UNIVERSITY of VIRGINIA

SCHOOL of ENGINEERING & APPLIED SCIENCE

Missile Optimization

OPTIMIZING MULTI-STAGE AND COMPARING BOOSTER PERFORMACE

05/05/2025

Missile Booster Code – Annie Bai, Joshua Lange, Rohan Radadiya, Ethan Rodgers, Xavier Turner

Brian Smith – US NAVY

DRAFT REPORT:

Optimizing Multi-Stage Booster Models

Project:

Missile Booster Code

Prepared for:

University of Virginia School of Engineering and Applied Sciences
Stephen Grossen

By:

Annie Bai, Joshua Lange, Rohan Radadiya, Ethan Rogers, Xavier Turner

First Year Engineering Center
Benjamin Goldschneider, Section 4, Group 1

University of Virginia
School of Engineering and Applied Sciences
Thornton Hall
351 McCormick Road
Charlottesville, VA 22904

Table of Contents

List of Figures

List of Tables

List of Acronyms

GMS	Gas Management System
TTB	Time to Burn
VBO	Velocity of Burnout

Executive Summary

The U.S. Navy is facing a major issue when trying to balance missile performance and launch safety. They are currently using a GMS to manage the exhaust of regular two-stage missile hot launches; however, upgrading to more powerful three stage missiles would put more strain on the GMS systems, making them complicated and less safe. Implementing pop-out boosters, rockets with small first stages that fire to get off the ship, would simplify the design of the GMS but would reduce maximum velocity and range. To fully address the differences, the Missile Booster Code project team made a computational model that would quantify the performance differences between optimized three stage missiles with and without pop-out boosters. To properly inform US Navy decision making, a velocity of less than five percent was targeted for the pop-out configuration compared to the conventional.

The solution was a Python-based application that used the Tsiolkovsky rocket equation to calculate the velocity of each stage. This is a hybrid approach that mixes calculus-based optimization and brute force iteration to ensure accuracy and create graphs of the optimization process. The algorithm used calculated the stage specific mass ratios and structural efficiencies using an increment of 0.001 to maximize velocity. Furthermore, a GUI was created in a Python library called DearPyGui, which took in critical variables such as ISP, fuel density, and structural efficiency and displayed the results of the optimized change in velocity as tables and other visualizations that showed the differences between constrained and unconstrained optimizations. Testing was done on the final GUI to ensure validity of results and functionality, with edge-case handling to prevent crashes during the optimization process and surveys to refine the clarity of the GUI and code readability.

The results showed the pop out booster had a velocity loss of 8.4%, which was significantly higher than the target set by the Navy. The traditional missile preferred to have a larger first stage, while the pop-out booster distributed the propellant to later stages. A sociotechnical analysis revealed that pop-out boosters would reduce GMS costs and the need for excessive maintenance, and testing showed a need to clarify graph labels.

Overall, the project evolved through many design phases. Initial calculus-based optimization methods were abandoned for brute-force methods to support graphical outputs. Furthermore, user feedback helped improve the GUI and the final program met client expectations while allowing for future growth such as stage-specific impulse values and trajectory modeling. The tool helps the Navy make decisions between missile efficiency and launch practicality. In the future, there could be more ISP variability per stage to align more to the real world, a transition to a web-based platform for more accessibility, and the addition of data export features. By transparently showing the performance tradeoffs, the solution offers safe and cost-effective missile launches without sacrificing mission readiness.

1. Introduction

1.1 Client Problem

The Navy currently uses GMS to redirect missile exhaust when hot launching from ships. Three stage missiles, when optimized for range, have a large first stage and successively smaller second and third stages, but this large first stage requires a more robust GMS for a safe launch. Alternatively, a restricted first stage, referred to as a pop out booster, which only fires to get the missile off the ship before larger stages fire, could be used to lessen the load on the GMS, but results in a shorter range because of less optimal propellant allocation. The Navy wants to model the performance difference between a traditionally optimized three stage missile and a three-stage missile using a pop-out booster to see if the losses are worth reducing GMS requirements.

1.2 Objectives and Approach

Both missile designs need to be modelled and optimized using code to compute the performance losses a pop-out booster would incur. Using Tsiolkovsky rocket equation to calculate the velocity gain of each stage, the optimization must test the performance of different staging ratios until the optimal solution is found. This optimization is run on an unconstrained missile and on one with a restricted first stage. The two resulting stage configurations and their velocity gains are compared, with a percent loss of performance between the two displayed.

1.3 Report Organization

This report is organized in chronological order, following the project from initial problem research to solution detailing the steps taken along the way. Chapter 2 covers the problem background in a more granular manner, describing the initial client meeting and providing a literature review highlighting gaps in research surrounding pop-out booster efficacy. Chapter 3 then covers initial solution ideation and refinement, discussing three potential solutions and the process used to decide on one. Chapter 4 discusses the initial design, testing and analysis of weak points, and revision to come to a final solution. Chapter 5 then details this final solution’s performance and meeting of client needs. Chapter 6 concludes with recommendations for future work and what steps would need to be taken to reach this further improved solution.

2. Problem Definition and Background

2.1 Client Interview Summary and Customer Discovery

This project began with an interview with Stephen Grossen, who described the problem and solution requirements. At the beginning of the interview, Mr. Grossen explained how the Navy is considering replacing its two stage missiles with more powerful three stage missiles. Mr. Grossen then explained the potential for a pop-out booster, describing how it would decrease performance but also decrease the stress on the ship's gas management system (GMS). He then gave us the expectations of our solution, explaining how he needs a program that optimizes the stage fractions for both missiles and then compares their performance.

He then explained formulas for acquiring the velocity provided by each stage in a multi-stage rocket. The other formula he gave calculated the burn time of each stage, which was necessary for ensuring the pop-out booster only burns for a small amount of time. After explaining the formulas relevant to this project, he gave assumptions our solution was allowed to make. This included assuming the earth is flat, assuming each stage has the same specific impulse and assuming aerodynamics would not directly affect our solution. After explaining the problem and solution requirements, the client began to accept questions. Through the questions of other teams, we learned our solution should output graphs that compare both optimized rockets. He also expected graphs that showed the optimized process, including how velocity changes relative to the mass fractions.

Model of problem

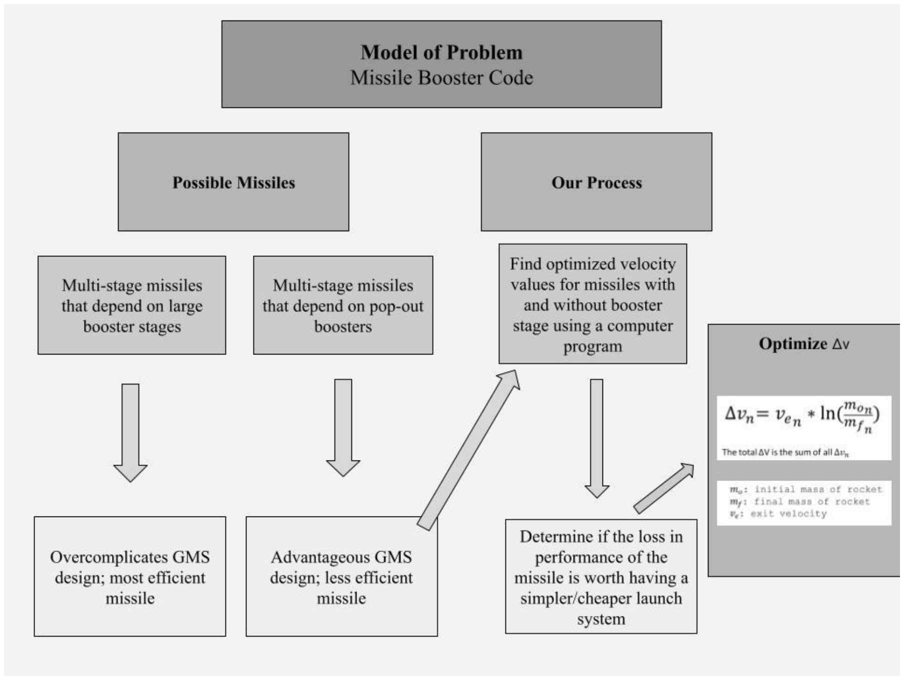

Model of Problem
Missile Booster Code
Possible Missiles
Our Process
Multi-stage missiles that depend on large booster stages
Multi-stage missiles that depend on pop-out boosters
Find optimized velocity values for missiles with and without booster stage using a computer program
Optimize Δv
$\Delta v_n = v_{e_n} * \ln(\frac{m_{o_n}}{m_{f_n}})$
The total ΔV is the sum of all Δv_n
m_o: initial mass of rocket
m_f: final mass of rocket
v_e: exit velocity
Overcomplicates GMS design; most efficient missile
Advantageous GMS design; less efficient missile
Determine if the loss in performance of the missile is worth having a simpler/cheaper launch system

Figure 1. Model of Problem and Solution Requirements

2.2 Literature Review

2.2.1 Introduction

Missiles are an integral part of the U.S. Navy's arsenal. Most of these missiles consist of multiple stages, which are released as they burn out. Optimizing the size of each stage of a multi-stage rocket is an ongoing challenge in the space of aerospace engineering. The distribution of mass between rockets stages significantly impacts a rocket's performance, fuel efficiency, and payload capacity. As rockets become larger and more complex, optimization methods must be updated to ensure they accurately represent modern technology.

This literature review begins by discussing current missile technology, including the advantages and disadvantages of a pop-out booster. It then discusses the current work that has been done involving theoretical missile optimization. Lastly, it discusses the limitations and gaps in the current work done on missile optimization.

2.2.2 Themes

The first theme in the research was that the current missiles either have two stages or require a complicated launcher. Currently, most multistage rockets used by the U.S. consist of two stages. Missiles such as the SM-2 use two stages, allowing it to achieve ranges of up to 100 miles ("SM-3 Interceptor," 2021). Newer missiles, such as the Trident II, consist of three stages ("Trident II (D5) Missile," 2021). However, these missiles require a much more complicated gas management system (GMS), which makes it extremely difficult for them to be launched from the deck of a ship. During a missile launch, the GMS is responsible for dispersing the exhaust the missile releases when it exits the launch tube (Sinha and Chakraborty, 2014). GMS are also put under a lot of strain during an abort launch (Black et al, 2017). It is integral to alleviate pressure from the GMS to both simplify its design and ensure aborted launches can be successfully performed when needed.

To decrease the pressure on the GMS, rocket research has considered using a pop-out booster (NASA., n.d.). A pop-out booster replaces the bottom stage of the rocket with a smaller stage that is only used to get the rocket farther from the ship before firing the larger boosters. While this does simplify the GMS needed on the ship, it decreases efficiency. Since most optimized rockets have the bottom stage as the largest for efficiency and structural reasons, pop-out boosters sacrifice range and velocity for ease of launch.

The second theme that we found was optimizing rockets requires considering a wide range of factors. Rocket optimization is a multidisciplinary endeavor which seeks some "best" solution with a given set of constraints. These constraints may be cost, ease of production, reliability or any other relevant aspects of design and production. Some goals may be at odds with each other, but ultimately a final solution with some consideration of all concerns must be reached

Multiple multidisciplinary methods are used in rocket optimization, with the multidisciplinary feasible method being commonly used. These methods however, by attempting to take into account multiple different aspects of the design at once, can require robust starting knowledge of what a final design may look like to converge to a true best solution (Morgado et al., 2022). While good designs can be developed with a bit of variance, trajectory optimizations require dynamic equations to be solved with numerical methods to find the best path. This path can serve as an initial value for the design to then be built around solving. The trajectory optimization solution is not concrete and can be solved in many ways, with different approaches used for varying applications, but some simpler methods have been developed to give a reasonable first guess which can be further refined by more specialized algorithms (Pontani & Teofilatto, 2014). Methods have also been developed that couple the design and trajectory problem in an iterative manner, with rocket constraints fed into a trajectory optimization method which then feeds back tighter constraints on the design until the process converges on a final best solution (Calabro et al., 2002). While the problems of rocket design and trajectory optimization can and are often tackled separately with one serving to provide the constraints for the other, newer methods have been produced coupling these processes to find an even more optimal solution.

