use as the second variable, it is a good idea to name secondVarEnabled as something more appropriate to the actual name, and set it to zero, so that you do not have to remove the extra code. This also applies to the step variable and the starting variable (if used).



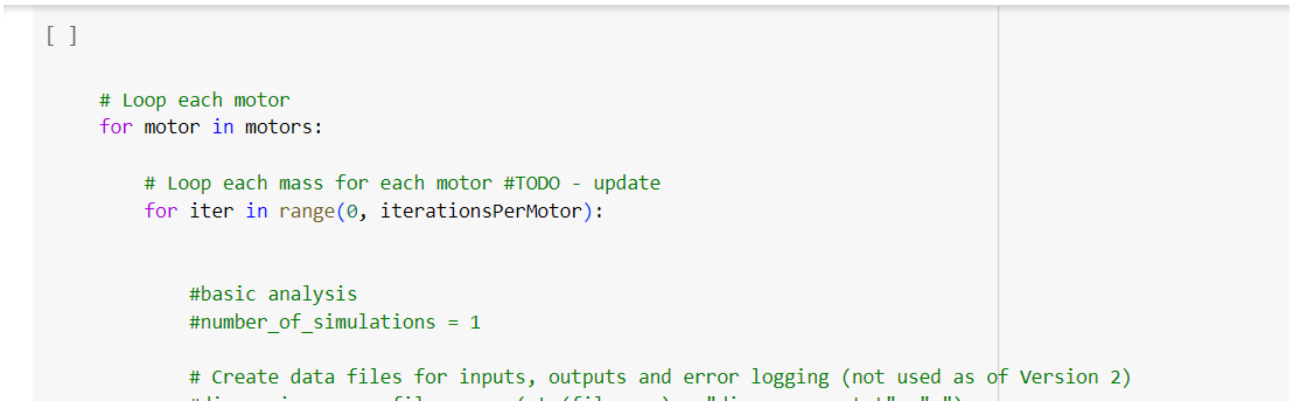
```
print(startBurnOut)
```

```
#V3 - future steps (ladder of abstraction)
#For additional variables only (these are not restricted to the 6 in the menu)
#Set more variables to run. enabled = 1, disabled = 0;
#example: tempEnabled = 1 #to add a variable/modifier that runs temperature changes along side the primary variable change
secondVarEnabled = 1
#the step is the amount the variable will change. Temperature should take big values. anything for distance should take floats that are less than 1..
secondVariableStep = 0.1 #or variable2Step, variablexstep etc.
```

```
#V2_1 debug line to check source of issue when using 'step' value after user input
```

Figure 3: Example code to be added to the file and then run

Then look for this section in the main simulation code (marked by #V1).



```
[ ]

    # Loop each motor
    for motor in motors:

        # Loop each mass for each motor #TODO - update
        for iter in range(0, iterationsPerMotor):


            #basic analysis
            #number_of_simulations = 1

            # Create data files for inputs, outputs and error logging (not used as of Version 2)
```

