The Brachistochrone Problem: a Finite Element Approach *

Guizhong (Harry) Luo

March 2025

Abstract

The Brachistochrone problem seeks the path $y(x)$ between two points that allows a particle sliding under gravity to travel in the minimum possible time. We present a numerical solution using the Finite Element Analysis (FEA) method. The continuous problem, formulated as minimizing the time functional

$$T[y(x)] = \int_0^{x_\mathrm{f}} \frac{\sqrt{1 + (y'(x))^2}}{\sqrt{2gy(x)}} \mathrm{d}x,$$

is discretized using P2 quadratic elements and approximated via Gaussian quadrature, including Gauss-Jacobi quadrature for the initial singularity inherent in the problem formulation. This transforms the variational problem into a finite-dimensional nonlinear optimization problem, which is then optimized using the L-BFGS-B algorithm. The numerical result for endpoints (0,0) to (2,1) is nearly identical to the analytical cycloid solution, with a relative difference in minimum time of only 0.1502%.

1 Introduction

The classic Brachistochrone problem, first stated by Johann Bernoulli in *Acta Eruditorum* in June, 1696, as a novel problem placed before the "finest mathematicians of our time" [1]. He wrote:

> Given two points A and B in a vertical plane, what is the curve traced out by a point acted on only by gravity, which starts at A and reaches B in the shortest time?

This challenge spurred rapid developments in mathematics, with solutions emerging shortly after from prominent figures including Leibniz, L'Hospital, Newton, and the Bernoulli brothers [9].

We establish the standard problem setup. Considering energy conservation, a bead subject only to gravity and moving from rest obtains a speed $v = \sqrt{2gy}$ after a vertical displacement y (assuming y is measured downwards from the starting point A, which is taken as the origin). In standard 2D Euclidean space, the infinitesimal arc length along a path $y(x)$ is given by

$$\mathrm{d}s = \sqrt{1 + (y'(x))^2}\mathrm{d}x,$$

*Honors project for MATH 521, advised by Prof. Chris H. Rycroft

where $y'(x) = \mathrm{d}y/\mathrm{d}x$. The total time T is found by integrating the time differential $\mathrm{d}t = ds/v$ over the path $y(x)$. This yields time as a functional of the path function:

$$T[y(x)] = \int_0^{x_\mathrm{f}} \mathrm{d}t = \int_0^{x_\mathrm{f}} \frac{ds}{v} = \int_0^{x_\mathrm{f}} \frac{\sqrt{1 + (y'(x))^2}}{\sqrt{2gy(x)}} \mathrm{d}x. \tag{1}$$

The solution is famously known to be a segment of an inverted cycloid. This result has been derived through various methods, including analogies to optics via Fermat's principle of least time and, most fundamentally for this work, through the calculus of variations using the Euler-Lagrange equation [3,4]. The cycloid path connecting the origin $(0, 0)$ to an endpoint $(x_\mathrm{f}, y_\mathrm{f})$ can be described parametrically:

$$\begin{aligned} x(\theta) &= a(\theta - \sin\theta) \\ y(\theta) &= a(1 - \cos\theta), \end{aligned}$$

where the parameter a and the range of θ are determined by the specific endpoint $(x_\mathrm{f}, y_\mathrm{f})$.

Interestingly, Galileo Galilei had considered a related problem earlier, in 1638, conjecturing incorrectly that the arc of a circle provided the fastest descent [5]. Nevertheless, his intuition about using a curved path provides a useful starting point for numerical optimization, as we will leverage later for setting our initial guess.

Some 329 years after Bernoulli's initial challenge, we revisit this problem using modern computational tools. This paper details a numerical solution based on the Finite Element Method (FEM), a powerful and versatile technique for approximating solutions to variational problems. While the Brachistochrone problem possesses an elegant analytical solution, applying FEM serves as an excellent benchmark for validating numerical techniques applicable to more complex variational problems where analytical solutions may not be readily available. Furthermore, the FEM framework offers flexibility to potentially explore variations of the problem, such as incorporating friction or analyzing paths within different geometries. The methodology and results of our P2 FEM approach are elaborated in the following sections.

2 Literature Review

Various numerical strategies have been applied beyond the direct analytical approach. These include discrete algorithms that approximate the Euler-Lagrange equations directly, as demonstrated by Pereira, Cruz & Torres [11], who focused on convergence properties. Alternative formulations leverage modern optimization theory, exploring strategies like infinite-dimensional optimistic optimization [7] or convex optimization techniques [3]. Interestingly, historical precursors to finite element ideas can be traced back to early treatments of the Brachistochrone [10]. However, despite the problem's natural fit as a variational challenge well-suited to the Finite Element Method (FEM), detailed applications leveraging specific modern FEM techniques are less commonly foregrounded in comparison.

This work contributes by applying a direct P2 (quadratic) Finite Element discretization to the time functional (Eq. (1)). A key aspect of our approach is the careful numerical integration using Gaussian quadrature. Specifically, we employ Gauss-Jacobi quadrature to accurately handle the integrable singularity arising from the $y(0) = 0$ boundary condition in the first element, complemented by standard Gauss-Legendre quadrature for subsequent elements where the integrand is well-behaved. The resulting finite-dimensional nonlinear optimization problem is then efficiently solved using the L-BFGS-B algorithm, guided by the analytically derived gradient of the discretized functional, providing a robust and accurate numerical framework demonstrated herein.

3 Finite Element Formulation

The core of the Brachistochrone problem is to find a function $y(x)$ that minimizes the time functional $T[y(x)]$ given by Eq. (1), subject to the boundary conditions $y(0) = 0$ and $y(x_f) = y_f$. From a mathematical analysis perspective, this variational problem is naturally posed in a Sobolev space setting. Since the functional involves the first derivative $y'(x)$, the appropriate space is $H^1(0, x_f)$, which consists of functions defined on the interval $(0, x_f)$ that are square-integrable and possess a weak first derivative that is also square-integrable [2,6]. Let V be the affine subspace of $H^1(0, x_f)$ containing functions that satisfy the essential boundary conditions:

$$V = \{v \in H^1(0, x_f) \mid v(0) = 0, v(x_f) = y_f\}.$$

The continuous variational problem is then: Find $y \in V$ such that

$$T[y] \leq T[v] \quad \text{for all } v \in V.$$

Solving this problem directly in the infinite-dimensional space V is generally intractable. The Finite Element Method provides a systematic way to approximate the solution by seeking it within a finite-dimensional subspace $V_h \subset V$.

In this work, we construct the subspace V_h using continuous, piecewise quadratic polynomials defined over a mesh of the domain $[0, x_f]$. This specific choice corresponds to using P2 finite elements. The function $y_h \in V_h$ that minimizes the functional $T[\cdot]$ over this subspace V_h serves as our approximate solution. Minimizing $T[y_h]$ over V_h is equivalent to minimizing the discretized functional $T_h(\mathbf{y}_{\text{int}})$ (derived later as Eq. (15)) with respect to the vector of unknown nodal values $\mathbf{y}_{\text{int}}$.

Now, we describe the construction of this finite element space V_h and the resulting discretized functional T_h. First, we divide the domain $[0, x_{\text{f}}]$ into N uniform finite elements, each of length $h = x_{\text{f}}/N$. We define a total of $2N + 1$ global nodes along the domain. The coordinate of the m-th global node (where $m = 0, 1, \ldots, 2N$) is given by

$$x_m = m\frac{h}{2}.$$

Note that nodes with even indices $m = 2n$ ($n = 0..N$) lie at the boundaries between elements (or domain ends), while nodes with odd indices $m = 2n + 1$ ($n = 0..N - 1$) lie at the midpoints of the elements.

