A Gradient Descent Approach to the Brachistochrone Problem

James Saslow
1/31/2023

***Abstract*—The Brachistochrone is defined to be the curve of least time between two points that a point mass travels on while subject to a uniform gravitational field. This paper will present an analytical derivation of the Brachistochrone curve as well as a numerical approach to arriving at the brachistochrone by a gradient descent algorithm.**

I. Introduction

A. History and Motivation

The Brachistochrone problem is a classical mechanics problem first proposed by Johann Bernoulli in the late 1600s. The Brachistochrone is defined to be the path of least time When guessing what function might fit this mysterious curve, it is often a common misconception to choose either a straight line, or a curve with a very steep drop off at the beginning. These may seem like plausible candidates for the Brachistochrone curve, however they are not. Although the straight line may be the path of least distance, it is not the path of least time. Likewise, a very steep drop off curve will be the path of greatest velocity following the object's release, but it is not the path of least time. The true path of the Brachistochrone curve has the perfect balance of optimizing distance and speed such that it guarantees the elapsed time of an object traveling on the curve to be a minimum.

II. Analytical Derivation

Thankfully, the Brachistochrone curve has an analytical solution that we may compare our numerical result to later. In this section, we will derive the equation of the Brachistochrone curve.

The velocity along some arclength of a curve may be written as

$$v = \frac{ds}{dt} \tag{1}$$

Rearranging we find

$$dt = \frac{ds}{v} \tag{2}$$

If we choose to represent our curve in cartesian coordinates, our arclength becomes $ds = \sqrt{dx^2 + dy^2}$. Substituting into (2), we find

$$dt = \frac{\sqrt{1 + (y')^2}}{v} dx \tag{3}$$

where $y'(x) = dy/dx$

Assuming our point mass was released from rest, we can calculate the velocity v at any altitude y by making use of the conservation of energy.

$$\frac{1}{2}mv^2 = mgy \tag{4}$$

Simplifying (4), we find that $v = \sqrt{2gy}$, which we can substitute into (3) to get

$$T = \frac{1}{\sqrt{2g}} \int \sqrt{\frac{1 + (y')^2}{y}} dx \tag{5}$$

where $T = \int dt$; the *total* elapsed time.

By definition, the Brachistochrone is the curve where T is minimized. Thus, we may extremize T by recognizing that $\sqrt{(1 + (y')^2)/y}$ is a Lagrangian of T. Thus we can solve the Euler Lagrange equations

$$\frac{d}{dx}\frac{\partial L}{\partial y'} = \frac{\partial L}{\partial y} \tag{6}$$

with

$$L = \sqrt{\frac{1 + (y')^2}{y}} \tag{7}$$

as the Lagrangian.

Solving (6) yields the differential equation

$$2y''(x)y(x) + [y'(x)]^2 + 1 = 0 \tag{8}$$

The solution to this differential equation are the equations of a *cycloid*.

$$x = R(\theta - sin(\theta)) \tag{9}$$

$$y = R(1 + cos(\theta)) \tag{10}$$

We may substitute equations (9) and (10) into the integral in (5) to find the elapsed time for a particle to travel from $\theta = 0$ to $\theta = \pi$ for a cycloid of radius R in a gravitational field g.

After the integration, we find that the elapsed time is

$$T = \pi\sqrt{R/g} \tag{11}$$

We may select the parametric equations of a cycloid of radius 1 and shifted down 2 units as a solution

$$x = \theta - sin(\theta) \tag{12}$$

$$y = 1 + cos(\theta) - 2 \tag{13}$$

We will be using this solution to compare to the numerical result in the next upcoming sections.

III. Setup

A. The Model

A convenient way to model a Brachistochrone curve is by connecting inclined planes in series. Basic kinematics can tell us the velocity of the particle after each inclined plane. Summing each time of travel of each inclined plane t_i over all planes, gives us the total elapsed time $T = \sum t_i$. There are N number of inclined planes each spaced a distance $\Delta x = \frac{\pi}{N}$. The brachistochrone in particular that we are interested in modeling is given by equations (9) and (10), where this curve intersects the points $(0, 0)$ and $(\pi, -2)$. In the model, there are $N - 1$ number of *joints* that connect two inclined planes (disregarding endpoints). We record the location of these joints by examining their y-value and store them in a vector $\mathbf{Y}$ (along with the y-value of the endpoints which will remained fixed)

$$\mathbf{Y} = (0, ..., y_i, ..., y_N, -2) \tag{14}$$

We can also define a vector $\mathbf{J}$ that just contains the joints

$$\mathbf{J} = (y_1, ..., y_N) \tag{15}$$

The ith inclined plane intersects points y_i and y_{i+1}, and has a travel time of t_i.

B. Kinematics

We must also derive an expression of time elapsed after a particle has traveled down each consecutive inclined plane.

Recognizing from basic kinematics that $\Delta v = at$, and that $v = \sqrt{2gy}$, we may derive an expression for the elapsed time of the ith inclined plane as

$$t_i = \sqrt{\frac{2}{g}} \frac{\sqrt{(y_{i+1} - y_i)^2 + \Delta x^2}}{\sqrt{y_{i+1}} + \sqrt{y_i}} \tag{16}$$

and summing over all inclined planes gives us the total elapsed time $T = \sum_{i=0}^{N} t_i$.

