The Brachistochrone Problem: The Path of Shortest Travel Time in a Uniform Gravitational Field

Thomas Nguyen[1]

[1]*Departments of Astronomy & Physics, University of Arizona, Tucson, AZ, USA*

ABSTRACT

The Brachistochrone problem is the path of shortest travel time as we travel between two points where one is below and horizontally spaced from another under the effect of a uniform gravitational field. Our goal in this paper is to generate such path using numerical computation. The shape of this path is governed by a simple 2nd-order ordinary differential equation, which was solved by reducing it to a system of 1st-order ODEs and then implementing the 4th-order Runge-Kutta method, along with the shooting method to figure out the initial slope of the path. The accuracy of our solution was verified via convergence and self-convergence. Our results point out the equation is not solvable when the start and end points are at same height. Otherwise, the solution curves appear to be cycloids, instead of parabolas, as predicted in the past. They were verified to be accurate with the specified methods from above.

Keywords: Brachistochrone — Convergence — Self-Convergence — Richardson extrapolation

1. INTRODUCTION

Suppose that we have two points A and B in space, where B is below A and they are horizontally spaced, and a smooth, frictionless wire with a bead that slides on the wire. In the Brachistochrone problem, our goal is to curve the wire in such way the bead slides under the influence of a uniform gravitational field (i.e. Earth's gravity) from A down to B in the shortest time possible. This solution curve is known as the "Brachistochrone", which originates from ancient Greek $\beta\rho\alpha\chi\iota\sigma\tau o\varsigma\ \chi\rho o\nu o\varsigma$ or "shortest time."

To simplify the problem, I shall assume that the bead is traveling in the 2-D xz-plane, as shown in **Figure 1**. Assume that the bead starts from rest, and Using conservation of energy, one can find the velocity at any point as follows,

$$\frac{1}{2}mv^2 = mgz \rightarrow v = \sqrt{2gz} \tag{1}$$

where z is the height descended and g = 9.81 m/s^2 is the approximate gravitational acceleration.

