# Rocket Flight Simulator: Design

## 1. Introduction

### 1.1. About the Project

This project is a semi-realistic rocket flight simulation tool. It allows users to customize their rocket and predict the trajectory of their rocket in flight. It contains methods to accurately measure thrust, gravity, atmospheric conditions, drag, and many more relevant models when simulating a rocket's trajectory. The program creates graphs to both visualize the rocket during its flight, but to allow the user to better understand its behavior following the completion of the simulation. Lastly, the program allows the user to save their simulation data to a .txt file, allowing them to further process the data and perform whatever analysis they would like.

### 1.2. Why I Created This Project

I chose to create this project for a couple of reasons. One of the first and biggest reasons was my interest in both rocketry and simulation tools. Here at University of Maryland, I have been a member of the Terrapin Rocket Team, working a lot with rocketry and aerospace projects. I have learned a lot about rockets, and the work that goes into designing, manufacturing, and testing them. Additionally, I have grown an increased interest in simulation and modeling tools, through my use of tools like MATLAB, ANSYS, and SolidWorks for various projects. I wanted to create a project where I could combine these two interests, to create a program that could be used in a realistic scenario. Through the process of creating the project, I knew I would also learn so much about rocketry and simulation, especially about the areas commonly of lesser focus, like the basic physics and background development of such tools.

Another major reason I chose to create this project is to create a framework for a rocket simulation tool, which I can add onto later. Ideally, this project is not something I am finished working on once I submit it, but rather something I can continue to develop. There are many other components I would like to add to the tool (which I will touch on in a later section) but simply do not have the time to effectively add them all presently. By creating this tool, I have designed it such that it should be easy to add more features to down the line, meaning it is a project that is constantly growing, and becoming more in-depth and realistic. Eventually, I may be able to create a tool which I could use in an actual engineering environment, like within the Terrapin Rocket Team. Again, the current project has much needed work and additions before I would consider anything ready for

professional use, but the idea of creating a foundation for a tool to be later added onto was really the purpose of this project.

My third and final reason for creating this project was to create a tool which bridged the gap between simple hand-calculations, and more complex software like OpenRocket or RASAero. This tool can be faster, and more accurate than hand calculations, yet it is not as complex to use as some more professional-grade software. This software takes advantages to both approaches, taking the simplicity and speed of hand calculations, while providing accurate simulation results and data like those from more professional tools. By doing so, the program fits nicely into a role where previous software may not have existed yet. It allows users to input simpler models of their rocket while finding accurate simulation results, allowing them to predict the behavior of their system in a quick yet precise manner.

### 1.3. Rocket Physics Overview

Before reading further, it is important to understand the basic physics behind rocketry. Rockets are not actually that complex, as to put it simply, they just go up, and come back down to Earth, and there is not much more to it.

To start understanding, it is important to understand all the forces on a rocket, at any time. We will mostly be looking at forces in the vertical direction, oriented parallel to the length of the rocket, as these dominate the behavior of the rocket. The first force is from your propulsion, and this force is your thrust. At launch, the rocket's motor generates massive amounts of thrust, sending the rocket flying into the air. The amount of time the propulsion is firing varies from system to system, but in model rockets the motor tends to burn out (or run out of fuel) within a few seconds. This means, the rocket only receives the thrust force for the first few seconds after launch. The next major force which plays a role in the system's behavior is gravity. Gravity is the attractive force trying to pull the rocket back down to earth's surface. The rocket experiences the same magnitude of gravitational force, however at large altitudes, this force begins to slowly decrease. The last major force which is important to understand is the drag force. The force of drag is the resistance to an object's motion through a fluid (in this case air). It can be calculated using the equation

$$D = \frac{1}{2}\rho v^2 C_d A$$

where D – drag force, $\rho$ – density of fluid, v- velocity of object, $C_d$ – coefficient of drag, and A – face area of rocket. All these values are known, or can be found, prior to calculation of the drag force.

One final force that is important to consider is the drag force from the rocket's chute. To slow a rocket down during its descent, preventing it from crashing at an extremely high

velocity, parachutes are often deployed. This force does not act on the rocket until the parachute is deployed, which is often a parameter designated by the engineers of the system. This parachute drag would use the same drag equation above, except $C_d$ and A are the coefficient of drag and surface area of the parachute itself, rather than of the rocket.

