Assignment 1: Brachistochrone

Mahathi Narayanaswamy
BSP20022

Goal:

To arrive at the Brachistochrone, that is, the curve of fastest descent, numerically.

Logic behind algorithm:

A Monte-carlo algorithm was used following the following logic

1. Define an arbitrary initial path by the means of discrete steps

2. Modify the initial path by varying the discrete points by a random amount within a reasonable range

3. Compute time taken by the initial path

4. Compute the time taken by the modified path.

5. Compare the times and reassign the path with lesser time as the new initial path.

6. Repeat to obtain paths that tend to the analytically expected path(computed by Euler-Lagrange to be a Cycloid).

Resultant path

1. What will be the shape of the road?
   Cycloid

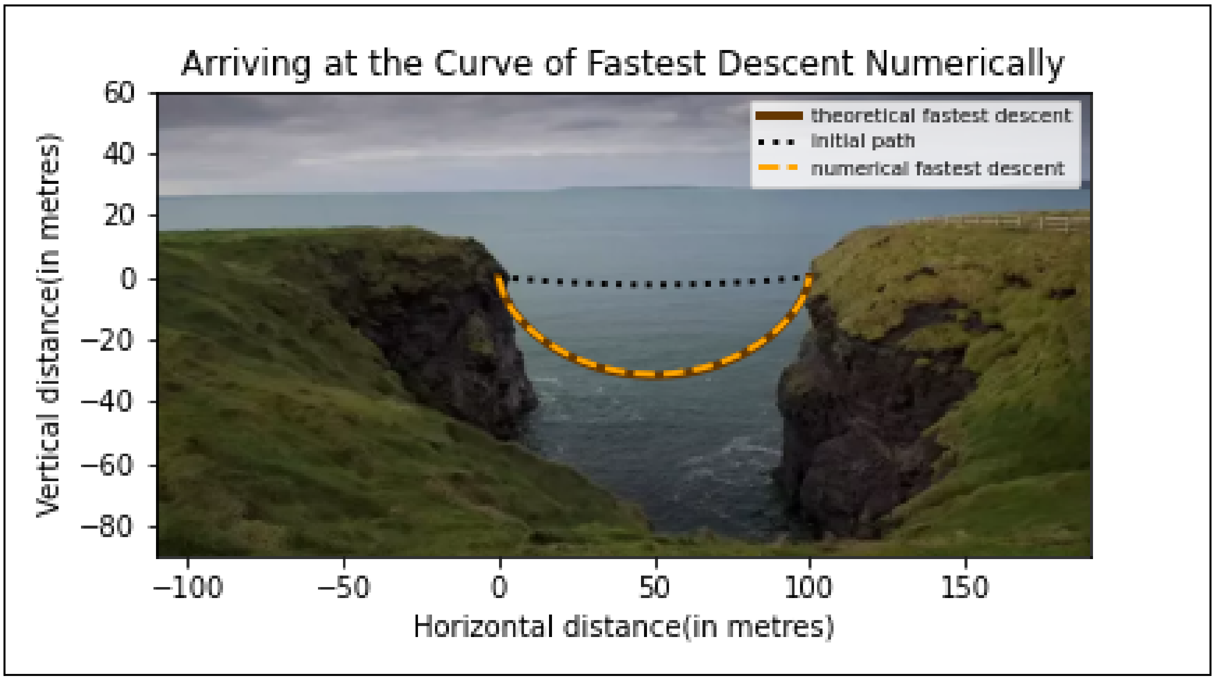
2. How low does it dip below the straight line connecting the two ends?
   31.56 m