The third theme was that rocket optimization can be approached using calculus. A lot of sources focused on the mathematical approach of optimization when looking at the mass ratio of three stage booster rockets. Specifically, calculus-based optimization methods are used to minimize mass while achieving velocity targets. One approach focuses on the Lagrange multiplier technique, which attempts to minimize mass enough to maximize velocity to reach the target (Rocket Sim, 2017). Since this work uses calculus, it is extremely efficient. This allows it to be applied to a rocket with any number of stages. For n-stage rockets, the mass ratio MR (initial mass to burnout mass) of each stage is derived by solving a system of equations balancing structural coefficients (ϵ) and payload ratios(λ) can be described as:

$$MR = \frac{1+\lambda}{\epsilon+\lambda}$$

Figure 2. Mass Ratio Formula from the Glenn Research Center at NASA

However, using a calculus-based approach is one that can be used if the researcher has the luxury of knowing what formulas can be applied to that rocket. It must also be possible to calculate functions for the derivatives. However, depending on the model used to represent a rocket, the formulas may not be differentiable. In this case, a brute force approach could be useful as it helps generalize popular formulas so that they can be used to optimize the values for a lesser-known rocket (Eldred and Gordon 1976). Known as the rapid analytical method, the brute force method takes given values and calculates the mass ratio and maximum velocity through the formulae, then increments the input variables by 0.001 and does the calculations again. The program will continue to do this until it runs out of values to calculate within the range it is given and can be programmed to display a results table and line graph of the optimized values (Patel et al. 2006).

2.2.3 Gaps

While the challenge of booster rocket optimization has been prevalent for a long time, there are still some significant gaps that remain in research. For a lot of projects and situations that have a time-constraint, there is a lack of comprehensive models which are able to optimize both mass ratios and rocket trajectories at the same time (Rey et al. 2022). Furthermore, a lot of theoretical mass ratios end up exceeding structural limitations, and the lack of research on new material might allow for more aggressive optimizations. Current models are also not able to adapt their algorithms to adjust mass ratios and booster trajectory parameters in real-time conditions. Ultimately, the gaps in research arise not in the ability of models being able to do the optimization, but it is more about the models being able to adapt to changes in parameters and work under a time crunch.

There is also a gap in research for pop-out booster optimization. Since a rocket with larger stages at the bottom is known to be the most efficient (Aguado-Agelet et al., 2022), it is reasonable to assume a rocket with a booster stage would be less efficient. Despite this, exactly how much range and velocity would be lost should be studied to determine if the drawbacks are worth the increased launch simplicity. It is integral that the optimizations for a pop-out booster are compared to the optimizations for an unconstrained three stage rocket to decide if the performance increase is worth the increased resources required to make a complicated GMS.

2.2.4 Closing

While a lot of work has been done on rocket optimization, there is still the need to consider the difficulty of launching. No matter how effective a missile is, if it requires a launcher that is unrealistic it will be unusable. Because maximizing speed and range is still important, ensuring that an optimization method exists for a pop-out booster helps ensure the increased effectiveness of the missile can be applied to a variety of launchers and applications. Overall, the research shows that in both practice and theory, the pop-out booster has been underexplored and under researched.

2.3 Problem Definition

Traditionally, the U.S. Navy utilizes a GMS (Gas Management System) to hot launch missiles from shipboard launchers. The GMS redirects the missile blast and exhaust below deck to the outside and must be able to do so safely during launch and in case of a restrained fire condition (Sinha and Chakraborty, 2014). Currently, multi-stage missiles with large booster stages are the most efficient missile performance based on basic missile performance. As a result of having large booster stages, the large amount of energy that it generates greatly complicates the design of the GMS, which is crucial for ensuring safety. While less efficient compared to large booster stages, shoot-out boosters, also known as pop-out boosters, are an alternative and can be advantageous to GMS design.

Missile usage is an integral part of how the U.S. Navy carry out their mission of protecting America at sea, with purposes including nuclear deterrence, fleet air area defense, and ship-self-defense (“Trident II (D5) Missile,” 2021; “SM-3 Interceptor,” 2021). Currently, multi-stage boosters with large booster stages and pop-out boosters are the most viable missile options for the Navy, and there does not exist a way to optimize both missile efficiency and the GMS design for the Navy to choose which booster option to use. If this problem is not solved, there are serious implications regarding the intended impacts of missiles and safety. Prioritization of the design of the GMS by using the pop-out booster will result in lower missile efficiency, which can diminish the intended impacts of missiles. As the purpose of missiles are largely to protect the American people through means such as nuclear deterrence and both sea and air defense, if the missile is not able to reach its intended impact due to low efficiency, there could be severe implications for the safety of the American people. In the most extreme cases, such as the inability to successfully deter nuclear weapons, those all around the globe could be affected on a catastrophic scale in the event of a nuclear war. However, there are also serious effects if missile efficiency is greatly prioritized over the functionality of the GMS. A complication of the design of the GMS in order to accommodate the large booster stages of a multi-stage missile can increase the risk of system issues arising in the GMS, which can put members of the Navy in direct danger if the GMS is not able to safely direct missile exhaust or function in case of a restrained fire condition. Ideally, there would be a way to simulate both the performance of an optimized multi-stage missile with both a large booster stage and a pop-out booster in order for the Navy to make a calculated decision when choosing one missile design over the other.

There are various constraints and criteria guiding the design and functionality of a possible solution. One main constraint regards the form of our solution and dictates that our solution must use a computer program whether that be through executable code or a common existing software such as Excel or Matlab. The major constraints involve the set of measurements provided by our clients that we must use in solving the problem. Namely, these constraints include that the missile diameter must be 1 meter, the propulsion stack length must be 10 meters, the payload must be 250 kg, there should be three booster stages, and the earth should be considered flat during our optimization process. Additional measurement constraints include that the first stage booster must burn for 10 seconds, and that the impulse must remain constant. Other constraints set by the Engineering Foundations II Course include time and money, as we are restricted to a time period of one semester and are allocated a budget of $500 dollars. In terms of criteria, in order to increase the effectiveness of our solution, it should include some kind of visual representation, such as in the form of a curve or plot, depicting our optimized result. Our solution should also aim for 250 seconds of total burn, and both the code and results should be easily understandable to an engineer.

2.4 Summary of sociotechnical considerations in problem

As a result of the development of a pop-out booster stage with limited range and impulse loss (end design goal), problems arise regarding societal and environmental impact, as well as the overall cost to develop and manufacture an effective solution. Key actors and individuals who would benefit from the final solution will be affected the most when sociotechnical issues are overlooked. As a result, it is necessary to address all possible problems that could be present within an effective solution, as well as the possible ways to mitigate such effects.

Missiles, and the improvements of missiles over time, have become heavily integrated in military strategy and funding. Advancements in missile research and integration of missiles into threats outside of the United States has had large impacts on the views of offensive missile strategy (particularly in regard to intercontinental ballistic missiles [ICBM]). Although nuclear and biological payload missile threats are necessary to address regarding international defense, missiles in general pose a serious threat, as missiles allow for precise targeting of defense and offensive structures that can be deployed with limited ground involvement. As Bowen (2001) noted, missiles have historically been a hot topic regarding national defense, and the threat of the missiles has challenged international relations and policies. As a result, future research in improving missiles has the ability to increase tensions. The Missile Booster Code research would likely have little impact on increasing such tensions given the nature of the project, though. Development of pop-out booster code would likely only change the need for a complex GMS rather than redevelop ICBMs. That said, given the historical views on missiles, discretion must be taken with any decision, and all possible considerations should be noted.

Based on the goal of the missile solution, maintaining a missile silo through a pop out booster would have economic benefits, both from limiting the necessary repairs in response to long first stage burns times without a pop out booster, as well as the implementation of less complicated gas management system designs. Regarding foreign adversaries, the development of an improved mass ratio for missiles containing pop-out boosters would likely not shift the global scale much if at all. Largely, the change economically would not appease any threats if never developed, and larger issues are still present militarily and diplomatically. All issues must be noted, nonetheless.

The targeted solution would have benefits for the United States, largely with the stated reasons of economic benefits and limiting a complicated gas management system. As stated by Mr. Grossen during the interview, a range and impulse loss of 5% would be a successful design, as a long burn within a silo causes damage that must be addressed over time. The damage that longer burns cause leads to larger costs associated with missile-based warfare and testing. As such, the US military, and the US Navy primarily, would likely receive the most benefit from such a solution

While economic benefits should be considered in regard to stakeholders and target audiences, the adverse environmental effects of missiles should also be considered. As noted by Panda et al. (2014), solid propellant missiles cause adverse environmental problems that can release toxic emissions and particulate matter into the atmosphere. From an environmental perspective, several key particle and gas emissions pose a threat to health. A few of the emissions include - NOX emissions, respirable particulate matter (RPM), and carbon dioxide. As noted in the study, RPM was locally elevated for an hour post test firing but would taper off. While missiles are incomparable to worldwide emissions (largely due to the sheer size and types of present), any emissions from test firings and impacts that are released into the atmosphere must be addressed as an environmental concern. NOX emissions, as an example, are key contributors to smog and toxic air pollution.

While issues regarding the environmental effects are present for missiles, the overall solution should not influence the environmental impact that missiles already have. As a result, the impact of the missile solutions should be made separate from the effects of missiles in general. That said, with any solution, the overall design must be tested to some capacity, so the effects of the design implementation would result in atmospheric particulate matter expulsion if deemed successful. To mitigate such effects, the best course of action would be to create a proper system initially that would limit the amount of testing over time (validating a solution rather than improving one).

Aside from environmental concerns, ethical concerns must be addressed, too. The solution would contribute to the development of an efficient pop-out booster, which would aid in US military practices. Ethically, missile designs can be concerning as missiles contribute to defense/offense structure damage and loss of human lives; however, the solution is not directly what would impact such events. The solution looks more towards how to fix an issue economically and structurally within the missile silos and missiles, and the effects of missile usage is more so in regard to the ongoing needs of international warfare.

Ultimately, to create a proper solution, the initial solution must be sound enough to limit excessive testing and effective enough to be adopted by supporting parties.

3. Initial Solution Ideation

3.1 Development of Multiple Designs

The primary difference between the team's prospective designs had to do with the algorithmic approach behind each idea. The first design the team created was a trial-and-error optimization which is visualized in Figure 3. The trial-and-error optimization works by essentially iterating through various mass ratios being input into our velocity equation. As one constraint was the mass of the rocket, this means that the mass of the rocket would stay constant and that there are a finite number of solutions that could be generated given a certain step-size. Once the final iteration is complete, the mass ratio that produces the greatest velocity would be considered approximately the best possible optimization. The pros of this algorithm are that it is compatible with any language of the team's choice, namely Python, Java, and Matlab. Furthermore, the algorithm is easy to implement and has straight-forward reasoning behind it. However, cons of trial-and-error optimization include that it is computationally inefficient. Iterating through as many mass ratios as possible may end up requiring a significant amount of time computationally, especially if the number of tests is relatively large. Another con of the trial-and-error optimization is that because it is essentially impossible to test every possible mass ratio, if the step-size chosen is too large, then it is possible that the algorithm will actually skip over the most optimal solution.