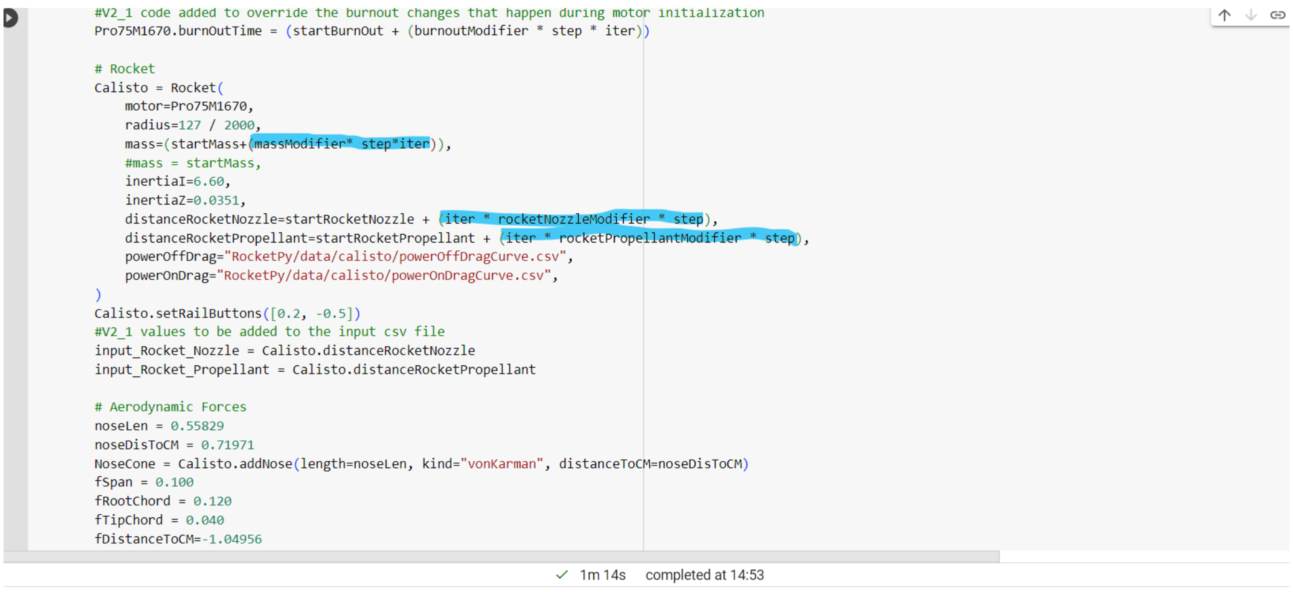
Figure 4: Main simulation loop code

Go down to where the variables are defined.

Choose the variable to change and modify the line so that it will be increased

You can use any variable that can be found in the all_inputs.csv file (There should already be one generated for Default_Inputs in Assets/Resources/AdditionalOutputs if you need examples).

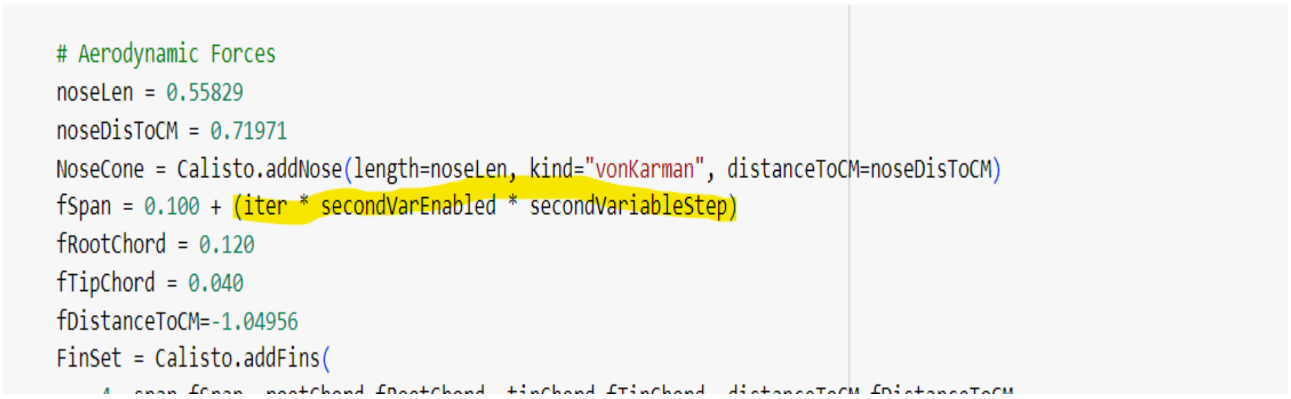
You're going to add something like the blue highlighted section to your chosen variable.



```
#V2_1 code added to override the burnout changes that happen during motor initialization
Pro75M1670.burnOutTime = (startBurnOut + (burnoutModifier * step * iter))

# Rocket
Calisto = Rocket(
    motor=Pro75M1670,
    radius=127 / 2000,
    mass=(startMass+(massModifier* step*iter)),
    #mass = startMass,
    inertiaI=6.60,
    inertiaZ=0.0351,
    distanceRocketNozzle=startRocketNozzle + (iter * rocketNozzleModifier * step),
    distanceRocketPropellant=startRocketPropellant + (iter * rocketPropellantModifier * step),
    powerOffDrag="RocketPy/data/calisto/powerOffDragCurve.csv",
    powerOnDrag="RocketPy/data/calisto/powerOnDragCurve.csv",
)
Calisto.setRailButtons([0.2, -0.5])
#V2_1 values to be added to the input csv file
input_Rocket_Nozzle = Calisto.distanceRocketNozzle
input_Rocket_Propellant = Calisto.distanceRocketPropellant

# Aerodynamic Forces
noseLen = 0.55829
noseDisToCM = 0.71971
NoseCone = Calisto.addNose(length=noseLen, kind="vonKarman", distanceToCM=noseDisToCM)
fSpan = 0.100
fRootChord = 0.120
fTipChord = 0.040
fDistanceToCM=-1.04956
```

