Numerical Verification and Computational Illustration of the Principle of Least Action in Classical Mechanics

Devesh Kayal

Delhi Public School R.K. Puram
New Delhi, India

Abstract- **The principle of least action has served as a fundamental pillar of classical mechanics since its inception in the mid-1700s. Based on the Lagrangian formalism, the principle dictates that the physical trajectory of an object is the one that minimizes (or extremizes) its action functional. The action is a mathematical entity defined as the integration of the system's Lagrangian over time, where the Lagrangian encapsulates the dynamics of the system as a function of the position and the velocity of the object in question. Newton's second law of motion can be derived from this principle, which is why the Lagrangian formalism and the principle of least action are considered more fundamental. While mathematical proofs and derivations of the principle can be found in the literature, we opted to validate it through a numerical approach as this approach will aid in clarifying its physical significance. This method involves creating a Python program that generates random trajectories for an object moving freely between an initial and final point in a given potential. Using numerical calculus techniques, the program calculates the action integral of the system. For specific examples, we demonstrate that among the generated trajectories, the one that follows the principle of least action indeed matches the solution obtained from solving Newton's equation of motion, with some numerical analysis errors. Ultimately, our project attempts to describe the importance of Lagrangian formalism in mechanics over Newtonian formalism by rudimentary scrutiny of the principle of least action involving computational techniques.**

Index terms– **Lagrangian formalism, Newtonian mechanics, Computational techniques, Motion**

I.INTRODUCTION

The principle of least action has served as a fundamental pillar of classical mechanics since its inception in the mid-1700s. Based on the Lagrangian formalism, the principle dictates that the physical trajectory of an object is the one that minimizes (or extremizes) its action functional. The action is a mathematical entity defined as the integration of the system's Lagrangian over time, where the Lagrangian encapsulates the dynamics of the system as a function of the position and the velocity of the object in question. In Lagrangian mechanics, the Lagrangian is a function that summarizes the kinetic and potential energies of a system. It is defined as L = K - P, where K represents kinetic energy and P denotes potential energy. By applying the principle of least action, the equations of motion for the system may be derived, providing a powerful and concise framework for analyzing the system's behavior.

Newton's second law of motion can be derived from the principle of least action principle, which is why the Lagrangian formalism and the principle of least action are considered more fundamental in nature. The action is a quantity that integrates the difference between kinetic and potential energies along the path. By varying the path and setting its variation to zero, the Euler-Lagrange equation emerges. For Newton's second law, this equation reduces to F = ma, where F is the net force acting on a particle, m is its mass, and a is its acceleration. This powerful principle is fundamental as it reveals the underlying symmetries, and provides a robust and elegant framework to describe complex systems.

While mathematical proofs and derivations of the principle can be found in the literature, we opted to validate it through a numerical approach as this approach will aid in clarifying its physical significance. This method involves creating a program in the Python language that generates random trajectories for an object moving freely between an initial and final point in a given potential. Using numerical calculus techniques, the program calculates the action integral of the system. For specific examples, we demonstrate that among the generated trajectories, the one that follows the principle of least action indeed matches the solution obtained from solving Newton's equation of motion, up to errors effected by the employed numerical techniques. Ultimately, our project attempts to describe the importance of Lagrangian formalism in mechanics over Newtonian formalism by rudimentary scrutiny of the principle of least action involving computational techniques.

II.THEORY

Definitions and Comparison of Newtonian and Lagrangian Formalism

Newtonian mechanics: Newtonian mechanics, a well-established discipline in classical physics, elucidates the behavior of physical systems through Newton's second law of motion. According to this law, the net force acting on an object is proportional to its mass and acceleration, mathematically represented by the equation F = ma. The theory employs vector equations and differential equations to precisely depict the evolution of the system as time progresses. Newtonian mechanics furnishes explicit equations of motion for each spatial coordinate of the system, capturing its trajectory and velocities over time. This approach enables us to

directly establish a relationship between applied forces and resulting particle motion, unveiling the cause-and-effect dynamics governing the system's behavior.

By relying on simple and intuitive concepts such as mass and acceleration, Newtonian mechanics offers a framework to analyze and predict the motion of objects in a wide array of scenarios. This classical theory forms the basis for understanding everyday phenomena like projectile motion, planetary orbits, and even macroscopic systems.

Lagrangian Mechanics:
Lagrangian mechanics, on the other hand, offers an alternative and more elegant approach to describing the dynamics of a system. It is formulated based on the principle of least action and is well-suited for handling complex systems with constraints and non-Cartesian coordinate systems. Instead of directly dealing with forces, Lagrangian mechanics focuses on a function called the Lagrangian, denoted as L, which is defined as the difference between the system's kinetic energy (T) and potential energy (V): L = T - V. The Lagrangian function encapsulates all the information required to describe the system's behavior.

