CSC 200H Assignment 6

Sifan Ye

February 11, 2019

1 Prompt

Use base-level numerical optimization to solve the brachistochrone problem (look it up if you don't know what it is) for the case of a sliding mass, initially at rest, moving down one meter (e.g. in y) and across one meter (e.g. in x) (in earth-normal gravity). By base-level I mean, a program calling only basic library functions (e.g. sin, sqrt, read, write, print, etc.) No numerical packages, no canned differential equation solvers. The idea is to figure out how this sort of computation is done. Splitting x into 100 or so segments and doing piecewise linear integration of the motion along each segment to compute the sliding time will probably suffice for this problem, as the solution is a smooth curve. The trick is the search efficiently through the space of piecewise linear curves for the best. Graph your solution curve, and compare it to the theoretically optimal cycloid. Also compare your time to the theoretical minimum.

2 Graph Theory Approach

Since I am finding a shortest path from a start point to end point, I decided to split each side into n divisions, thus creating a graph with $(n+1) \times (n+1)$ vertices, and I can search for a shortest path from the top left corner to the button right corner using Dijkstra's Algorithm.

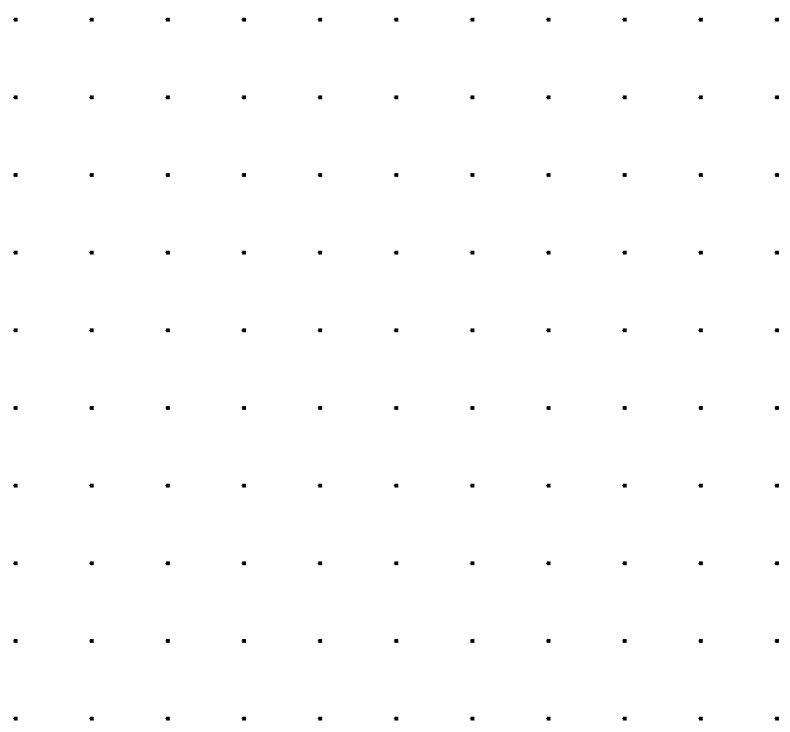

Figure 1: vertices when n = 10

3 The Cost Function

In order to search the graph using Dijkstra's Algorithm, I would need a cost function for an edge. This cost function would compute the time taken to slide from one vertex to its neighbor.
Since there is no friction in this scenario, no energy is dissipated, and thus I used conservation of mechanical energy [1] to derive a cost function:

$$\frac{1}{2}mv^2 = mgh$$

In this scenario, h will be the y coordinate of the vertex and assume $m = 1$:

$$\frac{1}{2}v^2 = gy$$

Rearranging the equation gives a function to obtain the velocity of the mass at any given point:

$$v = \sqrt{2gy}$$

Thus, I can compute the time taken for the mass to slide along any edge (v, w):

$$t = \frac{2d}{v_v + v_w}$$

with d being the straight line length of (v, w).