Figure 5: Examples of existing primary variable that can be changed via the menu

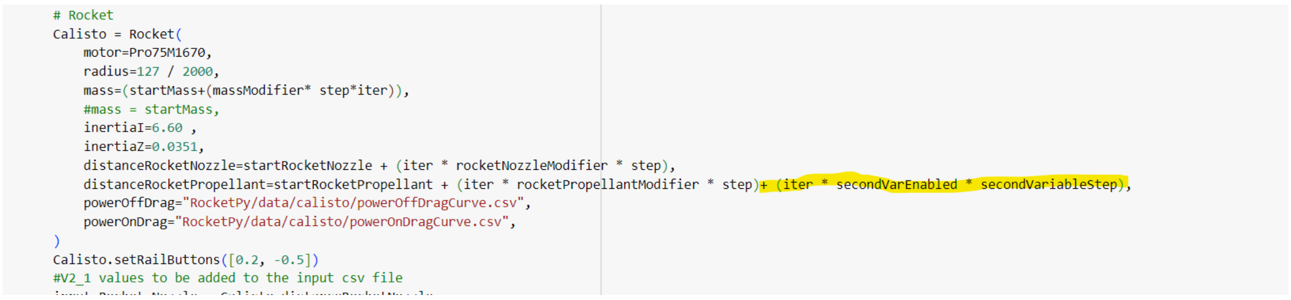
The adjustment will be to add (iter * secondVarEnabled * secondVariableStep) to the end to the line. The below is an example of choosing something already shown in all_inputs.csv, but was not yet used as a changing variable.



```
# Aerodynamic Forces
noseLen = 0.55829
noseDisToCM = 0.71971
NoseCone = Calisto.addNose(length=noseLen, kind="vonKarman", distanceToCM=noseDisToCM)
fSpan = 0.100 + (iter * secondVarEnabled * secondVariableStep)
fRootChord = 0.120
fTipChord = 0.040
fDistanceToCM=-1.04956
FinSet = Calisto.addFins(
```

Figure 6: Second variable (fSpan) that will be changed over the simulations

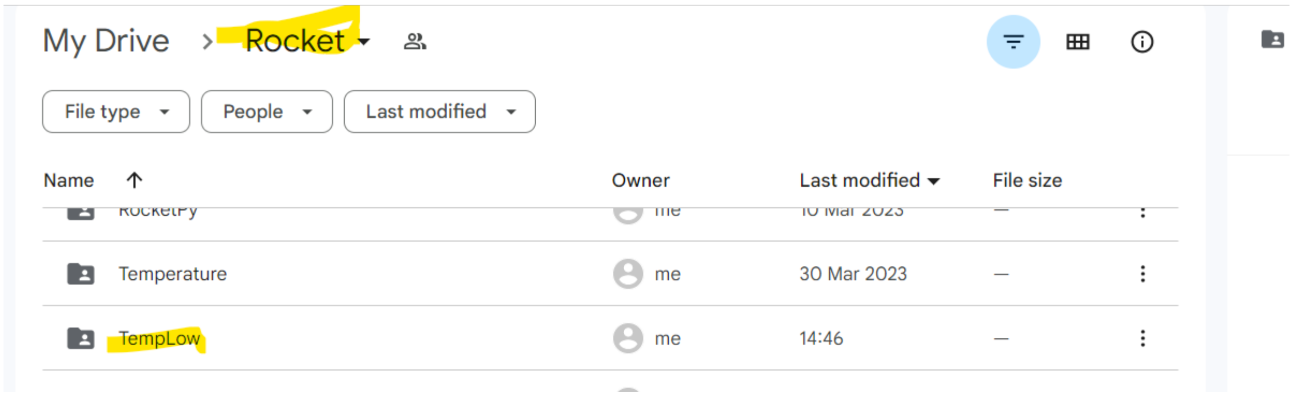
Alternatively, if you wish to use two of the originally changeable variables, run the cell with the menu to set up the primary variable, and add the following to the secondary variable:



```
# Rocket
Calisto = Rocket(
    motor=Pro75M1670,
    radius=127 / 2000,
    mass=(startMass+(massModifier* step*iter)),
    #mass = startMass,
    inertiaI=6.60 ,
    inertiaZ=0.0351,
    distanceRocketNozzle=startRocketNozzle + (iter * rocketNozzleModifier * step),
    distanceRocketPropellant=startRocketPropellant + (iter * rocketPropellantModifier * step)+ (iter * secondVarEnabled * secondVariableStep),
    powerOffDrag="RocketPy/data/calisto/powerOffDragCurve.csv",
    powerOnDrag="RocketPy/data/calisto/powerOnDragCurve.csv",
)
Calisto.setRailButtons([0.2, -0.5])
#V2_1 values to be added to the input csv file
```

Figure 7: Using a variable that was already possible to use as primary input as a secondary input

Run the cell. If correctly simulated (note that this will take a few minutes), you will find generated csv files in your Google Drive.



