Time Optimal Control using Interval Analysis

Submitted By

Vangapally Santhoshi
EEA232655

Under the guidance of
Prof.Shaunak Sen

Indian Institute of Technology, Delhi
Master of Technology
Control and Automation

Department Of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI
May, 2025

Thesis Certificate

This is to certify that the thesis entitled **Time Optimal Control using Interval Analysis** is a bonafide record of work done by **Vangapally Santhoshi(2023EEA2655)** at Indian Institute of Technology Delhi, as partial fulfilment of requirements for the degree of Master of Technology in Control and Automation. She has fulfilled the requirements for the submission of this thesis, which to the best of my knowledge has reached the required standard. This thesis is carried out under my supervision and guidance and has not been submitted elsewhere for the award of any other degree.

Prof. Shaunak Sen
Professor
Department of Electrical Engineering
Indian Institute of Technology Delhi

Date: June 2025

Acknowledgements

I would like to express my deepest gratitude to my supervisor, **Prof. Shaunak Sen**, for his invaluable guidance and consistent encouragement throughout the course of my M.Tech project. I am especially thankful for his patience, thoughtful feedback, and continued mentorship, which have greatly contributed to my academic and personal growth. It has truly been a privilege to work under his supervision.

I am also sincerely thankful to all faculty members of control and automation for their constant support and guidance.I am thankful to my colleagues for their constant support and collaborative spirit.

Finally, I extend my heartfelt thanks to my family for their unconditional love, encouragement, and support throughout this academic journey.

Vangapally Santhoshi

Contents

List of Figures

Abstract

The brachistochrone problem, a classical example in calculus of variations, serves as a benchmark in the study of time-optimal control, serves as the foundation for this thesis. This thesis rigorously investigates the classical brachistochrone problem through interval analysis with the goal of finding the minimum time trajectories. Standard numerical methods fails to guarantee the accuracy of the solution to the discretization. This can lead to inaccurate solutions due to discretization errors and fail to capture the exact time-optimal path, significant errors or missed solutions. To address this limitation, interval methods are employed to account for computational uncertainties and ensure that all possible solutions are enclosed within verified bounds. In this work, the problem of computing the time-optimal trajectory between two points under gravity the brachistochrone problem is revisited using verified numerical techniques that provide guaranteed bounds on the solution.

This study investigates three distinct approaches: a dynamic programming algorithm formulated with interval arithmetic, an application of the Interval Newton method to a segmented version of the curve, and a branch and bound algorithm integrated with interval analysis. Each method is evaluated and compared the computed time-optimal trajectories against the analytical cycloid solution , and the absolute errors relative to the analytical solution are explicitly quantified.

Chapter 1

Introduction

1.1 Background

1.1.1 Optimal control Theory

Optimal control theory is a significant component in engineering and mathematics. Optimal control is employed to determine the most effective control policy for a dynamic system. It involves determining how to influence the behavior of the system over time by adjusting control inputs in such a way that a desired performance criterion is optimized, such as minimizing fuel use, minimizing cost, reducing time, or maximizing performance, while ensuring that the system follows certain rules or dynamics. The optimal policy is determined through the application of a variety of methods, including dynamic programming, calculus of variations, the Hamilton-Jacobi-Bellman equation, and Pontryagin's minimum or maximum principle. In practice, solving optimal control problems often requires the use of numerical methods, particularly when exact analytical solutions are not feasible. However, conventional numerical methods often suffer from issues such as discretization errors, numerical round-off, and dependency on precise input values can affect the accuracy of the results. To address such challenges, interval analysis provides a robust alternative[1][2].

Interval analysis is a numerical method designed to handle uncertainty and imprecision in calculations. Rather than relying on exact numerical values, this approach uses *intervals* ranges of values that are guaranteed to include the true quantity. This is especially useful when dealing with rounding errors, incomplete data, or approximate models, as it ensures the results remain reliable.

The basic idea of interval analysis is built on *interval numbers*, typically written as $[a, b]$, where a and b are real numbers and $a \leq b$. This form represents every real number lying between the endpoints a and b, inclusive. Standard arithmetic operations—such as addition, subtraction, multiplication, and division can be extended to intervals so that the resulting interval contains all possible outcomes from applying the operation to any pair of values in the respective intervals[3].

A major advantage of this method is that it provides guaranteed bounds for the solution. That means the output is not merely a numerical approximation, but a confirmed range that the actual value must lie within. Due to this property, interval analysis finds applications in global optimization, control theory, validated numerics, and other areas where rigorously bounded results are essential.

Additionally, interval arithmetic enables the construction of robust algorithms capa-

ble of handling uncertainty in data and computation. For example, in the analysis of control systems, interval methods can reveal system behavior over a range of parameter values. They also offer solid error bounds in numerical integration and in the solution of differential equations.

1.1.2 Brachistochrone curve

The Brachistochrone problem is a classic problem in calculus of variations, first posed by Johann Bernoulli in 1696. The problem involves determining the shape of a curve along which a particle, moving solely under the influence of gravity and without any friction, travels between two specified points in the least amount of time [5]. Although a straight-line path offers the shortest distance, the particle's average speed remains relatively low due to gradual acceleration. In contrast, a curved trajectory allows the particle to accelerate more quickly, achieving higher speeds earlier, albeit covering a longer distance, as depicted in Figure 1.1. The right balance of these factors leads to the solution. The solution, a cycloid, surprised many at the time because it is not a straight line or a simple arc. This problem became a starting point for many later developments in optimization and control theory.

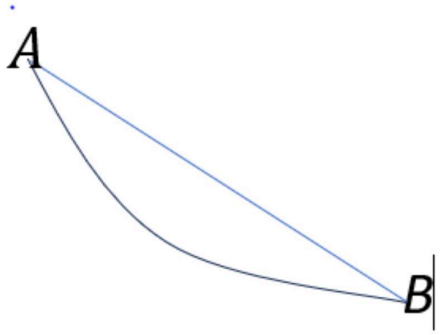

Figure 1.1

1.2 Motivation

The motivation behind using interval analysis approach to optimal control problems which helps in finding all possible solutions of the optimal control problem.

1.3 Problem Statement

The primary objective of this thesis is to apply interval analysis techniques to brachistochrone curve which is a benchmark problem in optimal control. In this the primary objective is to apply interval analysis to brachistochrone problem for bounding the minimum time trajectories.

1.4 Contribution to the thesis

The primary contributions of this thesis are:

- An interval analysis based dynamic programming algorithm is formulated and used to solve the brachistochrone problem, bounding the minimum time trajectories.
- An interval newton approach is applied to the discretized version of brachistochrone problem by bounding the minimum time trajectories. A branch and bound algorithm with interval analysis to the discretized version of brachistochrone problem by bounding the minimum time trajectories and by eliminating the trajectories which doesn't provide minimum time solution.
- Each method is evaluated and compared in terms of accuracy, and the error margins relative to the analytical cycloid solution are explicitly quantified.

1.5 Thesis organization

Chapter-1 provides a concise introduction to optimal control theory and brachistochrone curve along with its historical background. A brief overview of interval analysis and an introduction to the basic principles of interval analysis.

Chapter-2 covers the mathematical background required for the formulation of the brachistochrone problem and also the analytical solution of the brachistochrone problem.

Chapter-3 investigates the implementation of interval analysis based dynamic programming approach. The algorithm is illustrated in detail for implementing the dynamic programming method. It also includes the results obtained using dynamic programming method and interval dynamic programming method.

Chapter-4 investigates the implementation of interval newton method, branch and bound algorithm. These algorithms are illustrated in detail for implementing to the discretized version of the brachistochrone problem. It also includes the results obtained using interval newton method, branch and bound algorithm and compared with the cycloid curve.

Chapter-5 thesis concludes with the summary of the findings, contributions and future scope.

Chapter 2

Mathematical Formulation

Let us denote the initial point as $(0, h)$ and the final point as $(h, 0)$. The law of conservation of energy can be applied to analyze and calculate the time in problems where energy is conserved. When an object is in motion under the force of gravity ($g = 9.8\,\frac{\mathrm{m}}{\mathrm{s}^2}$), , its total mechanical energy is given by the combination of its kinetic energy and potential energy (PE) remains conserved throughout the motion.. The velocity at any point can be obtained using energy conservation:

$$mgh = \frac{1}{2}mv_i^2 + mgy_i$$
$$v_i = \sqrt{2g(h - y_i)}$$

2.1 Analytical solution of the cycloid curve

The time to traverse a segment of the curve is given by:

$$t = \int \frac{\sqrt{dx^2 + dy^2}}{\sqrt{2gy}}$$

Rewriting in terms of x as the independent variable, we have:

$$t = \int_0^{x_1} \frac{\sqrt{1 + \left(\frac{dy}{dx}\right)^2}}{\sqrt{2gy}}\, dx$$

This is a functional of the form:

$$t = \int_{x_0}^{x_1} L(y, y')\, dx, \quad \text{where} \quad L(y, y') = \frac{\sqrt{1 + (y')^2}}{\sqrt{2gy}}$$

To find the function $y(x)$ that minimizes this integral, we apply the Euler–Lagrange equation:

$$\frac{d}{dx}\left(\frac{\partial L}{\partial y'}\right) - \frac{\partial L}{\partial y} = 0$$

Substituting into the Euler–Lagrange equation gives:

$$\frac{d}{dx}\left(\frac{y'}{\sqrt{2gy}\sqrt{1 + (y')^2}}\right) - \frac{\sqrt{1 + (y')^2}}{2\sqrt{2g}\, y^{3/2}} = 0.$$

Solving this leads to the nonlinear differential equation:

$$y' + 2yy'y'' + (y')^3 = 0$$

$$\frac{d}{dx}\left(y + y(y')^2\right) = 0$$

$$y + y(y')^2 = c_1$$

$$1 + (y')^2 = \frac{c_1}{y}$$

$$(y')^2 = \frac{c_1 - y}{y}$$

$$\frac{dy}{dx} = \sqrt{\frac{c_1 - y}{y}}$$

$$dx = \sqrt{\frac{y}{c_1 - y}}\, dy$$

Assume:

$$y = c_1 \sin^2\theta$$

$$dy = 2C_1 \sin\theta \cos\theta\, d\theta$$

$$dx = \sqrt{\frac{c_1 \sin^2\theta}{c_1 - c_1 \sin^2\theta}} \cdot 2c_1 \sin\theta \cos\theta\, d\theta$$

$$dx = 2C_1 \sin\theta\, d\theta$$

$$x(\theta) = c_1\left(\theta - \frac{1}{2}\sin 2\theta\right) + c_2$$

$$y(\theta) = \frac{c_1}{2}(1 - \cos 2\theta)$$

The analytical solution which is known to be a cycloid curve. The cycloid represents the path that minimizes the time of descent under gravity, a result originally solved by Johann Bernoulli in 1696, which gave rise to the field of the calculus of variations. The classical derivation of the brachistochrone curve assumes that the particle starts at the origin $(0, 0)$ and to a point (x, y)

The objective is to compute the optimal time trajectory of a particle moving under the influence of gravity from an initial point (0,1) to a terminal point (1,0) as discretized in the problem domain. To correctly model this scenario, invert the classical cycloid vertically.This inversion reflects the curve about the horizontal axis. The cycloid path can be modified as:

$$x(\theta) = c_1\,(2\theta + \sin(2\theta)) + c_2,$$
$$y(\theta) = 1 - c_1\,(1 + \cos(2\theta))$$

Here, θ is a parameter varying from θ_0 to θ_f, and c_1, c_2 are constants chosen such that the curve passes through the given boundary points.

