Task: To find the path of quickest descent from point A[1,0] to B[0,1], i.e. the Brachistochrone curve

First, the form of the line joining A and B had to be established. For ease of computation, this was chosen to be a series of straight lines at equal x-spacings and variable y-spacings. Secondly, a function needed to be written which calculated the time required for a ball to travel down a given trajectory, assuming no friction and that all the gravitational energy is converted into kinetic energy. This was done assuming the conservation of energy (i.e. assuming the kinetic energy is conserved between segments in a line). The following formulae could then be written for the i^{th} point along the line:

$$v_i = \sqrt{v_{i-1}^2 - 2g(y_i - y_{i-1})}$$

$$v_{avg} = \frac{v_i + v_{i-1}}{2}$$

$$d = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

$$dt = \frac{d}{v_{avg}}$$

The dt refers to the time required to travel down a given section of a line. Summing all the dt values gave the time required for a ball to travel down a given trajectory. As a reference to see if the curves generated were indeed quicker trajectories than the straight line joining A and B, the latter was first calculated to be 1.990.

To expedite the process of minimisation, an initial guess for the shape of the curve was made which was qualitatively similar to a Brachistochrone curve. As A and B are equidistant from the origin, a circle with centre [1,1] and radius 1 would pass through both A and B and the resultant quarter arc would give a reasonable initial trajectory. The horizontal distance between A and B, equal to 1, was divided into sections with coordinates for the boundaries. The formula for calculating the y-values at each x-value is given below.

$$y_i = \sqrt{r^2 - (x_i - a)^2} + b$$

Where a and b are the abscissa and ordinate, respectively, of the centre of the circle; in this case, 1 and 1. As a further reference, the travel time for this initial curve was calculated to be 1.847 – already faster than the straight line.

With the initial shape of the curve established, a random search method was employed to modify the shape such that the time taken to get from A to B was minimised. This was achieved by utilising a for loop in which a random coordinate was selected at each iteration and the y-value was changed by a small amount randomly up or down. The first and last coordinates were kept constant as they represented the points A and B. If the resultant trajectory resulted in a shorter travel time, then this new trajectory was stored as the current curve shape. This

was done initially using 100 line segments and 10,000 iterations per segment, for a total 1,000,000 iterations. The maximum amount by which a y-value could change was set to ±5 % of its current value. As can be seen in Figure 1, this resulted in a jagged curve which intuitively would not be the Brachistochrone. The plot of travel time at each iteration further shows that the time did not converge at a minimum value after a certain number of iterations. The calculated travel time for this curve was 1.909, so still faster than the straight line but slower than the initial guess.

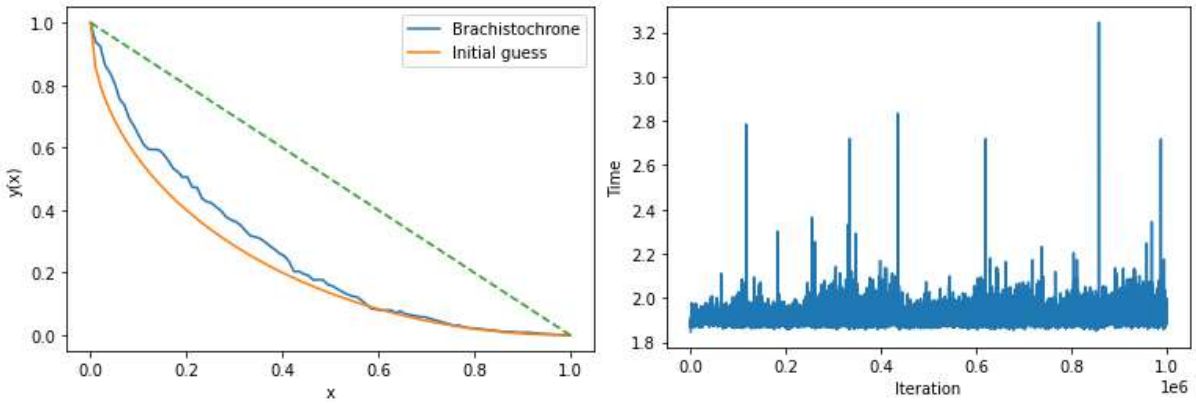

Figure 1. Estimated Brachistochrone with 5 % max change in y and 1,000,000 iterations.

One possible cause of this could be that the maximum allowed change in y was too large. This could result in a coordinate being close to the optimal value and the change being too great to allow it to get any closer. To abate this, the maximum change in y was set to ±0.5 %, with all other parameters remaining the same. Upon inspection of Figure 2, it is apparent that this improved the results greatly, with the final curve much smoother than the previous one and a reduced travel time of only 1.823. The time as a function of iteration graph however shows that convergence was not quite reached.