Name	Owner	Last modified	File size
RocketPy	me	10 Mar 2023	—
Temperature	me	30 Mar 2023	—
TempLow	me	14:46	—

Figure 8: Where to find the generated files in Google Drive

This folder can be downloaded and added directly to the AdditionalOutputs folder. (Be sure that the files are not compressed).

Configuring Unity to display the graph based on new data

The Unity project already has code that checks for data in the resources folder. The graph can be generated and displayed by launching the program, then using the folder name that will show up in the Input Conditions drop-down menu.

But to allow the graph config menu to track the variable changes, you need to go to the variable focus mapping text file found in the AdditionalOutputs folder.

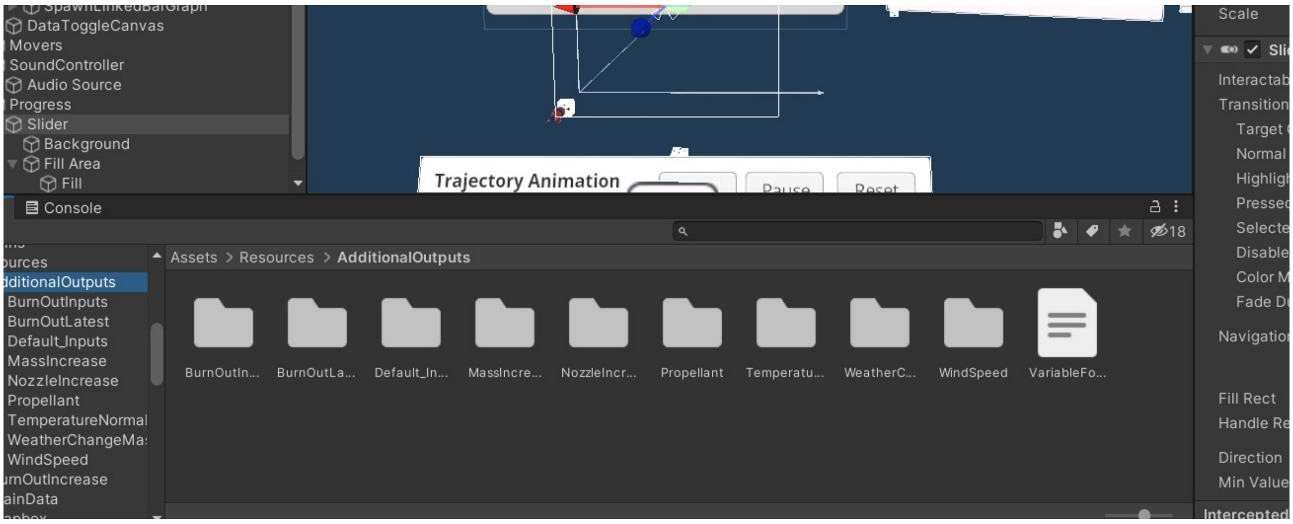


Figure 9: Location of VariableFocusMapping file

Copy your folder name exactly and add it to the file.

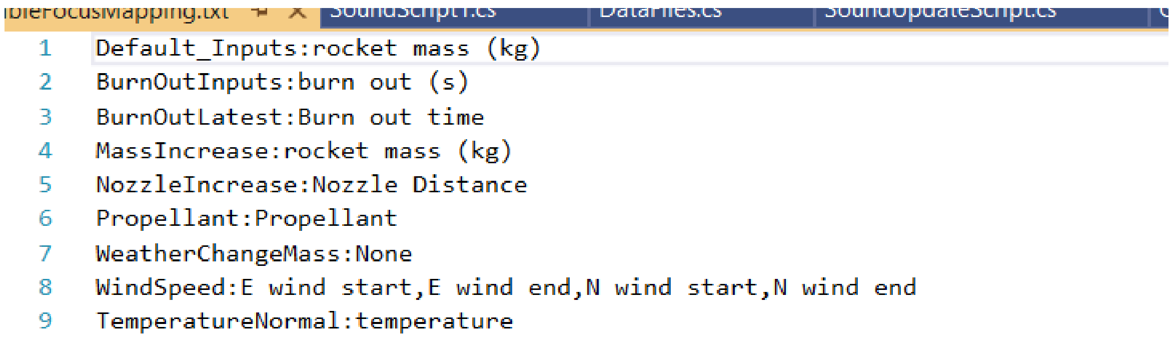

```
Default_Inputs:rocket mass (kg)
BurnOutInputs:burn out (s)
BurnOutLatest:Burn out time
MassIncrease:rocket mass (kg)
NozzleIncrease:Nozzle Distance
Propellant:Propellant
WeatherChangeMass:None
WindSpeed:E wind start,E wind end,N wind start,N wind end
TemperatureNormal:temperature
```

