Rocket design with Python

Mamfoumbi Jean-Michel Harkati Ilias

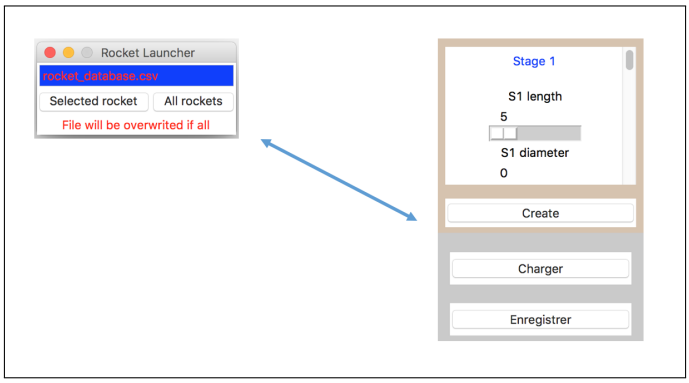
May 2020

1 Introduction

The main objective of the IN014 project was to design rockets and simulate trajectories. The idea was to use our computer courses on Object oriented programming and to learn how to use the package pandas to make a python class we could store in a retrievable, modifiable and exploitable database according to our needs. Our work was divided in three tasks : the first one was to create a Rocket class using OOP and with the Pandas library implement a database where we could store and load rockets. The second one was the simulation of trajectories given different characteristics and initial conditions and the last one was the creation of a graphical user interface (GUI) in order to display the rockets and manipulate them. Finally, we had to make a tool which is both simple in term of physics but useful to create preliminary design of rockets for example to conceive LEO or GEO mission. Here the link towards our git : `https://github.com/jm-mamfoumbi/IN_104_Harkati_Mamfoumbi_Rocket_design/tree/master/src`

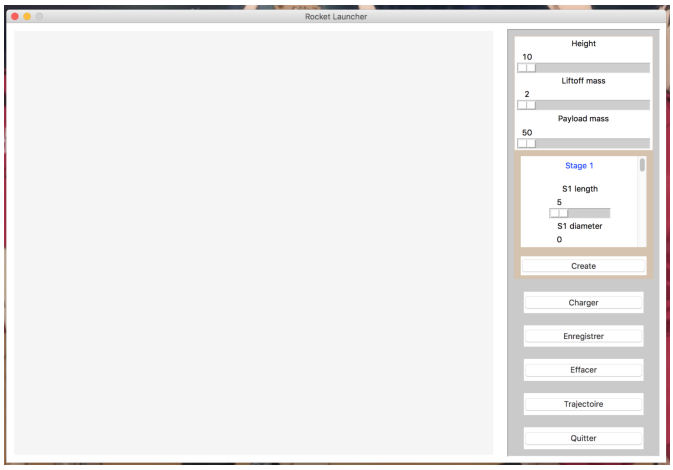
2 Database use

The database had to be really adaptable in order to be useful. As it was required, there is the possibility to load an existing rocket from the database and to work with it but also the possibility to create a rocket item and then to save it in the database. In order to make it, we created two functions which are available in **main.py** : **loadrockets** and **saverockets**. They allow to create a list of rockets from a csv file and to save a list of rockets in a csv file (we suppose it exists).



3 The GUI

The GUI is coded in the files **draw_rocket.py** and **rocket_window.py**. It was implemented by using the Tkinter library. It is divided in two parts : the big frame on the center - the display screen - that display rockets and on the right side a dashboard with buttons to create, load, save or erase rockets, but also to plot the trajectory and even to quit the application.
The display screen has been made so that it could display all the rockets that are currently loaded. All the rockets are shown faithful to their dimensions. The screen can also load as many rockets as the user wants, that thanks to a dynamic displaying that center the screen around the rocket currently selected. And a rocket can be selected by navigating using the directional keys of the keyboard or simply by clicking on it. Indeed, all the rockets have a clickable hitbox surronding them.
Concerning the dashboard all its buttons - except the *Quitter* button - give access to a small dedicated window so that all the manipulations - like load rockets, save rockets, create rockets etc. - can be done easily and powerfully.