The time dt to travel the infinitesimal distance ds in **Figure 1** is given by

$$dt = \frac{ds}{v} = \frac{\sqrt{dx^2 + dz^2}}{\sqrt{2gz}} \tag{2}$$

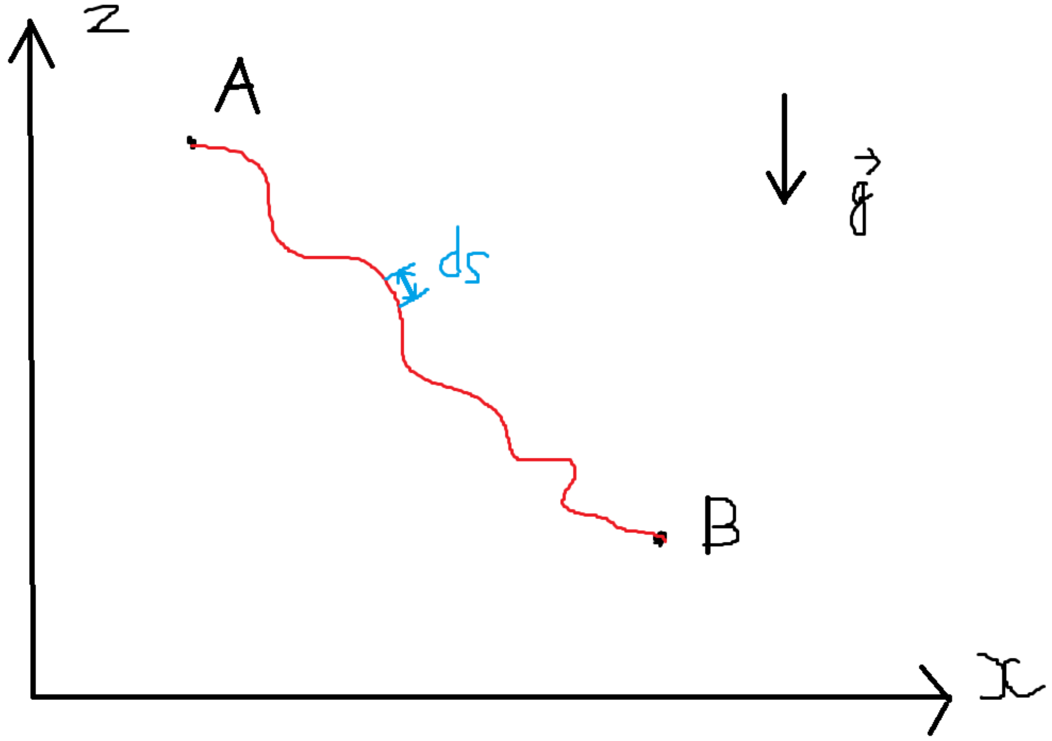

Figure 1. An illustrative sketch of the Brachistochrone problem, where we have two points A & B in space and B is below & horizontally spaced from A in Earth's gravitational field $\vec{g}$. The two points are connected by some random curve (red), and ds represents an infinitesimal segment of it.

where the second "=" comes from the fact that as the beads travel the distance ds, it will move some infinitesimal distances dx and dz in the x- and z-directions, respectively. Thus, ds is simply the magnitude of a position vector in Cartesian coordinates. Therefore, to find the total time to travel the entire curve in **Figure 1**, we just need to integrate dt from Equation (2):

$$T = \int_A^B \frac{ds}{v} = \int_A^B \frac{\sqrt{dx^2 + dz^2}}{\sqrt{2gz}} = \frac{1}{\sqrt{2g}} \int_{x_A}^{x_B} \frac{\sqrt{1+z'^2}}{\sqrt{z}} dx \tag{3}$$

where I denote z' = $\frac{dz}{dx}$ and the term in the integral as the Lagrangian of the system

$$L = \frac{\sqrt{1+z'^2}}{\sqrt{z}} \tag{4}$$

Since our goal is to find the curve of shortest descending time, we need to minimize the integral from Equation (3). The shape of this curve $z(x)$ is governed by the Euler-Lagrange equation

$$\frac{\partial L}{\partial z} - \frac{d}{dt}\left(\frac{\partial L}{\partial z'}\right) = 0 \tag{5}$$

Plugging the Lagrangian into Equation (5) yields

$$-\frac{1}{2z^{3/2}}\sqrt{1+z'^2}-\frac{d}{dx}\left(\frac{z^{-1/2}z'}{(1+z'^2)^{1/2}}\right)=\frac{-2zz''-z'^2-1}{2z^{3/2}(z'^2+1)^{3/2}}=0 \tag{6}$$

$$\rightarrow -2zz''-z'^2-1=0 \tag{7}$$

$$\rightarrow \frac{d^2z}{dx^2}=-\frac{z'^2+1}{2z} \tag{8}$$

Consequently, the Brachistochrone problem essentially reduces to a boundary value problem between x = 0 and x = b with known z(0) and z(b). We can also rescale the x-coordinate such that b = 1, and our goal is now solving this second-order ordinary differential equation (ODE) using numerical method, as discussed in Section **??** below. In addition, since the Lagrangian does not depend in x as seen in Equation (4), the Euler-Lagrange equation reduces to Beltrami's identity

$$L-z'\frac{\partial L}{\partial z'}=C \tag{9}$$

Plugging the Lagrangian into Equation (9) yields the constant C (defined as the integral of the motion) as follows:

$$C=\frac{1}{\sqrt{(1+z'^2)z}} \tag{10}$$

2. NUMERICAL METHODOLOGY

Before implementing numerical methods to solve for the Brachistochrone problem, Equation (8) must be reduced into a system of first-order ODEs. Let $u_0(x)=z(x)$ and $u_1(x)=z'(x)$. Then we have the following system of ODEs

$$\frac{du_0}{dx}=z'(x) \tag{11}$$

$$\frac{du_1}{dx}=z''(x)=-\frac{u_1^2+1}{2u_0} \tag{12}$$

Since there are two ODEs, we shall need two initial conditions in order to solve the system. The first is the starting position $z(0)=a$, which should be given. The second will be the initial slope of the curve $\frac{dz}{dx}|_{x=0}=b$. Since this is not given, we will use the shooting method with trials and errors to find an approximately accurate value of this initial slope. Essentially, I assume this initial slope to be some arbitrary value, and from the solution curve, I will check if the final point z(1) is consistent with the initial setup (i.e., the relative error is less than 10^{-3}). If not, I shall go back and change the initial slope, and this procedure is repeated until achieving the desired curve.