An element e (where $e = 1, 2, \ldots, N$) is defined by three consecutive global nodes: a start node $i = 2(e - 1)$, a midpoint node $j = 2e - 1$, and an end node $k = 2e$. The element thus spans the physical interval $[x_i, x_k]$.

We seek to determine the approximate vertical position y at each global node m. Let $y_m \approx y(x_m)$ denote the nodal value at node m. These y_m values are the fundamental variables parameterizing functions in V_h. The boundary conditions fix the values at the first and last nodes:

$$y_0 = 0 \quad , \quad y_{2N} = y_{\mathrm{f}}.$$

The actual unknowns to be solved for are the values at the interior nodes ($m = 1, 2, \ldots, 2N - 1$). We collect these unknown nodal values into the vector of degrees of freedom:

$$\mathbf{y}_{\mathrm{int}} = [y_1, y_2, \ldots, y_{2N-1}]^\top \in \mathbb{R}^{2N-1}. \tag{2}$$

Thus, the discrete problem involves solving for these $2N - 1$ unknown vertical positions, which define the optimal function y_h within the chosen finite element space V_h.

3.1 Local Coordinate System

To define approximations consistently within each element, it is convenient to map the physical coordinates x belonging to an element e (i.e., $x \in [x_i, x_k]$) to a dimensionless local coordinate $\xi \in [-1, 1]$. The mapping places the local origin $\xi = 0$ at the element's midpoint node x_j:

$$x(\xi) \coloneqq \frac{x_i + x_k}{2} + \frac{x_k - x_i}{2}\xi = x_j + \frac{h}{2}\xi. \tag{3}$$

Here, the local coordinate $\xi = -1$ corresponds to the element's start node x_i, $\xi = 0$ corresponds to the midpoint node x_j, and $\xi = 1$ corresponds to the end node x_k.

The Jacobian of this transformation relates the physical and local differentials:

$$J = \frac{\mathrm{d}x}{\mathrm{d}\xi} = \frac{x_k - x_i}{2} = \frac{h}{2}.$$

Hence, the differential transformation is

$$\mathrm{d}x = J\mathrm{d}\xi = \frac{h}{2}\mathrm{d}\xi.$$

3.2 Quadratic Shape Functions for Approximating $y(x)$ and $y'(x)$

Within element e, let the nodal values corresponding to its start, mid, and end nodes be collected in the element nodal vector $\mathbf{y}_e = [y_i, y_j, y_k]^T$, where i, j, k are the global indices for element e.

We approximate the function $y(x)$ within this element using an interpolation $y_h(x)$ based on these nodal values and quadratic shape functions of the local coordinate ξ:

$$y_h\left(x(\xi)\right) = N_1(\xi)y_i + N_2(\xi)y_j + N_3(\xi)y_k.$$

The quadratic shape functions $N_1(\xi), N_2(\xi), N_3(\xi)$ must satisfy the interpolation property

$$N_m(\xi_n) = \delta_{mn} \quad \text{for } m, n = 1, 2, 3,$$

where we associate local node numbers $1, 2, 3$ with local coordinates $\xi_1 = -1, \quad \xi_2 = 0, \quad \xi_3 = 1$, and where δ_{mn} denotes the Kronecker delta. This property ensures that the approximation y_h exactly matches the nodal values at the element's nodes: $y_h(x_i) = y_i$, $y_h(x_j) = y_j$, $y_h(x_k) = y_k$.

The unique quadratic polynomials satisfying these conditions are the Lagrange polynomials on $[-1, 1]$ for nodes at $-1, 0, 1$:

$$\begin{aligned}
N_1(\xi) &= \frac{(\xi - 0)(\xi - 1)}{(-1 - 0)(-1 - 1)} = \frac{1}{2}\xi(\xi - 1), \\
N_2(\xi) &= \frac{(\xi - (-1))(\xi - 1)}{(0 - (-1))(0 - 1)} = \frac{(\xi + 1)(\xi - 1)}{-1} = 1 - \xi^2, \\
N_3(\xi) &= \frac{(\xi - (-1))(\xi - 0)}{(1 - (-1))(1 - 0)} = \frac{1}{2}\xi(\xi + 1).
\end{aligned}$$

The element approximation can be written compactly using vector notation:

$$y_h\left(x(\xi)\right) = \mathbf{N}(\xi) \cdot \mathbf{y}_e, \tag{4}$$

where $\mathbf{N}(\xi) = [N_1(\xi),\ N_2(\xi),\ N_3(\xi)]$ is the vector of shape functions.

Since the time functional $T[y(x)]$ also depends on the derivative $y'(x)$, we approximate this with the derivative of the interpolation, $y_h'(x)$. Using the chain rule:

$$y_h'(x) = \frac{\mathrm{d}y_h}{\mathrm{d}x} = \frac{\mathrm{d}y_h}{\mathrm{d}\xi}\frac{\mathrm{d}\xi}{\mathrm{d}x}.$$

Notice

$$\frac{\mathrm{d}y_h}{\mathrm{d}\xi} = \frac{d}{\mathrm{d}\xi}(\mathbf{N}(\xi) \cdot \mathbf{y}_e) = \left(\frac{\mathrm{d}\mathbf{N}}{\mathrm{d}\xi}\right) \cdot \mathbf{y}_e,$$

where $\frac{d\mathbf{N}}{\mathrm{d}\xi} = \left[\xi - \frac{1}{2},\ -2\xi,\ \xi + \frac{1}{2}\right]$. Using the inverse Jacobian $\frac{\mathrm{d}\xi}{\mathrm{d}x} = 1/J = 2/h$, the approximation for the physical derivative becomes:

$$y_h'(x(\xi)) = \frac{2}{h}\left(\frac{\mathrm{d}\mathbf{N}}{\mathrm{d}\xi} \cdot \mathbf{y}_e\right) = \frac{2}{h}\left(\left[\xi - \frac{1}{2},\ -2\xi,\ \xi + \frac{1}{2}\right]\mathbf{y}_e\right). \tag{5}$$

A visualization of this finite-element mesh can be found in Figure 1.

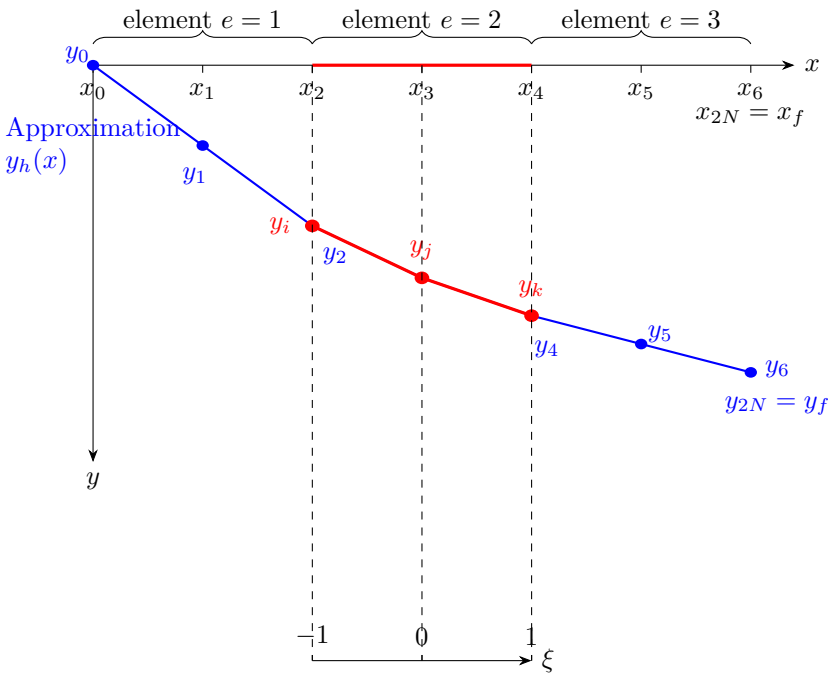

Figure 1: P_2 finite-element mesh for the brachistochrone. Global nodes $x_0, \ldots, x_{2N}$; piecewise-quadratic $y_h(x)$ (blue); element $e = 2$ with local nodes i, j, k (red); local coordinate $\xi \in [-1, 1]$.

