**Rocket Simulator**



**Name** Ross Hill

**Course** Grade 12 Physics

**Code** SPH4U

**Instructor** N. Chow

**Date** 24 May 2016

### Purpose

The goal of this project was to simulate a rocket taking off from an earth-like planet, and graphically display data about its motion. It takes inputs of a rocket’s base mass, thrust, amount of rocket fuel, fuel density, and rate of fuel ejection, and from that derives rocket’s acceleration, velocity, and displacement as a function of time.

## Procedure

To begin the project, I built an alpha program that used kinematic equations of either constant velocity, acceleration, or jerk, to solve for missing variables. The code to manage the program’s functions, hold the equations, and solve said equations were each held in their own classes, which improved the code’s readability.

After the code for that part of the project was complete, I began laying out how I would change the program so that instead of the user entering variables like “initial velocity” and “final position”, the user would instead input specifications about a rocket, and from those specs, the program would calculate the needed variables.

First, to get the rocket’s specifications, I wrote the function “**getSpecs()**”, which uses python’s standard input (the terminal) to ask for details about the rocket. Once those details are collected as strings, they are returned in a dictionary, as ‘key:value’ pairs. For instance, “thrust: 16000.0” could be one such pair.

Next, the function “**deriveVectors()**” is called, and it returns all the initial values (v1,t1,d1) as zero, since in the case of a rocket taking off of a planet’s surface, those values will always be zero. “a1”, however, is calculated based on the rocket’s thrust, it’s mass including fuel, and the acceleration due to gravity.

At the beginning of this phase of the project, I had acceleration due to gravity hardcoded as the constant “9.8”, however, it occurred to me that as a rocket travels away from the planet, the acceleration due to gravity will weaken. Because of this, I made a new function, “**accDueToGravity()**”, which returns a value based on the planet’s mass, the distance of the rocket from the center of the planet, and the gravitational constant.

After the necessary vectors of the rocket’s motion have been derived, the function “**drawGraph()**” in the class “**graph**” is called, which iterates over the amount of time the user specifies, calculating the acceleration, displacement, and velocity for every second of the rocket’s flight. If at any time the rocket’s displacement goes below 0 (under ground), the program is aborted and warning text is shown. Displacement, velocity, and acceleration are all found with their own functions within the “**rocket**” class (**position()**, **velocity()**, & **acceleration()**) which get passed the current time in the simulation, as well as any necessary information about the rocket. The current fuel remaining is necessary to find acceleration, so the operation to find that is kept as it’s own function, which is passed the current simulation time, the initial amount of fuel, and the fuel density.

Keeping these operations in their own functions improve the modularity and readability of the code, which makes it easier to maintain.

On each iteration of these calculations (one per second in simulation-time), their values are appended to lists which will be used to display on graphs. Once all of the iterations have been completed, the function display is called three times. Each time it is passed either the list of position, velocity, or acceleration values. The time values are shown on the X-axis of each graph, as time is the independent variable, and conversely, the position, velocity, or acceleration values are shown on the Y-axis.

Since this is showing the change in values over time, I opted to display a line-graph. These graphs are generated with the widely used “**matplotlib**” python module. This module allows for customization of many aspects of graphs. In this case, I included a main and axis titles, and I included major and minor grid-lines.

With the project fully built, I made a “**main**” class inside the file **physics.py**, which has a function “**start()**”, which calls all of the appropriate functions from the “**rocket**” and “**graph**” classes in **rocket.py and graph.py** respectively, to make the program run.

To run the project, all one has to do now is enter the command, “**python */path/*physics.py**” into Terminal. To make it even easier, I wrote a short applescript with opens terminal and enters that command. Then I made an Automator application that runs that applescript program, so one can simply click the application icon on the dock and have the **physics.py** program run.

## Observations

As I began writing this program, I expected acceleration to grow linearly, as fuel gets expelled linearly. However, since the total mass of the rocket decreases, the gravitation force acting on the rocket also decreases linearly, which makes the acceleration over time to grow quadratically. Then I realized that as the rocket moved further away from the earth’s surface, the acceleration due to gravity for the rocket would decrease. So, on the surface of the earth-like planet, the acceleration would be about 9.8m/s in the negative Y direction. However, this decreases as the rocket travels in the positive Y direction. This, in combination with the already quadratic curve of the rocket’s acceleration over time, combines to make the acceleration curve over time a cubic function, velocity quartic, and positon quintic.

## Conclusion

On the whole, I’m proud of what I achieved with this project. Using primarily python, I made an object-oriented program that not only accurately depicts the motion of a rocket in a vacuum, but in doing so, helped me discover that rockets travel with constant snap, not just constant jerk. Through this project, I connected many of the subjects covered by this year’s physics curriculum, in an efficient and organized program.