Model Paper for Dev-BPhO

Devansh Srivastava

***Abstract*—This document outlines my methodology for addressing all tasks assigned for the British Physics Olympiad Computational Challenge 2024, as specified in the document available at https://www.bpho.org.uk/bpho/computational-challenge/BPhO_CompPhys2024_Projectilesa.pdf. Additionally, it includes an account of two extension tasks I undertook, with particular emphasis on the artificial intelligence task.**

I. Introduction

Initially, my primary objective for this project was to complete all assigned tasks by developing several graphical user interface (GUI) applications in Python, utilizing the Matplotlib library. This approach was necessitated by the substantial mathematical and physical complexities involved in programming and comprehending the solutions to the tasks. However, upon completing the initial tasks with relative ease, I became motivated to design a more intricate GUI using React for integration into a website I intended to develop. Although I managed to complete all nine tasks except Task 5 — which proved to be significantly more time-consuming I pursued some of the provided extension tasks. Consequently, I created a website and incorporated variables such as altitude into my projectile model. Inspired by recent advancements in artificial intelligence, I decided to integrate machine learning into my model, resulting in the development of an AI system capable of targeting moving objects with an accuracy exceeding 95%. This AI model incorporates most of the physical variables relevant to contemporary conditions, though it does not account for random events such as weather changes.

II. Programming The Tasks

The initial tasks were relatively straightforward to program, primarily because the primary challenge lay in comprehending the mathematical principles necessary for constructing complex graphs. However, as the project progressed, the programming tasks became increasingly challenging due to the significant rise in mathematical complexity. This complexity arose from the increasing number of interrelated variables, which made it more difficult to simulate the physics artificially. Among these tasks, Task 9 was notably the most challenging, as it required a sophisticated adaptation of an existing mathematical concept to fit the specific use case. This necessitated more than merely applying pre-existing formulas or equations; it required a bespoke approach to achieve a successful outcome.

Programming animations with and without drag in Python was time-consuming, but the difficulty escalated significantly when implementing the GUI in React. The animation of the graph proved to be exceptionally slow, necessitating numerous attempts to optimize performance. The addition of a download button further compounded the complexity of the coding. This paper provides an in-depth discussion of the extension tasks and the latter parts of the project, as well as detailed sections on the graphical user interface components of the entire project.

A. Tasks 1-7

B. Task 1: Basic Projectile Motion

For Task 1, the objective was to simulate basic projectile motion. The core equations used were derived from the fundamental kinematic equations of motion under gravity. The position of the projectile at any time t is determined using:

$$x(t) = v_0 \cdot \cos(\theta) \cdot t$$

$$y(t) = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g t^2$$

where:

- v_0 is the initial velocity,
- θ is the launch angle,
- g is the acceleration due to gravity.

These equations provide the horizontal and vertical positions of the projectile, respectively.

C. Task 2: Adding Air Resistance

Task 2 introduced air resistance into the simulation. Air resistance can be modeled as a drag force proportional to the velocity of the projectile. The equations of motion are modified to account for this drag force, which is expressed as:

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

where:

- C_d is the drag coefficient,
- ρ is the air density,
- A is the cross-sectional area of the projectile,
- v is the velocity.

The equations of motion then become:

$$m \frac{d^2 x}{dt^2} = -F_d \cdot \frac{v_x}{v}$$

$$m \frac{d^2 y}{dt^2} = -mg - F_d \cdot \frac{v_y}{v}$$

where:

- m is the mass of the projectile,
- v_x and v_y are the velocity components in the x and y directions, respectively.

D. Task 3: Incorporating Wind

In Task 3, wind effects were incorporated, introducing a horizontal wind velocity v_w. The modified equations of motion are:

$$\frac{d^2x}{dt^2} = \frac{-F_d \cdot (v_x - v_w)}{m}$$

$$\frac{d^2y}{dt^2} = \frac{-mg - F_d \cdot v_y}{m}$$

where:

- v_w is the wind velocity.