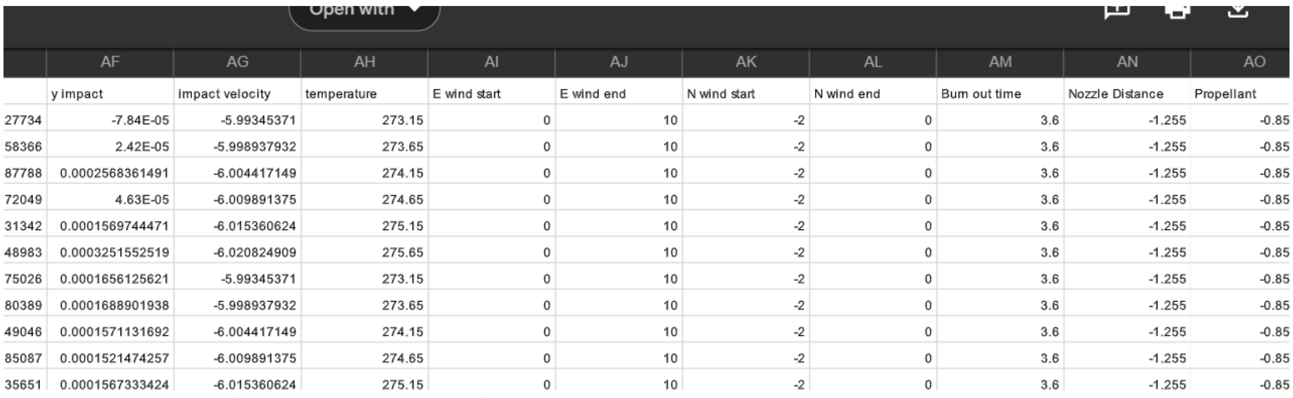
Figure 10: Existing input folders mapped to their primary input variable

Then you need to match it to the input or inputs that are being changed.

Choose one of the two variables to focus on

(Future development may include having a secondary file to check for additional variables, but for now pick one).

You will find the exact column name in the all_inputs.csv file.



	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO
	y impact	impact velocity	temperature	E wind start	E wind end	N wind start	N wind end	Burn out time	Nozzle Distance	Propellant
27734	-7.84E-05	-5.99345371	273.15	0	10	-2	0	3.6	-1.255	-0.85
58366	2.42E-05	-5.998937932	273.65	0	10	-2	0	3.6	-1.255	-0.85
87788	0.0002568361491	-6.004417149	274.15	0	10	-2	0	3.6	-1.255	-0.85
72049	4.63E-05	-6.009891375	274.65	0	10	-2	0	3.6	-1.255	-0.85
31342	0.0001569744471	-6.015360624	275.15	0	10	-2	0	3.6	-1.255	-0.85
48983	0.0003251552519	-6.020824909	275.65	0	10	-2	0	3.6	-1.255	-0.85
75026	0.0001656125621	-5.99345371	273.15	0	10	-2	0	3.6	-1.255	-0.85
80389	0.0001688901938	-5.998937932	273.65	0	10	-2	0	3.6	-1.255	-0.85
49046	0.0001571131692	-6.004417149	274.15	0	10	-2	0	3.6	-1.255	-0.85
85087	0.0001521474257	-6.009891375	274.65	0	10	-2	0	3.6	-1.255	-0.85
35651	0.0001567333424	-6.015360624	275.15	0	10	-2	0	3.6	-1.255	-0.85

