Simulation Framework for Rocket Launch Trajectories with Various Thrust Profiles

Dev Saxena

December 16, 2024

1 Framework Background

The aim of this simulation framework is to develop an automated pipeline that can be used efficiently in real-time systems to simulate rocket launch trajectories. This is no trivial task because the program involves complex numerical calculations when dealing with varying drag and thrust forces as they change over time. Many applications in the real world such as GNC (guidance, navigation, control) systems, risk analysis, and system diagnostics require the essential components of speed and automation in any simulation framework. Creating a scalable, modular framework to simulate this could save both compute time and resources needed for this kind of simulation. Here is a deeper look at how this form of simulation framework could benefit some common use cases.

1.1 Guidance Navigation Control (GNC) Systems

Due to the complex and critical nature of GNC aerospace applications, there is a heavy emphasis placed on the reliability and veracity of these GNC systems. Modularity of simulation software allows each component to be tested and debugged individually, separates further development of the software, and leads to higher code reusability. Although scalability is not a big issue for these systems as their main purpose is to be hosted on hardware in a physical application such as a rocket or airplane, when the time comes to update GNC systems, scalable software would prevent the rewriting of the existing codebase. Furthermore, modularity improves the parallelization of the program allowing simultaneous computations of data to be run individually. These attributes of my simulation software highlight computational development, performance, and adaptability to mission requirements.

1.2 Other Applications

Similar to GNC systems, risk analysis and system diagnostic applications of the simulation framework can also benefit from the aforementioned attributes making it easy to parallelize multiple simulations, assessing multiple possibilities, and identifying potential failures in similarly complex aerospace systems. The emphasis for these applications is on the minimization of compute time required to run heavy simulations. The goal of most of these applications is to take similar data and run multiple scenarios with it over time, making it a prime candidate for parallelization. As mentioned above for the GNC example, the modularity and scalability of this framework allows easier parallelization and accelerates processes that run multiple simulation scenarios. These tasks could include trajectory calculations, statistical analyses, and sensitivity analyses. The efficiency of these programs can then provide better insights and enable faster responses.

2 Simulation Description

From a high level point of view, this simulation algorithm takes factors such as thrust, drag, and gravity and calculates their effect on the altitude and the velocity over time of a rocket. With all these moving parts, there is no close-form solution to the equations that would describe a rocket's trajectory with many parameters. To solve this issue, we use a numerical method to calculate the rocket's altitude and velocity in a discrete time step. Each stage of the rocket's ascent can be computed in discrete time measurements. The algorithm that makes this computation requires optimizing the time steps and other parameters we choose for accuracy and stability [1]. We lay out this algorithm by first defining the forces that act on the rocket, then defining the differential equations that allow us to observe change over time, and finally, defining the numerical method we use to solve those differential equations. Table 1 describes the terms we use in defining the simulation for the rest of this paper.

2.1 Forces Acting on the Rocket

There are three main forces that act on the rocket in our simulation and they are defined in the following way:

$$F_T = \text{thrust\_profile}(t,\, t_b,\, T_{max}) \tag{1}$$

describes the upward force generated by the rocket in Newtons.
The thrust_profile(t, t_b, T_{max}) function is a specific description of the engine's thrust at time t. This thrust profile is inputted by the user. If a user does not input a custom thrust profile, then the default setting is a thrust profile where the engine thrust decreases linearly with the amount of fuel left in the rocket.

$$F_G = m \cdot G \tag{2}$$

Symbol	**Name**	**Units**
t	Time	s
h	Altitude	m
v	Velocity	m/s
a	Acceleration	m/s^2
m	Mass	kg
G	Gravitational Constant	m/s^2
C_D	Drag Coefficient	
A	Rocket Cross-Sectional Area	m^2
$\rho(h)$	Air Density	kg/m^3
T_{max}	Max Thrust	N
F_G	Gravitational Force	N
F_D	Drag Force	N
F_T	Thrust Force	N
t_b	Engine Fuel Burn Time	s

Table 1: Symbols and their corresponding names and units.

describes the gravitational force in Newtons acting on the rocket where m is the mass of the rocket in kg, and G is the downward acceleration due to gravity in $\frac{m}{s^2}$.

$$F_D = \frac{1}{2} \cdot \rho(h) \cdot v^2 C_D A \tag{3}$$

describes the downward drag force, in Newtons, acting on the rocket [2].