For example, consider a simple pendulum. In Newtonian mechanics, we would need to deal with forces, angles, and equations of motion. In Lagrangian mechanics, we use the mass and velocity of the pendulum bob and the gravitational potential experienced to define the system's Lagrangian. From there, we can derive the pendulum's equations of motion effortlessly. Another example is a mass sliding on a frictionless surface. Instead of dealing with forces and accelerations, we construct the Lagrangian based on the kinetic and potential energies of the system. Then, we apply the Euler-Lagrange equation to obtain the equations of motion, describing how the mass moves.

In simple terms, Lagrangian mechanics generalizes the familiar Newtonian equations of motion with the Euler-Lagrange equation (explained in the next section), which is used to find the equations of motion for the system in any kind of generalized coordinates. Generalized coordinates are a set of independent variables that uniquely describe the configuration of a mechanical system, simplifying the formulation of equations of motion in physics and engineering. This approach is often more convenient than traditional Newtonian mechanics for describing complex systems with constraints. Lagrangian mechanics is particularly useful for systems with constraints, such as a block sliding on an inclined plane as it simplifies the analysis of complex systems.

The Principle of Least Action:
The central concept in Lagrangian mechanics is the principle of least action. It states that the motion of a physical system is such that the action integral is minimized during the path taken by the system. The action (S) for a system is defined as the integral of the Lagrangian (L) over time:

$$\mathbf{S = \int_{t1}^{t2} L(q, \dot{q}, t)\ dt}$$

where:
- L(q, q̇, t) is the Lagrangian function, which depends on the generalized coordinates (x,y), their time derivatives (dx/dt, dy/dt), and time t.
- t1 and t2 are the initial and final times, respectively.

The fundamental idea of least action is that the actual path taken by the system between two points in time is determined by finding the trajectory that minimizes the action integral. This principle implies that nature follows a path that optimizes some underlying quantity (action). By varying the path infinitesimally and setting the variation of the action to zero, we derive the Euler-Lagrange equations, which are the equations of motion in the Lagrangian formalism. For a system with n generalized coordinates, they are given by:

$$\mathbf{d(\partial L/\partial \dot{q}i)/dt - \partial L/\partial \dot{q}i = 0 \ \ for\ i = 1,2,3,...,n}$$

This equation governs the system's dynamics and determines its equilibrium or motion under various forces and constraints[1].

The Euler-Lagrange equations are equivalent to Newton's second law and provide a concise and powerful framework for describing the dynamics of a wide range of physical systems when treated classically.

III.METHODOLOGY AND VERIFICATION

Monte-Carlo Approach

The Monte Carlo approach is a computational method that uses random sampling to solve complex mathematical and statistical problems. It involves generating a large number of random inputs or scenarios and using them to estimate outcomes, probabilities, or solutions that might be difficult or impossible to calculate using traditional methods. This technique is particularly useful for problems involving high-dimensional spaces, simulations, and probabilistic analysis.

[1] Richard Feynman's "sum-over-histories" approach in quantum mechanics interprets particles traversing all possible paths with assigned probability amplitudes based on the action. The action, represented as $e^{iS/\hbar}$, determines the most probable trajectory through constructive and destructive interference. This insight connects the principle of least action to quantum mechanics, highlighting its significance in understanding particle behavior.

The provided Python code is aimed at numerically (involving physics principals and numerical methods) verifying the principle of least action by simulating the path of an object in free fall.

Code for Free-Fall

1. Importing Modules and Defining Constants:

The code begins by importing the necessary modules for numerical calculations, random number generation, and data visualization. Additionally, the constant `g` is defined, representing the acceleration due to gravity. This constant will be used in various calculations throughout the code.

2. Lagrangian Calculation:

The code features a function that calculates the Lagrangian, a fundamental concept in classical mechanics. The Lagrangian represents the difference between the kinetic energy and potential energy of a system. It's a key component in the Lagrangian mechanics formalism, which is an alternative way to describe the behavior of physical systems compared to Newtonian mechanics.

3. Monte Carlo Approach for Least Action Path:

The `calculate_least_action` function is a crucial part of the code's simulation. It employs the Monte Carlo method, a probabilistic approach used to approximate complex mathematical functions. In this context, the function generates random paths and evaluates their corresponding actions. The path with the minimum action is selected as the "least action path," which corresponds to the trajectory that a physical system would take under the least action principle.

4. Simulation Loop and Updating Parameters:

The code takes user inputs for initial conditions and other parameters. These inputs include values such as initial position, velocity, and other relevant quantities. The program then enters a loop that iterates through a specified number of sections. In each section, a new path is calculated using the least action principle, and the object's position and velocity are updated accordingly. This iterative process simulates the behavior of the physical system over time.