The boundary conditions:
The particle starts at $(0, 1)$, which gives the constraint:

$$x(\theta_0) = 0,$$
$$y(\theta_0) = 1$$

$$\theta_0 = \frac{\pi}{2}$$
$$c_2 = -c_1\pi$$

It ends at $(1, 0)$, which gives the constraint:

$$x(\theta_f) = 1,$$
$$y(\theta_f) = 0,$$
$$c_1 = \frac{1}{1 + \cos(2\theta_f)}$$

Substituting these into the cycloid equations and simplifying yields the nonlinear equation:

$$f(\theta_f) = 1 + \cos(2\theta_f) - (2\theta_f - \pi + \sin(2\theta_f)) = 0$$

This equation cannot be solved analytically and hence it is solved using the interval Newton method, which ensures rigorously validated bounds on θ_f with initial guess as $[\theta_{f0}] = \left[\frac{\pi}{2},\ 2\pi\right]$

$$N([\theta_f]) = m - \frac{f(m)}{f'([\theta_f])}$$
$$\theta_{\mathrm{f}} = N \cap \theta_f$$

Here, m denotes the midpoint of the interval $[\theta_f]$, and $f'([\theta_f])$ represents the extended interval of the derivative over $[\theta_f]$. If the computed interval $N([\theta_f])$ is a subset of $[\theta_f]$, it can be concluded that a unique root exists within $[\theta_f]$. The interval θ_f is refined during each iteration by replacing it with $\theta_f \cap N(\theta_f)$. This process narrows the interval's range step by step. The refinement continues until the condition $\|f(\theta_f)\| < 0.0001$ is satisfied.

The Interval Newton Method has been effectively applied in various contexts, particularly in solving nonlinear systems and globally optimal solutions across continuous domains. Through this method,

$$\theta_f = [2.7768,\ 2.77681]$$
$$\mathrm{mid}(\theta_f) = 2.77680189$$
$$c_1 = \frac{1}{1 + \cos(2 \cdot 2.77680189)} = 0.5729170373653$$

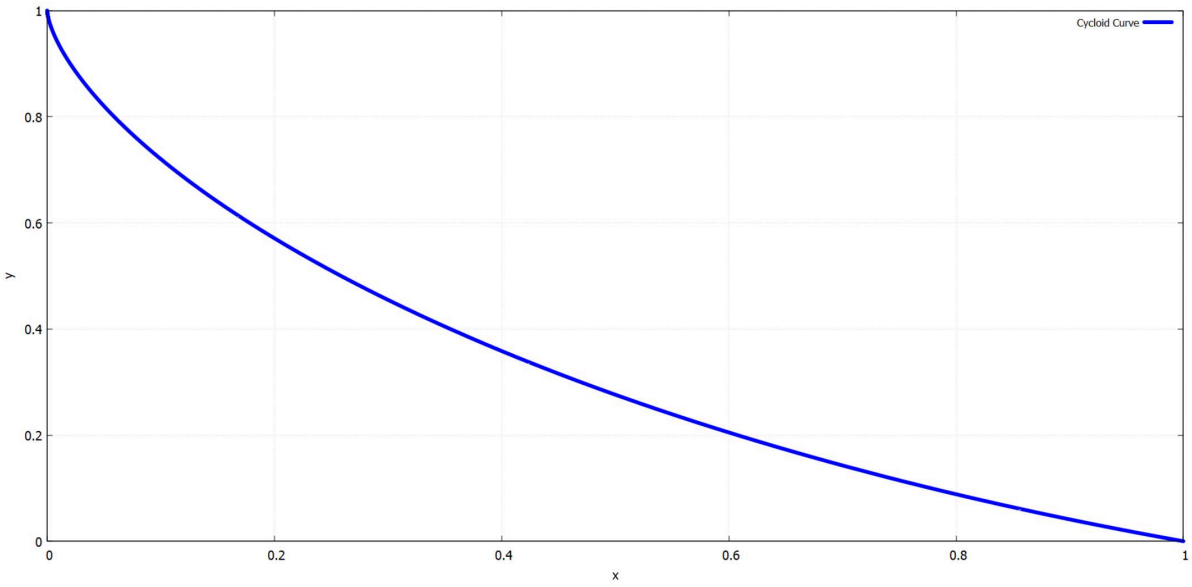

Figure 2.1: Cycloid curve

2.2 Minimum time calculation of cycloid curve

The minimum time taken is given by the integral:

$$\text{Minimum time} = \int_{\theta_f}^{\theta_0} \frac{\sqrt{dx^2 + dy^2}}{\sqrt{2g(h-y)}}$$

where

$$dx = c_1 \cdot 2(1 + \cos(2\theta))d\theta, \quad dy = c_1 \cdot 2\sin(2\theta)d\theta$$

$$\text{The minimum time taken is given by} = \int_{\theta_f}^{\theta_0} \frac{\sqrt{4c_1^2(1+\cos(2\theta))^2 + 4c_1^2\sin^2(2\theta)}}{\sqrt{2gc_1(1+\cos(2\theta))}}\, d\theta$$

$$= \frac{\sqrt{8c_1}\,(\theta_0 - \theta_f)}{\sqrt{2g}}$$

$$\text{Minimum time taken} = 0.582334 seconds$$

The analytical solution to the brachistochrone problem was derived using the Euler–Lagrange equation. The resulting path is cycloid, parameterized to connect the points (0,1) and (1,0).

Although the analytical approach provides a solution, it may be impractical. This motivates the use of numerical methods. The following chapter discusses interval-based and dynamic programming approaches that provide verified solutions.

Chapter 3

Space Discretization

3.1 Dynamic programming approach

Dynamic programming is a recursive method used for solving complex optimization problems by breaking them down into simpler subproblems. A key foundational concept of dynamic programming is the principle of optimality, first articulated by Bellman (1957), which states: "An optimal policy is one in which, no matter what the first state and decision are, the rest of the decisions must be optimal for the state that results from the first decision." This recursive nature allows the algorithm to systematically explore all possible solutions and build up an optimal strategy through backward or forward induction. This characteristic allows the original problem to be systematically divided into smaller, manageable parts, enabling it to be solved efficiently through either a top-down recursive process or a bottom-up iterative strategy[5].

In case of brachistochrone problem, the aim is to determine the trajectory of a particle moving under the influence of gravity in the shortest possible time, starting from an initial point and ending at a specified target. To make the problem tractable using dynamic programming, the continuous spatial domain is discretized into a grid of discrete states. Each state represents a potential position of the particle at a certain horizontal coordinate, and the time to traverse from one state to another. The algorithm initiates the computation from the final state typically the target position and then works backwards through the discretized grid, evaluating the minimum time required to reach each preceding state. For each transition, the algorithm calculates a cost (in this case, time) and retains the minimum among all possible transitions. Transitions between states are evaluated based on the time taken by a particle to traverse between them. This backward induction continues until the initial state is reached, and the path with the minimum accumulated cost represents the optimal solution.

3.2 Routing

To apply the dynamic programming to the brachistochrone problem, it requires transforming continuous domain into a discreted form. This discretization involves dividing the feasible region of motion into a structured grid, thereby converting a continuous trajectory planning problem into a combinatorial routing problem. The resulting framework facilitates recursive cost computation across discrete states.

A typical setup involves partitioning the two-dimensional space into an n×n grid, where each cell represents a potential state of the particle. The horizontal and vertical distances between adjacent grid points are each set to h/n, where h denotes the total length along the respective axis. This grid-based representation simplifies the evaluation of transitions and associated travel times across discrete stages.

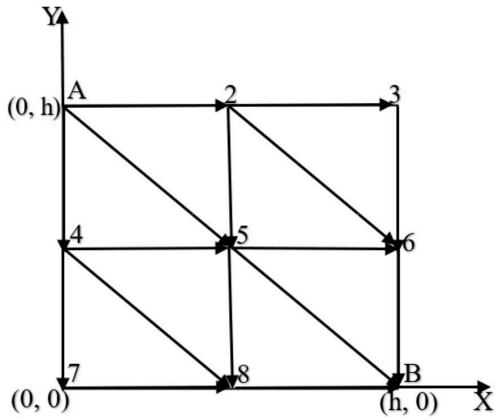

Figure 3.1: Dynamic Programming-Routing

To illustrate this discretization, consider a simplified 2×2 grid as shown in Figure-3.1. Here, the domain is divided into 4 square cells, with 9 discrete nodes labeled for convenience. The particle's movement is permitted only along predefined directions—rightward, downward, or diagonally downward.

The node labeled A represents the starting point, located at (0,h), while the node B denotes the final position at (h,0). Transitions between adjacent nodes correspond to feasible particle paths, and each such path is associated with a travel time computed using classical mechanics. These transition times serve as the stage-wise costs in the dynamic programming formulation.

Decision-making in this framework proceeds in a backward fashion. The computation begins from the final row of nodes (in this case, nodes 7, 8, and B) and evaluates the minimum cost-to-go for each preceding node by recursively selecting the transition yielding the least time. For instance, once the minimum cost from nodes 4, 5, and 6 to B is determined, the method proceeds to evaluate optimal transitions from nodes 1, 2, and 3. This backward induction is not only computationally efficient but also aligns with the structure of Bellman's principle of optimality. The grid-based discretization further allows for systematic storage of decisions and costs, which are essential for reconstructing the optimal trajectory.

$$t^*_{AB} = \min\left(t_{A5} + t^*_{5B},\ t_{A4} + t^*_{4B},\ t_{A2} + t^*_{2B}\right)$$
$$t^*_{4B} = \min\left(t_{47} + t_{78} + t_{8B},\ t_{45} + t^*_{5B},\ t_{48} + t_{8B}\right)$$
$$t^*_{5B} = \min\left(t_{5B},\ t_{36} + t_{58} + t_{8B},\ t_{56} + t_{6B}\right)$$

A key advantage of this discretized formulation lies in its extensibility to finer grids. By increasing the number of discretization points, one can approximate the continuous trajectory with greater precision, albeit at the expense of computational complexity. However, this trade-off is manageable due to the inherently recursive and parallelizable structure of dynamic programming.

