**Analyzing Projectile Motion with Air Resistance**

**Introduction**

When I first approached this project, I knew that adding air resistance to projectile motion would complicate things significantly. Still, I wanted to challenge myself and push my understanding of dynamics and MATLAB programming. Instead of just calculating the simple parabolic path of a projectile, I wanted to consider real-world conditions, like how air resistance slows down objects depending on their velocity. To make this project meaningful, I also thought about how to break the problem into manageable parts, starting with basic equations of motion and building up to a complete simulation.

**Defining My Problem**

I asked myself: what do I need to simulate this motion accurately? First, I set out the main components—a mass, an initial velocity, and an angle for the launch. I reminded myself to include air resistance as a force proportional to velocity, which I realized would require trigonometry to calculate horizontal and vertical components. Then, I thought about the ground rules for the simulation. When should it stop? It seemed obvious to stop when the projectile hit the ground, but I also needed to account for a maximum simulation time to avoid endless loops.

I realized that these initial questions would guide how I wrote the code. I needed the program to be simple yet flexible enough to handle changes like different masses, initial angles, or air resistance coefficients. Keeping everything modular and logical would save me headaches later.

**Setting Up the Simulation**

To get started, I told myself to keep the parameters clear. I assigned values for the mass (1 kg seemed reasonable), gravity (9.81 m/s² as usual), and air resistance (a coefficient of 0.1). For the launch, I decided on an initial height of 2 meters and a velocity of 40 m/s at a 60-degree angle. I deliberately kept the time step small (0.01 seconds) because I wanted precision in the trajectory calculations.

One thing I reminded myself over and over was to pay attention to units. I worked in meters, seconds, and degrees, and I knew MATLAB defaulted to radians for trigonometric functions. So, whenever I calculated components like the horizontal or vertical velocity, I made sure to use the cosd and sind functions to avoid mistakes.

**Writing the Code: A Step-by-Step Approach**

I approached the code methodically, starting with basic initialization. I defined vectors for time, position, velocity, and acceleration. To track motion, I created a loop that updated these values at each time step using Newton’s laws. The air resistance forces, I realized, depended not just on velocity but also on its direction. I calculated the net forces in the x and y directions separately, accounting for both air resistance and gravity in the vertical direction.

As I coded, I kept asking myself, “Does this make sense physically?” For example, I double-checked that my forces had the correct signs—air resistance always opposes motion, and gravity always pulls downward. If something felt off, I debugged it by printing intermediate values. It was tedious but worth it because I caught small errors like forgetting to square a term or mixing up a variable name.

**Visualizing the Results**

The first time I ran the code, I was relieved it didn’t crash. But then I looked at the plot—something seemed wrong. The trajectory was distorted because the axes weren’t scaled equally. I adjusted this by using MATLAB’s axis equal command, which made the trajectory look much more realistic.

I wanted the graph to tell a complete story, so I labeled the axes as “Horizontal Position (m)” and “Height (m)” and added a title: “Projectile Motion with Air Resistance.” Then, I annotated key points like the peak height and the final range. Seeing these details on the plot made the simulation feel more complete.

**Reflecting on My Challenges**

One of the trickiest parts was balancing precision and runtime. When I reduced the time step for more accuracy, the simulation took longer to run. I reminded myself to find a balance—it didn’t need to be perfect; it just needed to be good enough to visualize the effects of air resistance.

Another challenge was making the code adaptable. I knew I might want to test different conditions later, so I structured the parameters in a way that made them easy to modify. For instance, if I wanted to simulate a heavier object or a steeper launch angle, I could change just a few lines without breaking the whole program.

**What I Learned**

This project taught me to think critically about every step of the modeling process. From defining the problem to interpreting the results, I had to constantly check my assumptions and make sure the math aligned with reality. I also realized how powerful MATLAB is for this kind of work—it made visualizing and tweaking the simulation surprisingly straightforward once I got the hang of it.

Most importantly, I came away with a deeper appreciation for the complexity of real-world motion. Adding air resistance turned a simple physics problem into something much richer and more interesting. It was frustrating at times, but in the end, I was proud of what I had created—a simulation that felt both realistic and rewarding to build.

**Conclusion**

Looking back, I can see how much I’ve grown from this project. By breaking the problem into manageable parts and tackling it one step at a time, I managed to create a simulation that not only works but also gives meaningful insights into projectile motion. Next, I want to explore how to incorporate terrain or even simulate motion in 3D. For now, though, I’ll let myself enjoy the satisfaction of a job well done.