3. What is the full length of the road?
   126.24 m

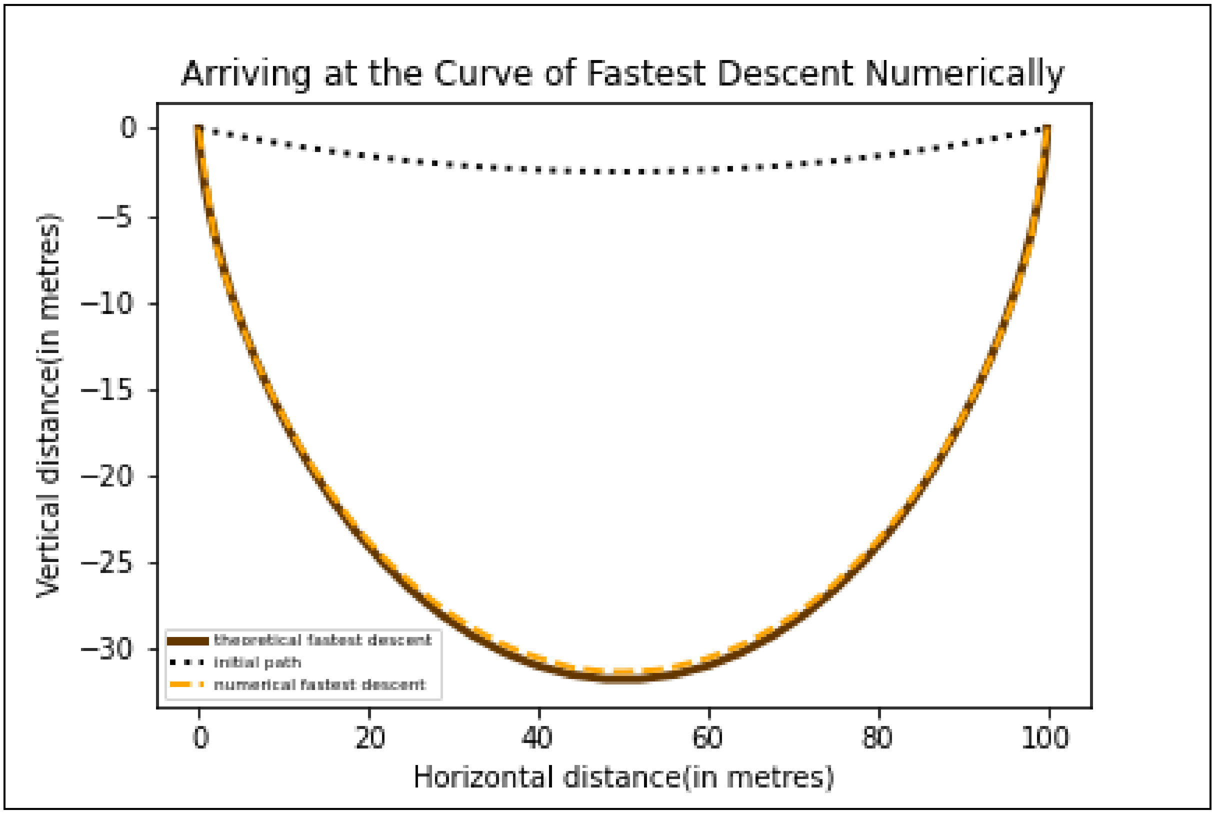
4. What is the time taken by the object to slide through the road?
   8.04 secs

Resultant Graphs:

Presentable Graph:



Representative Graph:



Program code:

Language: Python

```
#importing required libraries
import numpy as np
import random as rnd
import matplotlib.pyplot as plt

import time

"""
Let us consider 2 points A(x1, y1) and B(x2, y2).
We want to find a trajectory that minimises the time taken from A to B when
released from rest.
Since frames are relative, let A be at the origin (0,0).
"""

#Final Point
x2 = 100
```

```
y2 = 0
X = np.sqrt(x2**2 + y2**2)

#Gravity influencing the trajectory
g = 9.8

"""
In order to obtain this path let us begin by considering N piecewise linear
paths.
Let the projection of each piecewise linear segment be fixed.
This allows us to compute the increment in x for each segment.
"""

N = 50 #N is the number of segments
dx = x2/N #dx is the increment of x in each segment

"""
Given N segments, we have len(x) points that presently define our
trajectory.
Let us initialise the positions of the len(x) points
"""

# intialising x coordinates of the len(x) points
# these coordinates will remain unchanged throughout
x = np.arange(0, X+dx, dx)

#let us define an initial curve
def path(x):
    return 0.001*x**2 - 0.1*x

#the time taken in each piecewise linear segment of a path is given by
def pathtime(dx_, y_):

    T = 0
    for i in range(1, len(y_)):
        dy_ = y_[i] - y_[i - 1]
        T += np.sqrt(2 * (1 + (dx_/dy_)**2)/(g)) * (abs(np.sqrt(abs(y_[i]))
- np.sqrt(abs(y_[i - 1]))))

    return T

#let us now modify our initial path
def modpath(y_):
```

```
    y0 = np.zeros(len(y_))
    for i in range(len(y_)):
        c = rnd.uniform(0.999, 1.001)
        y0[i] = c*y_[i]
        y0[0] = 0
        y0[-1] = 0

    return y0


#comparing times of the modified path with that of the initial path and
accepting or rejecting new path
def checking():
    y = path(x)
    plt.plot(x,y)
    for i in range(0, 200000):

        newy = modpath(y)

        t1 = pathtime(dx,newy)
        t2 = pathtime(dx,y)

        if t1 <= t2:
            for i in range(len(x)):
                y[i] = newy[i]
        elif t1 > t2:
            for i in range(len(x)):
                y[i] = y[i]

    plt.plot(x,newy)
    pathtime(dx,newy)
    print("Computational time = ", pathtime(dx, newy))

#Analytical solution which is a cycloid as derived from Euler-Lagrange
equation
def analytical():
    theta = np.linspace(0, 2*np.pi, N)
    r = x2/(2*np.pi)
    ax =  r*(theta - np.sin(theta))
    ay = r*(np.cos(theta) - 1)

    plt.plot(ax,ay)
    analytical_time = 2*np.pi*np.sqrt(r/g)
```

```
    print("Analytical time = ", analytical_time)

#function call
starttime = time.time()
checking()
analytical()
runtime = (time.time() - starttime)
print(runtime)
```