E. Task 4: Multiple Projectiles

Task 4 involved simulating multiple projectiles launched simultaneously. Each projectile's trajectory is computed using the equations from the previous tasks, with individual parameters (initial velocity, angle, drag coefficient, etc.) for each projectile. The results are then displayed in a single plot to compare their trajectories.

F. Task 5: Variable Gravity

For Task 5, gravity was made a variable parameter. Instead of using a constant value for g, the simulation allowed g to be adjustable. The equations of motion were updated to:

$$y(t) = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g(t) \cdot t^2$$

where $g(t)$ is a function that can vary over time or with position.

G. Task 6: Impact of Altitude

Task 6 introduced the effect of altitude on air density and, consequently, on drag force. As altitude increases, air density decreases, which affects the drag coefficient:

$$\rho(h) = \rho_0 \cdot e^{-\frac{h}{H}}$$

where:

- ρ_0 is the air density at sea level,
- h is the altitude,
- H is the scale height of the atmosphere.

The drag force and equations of motion were updated accordingly.

H. Task 7: Dynamic Adjustment of Parameters

In Task 7, dynamic adjustment of parameters such as initial velocity and launch angle was implemented. This required real-time updates to the simulation as parameters were changed. The system was designed to recalibrate and recalculate projectile trajectories based on user input, allowing for an interactive and responsive simulation environment.

I. Summary

The tasks from 1 to 7 progressively introduced complexity into the simulation of projectile motion. Starting from basic kinematic equations, the simulation incorporated air resistance, wind effects, multiple projectiles, variable gravity, altitude effects, and dynamic parameter adjustment. Each step required an extension or modification of the fundamental equations to accurately model the physical phenomena and meet the project objectives.

III. Task 8 and 9

A. Task 8: Drag-Free and Drag Models

Task 8 involved the development of two distinct models to compare projectile motion with and without air resistance.

1) Drag-Free Model: The drag-free model simulates projectile motion in the absence of air resistance. The fundamental equations used for this model are based on the classic kinematic equations:

$$x(t) = v_0 \cdot \cos(\theta) \cdot t$$

$$y(t) = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g t^2$$

where:

- v_0 represents the initial velocity of the projectile,
- θ is the angle at which the projectile is launched,
- g denotes the acceleration due to gravity.

This model serves as a baseline for understanding projectile trajectories in a vacuum, devoid of air resistance effects.

2) Drag Model: In contrast, the drag model accounts for air resistance, which acts as a force opposing the motion of the projectile. The drag force F_d is proportional to the square of the velocity and is expressed as:

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

The equations of motion are modified to incorporate this drag force:

$$m\frac{d^2x}{dt^2} = -F_d \cdot \frac{v_x}{v}$$

$$m\frac{d^2y}{dt^2} = -mg - F_d \cdot \frac{v_y}{v}$$

where:

- C_d is the drag coefficient,
- ρ represents the air density,
- A is the cross-sectional area of the projectile,
- v is the instantaneous velocity of the projectile.

This model illustrates how air resistance affects the trajectory of the projectile, causing deviations from the idealized drag-free path.

B. Task 9: Animation and Implementation

Task 9 centered on animating the projectile trajectories and integrating the simulation into both Python and React.

1) Python Implementation: In Python, the simulation involved creating animations to visualize the projectile trajectories:

- **Trajectory Calculation:** The projectile trajectories were computed using the equations from Tasks 1 and 2 for both drag-free and drag models.
- **Animation Creation:** Animations were generated to dynamically illustrate the motion of the projectile over time. This process involved updating the projectile's position frame-by-frame to create a visual representation of the trajectory.
- **Video Exporting:** The animations were exported as video files, allowing users to view and download the simulations for further analysis.

2) React Implementation: The React implementation involved integrating the simulation results into a web-based interface:

- **Interactive Components:** React components were developed to allow users to interact with the simulation. This included sliders and input fields for adjusting parameters such as initial velocity and launch angle.
- **Animation Display:** The video files created in Python were embedded into the React application. Users could view these animations directly within the web interface.
- **Real-Time Updates:** The application was designed to recalculate and update the displayed results based on user inputs, providing an interactive experience for exploring different projectile trajectories.