With that, I shall use the fourth order Runge-Kutta (hereafter denoted RK4), which has an accuracy of $O(h^4)$, to compute the approximate solution to the curve. This was done by estimating the value of the next point based on the value of the current point and the weighted average of four increments as follows,

$$z_{current}=z_{current}+\frac{h}{6}(k_1+2k_2+2k_3+k_4) \tag{13}$$

where I denote $y_{current}$ as the current value of the solution, h is the step size, and k_1, k_2, k_3, and k_4 are estimated slopes of the function on the right-hand side of an ODE at the beginning and midpoint of the interval.

There are multiple ways to verify the accuracy of the computed solution that I implemented. First, for a set of many step sizes and a set of initial conditions (starting & ending points and the initial slope), I checked if the relative difference in the value integral of the motion (defined by Equation 10) with respect to the initial value converges to 0 as the number of steps increases. The relative difference was calculated as follows

$$\delta C = \frac{|C(x_{final}) - C(x_{initial})|}{C(x_{initial})} \tag{14}$$

The second method to verify accuracy is to check if the solution curve self-converges at x = 1. Let us pretend that the BVP from equation (1) has an unknown exact solution $y_{exact}(t)$ and a numerical approximation to this analytic solution, $y_{num}(t)$, at some resolution h. And this numerical solution converges at order n as follows:

$$y_{num}(h) = y_{exact} + Ah^n \tag{15}$$

where A is some constant. With that, we can consider the case where we have many solutions at different solutions h_1, h_2, h_3 (h_1 ¿ h_2 ¿ h_3). Then, the absolute errors of the numerical solutions are given as follows:

$$y_{num}(h_1) = y_{exact} + Ah_1^n \tag{16}$$

$$y_{num}(h_2) = y_{exact} + Ah_2^n \tag{17}$$

$$y_{num}(h_2) = y_{exact} + Ah_2^n \tag{18}$$

Subtracting equation (18) from equation (17) and equation (18) from equation (16) yields

$$y_{num}(h_1) - y_{num}(h_3) = A(h_1^n - h_3^n) \tag{19}$$

$$y_{num}(h_2) - y_{num}(h_3) = A(h_2^n - h_3^n) \tag{20}$$

Then, if we divide equation (19) by equation (20), we will obtain the following:

$$\frac{y_{num}(h_1) - y_{num}(h_3)}{y_{num}(h_2) - y_{num}(h_3)} = \frac{(\frac{h_1}{h_3})^n - 1}{(\frac{h_2}{h_3})^n - 1} \tag{21}$$

If we assume that h_3 is the best solution that we have, $h_2 = 2h_3$, and $h_1 = h$ is some random lower resolution that we want to check, equation (21) becomes

$$\frac{y_{num}(h_1) - y_{num}(h_3)}{y_{num}(h_2) - y_{num}(h_3)} = \frac{(\frac{h}{h_3})^n - 1}{2^n - 1} \tag{22}$$

where I shall denote the LHS as the "numerical ratio" and the RHS as the "theoretical ratio" in my algorithm.

3. PROGRAM EXPLANATIONS

The programs involved in this project were written in Python and put together in a Jupyter notebook, included in the submitted folder. In the notebook, I first imported the packages *NumPy* to perform mathematical and vectorized operations, *Matplotlib* to create plots, and *SciPy* to perform Cubic Hermit spline interpolation, which will be useful to find the solution at z(0.9) based on the known values of the solution to the curve. In the following subsections, I shall break down what each code element in the notebook does

3.1. *Defining the System and Integral of Motion*

Code Cell 4: `system_of_odes` *function*—

```
def system_of_odes(x,u):
    z = u[0]
    dz_dx = u[1]
    d2z_dx2 = - ((dz_dx**2) + 1) / (2*z)
    return np.array([dz_dx, d2z_dx2])
```

This function defines the right-hand side (f) of the first-order system $\mathbf{u}' = f(x, \mathbf{u})$ for the Brachistochrone curve. The system is defined by $\mathbf{u} = [z, z']$, where $u[0] = z$ and $u[1] = z'$. The output array returns the derivatives: dz/dx (which is u_1) and $d^2z/dx^2 = -\frac{(z'^2)+1}{2z}$ (which is u_1').