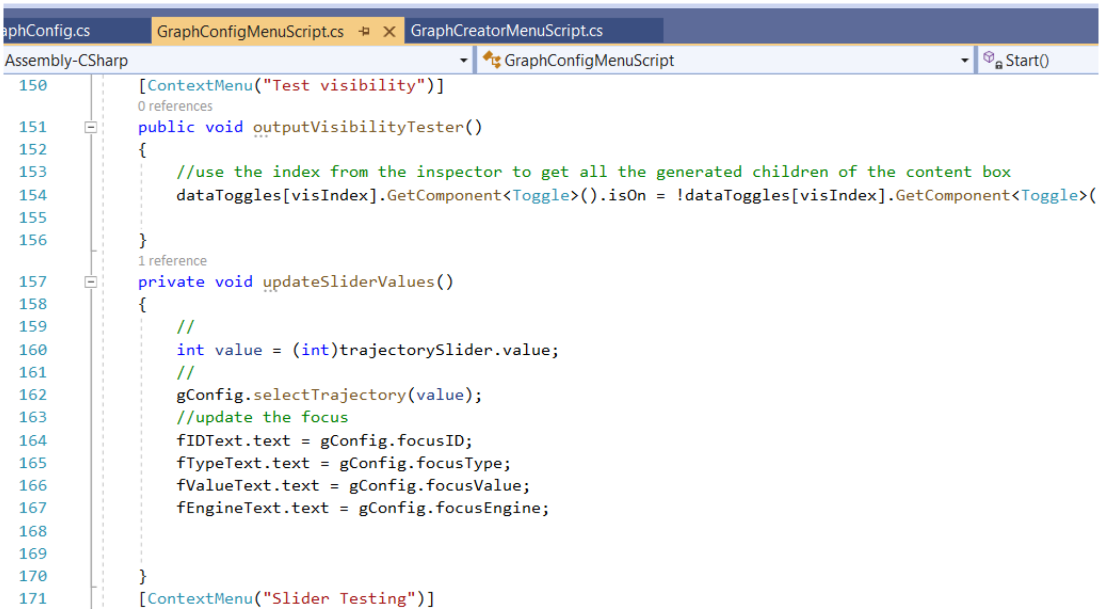
Figure 11: One of the all_inputs.csv files that has been generated

After a ‘:’, paste in the column name next to it’s folder name.

Now the variable details will be updated on the graph config menu panel when the trajectory changes.

Additional Work

The GraphConfig file (attached to every generated graph) accesses the trajectory data and extracts the focus data information from it.
This information is then accessed by the script that controls the graph config menu, GraphConfigMenuScript.cs. Shown below is the function “updateSliderValue”

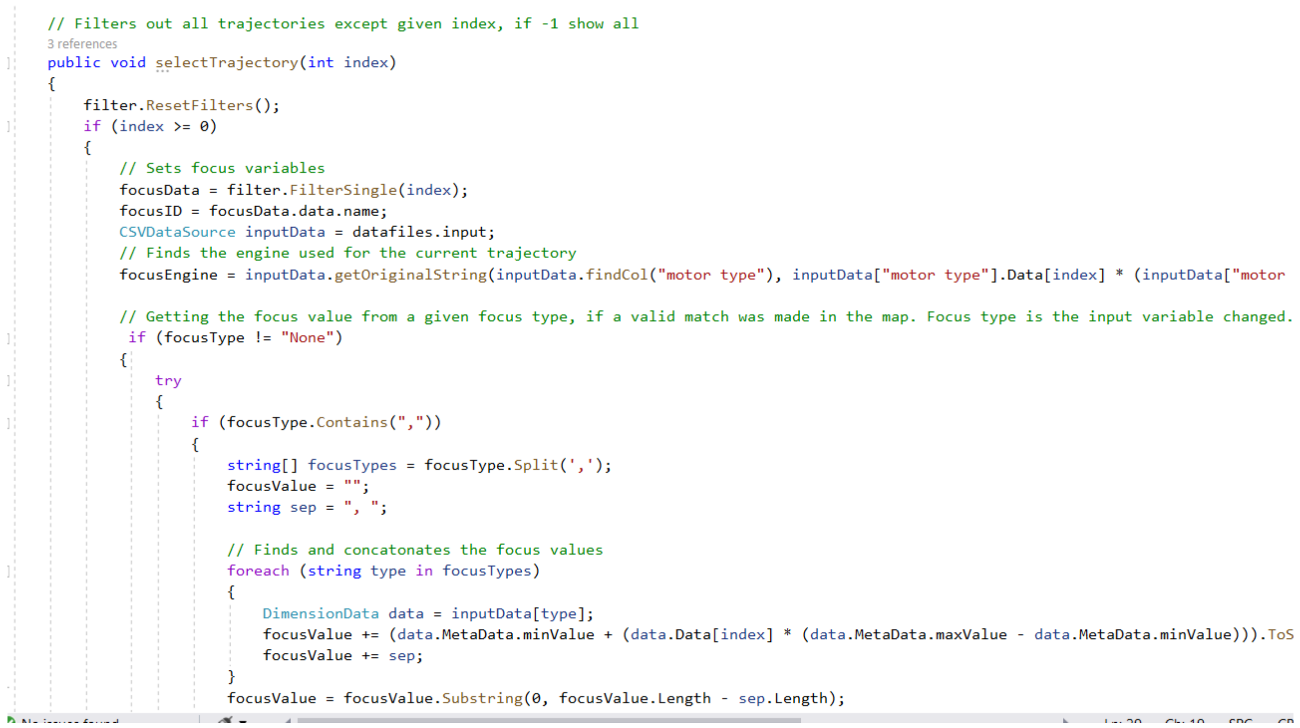


```
[ContextMenu("Test visibility")]
0 references
public void outputVisibilityTester()
{
    //use the index from the inspector to get all the generated children of the content box
    dataToggles[visIndex].GetComponent<Toggle>().isOn = !dataToggles[visIndex].GetComponent<Toggle>(

}
1 reference
private void updateSliderValues()
{
    //
    int value = (int)trajectorySlider.value;
    //
    gConfig.selectTrajectory(value);
    //update the focus
    fIDText.text = gConfig.focusID;
    fTypeText.text = gConfig.focusType;
    fValueText.text = gConfig.focusValue;
    fEngineText.text = gConfig.focusEngine;


}
[ContextMenu("Slider Testing")]
```