The thrust and gravitational force are pretty straightforward in their definitions. The drag force however, depends heavily on C_D, A, and $\rho(h)$. C_D is difficult to determine as there can be multiple sources of drag forces and testing of the rocket body in a wind tunnel is usually required to accurately determine drag forces. As we do not have access to such resources, we test our rockets with varying values of C_D that rely solely on how streamlined the body of the rocket may be. The drag depends on the size of the body which is defined by a reference area, A, which in this case will be the frontal area of the rocket orthogonal to the direction of the drag flow [2]. Finally, the air density is determined based on the current altitude of the rocket.

With all our forces defined, we first determine the net force acting on the rocket using equations (1), (2), and (3). Then, we use Newton's 2nd law of motion to define the acceleration of the rocket which will be useful when defining the differential equations. [3].

$$F_{\text{net}} = F_T - F_G - F_D$$

$$a = \frac{F_{\text{net}}}{m} \tag{4}$$

2.2 Defining the Differential Equations

The equations we have so far describing the rocket's acceleration are for a single point in time. However, the aim of this project is to obtain the altitude and velocity of the rocket as it moves through time given some initial values. For this, we use a coupled system of ordinary differential equations [6] derived from equation (4) that will define the velocity and altitude of the rocket as it changes over time.

To simplify the differential equations to follow, we define 2 functions, f and g, to compute the velocity and acceleration of the rocket:

$$f(t, h, v) = v \tag{5}$$

$$g(t, h, v) = a = \frac{F_{\text{net}}}{m} \tag{6}$$

Then, our coupled system of ODEs is simply

$$\frac{dh}{dt} = f(t, h, v) \tag{7}$$

$$\frac{dv}{dt} = g(t, h, v) \tag{8}$$

2.3 Numerical Integration by RK4 Method

The system of ODEs given by (7) and (8) are very nonlinear because they depend on dynamic forces, F_G, F_D, and F_T, that depend on time, altitude, and velocity. This results in the system, (7) and (8), to be very difficult or even impossible to solve analytically. Therefore, we must rely on numerical integration techniques that move through time in discrete steps and integrate the system at each step. Although there are many different numerical methods that can solve this system from the Euler method to the trapezoidal method, we choose the fourth-order Runge Kutta (RK4) method. The RK4 method gives us convergence properties superior to methods of the first and second order [4]. The approximation error from the RK4 method also drops much faster than other methods.

When applied to our system of ODEs given by (7) and (8), RK4 approximates the change in velocity and altitude requiring only an initial condition to start. In a certain time step, dt, RK4 uses four slopes at sub-time steps within

dt to estimate how the rocket's velocity and altitude will change.

Let an arbitrary dt (time step) be given by the interval $[t_i, t_i + dt]$. Then, the first slope which calculates the change in altitude and velocity at the left end of the interval, t_i, using the initial values of altitude (h_{t_i}) and velocity (v_{t_i}) is given by

$$s_1^h = dt \cdot f(t_i, h_{t_i}, v_{t_i}) \tag{9}$$
$$s_1^v = dt \cdot g(t_i, h_{t_i}, v_{t_i}) \tag{10}$$

The second slope, which calculates change in h and v at the midpoint of the interval ($t_i + \frac{dt}{2}$) using the slope provided from s_1, is given by

$$s_2^h = dt \cdot f(t_i + \frac{dt}{2}, h_{t_i} + \frac{dt}{2}s_1^h, v_{t_i} + \frac{dt}{2}s_1^v) \tag{11}$$
$$s_2^v = dt \cdot g(t_i + \frac{dt}{2}, h_{t_i} + \frac{dt}{2}s_1^h, v_{t_i} + \frac{dt}{2}s_1^v) \tag{12}$$

The third slope also calculates the midpoint change in values of h and v but uses the values obtained in s_2, giving a better estimate at this point in the time interval.

$$s_3^h = dt \cdot f(t_i + \frac{dt}{2}, h_{t_i} + \frac{dt}{2}s_2^h, v_{t_i} + \frac{dt}{2}s_2^v) \tag{13}$$
$$s_3^v = dt \cdot g(t_i + \frac{dt}{2}, h_{t_i} + \frac{dt}{2}s_2^h, v_{t_i} + \frac{dt}{2}s_2^v) \tag{14}$$