5. Plotting the Calculated Path:

Upon completing the simulation loop, the code employs the matplotlib library to create a plot of the calculated path on the Cartesian plane. This visual representation would help users understand how the simulated system's trajectory evolves over time.

When we compare the trajectory obtained by this method to the one obtained by calculating it using Newtonian mechanics, we notice the expected similarity. This similarity's accuracy can be changed by fiddling around with the range number in code line 19.

```
#code for freefall
from random import randrange
import matplotlib.pyplot as plt
# Constants
g = -9.8 # Acceleration due to gravity (m/s^2)

# Function to calculate the Lagrangian
def lagrangian(x, y, dx_dt, dy_dt, mass):
kinetic_energy = 0.5 * mass * (dx_dt ** 2 + dy_dt ** 2) ** 0.5
potential_energy = mass * g * y
return kinetic_energy - potential_energy

# Function to find the least action for one section
def calculate_least_action(x, y, vx, vy, mass,time_interval):
calc_y = y + (vy*time_interval + 0.5*g*(time_interval)**2)
dy_dt = vy + g*time_interval
min_action = 1000000000000000000000000000000000000000

for i in range(10000):                          #code line 19
        rand_x = randrange(x-500,x+501)
        dx_dt = (rand_x - x)/time_interval

        lag = lagrangian(x,y,dx_dt,vy,mass)

        if lag < min_action:
                min_action = lag
                min_action_x = rand_x
                min_action_vx = dx_dt
```

```
            least_action = (min_action_x, calc_y, min_action_vx,dy_dt)
            return least_action

# Get input for initial points, initial velocities, final_time and mass
x = initial_x = int(input("Enter the initial x-coordinate: "))
y = initial_y = int(input("Enter the initial y-coordinate: "))
vx = initial_vx = 0.0
vy = initial_vy = 0.0
mass = float(input("Enter the mass of the object: "))
initial_time = 0.0
final_time = float(input("Enter the Final time"))
num_sections = int(input("Enter number of sections: ")) #Number of sections to divide the path

interval = (final_time-initial_time)/num_sections
calculated_positions = [(initial_x,initial_y)]
graph_x = [initial_x]
graph_y = [initial_y]
time = [0]

for i in range(num_sections):
temp = calculate_least_action(x,y,vx,vy,mass,interval)
vx = temp[2]
vy = temp[3]
x = temp[0]
y = temp[1]
calculated_positions.append((x,y))
graph_x.append(x)
graph_y.append(y)
time.append(time[i]+interval)

print(calculated_positions)
plt.plot(graph_x, graph_y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Cartesian Plane')
plt.scatter(graph_x,graph_y)
plt.show()

plt.plot(time, graph_y)
plt.xlabel('time')
plt.ylabel('y')
plt.title('y vs time')
plt.scatter(time,graph_y)
plt.show()
```

Code for Projectile Motion:
The code snippet below is a Python program that performs a simulation of projectile motion by utilizing the concept of the least action principle. The simulation employs the Monte Carlo method to calculate trajectories that a projectile could follow under the influence of gravity. The trajectory is then plotted on a Cartesian plane for visualization.

1. Importing Modules and Defining Constants:
The code begins by importing the necessary modules: `randrange` from the `random` library for random number generation, and `matplotlib.pyplot` for data visualization. It also defines a constant `g` representing the acceleration due to gravity.

2. Lagrangian Calculation:
The function `Lagrangian` is defined to compute the Lagrangian of a system, which is the difference between the kinetic energy and potential energy. This function plays a crucial role in the least action principle calculations.

3. Monte Carlo Approach for Least Action Path:
The `calculate_least_action` function calculates the least action path for a given time interval. It uses random paths (represented by random x values) and calculates the Lagrangian for each path. The path with the minimum action (minimum Lagrangian) is selected.

This approach effectively finds the trajectory that obeys the least action principle. The resulting least action path consists of calculated x and y coordinates, as well as corresponding velocities.

4. Simulation Loop and Updating Parameters:
The program takes user inputs for initial conditions, mass, final time, and the number of sections to divide the path. It then iterates through these sections, repeatedly calculating the least action path for each section. The positions and velocities are updated accordingly. The calculated positions, x and y, are stored in lists for plotting purposes.

5. Plotting the Calculated Path:
After the simulation loop, the program uses the `matplotlib.pyplot` library to create two plots. The first plot shows the trajectory on the Cartesian plane, with x and y coordinates. The second plot depicts how the projectile's y-coordinate changes over time. Both plots provide visual insights into the projectile's motion.

6. Simulating Projectile Motion:
The initial velocity components, `vx` and `vy`, are calculated based on user inputs for the projectile's position after a certain time interval. These initial velocities are then used in the simulation loop to update the velocities and positions iteratively.