Figure 12: Where the menu gets the focus values from

At present, when the slider is moved, only the single variable listed in the VariableFocusMapping file is tracked and used to update the screen.
The function used to do that is selectTrajectory in GraphConfig. As seen below, the CSV data is parsed to get the values based on the trajectory index.

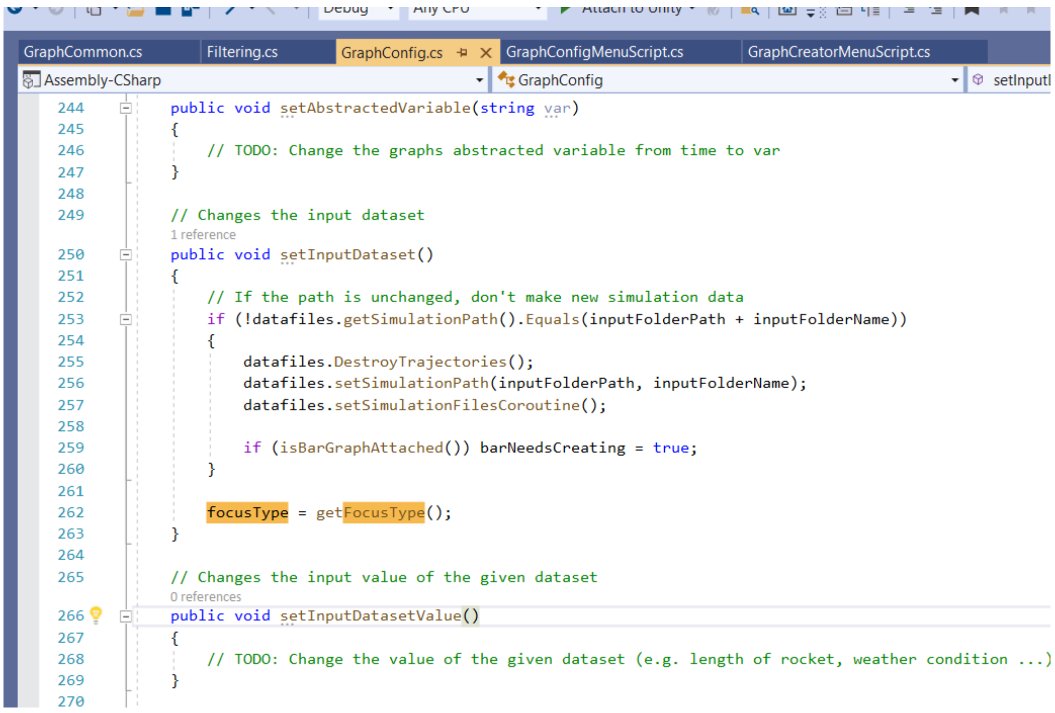
A possible next step would be to determine how best to display the secondary variable and it’s values to the screen when the slider is moved. This might mean checking a secondary file for the input folder name and the secondary variable.



```
// Filters out all trajectories except given index, if -1 show all
3 references
public void selectTrajectory(int index)
{
    filter.ResetFilters();
    if (index >= 0)
    {
        // Sets focus variables
        focusData = filter.FilterSingle(index);
        focusID = focusData.data.name;
        CSVDataSource inputData = datafiles.input;
        // Finds the engine used for the current trajectory
        focusEngine = inputData.getOriginalString(inputData.findCol("motor type"), inputData["motor type"].Data[index] * (inputData["motor

        // Getting the focus value from a given focus type, if a valid match was made in the map. Focus type is the input variable changed.
         if (focusType != "None")
        {
            try
            {
                if (focusType.Contains(","))
                {
                    string[] focusTypes = focusType.Split(',');
                    focusValue = "";
                    string sep = ", ";

                    // Finds and concatonates the focus values
                    foreach (string type in focusTypes)
                    {
                        DimensionData data = inputData[type];
                        focusValue += (data.MetaData.minValue + (data.Data[index] * (data.MetaData.maxValue - data.MetaData.minValue))).ToS
                        focusValue += sep;
                    }
                    focusValue = focusValue.Substring(0, focusValue.Length - sep.Length);
```