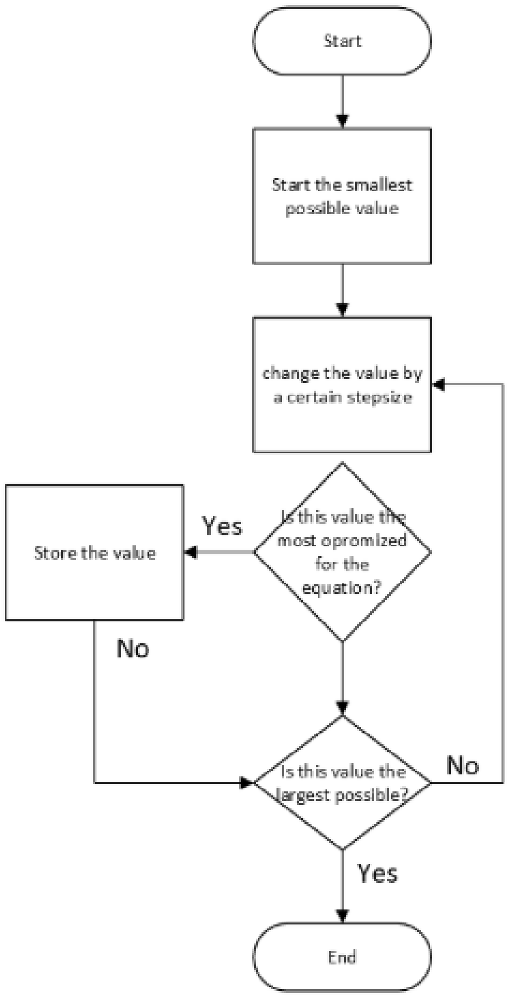

Start
Start the smallest possible value
change the value by a certain stepsize
Is this value the most opromized for the equation?
Yes
Store the value
No
Is this value the largest possible?
No
Yes
End

Figure 3. Flow chart visualization of trial-and-error optimization

The second design the team created was a calculus-based optimization algorithm which is visualized in Figure 4. The idea behind this design is that the program would utilize derivatives and principles of calculus in order to calculate the exact point mass ratio where the velocity is maximized. Then, after the mass ratio calculates the max velocity, it would simply be returned as the most optimal mass ratio. The pros of this algorithm include that it is compatible with the team's language of choice, whether that end up being Python, Java, and Matlab, due to the surplus of math libraries that can be imported with each language to help solve the optimization problem. Calculus-based-optimization is also more computationally efficient compared to trial-and-error optimization, as it does not require any iterations. However, despite having access to several math libraries that can help perform operations in the program, it is still difficult to implement. This is because, while the goal is to optimize the velocity, the velocity equation exists in the context of several other rocket equations that must be considered in the optimization process. That means that merely maximizing the velocity equation is not enough to consider the implicit processes of the optimization.

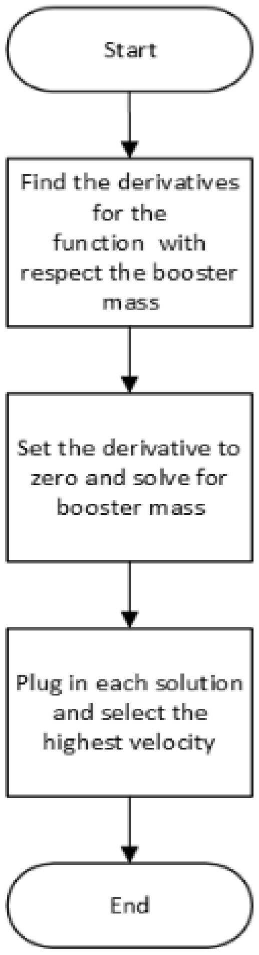

Start
Find the derivatives for the function with respect the booster mass
Set the derivative to zero and solve for booster mass
Plug in each solution and select the highest velocity
End

Figure 4. Flow chart visualization of calculus-based optimization program

The third prospective design is an evolutionary algorithm which is visualized in Figure 5 In general, evolutionary algorithms, specifically genetic algorithms, consist of random searches which provide data to help direct the search into areas of higher performance in the range of solutions. In the context of optimizing the velocity, the evolutionary algorithm would simulate the velocity function as a population and continue making changes to the variables each "generation", making sure to preserve the best performing values to approximate an optimized mass ratio. The main pro of this design is that it is the most expandable out of the three solution ideas and can adapt easily to more complex scenarios. Another advantage is that it can be implemented in Python, which is a language that most of the team already has experience writing in. However, in the case of the problem, a complex solution does not necessarily indicate a good one. The cons of this design are largely that it is both difficult to implement and computationally intensive. As the team is generally unfamiliar with the implementation of evolutionary algorithms, time that could be spent building and testing code will instead be used learning and researching this complex algorithm. Furthermore, as several variables in the set of functions involved are constrained by the client and remain constant, diminishing the need for the algorithm 's main benefit is its expandability given different conditions.

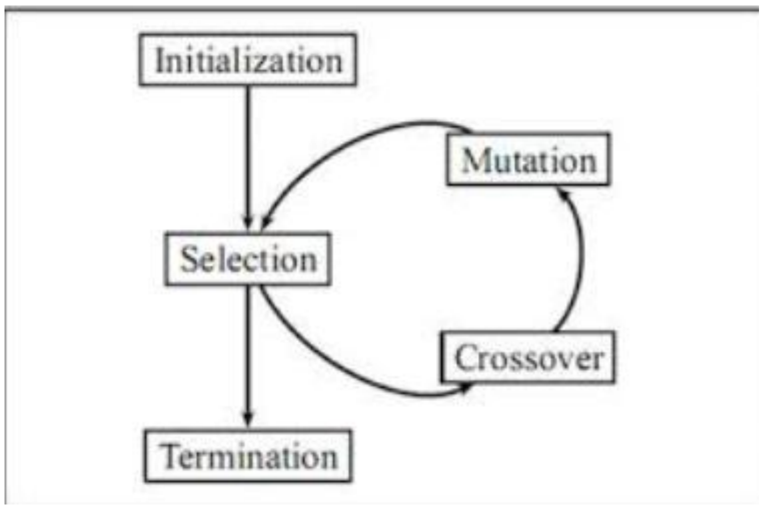


Figure 5: Visualization of general genetic algorithm logic

3.2 Design Decision Discussion

The team favored a calculus-based approach before the roundtable, because of its perceived ease of implementation and speed once the initial calculations were made by hand to find partial derivatives and boundaries for the program to work with. Going into the round table the team was more focused on which language the method should be implemented in, but feedback was given that pointed us towards using Python instead of the other prospective choices. The calculus-based approach was also favored by peers giving feedback, cementing the idea as the final solution.

Coming out of the roundtable, the team decided to use Python as the language and combine the calculus and trial and error approaches into one. This meant that the solution consisted of the best of both worlds, with the calculus done first to give the expected result and a trial-and-error method run after checking around the result to confirm its validity. Python was chosen as most of the group had experience in it whereas MATLAB, a language recommended by the client and standard for similar problems, was unknown to all but one member. Learning the nuances of MATLAB before being able to implement a solution would take away the time allowed and therefore reduce the functionality of the completed project. Python has many math libraries allowing it to perform the operations needed for the problem and has GUI libraries allowing for an interface to be implemented for an easier user experience.

Decision Matrix

A combination of heuristic thinking and a decision matrix was used to choose the final solution to be sent to the client. Some loose rules guiding the decision were that the method must be realistically implementable in the given time and not introduce unneeded complexity to the problem. These rules helped rule out the evolutionary algorithm, because the team does not have much experience with learning models and the concept was more suited to a more complex problem than the given problem, which only operates on a few variables to come to a result. With the evolutionary model ruled out, a decision matrix was used to decide between the trial and error and calculus-based approach. The decision matrix used weighted values of ease of ideation and implementation, accuracy, and program speed to come to a final solution. Approaches were rated from 1 to 5 for each aspect, and each aspect had a weight given from 1 to 5. The calculus-based approach scored the highest on the matrix, lagging behind the trial-and-error method only in ease of ideation and implementation because it requires some calculation by hand to come to the final equations to be implemented in the program. The approach wins in accuracy as it finds the theoretical maximum and will result in a program with lower time complexity as the iterations associated with a trial-and-error method are not present. The approach also makes use of a trial-and-error confirmation method after the result is given to verify that the method returned the correct value. This eliminates the cost of iterations as the trial and error has a smaller area to check around the given result and not the whole set of possible points initially given.

	Trial and Error	Calculus Based	Evolutionary
Ease of implementation (*4)	5	4	2
Accuracy (*5)	4	5	5
Program Speed (*2)	2	4	3
Total	44	49	39

Table 1. Decision matrix for solution selection

Sociotechnical Considerations in Selected Design

Sociotechnical considerations were applied to the solution decision process. The team identified that economic actors were the main consideration our solution addressed, so the solutions were viewed through the lens of which would provide the most accurate results for economic considerations. The solution chosen addresses this by returning the theoretically most accurate results, informing the decision between choosing to use a pop out booster or install and maintain a gas management system for use with missiles with longer burning first stages. In addition, it is necessary to include the proper implementation of a calculus approach (for the solution) for the most accurate and fastest code. As a result, future testing would have a decreased need for funding as a proper solution. With simple testing of points, there is more testing necessity as the solution is not theoretically confirmed.

From a humanitarian perspective, the solution also avoids the issue of ethical considerations in regard to the influence of our code on humans. Given the nature of the code, an improved pop-out booster would not provide any additional issues regarding international conflicts, and, instead, the code would provide the opportunity to increase the longevity of missile silos to preserve resources and money.

3.3 Detailed description of design

After discussing with our client, there were a couple of key ideas that must be considered when developing the application. This first is ensuring it is easy to use. The original plan was to have the user enter certain variables, such as ISP and payload. But after discussing with the client, the software will now use default values that can be changed inside of a submenu. The second key consideration is the readability of the data. Along with the values found from the optimization, the client also wants graphs showing the optimization process.

The key idea our team decided upon was ensuring the library we used for development was easy to use. It was also important that this library supported variable input and had many different graphing methods. It was also important that our code could be deployed in a common format, such as a website or exe file.

In addition, sociotechnical considerations were also made.

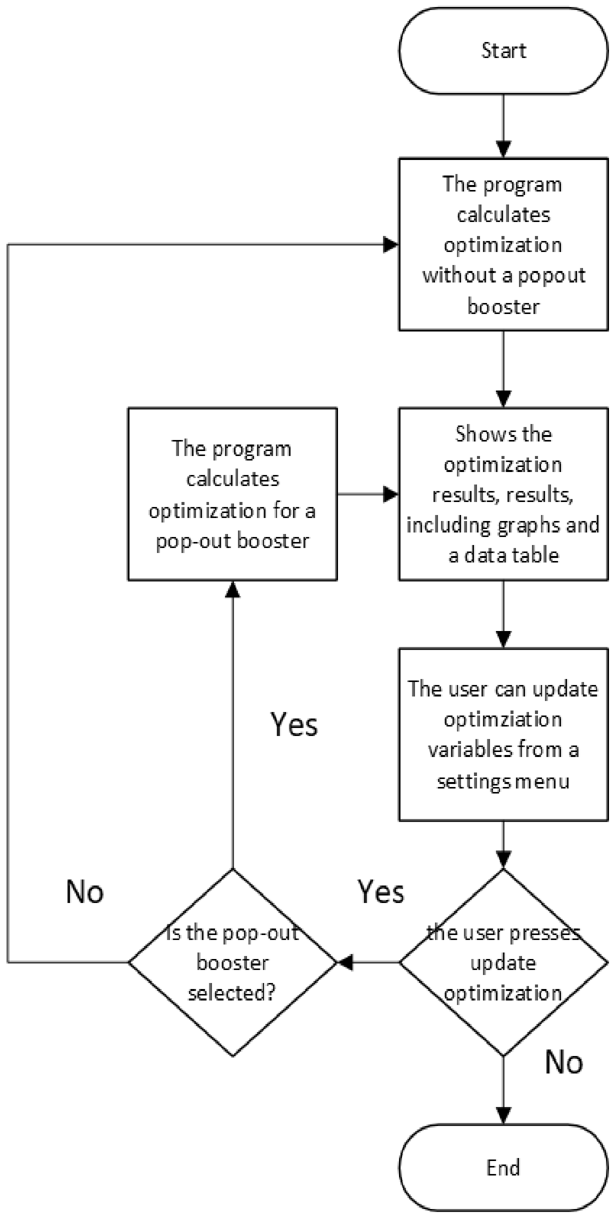
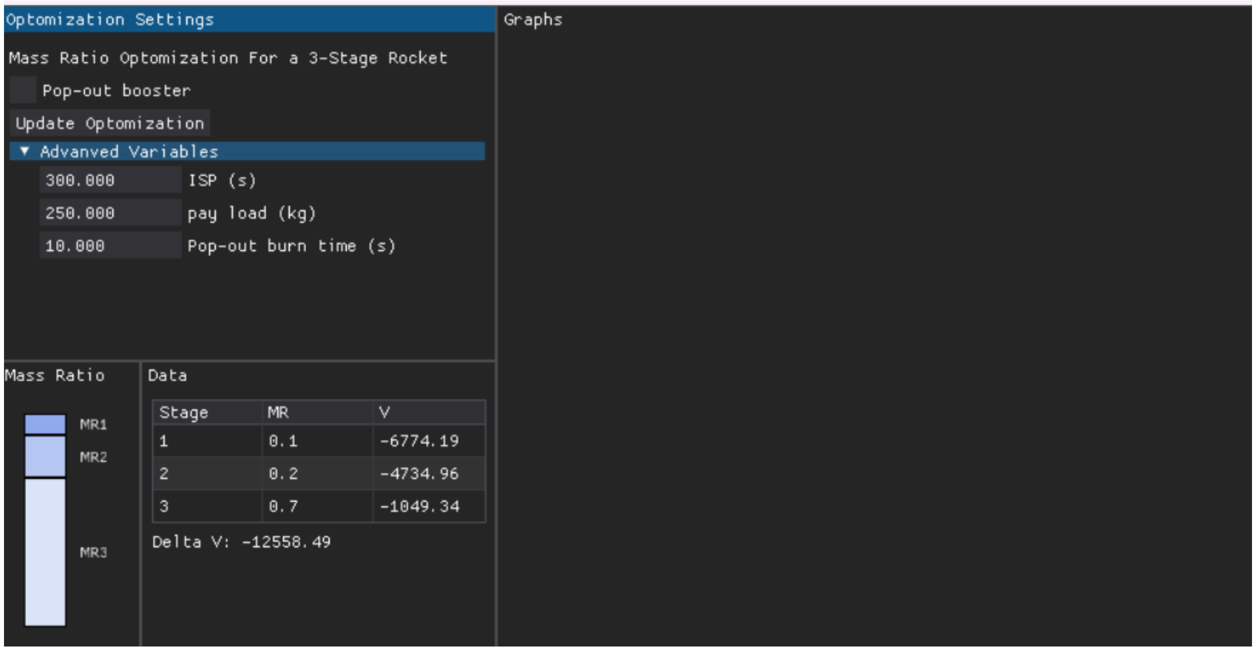