3.3 Discretized Functional

For numerical optimization, we minimize the functional T^* which omits the constant factor $\frac{1}{\sqrt{2g}}$ from the original total time functional (Eq. (1)) to get:

$$T^*[y(x)] = \int_0^{x_\mathrm{f}} \sqrt{\frac{1 + (y'(x))^2}{y(x)}} \mathrm{d}x. \tag{6}$$

We approximate this functional by replacing the exact function $y(x)$ and its derivative $y'(x)$ with their finite element approximations $y_h(x)$ and $y'_h(x)$. The integral over the full domain $[0, x_\mathrm{f}]$ is computed as the sum of integrals over each element e:

$$T^*[y(x)] \approx T_h(\mathbf{y}_\mathrm{int}) = \sum_{e=1}^{N} T_e^*(\mathbf{y}_e) \tag{7}$$

where T_e^* is the contribution from element e, whose associated global nodes are i, j, k:

$$T_e^*(\mathbf{y}_e) = \int_{x_i}^{x_k} \sqrt{\frac{1 + (y'_h(x))^2}{y_h(x)}} \mathrm{d}x. \tag{8}$$

The total approximate functional T_h is now expressed solely in terms of the nodal values $\mathbf{y}_m$, specifically the unknown ones contained in the vector $\mathbf{y}_\mathrm{int}$ (Eq. (2)).

3.4 Element Integrals T_e^*

We transform the integral T_e^* (Eq. (8)) to the local coordinate system using the change of variables $x = x(\xi)$ and $\mathrm{d}x = J\mathrm{d}\xi$, where $\xi \in [-1, 1]$. We also make the substitution of the element approximations for y_h and y_h' as derived in Eq. (4) and (5).

Substituting these into the integral for T_e^* (Eq. (8)):

$$T_e^*(\mathbf{y}_e) = \int_{-1}^{1} \left(\frac{h}{2}\right) \sqrt{\frac{1 + \left[\frac{2}{h}\left(\frac{d\mathbf{N}}{\mathrm{d}\xi}(\xi) \cdot \mathbf{y}_e\right)\right]^2}{\mathbf{N}(\xi) \cdot \mathbf{y}_e}} \mathrm{d}\xi$$

For simplicity, we define the following terms evaluated at a specific local coordinate ξ :

$$Y(\xi, \mathbf{y}_e) \coloneqq \mathbf{N}(\xi) \cdot \mathbf{y}_e \,, \tag{9}$$

$$Y'(\xi, \mathbf{y}_e) \coloneqq \frac{2}{h}\left(\frac{d\mathbf{N}}{\mathrm{d}\xi}(\xi) \cdot \mathbf{y}_e\right), \tag{10}$$

$$f_e(\xi, \mathbf{y}_e) \coloneqq \sqrt{\frac{1 + [Y'(\xi, \mathbf{y}_e)]^2}{Y(\xi, \mathbf{y}_e)}}. \tag{11}$$

Then the element integral is written in compact form as

$$T_e^*(\mathbf{y}_e) = \int_{-1}^{1} \frac{h}{2} f_e(\xi, \mathbf{y}_e)\mathrm{d}\xi. \tag{12}$$

4 Numerical integration via Gauss Quadrature

The element integral in Eq. (12) cannot generally be solved analytically, necessitating numerical approximation. We employ Gaussian Quadrature, which approximates an integral over $[-1, 1]$ as a weighted sum of the integrand evaluated at specific points (quadrature points) within the interval:

$$\int_{-1}^{1} g(\xi)\mathrm{d}\xi \approx \sum_{p=1}^{P} w_p g(\xi_p),$$

where P is the number of quadrature points, w_p are the quadrature weights, and ξ_p are the P distinct quadrature points within $(-1, 1)$. The choice of points and weights depends on any weighting function present in the integral.

4.1 Gauss-Legendre Quadrature for $e > 1$

For elements $e = 2, 3, ..., N$, the path $y_h(x)$ is expected to be strictly positive, meaning $y_h(x(\xi)) > 0$ for $\xi \in [-1, 1]$. In these elements, the integrand $f_e(\xi, \mathbf{y}_e)$ (Eq. (11)) has no singularities within or at the ends of the integration interval $[-1, 1]$. For such regular integrals, standard Gauss-Legendre quadrature is highly efficient.

Let (ξ_p^L, w_p^L) (for $p = 1, \ldots, P$) denote the Gauss-Legendre quadrature points and weights. The element integral T_e^* for $e > 1$ is approximated as:

$$T_e^*(\mathbf{y}_e) \approx \sum_{p=1}^{P} w_p^L \frac{h}{2} \left(f_e(\xi_p^L, \mathbf{y}_e) \right).$$

Substituting the definition of f_e (Eq. (11)), and including a small positive constant ϵ to prevent potential division by zero during numerical evaluation (although theoretically $Y > 0$ here), we have:

$$T_e^*(\mathbf{y}_e) \approx \sum_{p=1}^{P} w_p^L \frac{h}{2} \left(\sqrt{\frac{1 + [Y'(\xi_p^L, \mathbf{y}_e)]^2}{\max(Y(\xi_p^L, \mathbf{y}_e), \epsilon)}} \right) \quad \text{for } e = 2, \ldots, N. \tag{13}$$

4.2 Gauss-Jacobi Quadrature for First Element

The first element ($e = 1$) requires special attention due to the boundary condition $y_0 = y(0) = 0$. This introduces an integrable singularity into the functional.

Recall the approximation within this element, where $\mathbf{y}_1 = [y_0, y_1, y_2]^\top = [0, y_1, y_2]^\top$:

$$\begin{aligned} Y(\xi, \mathbf{y}_1) &= N_1(\xi) y_0 + N_2(\xi) y_1 + N_3(\xi) y_2 \\ &= (1 - \xi^2) y_1 + \frac{1}{2} \xi(\xi + 1) y_2 \\ &= (1 + \xi) \left[(1 - \xi) y_1 + \frac{1}{2} \xi \, y_2 \right]. \end{aligned}$$

As $\xi \to -1$ (corresponding to the start point $x = 0$), the term $(1 + \xi) \to 0$. Let $H(\xi, \mathbf{y}_1) \coloneqq (1-\xi)y_1 + \frac{1}{2}\xi y_2$. For a physically meaningful path descending from the origin, we expect $y_1, y_2 > 0$, and typically $H(-1, \mathbf{y}_1) = 2y_1 - 0.5y_2 \neq 0$ (unless the path initially moves horizontally, which is not optimal). The integrand $f_1(\xi, \mathbf{y}_1)$ in Eq. (11) contains the term $1/\sqrt{Y(\xi, \mathbf{y}_1)}$, which behaves like:

$$\frac{1}{\sqrt{Y(\xi, \mathbf{y}_1)}} = \frac{1}{\sqrt{(1 + \xi) H(\xi, \mathbf{y}_1)}} \propto (1 + \xi)^{-1/2} \quad \text{as } \xi \to -1.$$

This $(1 + \xi)^{-1/2}$ behavior indicates an integrable singularity at $\xi = -1$. Standard Gauss-Legendre quadrature is less effective for integrands with endpoint singularities like this, potentially leading to slow convergence or inaccurate results, hence the choice of a specialized rule. This invites the application of Gauss-Jacobi quadrature, designed for integrals with weights of the form $(1-x)^\alpha(1+x)^\beta$.