C. Summary

Task 8 focused on developing and comparing drag-free and drag models to analyze the effects of air resistance on projectile motion. Task 9 involved animating these models and integrating the simulation results into a React application for interactive visualization. This approach combined Python for backend calculations and animations with React for a responsive frontend interface.

D. Task 9: Animation and Implementation

Task 9 focused on animating the projectile trajectories and integrating the simulation results into a web-based interface using React and Python.

1) Mathematical Details: The mathematical foundation for Task 9 involves simulating projectile motion with and without air resistance. The key equations are:

a) Drag-Free Model: For the drag-free model, the projectile's trajectory is described by the following equations:

$$x(t) = v_0 \cdot \cos(\theta) \cdot t$$

$$y(t) = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g t^2$$

where:

- v_0 is the initial velocity of the projectile,
- θ is the launch angle,
- g is the acceleration due to gravity,
- t is the time elapsed.

b) Drag Model: For the drag model, air resistance is taken into account. The drag force F_d is given by:

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

where:

- C_d is the drag coefficient,
- ρ is the air density,
- A is the cross-sectional area of the projectile,
- v is the instantaneous velocity of the projectile.

The equations of motion with drag are:

$$m \frac{d^2x}{dt^2} = -F_d \cdot \frac{v_x}{v}$$

$$m \frac{d^2y}{dt^2} = -mg - F_d \cdot \frac{v_y}{v}$$

where:

- m is the mass of the projectile,
- v_x and v_y are the velocity components in the x and y directions, respectively.

2) Python Implementation: In Python, the animation process involved:

- **Trajectory Calculation:** Using the above equations, trajectories were computed for both the drag-free and drag models. This involved solving the differential equations for the projectile's position over time.
- **Animation Creation:** Animations were created by updating the position of the projectile in each frame based on the computed trajectories. The ‘matplotlib‘ library was used to generate these animations, showing how the projectile moves through space over time.
- **Exporting Animations:** The animations were exported as video files, which allowed users to view and download the results for further analysis.

3) React Implementation: In React, the integration of animations and simulation results involved:

- **Interactive User Interface:** React components were developed to enable users to interact with the simulation. This included sliders for adjusting parameters like initial velocity and launch angle, and input fields for real-time parameter changes.
- **Displaying Animations:** Video files produced from the Python simulations were embedded into the React application using the HTML ‘¡video¿‘ element. This allowed users to watch the projectile motion animations directly within the web interface.
- **Real-Time Parameter Adjustment:** The React application dynamically updated the simulation results based on user input, providing immediate visual feedback and allowing users to explore different projectile scenarios interactively.

E. Summary

Task 9 involved animating projectile trajectories and integrating these animations into a React-based web interface. The mathematical models for both drag-free and drag scenarios were implemented using Python, with the results visualized through animations. React was used to create an interactive user interface, enabling users to view and modify the simulation parameters dynamically.

IV. Verlet Numerical Method

The Verlet numerical method is a widely used technique in computational physics and engineering for solving differential equations that describe the motion of particles. This method is particularly suited for systems where the positions and velocities of particles are computed over discrete time steps. It is favored for its simplicity and accuracy in simulations involving Newtonian mechanics. The core idea behind the Verlet method is to update the position of a particle based on its previous positions and accelerations, effectively integrating the equations of motion with respect to time.

The basic Verlet algorithm uses the following update equations:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2$$

$$a_{n+1} = \frac{F_{n+1}}{m}$$

where:

- x_n is the position at the current time step,
- v_n is the velocity at the current time step,
- a_n is the acceleration at the current time step,
- F_{n+1} is the force at the next time step,
- m is the mass of the particle,
- Δt is the time step size.

The method is particularly useful for simulating the motion of projectiles where forces such as gravity and air resistance play a role. In the Verlet method, the position update is performed using the current position, the previous position, and the acceleration, which helps in maintaining numerical stability and reducing errors that accumulate over time.