4 Unit test

We decided to focus most of our unit test on trajectories issue because it was easier to see if the GUI worked correctly. In addition, the GUI limits the user possibilities so the user has to follow the instructions to create a rocket item, load it and plot its trajectory. On the contrary, the simulation of trajectory is way more unpredictable since it always depends on rockets characteristics. Even if the GUI globally prevents the user from choosing inconsistent characteristics (he can't choose negative parameters), it's safer to make sure the simulation of trajectory won't start if the rocket item doesn't have physical sense. For example if the propellant mass in stage one is more important than the total mass of stage 1 it means the rocket item doesn't have physical (the stage one mass is supposed to be greater than the propellant mass because it also takes into account the construction mass). That's why we create an exception class called Badvalue in the file **rocket trajectories.py** to take care of inconsistent parameters. Then we detailed the cases of physical abberation and made the program raise an error when it happens. Finally to make sure our simulation correctly handle we made unit tests in the file **test-rockets-trajectories.py**. We made the following tests :

- negative lift-off-mass
- negative stage-one mass
- negative propellant mass on stage-one
- propellant mass greater than stage-one mass
- negative stage-two mass
- negative propellant mass on stage-two
- propellant mass greater than stage-two mass

5 Simulation of the trajectories

5.1 Objective

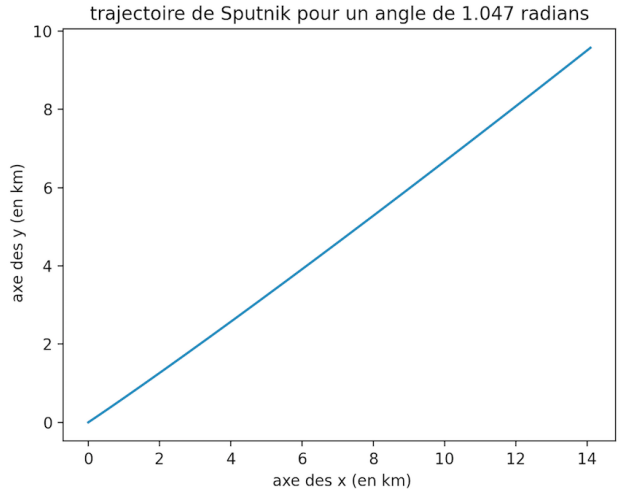
The simulation of the trajectories has been implemented given few starting hypothesis : rockets paths are consider to be flat trajectories, rockets move in a cartesian x-z framework, gravity is constant, drag is neglected, thrust, propellant mass flow and pitch angle are constant.
The resolution of the dynamics equations is much more simple and given the analytical equations of $x(t)$ and $z(t)$, we can plot the trajectories until the rocket lands ($z = 0$).

5.2 First attempt and major issues

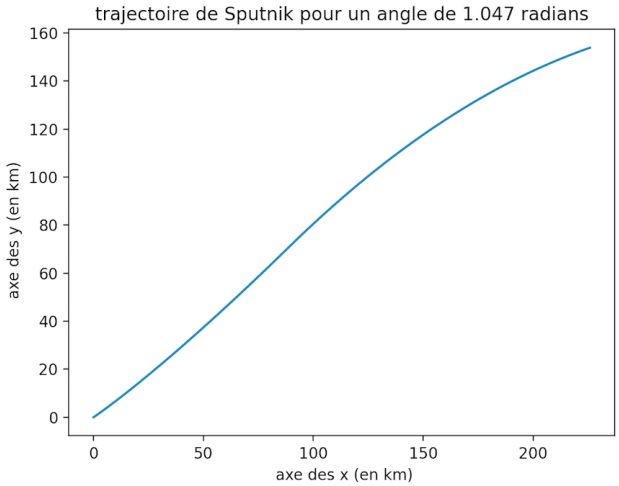
In order to plot a rocket trajectory we thought it was a good option to create simple functions so we choose to create two functions for the trajectories : one which deals with the one-stage rockets and another which deals with the two-stage rockets. Then we created a function which takes into argument a rocket and then use the right function given the number of stages of the rocket. We created another one which deals with the display of the trajectories with the matplotlib.pyplot library given two list of coordinates ($[x_1, ..., x_n]$ and $[z_1, ..., z_n]$) at consecutive time with a time-framework of $0.1s$.
Another difficult aspect of the simulation was the modeling of a two-stages rocket because there are three different phases in the flight contrary to a one-stage rocket. Indeed the first phase is a powered flight due to the first stage, the second phase is a powered flight due to the second stages with initials conditions especially in speed (because the rocket is already moving) and only after the extinction of the second engine the rocket enter in a ballistic flight.
We had no major issues when we tried to implement the trajectory of a one-stage rocket but when we started implementing we observed inconsistent results for several rockets.