4 Neighbors

Before searching the graph, we need to define which edges are connected. At first, I only connected 1 layer of neighbors. This gives a slope range of -1 to 1, and the slope from search is far from the optimal curve.

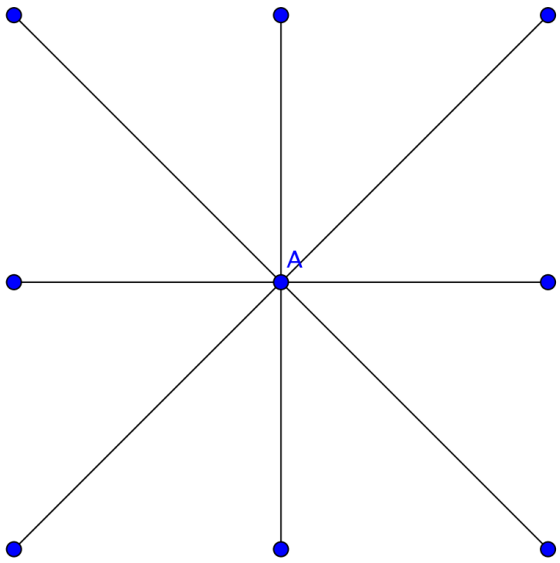

Figure 2: 1 layer of neighbors

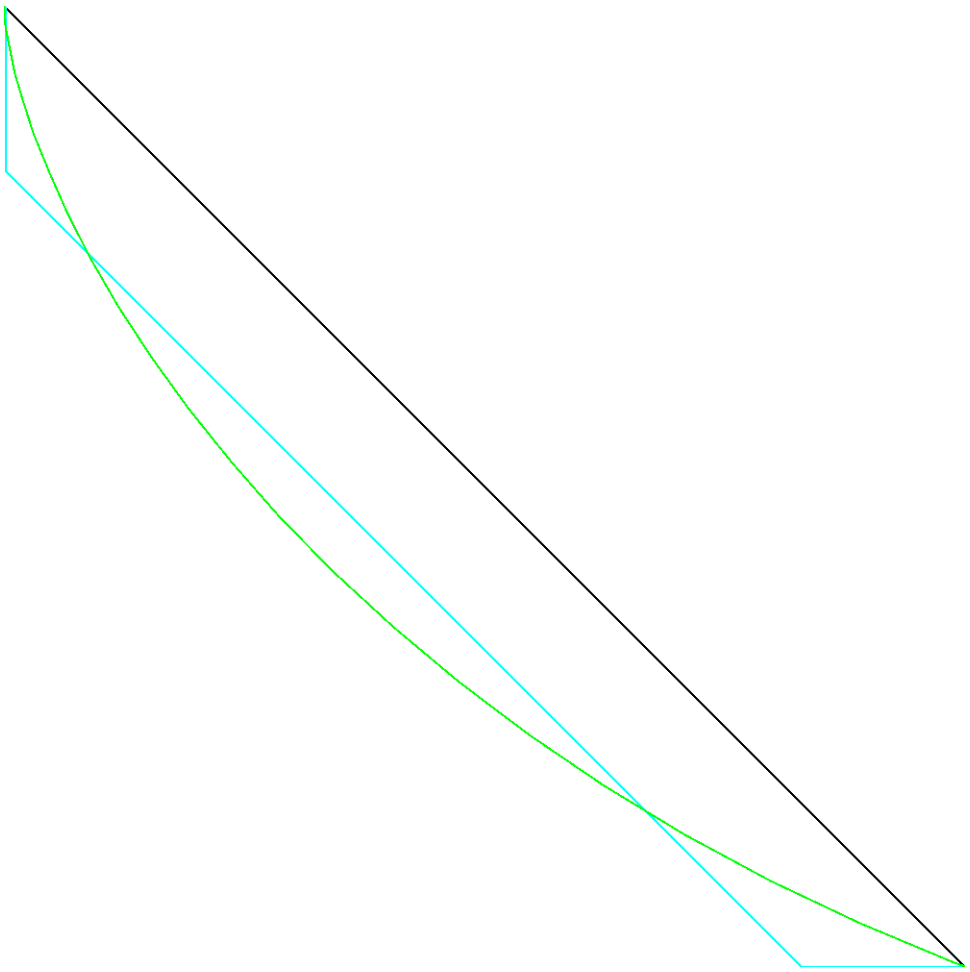

Figure 3: n = 500, s = 1. Green: optimal curve, Cyan: search result

At first, I varied n to see if I can get more accurate results, but the curves generated are similarly far from