3.3 Interval variant

To enhance the reliability of dynamic programming method an interval analysis based variant is introduced. Unlike conventional dynamic programming where each state is represented as a single point, this approach models each state as an interval, capturing a range of possible values rather than a fixed quantity. Using interval arithmetic, the cost-to-go function is evaluated over intervals rather than scalar values. By treating state variables as intervals, the algorithm ensures that all feasible trajectories within the defined bounds are considered. This provides a more rigorous framework for solving the problem, particularly in systems governed by nonlinear dynamics such as the brachistochrone.

The interval dynamic programming method thus combines the structural clarity of Bellman's recursive formulation with the rigor of validated numerics. It finds applications in engineering disciplines where uncertainty quantification and solution guarantees are essential, such as in control systems, robotics path planning.

The corresponding interval for travel time across the segment is: Each state is interval, and

$$t = \left[\frac{d}{v_h}, \frac{d}{v_\ell}\right]$$

$$t^*_{AB} = \min\left(t_{A5} + t^*_{5B},\ t_{A4} + t^*_{4B},\ t_{A2} + t^*_{2B}\right)$$

$$t^*_{4B} = \min\left(t_{47} + t_{78} + t_{8B},\ t_{45} + t^*_{5B},\ t_{48} + t_{8B}\right)$$

$$t^*_{5B} = \min\left(t_{5B},\ t_{58} + t_{8B},\ t_{56} + t_{6B}\right)$$

$$t^*_{5B} = \min\left(\left[\frac{d_{5B}}{v_{5Bh}}, \frac{d_{5B}}{v_{5B\ell}}\right], \left[\frac{d_{58}}{v_{58h}} + \frac{d_{8B}}{v_{8Bh}}, \frac{d_{58}}{v_{58\ell}} + \frac{d_{8B}}{v_{8B\ell}}\right], \left[\frac{d_{56}}{v_{56h}} + \frac{d_{6B}}{v_{6Bh}}, \frac{d_{56}}{v_{56\ell}} + \frac{d_{6B}}{v_{6B\ell}}\right]\right)$$

$$= \min\left(\left[\frac{d_{5B}}{v_{5B\ell}}, \frac{d_{5B}}{v_{5Bh}}\right], \left[\frac{d_{58}}{v_{58\ell}} + \frac{d_{8B}}{v_{8B\ell}}, \frac{d_{58}}{v_{58h}} + \frac{d_{8B}}{v_{8Bh}}\right], \left[\frac{d_{56}}{v_{56\ell}} + \frac{d_{6B}}{v_{6B\ell}}, \frac{d_{56}}{v_{56h}} + \frac{d_{6B}}{v_{6Bh}}\right]\right)$$

$$= \left[\min\left(\frac{d_{5B}}{v_{5Bh}}, \frac{d_{58}}{v_{58h}} + \frac{d_{8B}}{v_{8Bh}}, \frac{d_{56}}{v_{56h}} + \frac{d_{6B}}{v_{6Bh}}\right), \min\left(\frac{d_{5B}}{v_{5B\ell}}, \frac{d_{58}}{v_{58\ell}} + \frac{d_{8B}}{v_{8B\ell}}, \frac{d_{56}}{v_{56\ell}} + \frac{d_{6B}}{v_{6B\ell}}\right)\right]$$

3.3.1 Extended Routing with Five Directions

To improve the trajectory accuracy, the particle's movement is permitted with 5 predefined directions—rightward, downward,diagonal, diagonally downward and diagonally rightward.

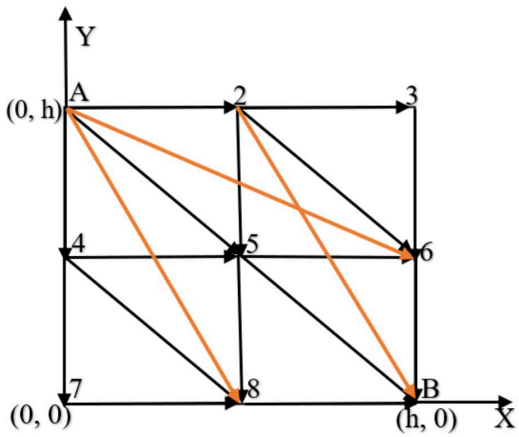

Figure 3.2: Dynamic Programming-Extended Routing

$$t^*_{AB} = \min\left(t_{A5} + t^*_{5B},\ t_{A4} + t^*_{4B},\ t_{A2} + t^*_{2B},\ t_{A8} + t_{8B},\ t_{A6} + t_{6B}\right)$$
$$t^*_{4B} = \min\left(t_{47} + t_{78} + t_{8B},\ t_{45} + t^*_{5B},\ t_{48} + t_{8B}\right)$$
$$t^*_{2B} = \min\left(t_{25} + t^*_{5B},\ t_{23} + t_{36} + t_{3B},\ t_{26} + t_{6B},\ t_{2B}\right)$$
$$t^*_{5B} = \min\left(t_{5B},\ t_{36} + t_{58} + t_{8B},\ t_{56} + t_{6B}\right)$$

Algorithm for Finding the Minimum Time Path (Brachistochrone Problem Using Dynamic Programming):

1. **Initialize:** Set the size of the grid and the height of the grid.
2. **Define a 2D array** `dp` of size `rows × cols`, where `dp[i][j]` stores the minimum time to reach cell (i, j). The starting point of the grid $(0, 0)$ corresponds to $(0, h)$ and the ending point (n, n) corresponds to $(h, 0)$ and also direction[i][j] to update the path.
3. **Loop through grid in reverse:**
   - **Outer loop:** For $i =$ `rows` down to 0
   - **Inner loop:** For $j =$ `cols` down to 0
4. **At each grid point** (i, j)**, compute time from neighboring cells:**
   - **Moving down** (from $(i+1, j)$ to (i, j)): Calculate time using `calculateTime()` and update `dp[i][j]` , `direction[i][j] = 0`.
   - **Moving right** (from $(i, j+1)$ to (i, j)): Calculate time. If it is less than the current `dp[i][j]`, update `dp[i][j]`, `direction[i][j] = 1`.
   - **Diagonal movement** (from $(i+1, j+1)$ to (i, j)): Calculate time. If it is less than the current `dp[i][j]`, update `dp[i][j]`, `direction[i][j] = -1`.
5. **calculateTime**(h_1, h_2, d) **Function**
   - v_1, v_2 are the velocities at heights h_1 and h_2
   - Approximate the average velocity:
   $$v_{\text{avg}} = \frac{v_1 + v_2}{2}$$
   - Return :
   $$t = \frac{d}{v_{\text{avg}}}$$
6. **Result**: The value dp[0][0] denotes the least amount of time necessary for the particle to travel from the initial position to the target, following the optimal path.

3.4 Numerical comparison with analytical solution

Results for dynamic programming and interval dynamic programming method

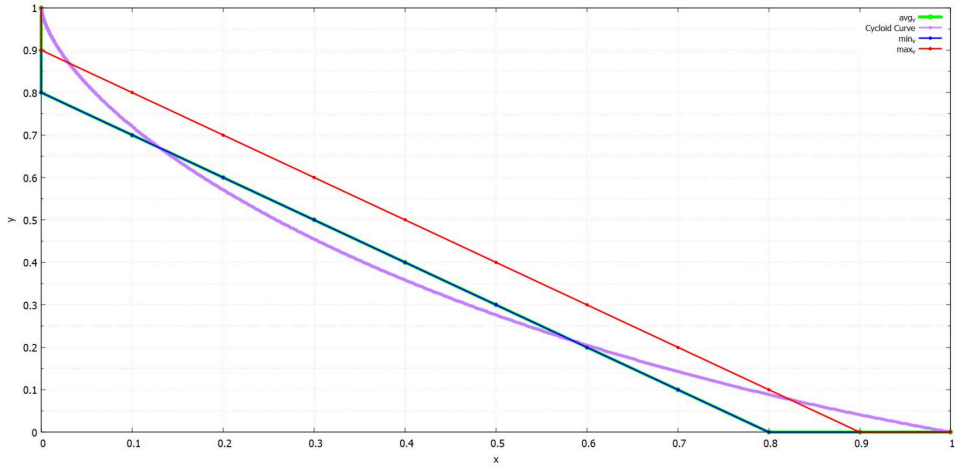

Figure 3.3: Comparison between the analytical cycloid curve and interval dynamic programming for grid size = 10. The corresponding computed times are [tmin, tavg, tmax] = [0.428551, 0.457278, 0.490592]

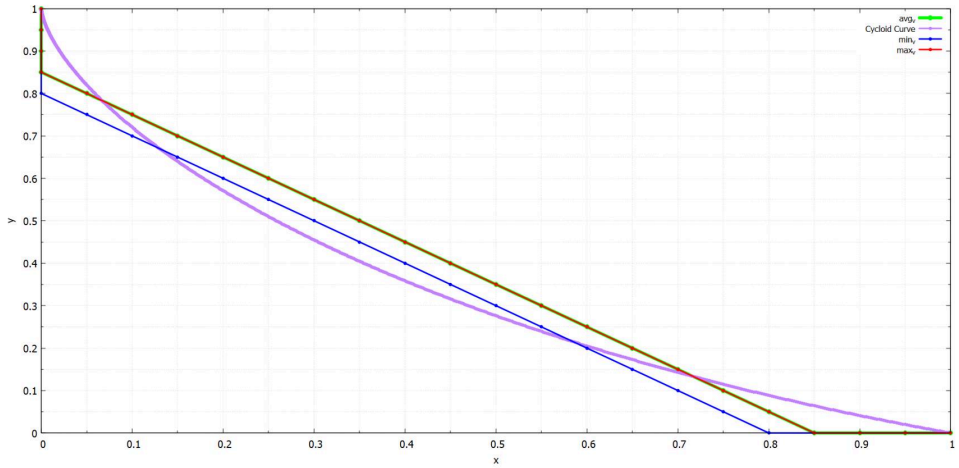

Figure 3.4: Comparison between the analytical cycloid curve and interval dynamic programming for grid size = 20. The corresponding computed times are $[t_{\min},\ t_{\text{avg}},\ t_{\max}] = [0.477855,\ 0.499016,\ 0.523662]$ s.