### 1.4. Research

Before beginning the code for my project, it was important to have a strong understanding of all the principles and concepts I would need to apply in my simulation tool. I already had a general understanding of the physics of a rocket, and the steps for launching and landing it, but I wanted to reinforce my understanding. I looked at *Introduction to Flight* by John D. Anderson & Mary L. Bowden. This textbook is very well written, and has a few small sections on rocketry, specifically physics and its propulsion. I reviewed these sections, allowing me to move further within the project.

Another area of research which was important was understanding how to make a simulation tool from scratch. Creating a simulation tool, trying to accurately model a rocket system and its physics, was something new to me. One important area was looking into integration methods. I had to find the various integration methods often used in simulation software like the one I was planning to create. I had to find the benefits and drawbacks of each, to select the best one for my tool. I then had to further investigate the importance of, and how to optimize your simulation's time-step selection. In my simulations, especially with the preset simulations, it was important to find a time-step between calculations such that I could accurately model the system, while not completely slowing down the simulation. This software was to be written in Python, which is not the fastest, and designed to be quick and efficient, so it was crucial to find a time-step to best optimize my model.

The last major area of research I had to do for this project was to learn how to accurately model our environment. As altitude changes and you leave Earth's surface, the environment changes as well. I had to find how I could best model this, in an accurate manner, without completely slowing down simulations. I looked into modeling the environment, including air density, pressure, and temperature, and well as gravity, and how all these variables change with altitude.

### 1.5. Assumptions

It is important to discuss the assumptions made within this project.

One of the first major assumptions made, is that rocket's fly straight up, and come straight back down. This is unrealistic in the sense that even if you are not designing your rocket to launch from an angle, or move laterally, it still will be due to wind which can

create a horizontal force on your rocket. Initially, I planned on modeling wind, however, I did not see a purpose in the beginning steps of my simulation, as it does not affect the velocity or apogee of a rocket system.

Another major assumption I made within my tool was that air is incompressible. This assumption is only valid at lower mach numbers, ~0.3 and below. However, once you reach higher mach numbers, this assumption begins to breakdown, and the compressibility effects of air begin to show in the system's behavior. Therefore, this incompressible assumption is valid for low-speed rocket flights, where the rocket is flying at low mach numbers, however for higher velocity flights, and even transonic and supersonic flights, this assumption can lead to larger errors in the simulation.

### 1.6. Similar Projects

There are many tools that can be used both academically and professionally for rocket simulation. There are open-source tools like Open Rocket, which offer a very accurate and detailed tools to allow users to model their rocket in software, and simulate its performance. There are also more professional tools, which often come with high price points, but are even more accurate models for general rocketry and aerospace simulation. Some of these tools include ANSYS packages, RASAero, and Rocksim.

## 2. Design Process

### 2.1. Initial Goals

As mentioned in earlier sections, the main goal of this project was to create a realistic, yet easy to use flight simulation tool, which could accurately predict the trajectory of a rocket given its parameters. I wanted to create a tool where I could input a set of parameters describing a rocket I would like to model and be able to accurately simulate what its flight would look like. I wanted to understand its apogee, its peak velocity, and its landing. My goal was to create a tool where I could quickly change the characteristics of a rocket and quickly understand how those changes affected the rocket's behavior.

### 2.2. First Prototype Implementation

My very first step when I first began writing the code for the project was to set up the environment. Using the knowledge I had gained from my research, I found simple models I could use to accurately model both the atmosphere and gravity. I used the ISA model for the atmosphere, allowing my simulation to accurately model the air density, temperature, and pressure as a function of altitude. This would allow me to create a much more realistic model, rather than assuming MSL conditions everywhere, especially for rockets which are

flying to great altitudes. Additionally, I modeled gravity, using a simple equation which determines the acceleration due to gravity given an altitude. Although the change in gravity is almost immeasurable for many smaller scale flights, I wanted to include it for more accurate results, and it does not slow down the simulation much if at all.