The general Gauss-Jacobi rule approximates:

$$\int_{-1}^{1} (1 - x)^\alpha (1 + x)^\beta g(x) \mathrm{d}x \approx \sum_{p=1}^{P} w'_p g(x'_p),$$

where x'_p are roots of the Jacobi polynomial $P_P^{(\alpha,\beta)}(x)$, and w'_p are the corresponding weights. In

our case, the integrand has the form $(1+\xi)^{-1/2}$ multiplying a remaining smooth function. Thus, we set $\alpha = 0, \beta = -1/2$. The specific rule becomes:

$$\int_{-1}^{1} (1+\xi)^{-1/2} g(\xi) \mathrm{d}\xi \approx \sum_{p=1}^{P} w_p^J g(\xi_p^J),$$

where ξ_p^J are the roots of the P-th degree Jacobi polynomial $P_P^{(0,-1/2)}(\xi)$, and w_p^J are the corresponding weights for the weight function $(1+\xi)^{-1/2}$.

To apply this, we rewrite the element integral T_1^* (from Eq. (12)) by factoring out the singular term:

$$\begin{aligned} T_1^*(\mathbf{y}_1) &= \int_{-1}^{1} \frac{h}{2} \sqrt{\frac{1+[Y'(\xi,\mathbf{y}_1)]^2}{Y(\xi,\mathbf{y}_1)}} \mathrm{d}\xi \\ &= \int_{-1}^{1} \frac{h}{2} \sqrt{\frac{1+[Y'(\xi,\mathbf{y}_1)]^2}{(1+\xi)H(\xi,\mathbf{y}_1)}} \mathrm{d}\xi \\ &= \int_{-1}^{1} (1+\xi)^{-1/2} \underbrace{\left[\frac{h}{2} \sqrt{\frac{1+[Y'(\xi,\mathbf{y}_1)]^2}{H(\xi,\mathbf{y}_1)}} \right]}_{g(\xi)} \mathrm{d}\xi. \end{aligned}$$

Applying the Gauss-Jacobi quadrature rule to this form yields:

$$T_1^*(\mathbf{y}_1) \approx \sum_{p=1}^{P} w_p^J \left(\frac{h}{2} \sqrt{\frac{1+[Y'(\xi_p^J,\mathbf{y}_1)]^2}{H(\xi_p^J,\mathbf{y}_1)}} \right),$$

where $H(\xi_p^J, \mathbf{y}_1) = (1-\xi_p^J)y_1 + \frac{1}{2}\xi_p^J y_2$.

Introducing the small positive constant ϵ to avoid potential division by zero in the numerical evaluation of H, we replace the denominator term $H(\xi_p^J, \mathbf{y}_1)$ with $\max(H(\xi_p^J, \mathbf{y}_1), \epsilon)$. This gives the final approximation for the first element integral:

$$T_1^*(\mathbf{y}_1) \approx \sum_{p=1}^{P} w_p^J \frac{h}{2} \left(\sqrt{\frac{1+[Y'(\xi_p^J,\mathbf{y}_1)]^2}{\max(H(\xi_p^J,\mathbf{y}_1),\epsilon)}} \right). \tag{14}$$

Combining the treatments for the first and subsequent elements, we arrive at the final discretized functional T_h, which approximates the modified time functional T^* (Eq. (6)):

$$T^*[y(x)] \approx T_h(\mathbf{y}_{\text{int}}) = T_1^*(\mathbf{y}_1) + \sum_{e=2}^{N} T_e^*(\mathbf{y}_e), \tag{15}$$

where the element vectors $\mathbf{y}_e$ are defined as:

$$\begin{aligned}\mathbf{y}_1 &= [0, y_1, y_2]^\top, \\ \mathbf{y}_e &= [y_{2(e-1)}, y_{2e-1}, y_{2e}]^\top \quad \text{for } e = 2, \ldots, N-1, \\ \mathbf{y}_N &= [y_{2N-2}, y_{2N-1}, y_{\mathrm{f}}]^\top,\end{aligned}$$

and the element contributions T_1^* and T_e^* (for $e > 1$) are approximated using Gauss-Jacobi (Eq. (14)) and Gauss-Legendre (Eq. (13)) quadrature, respectively.

5 Minimizing T_h for Fastest Descent

The Brachistochrone problem seeks the path of fastest descent, which mathematically corresponds to minimizing the time functional $T^*[y(x)]$ (Eq. (6)) over the space of admissible paths $y(x)$. In our Finite Element framework, this translates to minimizing the discretized functional $T_h(\mathbf{y}_{\text{int}})$ (Eq. (15)) with respect to the vector of unknown interior nodal values $\mathbf{y}_{\text{int}}$. Thus, we aim to solve the optimization problem:

$$\min_{\mathbf{y}_{\text{int}} \in \mathbb{R}^{2N-1}} T_h(\mathbf{y}_{\text{int}}), \quad \text{subject to } y_m \geq \epsilon \text{ for } m = 1, \ldots, 2N-1. \tag{16}$$

The constraints $y_m \geq \epsilon$ ensure the path stays below the starting point and maintain numerical stability.

Gradient-based optimization algorithms are highly effective for such problems. We therefore compute the gradient of the discretized functional T_h with respect to the unknown variables $\mathbf{y}_{\text{int}}$:

$$\nabla T_h(\mathbf{y}_{\text{int}}) = \left[\frac{\partial T_h}{\partial y_1}, \frac{\partial T_h}{\partial y_2}, \ldots, \frac{\partial T_h}{\partial y_{2N-1}}\right]^\top. \tag{17}$$

Since $T_h = \sum_{e=1}^{N} T_e^*$, the m-th component of the global gradient is obtained by summing contributions from the elements that contain node m:

$$\frac{\partial T_h}{\partial y_m} = \sum_{e \text{ s.t. node } m \text{ in element } e} \frac{\partial T_e^*}{\partial y_m}.$$

This structure allows for efficient computation via an assembly process. It is computationally advantageous to first compute the gradient of each element's contribution T_e^* with respect to its own local nodal vector $\mathbf{y}_e = [y_i, y_j, y_k]^T$. We denote this 3-component element gradient vector as:

$$\nabla_{\mathbf{y}_e} T_e^* = \left[\frac{\partial T_e^*}{\partial y_i}, \frac{\partial T_e^*}{\partial y_j}, \frac{\partial T_e^*}{\partial y_k}\right]^T.$$

The global gradient $\nabla T_h(\mathbf{y}_{\text{int}})$ is then constructed by assembling these element gradient vectors into the appropriate positions corresponding to the interior nodes.