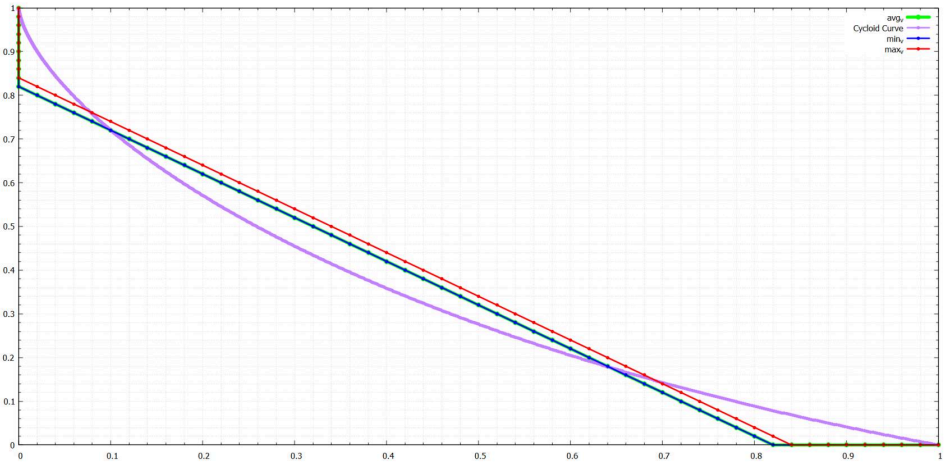

Figure 3.5: Comparison between the analytical cycloid curve and interval dynamic programming for grid size = 50. The corresponding computed times are $[t_{\min},\ t_{\text{avg}},\ t_{\max}] = [0.522183,\ 0.535984,\ 0.55224]$ s.

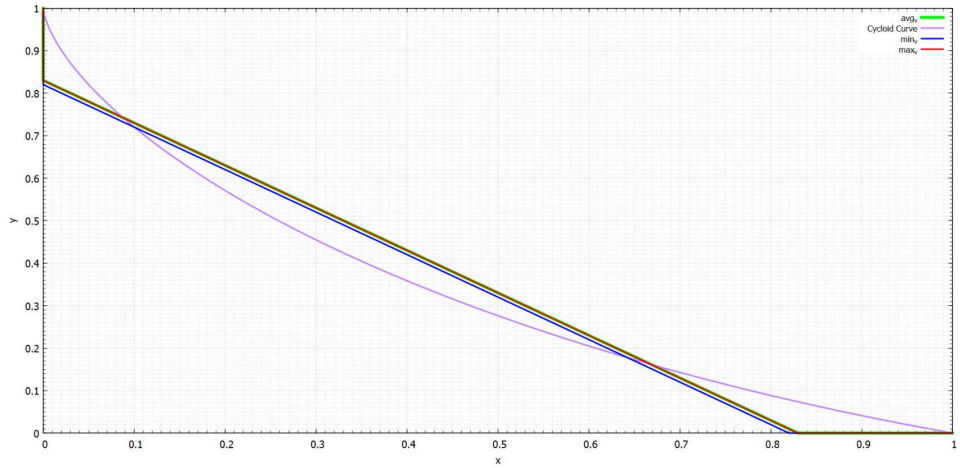

Figure 3.6: Comparison between the analytical cycloid curve and interval dynamic programming for grid size = 100. The corresponding computed times are $[t_{\min},\ t_{\text{avg}},\ t_{\max}] = [0.544738,\ 0.554665,\ 0.566379]$ s.

3.4.1 Results for Extended Routing

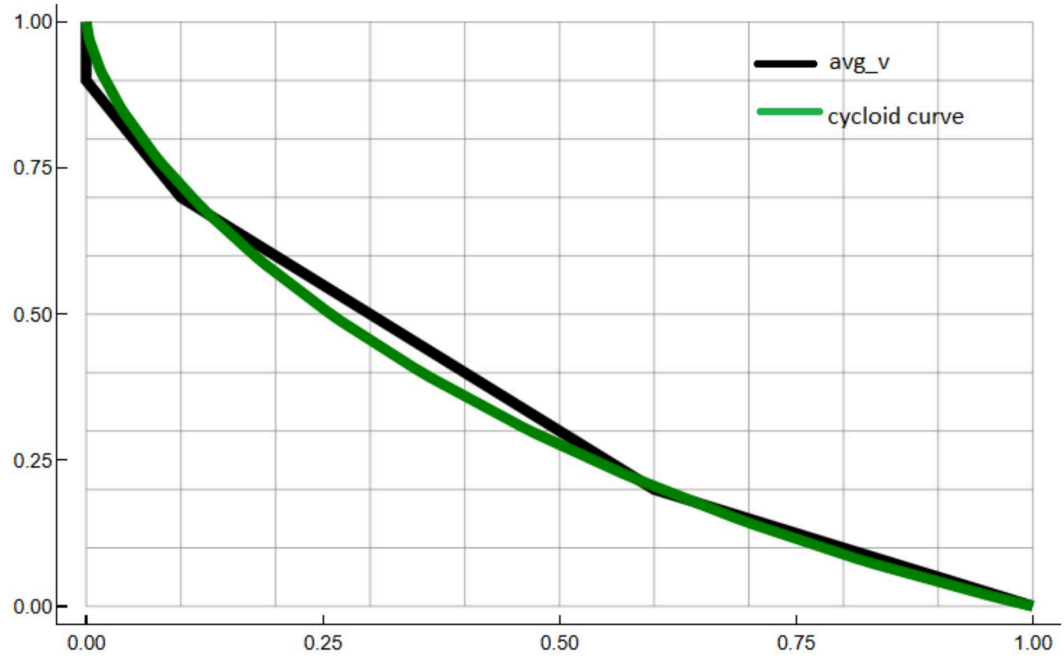

Figure 3.7: Comparison between the analytical cycloid curve and dynamic programming for grid size = 10. The time corresponding to avg_v = `[0.444841]`

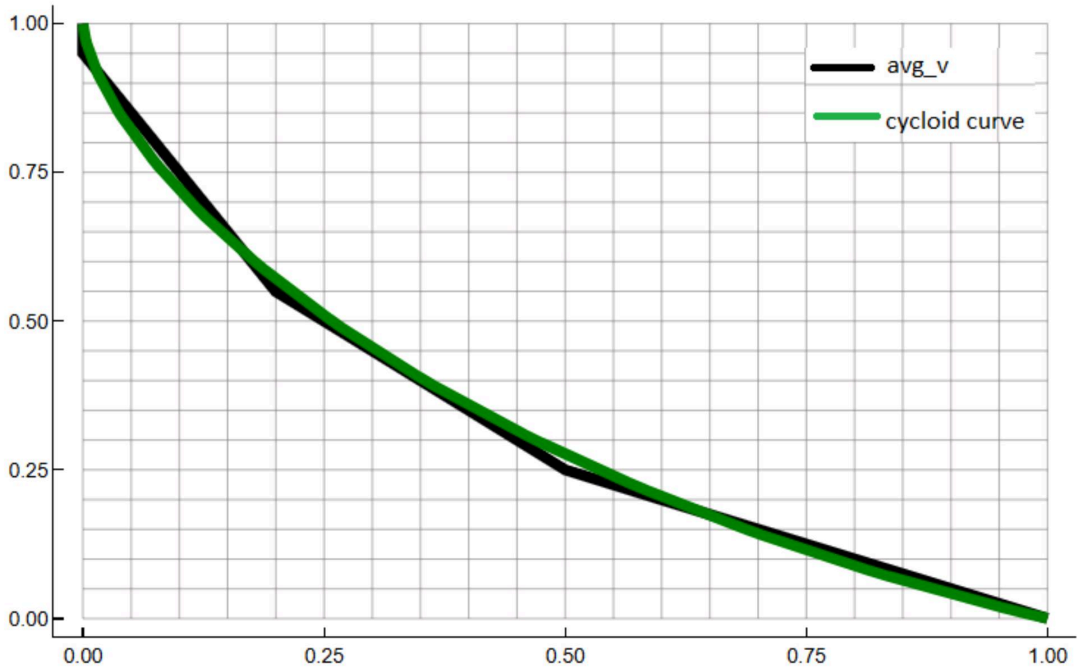

Figure 3.8: Comparison between the analytical cycloid curve and dynamic programming for grid size = 20. The time corresponding to avg_v = [0.485675]

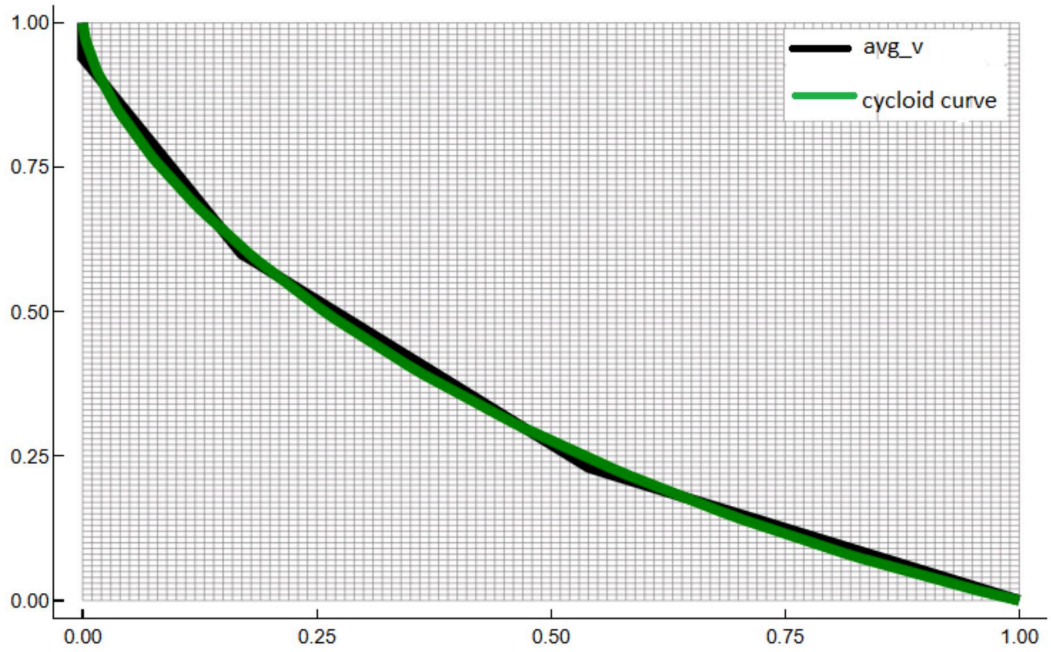

Figure 3.10: Comparison between the analytical cycloid curve and dynamic programming for grid size = 100. The time corresponding to avg_v = [0.541223]