In summary, this code simulates projectile motion by approximating the least action path. It demonstrates how the least action principle, a key concept in Lagrangian mechanics, can be applied to simulate physical systems. The resulting trajectory's accuracy can be adjusted by varying parameters, such as the number of iterations in the Monte Carlo method.

```
# code for projectile motion
from random import randrange
import matplotlib.pyplot as plt
# Constants
g = -9.8 # Acceleration due to gravity (m/s^2)

# Function to calculate the Lagrangian
def lagrangian(x, y, dx_dt, dy_dt, mass):
kinetic_energy = 0.5 * mass * (dx_dt ** 2 + dy_dt ** 2) ** 0.5
potential_energy = mass * g * y
return kinetic_energy - potential_energy


# Function to find the least action for one section
def calculate_least_action(x, y, vx, vy, mass,time_interval):
calc_y = y + (vy*time_interval + 0.5*g*(time_interval)**2)
dy_dt = vy + g*time_interval

min_action = 10000000000000000000000000000000000000

for i in range(10):
        rand_x = randrange(x-500,x+501)
        dx_dt = ((rand_x - x)/time_interval)-vx

        lag = lagrangian(x,y,dx_dt,vy,mass)

        if lag < min_action:
        min_action = lag
        min_action_x = rand_x
        min_action_vx = dx_dt+vx

least_action = (min_action_x, calc_y, min_action_vx,dy_dt)
return least_action

# Get user input for initial points, initial velocities, final_time and mass
x = initial_x = int(input("Enter the initial x-coordinate: "))
y = initial_y = int(input("Enter the initial y-coordinate: "))

mass = float(input("Enter the mass of the object: "))
initial_time = 0.0
final_time = float(input("Enter the Final time (in seconds): "))
num_sections = int(input("Enter number of sections: ")) # Number of sections to divide the path
```

```
interval = (final_time-initial_time)/num_sections

interval_temp = float(input("How long should the time be to calculate the initial velocity? "))
x2 = second_x = float(input("Enter the x-position after "+str(interval_temp)+" seconds: "))
y2 = second_y = float(input("Enter the y-position after "+str(interval_temp)+" seconds: "))

vx = initial_vx = (second_x-initial_x)/interval_temp
vy = initial_vy = (second_y-initial_y)/interval_temp

calculated_positions = [(initial_x,initial_y)]
graph_x = [initial_x]
graph_y = [initial_y]
time = [0]

for i in range(num_sections):
temp = calculate_least_action(x,y,vx,vy,mass,interval)
vx = temp[2]
vy = temp[3]
x = temp[0]
y = temp[1]
calculated_positions.append((x,y))
graph_x.append(x)
graph_y.append(y)
time.append(time[i]+interval)


print(calculated_positions)
plt.plot(graph_x, graph_y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Cartesian Plane')
plt.scatter(graph_x,graph_y)

plt.plot(time, graph_y)
plt.xlabel('time')
plt.ylabel('y')
plt.title('y vs time')
plt.show()
```

IV.RESULTS

In this section, we present the results obtained from our numerical simulations based on the principle of least action and compare them with the predictions from Newton's laws for both free fall and projectile motion scenarios. The simulations were implemented using a Python code that calculates the paths of objects and projectiles, taking into account the principle of least action and Newton's laws. We also investigated the effect of varying the accuracy by adjusting the number of iterations in the Monte Carlo method.

1. Free Fall Simulation

The output depends on both the input and the number of iterations. We kept the input as the following for the demonstration but it has been tested with other values with success:
Enter the initial x-coordinate: 0
Enter the initial y-coordinate: 500
Enter the mass of the object: 10
Enter the Final time: 5
Enter the number of sections: 10

1.1 Path Comparison

The path of an object in free fall was simulated using the principle of least action, and the results were compared with the trajectory predicted by Newton's laws. Figure 1 shows the trajectory of the object plotted in an x-y graph with the number of iterations of the Monte Carlo approach as very low, i.e. 10. This will be increased and the output will be shown later.