Start
The program calculates optimization without a popout booster
Shows the optimization results, results, including graphs and a data table
The program calculates optimization for a pop-out booster
The user can update optimziation variables from a settings menu
the user presses update optimization
Is the pop-out booster selected?
Yes
No
Yes
No
End

Figure 6. Flowchart of porposed program functionality

Figure 6 shows how a user would interact with the program. The largest consideration is that the program works even without user intervention. This is done by showing optimization using default values on startup. This ensures that even someone with zero technical expertise can use our program. Despite its simplicity, it is important that this software offers some level of customizability. However, the customizability must not come at the cost of usability.

The software will allow the user to recalculate the optimization as many times as they would like. If given enough time, we would also like to give the user the ability to save the graphs as a png or jpg file. This would allow the graphs to be saved since all custom settings are reset once the program is closed.



300.000	ISP (s)
250.000	pay load (kg)
10.000	Pop-out burn time (s)

Stage	MR	V
1	0.1	-6774.19
2	0.2	-4734.96
3	0.7	-1049.34

Delta V: -12558.49

Figure 7. First iteration of the GUI

Figure 7 shows the incomplete UI mockup. Since the optimization has not been completed, the values in the data table and size of the mass ratios were chosen arbitrarily. There are four sections. The first section is the optimization settings. This has a check box which allows the user to decide whether this optimization will be constrained to a pop-out booster. Since the program is limited in space, the best solution was to allow the user to only calculate one optimization at a time. Since the data can be exported, it would be extremely easy for the user to compare the values of both optimizations. This allows users to compare results while also giving them the ability to focus on the design for one specific rocket. This is also more efficient, since if the user only wants to view one rocket the program only must calculate one optimization.

Below the checkbox, there is a button which recalculates the optimization, taking into account any variables the user changed. There is also a submenu in this section called Advanced Variables. This menu is defaulted as hidden and can be accessed by pressing an arrow to reveal the input fields. This option is hidden to ensure the user understands these variables can be left unchanged and that the default values will provide an optimization that fits the problem's constraints. The variables that the user can change will be the ISP, payload and pop-out booster burn time. The pop-out booster burn time will only affect the optimization if the pop-out booster checkbox is selected.

The other three sections are intended to show the user the results of the optimization. All the values inside the example GUI are purely meant to show the functionality of each section, and do not represent our final calculations. The Mass Ratio section shows a visual representation of the optimal mass ratios. This is done using proportional vertical bars. Each of these bars are labeled, ensuring the user can tell which bar corresponds to which value. The next section is labeled data. This shows a table with the velocity and mass ratio of each stage, along with the combined delta v. While the mass ratio section shows a visual representation, this section gives exact values.

The final section is the Graphs section. This section has not yet been implemented in the GUI mockup, primarily because we will not know the best way to show the data we find until we actually find the data. The current plan is to have a 2D histogram. The x-axis will show MR1, while the y-axis will show MR2. The color or intensity of the bin will show the delta v for those values. This graph does not have to show MR3 since the last mass ratio can be dependent on the other two, meaning only one delta v value can be found for any two mass ratios. Another possibility would be to show this same data in a 3D line graph, similar to 3D graphing software such as Desmos. The library we are using to create this UI supports scrolling, so we can put as many graphs as needed inside of this section.
Our program will be compiled into an exe. Exe files are the most common way to run applications and will likely be supported for the foreseeable future. The program is developed using Python. Python is a massively popular programming language that will also be supported for the foreseeable future. The most likely cause of a stunt in development is the shutdown of the dearpygui library. If the developers discontinue maintenance, the library may no longer be supported by Python. If this does happen, the exe will still run but there will no longer be a way to edit the program and recompile the source code. dearpygui is still in active development and even if that changes the exe will still be functional, but we will be unable to update the program. Overall, there are no reasonable threats to the lifecycle of our final product.

Material Considerations

This project does not require any physical materials or software to be bought for its completion. It can be completed using free software that can be found online and is available to the public.

3.4 Initial Fabrication and Testing Plan

Project Workflow

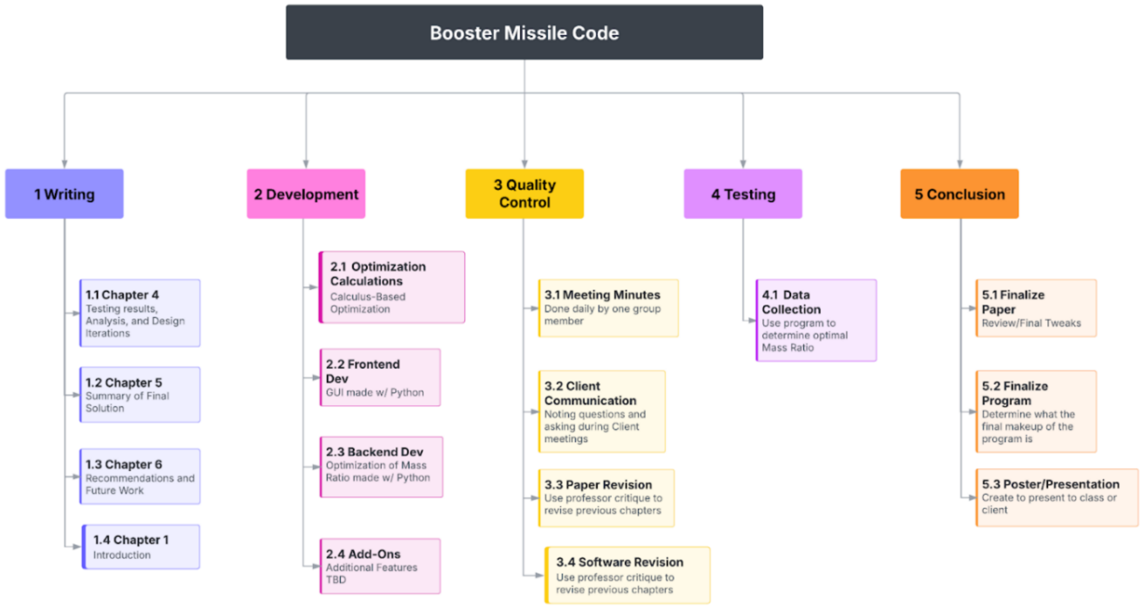

Booster Missile Code
1 Writing
2 Development
3 Quality Control
4 Testing
5 Conclusion
1.1 Chapter 4
Testing results, Analysis, and Design Iterations
1.2 Chapter 5
Summary of Final Solution
1.3 Chapter 6
Recommendations and Future Work
1.4 Chapter 1
Introduction
2.1 Optimization Calculations
Calculus-Based Optimization
2.2 Frontend Dev
GUI made w/ Python
2.3 Backend Dev
Optimization of Mass Ratio made w/ Python
2.4 Add-Ons
Additional Features TBD
3.1 Meeting Minutes
Done daily by one group member
3.2 Client Communication
Noting questions and asking during Client meetings
3.3 Paper Revision
Use professor critique to revise previous chapters
3.4 Software Revision
Use professor critique to revise previous chapters
4.1 Data Collection
Use program to determine optimal Mass Ratio
5.1 Finalize Paper
Review/Final Tweaks
5.2 Finalize Program
Determine what the final makeup of the program is
5.3 Poster/Presentation
Create to present to class or client

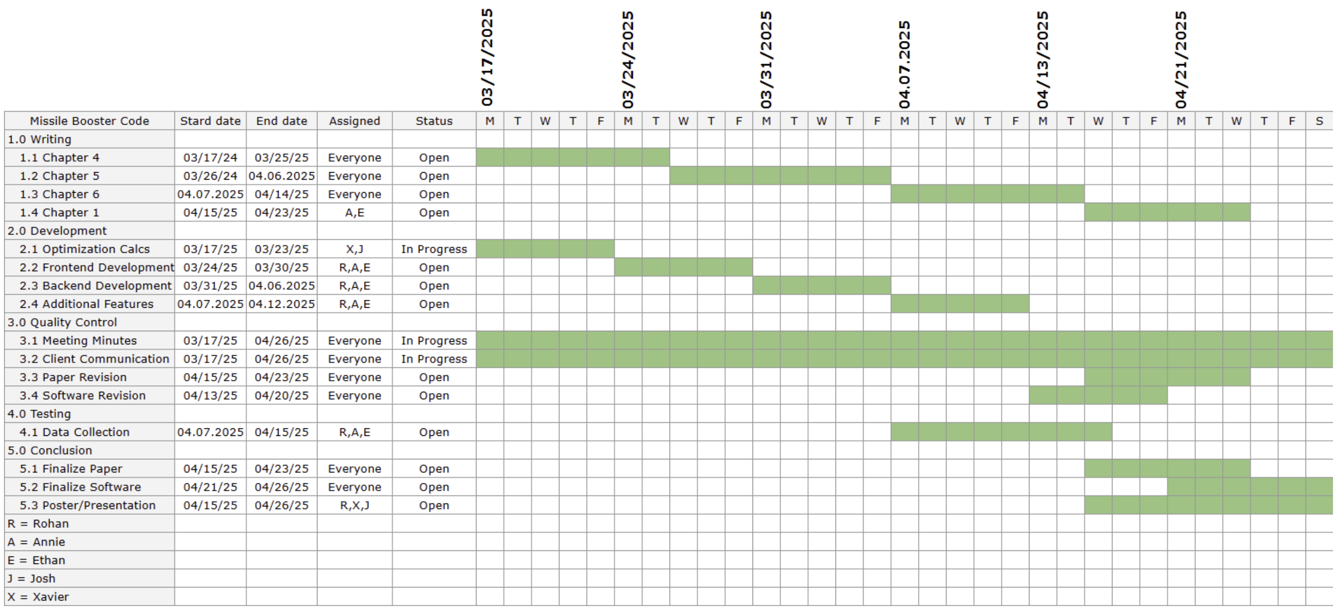
Figure 8: Teamwork Breakdown Structure (WBS)

The following WBS shows how the team has broken down tasks for the rest of the project. In the future, more detailed tasks will be added as what exactly about the GUI needs to be developed, what exact formulas must be calculated by hand, and what exact formulas must be integrated into Python. However, at this moment, that has not been accounted for.