5.1 Gradient of T_e^* for $e > 1$

We start with the expression for T_e^* approximated using Gauss-Legendre quadrature (Eq. (13)):

$$T_e^*(\mathbf{y}_e) \approx \sum_{p=1}^{P} w_p^L \frac{h}{2} \left(\sqrt{\frac{1 + [Y'(\xi_p^L, \mathbf{y}_e)]^2}{\max(Y(\xi_p^L, \mathbf{y}_e), \epsilon)}} \right).$$

Let $F_p(\mathbf{y}_e) \coloneqq 1 + [Y'(\xi_p^L, \mathbf{y}_e)]^2$ and $Y_p(\mathbf{y}_e) \coloneqq \max(Y(\xi_p^L, \mathbf{y}_e), \epsilon)$. Then Eq. (13) becomes:

$$T_e^*(\mathbf{y}_e) \approx \sum_{p=1}^{P} w_p^L \frac{h}{2} \left(\sqrt{\frac{F_p}{Y_p}} \right). \tag{18}$$

The gradient of T_e^* is the sum of the gradients of each term in the quadrature sum. We need to compute $\nabla_{\mathbf{y}_e} \left(\sqrt{F_p/Y_p} \right)$. Applying the chain rule and quotient rule, assuming $Y_p(\mathbf{y}_e) > \epsilon$:

$$\begin{aligned}
\nabla_{\mathbf{y}_e} \left(\sqrt{\frac{F_p}{Y_p}} \right) &= \frac{1}{2\sqrt{F_p/Y_p}} \nabla_{\mathbf{y}_e} \left(\frac{F_p}{Y_p} \right) \\
&= \frac{1}{2} \sqrt{\frac{Y_p}{F_p}} \left(\frac{(\nabla_{\mathbf{y}_e} F_p) Y_p - F_p (\nabla_{\mathbf{y}_e} Y_p)}{Y_p^2} \right) \\
&= \frac{1}{2Y_p^2} \sqrt{\frac{Y_p}{F_p}} \left[(\nabla_{\mathbf{y}_e} F_p) Y_p - F_p (\nabla_{\mathbf{y}_e} Y_p) \right].
\end{aligned}$$

We require the gradients of F_p and Y_p with respect to the element nodal vector $\mathbf{y}_e$. Recall $Y(\xi, \mathbf{y}_e) = \mathbf{N}(\xi)^T \mathbf{y}_e$ and $Y'(\xi, \mathbf{y}_e) = \frac{2}{h} (\frac{d\mathbf{N}}{d\xi}(\xi))^T \mathbf{y}_e$.

- Gradient of Y_p: Assuming $Y_p > \epsilon$,

  $$\nabla_{\mathbf{y}_e} Y_p = \nabla_{\mathbf{y}_e} (\mathbf{N}(\xi_p^L)^T \mathbf{y}_e) = \mathbf{N}(\xi_p^L).$$

  Let $\mathbf{N}_p \coloneqq \mathbf{N}(\xi_p^L)$. (Note: gradient is a column vector, shape functions N are row vectors usually, so transpose is implicit/explicit depending on convention. Assuming ∇ produces column vector, $\mathbf{N}_p$ is column).

- Gradient of F_p:

  $$\nabla_{\mathbf{y}_e} F_p = \nabla_{\mathbf{y}_e} (1 + [Y'(\xi_p^L, \mathbf{y}_e)]^2) = 2Y'(\xi_p^L, \mathbf{y}_e)(\nabla_{\mathbf{y}_e} Y'(\xi_p^L, \mathbf{y}_e))$$

  where the gradient of Y' is:

  $$\nabla_{\mathbf{y}_e} Y'(\xi_p^L, \mathbf{y}_e) = \nabla_{\mathbf{y}_e} \left(\frac{2}{h} \left(\frac{d\mathbf{N}}{d\xi}(\xi_p^L) \right)^T \mathbf{y}_e \right) = \frac{2}{h} \left(\frac{d\mathbf{N}}{d\xi}(\xi_p^L) \right).$$

Let $\mathbf{N}'_p \coloneqq \left(\frac{d\mathbf{N}}{d\xi}(\xi_p^L)\right)$. Then,

$$\nabla_{\mathbf{y}_e} F_p = 2Y'(\xi_p^L, \mathbf{y}_e)\frac{2}{h}\mathbf{N}'_p.$$

Let $Y'_p = Y'(\xi_p^L, \mathbf{y}_e)$. Substituting these back, the gradient contribution from a single quadrature point is:

$$\nabla_{\mathbf{y}_e}\left(\sqrt{\frac{F_p}{Y_p}}\right) = \frac{1}{2Y_p^2}\sqrt{\frac{Y_p}{F_p}}\left[\left(2Y'_p\frac{2}{h}\mathbf{N}'_p\right)Y_p - F_p\mathbf{N}_p\right].$$

The full element gradient vector $\nabla_{\mathbf{y}_e} T_e^*$ for $e > 1$ is obtained by summing these contributions over all Gauss-Legendre quadrature points, weighted by $w_p^L(h/2)$:

$$\nabla_{\mathbf{y}_e} T_e^* \approx \sum_{p=1}^{P} w_p^L\left(\frac{h}{4Y_p^2}\sqrt{\frac{Y_p}{F_p}}\left[\left(2Y'_p\frac{2}{h}\mathbf{N}'_p\right)Y_p - F_p\mathbf{N}_p\right]\right). \tag{19}$$

Note that if $Y_p(\mathbf{y}_e)$ hits ϵ, the gradient of the max function is technically undefined or needs careful handling (e.g., subgradient), but in practice for $e > 1$ this is unlikely if ϵ is small.

5.2 Gradient of T_1^* for the First Element ($e = 1$)

For the first element, we used Gauss-Jacobi quadrature and the denominator involved $H_p = \max(H(\xi_p^J, \mathbf{y}_1), \epsilon)$, where $H(\xi, \mathbf{y}_1) = (1-\xi)y_1 + \frac{1}{2}\xi y_2$ and $\mathbf{y}_1 = [0, y_1, y_2]^T$.

$$T_1^*(\mathbf{y}_1) \approx \sum_{p=1}^{P} w_p^J \frac{h}{2}\left(\sqrt{\frac{F_p}{H_p}}\right),$$

where $F_p = 1 + [Y'(\xi_p^J, \mathbf{y}_1)]^2$. The gradient calculation follows the same structure as before, requiring $\nabla_{\mathbf{y}_1} F_p$ and $\nabla_{\mathbf{y}_1} H_p$. Note that the gradient is with respect to the 3-component vector $\mathbf{y}_1 = [y_0, y_1, y_2]^T$, even though $y_0 = 0$.

- Gradient of H_p: Since H does not depend on y_0, and assuming $H(\xi_p^J, \mathbf{y}_1) > \epsilon$, we have:

$$\nabla_{\mathbf{y}_1} H_p = \left[\frac{\partial H_p}{\partial y_0}, \frac{\partial H_p}{\partial y_1}, \frac{\partial H_p}{\partial y_2}\right]^T = \left[0, \quad 1-\xi_p^J, \quad \frac{1}{2}\xi_p^J\right]^T.$$

  Let $\mathbf{H}'_p \coloneqq \nabla_{\mathbf{y}_1} H_p$.