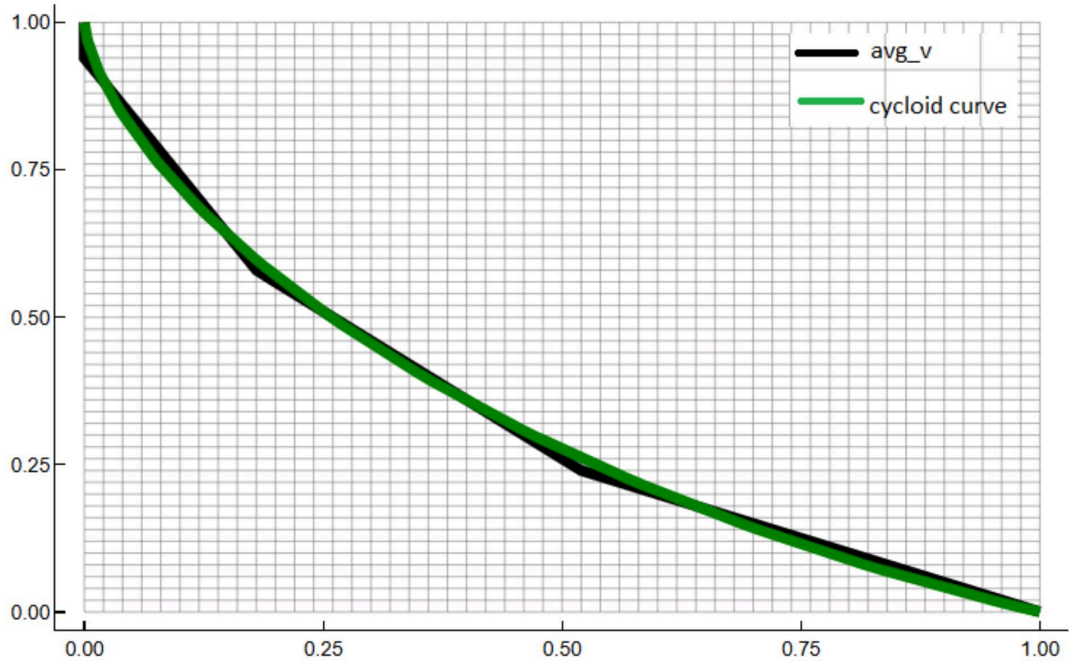

Figure 3.9: Comparison between the analytical cycloid curve and dynamic programming for grid size = 50. The time corresponding to avg_v = [0.522553]

Table 3.1: Comparison of the computed time standard DP and interval DP methods with particle moving in 3 and 5 directions for different grid sizes(N)

N	**3 Directions**				**5 Directions**	
	DP Time	Error	Interval DP Time	Error	DP Time	Error
10	0.457278	0.125056	[0.428551, 0.490592]	0.153783	0.444841	0.137492
20	0.499016	0.083318	[0.477855, 0.523662]	0.104479	0.485675	0.096658
50	0.535984	0.04635	[0.522183, 0.55224]	0.060151	0.522553	0.059781
100	0.554665	0.027669	[0.544738, 0.566379]	0.037596	0.541223	0.041111

The "DP Time" column shows results obtained through Dynamic Programming method. The "Interval DP Time" column shows results obtained through Interval Dynamic Programming method, while for the 5-direction case, only standard DP is used.
The error for the standard DP is calculated as the absolute difference between the computed time and the analytical minimum time (0.582334). For Interval DP, the error is taken as the maximum absolute difference between the endpoints of the interval and the analytical minimum, i.e., the worst-case deviation within the interval.
As N increases, both the DP and Interval DP methods show improved accuracy, with reduced errors. The results for 5 directions demonstrate that allowing more movement directions leads to more precise time estimates.

Chapter 4

Curve Discretisation

In the classic formulation of the brachistochrone problem, the objective is to determine the curve that minimizes the time under the influence of gravity. This problem is inherently continuous, but in order to solve it numerically, we must discretize the continuous space. Discretization involves breaking down the continuous trajectory into a finite number of points or segments shown in 4.1, the minimization problem is a problem in differential calculus, where the partial derivatives with respect to the x_i and y_i is equal to zero enabling the use of numerical methods for solving the problem.

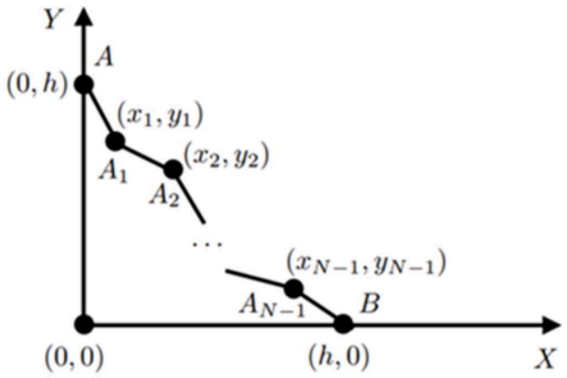

Figure 4.1: Differential calculus

$$t = \frac{\sqrt{x_1^2 + (1-y_1)^2}}{0.5\sqrt{2g(1-y_1)}} + \frac{\sqrt{(1-x_1)^2 + y_1^2}}{0.5\left(\sqrt{2g} + \sqrt{2g(1-y_1)}\right)} + \cdots$$

$$\frac{\partial t}{\partial x_1} = 0, \qquad \frac{\partial t}{\partial y_1} = 0, \qquad \cdots$$

This chapter primarily focuses on discretizing the desired curve only in the x-direction, i.e., the x-axis values on the curve are fixed as $x_i = \frac{ih}{n}$. Consequently, the minimization is performed with respect to the y_i values only, and the necessary condition for optimality reduces to setting the partial derivatives with respect to y_i to zero. This chapter focuses mainly on numerical techniques employed in solving the discretized brachistochrone problem. The discussion encompasses the classical Newton-Raphson method, its extension via

interval arithmetic known as the Interval Newton Method and the Branch-and-Bound algorithm. These methodologies offer rigorous frameworks for identifying and verifying solutions to nonlinear equations arising from the discretization of the brachistochrone problem.

4.1 Classical Newton-Raphson Method

The Newton-Raphson method is a widely utilized iterative technique for approximating the roots of real-valued functions. Given a differentiable function f(x) and an initial estimate x_0, the method improves this estimate step by step, using both the function and its derivative. Each new estimate is calculated using the following formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.1}$$

This approach assumes that the function f is sufficiently smooth and that its derivative f' does not vanish near the root. The method is renowned for its rapid convergence properties, particularly when the initial guess is close to the actual root. However, its performance is highly sensitive to the choice of the initial estimate and the nature of the function. In cases where the function exhibits multiple roots or complex behavior near the root, the method may diverge or converge to an unintended solution. Moreover, the requirement of computing the derivative f' can be computationally intensive or analytically intractable for certain functions.

Results obtained by Classical Newton-Raphson method

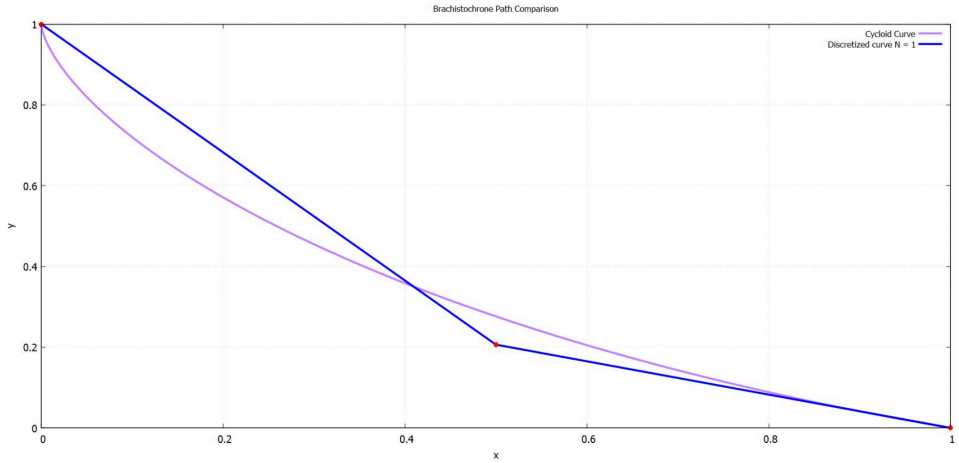

Figure 4.2: Newton Method for N = 1 with $x_1 = 0.5$, the corresponding value of $y_1 = 0.206164667$ and Min time = 0.6048926860909213sec

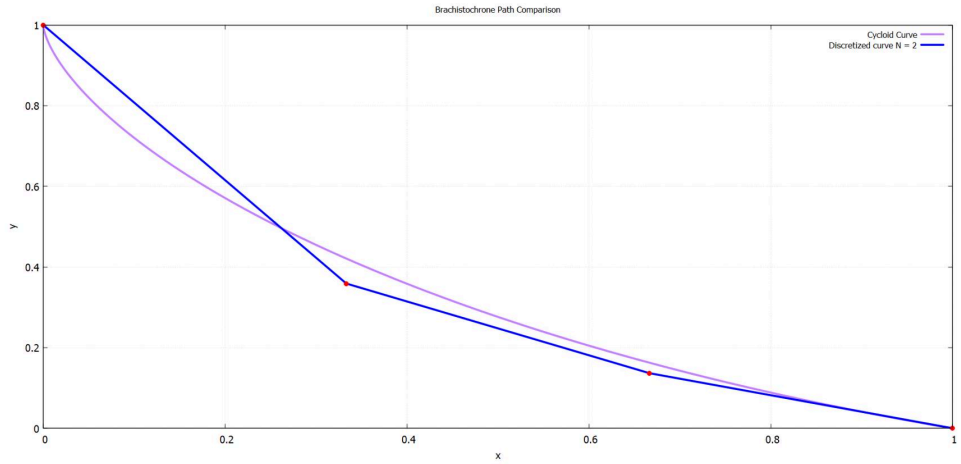

Figure 4.3: Newton Method for N = 2 with $x_1 = 0.333$, $x_2 = 0.667$ the corresponding value of $y_1 = 0.35886977818489324$, $y_2 = 0.13661175198231584$ and Min time = 0.5966717221533048sec