IV. Gradient Descent Algorithm

A. The Algorithm

Now that we have a numerical expression for T, we must optimize it to find the path of least time. The way we do this is through a *gradient descent* algorithm. The gradient of a scalar field always returns a vector field pointing in the direction of steepest *ascent*. Meaning, the negative gradient would always point in the direction of steepest *descent*.

With this algorithm, we could have an apriori guess $\mathbf{Y}^{old}$ that we can update using $\vec{\nabla} T^{old}$. And our aposteriori updated information would be some $\mathbf{Y}^{new}$.

We can outline such an algorithm as

$$\mathbf{Y}^{new} = \mathbf{Y}^{old} - (lr)\vec{\nabla} T^{old} \tag{17}$$

where lr is the *learning rate*, which is a constant and a scaling factor for the gradient.

B. Computing the Gradient

In (14), the expression that seems the most complicated to compute is $\vec{\nabla} T^{old}$, which I'll just write as $\vec{\nabla} T$ for now.

$\vec{\nabla} T$ is just a vector itself.

$$(\vec{\nabla} T)_i = (\frac{\partial T}{\partial y_i}) \tag{18}$$

In the code, we can shift each y-value individually by an infinitesimal element dy to compute each partial derivative in this vector.

V. Results

I chose random apriori values for $\mathbf{Y}$ within the window of $(0, -2)$. Since this is the window where the true values of $\mathbf{Y}$ live, it would make the gradient descent algorithm converge faster.

A low lr is prefered $\quad lr \leqslant 0.05$ for $N \leqslant 60$ such that the time evolution of T doesn't get stuck in a periodic, non converging well. Low learning rates dampen this oscillation and allow T to steadily converge to $\pi\sqrt{R/g}$. Since R and g are made from relative units of measure, allow us to set $R = g = 1$. This choice will make it clear that T has converged to π for a sufficient number of planes N and a sufficient number of gradient descent corrections (*itn*).

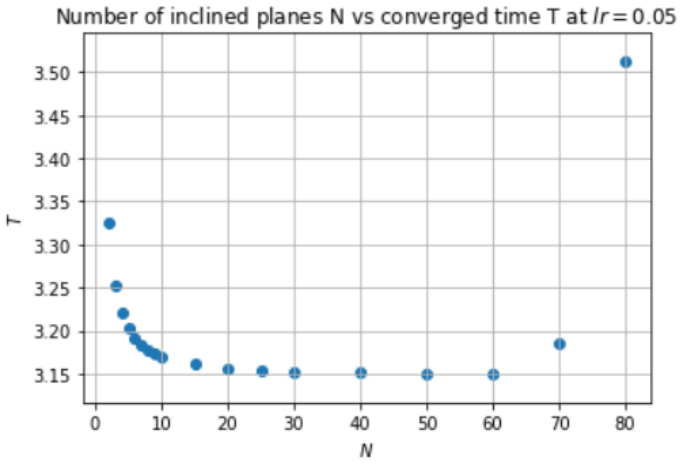

Figure: 1 Regime of steady convergence $N \leqslant 60$ for $lr = 0.05$

As described by Figure 1, attempting to approximate the Brachistochrone with more than 60 ramps at $lr = 0.05$ results in increasing error. The reason for this is because the learning rate isn't small enough to make such fine adjustments to the multi-level geometry of a 60+ dimensional function. The only way to reduce this error is to decrease the value of lr and increase the number of runs until convergence in T.

In this specific regime, we see T converging to 3.150 which results in a 0.268% error from $T = \pi$.

Our numerical result compared to the analytical closed form expression of the brachistochrone can be seen in the following figure

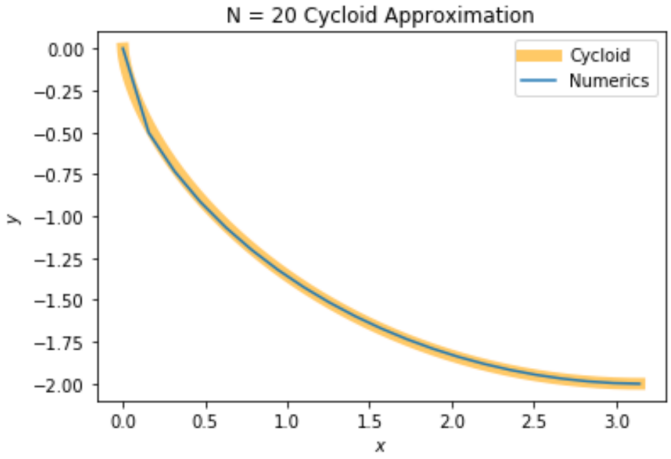

Figure: 2 Numerical Approximation of the Brachistochrone curve

VI. Conclusion

From this study, we can conclude that a gradient descent algorithm is a reliable method for approximating the Brachistochrone curve within the $N \leqslant 60$ and $lr = 0.05$ regime. The best optimized candidate at $N = 60$ yields $T = 3.150$ which is a 0.268% error from the analytical result.

VII. Code

The source code and Jupyter notebook file can be found on my wesbsite

https://jamessaslow.github.io/projects.html

VIII. References

References

[1] "Problem of the Week." DEPARTMENT OF PHYSICS, https://www.physics.harvard.edu/undergrad/problems.