Once I had modeled the environment, I wanted to create a class which would be the basics behind the rocket which was being simulated. This class would contain all the relevant variables and parameters related to the rocket, including its dry mass, its motor choice, chute area, and many more. Additionally, this class would include a few important helper functions, including an initialization function, allowing the user to initialize their rocket object when setting up the simulation, as well as functions to determine drag and thrust at a given time t.

Once the environment and rocket class were created, I then moved onto the setup of the program. I created a setup function, which would initialize the user's rocket after they input all their rocket's parameters.

Lastly came the real physics loop that was doing the real simulation. This main function included first initializing the rocket and some initial environment/motion variables, like force and acceleration. Then the function moves into a loop, which would continue until the code detected the rocket to have landed. At the beginning of the loop, the program would calculate all the forces on the rocket, including thrust, gravity, and drag. Then, using Newton's second law, it would calculate the acceleration of the rocket at that time. I next implemented an explicit Euler integration method, where at each timestep it solves for the velocity and position, given previous values and the current acceleration. To put it simply, the program was solving these below kinematic equations.

$$v_f = v_i + a\Delta t$$

$$x_f = x_i + v_i\Delta t + \frac{1}{2}a\Delta t^2$$

By solving these equations for distance and velocity, using small $\Delta t$ values, the simulation tool can accurately integrate the trajectory of the system. Lastly, to finish this physics loop, I printed out altitude value in the terminal, to monitor the physics loop, debugging where needed.

### 2.3. Issues in Early Development

One of the first issues I experienced in early development was the lack of a motor in my rocket simulation. In my rocket class, I rather just had a net impulse caused by the motor, and while this allowed for a rough estimation of the thrust caused by the motor, it was not

nearly as accurate as creating a model for the motor itself. This assumed that thrust output is constant during motor burnout, and this is never the case.

Another problem I had initially was the lack of visualization tools. While I could use print statements to the terminal for debugging, I had no real way to effectively monitor the behavior of my simulated system. I needed a better way to view the rocket throughout the simulation, to understand its motion, without having to look through the thousands of lines of output in the terminal.

One major problem I had with my mathematical modeling was the instability from my Euler integration method. In some equations, I was using a mix of current time-step variables with other variables from the previous time-step. For example, I was originally using the previous time-step's velocity with the current time-step's acceleration in one equation, which is mathematically incorrect.

This mathematical modeling instability was also causing a problem in my altitude calculations, where at the very beginning of the sim, when thrust is low, the model was calculating the rocket as moving in the negative direction, creating a negative altitude. This caused a long series of problems with my landing logic that finished the loop, which was originally just designed around a negative velocity and altitude being less than or equal to 0.

### 2.4. Second Iteration: Improved Physics Modeling

After my initial iteration, I then moved on to improving the physics models within my simulation.

The first major change I made was adding logic ensuring that the physics loop does not close early, requiring a certain number of passes are made in the loop before it can close. This prevented any of the issues where the landing logic is read as true before the rocket has even left the ground, ending the simulation prematurely.

Another change I then made was to add a motor class, containing a few motors to select from with varying thrust curves. I found these thrust curves from thrustcurve.org, which contain accurate data from plenty of different motors, which I could then implement into the motor class. The class contains other information about each motor which are then used in other calculations and allowed me to model around more accurate thrust from the motors. Using interpolation, I could create non-static thrust during motor burnout, adding a major feature of realism.

Like the motor class I added, I added a mass depletion model. Obviously as the motor burns out, it loses mass which was the propellant. I modeled this change, so the physics

loop would only include the rocket's structural mass, and remaining mass from propellant in its calculations. This makes a substantial difference in rockets where the propellant makes up a significant portion of the mass.

Lastly, I wanted methods to allow the user to visualize their rocket during its flight, and so I added some graphing. I added graphs, which would allow the user to visualize the altitude and velocity as a function of time, adding more qualitative results to the simulation and another means to ensure the simulation was working properly.

### 2.5. Final Implementation and Feature Additions

After I was happy with my physics modeling and simple graphing, I wanted to add more features to make the simulation more realistic and complex.