Linear Responsibility Chart	Rohan	Annie	Ethan	Xavier	Josh
1.0 Writing	1	1	1	1	1
1.1 Chapter 4	1	2	4	1	1
1.2 Chapter 5	2	1	2	4	1
1.3 Chapter 6	4	1	1	2	1
1.4 Chapter 1	1	4	1	1	2
2.0 Development	2	1	1	2	2
2.1 Optimization Calculations	4	3	3	1	1
2.2 Frontend Development	1	1	1	3	4
2.3 Backend Development	1	1	1	3	4
2.4 Additional Features	1	1	1	3	4
3.0 Quality Control	1	1	2	2	2
3.1 Meeting Minutes	2	1	4	1	2
3.2 Client Communication	2	2	1	1	2
3.3 Paper Revision	1	1	2	1	2
3.4 Software Revision	1	1	1	3	4
4.0 Testing	1	1	1	3	4
4.1 Data Collection	1	1	1	3	4
5.0 Conclusion	1	1	1	1	1
5.1 Finalize Paper	1	1	1	1	1
5.2 Finalize Software	1	1	1	3	4
5.3 Create Poster/Presentation	1	1	1	1	1
Key:					
1 = Primary Responsibility					
2 = Secondary Responsibility					
3 = Responsible if asked for assistance					
4 = Review before submission					

Table 2. Team Linear Responsibility Chart (LRC)

Above is shown the LRC, where all tasks are broken down to show which team members will be working on and what the focus of that team member will be. For some, the focus will go into software development for the program and GUI, while for others, focus will go into calculations and research to make the program successful. Further broken-down tasks shall be added to the LRC in the future.



					03/17/2025					03/24/2025					03/31/2025					04.07.2025					04/13/2025					04/21/2025					
Missile Booster Code	Stard date	End date	Assigned	Status	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	S
1.0 Writing																																			
1.1 Chapter 4	03/17/24	03/25/25	Everyone	Open	🟩	🟩	🟩	🟩	🟩	🟩	🟩																								
1.2 Chapter 5	03/26/24	04.06.2025	Everyone	Open								🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩																
1.3 Chapter 6	04.07.2025	04/14/25	Everyone	Open																🟩	🟩	🟩	🟩	🟩	🟩	🟩									
1.4 Chapter 1	04/15/25	04/23/25	A,E	Open																							🟩	🟩	🟩	🟩	🟩	🟩			
2.0 Development																																			
2.1 Optimization Calcs	03/17/25	03/23/25	X,J	In Progress	🟩	🟩	🟩	🟩	🟩																										
2.2 Frontend Development	03/24/25	03/30/25	R,A,E	Open						🟩	🟩	🟩	🟩	🟩																					
2.3 Backend Development	03/31/25	04.06.2025	R,A,E	Open											🟩	🟩	🟩	🟩	🟩																
2.4 Additional Features	04.07.2025	04.12.2025	R,A,E	Open																🟩	🟩	🟩	🟩	🟩											
3.0 Quality Control																																			
3.1 Meeting Minutes	03/17/25	04/26/25	Everyone	In Progress	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
3.2 Client Communication	03/17/25	04/26/25	Everyone	In Progress	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
3.3 Paper Revision	04/15/25	04/23/25	Everyone	Open																							🟩	🟩	🟩	🟩	🟩	🟩			
3.4 Software Revision	04/13/25	04/20/25	Everyone	Open																					🟩	🟩	🟩	🟩	🟩						
4.0 Testing																																			
4.1 Data Collection	04.07.2025	04/15/25	R,A,E	Open																🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩								
5.0 Conclusion																																			
5.1 Finalize Paper	04/15/25	04/23/25	Everyone	Open																							🟩	🟩	🟩	🟩	🟩	🟩			
5.2 Finalize Software	04/21/25	04/26/25	Everyone	Open																										🟩	🟩	🟩	🟩	🟩	🟩
5.3 Poster/Presentation	04/15/25	04/26/25	R,X,J	Open																							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
R = Rohan																																			
A = Annie																																			
E = Ethan																																			
J = Josh																																			
X = Xavier																																			

Figure 9. Team Gantt Chart

Above is the team Gantt Chart, which is depicted to break down tasks and how they have been planned on being completed time wise. While there is flexibility in the dates on which some tasks have been chosen to start and be completed, this is more for general clarity. Some tasks, such as the meeting minutes, are done every day.

Preliminary testing plan

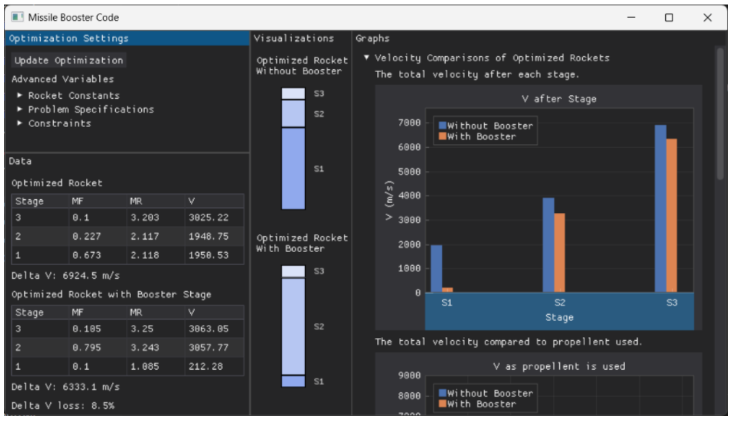
There are several elements of the product that the team intends to test, mainly including user experience with the GUI, edge case values, code understandability and readability, as well as code optimization. Testing for user experience with the program GUI is valuable to the project to ensure that users can easily navigate and use the program. This test will likely be conducted through a survey that is distributed to users that have the chance to test/use the product. Feedback on this survey will likely influence the team to make any necessary changes reflected in the responses to the GUI in an effort to increase usability. Testing for edge case values is valuable to the project to ensure that the program does not break under any circumstances having to do with user input, which is a standard expectation for all software developers. This test will likely be conducted by the team inputting extreme values for the program variables and observing how the program behaves. Depending on the behavior of the program when run with specific inputs, if the program crashes, the team will hardcode preventative measures to ensure that the program does not crash, such as automatically defaulting to “safe” values if the user value is going to cause an error. Testing for code understandability and readability is valuable to the project in order to fufill the success criteria of ensuring that the product can be easily understandable to an engineer who may not have extensive coding experience. This test will likely also be conducted with a survey that is distributed to test users who read through the optimization code. Then, depending on the feedback of these users, the team will make necessary adjustments to the actual code, whether that be through adding more comments or renaming functions. Finally, testing for code optimization is valuable to the project to ensure that the program still runs relatively quickly in the case that the user chooses to input values on the larger side of reasonability. The team plans to test this through adjusting the structure of the code, running the program with large test value, and finding the time it takes for the program to run using the Python timeit module. The code structure that runs the fastest will then be chosen as the final structure for the optimization code of the application.

4. Testing Results, Analysis, and Design Iterations

4.1 Initial Design Description

Our solution requires two stages. The first is solving the optimization process. After solving the problem, the second stage requires us to create a way for the user to interact with and view the optimization. When creating the optimization, the first major change was the approach. Originally, the plan was to implement a calculus-based optimization method. This would have been more efficient and given us exact values. Despite this, our client made it very clear he wanted the program to graph the optimization process. Since the program would have to calculate the Delta V for many different mass fractions, the computer would have to essentially perform a brute force optimization to create these graphs. Since the calculus-based method is no longer more efficient, our program uses step size optimization by trying mass fractions for the stages at each .001 interval. This allows us to find the best mass fraction while creating graphs of how Delta V changes relative to the mass fraction.

The next major question relating to the optimization was how to calculate the mass of the structure for each stage. The first iteration used an algorithm that calculated the structure of the mass by using a structural efficiency relative to the load. The load would include the payload, propellant of that stage, propellant of the stages above it and structural mass of the stages above it. After testing, it was clear this method made the lower stages underperform. The optimization was attempting to minimize stages 1 and 2, essentially making a one stage rocket. Since this does not line up with the expected results, it was clear that this model did not accurately represent how the structural mass would be separated in a rocket. So instead, the program calculated structural mass per stage as a percentage of the propellant mass for that stage. This gave much better results for the optimization that lined up the behavior of a physical rocket.



Data

Optimized Rocket

Stage	MF	MR	V
3	0.1	3.203	3025.22
2	0.227	2.117	1948.75
1	0.673	2.118	1950.53

Delta V: 6924.5 m/s

Optimized Rocket with Booster Stage

Stage	MF	MR	V
3	0.105	3.25	3063.05
2	0.795	3.243	3057.77
1	0.1	1.085	212.28

Delta V: 6333.1 m/s

Delta V loss: 8.5%

The total velocity after each stage.

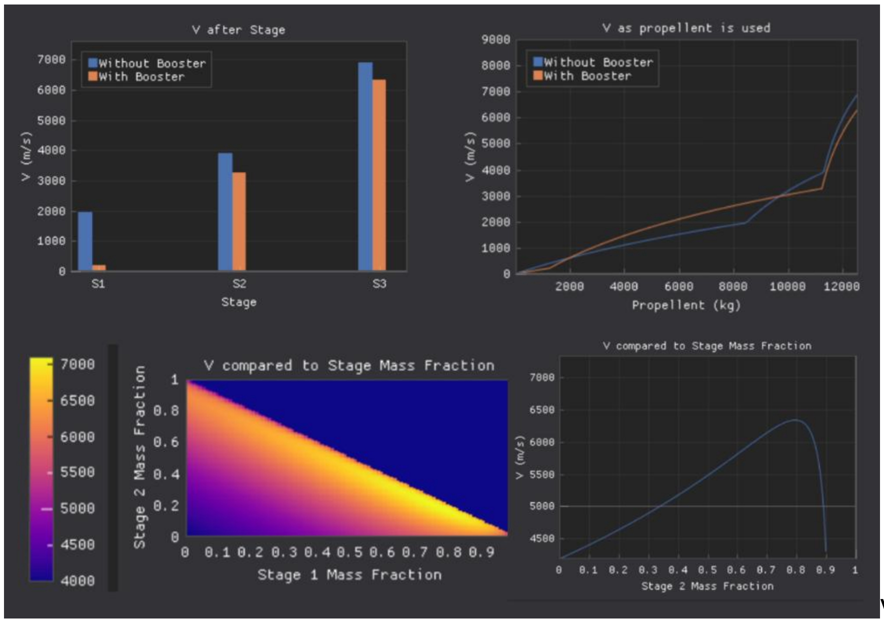
The total velocity compared to propellent used.

Figure 10. A screenshot of the finished initial GUI

After finishing the optimization, the next step was to create the UI. As shown by the screenshot in Figure 10, the UI is separated into four sections; data, graphs, settings and visualizations. Each section is meant to be as simple as possible, without sacrificing accuracy or functionality. This required multiple iterations, ensuring that all the necessary data fit on the screen without having to navigate a confusing number of menus or tabs. The layout and size of the windows were adjusted multiple times throughout development since our initial prototype.

The biggest change in our software from the earliest prototype was the removal of the booster stage checkbox. Originally, the user would have to run a separate optimization for a rocket with a booster stage. After trying to design the UI like this, it was clear that this would not properly solve the problem, and it would make it difficult to compare the two rockets. The optimizations were combined into one button, which then generated two data tables and graphs showing both optimization processes and the performance of both optimized rockets.

The optimization processes for both rockets are shown and compared in the graphs section. Figure 11 shows all the graphs laid out side by side. In the actual program, the user scrolls to view the graphs. The first two graphs compare the performance of both optimized rockets. One graph is a bar graph that shows the velocity after each stage. This graph is extremely simple and easiest to read. Because of this, it is shown at the top of the graphs section. This is meant to very quickly show the differences in Delta V between the rockets. The next graph shows the change in velocity as propellant mass is used. This gives a more in-depth display of how the velocity changes throughout the entire flight of the rocket. Assuming the rate propellent is used remains constant, this graph shows how the velocity changes over time.