optimal: they all have 2 bends, with the slope changing from ∞ to -1 then to 0.

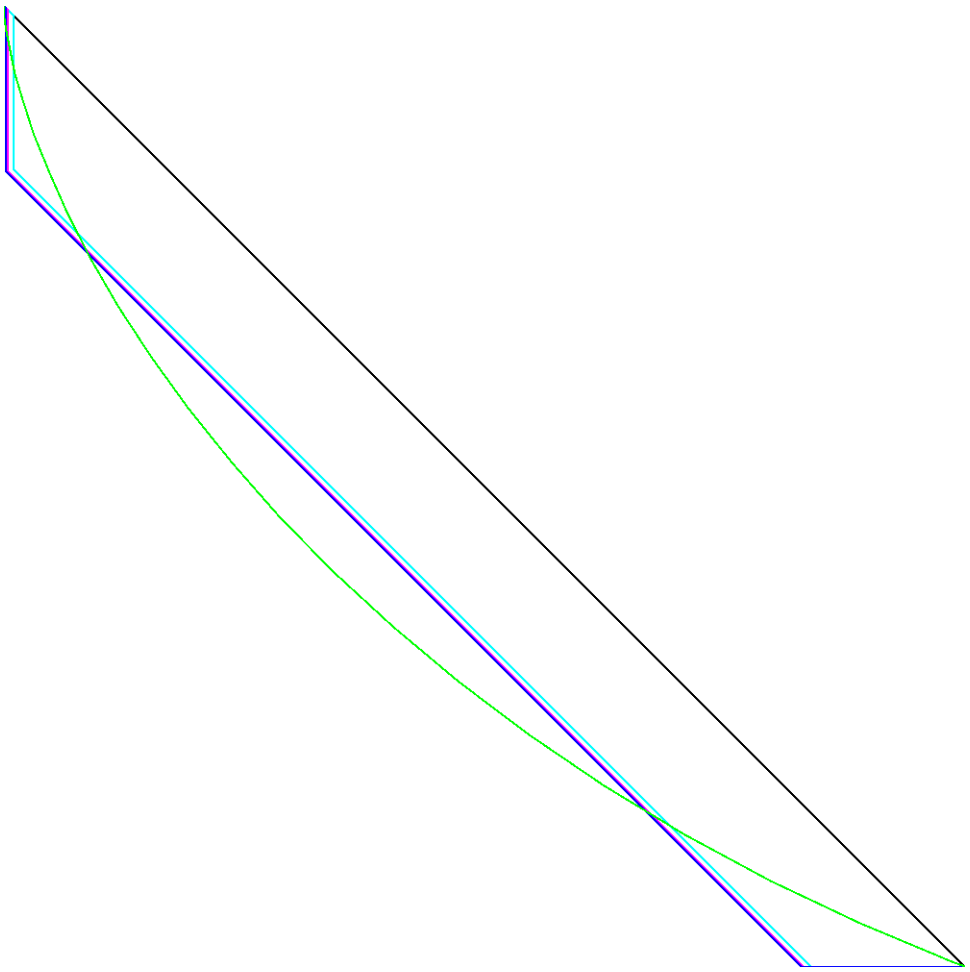

Figure 4: Cyan: n = 100, Magenta: n = 250, Blue: n = 500, Green: Optimal

Thus I added a variable s to change the number of layers of possible neighbors, and compared the results, showing that higher s gives a curve closer to the optimal cycloid as s layers of neighbors gives slopes in the range of

$$\frac{1}{s} \leqslant |m| \leqslant s$$

presenting more turns to approximate the optimal curve.