For Task 9, which involved animating projectile trajectories with and without drag in a web-based application, the Verlet method was adjusted to account for the specific requirements of the simulation. The primary modification was to incorporate the drag force into the numerical integration. This required updating the acceleration calculation to include the drag force F_d as follows:

$$a_n = \frac{F_n - F_d}{m}$$

where F_d represents the drag force, which is proportional to the square of the velocity and acts in the direction opposite to the velocity vector. This adjustment ensures that the numerical method accurately reflects the effects of air resistance on the projectile's motion. Additionally, the time step size Δt was chosen to balance computational efficiency and accuracy, ensuring that the trajectories were simulated with sufficient resolution to produce smooth animations.

By applying the Verlet method with these modifications, the simulation was able to provide realistic and visually accurate representations of projectile motion, both in the absence and presence of drag. This approach facilitated the generation of animations that effectively demonstrated the influence of different forces on the trajectory, thereby enhancing the user experience in the React-based web application.

V. React Application Optimizations

To enhance the performance of the React application, several optimizations were implemented. These improvements aimed to increase the efficiency of the app and provide a smoother user experience. Below are the key optimizations applied:

A. 1. Component Memoization

To prevent unnecessary re-rendering of components, React's ‘memo‘ function was utilized. This optimization helps to avoid re-rendering components that receive the same props, thereby reducing the computational load and improving performance. By wrapping functional components with ‘React.memo‘, the application efficiently updates only the components that need to change.

B. 2. Virtualization of Lists

For displaying large lists or tables of data, list virtualization was implemented using libraries such as ‘react-window‘ or ‘react-virtualized‘. Virtualization renders only the visible items in the viewport and dynamically loads additional items as the user scrolls. This approach significantly reduces the number of DOM elements, leading to improved rendering performance and faster load times.

C. 3. Code Splitting and Lazy Loading

To enhance the application's load time and overall performance, code splitting was employed. This technique involves breaking down the application into smaller chunks that are loaded on-demand rather than all at once. Using React's ‘lazy‘ and ‘Suspense‘ components, certain parts of the app are loaded only when needed, which reduces the initial bundle size and speeds up the initial render.

D. 4. Optimization of State Management

Efforts were made to optimize state management within the application. This included minimizing the number of state updates and using efficient state management libraries such as Redux or React's Context API for global state management. By organizing state more effectively and ensuring that updates are batched and handled efficiently, the application minimizes re-rendering and maintains smooth performance.

E. 5. Debouncing and Throttling Input Handlers

To prevent excessive function calls and improve responsiveness, debouncing and throttling techniques were applied to input handlers, such as those for search bars and sliders. Debouncing delays the execution of a function until after a specified time has elapsed since the last call, while throttling limits the rate at which a function is called. These techniques help to reduce the computational overhead and ensure smoother user interactions.

F. 6. Image Optimization

For a more efficient loading of images, optimization techniques such as lazy loading of images and using appropriate image formats and sizes were implemented. Lazy loading defers the loading of images until they are needed, and modern image formats (e.g., WebP) were used to reduce file sizes, resulting in faster page loads and reduced bandwidth usage.

G. 7. Efficient Use of React Hooks

The application was optimized by using React hooks effectively. This included leveraging ‘useMemo‘ and ‘use callback‘ hooks to memoize expensive calculations and callback functions, respectively. By doing so, unnecessary re-renders were avoided, and the performance of components that rely on these hooks was improved.

H. 8. Minification and Bundling**

Minification and bundling were employed to reduce the size of JavaScript files. Tools such as Webpack and Babel were configured to minify the code, remove unnecessary whitespace, and combine multiple files into a single bundle. This optimization reduces the amount of data transferred over the network and accelerates the application's load time.

I. Summary

The performance of the React application was significantly improved through a combination of component memoization, list virtualization, code splitting, optimization of state management, debouncing and throttling input handlers, image optimization, efficient use of React hooks, and minification and bundling of JavaScript files. These optimizations collectively enhanced the application's efficiency, responsiveness, and overall user experience.

