Brachistochrone Numerical Optimization

Prikshet Sharma

February 10, 2019

In the Brachistochrone Problem (on frictionless surface), our goal is to find a path for a frictionless surface, between two given points P1 and P2, such that a sliding mass that starts at P1 takes the minimum amount of time to travel to P2. First, minimum travel time given points $P1 = (0, 1)$ and $P2 = (1, 0)$ is analytically calculated. Then approximation using gradient descent is shown, followed by comparison of results.

1 Analytical Evaluation

The time that an object takes to slide on a frictionless surface from point $P1$ to point $P2$ is given by

$$t_{12} = \int_{P1}^{P2} \sqrt{\frac{1 + y'^2}{2gy}} dx \tag{1}$$

The function

$$x = \frac{1}{2}(\theta - \sin\theta) \tag{2}$$

$$y = \frac{1}{2}(1 - \cos\theta) \tag{3}$$

gives the smallest value for t_{12}

The derivatives with respect to θ are given by

$$\frac{dx}{d\theta} = \frac{1}{2}(1 - \cos\theta) \tag{4}$$

$$\frac{dy}{d\theta} = -\frac{1}{2}\sin\theta \tag{5}$$

If we plug in (2), (3), (4) and (5) in (1) we get

$$\begin{aligned}
t_{min} &= \int_0^1 \sqrt{\frac{1 + \frac{dy}{dx}^2}{2g(1-y)}}dx \\
&= k\int_0^1 \sqrt{\frac{1+y^2}{1-y}}dx, (k = \sqrt{\frac{1}{2g}}) \\
&= k\int_0^{2.55} \sqrt{\frac{1 + \frac{\cos^2\frac{\theta}{2}}{\sin^2\frac{\theta}{2}}}{\frac{1}{2}(1-\cos\theta)}}d\theta \\
&= k\int_0^{2.55} \sqrt{\frac{\frac{1}{2}(1-\cos\theta)}{\sin^2\frac{\theta}{2}}}d\theta \\
&= k\int_0^{2.55} \sqrt{\frac{\sin^2\frac{\theta}{2}}{\sin^2\frac{\theta}{2}}}d\theta \\
&= k\int_0^{2.55} \sqrt{1}d\theta \\
&= 2.55k \\
&= 2.55\sqrt{\frac{1}{2g}} \approx 0.576
\end{aligned}$$

2 Numerical Optimization

To compute the solution using Numerical Optimization, the interval was discretized into hundred intervals $\in [0, 1]$, where the end points of the intervals are given by $(x_0, \ldots, x_{100})$. Note that $0 = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = 1$. Let $y_0, y_1, \ldots, y_{100}$ be the corresponding y values, where $y_0 = 1$ and $y_{100} = 0$. In the discrete version, we treat the function as a piece-wise linear function, and the integral of all segments is given by

$$t_{100}(x) = 2k\sum_{i=1}^{99}\sqrt{1+(\frac{x_{i+1}-x_i}{y_{i+1}-y_i})^2}(\sqrt{1-y_{i+1}}-\sqrt{1-y_i}) \tag{6}$$

2.1 Gradient Descent

Since $k > 0$, gradient descent was performed on

$$T_{100}(x) = 2\sum_{i=1}^{99}\sqrt{1+(\frac{x_{i+1}-x_i}{y_{i+1}-y_i})^2}(\sqrt{1-y_{i+1}}-\sqrt{1-y_i}) \tag{7}$$

After finding T_{min}, time to travel from $(0, 1)$ to $(1, 0)$ is $t_{min} = T_{min}k$

The i'th gradient of $T(x)$, with $x_1, \ldots, x_{100}$ as variables is given by

$$(\nabla T)_i = \frac{\partial T}{\partial x_i} = 2\frac{d_{i-1}}{q_{i-1}}\frac{x_i - x_{i-1}}{(y_i - yi - 1)^2} - 2\frac{d_i}{q_i}\frac{x_{i+1}-x_i}{(y_{i+1}-y_i)^2} \tag{8}$$

Gradient Descent Algorithm with learning rate $\alpha = 0.001$ is given by

Algorithm 1: Gradient Descent

```
α = 0.001
for iteration ← 0 to 100000 do
    yTemp ← [1]
    for i ← 1 to 100 do
        newY = y[i] − α ∗ gradientT(i)
        if newY <= 1 then
            yTemp.append(newY)
        else
            yTemp.append(1)
    y = yTemp
    y.append(0)
```

$gradientT(i)$ computes $(\nabla T)_i$, given by equation (8).