Vb

Figure 11. All the graphs shown by the program using the default optimization settings.

The last two graphs focus on the optimization process. One graph shows the optimization without the booster stage while the other shows the optimization for the rocket with the booster stage. Since the one graph must optimize all three stages, it is shown using a heatmap. One axis is the mass fraction for stage one while the other is the mass fraction for stage two. Since both stage one and two are shown, there is only one possible value for stage three for each point on the heatmap. If both mass fractions add up to a value greater than one, then the graph shows the lowest velocity color. The graph showing the booster stage optimization is a simple line graph. Since the 1st stage is constrained to the burn time, only one stage is needed to represent a rocket configuration. The x-axis is the mass fraction for stage two and the y-axis is the Delta V of that rocket.

The visualization of the mass fractions, graphs and data table all convey the results of the optimization in a way that is intuitive and easy to comprehend quickly, without sacrificing accuracy or clarity. All the graphs that were chosen were done so because they provide a meaningful insight into the data. When deciding which graphs to show, the intention was to go from simple to complex. The first graph is an extremely simple representation of the velocities for each rocket and the last graphs show the details of the optimization process. All of this allows the user to view the inner workings of our program without being overwhelmed initially. This is meant to create a program that is not intimidating to those without a lot of background knowledge, while still being useful for experts in the field.

4.2 Testing procedure

The first test the team conducted was for usability, specifically focusing on the usability of the GUI (Graphical User Interface) for the program. This test was conducted through a Google Form survey which was distributed to random users who had the opportunity to interact with the program GUI. The users then answered questions related to the accessibility and navigational ability of the GUI, as well as provided further elaborations on some of their answers. These questions consisted of asking users to evaluate the following criteria: visual appeal, ease of navigation, editability, overall usability, accessibility, intuitiveness, and user-friendliness. The data collected was both quantitative and qualitative. The quantitative data collected involved counting the percentage of users who believed the GUI to be easily usable overall, and the qualitative data involved the actual descriptive feedback that was received on certain aspects of the GUI, such as accessibility and user-friendliness. The team chose this test to gain insight into the usability of the product, as well as receive any unbiased feedback in order to adapt the GUI to be as user-friendly as possible. This directly links to the success criteria of creating a program with user-friendly GUI that provides a visual representation of the data in a meaningful and useful way. Important breakthroughs with this test include the feedback that the text could be larger and that some of the graph labeling was unclear, leading to confusion on what data was being visually represented. An important threshold of this test was that the GUI would be considered reasonably usable if at least 90% of test users agreed with all the questions assessing the criteria mentioned earlier. The outcomes of most interest relate to the important breakthroughs, mainly the fact that the labeling of the graphs, or lack thereof, were preventing the visual representation of the program data from being interpreted in a meaningful manner. The data collected will be used to improve the GUI from a usability standpoint, such as increasing the size of both the program window and the font, as well as clarifying graph labels and using colors that make the graphs more visible.

The second variable the team tested was the readability and understandability of the missile optimization code. This test was also conducted through a Google Form survey that was distributed to users who read through the code in the missile_optimization.py file. After reading through the code, users answered questions related to whether they thought the code was readable and understandable, as well as provided feedback towards specific components of difficulty and suggestions for improvement. Similarly to the test for GUI usability, the data collected from this test was both quantitative and qualitative. The quantitative data collected consisted of the percentage of test users who believed that the code was readable and understandable. The qualitative data consisted of specific feedback towards specific components that caused difficulty and suggestions for improvement. The team chose this test to gain a non-biased assessment of whether or not the code was easily readable and understandable to those not directly involved in the code-writing process. The team also chose this test to receive feedback on areas for improvement to ensure the code would be as easily readable and understandable as possible. This directly links to the success criteria set by the clients in which the program code should be understandable to any engineer with basic programming knowledge. As code must also be readable to be understandable, testing for readability and understandability was vital in ensuring that the code met the clients' success criteria. An important threshold of this test was that the code would be considered overall readable and understandable if at least 90% of test readers agreed that it was relatively simple both to read and understand the code given basic coding experience. Important breakthroughs and outcomes of interest include that the most common component that prevented code understandability was the lack of comments preceding each method. The data collected will be used to adapt the actual code of the missile optimization program to improve understandability, namely adding more descriptive comments prior to every separate method.

The third variable the team tested for was functionality, specifically the ability of the program to function with edge case inputs. In this case, edge case inputs refer to user input values for the rocket constants, problem specifications, and constraints that are "extreme" in the sense that they are unreasonably small and unreasonably large. The only required material/equipment to conduct this test was the program. This test was conducted by repeatedly running the program with a series of extreme inputs for the customizable rocket constants, problem specifications, and constraints. Some of these include putting values of zero for supposedly non-zero rocket constants and setting negative values to the problem specifications. The only required material/equipment to conduct this test was the program and the computer used to run it. The data collected from this test was qualitative in the sense that the team observed the program behavior and how the program reacted to extreme and invalid inputs. The most important threshold for this test was that the program would be considered functional if it continued to work as intended and did not break no matter what input the user decides to set the values to. This means that even if the user inputs values that are infeasible and invalid, the program should not crash and rather continue to display results with a pre-set valid set of variable values. Important breakthroughs and outcomes of interest include that initially, the program was processing negative inputs for the rocket constraints and problem specifications which would cause errors in the code, essentially breaking the program. Furthermore, when the minimum mass fraction was greater than around the value of 0.300 no matter the maximum, and the maximum mass fraction was less than the value of 0.300 no matter the minimum, an error in the code would occur and the correct optimization could not be calculated. The data from this test will be used to integrate defensive programming techniques into the code of the program, such as hard-coding pre-set values that the variables can default to if the user-input is unreasonable and/or invalid. The purpose of testing for functionality of edge cases was to ensure that the product was built to withstand any input from the user, and to make sure that even if the user makes a mistake in their input, the program reacts appropriately and does not crash. This directly relates to one of the most important success criteria of the product, that the program consistently functions and can accurately provide optimized values for a three-stage missile with and without a pop-out booster.

4.3 Testing results and analysis

Survey data was collected regarding the usage and readability of the code, as well as how accessible and visually understandable the program was with regards to both inputting to the UI and reading the output graphs. The survey revealed that 82% of respondents with introductory coding knowledge described the code as readable, and an even greater 90% describing the UI as intuitive, and 100% responded that the UI was well formatted and good looking. The output graphs, however, were said to be hard to interpret, with some basic knowledge of the problem required before they are readable. The last variable tested was functionality and handling of edge cases, which resulted in the program being written with only safe user inputs preventing the program from encountering errors in execution.

Within a second survey conducted, 100% of participants agreed that the code was readable. In the same survey, only 75% understood the code. Due to the disconnect between the readability and the overall understanding of the code, solutions regarding improved commenting within the code were proposed. The commented code both had the added benefit of allowing individuals who are not adept in physics to understand the optimization portion of the code, as well as preventing the scope of the code to just people who study CS or understand Python. Overall, the main issue between each iteration of the code fell upon client-code interactions. Within the client interactions, the interpretation of variables was skewed from lack of commenting and challenging variable naming. Although variable names would likely be understood by the “main” client without challenge, within universal design it is necessary to make the code both accessible and readable for the average person, even if the average person would likely not be reading the code.

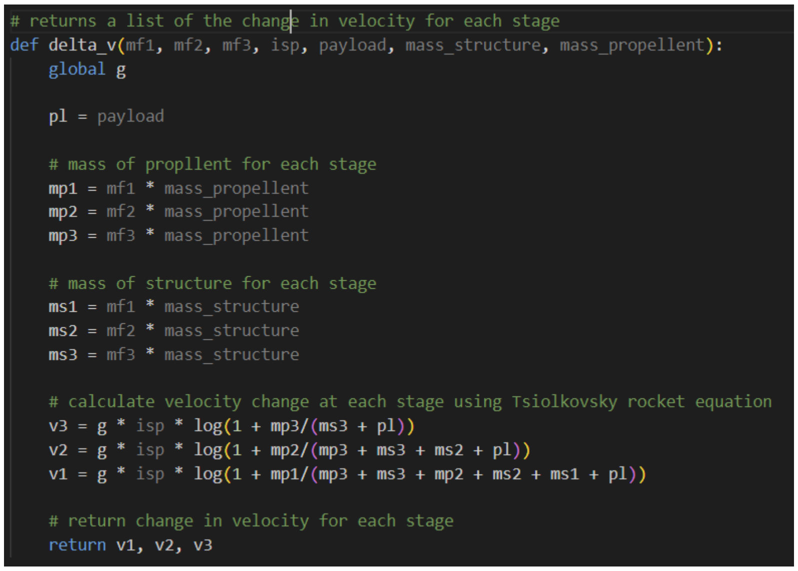
Surveyed participants declared that the GUI was both intuitive and accessible, with the user and graphic interface being straightforward and easy to use. As a result, during the second wave of changes, the integrity of the GUI had to both be preserved from a graphics perspective, but also through the implementation of reader interaction within the code (including the accessibility of graphs).

In regard to the graphs, users did have challenges with discerning what items were on the data visualizations. Most concerns fell under improper labelings and interpretations of results. Improved axis labeling and stating units would not harm the GUI, so future fixes along the graph route would pose no issue to the code and existing successes.

4.4 Informed Design Revision

Justifications based on data

As stated before, the variables that were tested were code readability and understanding, GUI usability, and functionality with edge cases. Although most data that came from the testing results was positive, there were certain parts of the code and GUI that were changed after analyzing the data.



```
# returns a list of the change in velocity for each stage
def delta_v(mf1, mf2, mf3, isp, payload, mass_structure, mass_propellent):
    global g

    pl = payload

    # mass of propllent for each stage
    mp1 = mf1 * mass_propellent
    mp2 = mf2 * mass_propellent
    mp3 = mf3 * mass_propellent

    # mass of structure for each stage
    ms1 = mf1 * mass_structure
    ms2 = mf2 * mass_structure
    ms3 = mf3 * mass_structure

    # calculate velocity change at each stage using Tsiolkovsky rocket equation
    v3 = g * isp * log(1 + mp3/(ms3 + pl))
    v2 = g * isp * log(1 + mp2/(mp3 + ms3 + ms2 + pl))
    v1 = g * isp * log(1 + mp1/(mp3 + ms3 + mp2 + ms2 + ms1 + pl))

    # return change in velocity for each stage
    return v1, v2, v3
```

Figure 12. Example of comments put into code to enhance user understanding

The data from the survey that asked users how readable and understandable the code indicated that a lot of non-technical users had a hard time understanding what the code did. This resulted in there being more comments put into the code that clarifies what certain functions do and how they are used in other places. Although we know that our client does have a technical background and will be able to read and understand what the python code does, it makes it simpler if there are comments that explain what the code does instead of leaving it up to interpretation. This was the only iteration made to the code in terms of improving user understandability and the comments that were added do not change anything in the actual implementation of the application itself. This means that there were no drawbacks that came from adding comments to the missile optimization code that described what each segment of code was doing. Compared to the original, the comments make a significant improvement in being able to understand the code to someone who is looking at it for the first time.

Design revisions

When looking at the GUI, the results that came from the surveys that were given out to random users indicated that the graphs that were constructed of the optimized missiles could be confusing to someone who is looking at it for the first time. This was indicated as some users stated that the graph axes could have more descriptive labels as that could lead to more understanding of what the graphs mean. The client required a graph be made that visualizes the results of the optimized rockets, so more descriptive labels could increase client understanding of what they are looking for and if the requirement is being met. The major changes were made to the heatmap that is in the GUI, which now has labels that describe how the x-axis is the mass fraction of stage 1, the y-axis is the mass fraction of stage 2, and the heat represents the mass fraction of stage 3. Compared to the original heatmap that was present in the GUI, which only had the letter V to indicate the max velocity according to the heat, the heatmap now is significantly more user-friendly and descriptive for a first-time user. Making the graph axes more descriptive should have no major drawbacks as there is nothing being sacrificed in terms of the look and feel of the GUI.