The first feature I added was chute deployment. In real rocketry, you always plan to bring down your rocket with a parachute, slowing its descent to a safe speed. To add this, I added a chute area and chute deployment height attribute to the rocket class. In the physics logic, once the rocket has reached apogee, and then reaches the chute deployment height, it triggers the chute to deploy. To calculate the drag the chute deploys, I used the same drag equation shown earlier, where A is the surface area of the chute, and I used a universal $C_d$ of 1.5 for all parachutes, as this is a realistic value.

The next feature I added was live graphing, to allow the user to even better visualize their system's behavior. While the position and velocity graphs were great, I wanted a 'real-time' visualization, so the user could better understand the behavior. I considered using a video game-like animation for this but thought an animated graph would better fit the project. The graph animates the behavior of the rocket in almost real-time (speed depends on your computer processors performance), and then closes when complete, allowing the user's to still view the static graphs which are easier to use and look closer at.

Last, I wanted to add some preset rockets which the user could use. These rockets would offer examples to the user, allowing them to first mess around with the software using verified rocket setups. Once they became more comfortable with the simulation tool, they could then move onto experimenting with their own setups. I added five different presets, each containing a different motor and realistic parameters to a rocket you would use that motor on. These presets would also improve my efficiency in debugging as well as verification, which is talked about in the next section.

### 2.6. Verification and Validation

After much of my project was done, I wanted to create means to ensure that my models were valid and meaningful.

The first and simplest way I did so was with simple hand calculations. These calculations could give me a rough estimate of the rocket's trajectory given a set of parameters, allowing me to compare the simulation outputs, ensuring they were not junk.

Next, I wanted an even easier way to verify these simulations, and by using methods that even the user could benefit from. To do this, I added some code which tracked energy conservation throughout the physics simulation. After the simulation was complete, I then graphed these energy values, comparing total energy, potential energy, kinetic energy, and energy loss due to drag. This graph shows that (after motor burnout), the total energy does not change and is therefore conserved. This is a key step to verify my simulation and prove that it is logical and obeys simple physical laws.

### 2.7. Final Adjustments & Polishing

It was important at the end of my project and validation, to work through the code again, polishing it up. I added comments where needed to allow users to understand the code logic. I cleaned up the user interface, ensuring proper and consistent spacing and spelling, and making it as easy as possible to use.

In my final look through, I realized my chute deployment model was quite inaccurate. I originally modeled the parachute to instantly deploy, meaning at one time-step it created no drag, then at the next it provided maximum drag. Although maybe optimal, this is unrealistic with how parachutes deploy. They often take at least a second to fully "catch", and I wanted to model this. To do so, I created a simple exponential model which affected the $C_d$ at deployment, meaning at deployment it starts at 0, and after about 1.5-2 seconds, would be at maximum value, meaning the parachute was fully deployed. This shows in the altitude plot, where at deployment, the rocket gradually changes velocity, rather than creating an instantaneous change in acceleration, creating a corner.

Lastly, I wanted means for the user to be able to export their simulation data, and complete whatever further analysis they need in other tools like MATLAB or even their own Python scripts. So, I created a simple file output function, which would write all the rocket and simulation's parameters and data from the simulation to a file. I chose to write to .txt files, as these would allow for easy formatting, allowing the file to include both the simulation data but also simulation parameters. The user is asked if they would like this file to be created, and what they would like to name, and is then saved to their machine.

## 3. Results

### 3.1. Overview of the Final Project

This program is a 1-DOF simulation tool to analyze the flight of a rocket. It takes parameters from the user to model a rocket and its vertical flight that the user has designed. It presents the user with multiple graphs, including those tracking altitude and velocity, as well as the breakdown of energy throughout the flight. It then gives the users outputs about the rocket's simulated flight and offers to create a file the user can save to store all the data and these outputs.

The simulation uses real physics modeling, accounting for gravity, changes in the atmospheric conditions, mass change, and drag. The software even allows the user to change the time-step between calculations, allowing them to control the degree of accuracy and the speed of the simulation.

### 3.2. Configurable Inputs

The simulation tool allows the user to control many different inputs into their simulation.

