Report of IN104: Rocket Design with Python

Johnny Yammine and Olivier Laurent

22 May 2020

Abstract

The objective of this project was to build a *Graphical User Interface* for rocket design. The idea was to developp a software at least capable of:

- Handling a database using pandas. This means adding, reading and removing rows. This database includes for instance the name, the year of first launch, the number of stages and, for each stage, the height, the diameter, the specific impulse, the initial mass and the propellant mass. For further personalization, we chosed to add the possibility of having boosters on the rocket but also to choose colors for each stage.
- To draw a sketch of a rocket when given the necessary parameters.
- To allow the user to compare the rockets in a separated interface
- To be able to show the trajectories of rockets corresponding to a certain set of hypotheses.

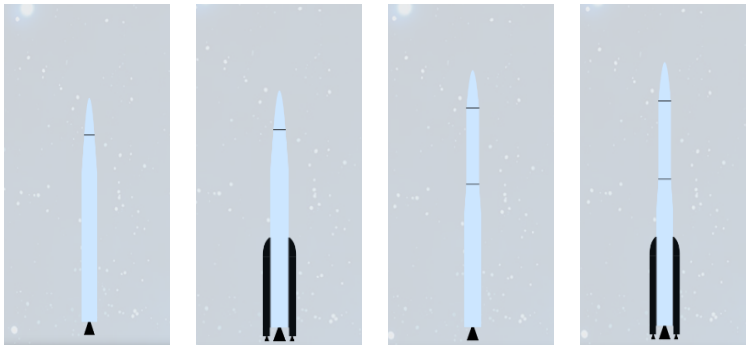
These necessary specifications led to the choice of the technologies we used. We decided that we would use python's FLASK to run the server (locally) which routes would lead to HTML5 templates styled with CSS3, animated and secured by JAVASCRIPT. Using flask allowed us to keep a Object-Oriented Development but also to lead many tests with the UNITTEST module. We used the UI-KIT framework to have a better rendering without too much effort.

Going forward in the project, we decided to use the knowledge aquired in -MO102: Optimal Command for Ariane 5- MATLAB course to build a realistic physics engine and *machine-learning* optimizers. This part has been developped in C for optimization purposes. These C functions are directed by python algorithms using the module CTYPES. This choice here again allowed the development team to take advantage of the powerful UNITTEST module.

1 Frontend Features

1.1 Drawing the rockets

In this project, we restricted ourselves to 4 types of sketches that have been included below. We indeed limited the number of stages to 2 but we took into account the possibility of adding boosters to the rocket.



This part is thus composed by four functions which can each draw a sketch on a HTML canvas. These functions take many arguments such as the height of each stage but also the zoom decided by the user. They propose a scaled graphical representation of the rocket. Each function is also able to roughly print the initial mass vs. propellant mass ratio if it is desired by the user.

1.2 First Page - "Design your Rocket"

This page allows the user to design new rockets while seeing how the rocket looks but also to edit an existing rocket. It is composed of a form and a canvas put side by side. While the user enters the value of each input making up the form these values are taken into account and the rocket is sketched accordingly.

The form is secured in two ways. First, one cannot add the rocket to the database (pressing the button at the bottom of the form) if any information has not been given. Second, if the sum of the heights (respectively mass) is bigger than the total height (respectively mass) the inputs turn red to warn the user.

1.3 Second Page - "Compare two Rockets"

This page allows the user to compare two rockets already built in the database. The comparison goes two ways. The rockets are sketched side by side and their parameters are shown beside them.

The user can choose the rockets thanks to two deployable lists which are automatically filled while the page is loading.

1.4 Third Page - "Optimize the Trajectory"

This page allows the user to lead an optimization process in order to know if the designed rocket is capable of reaching the specified orbit, but also to find which command will be needed to reach this orbit. The command is the angle of the rocket at a certain time.

Once again, the user can load the rocket he wants to use with a deployable list. The rocket is sketched and its parameters can be all reviewed before the test.

The user can then specify the parameters of the mission in the corresponding form. The user can choose the type of flight (surborbital, low orbit or geostationnary transfer orbit) and provide the corresponding parameters (which are currently the excentricity and the semi-major axis) but the user may also change the default payload mass which is automatically imported from the rocket's parameters.

Finally, the user can choose the optimization model and its parameters which will then provide a certain precision but will take time. Right now, the progress cannot be seen on the browser's screen as it would require a Websocket that the development team didn't have time to build. Yet, this information can be seen on the terminal.