- Gradient of F_p: We need $\nabla_{\mathbf{y}_1} Y'(\xi_p^J, \mathbf{y}_1)$. Recall from Eq. (5) applied to element 1 ($y_0 = 0$): $Y'(\xi, \mathbf{y}_1) = \frac{2}{h}((-2\xi)y_1 + (\xi + 1/2)y_2)$. The gradient with respect to $\mathbf{y}_1 = [y_0, y_1, y_2]^T$ is:

$$\nabla_{\mathbf{y}_1} Y'(\xi_p^J, \mathbf{y}_1) = \left[\frac{\partial Y'}{\partial y_0}, \frac{\partial Y'}{\partial y_1}, \frac{\partial Y'}{\partial y_2}\right]^T = \frac{2}{h}\left[0, \quad -2\xi_p^J, \quad \xi_p^J + \frac{1}{2}\right]^T.$$

Let $\mathbf{Y}'_{p,1} \coloneqq \nabla_{\mathbf{y}_1} Y'(\xi_p^J, \mathbf{y}_1)$. (Changed notation slightly from original draft for clarity). Then,

$$\nabla_{\mathbf{y}_1} F_p = 2Y'(\xi_p^J, \mathbf{y}_1)(\nabla_{\mathbf{y}_1} Y'(\xi_p^J, \mathbf{y}_1)) = 2Y'_p \mathbf{Y}'_{p,1}.$$

Let $Y'_p = Y'(\xi_p^J, \mathbf{y}_1)$. The gradient contribution from a single Gauss-Jacobi point is (assuming $H_p > \epsilon$):

$$\nabla_{\mathbf{y}_1} \left(\sqrt{\frac{F_p}{H_p}} \right) = \frac{1}{2H_p^2} \sqrt{\frac{H_p}{F_p}} \left[(\nabla_{\mathbf{y}_1} F_p) H_p - F_p (\nabla_{\mathbf{y}_1} H_p) \right].$$

Summing over the quadrature points gives the element gradient vector $\nabla_{\mathbf{y}_1} T_1^*$:

$$\nabla_{\mathbf{y}_1} T_1^* \approx \sum_{p=1}^{P} w_p^J \left(\frac{h}{4H_p^2} \sqrt{\frac{H_p}{F_p}} \left[(2Y'_p \mathbf{Y}'_{p,1}) H_p - F_p \mathbf{H}'_p \right] \right). \tag{20}$$

5.3 Global Gradient Assembly

The global gradient vector $\nabla T_h(\mathbf{y}_{\text{int}}) \in \mathbb{R}^{2N-1}$ is assembled by summing the contributions from the relevant element gradient vectors. Let $\mathbf{G} \coloneqq \nabla T_h(\mathbf{y}_{\text{int}})$ denote this global vector, indexed such that its m-th component is $G_m = \partial T_h / \partial y_m$ for $m = 1, \ldots, 2N-1$. The assembly process calculates $\mathbf{G}$ by iterating through the elements, adding each element gradient's components to the corresponding global degrees of freedom.

Initialize $\mathbf{G}$ as a zero vector of size $2N-1$. Then, for each element $e = 1, \ldots, N$:

1. Compute the 3-component element gradient vector $\mathbf{g}_e \coloneqq \nabla_{\mathbf{y}_e} T_e^* = [g_{e,i}, g_{e,j}, g_{e,k}]^T$ using Eq. (20) (for $e = 1$) or Eq. (19) (for $e > 1$). Here $i = 2(e-1), j = 2e-1, k = 2e$ are the global node indices for element e.

2. Add the components of $\mathbf{g}_e$ to the global gradient vector $\mathbf{G}$ at the entries corresponding to the **unknown interior** nodes:

   - If node i is an interior node (i.e., $i > 0$), perform the update: $G_i \leftarrow G_i + g_{e,i}$. (Node $i = 0$ corresponds to y_0, which is fixed).
   - Node $j = 2e-1$ is always an interior node for $e = 1..N$, so its global index is j. Perform the update: $G_j \leftarrow G_j + g_{e,j}$.
   - If node k is an interior node (i.e., $k < 2N$), perform the update: $G_k \leftarrow G_k + g_{e,k}$. (Node $k = 2N$ corresponds to $y_{2N} = y_f$, which is fixed).

After iterating through all elements, the vector $\mathbf{G}$ holds the complete gradient $\nabla T_h(\mathbf{y}_{\text{int}})$. This assembled gradient is then used by the optimization algorithm.

6 Numerical Solution and Results

With the discretized objective function $T_h(\mathbf{y}_{\text{int}})$ (Eq. (15)) and its analytical gradient $\nabla T_h(\mathbf{y}_{\text{int}})$ (Eq. (17)) derived, the continuous variational problem of finding the Brachistochrone curve has

been effectively transformed into a finite-dimensional nonlinear optimization problem (Eq. (16)).

To solve this optimization problem, we employ the Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with bound constraints (L-BFGS-B), as implemented in `SciPy`. This quasi-Newton method is well-suited for this task as it efficiently utilizes the gradient information we derived and can directly handle the simple lower-bound constraints $y_m \geq \epsilon$ on the nodal values. The L-BFGS-B algorithm iteratively refines the solution vector $\mathbf{y}_{\text{int}}$ by repeatedly evaluating the objective function T_h and its gradient ∇T_h.

The precise computational procedure, encompassing the initialization, the element-wise calculation and assembly of T_h and its gradient via Gaussian quadrature (using Gauss-Jacobi for the first element and Gauss-Legendre otherwise), and the iterative optimization loop driven by L-BFGS-B, is formalized in Appendix A (Algorithm 1).

For the practical implementation of this algorithm, several choices were made. We consider the specific case with endpoints $(x_0, y_0) = (0, 0)$ and $(x_f, y_f) = (2, 1)$. Following Galileo's early insight, we selected a circular arc passing through these points as the initial guess $\mathbf{y}_{\text{int}}^{(0)}$ for the optimization process. The simulation was configured with $N = 20$ quadratic elements, resulting in $2N + 1 = 41$ total nodes and $2N - 1 = 39$ unknown interior nodal values (degrees of freedom). We used $P = 10$ quadrature points for both the Gauss-Legendre and Gauss-Jacobi integrations within each element. The numerical stability constant was set to $\epsilon = 1 \times 10^{-15}$. The L-BFGS-B optimizer was configured to terminate when the norm of the gradient vector $\mathbf{G}$ fell below a tolerance of 10^{-7}. The Python code implementing this entire procedure, utilizing `NumPy` and `SciPy`, is provided in Appendix B.

Executing Algorithm 1 with these parameters yields the optimal nodal values $\mathbf{y}_{\text{int}}^{*}$. The resulting Brachistochrone path, reconstructed by plotting the quadratic interpolation function $y_h(x)$ across all elements based on these optimal nodal values, is visualized alongside the analytical cycloid solution in Figure 1.

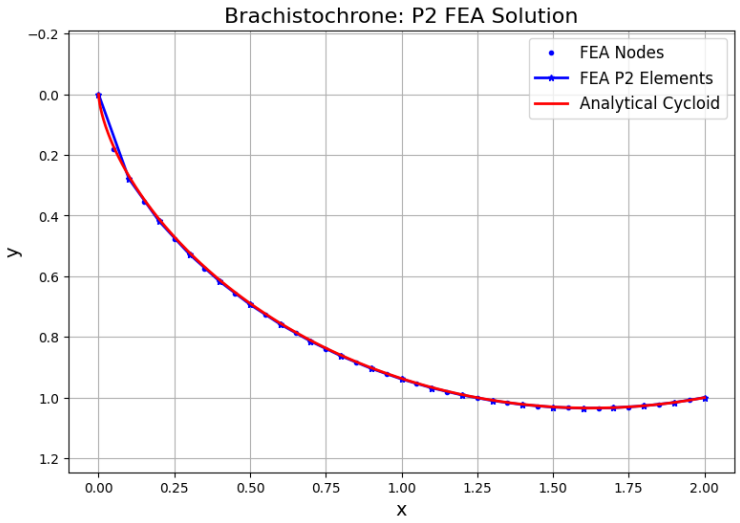

Figure 2: Comparison of the P2 FEA solution (blue nodes and interpolated curve) with the analytical cycloid solution (read curve) for endpoints $(0, 0)$ and $(x_f, y_f) = (2, 1)$.