Code Cell 5: `integral_of_motion` *function*—

```
def integral_of_motion(z, z_prime):
    if z <= 0:
        return np.nan
    return 1.0/(np.sqrt(z*(1+z_prime**2)))
```

This function computes the conserved integral of motion ($C = \frac{1}{\sqrt{z(1+z'^2)}}$) for the system. It is used to test the numerical stability and convergence of the RK4 method. It includes a check to handle non-positive vertical positions z.

3.2. *Part 1: RK4 Method*

Code Cell 6: RK4 Integrator Functions—This cell contains the core numerical integration functions:

1. `rk4_step(f, x, u, h)`: Performs a single step of the fourth-order Runge-Kutta (RK4) method. It calculates the four k values (k_1, k_2, k_3, k_4) and uses them to compute the next state vector u_{next} with an error proportional to $O(h^5)$.

2. `rk4_interpolation(f, a, b, N, alpha)`: Integrates the system of ODEs from $x = a$ to $x = b$ using N steps, given the initial conditions $\alpha = [z(a), z'(a)]$. It returns the array of x-points and the entire solution history $\mathbf{u}(x) = [z(x), z'(x)]$.

Code Cell 7: `shooting_method` *function*—The Shooting Method is a technique used to solve Boundary Value Problems (BVPs) by converting them into Initial Value Problems (IVPs).

1. It takes an initial guess for the starting slope $z'(0)$ (`dzdx_guess`).

2. It uses `rk4_interpolation` to integrate the ODE from $x = 0$ to $x = 1$ with the current guess.
3. It extracts the final vertical position, $z_{final} = z(1)$, and prints a comparison against the target boundary value z_{end}. Then it checks if the computed value is appreciably close to the expected value by computing the relative error

Code Cell 8: Case 1 Example—This cell executes the `shooting_method` with specific parameters ($z(0) = 1.0$, $z_{end} = 0.5$) and an initial guess for the slope, illustrating the process of solving a boundary value problem. This process was repeated for other cases

3.3. *Part 2: Convergence of Integral of Motion*

Code Cell 11: Convergence Test—This cell verifies the fourth-order convergence of the RK4 integrator by analyzing the conservation of the integral of motion C.

1. It loops over a range of step counts (N) to vary the step size $h = 1/N$.
2. For each N, it calculates the relative change in the conserved quantity after integrating from $x = 0$ to $x = 1$.
3. It generates a log-log plot of the error δC versus the step size h.
4. It plots a reference line of $O(h^4)$ to visually confirm that the numerical error scales with the fourth power of the step size, which is the expected convergence rate for the RK4 method.

3.4. *Part 3: Self-convergence of Solution for* $\mathbf{z(1)}$

This part computes the numerical and theoretical ratios of the solution for $z(1)$ with different resolutions and two reference resolutions and plot the ratios against the resolutions to check if the solution self-converges with the expected order

4. RESULTS & DISCUSSIONS

4.1. *Solution for z(x)*

I plotted the solution for three different scenarios: when the endpoints are at the same height, when the starting (left) point is at larger height, and when the ending (right) point is at lower height. In the following subsections, I shall discuss the resulting curves that I had. One thing in common is that the shapes of the curves in all scenarios were not parabolas, as suspected in the past, but more like a cycloid.

4.1.1. *Scenario 1: Endpoints at the same height*

In this scenario, even though the program can solve the boundary value problem, the so-called "solution" is nowhere near accurate, as seen in **Figure 2** below. The final points of the curve for both case are far off from their expected location. In fact, this curve was generated by ignoring the tolerance of 10^{-3} of the difference between the computed and expected values of z(1) just to show that the equation could not be solved for this scenario. The solution curve for the larger height case is merely the same as that in the lower height but simply shifted to the right. Another thing to note is that the generated curves varied significantly with the change of number of steps and initial slopes, which further strengthened my conclusion that the equation is not solvable in this scenario.