Figure 13: selectTrajectory method in the GraphConfig script

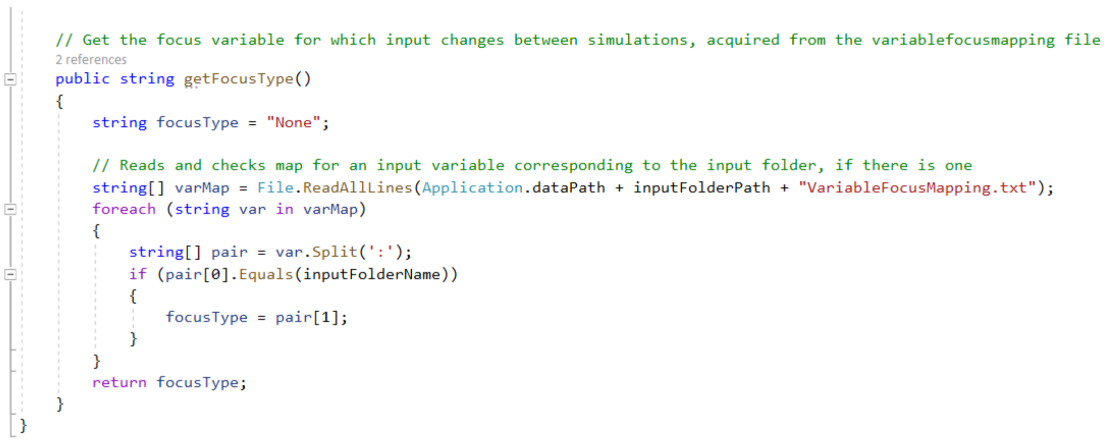
The getFocusType code is inherited from GraphCommon



```
public void setAbstractedVariable(string var)
{
    // TODO: Change the graphs abstracted variable from time to var
}

// Changes the input dataset
1 reference
public void setInputDataset()
{
    // If the path is unchanged, don't make new simulation data
    if (!datafiles.getSimulationPath().Equals(inputFolderPath + inputFolderName))
    {
        datafiles.DestroyTrajectories();
        datafiles.setSimulationPath(inputFolderPath, inputFolderName);
        datafiles.setSimulationFilesCoroutine();

        if (isBarGraphAttached()) barNeedsCreating = true;
    }

    focusType = getFocusType();
}

// Changes the input value of the given dataset
0 references
public void setInputDatasetValue()
{
    // TODO: Change the value of the given dataset (e.g. length of rocket, weather condition ...)
}
```

Figure 14: Where the focus Type is called in from GraphCommon within setInputData

The code may be adapted to retrieve a secondary focus type from either the variableFocusMappingFile or a new file created to hold secondary variable types.



```
    // Get the focus variable for which input changes between simulations, acquired from the variablefocusmapping file
    2 references
    public string getFocusType()
    {
        string focusType = "None";

        // Reads and checks map for an input variable corresponding to the input folder, if there is one
        string[] varMap = File.ReadAllLines(Application.dataPath + inputFolderPath + "VariableFocusMapping.txt");
        foreach (string var in varMap)
        {
            string[] pair = var.Split(':');
            if (pair[0].Equals(inputFolderName))
            {
                focusType = pair[1];
            }
        }
        return focusType;
    }
}
```

Figure 15: getFocusType from GraphCommon

A more complicated task might be to connect multiple single variable datasets to the same axes and layer them over each other. All sets except the currently selected set would be hidden from sight. A click of a button could cycle through the set, allowing a better view of how changing the inputs will affect the outputs, as even small changed would be easier to detect. This would require more work with the IATK, and also possibly some custom work. This is a task that is not really necessary, since multiple datasets can already be displayed in the same scene for comparison.