Figure 2 demonstrates excellent qualitative agreement between the path determined by our P2 Finite Element approach and the exact cycloid curve. To assess the quantitative accuracy, we compare the minimum descent time. The time computed from the FEA solution is $T_{\text{FEA}} = T_h(\mathbf{y}^*_{\text{int}})/\sqrt{2g} \approx 0.806744$ seconds (using $g = 9.81\, m/s^2$). This value aligns remarkably well with the theoretical minimum time calculated from the analytical cycloid solution for these endpoints, $T_{\text{Exact}} \approx 0.805564$ seconds. The relative difference between the numerical approximation and the exact analytical result is a mere

$$\frac{|T_{\text{FEA}} - T_{\text{Exact}}|}{T_{\text{Exact}}} \times 100\% \approx 0.1502\%,$$

underscoring the high accuracy achieved by the P2 Finite Element formulation coupled with appropriate numerical integration and optimization techniques.

7 Conclusion

In this work, we presented a numerical solution to the classic Brachistochrone problem utilizing the Finite Element Method with quadratic (P2) shape functions. By discretizing the time functional and employing Gaussian quadrature — specifically, using Gauss-Jacobi quadrature to handle the integrable singularity at the starting point ($y(0) = 0$) and Gauss-Legendre quadrature elsewhere — we transformed the variational problem into a finite-dimensional nonlinear optimization task. This task was effectively solved using the L-BFGS-B algorithm, leveraging the analytically derived gradient of the discretized functional.

Our results, obtained for the specific endpoints $(0, 0)$ to $(2, 1)$, demonstrate excellent agreement with the known analytical cycloid solution, achieving a minimum descent time within 0.15% of the theoretical value. This validates the robustness and accuracy of the P2 FEM approach combined with specialized quadrature for this canonical variational problem.

Future work could involve investigating the convergence rate of the P2 FEM approach with mesh refinement ($N \to \infty$), applying the method to Brachistochrone problems with added constraints like friction (building upon work like [8]), or exploring the use of adaptive mesh refinement strategies concentrated near the start point where the solution changes rapidly. The developed framework also serves as a foundation for tackling other variational problems where analytical solutions are unknown.

References

[1] Johann Bernoulli. Problema novum ad cujus solutionem mathematici invitantur. *Acta Eruditorum*, 18:269, 1696.

[2] Dietrich Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge University Press, 3rd edition, 2007.

[3] Philippe G. Ciarlet and Cristinel Mardare. On the brachistochrone problem. *Communications in Mathematical Analysis and Applications*, 1(1):213–240, 2022.

[4] Rodney Coleman. A detailed analysis of the brachistochrone problem. HAL preprint hal-00446767v2, 2012. Accessed March 30, 2025.

[5] Galileo Galilei. *Discorsi e dimostrazioni matematiche intorno a due nuove scienze [Discourses and Mathematical Demonstrations Concerning Two New Sciences]*. Lodovico Elzevirii, Leiden, 1638.

[6] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987.

[7] Muhammad F. Kasim and Peter A. Norreys. Infinite dimensional optimistic optimisation with applications on physical systems. *arXiv preprint arXiv:1611.05845*, 2016.

[8] Alexander Kurilin. Sliding with friction and the brachistochrone problem. *arXiv preprint arXiv:2501.18622*, 2025.

[9] R. Padyala. Brachistochrone – the path of quickest descent. *Resonance*, 24(2):201–216, 2019.

[10] Giuseppe Pelosi and Stefano Selleri. A prelude to finite elements: The fruitful problem of the brachistochrone. *URSI Radio Science Bulletin*, (367):10–14, December 2018.

[11] Celia T. L. M. Pereira, Pedro A. F. Cruz, and Delfim F. M. Torres. A discrete algorithm to the calculus of variations. *arXiv preprint arXiv:1003.0934*, 2010.

A Optimization Algorithm

Algorithm 1 P2 Finite Element Algorithm for Brachistochrone

Require: Endpoint coordinates (x_f, y_f); Number of elements N; Number of quadrature points P; Small positive constant ϵ; Optimization convergence tolerance 'gtol'.

Ensure: Optimal interior nodal values $\mathbf{y}_{\text{int}}^*$ minimizing T_h; Approximated minimum time T.

Set element size $h \leftarrow x_f/N$.
Set global node coords $x_m \leftarrow m\frac{h}{2}$ for $m = 0, \ldots, 2N$.
Set boundary values: $y_0 \leftarrow 0$, $y_{2N} \leftarrow y_f$.
Set lower bounds $lb_m \leftarrow \epsilon$ for $m = 1, \ldots, 2N-1$.
Pre-compute Gauss-Legendre points (ξ_p^L, w_p^L) for $p = 1..P$.
Pre-compute Gauss-Jacobi points (ξ_p^J, w_p^J) (for $\alpha = 0, \beta = -1/2$) for $p = 1..P$.
Generate initial guess $\mathbf{y}_{\text{int}}^{(0)}$ (e.g., based on a circular arc).
procedure ObjectiveAndGradient($\mathbf{y}_{\text{int}}$) ▷ Computes T_h and ∇T_h
    Initialize total objective $T_h \leftarrow 0$.
    Initialize global gradient $\mathbf{G} \leftarrow \mathbf{0} \in \mathbb{R}^{2N-1}$.
    Construct full nodal vector $\mathbf{y} \leftarrow [y_0, y_1, \ldots, y_{2N-1}, y_{2N}]^T$, using $\mathbf{y}_{\text{int}}$ for $y_1..y_{2N-1}$.
    for $e = 1$ to N **do**
        Get global node indices for element e: $i \leftarrow 2(e-1)$, $j \leftarrow 2e-1$, $k \leftarrow 2e$.
        Get element nodal values $\mathbf{y}_e \leftarrow [y_i, y_j, y_k]^T$.
        if $e = 1$ **then** ▷ Use Gauss-Jacobi for first element
            Calculate T_1^* using Eq. (14).
            Calculate element gradient $\mathbf{g}_e \leftarrow \nabla_{\mathbf{y}_1} T_1^*$ using Eq. (20).
        else ▷ Use Gauss-Legendre for other elements
            Calculate T_e^* using Eq. (13).
            Calculate element gradient $\mathbf{g}_e \leftarrow \nabla_{\mathbf{y}_e} T_e^*$ using Eq. (19).
        end if
        $T_h \leftarrow T_h + T_e^*$. ▷ Accumulate total objective
        ▷ Assemble global gradient (add $\mathbf{g}_e = [g_{e,i}, g_{e,j}, g_{e,k}]^T$ to $\mathbf{G}$)
        if $i > 0$ **then** $G_i \leftarrow G_i + g_{e,i}$.
        end if ▷ Skip $i = 0$
        $G_j \leftarrow G_j + g_{e,j}$. ▷ $j = 2e-1$ is always interior
        if $k < 2N$ **then** $G_k \leftarrow G_k + g_{e,k}$.
        end if ▷ Skip $k = 2N$
    end for
    return $T_h, \mathbf{G}$
end procedure
Initialize L-BFGS-B solver with initial guess $\mathbf{y}_{\text{int}}^{(0)}$, bounds **lb**, and tolerance 'gtol'.
Run L-BFGS-B optimization, providing the ObjectiveAndGradient procedure for function and gradient evaluations.
Obtain optimal solution $\mathbf{y}_{\text{int}}^*$ from the converged L-BFGS-B result.
Calculate final physical time $T \leftarrow T_h(\mathbf{y}_{\text{int}}^*)/\sqrt{2g}$.
return $\mathbf{y}_{\text{int}}^*, T$.