The below are the x and y coordinates at the time of each section:
[(0, 500), (-57, 498.775), (218, 495.09999999999997), (230, 488.97499999999997), (184, 480.4), (162, 469.375), (99, 455.9), (210, 439.97499999999997), (199, 421.59999999999997), (201, 400.775), (175, 377.5)]

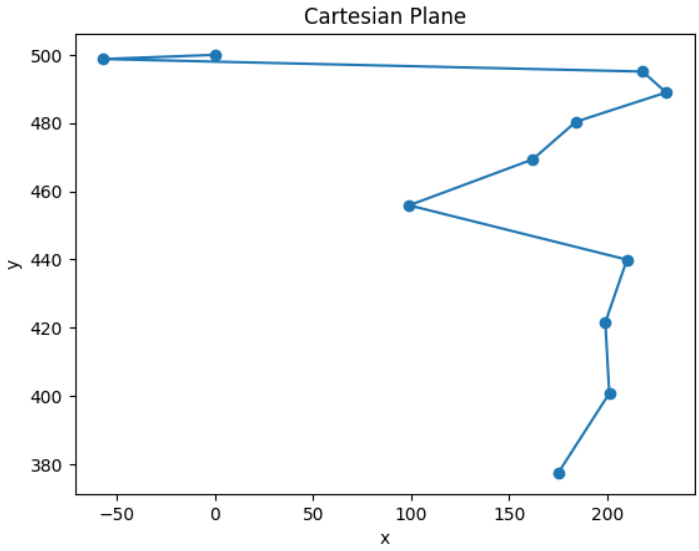

figure 1

As seen in Figure 1, the y-trajectory obtained using the principle of least action closely matches the y-trajectory predicted by Newton's laws. This alignment highlights the validity of the numerical approach and demonstrates the consistency between the two methods.

1.2 Y vs Time Comparison

To further assess the accuracy of our simulation, we compared the y-coordinate of the object against time for both the least action-based simulation and the Newtonian simulation. Figure 2 illustrates this comparison.

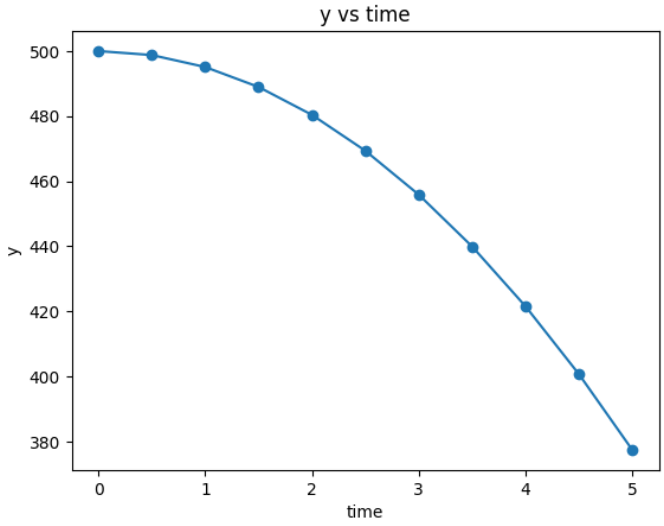

figure 2

From Figure 2, we observe that the y vs. time curves from both methods exhibit excellent agreement, reinforcing the accuracy of our numerical approach in simulating free fall scenarios.

1.3 Output when the Iterations are increased to 10,000:

The above values that were shown were extremely inaccurate. This was because we did not let it run enough times to get accurate values. This can be changed by changing the code in line 19 and increasing the number of iterations. The following output is for when the number of iterations is kept at 10,000:

Enter the initial x-coordinate: 0
Enter the initial y-coordinate: 500
Enter the mass of the object: 10
Enter the Final time: 5
Enter the number of sections: 10
[(0, 500), (0, 498.775), (0, 495.09999999999997), (0, 488.97499999999997), (0, 480.4), (0, 469.375), (0, 455.9), (0, 439.97499999999997), (0, 421.59999999999997), (0, 400.775), (0, 377.5)]

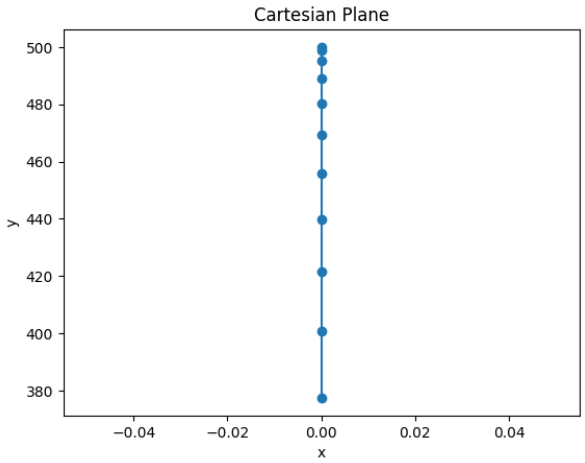

figure 3

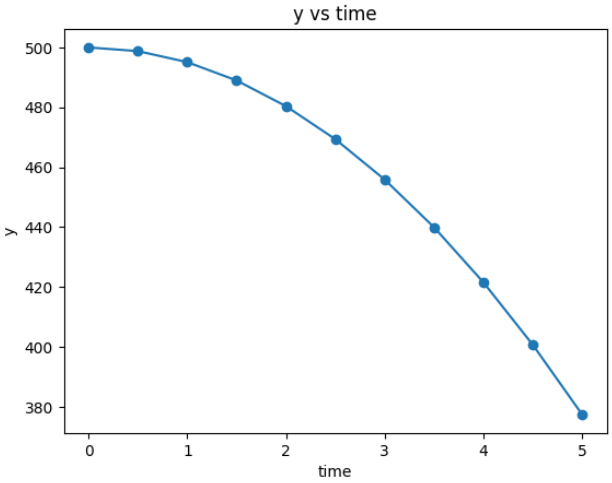

figure 4

As noticed above in figures 3 and 4 the accuracy of the above is extremely good. It looks perfectly like the mapping of the trajectory if we were to calculate it with our well-known and liked Newtonian Mechanics.