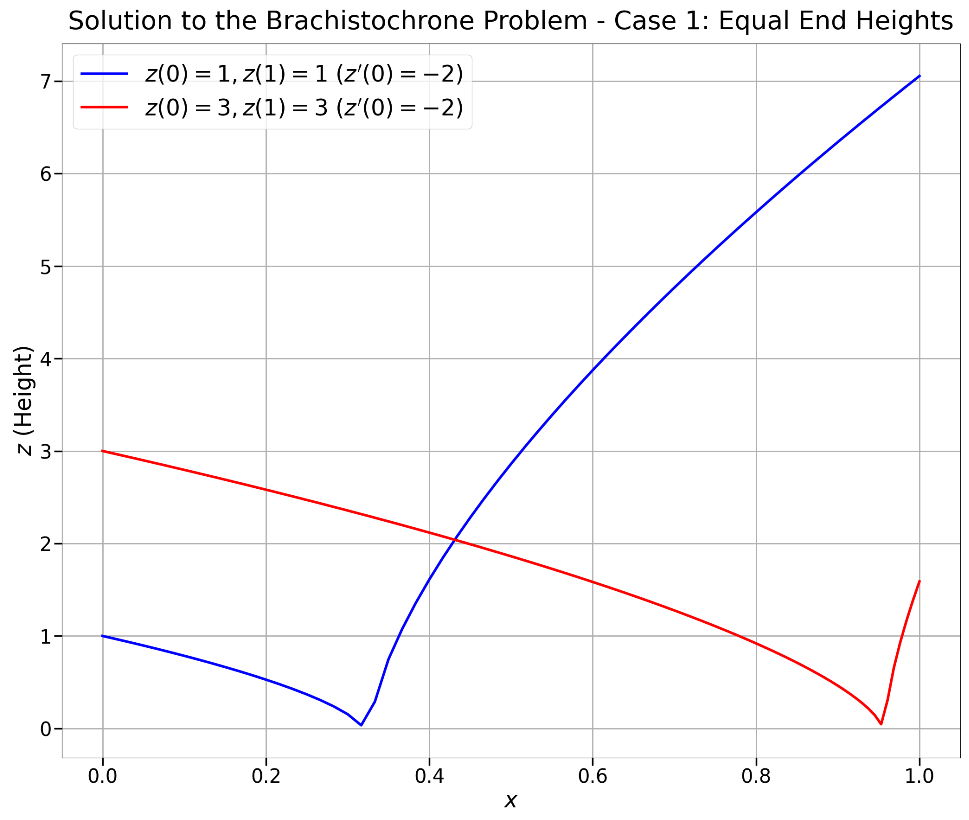

Figure 2. The "solution" curve to the Brachistochrone problem when the endpoints are at equal heights of 1 (blue) and 3 (red) with same initial slope of -2. The numbers of step for each case are 60 for endpoints of 1 and 128 for endpoints of 3.

4.1.2. *Scenario 2: Left points at larger heights*

In this scenario, the equation is now solvable as the final endpoint is much closer to its expected value (specified in the legend), as in **Figure 3**. Similar to the "solution" in Scenario 1, the shape at larger heights appear to be the same as that in lower heights, but simply shifted down (and also a bit more concave down), which might have been due to the different values of initial slopes.

4.1.3. *Scenario 3: Right point at larger height*

In this scenario, the equation is also solvable as the final endpoint is much closer to its expected value (specified in the legend), as in **Figure 4**. In this case, I re-plotted the solution curve for when the endpoints are reversed (the blue curve in **Figure 3** for comparison. It turns out that the solution is symmetric to the line x=0.5, which makes sense since the change in the boundary value problem is simply inverted, although it is unusual that the two curves have very different initial slopes.