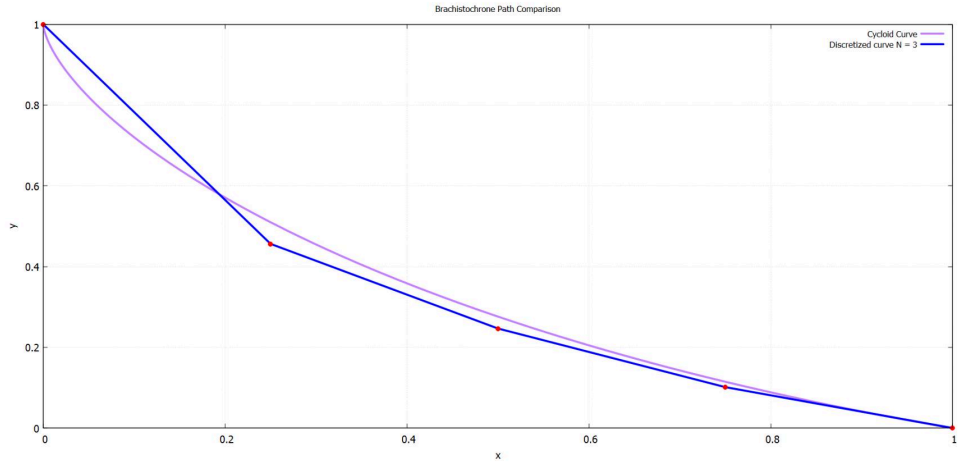

Figure 4.4: Newton Method for N = 3 with $x_1 = 0.25$, $x_2 = 0.5$, $x_3 = 0.75$ the corresponding value of $y_1 = 0.4561274654764423$, $y_2 = 0.24663357559130605$, $y_3 = 0.10137502319084135$ and Min time = 0.5929435291827193sec

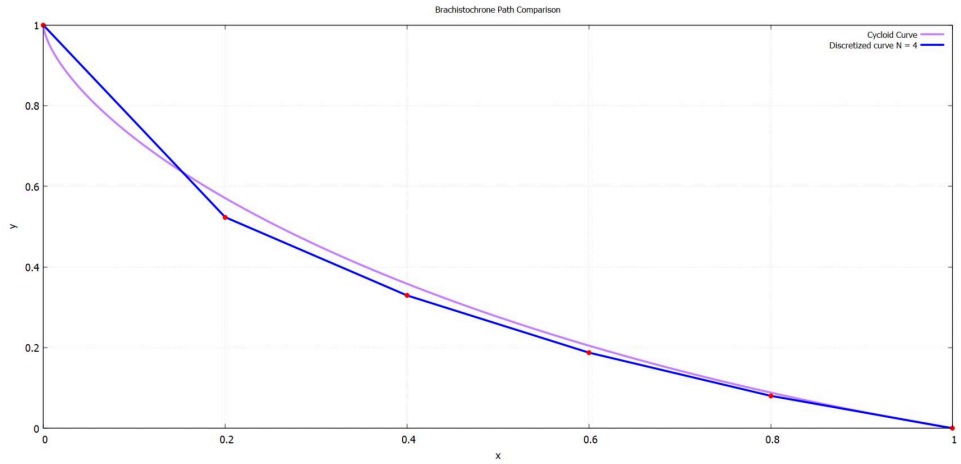

Figure 4.5: Newton Method for N = 4 with $x_1 = 0.2$, $x_2 = 0.4$, $x_3 = 0.6$, $x_4 = 0.8$ the corresponding value of $y_1 = 0.5235699510493174$, $y_2 = 0.32940796949173234$, $y_3 = 0.1875942490756577$, $y_4 = 0.080415281499471$ and Min time = 0.5908194273954204sec

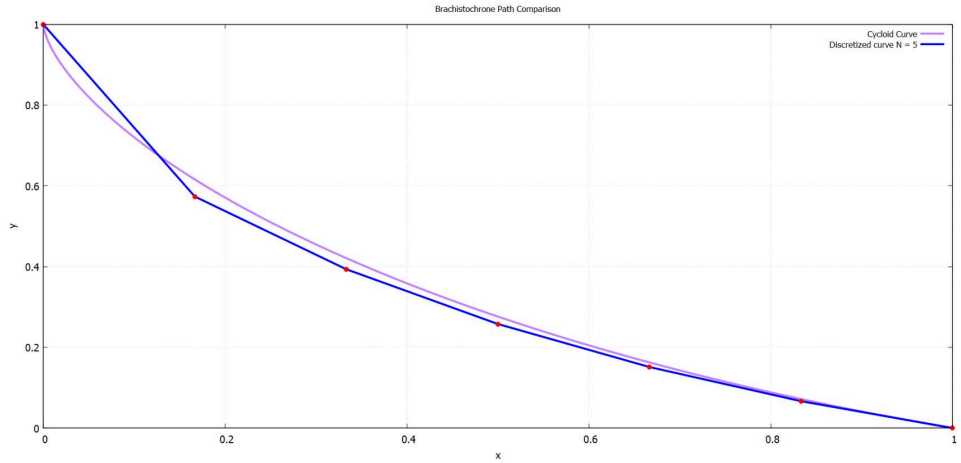

Figure 4.6: Newton Method for N = 5 with $x_1 = \frac{1}{6}$, $x_2 = \frac{2}{6}$, $x_3 = \frac{3}{6}$, $x_4 = \frac{4}{6}$, $x_5 = \frac{5}{6}$; the corresponding values of $y_1 = 0.573417359650963$, $y_2 = 0.393247828839364$, $y_3 = 0.25774210442218$, $y_4 = 0.15119880624595$, $y_5 = 0.066577707864137$ and Min time = 0.589449468181657sec

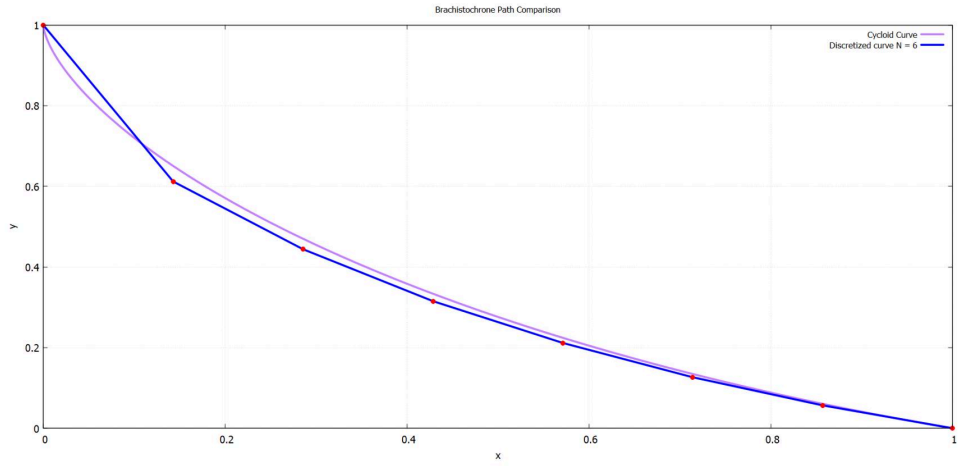

Figure 4.7: Newton Method for N = 6 with $x_1 = \frac{1}{7}$, $x_2 = \frac{2}{7}$, $x_3 = \frac{3}{7}$, $x_4 = \frac{4}{7}$, $x_5 = \frac{5}{7}$, $x_6 = \frac{6}{7}$; the corresponding values of $y_1 = 0.6119661574941$, $y_2 = 0.4439587107460$, $y_3 = 0.3151842286348$, $y_4 = 0.21149889117840$, $y_5 = 0.1265419390849$, $y_6 = 0.056774678985385$ and Min time = 0.5884935513673sec

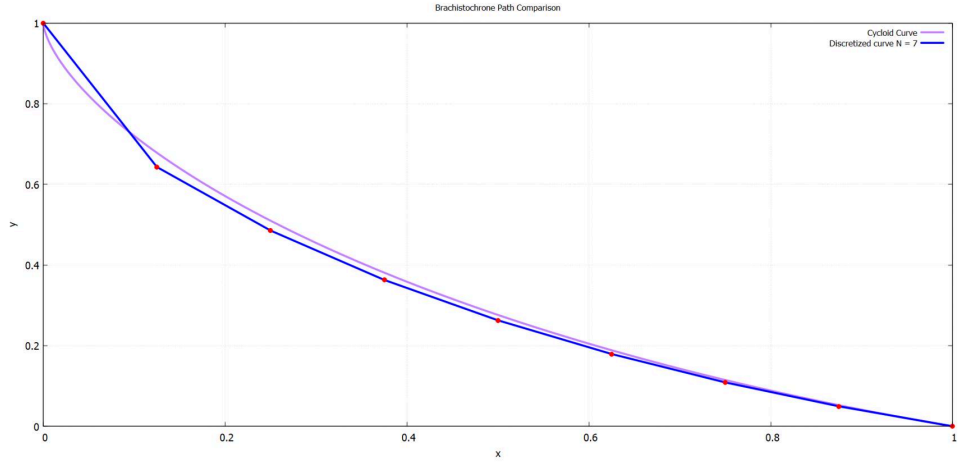

Figure 4.8: Newton Method for N = 7 with $x_1 = \frac{1}{8}$, $x_2 = \frac{2}{8}$, $x_3 = \frac{3}{8}$, $x_4 = \frac{4}{8}$, $x_5 = \frac{5}{8}$, $x_6 = \frac{6}{8}$, $x_7 = \frac{7}{8}$; the corresponding values of $y_1 = 0.6428303579372$, $y_2 = 0.4853090729346$, $y_3 = 0.36295250805483$, $y_4 = 0.2628534303635$, $y_5 = 0.17920285227424$, $y_6 = 0.10875479991531$, $y_7 = 0.04947544675184$ and Min time = 0.5877891193545sec

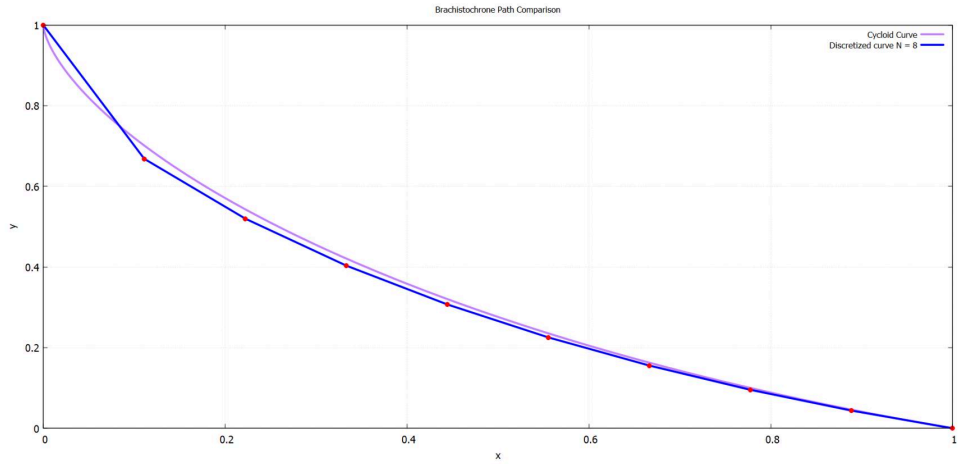

Figure 4.9: Newton Method for N = 8 with $x_1 = \frac{1}{9}$, $x_2 = \frac{2}{9}$, $x_3 = \frac{3}{9}$, $x_4 = \frac{4}{9}$, $x_5 = \frac{5}{9}$, $x_6 = \frac{6}{9}$, $x_7 = \frac{7}{9}$, $x_8 = \frac{8}{9}$; the corresponding values of $y_1 = 0.6681992205824$, $y_2 = 0.5197559828636$, $y_3 = 0.40330061537855$, $y_4 = 0.30693165386508$, $y_5 = 0.22529368913360$, $y_6 = 0.1553879559848$, $y_7 = 0.09532667726398$, $y_8 = 0.0438325712822$ and Min time = 0.5872487471717sec