5.3 Alternative solution

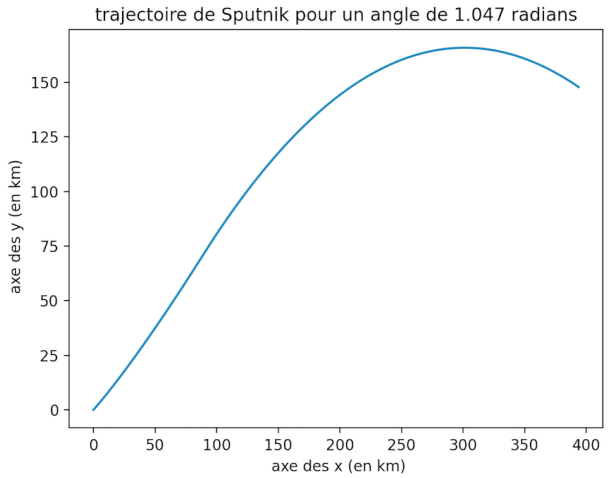
Since the physics aspect wasn't the main objective of the project we decided to simplify our hypothesis in a first time to obtain correct results and add the simulation part to the GUI. We considered the first and second phases take place at the sime time by taking into account one single thrust (S1 thrsut + S2 thrust) and one specific impulse (S1 Isp + S2 Isp). Then we worked as if the rocket has one stage.



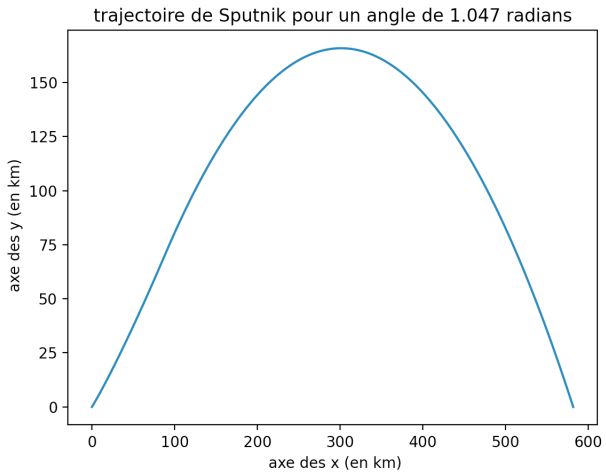
(a) Powered flight



(b) Balistic flight after extinction



(a) Balistic flight after atfer the culmination point



(b) global Sputnik trajectory with an angle of 60

6 Conclusion

As a conclusion, we would say that our main issue was to think every function at a larger scale, indeed at the end every part of the project had to connect with the others in order to deliver a useful tool. It was all the more difficult since we worked as a team and had different habits and point of views. We also had to take in the hand new tools both for programming as the pandas package to work with database and also the tkinter library but also for working in group as git. Finally we didn't understand why our first simulation model didn't work, the simulation of trajectories lost in accuracy but we managed to connect the alternative solution and the gui to make a tool which managed to create and design a rocket item and plot a simulation of trajectory which seems rather relevant in most of cases.