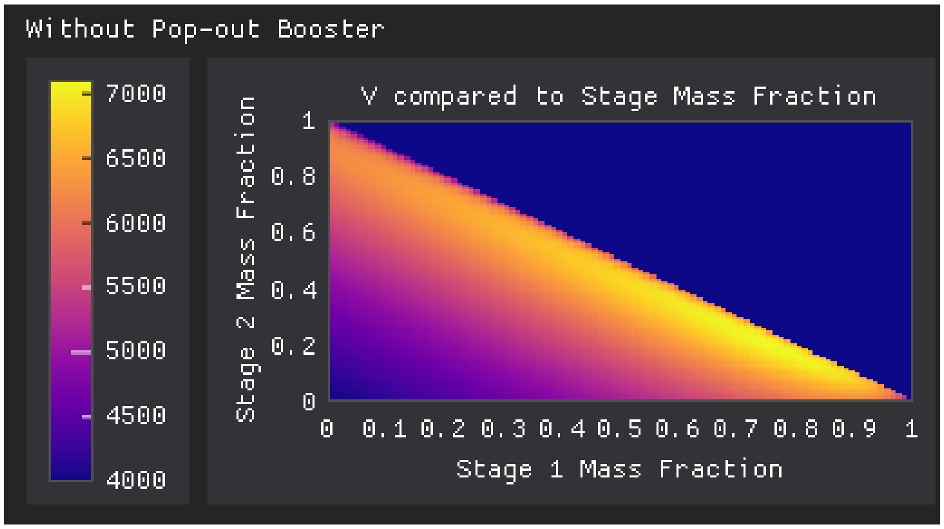


Figure 13. Heatmap in GUI with more descriptive labels on axes and title

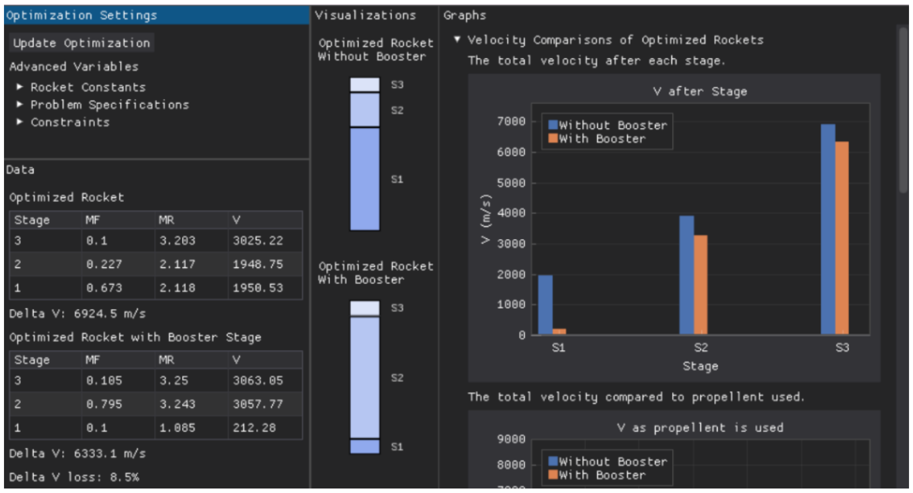
When looking at the testing for edge cases and unreasonable inputs, the results were shown instantly as the program would either calculate values that were mathematically incorrect or would simply throw an error. For the client specifically, because they will have the technical experience to know what values are reasonable for that specific variable, there should be little issues. The problem becomes more apparent when there are technical and non-technical first-time users that input numbers that are out of the bounds for certain variables or enter in negative numbers, and in this case, the program will start running into problems. To mitigate this issue, the code was changed so that when such invalid numbers were put into the inputs, the program would instead just go back to the default inputs that were in the GUI and calculate the tables and graphs for those values. Compared to the initial code, this iteration saves a lot of time and load that is put into the computer when it attempts to do calculations with numbers that will cause problems mathematically. One drawback that has popped up is that when users first open the GUI, and put in an invalid variable, and the GUI simply shows the graphs that were already there, the user assumes that the program is not working, causing confusion.

5. Summary of Final Solution

5.1 Description of final design

Detailed description of final design

Our final solution was very similar to our initial fabrication. On startup, the program calculates and displays the optimization to the problem described by our client. This is done so the user does not have to enter any values or even interact with the program to see the default optimization. The program has four windows, one changing the settings of the optimization and three showing the results. The settings window is separated into three sections. The rocket constants are assumptions on how the rocket behaves and is constructed. This includes the ISP, fuel density and structural efficiency. The problem specifications allow the user to change the parameters our client provided. This includes the payload, stack height, diameter and booster burn time of the rocket. While these are already set what the client requested, the user has the ability to change them to see how these affect the results of the optimization. The final section is the constraints. This sets limits to the values of the mass fractions. Since when building an actual rocket, the stages need to fit an engine and electronics. Because of this, the user can decide the smallest size a stage can be. The default allows mass fractions to be between 0.1 and 0.8.



Data

Optimized Rocket

Stage	MF	MR	V
3	0.1	3.203	3025.22
2	0.227	2.117	1948.75
1	0.673	2.118	1950.53

Delta V: 6924.5 m/s

Optimized Rocket with Booster Stage

Stage	MF	MR	V
3	0.105	3.25	3063.05
2	0.795	3.243	3057.77
1	0.1	1.085	212.28

Delta V: 6333.1 m/s

Delta V loss: 8.5%

The total velocity after each stage.

The total velocity compared to propellent used.

Figure 14. Screenshot of the final program.

The next three windows show the results of the optimization. The data section shows the exact values of the optimized rockets with and without a booster stage. This includes the mass ratio for each stage, the mass fraction for each stage and the Delta V each stage contributes. The data section also shows the final velocity of each rocket. After preliminary testing and reviewing the client's requests, it was clear there was not a way for the user to easily get a numerical value comparing the two rockets. Because of this, the solution now also shows the percentage of velocity lost by the booster. While the initial fabrication visually showed this in this graphs section, having the numerical percent difference is also extremely important. The next window shows a visual representation of both rockets. This is a simple set of bars proportional to the mass fractions that are meant to give the user an idea of what the physical rocket would look like. This is very important to show the user how our findings translate the physical construction of the rocket.

The most important and complicated window is the graphs window. It includes four graphs. The first two compare the velocity of both optimized rockets. The next two show the optimization process and two compare the optimization for both rockets. The final two graphs show the optimization process. One graph shows the optimization process without the booster while the other shows the optimization process with the booster.

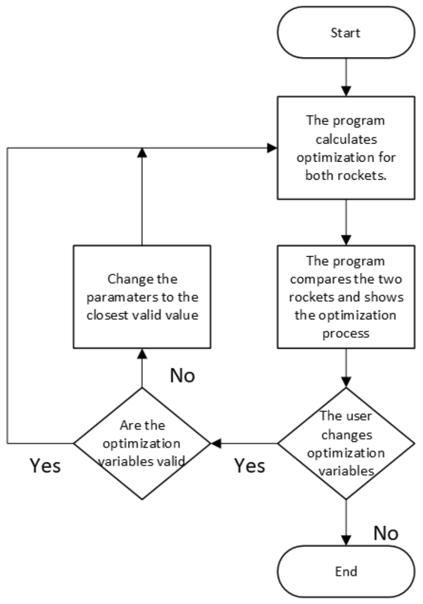

Start
The program calculates optimization for both rockets.
The program compares the two rockets and shows the optimization process
The user changes optimization variables
Yes
Are the optimization variables valid
No
Change the paramaters to the closest valid value
Yes
No
End

Figure 15. Updated flowchart of final program functionality

After finalizing the program, initial testing showed some problems. When setting variables for the optimization settings, it was possible for the user to break the problem. To solve this, we need a precheck before our optimization calculation as shown in Figure 15. For example, if the constraints were set so that the rocket was not possible, the program would simply not respond. While the constraints were clamped between 0 and 1, if a user set the minimum to 0.5 and the maximum to 0.8, the rocket would not be possible. Because of this, the values of minimum constraint could not exceed ⅓ and the maximum constraint could not be lower than ⅓. This ensured no matter what the user entered a rocket could still exist under the entered constraints. Another example of a program breaking value would be a booster burn time larger than the burn time of the entire rocket. If this occurred, the burn time was set to 80% of the rocket's burn time.

After these changes, the code was compiled to an executable and sent to our client. Compiling to an executable format allows the user to run the program without downloading python or the libraries used in our program. This is especially useful for the U.S. Navy, since employees may not have access to the terminal without admin privileges, which is required to download python libraries. The final changes to our solution were made to improve the readability and understandability of the code. This included meticulously commenting and creating documentation inside of a README file in the repository. This will allow our work to be useful to others. Since our optimization is calculated in a separate module than the UI, the user could use that to create separate graphs or an entirely new GUI. With this documentation, the work done on this project can be recreated and built upon, ensuring that the findings of this project are not just limited to the application we built.

Design performance

The first requirement of our design was that the optimization returns accurate values. While we were not able to meet with our client, other groups working on this research obtained similar results and had similar methodology. The values our program returned for every test case had reasonable results. For example, the program never claimed the rocket with a booster stage would be more efficient. The program also never gives an optimized rocket with larger bottom stages. While there is no way to test the objective accuracy of our results without building physical rockets, the results of our program followed what we expected based on prior research.

The second important requirement is functionality. During initial testing, some users were able to break the program by inputting invalid values as explained in 5.1.1. But once this was fixed, no user encountered an error while using the program. The program has no known bugs despite extensive testing by the developers, test groups and presentation attendees.

The next important part of the program is the usability and functionality of the UI. Overall, the UI was very well received. This includes the results of a formal survey described in the previous chapter and the observations from observing people use our UI during the final presentation. Most people with some forms of technical literacy were able to quickly understand how to use our program without instruction.

5.2 Reflection of final solution

Fulfillment of objectives and customer needs

The final product was updated heavily after the survey data for the first iteration was taken. Viewers noted challenges with comprehension of the code and general readability. As a result, the final iteration included a fully commented set of code with each section of the product fully defined and outlined for the consumer. The code had sections which described which parts of the code would influence which part of the visuals and data, and overall, the changes were made in a way that anybody who could understand what was commented could also understand the code

As for other product needs, the code did not produce the wanted outcome of within 5% delta v and range. The code saw a reduced delta v of 8.4%, which did indicate that for the size of the missile and pop-out booster burn time the solution would not be ideal. Although the product was not within the desired range, based on the code provided, there is still the opportunity to achieve the ideal range through tweaking the given constraints set by the client. Essentially, the code can provide a future success framework for future missiles.

The client was seeking if the solution would be useful or not, and the question was answered in the process: the pop out booster with the constraints given would likely not be worth it. As a result, the client would be satisfied with the given outcome because it proves why the tested solution is not ideal.

Sociotechnical considerations in final design

After final revisions, new considerations for the final product fell under understandability and access. Although the client and Navy would likely be the only people who would benefit from the code and/or view it, to make the code universally acceptable, the code must be understood by people with the most basic levels of coding and physics experience. As a result, the commented code was made in a way that fully explained every part, including the graphs and visuals so that anybody from the public could recreate the code with a small amount of CS understanding.

Previous concerns fell under humanitarian questions, climate effects, and international relations. Concerns fell under how the code would be utilized, primarily in the scope of conflict, and how the concerns would ethically weigh against an engineering design. Given that the code was not within the expected range, the previous concerns are likely nullified, as the final mass ratio design would not be utilized in any conflict or tests. If the code was utilized in some way, perhaps as a framework, then the concerns are still limited. Ethically, designs that could be used against another person or that could harm the environment must be addressed with caution; however, missiles and subsequent variants can be and will be used in war until missiles become obsolete. Given that the code only provides a framework for the editing of the constraints and variables, the missile design would not be reinventing a new way to change international tides. As such, ethically the code is safe.