The inputs related to the rocket itself include: the mass of the rocket itself (excluding the motor's mass), their choice of a motor, the rocket's (constant) coefficient of drag, and the cross-sectional area of the rocket.

There are some other inputs, which control the mathematical modeling and control systems, including the height the chute will deploy, the size (area) of the chute itself, and the time-step the user would like to use. For the time-step, smaller is better (the preset rockets use a standard time-step of .001 s).

With these sets of customizable parameters, it allows the user to control many different inputs to the simulation and characteristics of their rocket itself.

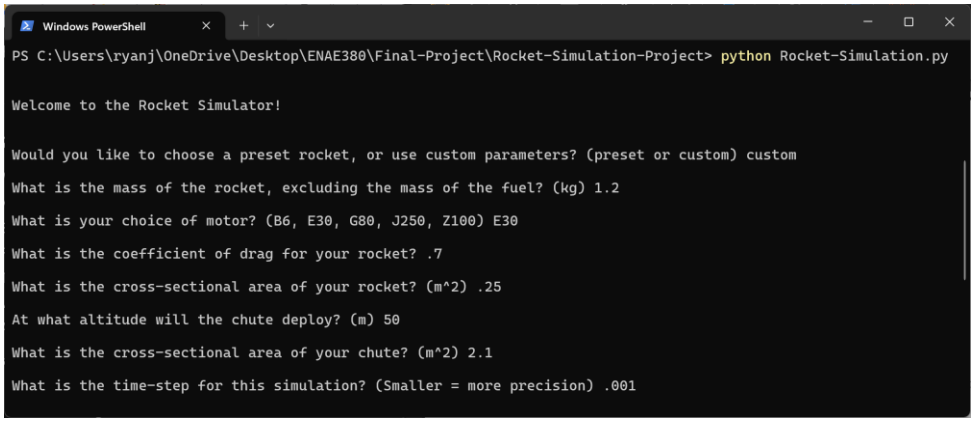


*Figure 1: Example of all user inputs into the simulation.*

### 3.3. Simulation Outputs

When the simulation begins, the program will show an animated graph, which updates the altitude of the rocket in almost real-time. This allows the user to visualize the rocket in motion better than just looking at a static graph. The speed of the graph processing will depend on processor speed and the user's chosen time-step.

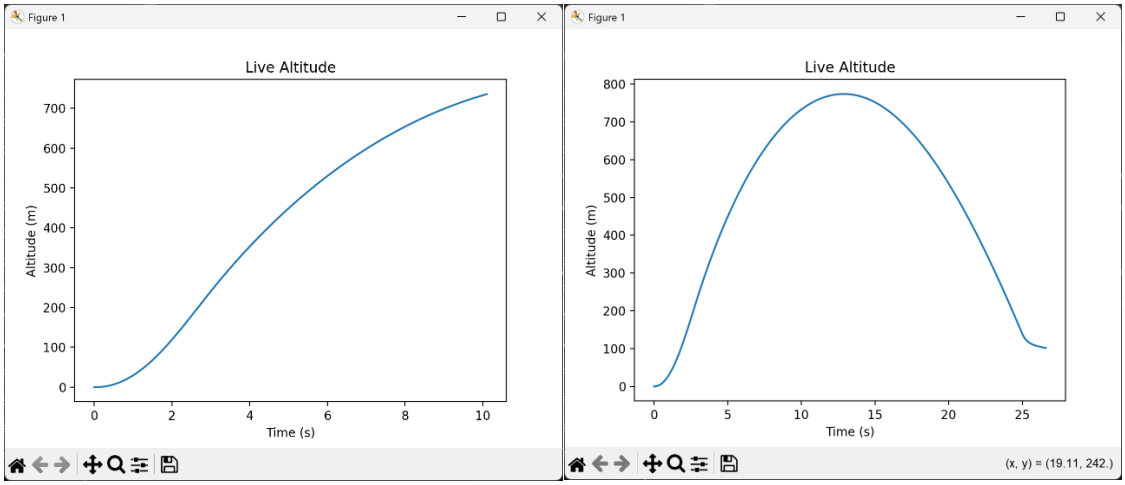


*Figure 2: Screenshots from the animated altitude graph.*

Once the simulation is complete, the program will present a few more graphs to the user. These graphs include a static altitude graph, which is easier to interact with than the animated one, a velocity graph, and two graphs breaking down the system's energy over the period of the flight. These graphs can easily be saved by the user and used for their own interpretation and understanding.