Finally, the fourth slope calculates the change in values of h and v at the endpoint of the interval using the previous slope s_3

$$s_4^h = dt \cdot f(t_i + dt, h_{t_i} + s_3^h, v_{t_i} + s_3^v) \tag{15}$$
$$s_4^v = dt \cdot g(t_i + dt, h_{t_i} + s_3^h, v_{t_i} + s_3^v) \tag{16}$$

We can then write the final change in time, altitude, and velocity for the time interval $[t_i, t_i + dt]$ as

$$t_{i+1} = t_i + dt \tag{17}$$
$$h_{t_i+1} = h_{t_i} + \frac{1}{6}(s_1^h + 2s_2^h + 2s_3^h + s_4^h) \tag{18}$$
$$v_{t_i+1} = v_{t_i} + \frac{1}{6}(s_1^v + 2s_2^v + 2s_3^v + s_4^v) \tag{19}$$

where h_{t_i+1} and v_{t_i+1} describe the altitude and velocity at the next time step. This numerical scheme is calculated over multiple time intervals to obtain the overall changes in altitude and velocity of the rocket.

3 Simulation Implementation

The simulation part of the code is implemented by splitting the work between two classes, a Rocket class and a Simulation class. These are described in tables 2 and 3.

Attributes	Methods
rocket mass	thrust_profile()
fuel mass	thrust_at_time()
T_{max}	fuel_status()
t_b	
burn rate	
C_D	
A	

Table 2: Rocket Class

Attributes	Methods
rocket	air_density()
h_0	drag()
v_0	f()
launch angle	g()
air temp	rk4_step()
air pressure	run()
G	visualize()
dt	analysis()
end time	

Table 3: Simulation Class

The idea behind separating the functionality of the rocket and simulation parts is to be able to test multiple rockets on the same simulation, or testing one rocket on multiple simulations.

The functionality of the Rocket class is limited to storing properties of the rocket itself such as its mass, drag coefficient, and cross-sectional nose area. The Rocket class also contains attributes that describe the engine properties such as fuel mass, max thrust, burn time, and burn rate. If the rocket has multiple engines powering it, they are grouped together and the class stores the combined properties of each engine. The Rocket class also contains methods to calculate the thrust given a thrust profile and update the amount of engine fuel left, which in turn will update the total mass of the rocket at each time step.

The functionality of the Simulation class is to calculate the environmental forces and properties external to the rocket itself, as well as running the Runge-Kutta 4th order numerical integration method. The numerical scheme is computed by running multiple RK-4 steps until the end time step. We first calculate the drag forces at time t and altitude h on the rocket using the air density at the current altitude. Then, the rocket's thrust is obtained and the net forces acting on the rocket are calculated. Finally, we use equations 9-19 and compute the resulting altitude and velocity predictions for the entire time interval.

The Simulation class then runs a visualization function to create a plot of the rocket's trajectory over time and the rocket's velocity over time. For example, consider a model rocket with a total mass of 141.8 g, max thrust of 3.4 N, burn time of 0.5 s, drag coefficient of 0.8, and a cross-sectional nose area of 0.012

m. This rocket was then simulated over 4.0 s, or until it hits the ground, with initial heights and velocities at 0 and a step size, dt, of 0.01. The results are shown in Figure 1.

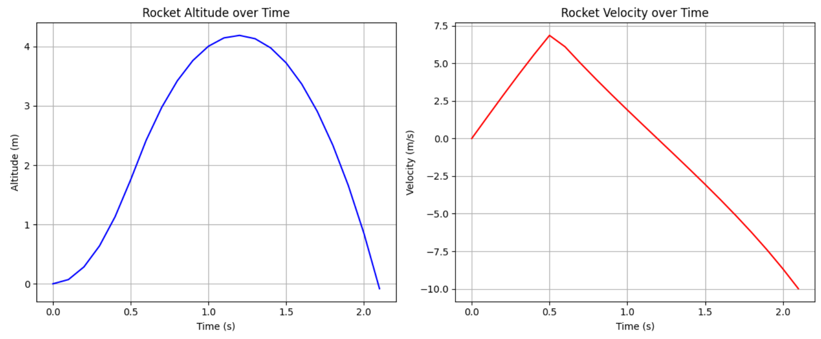

Figure 1: Altitude and velocity of a model rocket over time.

Although this is only a simple example to show what the results would look like, the program can also handle larger model rockets, medium sized rockets such as missiles, and larger rockets such as the Saturn V. Examples testing other rockets with different thrust profiles can be found in the code repository [5].