6. Recommendations and Future Work

6.1 Recommendations

The key recommendations that the team identified to improve the product in the future fall into two main categories: software features/quality and algorithm accuracy. The first recommendation the team identified under software quality was to transfer the deployment of the product to the form of a website, rather than an executable file that must be downloaded. While the code would still be available to any engineer who wishes to see it, it is likely that if the user simply wants to use the product, it is much more familiar to them to open a website rather than an executable. This recommendation works towards further improving the usability of the product, specifically in terms of ensuring the user does not have difficulty opening and closing the software. The second recommendation the team identified under a software standpoint was to add a feature within the program that allows users to export the generated graphs to an established spreadsheet software such as Excel. This improvement works towards fulfilling the success criteria of producing a visual representation of the data the code generates. Currently, the GUI displays the program generated graphs on the right-side of the application. However, the minimal size of the GUI prevents any detailed analysis from being done on the graphs, which may diminish their effectiveness to the user. Adding a feature that allows the user to export the data and generated graphs into a more advanced software system significantly improves the effectiveness of the program, allowing users the chance to perform more detailed graph analysis.

Tthe last recommendation in terms of improving the software the team identified was to advance upon the feedback and results received through the GUI usability test and the edge case functionality test. While the team was able to make some changes in response to the feedback in the survey about improving GUI navigation ability and accessibility, such as labeling the graphs in a clearer manner, there was still a lot of feedback the team was not able to act upon before the final presentation. In terms of improving the GUI usability, these recommendations include increasing the font size overall as well as the window size of the application. Furthermore, while the team implemented changes in the code that prevented edge cases from crashing the program, the team would also recommend implementing changes in the GUI to hopefully prevent edge case values from being input in the first place. For example, this could include labels before each variable specifying the meaning of the variable in context as well as a clear reasonable range of values. This recommendation in response to the GUI usability and edge case functionality tests further work towards ensuring the product is as easily usable as possible, and that the software does not break under any user-caused circumstances, which is a standard practice in software development.

The first recommendation the team has in terms of improving the accuracy of the algorithm is to use a different ISP value for each stage of the missile. Currently, the program is set to run the algorithm with a default ISP value of 300 for all the stages. Although the program allows the user to change the value of the ISP to their input of choice, the algorithm still runs using the same user input value for all the stages. Advancing the algorithm to solve for the highest optimization given different ISP values for each stage is much more realistic and can thus greatly improve the effectiveness of the product should it be used by the U.S. Navy. Furthermore, the team ultimately had to switch from a calculus-based optimization approach to brute forcing the optimized velocity with an approximation to three decimal points. Approximating to three decimal points limits the possible mass ratios generated by the program and has the possibility to actually skip over a more precise, optimized value. Once again, to increase the accuracy of the results of the program, the team would like to dedicate more time to solving the optimization with calculus to hopefully get a mathematically deduced optimized value.

Several changes must occur when moving towards the recommendation of integrating the program into the form of a website. First, a domain name must be purchased so that the website can be accessed on a global server. While nothing needs to be done regarding the coding of the algorithm, the entire GUI needs to be redone as the current library used, dearpygui, is not compatible with website development. Instead, other frameworks must be used to transfer the frontend and backend interactions, such as React for the frontend and Flask for the backend. This change would require a lot more planning, as well as involve a change in fabrication, as other frameworks need to be studied prior to coding, and the current dearpygui approach would not be appropriate anymore. Similarly, when moving towards the recommendation of implementing a graph-export feature, nothing needs to be done with the actual optimization algorithm. However, a GUI component would need to be added, and the logic behind the GUI elements would need to be altered to accommodate this new feature. Furthermore, a new script that deals with exporting the data would also need to be written, and the team would likely have to use new libraries such as Python pandas in the implementation. This change would also occur in the planning and fabrication stages as the team would need to be familiar with Python pandas as well as actually implement it. Furthermore, additional testing would need to be conducted to ensure the functionality of this new feature works under any circumstances and does not break for any reason. Finally, moving towards the recommendation of addressing the testing results of the GUI usability test and edge case functionality test is simply another fabrication step and likely would only involve changing the code for the GUI. To maximize user ease, more tests about GUI usability could be distributed to a wider range of users to collect more data. However, it is not completely necessary to work towards this recommendation.

Much of the work that needs to be done when moving towards the recommendation of using a different ISP value for each missile stage involves research to find the most accurate ISP values for each stage. From there on, implementing the change in the algorithm would require some coding changes but nothing too drastic. Similarly, when moving towards the recommendation of using a calculus approach to solve the optimization, the bulk of the work must be done outside of program fabrication. The team would have to dedicate much more time outside of coding to planning and solving the series of optimization equations. The entire algorithm code would need to be changed, as a completely new approach would be taken. However, the GUI code does not have to be changed and could remain the same.

In order to continue this work, the team recommends that other engineers first take the time to understand the code in both the missile_optimization.py file as well as the main.py file to truly understand how the algorithm code and GUI code interact with each other. After future engineers gain a detailed grasp on how the program works, the team recommends that they focus on improving the quality and usability of the software first. The team spent a lot of time dedicated to getting the optimization algorithm code to work, and consequently did not have as much time to implement features and changes into the GUI that would greatly improve user experience. The team also recommends that any future engineers conduct various tests regarding user experience and program functionality. The team found that the feedback received through these tests provided a lot of valuable insight into the program from a consumer point of view, and believed that if the tests were applied to a larger test audience, then the changes made in response could benefit an even greater range of users in terms of usability and accessibility.

References

Bowen, W. Q. (2001). Missile defence and the transatlantic security relationship. *International Affairs*, *77*(3), 485–507. https://doi.org/10.1111/1468-2346.00203

Calabro, M., Dufour, A., & Macaire, A. (2002). Optimization of the propulsion for multistage solid rocket motor launchers. *Acta Astronautica*, *50*(4), 201–208. https://doi.org/10.1016/S0094-5765(01)00164-3

Leishman, J. G. (2023). *Rockets & Launch Vehicle Performance*. https://eaglepubs.erau.edu/introductiontoaerospaceflightvehicles/chapter/rocket-performance/

Mass Ratios. (n.d.). Glenn Research Center | NASA. Retrieved February 11, 2025, from https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/mass-ratios/

Morgado, F. M. P., Marta, A. C., & Gil, P. J. S. (2022). Multistage rocket preliminary design and trajectory optimization using a multidisciplinary approach. *Structural and Multidisciplinary Optimization*, *65*(7), 192. https://doi.org/10.1007/s00158-022-03285-y

NASA - Solid Rocket Boosters. (2020, July 27). https://web.archive.org/web/20200727022348/https://www.nasa.gov/returntoflight/system/system_SRB.html

Numerical study of hot launch of missile inside a tube—PK Sinha, Debasis Chakraborty, 2014. (n.d.). Retrieved February 10, 2025, from https://journals.sagepub.com/doi/10.1177/0954410014522609

Orgeira-Crespo, P., Rey, G., Ulloa, C., Garcia-Luis, U., Rouco, P., & Aguado-Agelet, F. (2022). Optimization of the Conceptual Design of a Multistage Rocket Launcher. *Aerospace*, *9*(6), Article 6. https://doi.org/10.3390/aerospace9060286

Performance and Optimization (Multi-stage rockets) – Rocket Sim. (n.d.). Retrieved February 11, 2025, from https://rocketsim.wordpress.com/2017/05/07/performance-and-optimization-multi-stage-rockets/

Pontani, M., & Teofilatto, P. (2014). Simple method for performance evaluation of multistage rockets. *Acta Astronautica*, *94*(1), 434–445. https://doi.org/10.1016/j.actaastro.2013.01.013

Sinha, P., & Chakraborty, D. (2014). Numerical study of hot launch of missile inside a tube. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, *228*(14), 2604–2611. https://doi.org/10.1177/0954410014522609

SM-3 Interceptor. (n.d.). Retrieved February 8, 2025, from https://www.rtx.com/raytheon/what-we-do/strategic-missile-defense/sm-3-interceptor

Standard Missile. (n.d.). United States Navy. Retrieved February 8, 2025, from https://www.navy.mil/Resources/Fact-Files/Display-FactFiles/Article/2169011/standard-missile/https%3A%2F%2Fwww.navy.mil%2FResources%2FFact-Files%2FDisplay-FactFiles%2FArticle%2F2169011%2Fstandard-missile%2F

Trident II (D5) Missile. (n.d.). United States Navy. Retrieved February 8, 2025, from https://www.navy.mil/Resources/Fact-Files/Display-FactFiles/Article/2169285/trident-ii-d5-missile/https%3A%2F%2Fwww.navy.mil%2FResources%2FFact-Files%2FDisplay-FactFiles%2FArticle%2F2169285%2Ftrident-ii-d5-missile%2F

Appendix A: Testing Procedure

Link to GitHub repository. This includes the source code and instructions on how to run the program.

https://github.com/holt-rogers/missile_booster_code

Appendix B: Testing Procedure

GUI Usability:

1. Create a Google Forms survey containing the following questions and corresponding answer choices:
    a. Is the GUI appealing to the eyes?
        i. Yes
        ii. No
    b. Is the GUI easy to navigate, edit, and use?
        i. Yes
        ii. No
    c. Is the GUI accessible? Please elaborate.
        i. N/A. This is a free response question with no answer choices.
    d. Is the GUI intuitive?
        i. Yes
        ii. No
    e. Is the GUI user-friendly? Please elaborate.
        i. N/A. This is a free response question with no answer choices.
2. Provide five minutes for a test user to interact with the GUI by instructing them to provide user input and view the program results.
3. After the five minutes are up, ask the test user to fill out the survey honestly, recording the results and feedback for each question.
4. Repeat steps 2 and 3 with 10 test users

Code Readability and Understandability:

1. Create a Google Forms survey containing the following questions and corresponding answer choices:
    a. Is the code readable?
        i. Yes
        ii. No
    b. Is the code understandable?
        i. Yes
        ii. No
    c. If you answered no to any of the questions above, what components of the code were hard to read/hard to understand (ex: function calls, unclear variable names, too many words, etc.)?
        i. N/A. This is a free response question with no answer choices.

   d. Any suggestions for improvement?
      i. N/A. This is a free response question with no answer choices.
2. Provide five minutes for a test reader to read the code in the missile_optimization.py file. Instruct them to take a mental note of what they understand and do not understand about the solution based on the code alone.
3. After the five minutes are up, ask the test reader to fill out the survey honestly, recording the results and feedback for each question.
4. Repeat steps 2 and 3 with 10 test readers.

Edge-Case Functionality:

1. Run the program and input a negative value for ISP under the Rocket Constants section
2. Press "Update Optimization" and record program behavior and program results
3. Repeat steps 1 through 2 with all variables under their respective sections, ensuring that all other variables have default values, including:
   a. Rocket Constants
      i. ISP (already done in Step 1)
      ii. Fuel Density
      iii. Structural Efficiency
   b. Problem Specifications
      i. Payload (kg)
      ii. Stack Height (m)
      iii. Rocket Diameter (m)
      iv. Pop-out Burn Time (s)
   c. Constraints
      i. Min Mass Fraction
      ii. Max Mass Fraction
4. Run the program and input the value zero for ISP under the Rocket Constants section
5. Press "Update Optimization" and record program behavior and program results
6. Repeat steps 4 through 5 with all variables under their respective sections, ensuring that all other variables have default values, including:
   a. Rocket Constants
      i. ISP (already done in Step 4)
      ii. Fuel Density
      iii. Structural Efficiency
   b. Problem Specifications
      i. Payload (kg)

        ii. Stack Height (m)
        iii. Rocket Diameter (m)
        iv. Pop-out Burn Time (s)
    c. Constraints
        i. Min Mass Fraction
        ii. Max Mass Fraction

7. Run the program and input a value around 500% larger than the default value for ISP under the Rocket Constants section
8. Press “Update Optimization” and record program behavior and program results
9. Repeat steps 7 through 8 with all variables under their respective sections, ensuring that all other variables have default values, including:
    a. Rocket Constants
        i. ISP (already done in Step 7)
        ii. Fuel Density
        iii. Structural Efficiency
    b. Problem Specifications
        i. Payload (kg)
        ii. Stack Height (m)
        iii. Rocket Diameter (m)
        iv. Pop-out Burn Time (s)
    c. Constraints
        i. Min Mass Fraction
        ii. Max Mass Fraction
10. Run the program with different combinations of values for the Min Mass Fraction and the Max Mass Fraction starting from 0 for both up until 1.1 for both, adding with intervals of 0.1., and press “Update Optimization” per new combination. Keep all other variables at their default value.