4.2. *Convergence of δC at x=1*

For the convergence test with δC, I chose the initial conditions to be $z(0) = 5$, $z(1) = 1$, and $z'(0) = -2$, and the step sizes to be N = 100,...,700. The result is shown in **Figure 5**. From **Figure**

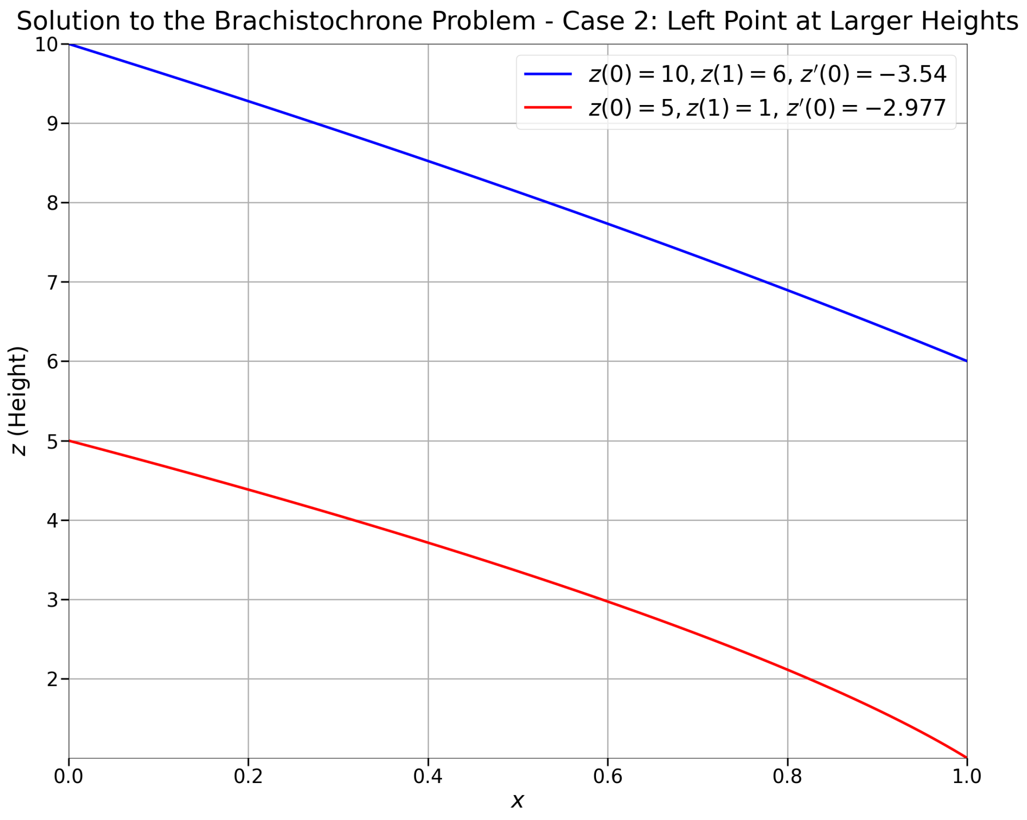

Figure 3. The curve to the Brachistochrone problem when the left endpoints are at larger heights of 10 (blue) and 6 (red) with the guessed initial slopes as specified. The number of step for both cases is 1000.

5, the relative change in δC does indeed converge to 0 with an accuracy of $O(h^4)$, as expected. Thus, the generated curves from **Figure 1**, **Figure 2**, **Figure 3** above are accurate.

4.3. *Self-convergence of z(1)*

For the sake of consistency in verifying the accuracy of my code, I chose the same initial conditions as for the convergence test: $z(0) = 5$, $z(1) = 1$, and $z'(0) = -2$, and the step sizes to be N = 100,...,700. The result is shown in **Figure 6**. According to that, the numerical ratio (dotted) is close to the theoretical ratio (dashed) and converges towards it with increasing resolution (i.e. decreasing h), which is the key characteristic of self-convergence, as described above. In addition, both ratios grow with the expected order of $O(h^4)$, indicated by the similarities between their corresponding lines and the black reference line. This once again implies that my program and the generated curves from **Figure 1**, **Figure 2**, **Figure 3** above are accurate.