	$T(x)$	Time
Equally Spaced y_i	2.82	0.638
After Gradient Descent	2.58	0.583
Analytical Result	-	0.576

Table 1: Results

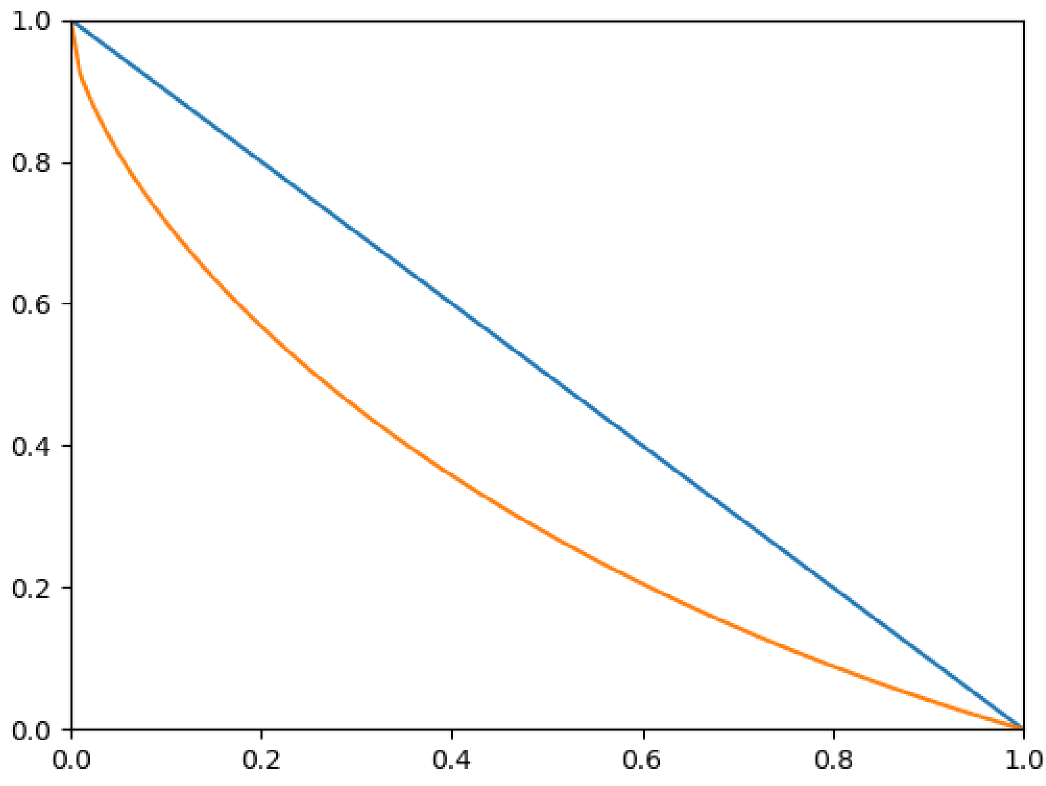

Figure 1: Equally Spaced y_i vs Numerically Optimized solution

3 Python Code

```
from math import sqrt, pi
import matplotlib.pyplot as plt

def q(i):
    return sqrt(1+((x[i+1] - x[i])/(y[i+1]-y[i]))**2)

def d(i):
    if(y[i]>1):
        print(i, y[i])
    return sqrt(1-y[i+1]) - sqrt(1-y[i])

def gradientT(i):
    return 2*(d(i-1)/q(i-1))*((x[i]-x[i-1])/(y[i]-y[i-1])**2)\
   -2*(d(i)/q(i))*((x[i+1]-x[i])/(y[i+1]-y[i])**2)

n = 100

def T(x):
    sum = 0
    for i in range(0, n):
        sum += d(i) * q(i)
    return 2 * sum


def printResults():
    print('T(x)', T(x))
    print('Time:', T(x)*k)

x = []
y = []

xEndPoint = 1
for i in range(0, 101):
    x.append(xEndPoint*i/100)
print(x)
```

```python
yEndPoint = 1
for i in reversed(range(0, 101)):
    y.append(yEndPoint*i/100)
print(y)

g = 9.8
k = sqrt(1/(2*g))

printResults()
print(gradientT(2), gradientT(3), gradientT(4))

plt.plot(x, y, '-o', markersize = 0.1)
plt.axis([0, 10, 0, 10])



a = 0.001
# gradient descent
for iter in range(0, 100000):
    yTemp = [1]
    for i in range(1, 100):
        newY = y[i] - a * gradientT(i)
        if newY <= 1:
            yTemp.append(newY)
        else:
            yTemp.append(1)
    y = yTemp
    y.append(0)
printResults()

plt.plot(x, y, '-o', markersize = 0.1)
plt.axis([0, 1, 0, 1])
plt.show()
```

References

[1] Weidong Shao. *Solve the Brachistochrone Problem with Numerical Optimization*. 2008.

[2] Weisstein, Eric W. *Brachistochrone Problem*. From MathWorld–A Wolfram Web Resource

[3] *Cycloid*. https://en.wikipedia.org/wiki/Cycloid.