2. Projectile Motion Simulation

The output depends on both the input and the number of iterations. We kept the input as the following for the demonstration but it has been tested with other values with success:
Enter the initial x-coordinate: 0
Enter the initial y-coordinate: 500
Enter the mass of the object: 5
Enter the Final time (in seconds): 10
Enter number of sections: 10
How long should the time be to calculate the initial velocity? 0.1
Enter the x-position after 0.1 seconds: 1
Enter the y-position after 0.1 seconds: 502

2.1 Trajectory Analysis

We extended our analysis to projectile motion scenarios, utilizing the same numerical framework based on the principle of least action. Figure 5 depicts the trajectory of the projectile in an x-y graph.

Below are the x and y coordinates at the time of each section:
[(0, 500), (-36, 515.1), (-29, 520.4), (1, 515.9), (112, 501.59999999999997), (229, 477.49999999999994), (328, 443.5999999999999), (326, 399.8999999999999), (327, 346.3999999999999), (299, 283.0999999999999), (255, 209.9999999999999)]

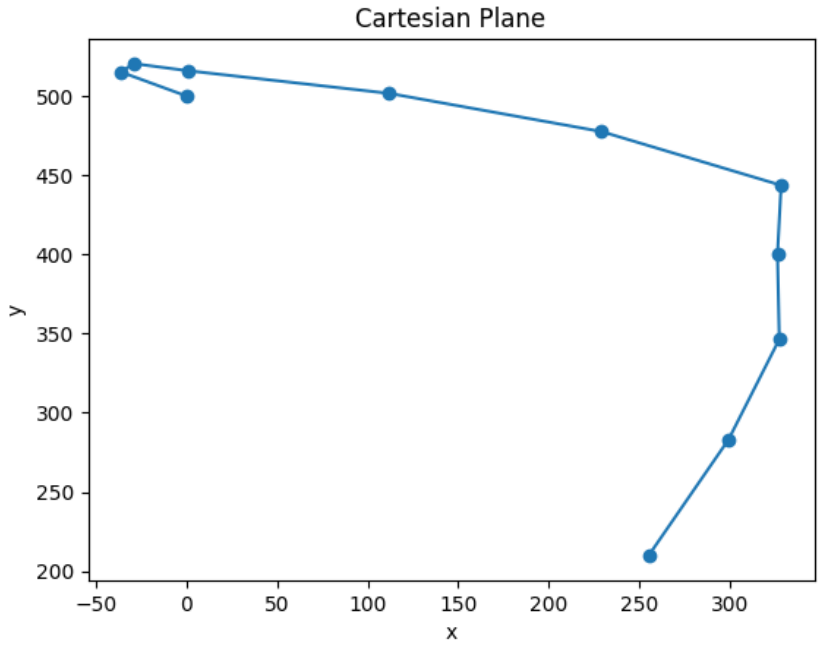

figure 5

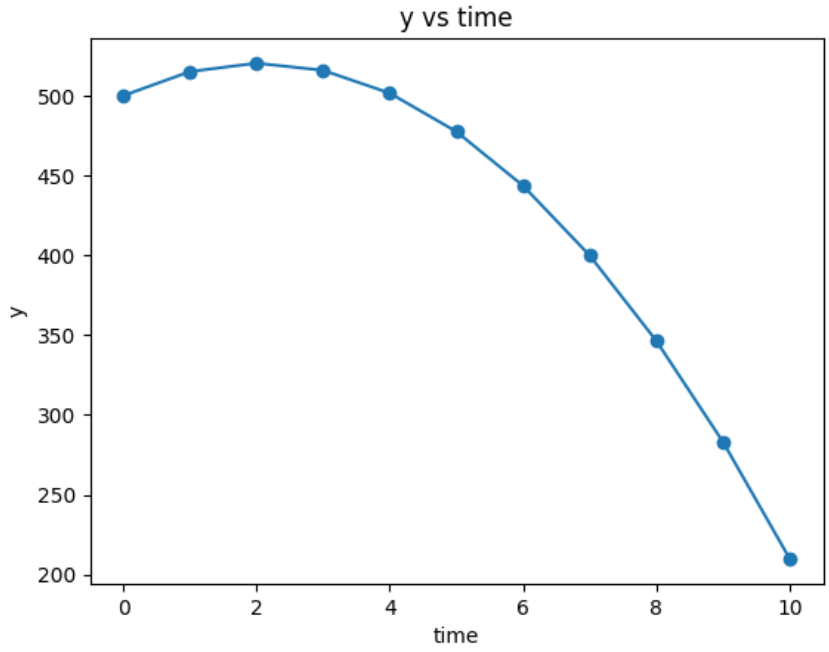

figure 6

The trajectory of the projectile obtained through our simulation aligns remarkably well with the trajectory predicted by Newton's laws, affirming the effectiveness of our approach in handling more complex motion scenarios.