1.5 Fourth Page - "Trajectory"

This page allows the user to see the trajectory obtained by the optimization process and compare it (currently only in semi-major axis) to the trajectory that was planned. The trajectory is plotted using JAVASCRIPT's d3. The development team found it very difficult to use but very flexible and interesting.

2 Use & Backend Features

2.1 Use

The project is based on a FLASK server. It could have thus been easily put online on the server of DaTA (ENSTA's association for computer science). The development team has though found that it would probably put too much weight on these servers and thus decided that the use would remain local. This means that a few manipulations are required by the user. The user must for instance build a python environment. A "make" command might also be necessary though the C shared libraries have been put directly on git by the development team. Every information concerning the installation can be found on the readme.markdown (in the root directory).

2.2 Flask Server

The flask server is mainly composed of *app.py* which contains what is needed for its initialization but also the API to redirect the user and to fill its requests. Yet,

app.py imports different modules for instance to use the Rocket class defined in *rocket_ modules.py* but also the optimizers developped in *optimizers.py*.

2.3 Database handling

The database consists in one CSV file containing all the information about the stored rockets. This database is loaded in a PANDAS DataFrame each time the user changes its location on the website. It is then used by many functions, which can extract a requested rocket by name or add a rocket for instance.

2.4 Physics Engine

The whole physics engine has been developped in C. We made the assumption that the trajectory stayed in a plane in which we defined a cartesian basis. In this coordinate system we can use Newton's second law to find the acceleration and later deduce the speed and the position of the rocket. The implementation thus mainly consists in functions computing the forces that are exerted on the rocket such as its thrust and the weight. Friction has been negleted so far. There is also an implementation in C of Runge-Kutta 4, an ordinary differential equation solver and of the Verlet method which garanties that the energy of the system stays the same when the thrust is null.

2.5 Optimization

Let us consider the e_T and a_T, respectively the target orbit's excentricity and semi-major axis. The idea of our optimization functions is to minimize the following function:

$$J = \frac{(e - e_T)^2}{e_T^2} + \frac{(a - a_T)^2}{a_T^2}$$

This fonction is computed in C.

Our lever to find the best trajectory is the command. It is the angle of the rocket in our cartesian basis. The idea, if the dimension of the hypercube in which we define the command c is called n, is thus to find the following value

$$\boxed{\min_{c \in]-\pi;\pi]^n} J}$$

One way to do this is to try randomly a certain number of commands. The best one is kept and enhanced with a C gradient-descent algorithm. This is the principle of the *random optimizer*. One other way to do this is to use a *genetic algorithm*. In this algorithm, we build for each epoch a new population (constituted of c vectors) that we mix, mutate and then enhance with the gradient-descent algorithm.

The gradient-descent is done in C in a non-fixed dimension. This means that the command of the rocket is stored in a linked-list. At each turn, the algorithm modifies the commands of a decreasing value h, computes h times the gradient

of J, and uses this product to modify the linked-list. Conditions compel the result to be improved a certain number of times (which we call gradient-descent steps).

We have not forbidden the rocket to go below the radius of the Earth because it made the optimization process much longer. This is why impossible trajectories are sometimes proposed.

3 Going Further and Conclusion

3.1 Going further

During the project, we had many ideas to make our project as good as possible. If we managed to realize some of them (command optimization, boosters, colors) we were unable to materialize others because of the lack of time or technical issues. Here is a non-exhaustive list of these ideas:

- Allow the user to see through the fairing
- Compute the height of the fairing with the total height of the rocket and not with a specified number of pixels
- Add a new database to store the different optimization results
- Add a new page to launch a rocket in real time
- Multithread the gradient-descent algorithm with pthreads (it is easy and does not even need any mutex)
- Developp an intelligent auto-rocket creator
- Take the friction into account
- Maximise the payload mass for a specified orbit
- Add the Moon and Planets and allow more complex commands (cut the liquid engines, ...)

3.2 Conclusion

Throughout the project, the development team has had to familiarize with many new technologies such as FLASK, d3 or the module CTYPES. The web and tutorials have been essential to learn how to use these technologies and thus to be able to achieve this work. The team has also been facing many technical issues and has struggled much to plot the trajectory for instance. And we sometimes have not been able to solve issues. We also realized that our project could have been endlessly improved and we so understood the importance of deadlines when it comes to IT management.