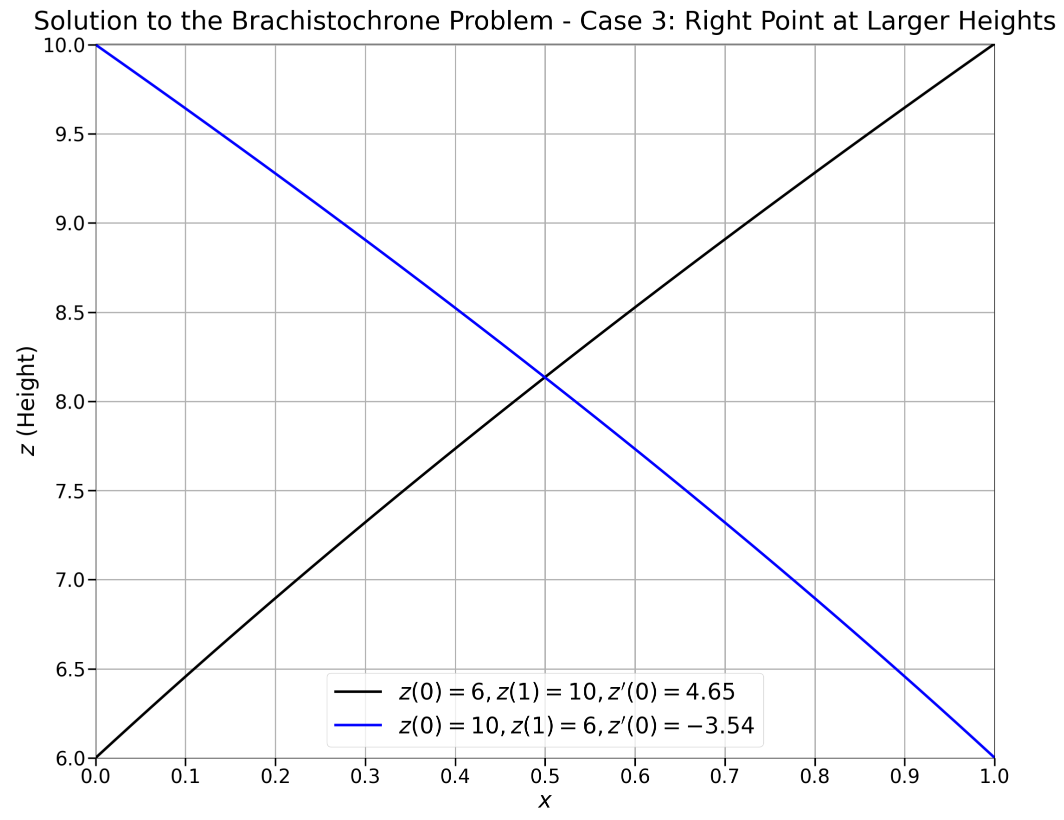

Figure 4. The curve to the Brachistochrone problem when the right endpoint are at larger heights of 10 (red) and its counterpart where the endpoints are reversed, with the specified initial slopes. The number of step for both cases is 1000.