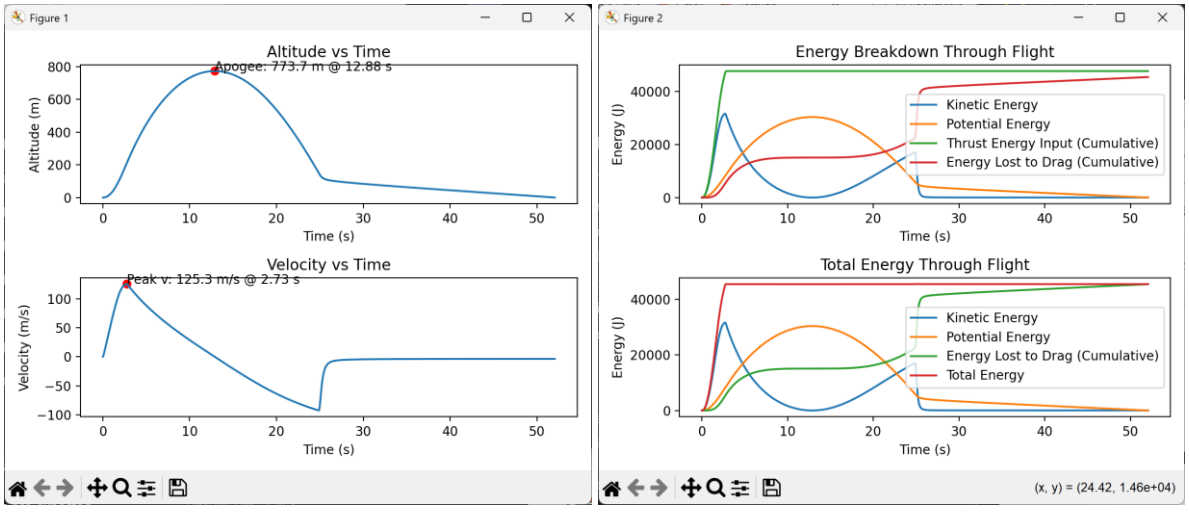


*Figure 3: The graphs presented to user after the simulation is complete. Shows altitude, velocity, and energy breakdowns.*

Additionally, a message will be displayed, making the user aware the simulation has been completed successfully. Next, the program will output some values regarding the simulation. Some of these values include coast time, maximum velocity, and flight apogee.

These values can be helpful for a quick understanding of your rocket's flight but also can be used for further analysis or comparison.


*Figure 4: An example output of all the flight metrics from the simulation.*

Once the user is finished with the graphs and output metrics, the program will ask the user if they would like to save their simulation results to a file. The results will be saved in a .txt file, containing all the input parameters, the output metrics, and the data for position, velocity, and acceleration at each time-step in the simulation.