B Code used for Numerical Solution

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize, brentq
from scipy.special import roots_jacobi

# --- Configuration ---
QUAD = 10  # Quadrature points
EPS = 1e-15 #epsilon
xi_leg, w_leg = np.polynomial.legendre.leggauss(QUAD)
xi_jac, w_jac = roots_jacobi(QUAD, 0, -0.5)  # For singularity at x=0

def shape(xi):
    """P2 shape functions and derivatives"""
    N = np.array([0.5*xi*(xi-1), 1-xi**2, 0.5*xi*(xi+1)])
    dN = np.array([xi-0.5, -2*xi, xi+0.5])
    return N, dN

def element(y, h, first=False):
    """Element T* and gradient calculation"""
    T, g = 0, np.zeros(3)
    xi, w = (xi_jac, w_jac) if first else (xi_leg, w_leg)

    for i in range(len(xi)):
        N, dN = shape(xi[i])
        Y = max(y @ N, EPS)
        Yp = y @ dN * (2/h)
        F = 1 + Yp**2
        dF = 2 * Yp * (2/h) * dN

        if first:
            # First element (singularity)
            G = max(y[1]*(1-xi[i]) + y[2]*0.5*xi[i], EPS)
            T += w[i] * np.sqrt((F+EPS)/G) * (h/2)
            dG = np.array([0, 1-xi[i], 0.5*xi[i]])
            fac = (h/4) * np.sqrt(G/(F+EPS)+EPS) / (G**2+EPS)
            g += w[i] * fac * ((dF*G) - (F*dG))
        else:
            # Regular element
            T += w[i] * np.sqrt((F+EPS)/Y) * (h/2)
            fac = (h/4) * np.sqrt(Y/(F+EPS)+EPS) / (Y**2+EPS)
            g += w[i] * fac * ((dF*Y) - (F*N))

    return T, g

def global_calc(y_unk, h, yf, N):
    """Global T* and gradient assembly"""
    y = np.zeros(2*N+1)
    y[1:-1], y[-1] = y_unk, yf

    T, g = 0, np.zeros(len(y_unk))

    for i in range(1, N+1):
```

```
        idx = [2*(i-1), 2*i-1, 2*i]
        Ti, gi = element(y[idx], h, first=(i==1))
        T += Ti

        if idx[0] > 0: g[idx[0]-1] += gi[0]
        g[idx[1]-1] += gi[1]
        if idx[2] < 2*N: g[idx[2]-1] += gi[2]

    return T, g

class BrachSolver:
    def __init__(self, xf, yf, N):
        self.xf, self.yf, self.N, self.h = xf, yf, N, xf/N

    def obj(self, y): return global_calc(y, self.h, self.yf, self.N)[0]
    def grad(self, y): return global_calc(y, self.h, self.yf, self.N)[1]

    def solve(self):
        # Initial circular arc guess
        x = np.linspace(0, self.xf, 2*self.N+1)
        hc = (self.xf**2 + self.yf**2)/(2*self.xf)
        y0 = np.sqrt(np.maximum(hc**2 - (x[1:-1]-hc)**2, 0))

        # Run optimizer
        res = minimize(
            self.obj, y0, jac=self.grad, method='L-BFGS-B',
            bounds=[(EPS, None)]*len(y0),
            options={'disp': True, 'gtol': 1e-7}
        )

        if res.success:
            y_sol = np.zeros(2*self.N+1)
            y_sol[1:-1], y_sol[-1] = res.x, self.yf
            return {'y': y_sol, 'T': res.fun/np.sqrt(2*9.81), 'success': True}
        return {'success': False}

def draw_quadratic_elements(x_nodes, y_nodes, num_points=10):
    """Draw smooth quadratic elements"""
    x_smooth = []
    y_smooth = []

    for i in range(0, len(x_nodes)-2, 2):
        # Extract the 3 nodes of this element
        x_elem = x_nodes[i:i+3]
        y_elem = y_nodes[i:i+3]

        # Map to reference element [-1, 1]
        h_elem = x_elem[2] - x_elem[0]
        x_mid = (x_elem[0] + x_elem[2])/2

        # Generate points within the element
        xi_local = np.linspace(-1, 1, num_points)
        x_local = []
        y_local = []
```

```
        for xi in xi_local:
            # Shape functions at xi
            N1 = 0.5*xi*(xi-1)
            N2 = 1-xi**2
            N3 = 0.5*xi*(xi+1)

            # Compute physical coordinates using shape functions
            x = x_elem[0]*N1 + x_elem[1]*N2 + x_elem[2]*N3
            y = y_elem[0]*N1 + y_elem[1]*N2 + y_elem[2]*N3

            x_local.append(x)
            y_local.append(y)

        x_smooth.extend(x_local)
        y_smooth.extend(y_local)

    return np.array(x_smooth), np.array(y_smooth)

def cycloid(xf, yf):
    """Find analytical cycloid parameters"""
    f = lambda t: (t-np.sin(t))/(1-np.cos(t)+EPS) - xf/yf
    theta = brentq(f, 1e-9, 2*np.pi-1e-9)
    a = yf/(1-np.cos(theta))
    return a, theta

# Main execution
if __name__ == "__main__":
    xf, yf = 2.0, 1.0
    N = 20
    g = 9.81

    # Analytical solution
    a, theta = cycloid(xf, yf)
    T_exact = theta * np.sqrt(a/g)
    print(f"Exact T = {T_exact:.6f}")
    # FEA solution
    solver = BrachSolver(xf, yf, N)
    result = solver.solve()

    if result['success']:
        print(f"FEA T = {result['T']:.6f}")

        # Plot
        plt.figure(figsize=(9, 6))
        x = np.linspace(0, xf, 2*N+1)

        # Plot nodes
        plt.plot(x, result['y'], 'bo', ms=3, label='FEA Nodes')

        # Plot quadratic elements
        x_smooth, y_smooth = draw_quadratic_elements(x, result['y'], num_points=20)
        plt.plot(x_smooth, y_smooth, 'b-', lw=1.5, label='FEA P2 Elements')
```

```
# Plot analytical solution
t = np.linspace(0, theta, 200)
plt.plot(a*(t-np.sin(t)), a*(1-np.cos(t)), 'g-', lw=2, label='Analytical Cycloid')

plt.xlabel('x', fontsize=14)
plt.ylabel('y', fontsize=14)
plt.title('Brachistochrone: P2 FEA Solution', fontsize=16)
plt.gca().invert_yaxis()
plt.grid(True)
plt.axis('equal')
plt.legend(fontsize=12)
plt.show()
```