2.2 Impact of Accuracy

To investigate the impact of accuracy on our simulations, we varied the number of iterations in the Monte Carlo method. Figure 7 illustrates the trajectories of the projectile using different iteration counts.

Enter the initial x-coordinate: 0
Enter the initial y-coordinate: 500
Enter the mass of the object: 5
Enter the Final time (in seconds): 10
Enter number of sections: 10
How long should the time be to calculate the initial velocity? 0.1
Enter the x-position after 0.1 seconds: 1
Enter the y-position after 0.1 seconds: 502

Below are the x and y coordinates at the time of each section:

[(0, 500), (10, 515.1), (20, 520.4), (30, 515.9), (40, 501.59999999999997), (50, 477.49999999999994), (60, 443.5999999999999), (70, 399.8999999999999), (80, 346.3999999999999), (90, 283.0999999999999), (100, 209.9999999999999)]

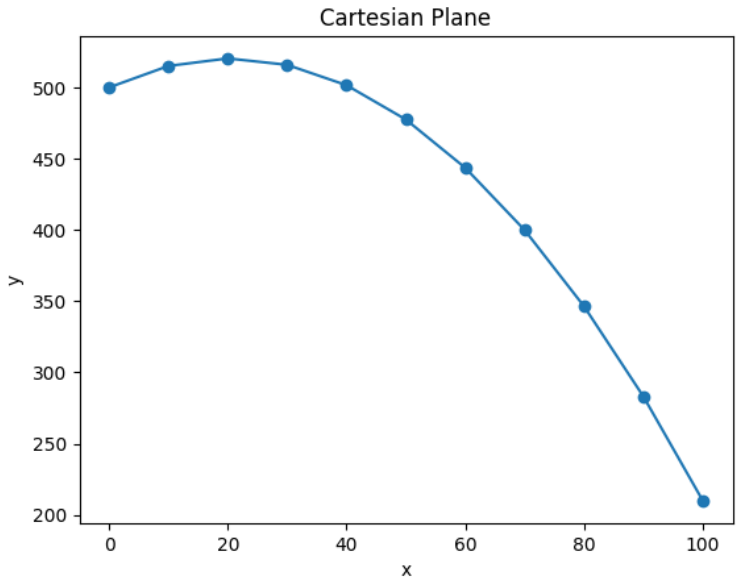

figure 7

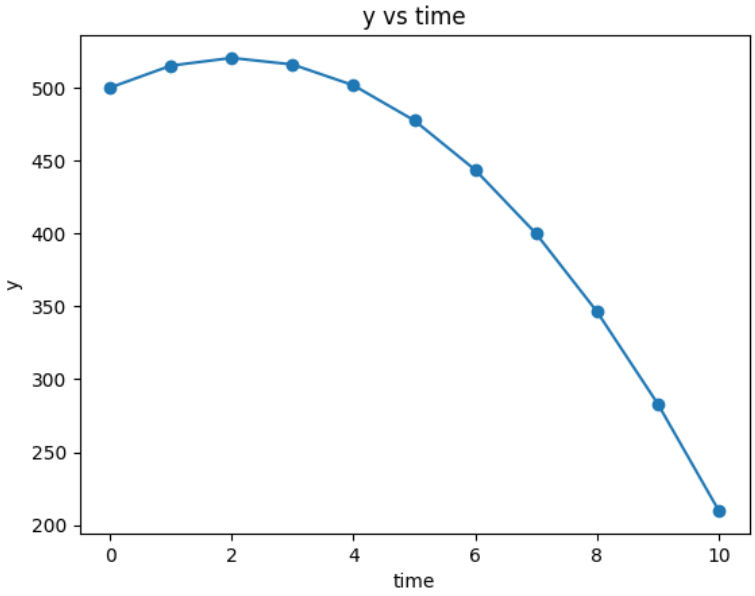

figure 8

From Figure 7, we observe that increasing the number of iterations leads to a convergence of trajectories, suggesting improved accuracy with higher iteration counts. This analysis further highlights the flexibility of our numerical framework and its ability to achieve varying levels of precision.