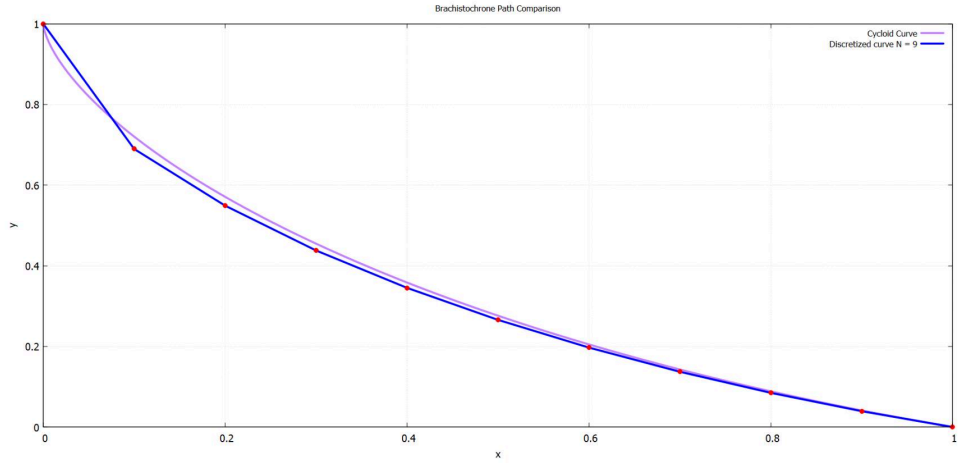

Figure 4.10: Newton Method for N = 9 with $x_1 = \frac{1}{10}$, $x_2 = \frac{2}{10}$, $x_3 = \frac{3}{10}$, $x_4 = \frac{4}{10}$, $x_5 = \frac{5}{10}$, $x_6 = \frac{6}{10}$, $x_7 = \frac{7}{10}$, $x_8 = \frac{8}{10}$, $x_9 = \frac{9}{10}$; the corresponding values of $y_1 = 0.6894726722628$, $y_2 = 0.5489506347155$, $y_3 = 0.4378562988395$, $y_4 = 0.3451238211051$, $y_5 = 0.26577663042926$, $y_6 = 0.19702541080306$, $y_7 = 0.13710970858111$, $y_8 = 0.08483287978730$, $y_9 = 0.039340171773004$ and Min time = 0.586821264323sec

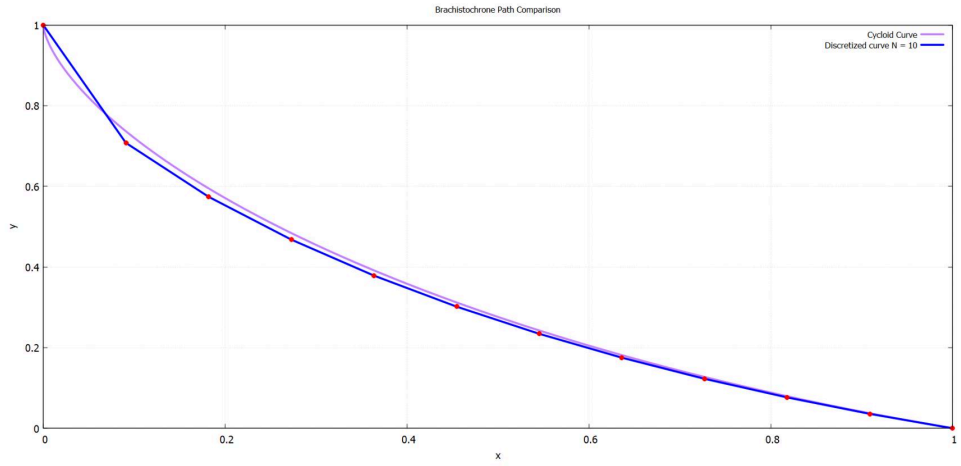

Figure 4.11: Newton Method for N = 10 with $x_1 = \frac{1}{11}$, $x_2 = \frac{2}{11}$, $x_3 = \frac{3}{11}$, $x_4 = \frac{4}{11}$, $x_5 = \frac{5}{11}$, $x_6 = \frac{6}{11}$, $x_7 = \frac{7}{11}$, $x_8 = \frac{8}{11}$, $x_9 = \frac{9}{11}$, $x_{10} = \frac{10}{11}$; the corresponding values of $y_1 = 0.7076310702808$, $y_2 = 0.5740730514719$, $y_3 = 0.46783021167$, $y_4 = 0.378542556741$, $y_5 = 0.3015543695340$, $y_6 = 0.23425520188825$, $y_7 = 0.1749950985426$, $y_8 = 0.12264945225804$, $y_9 = 0.07641143472170$, $y_{10} = 0.0356809280278$ and Min time = 0.586474743322sec

4.2 Interval Newton Method

To address the limitations inherent in the classical Newton-Raphson method, particularly in the presence of multiple roots, the Interval Newton Method extends the traditional approach by incorporating interval arithmetic. Instead of operating on point estimates, this method utilizes intervals to represent the range of possible values, thereby providing guaranteed enclosures for the roots of nonlinear equations. The theoretical basis of interval newton method were introduced in chapter-2. This method not only verifies the existence of a root but also provides a rigorous bound within which the root lies, making it particularly useful for problems where solution verification is critical.

Algorithm for Interval Newton method:

Julia version: 1.11.3

packages: IntervalArithmetic, IntervalBoxes, Plots, ForwardDiff, LinearAlgebra, IntervalArithmetic.Symbols

1. **Initialization:** Define $g = 9.8\,\mathrm{m/s^2}$ and $h = 1\,\mathrm{m}$. Set the initial interval $y_0 = [0, h]$. Create an empty list `solution` to store intervals containing the roots.

2. Define the cost function $F(y)$.

3. Initialize a guess y_0 in the interval.

4. Find F_{dash}, which is the first-order derivative of the cost function.

5. Compute the Jacobian matrix J.

6. Perform Newton's update step:

$$N = \text{mid}(y_0) - J^{-1}F(\text{mid}(y_0))$$

$$y_{\text{new}} = N \cap y_0$$

7. **Check the stopping condition:** If $\|F_{\text{dash}}(y_{\text{new}})\| < 0.0001$, stop and return y_{new}. Otherwise, set $y_0 = y_{\text{new}}$ and repeat from step 4.

Results obtained by Interval Newton method

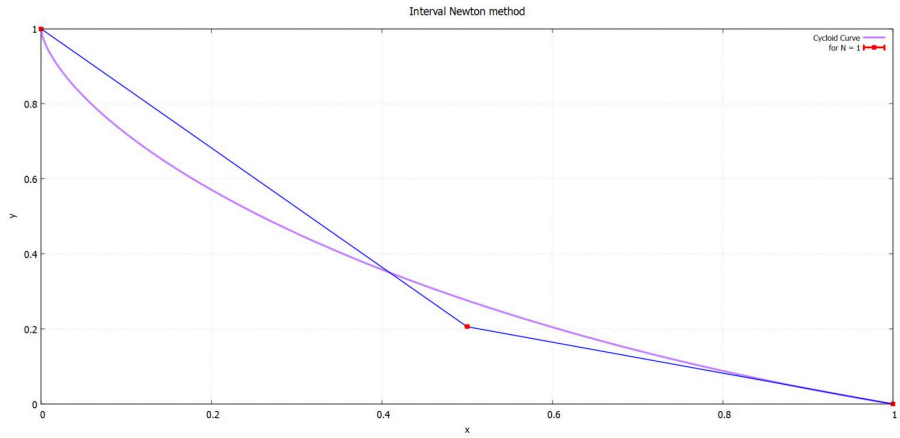

Figure 4.12: Time optimal trajectory using Interval Newton Method for N = 1 with x_1 = 0.5 the range of y_1 = [0, 1], the corresponding value of y_1 = [0.206164, 0.206165] and the minimum time taken = [0.604892, 0.604893]

4.3 Branch-and-Bound Algorithm

The branch-and-bound technique has played a key role in solving discrete and combinatorial optimization problems. It offers a structured approach to reduce the search space efficiently, improving the feasibility of finding optimal solutions for large-scale problems. The algorithm requires no special techniques beyond those used in ordinary linear programming and lends itself to automatic computing, making it a practical tool for solving complex optimization problems. Moreover, this algorithm does not require the calculation of first-order or higher-order partial derivatives to find the optimal solution, which significantly improves its efficiency and reduces computational complexity.

The Branch-and-Bound algorithm is a well-established method used in global optimization to identify the maxima and minima of a function over a specified domain. The method systematically divides the problem's domain into smaller subdomains (branching) and computes bounds on the objective function within these subdomains (bounding). Subdomains that cannot contain the optimal solution are discarded, thereby narrowing the search space.

In the context of the brachistochrone problem, the Branch-and-Bound algorithm can be applied to the discretized version of the problem to efficiently search for the global minimum time trajectory. By combining this method with interval analysis, it is possible to rigorously eliminate subdomains that do not contain feasible solutions, thus enhancing computational efficiency and ensuring the reliability of the obtained solution. The Branch-and-Bound algorithm has been widely used in various fields, including operations research, engineering design, and combinatorial optimization, due to its ability to provide global optimality guarantees.

Algorithm for Branch and Bound method using interval analysis :

packages: IntervalArithmetic, IntervalArithmetic.Symbols, IntervalBoxes, Plots

1. **Initialize** the cost function F and the initial guess, i.e., the entire interval over which we are finding the minimum. Initialize the upper bound as the supremum of the function F over this interval.

2. **Update** the upper bound as:

$$\text{upper bound} = \min(\text{upper bound}, \sup(f(\text{mid}(X))))$$

3. **Check** if $\inf(f(X))$ is less than the current upper bound.

   - If so, bisect the interval into X_1 and X_2, and push them into the working list.
   - Otherwise, discard the interval.

4. **Repeat**: Pop the next interval X from the working list and go back to step-2. Repeat this process for N iterations.