4 Error Analysis

One of the most useful factors to analyze of any numerical simulation is the error produced from the numerical scheme used. For this application, the Runge-Kutta 4th order method was chosen because it was stable and performed better than other, slightly simpler, numerical integration methods such as the Euler method. However, it is still worthwhile to analyze the convergence of the RK-4 method by calculating the truncation error for various time steps. This can help the user not only analyze the method's integrity, but also help decide the time step to use for the simulation.

There is a trade off that occurs with these numerical integration methods between compute time and accuracy. Due to this trade off, we leave the choice of what time step to use for the simulation up to the user. By analyzing the graph produced in Figure 2, the user will be able to choose other time steps given the stability of the method.

These graphs show the log-log relationship between the time step and the local truncation error for the altitude and velocity. The grid lines are misaligned to emphasize the linear relationship as that is an indicator of how stable the method is. When analyzing these graphs for truncation error, we expect the line to be as linear as possible with constant slope throughout for the method to be stable.

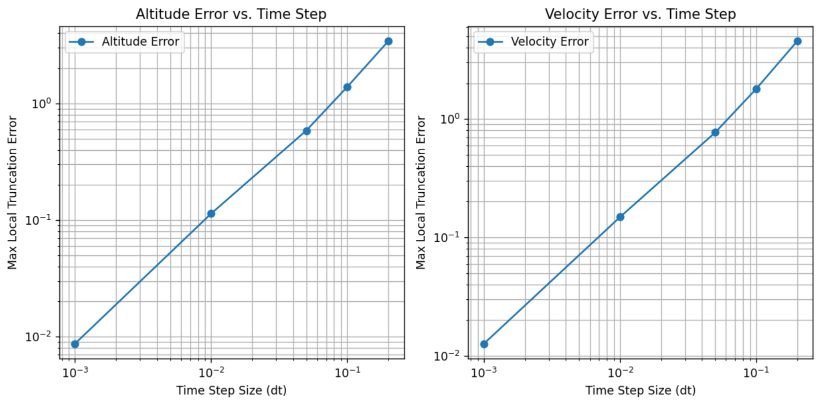

Figure 2: Log-log graphs of altitude and velocity local truncation errors.

5 Conclusion

The aim of this simulation framework was to develop an automated pipeline to simulate rocket launch trajectories with various thrust profiles. It uses a numerical integration method to model the effects of thrust, drag, and gravity on the rocket's altitude and velocity over time. This framework was needed to efficiently handle all the moving parts of the simulation at each time step. The program can handle diverse rocket and engine types as well as different simulation environments. With flexibility in mind, the program allows for easy scalability and a smooth workflow that is easy to understand even with various configurations and conditions.

This simulation framework is applicable for many real-world applications in the aerospace world such as GNC systems, risk analysis, and real-time system diagnostics. The framework can also be used by users to analyze rocket flight before physical experimentation. This can help users design a rocket with the ideal properties for their use-case without having to procure the hardware.

Lastly, the framework's modular design and separation of tasks allows for independent processing, optimization, and scalability. For future work, there are many ways in which the numerical integration method, data flow, and force calculations can be parallelized. For larger scale applications, this parallelization could massively reduce the compute time for many of the calculations that are necessary to model the rocket trajectories. To view all the code used to build this simulation framework, visit the repository hosted on GitHub [5].

References

[1] Andøya Space Center. *Simulating a Rocket Launch.* https://learn.andoyaspace.no/ebook/student-rocket-pre-study/rocket-dynamics-2/simulating-a-rocket-launch/. 2024.

[2] NASA Glenn Research Center. *Drag Equation.* https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-equation-2/. 2024.

[3] The Physics Classroom. *Newton's Second Law.* https://www.physicsclassroom.com/class/newtlaws/lesson-3/newton-s-second-law. 2024.

[4] Timothy Sauer. *Numerical Analysis.* 3rd. Pearson, 2017, pp. 328–331.

[5] Dev Saxena. *RocketTrajectory: A Python-based simulation of rocket trajectories.* GitHub repository. 2024. URL: https://github.com/devsaxena974/RocketTrajectory (visited on 11/28/2024).

[6] Massachusetts Institute of Technology. *Unified Engineering: Fluid Mechanics, Structural Mechanics, Materials and Dynamics.* https://ocw.mit.edu. https://ocw.mit.edu/courses/16-01-unified-engineering-i-ii-iii-iv-fall-2005-spring-2006/af59df7c34ec80675d75c0bc2ac61e02_spl2.pdf. 2005.