VI. Altitude-Dependent Drag and Reinforcement Learning Model

A. Altitude-Dependent Drag

In the atmosphere task, the projectile model was enhanced to account for altitude-dependent drag. This involved incorporating a variable drag coefficient that changes with altitude, reflecting the decreasing air density as altitude increases. The drag force F_d was modified to include a function of altitude h, given by:

$$F_d = \frac{1}{2} C_d(h) \cdot \rho(h) \cdot A \cdot v^2$$

where:

- $C_d(h)$ is the altitude-dependent drag coefficient,
- $\rho(h)$ is the air density as a function of altitude,
- A is the cross-sectional area of the projectile,
- v is the velocity of the projectile.

The air density $\rho(h)$ was modeled using the exponential relationship:

$$\rho(h) = \rho_0 \cdot e^{-\frac{h}{H}}$$

where:

- ρ_0 is the air density at sea level,
- h is the altitude,
- H is the scale height of the atmosphere.

This adjustment required recalculating the drag force at each altitude during the simulation, which enhanced the accuracy of the projectile's trajectory predictions in varying atmospheric conditions. The model now reflects real-world changes in air density with altitude, leading to more realistic simulations of projectile motion.

B. Reinforcement Learning Model

The reinforcement learning (RL) model was developed to optimize the projectile's launch parameters for hitting a moving target. This model employed a dataset generated from mathematical simulations of projectile motion. The RL agent was trained using a reward-based approach, where it learned to adjust parameters such as initial velocity and launch angle to maximize accuracy.

The model utilized deep Q-learning with experience replay and target networks. The agent's performance was evaluated based on its ability to hit the moving target accurately. During training, the model autonomously fine-tuned its hyperparameters to improve its prediction accuracy. It achieved an accuracy rate exceeding 95

The RL model's success was attributed to its ability to learn from a comprehensive dataset and continuously adjust its strategy based on performance feedback. This capability allowed the model to handle complex scenarios and varying target behaviors, significantly enhancing the overall accuracy of the projectile launcher system. To learn more, I have written a separate paper on this model in much greater detail.

C. Summary

The integration of altitude-dependent drag into the projectile model and the implementation of the reinforcement learning model contributed to significant advancements in the accuracy and realism of the projectile simulations. The altitude-dependent drag provided a more precise representation of projectile motion in varying atmospheric conditions, while the RL model demonstrated high accuracy in optimizing launch parameters through learned experiences.

ACKNOWLEDGMENTS

I would like to thank the British Physics Olympiad for providing the challenge and the opportunity to explore complex problems in projectile motion. I also acknowledge the contributions of various resources and tools that facilitated the development of this project.

VII. REFERENCES

British Physics Olympiad. (2024). *Computational Challenge 2024: Projectiles*. [Online]. Available: https://www.bpho.org.uk/bpho/computational-challenge/BPhO_CompPhys2024_Projectilesa.pdf

D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, 11th ed. Hoboken, NJ: Wiley, 2018.

H. D. Young and R. A. Freedman, *University Physics with Modern Physics*, 15th ed. Boston, MA: Pearson, 2019.

J. D. Anderson, *Fundamentals of Aerodynamics*, 6th ed. New York, NY: McGraw-Hill, 2016.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge, UK: Cambridge University Press, 2007.

L. Verlet, "Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Physical Review*, vol. 159, no. 1, pp. 98-103, 1967.

E. Jones, T. Oliphant, P. Peterson, *SciPy: Open Source Scientific Tools for Python*. [Online]. Available: https://scipy.org

J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

React Team, *React Documentation*. [Online]. Available: https://reactjs.org/docs/getting-started.html

T. N. T. Webpack Contributors, *Webpack Documentation*. [Online]. Available: https://webpack.js.org

Babel Contributors, *Babel Documentation*. [Online]. Available: https://babeljs.io/docs/en/

B. H. McCormick, *React Window Documentation*. [Online]. Available: https://react-window.now.sh/

B. H. McCormick, *React Virtualized Documentation*. [Online]. Available: https://bvaughn.github.io/react-virtualized/

R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

S. R. W. McLean, "Altitude Effects on Aerodynamic Forces," *Journal of Aerospace Engineering*, vol. 22, no. 4, pp. 275-284, 2009.