5. **Result**: Minimum time taken = upperbound

Results obtained by Branch and Bound method

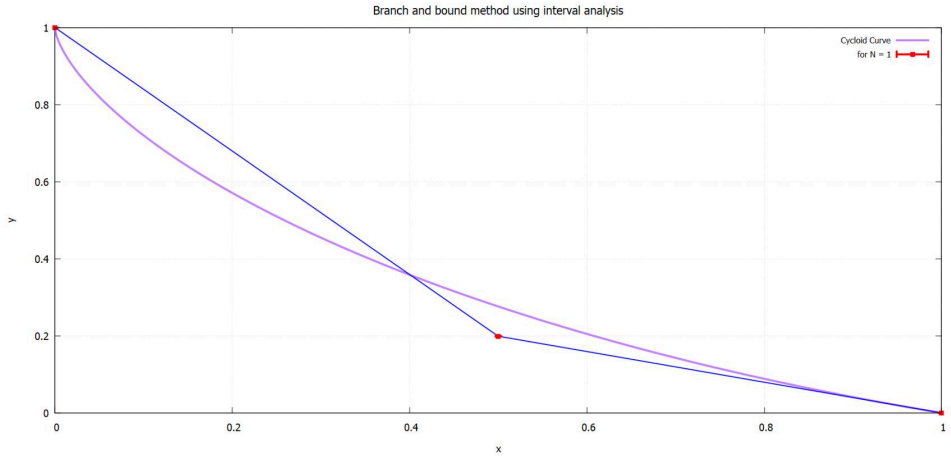

Figure 4.13: Optimal time trajectory using interval-based branch and bound algorithm for a discretization of N = 1.

From Figure 4.13 With $x_1 = 0.5$ the range of $y_1 = [0, 1]$; the corresponding value of $y_1 = [0.199035, 0.199097]$ and the minimum time taken = 0.6048926.

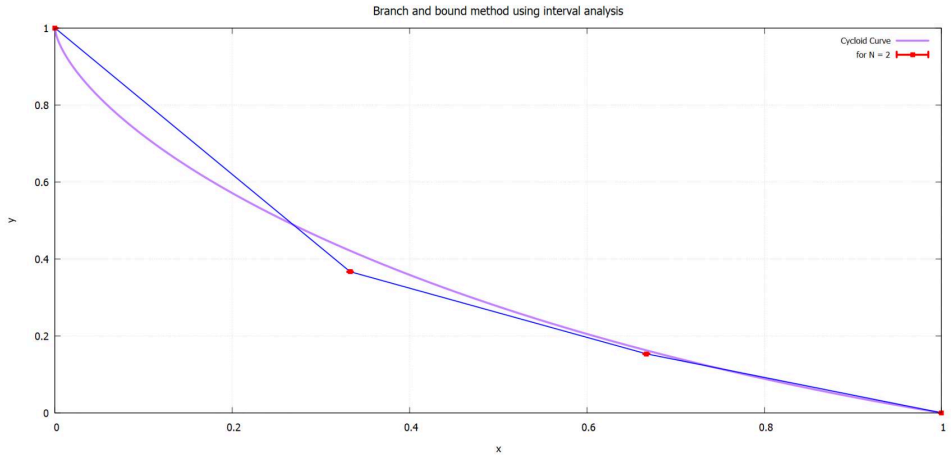

Figure 4.14: Optimal time trajectory using interval-based branch and bound algorithm for a discretization of N = 2

From Figure 4.14 With x_1, $x_2 = 0.333, 0.667$ the range of y_1, $y_2 = [0, 1]$;the corresponding value of $y_1 = [0.36621, 0.367188]$, $y_2 = [0.152343, 0.154297]$ and the minimum time taken = 0.5966720 .

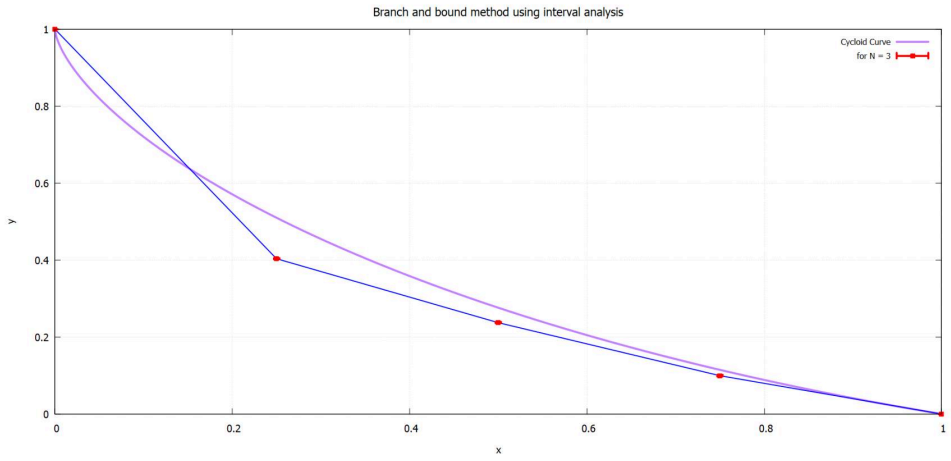

Figure 4.15: Optimal time trajectory using interval-based branch and bound algorithm for a discretization of N = 3

From Figure 4.15 With x_1, x_2, x_3 = 0.25, 0.5, 0.75 the range of y_1, y_2, y_3 = [0, 1]; the corresponding value of y_1 = [0.402343, 0.404297], y_2 = [0.236328, 0.238282], y_3 = [0.0976562, 0.101563] and the minimum time taken = 0.5929443.

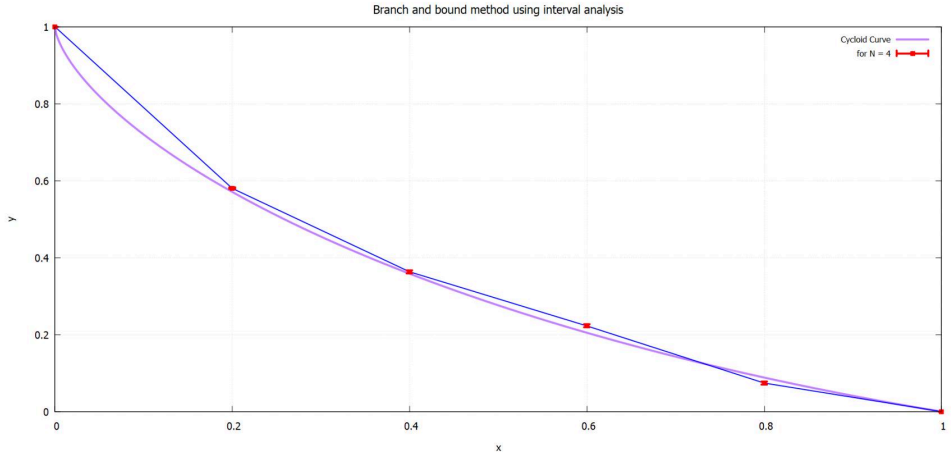

Figure 4.16: Optimal time trajectory using interval-based branch and bound algorithm for a discretization of N = 4

From Figure 4.16 With x_1, x_2, x_3, x_4 = 0.2, 0.4, 0.6, 0.8 the range of y_1, y_2, y_3, y_4 = [0, 1]; the corresponding value of y_1 = [0.578124, 0.582032], y_2 = [0.359374, 0.367188], y_3 = [0.218749, 0.226563], y_4 = [0.0703124, 0.0781251] and the minimum time taken = 0.59082633.

Table 4.1: Comparison of Newton Method, Interval Newton Method, and Branch-and-Bound Algorithm

N	Newton Method	Error	Interval Newton	Error	Branch-and-Bound	Error
1	0.604892	0.022558	[0.604892, 0.604893]	0.037152	0.6048926	0.0225586
2	0.596671	0.014337	-	0.020334	0.5966720	0.014338
3	0.592943	0.010609	-	0.010334	0.5929443	0.0106103
4	0.590819	0.008485	-	0.010334	0.59082633	0.0084923
5	0.589449	0.007115	-	0.010334	-	-
6	0.588493	0.006159	-	0.010334	-	-
7	0.587789	0.005455	-	0.010334	-	-
8	0.587248	0.004914	-	0.010334	-	-
9	0.586821	0.004487	-	0.010334	-	-
10	0.586474	0.00414	-	0.010334	-	-

The "Newton Method" column shows point estimates obtained through standard Newton's method. The "Interval Newton" column lists computed enclosures; the error is calculated as the absolute difference between the computed time and the analytical minimum time (0.582334). For Interval methods, the error is taken as the maximum absolute difference between the endpoints of the interval and the analytical minimum, i.e., the worst-case deviation within the interval. For the Branch-and-Bound method, only the upper bound of the computed time interval was available from the implementation and is used to compute the error. Dashes indicate values not available or not computed for certain values of N. As N increases, Newton method, Interval Newton method, Branch-and-Bound method showed improved accuracy, with reduced errors.

Chapter 5

Discussion and Conclusion

In this work, a study of the brachistochrone problem was carried out using methods based on interval analysis. This classical problem, which seeks the curve of fastest descent under gravity, was used as a test case to explore how numerical uncertainty can be managed through reliable computation.

By using interval-based dynamic programming, the problem was discretized, and minimum time paths were computed while ensuring that all possible values were bounded.The results, compared with the analytical cycloid solution, showed that the error generally decreases as the discretization(N) increases across all dynamic programming methods, indicating improved accuracy with finer grids. Although the extended routing (5-direction DP method) exhibited slightly higher numerical errors and longer computation times compared to the 3-direction method, it produced curves that more closely approximated the theoretical cycloid path.

The interval Newton method together with the branch-and-bound technique enhanced the analysis by tightening the solution bounds and eliminating areas unlikely to contain the optimum. The classical Newton method demonstrates consistent error reduction as discretization increases. The interval Newton method, while providing validated solution bounds, shows slightly higher errors but offers rigorous guarantees on solution. Although the branch-and-bound method does not always yield the lowest error, it effectively narrows the solution space and complements other methods by focusing search efforts on relevant intervals and improving confidence in the results.

In summary, this work demonstrates that interval analysis provides a robust approach for managing uncertainty and delivering verified solutions in optimal control challenges. Combining multiple interval strategies enabled a thorough and trustworthy resolution of the brachistochrone problem. This approach holds promise for application to more intricate systems where dependable solutions are critical.

The interval analysis framework is well-suited for a broader class of optimal control problems. Examples include spacecraft trajectory optimization, robotic motion planning

Bibliography

[1] D. E. Kirk, Optimal Control Theory: An Introduction, Dover Publications, 2004

[2] F. L. Lewis, D. Vrabie, and V. L. Syrmos, Optimal Control, 3rd ed., Wiley, 2012

[3] R. E. Moore Interval analysis, Prentice-Hall, Englewood Cliffs, New Jersey 1966

[4] Daniel Liberzon, Calculus of Variations and Optimal Control

[5] Bellman, R. , Dynamic Programming, Princeton University Press, 1957