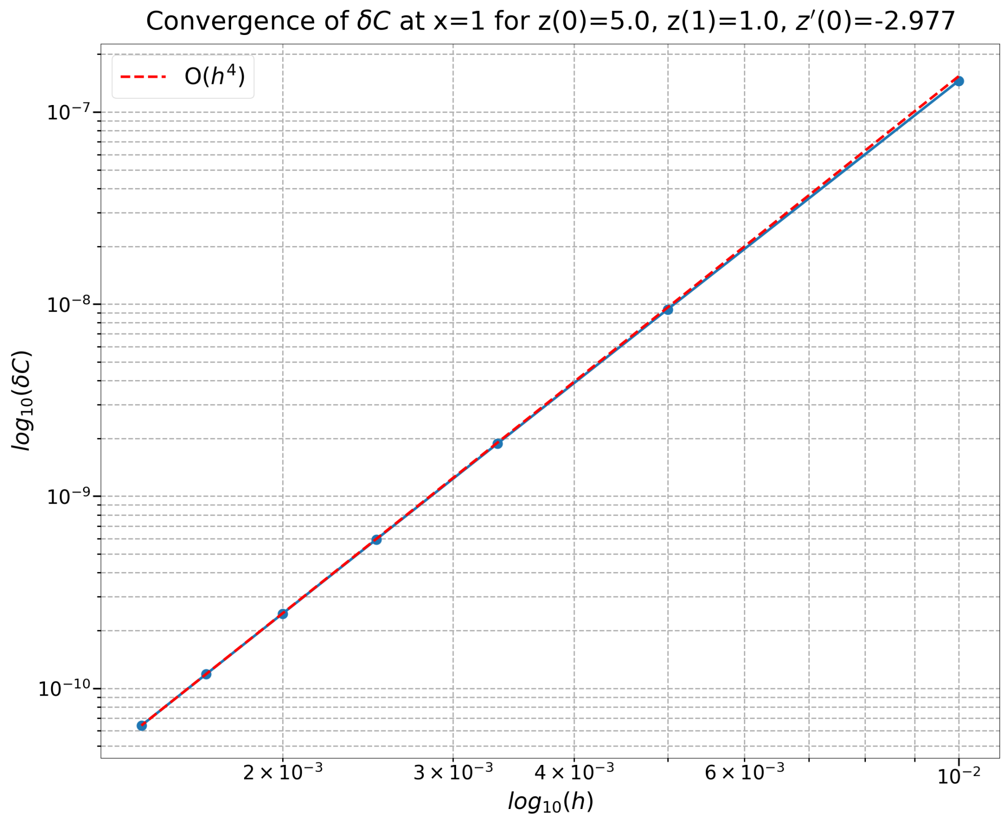

Figure 5. Relative change δC of the integral of motion from the solution curve with initial conditions $z(0) = 5$, $z(1) = 1$, and $z'(0) = -2$. The red dashed lines indicate the expected order of accuracy of $O(h^4)$. Both axes are in log scales, as indicated.

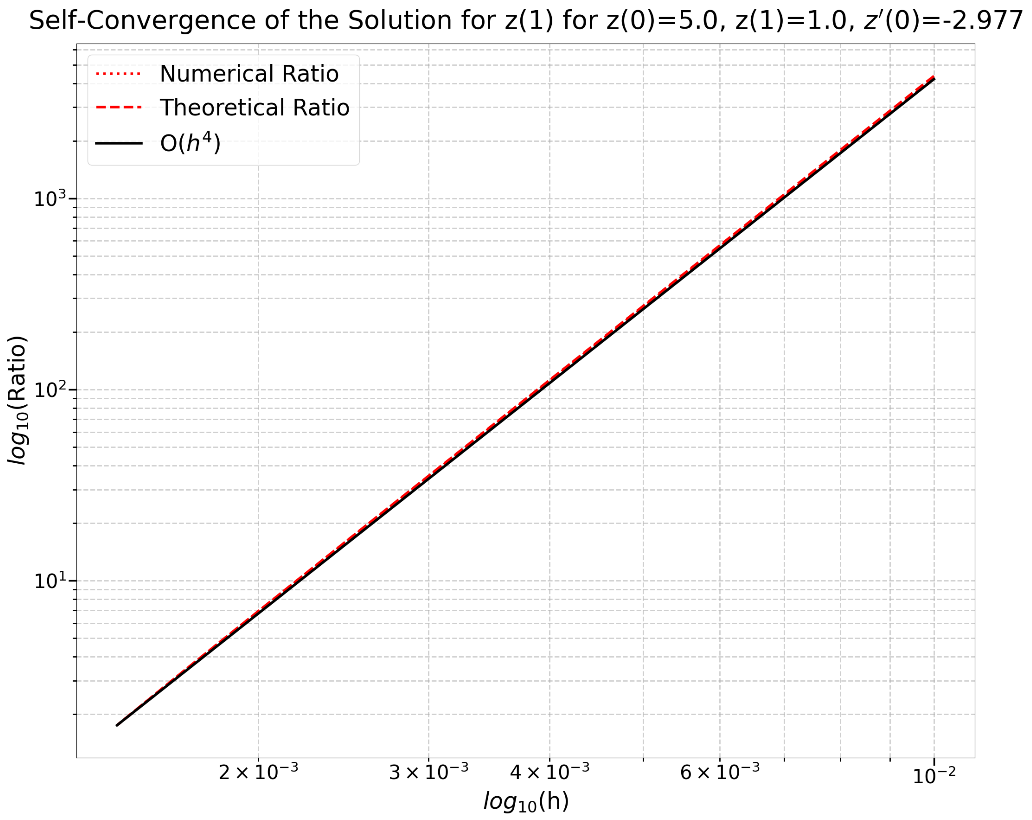

Figure 6. The value of the theoretical and numerical ratios with respect to the resolution h, showing that the computed solution curves from above is self-convergent to the accurate ones. The black line indicate the order of convergence. Both axes are in log scales, as indicated.