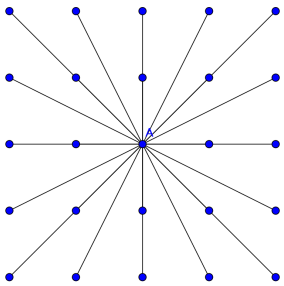

Figure 5: s = 2

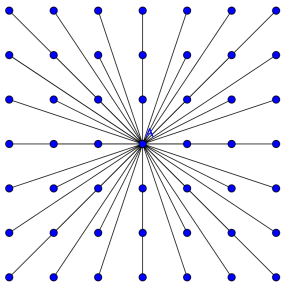

Figure 6: s = 3

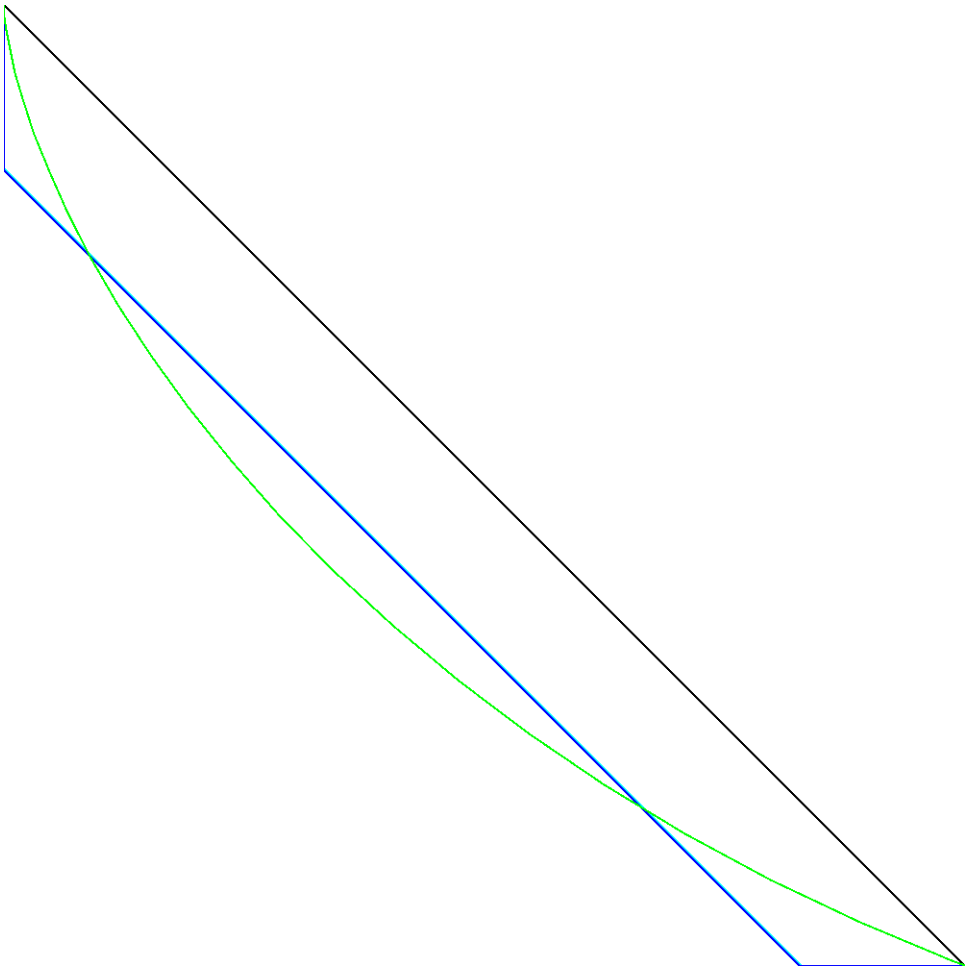

Figure 7: Result when s = 3. Green curve is the optimal cycloid

5 Results

For searching the optimal curve and descent time, I ran the graph with $n = 500$ and $s \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$.