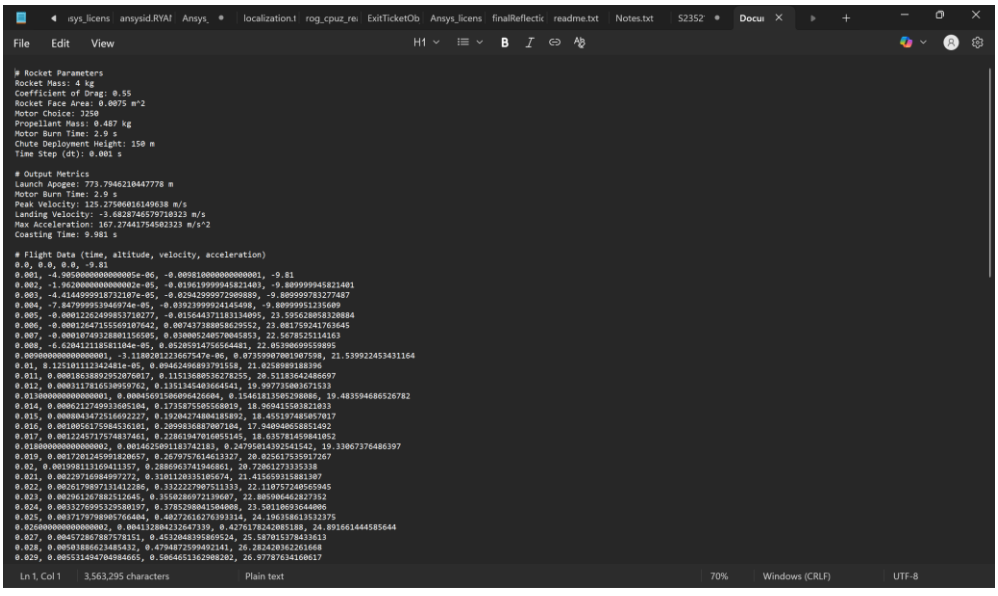


*Figure 5: An example of the saved data file for the user.*

# 4. Concepts Learned

## 4.1. Concepts Learned from Class

One major programming concept I used from class is python classes. My program contains two different classes, a motor class, and a rocket class. These classes are crucial in the effectiveness and efficiency of my simulation, as they both play a key role in the setup. The motor class contains thrust curves and attributes of various motors, as well as a method to calculate the thrust created from the motor at a given time, using interpolation and the associated thrust curve. My rocket class stores all the attributes of the rocket the user is simulating, as well as a few methods for calculating thrust and drag. These classes are crucial in my simulation, as they keep everything nicely organized within the code, and allow for quick and simple initialization of the rocket and its motor during each simulation.

Another concept I applied from class is using the Matplotlib library for plotting. My project creates multiple graphs for the user, and I was able to apply my knowledge of the library I learned in class and lab to easily create these graphs, and present information in the best way possible to the user.

A third major skill/concept I applied from class is writing data to files from python. In class, I learned how to read and write files, and I was able to use this knowledge to create a function which writes all the simulation's data to a file for the user to save. This was an important skill, as this file saving feature allows the user to complete any further analysis or comparison using external tools, which my program does not directly support/implement.

## 4.2. Concepts Learned from Research

One major concept which I learned from research was basic rocket flight physics. Coming into the project, I needed to ensure I had a clear understanding of all the physics which played a part in a rocket's behavior, so I could accurately model all of this in my simulation. I learned about drag, rocket thrust and propulsion, and even a bit about parachutes and their deployment.

Another concept which I learned a lot about from research was about creating optimal simulations. I had to learn about integration methods, whether discrete or continuous, and how to choose the most optimal time-step for discrete integration methods (which I eventually chose). I learned a lot about modeling the atmosphere, in a complex yet computationally affordable way. I had to learn so much about simulation, and especially the core concepts of how they work, to create the most effective and accurate 1-DOF rocket simulation that I could.

### 4.3. New Technical Skills Developed During Implementation

One major programming skill I learned from this project is formatting these larger software programs. I learned a lot about effectively using different functions, and class, to best organize your software. I learned where and how I should create classes, write methods within those classes, and when to create other functions to simplify your code's main loop. Proper formatting can help make code more readable and easier to debug.

Another skill I developed was handling external data files in code, specifically with the thrust curves for each motor. Although I was not directly importing these files, I had to learn how to best store this data, to be more easily accessible for use by the rest of the code. I learned about using dictionaries within a physics situation, and even how to use interpolation from NumPy to better use this data.

Lastly, I learned a lot about validating code output. Within my physics simulation, there could easily be an error with my equations, and even small inaccuracies can multiply over these simulations containing tens of thousands of steps. It was important that I could find methods to double-check my work, and ensure that my simulation was making sense, and following physical laws, besides just looking for any outputs that seemed unrealistic. I was able to do so by creating a conservation of energy model, allowing the user to track energy over the period of the flight, showing it was conserved.

### 4.4. Modeling Insights

One major idea I learned about simulation and modeling, is that simulations are just approximations. While they can still be very accurate, they are not always perfect. Because of this, it is important to consider how you create your model. Finding methods that can accurately model your system while maintaining efficiency is important, and therefore it is crucial to consider different modeling techniques. Within my project, one area I applied this idea was choosing my integration method. I chose to use explicit Euler approximation due to its accuracy (at lower time-step values), while maintaining efficiency within the simulation.