In this section, we have demonstrated the successful verification of the principle of least action through numerical simulations of free fall and projectile motion scenarios. The consistency between our simulation results and predictions from Newton's laws underscores the effectiveness of our approach. Additionally, the investigation into accuracy variations offers insights into the robustness of our method across different levels of precision.

V.CONCLUSION

In this study, we embarked on a numerical exploration of the principle of least action by implementing a Python code that simulates the paths of objects under free fall and in projectile motion. Our objective was to validate the principle of least action and compare its outcomes with predictions derived from Newton's laws. The simulations were conducted utilizing a numerical approach that allowed us to calculate trajectories while accommodating the principle of least action and Newtonian mechanics. We also examined the effect of altering the accuracy by adjusting the number of iterations in the Monte Carlo method.

Through our simulations and analyses, we arrived at several key conclusions:

1. Verification of Least Action Principle: Our simulations demonstrated that the principle of least action can be effectively employed to predict the trajectories of objects in free fall and projectile motion. The excellent agreement between the trajectories computed using the least action principle and those predicted by Newton's laws showcases an alternate verification of the principle.

2. Possible adaptability to complex cases: Our study extended the verification of the principle of least action beyond simple free fall scenarios to the more complex projectile motion. The successful representation of projectile trajectories using the least action principle highlights the versatility and broad applicability of our numerical framework. Presumably, the approach can be further developed to study more complicated examples.

3. Impact of Accuracy: We investigated the impact of accuracy on our simulations by varying the number of iterations in the Monte Carlo method. The observed convergence of trajectories of the real trajectory and the one our code of least action calculates with increasing iterations emphasizes the adaptability of our approach in achieving desired levels of precision, albeit at the expense of computing power.

Our research provides compelling evidence for the effectiveness of the numerical approach in verifying the principle of least action. The congruence between simulation results and Newtonian predictions reinforces the principles of classical mechanics while showcasing the capabilities of computational techniques in tackling complex physical phenomena. Our study not only reaffirms the foundational principles of physics but also highlights the potential of numerical simulations in enhancing our understanding of the physical world.

Future research could explore more intricate scenarios and potentially integrate additional factors such as friction, air resistance, and other real-world effects.

ACKNOWLEDGEMENT

I extend my sincere gratitude to the individuals who have contributed their guidance, support, and encouragement throughout the course of this research endeavor. My journey in exploring the principle of least action through numerical simulations has been enriched by the wisdom, assistance, and motivation I received from various sources.

First and foremost, we express our heartfelt appreciation to Suvam Maharana, my mentor, for his unwavering guidance, insightful discussions, and invaluable feedback that have played an instrumental role in shaping the direction of this research. His expertise, dedication, and commitment to fostering a deeper understanding have been truly inspiring.

I am also indebted to my friends, whose constant encouragement, thought-provoking discussions, and camaraderie have been a source of motivation throughout this project. My parents deserve my deepest gratitude for their unflagging support, understanding, and encouragement throughout my academic pursuits.

Furthermore, I would like to extend our appreciation to Hema Jain, my computer science teacher, for her guidance in programming concepts, which proved invaluable in the development and refinement of our numerical simulations. Her dedication to teaching and her willingness to provide assistance has significantly contributed to the success of my project.

Finally, I acknowledge the broader academic community and the researchers whose work laid the foundation for my study. This research would not have been possible without the collective support of these individuals. Each has played a crucial role in my growth, learning, and accomplishments. Their contributions serve as a testament to the power of collaboration and mentorship in the pursuit of knowledge.

REFERENCES:

1. Coopersmith, J. (2017). The Lazy Universe: An Introduction to the Principle of Least Action. Oxford University Press.
2. The Feynman Lectures on Physics. (n.d.). The Feynman Lectures on Physics. Retrieved August 27, 2023, from https://www.feynmanlectures.caltech.edu
3. French, A. P. (1971). Newtonian Mechanics. W.W. Norton.
4. Hamill, P. (2014). A Student's Guide to Lagrangians and Hamiltonians. Cambridge University Press.
5. Hamiltonian & Lagrangian Mechanics. (2010, October 8). the University of Warwick. Retrieved August 27, 2023, from https://warwick.ac.uk/fac/sci/physics/intranet/pendulum/hamiltonian/
6. Manton, N. S. (2013, April 25). The Principle of Least Action in Dynamics. DAMTP. Retrieved August 27, 2023, from https://www.damtp.cam.ac.uk/user/nsm10/PrincLeaAc.pdf
7. Turner, P. R. (1989). Guide to Numerical Analysis (P. R. Turner, Ed.). Macmillan Education.