Layers of Neighbors	Search Time (s)	Descent Time (s)
1	0.442	0.60773
2	0.906	0.58932
3	1.943	0.58554
4	3.85	0.58433
5	6.36	0.58365
6	9.252	0.58336
7	15.669	0.58318
8	20.946	0.58310
9	26.636	0.58304
10	42.041	0.58301
11	51.54	0.58299
12	62.158	0.58298

The most precise optimal result I found online [2] gives ≈ 0.58s for the optimal time of descent, while this search method, using $s = 12$, gives a descent time that is very close to the optimum, within 1 minute of searching.

The curve generated is also very close to the optimal cycloid. At $s = 5$ on canvas of dimension $(1000, 1000)$, the search result is already overlapping with the optimal cycloid.

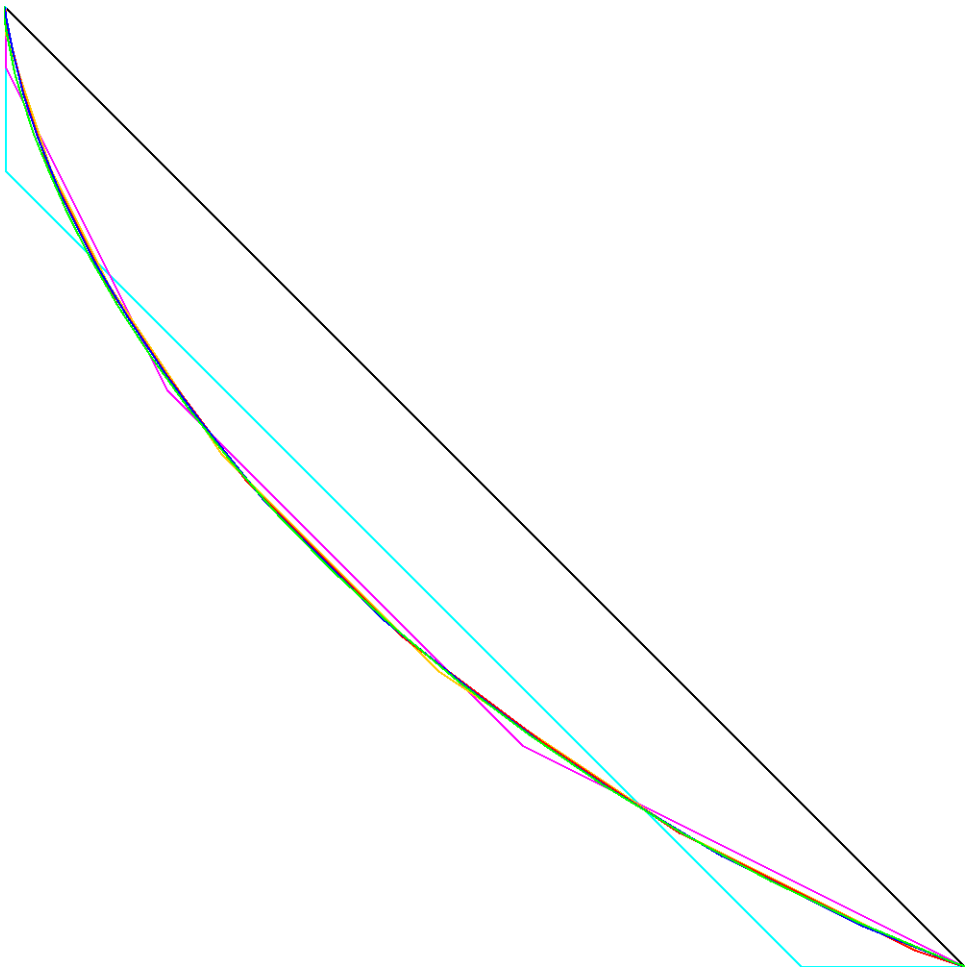

Figure 8: Result upto s = 5. Green curve is the optimal cycloid

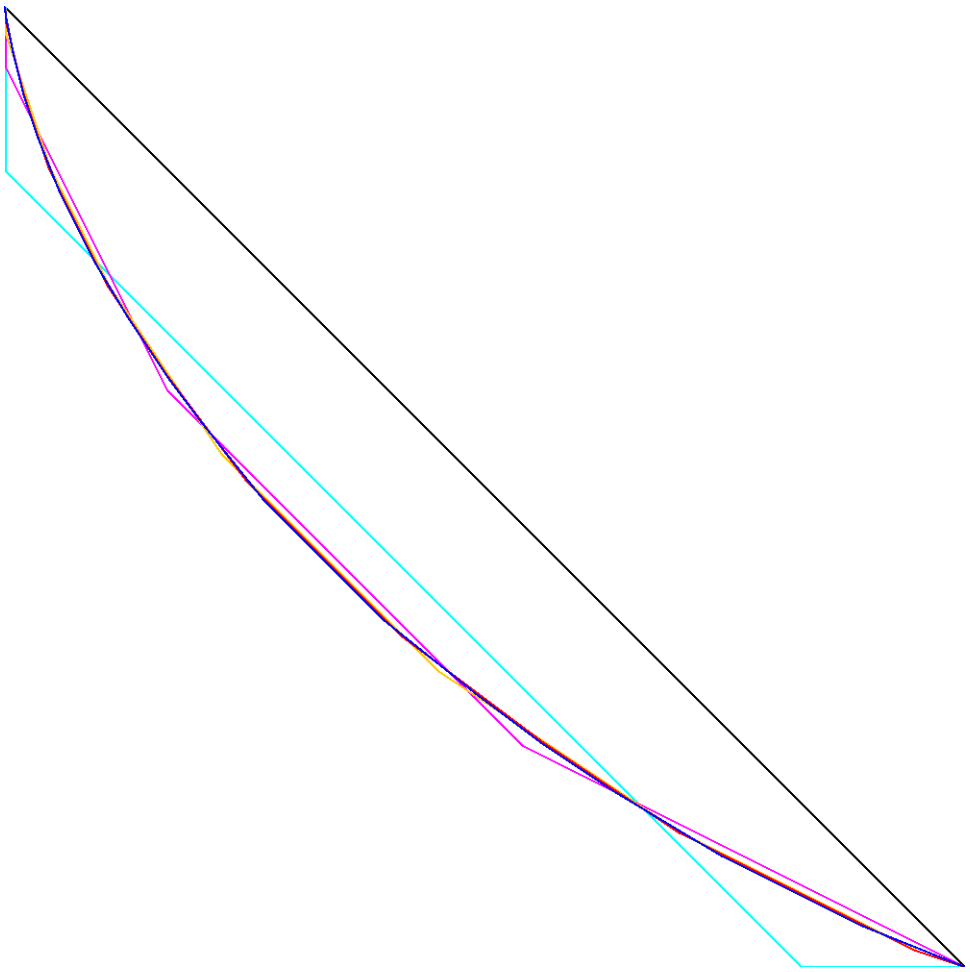

Figure 9: Result upto s = 5, without the optimal cycloid

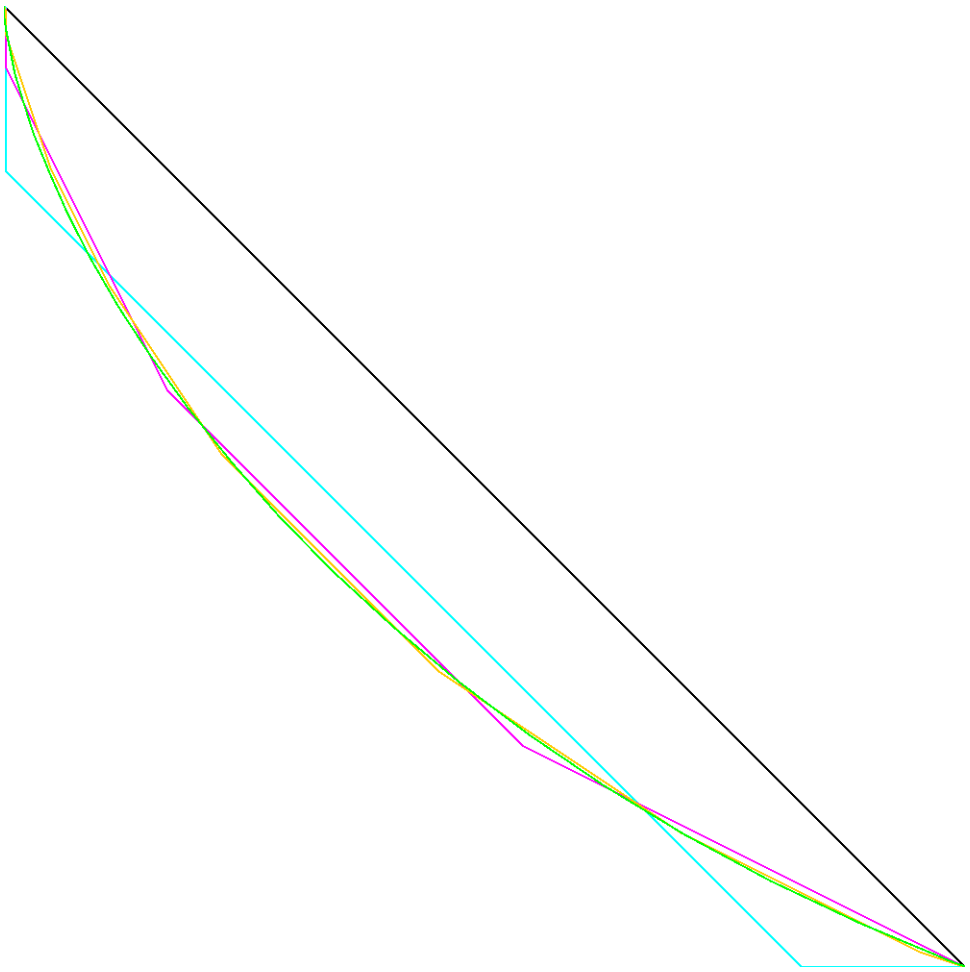

Figure 10: Result upto s = 3. Green curve is the optimal cycloid

6 Runtime Analysis

The runtime of Dijkstra's Algorithm is

$$\mathcal{O}(|V| + |E| \log |E|)$$

The graph used in the search divides each side of the square into n segments, thus giving $n + 1$ vertices on each side, and so a total of $(n + 1)^2$ vertices. Each vertex has a maximum of $\frac{s(1+s)}{2} \times 8$ edges, which is $\mathcal{O}(s^2)$ edges.

Thus the runtime of the searching algorithm is:

$$\mathcal{O}((n+1)^2 + (s^2)\log(s^2))$$

after simplification:

$$\mathcal{O}(n^2 + s^2 \log s)$$

7 Conclusion

Using a graph theory approach, the search for the answer to the Brachistochone can be done in a reasonable amount of time and the precision of the result can be increased by increasing the layers of neighbors to search, at the expense of runtime.

8 Source Code

The source code can be found here:
`https://github.com/ysf199711/BrachistochroneGraph`

References

[1] H. H. Roomany, "A graph theoretic approach to the brachistochrone problem," *Computers In Physics*, pp. 303 – 308, 1990.

[2] "Brachistochrone curve," 2017.