Another idea I learned was how small assumptions can significantly shift results of a simulation. In my simulation, I made many assumptions, one being that the air is incompressible. While this is a valid assumption at lower speeds, once you reach higher mach numbers, this estimation starts to break down. Because of this, if you were to compare my model to another simulation tool which accounted for air's compressibility effects, you would start to see some significant differences as the rocket's mach number increased. This project taught me a lot about how it is important to consider the

assumption you are making within a model and trying to understand how that can affect the accuracy of your simulation.

One last modeling idea I learned was the importance of verifying results. I initially verified my model by just looking at the outputs of a couple different simulations and compared that with simple hand calculations. I compared these values, and ensured there were no crazy differences, and thought that would be enough. But as I decided to implement a more mathematical and scientific approach to verification, creating an energy through flight graph, I realized there was a problem with my model. The total energy was slightly decreasing after burnout, and this would disobey conservation of energy. I then looked back at my model and equations and realized there was a small indexing problem within two of my equations. This slightly changes the values of velocity and altitude compared to what they should be, but over the period of a longer flight, these effects were really magnified. This taught me the importance of using scientific approaches to verifying results, as this was an error I would not have caught by just looking at the results.

## 5. Retrospective

### 5.1. What I Would Have Done Differently

Although I am proud of my work, and happy with how my simulation came out, there are still some things within my program I wish I did differently.

The first aspect of my project I would have done differently was to eliminate my assumption of incompressible air. Although I originally designed the sim around smaller scale, model rocketry simulations, a lot of the physics are the same in larger scale rockets. Even if it was a simpler model, that just used a few equations the model the compressibility of air, it is something I wish I had included. This would allow the user to better predict the behavior of their rocket, especially for rockets that are flying at larger mach numbers.

Another part of my project which I wish I had done differently, was to use another language. Although python is quite simple, easy to write, and contains thousands of free and easy to use libraries, I wish I had written the project in a more low-level and efficient language like C++. Although there is not a significant difference for smaller simulations, the program would operate much faster if written in a lower-level language and operate more efficiently. Mathematically complex simulations are commonly written in these languages because of their operational speed, and I think writing in a lower-level language would have made my project more realistic in that way.

The last major aspect I wish I had done differently, was to create a more presentable UI. While the terminal commands can be simple and effective, they are not the most effective means for the user to control the simulation. Even adding a simple UI with buttons, and input boxes, would make the simulation much easier to control and more intuitive. When creating a program that a user is interacting with, the UI is sometimes just as important as the back end of the program itself. By adding a simple GUI to the program, I would have made the tool much easier to use.

### 5.2. Improvements I Would Add with More Time

There are a lot of improvements I would like to make, like those highlighted in the previous section, as well as simple feature additions if I had more time.

The first improvement I would have liked to make was to implement multiple integration methods. This would allow both the programmer and the end user to compare different methods and find which are best for the simulation. One method is faster and more accurate for smaller scale simulations, while another is better for larger systems. I think adding a few of these integration methods would make the program more in depth, more accurate, and add another element of customization for the user.

Another feature I would have loved to add would be a comparison mode. This would be a mode where the user could compare multiple different configurations and allow them to look at the differences between each. They could generate plots for each and look at the different output metrics of each. This would add another element of engineering and simulation to the project, as it would allow users to compare rockets and understand how what parameters they change affect the behavior of the system.

One smaller feature I would like to add would be to add horizontal motion within the simulation. I believe making the simulation 2D or even 3D would not be too difficult and would add a major element of realism. I would be able to model the effects of wind and even launch angles, to determine and better understand how these changes affect a rocket's behavior.

The last major feature I would have liked to implement would be a feature to use your own motors. In the current simulation, I have five motors from which you can choose. However, there are plenty of commercial motors out there, that my current simulation simply cannot model. I would love to add a feature where a user can input the burn time of the motor, the motor's mass, and a data file containing the thrust curve of that motor. This would add another element of realism to the simulation and create endless possibilities for the user.