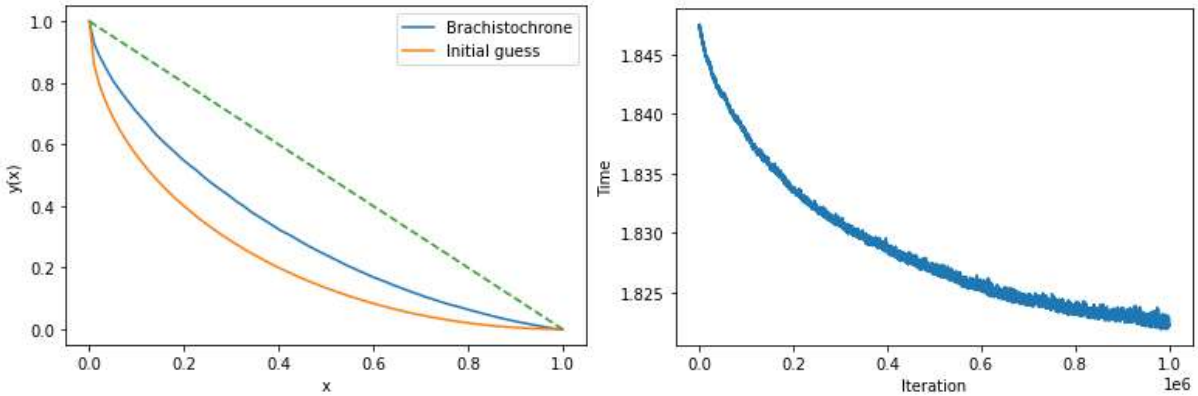

Figure 2. Estimated Brachistochrone with 0.5 % max change in y and 1,000,000 iterations.

To check if convergence could eventually be reached using this function, the number of iterations was increased from 1,000,000 to 10,000,000. As can be seen in Figure 3, this gave a similar shaped graph with a convergence occurring between 1-2 million iterations. The final calculated travel time was 1.822, so only a marginal difference from the previous value with 1,000,000 iterations. It should be noted that the computation time was much longer with 10,000,000 iterations for only a very small increase in accuracy.

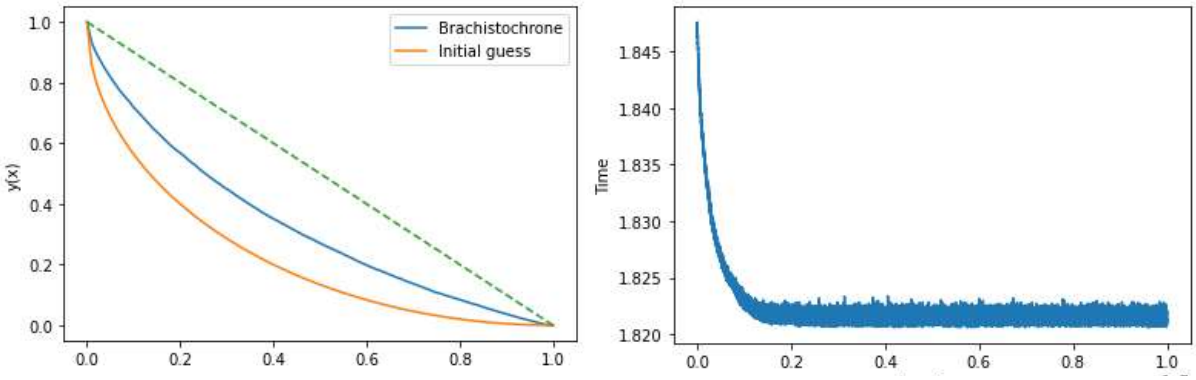

Figure 3. Estimated Brachistochrone with 0.5 % max change in y and 10,000,000 iterations.

Reducing the number of line segments from 100 to 10 whilst keeping all other parameters the same also had a marked impact on the travel time and number of iterations required for convergence. As can be seen in Figure 4, it changed the shape of the graph to be less smooth, whilst maintaining the same general shape, and resulted in convergence occurring after only 7000 or so iterations. With only 10 segments, the travel times for the straight line between A and B, the initial guess curve and the final Brachistochrone curve reduced to 1.886, 1.781 and 1.764, respectively. Compared to the values with 100 segments which were 1.990, 1.847 and 1.822, it is clear that using fewer segments to construct the curve reduced the travel time in all scenarios. This is unexpected as using more segments should allow for a curve which more closely resembles the true Brachistochrone to be constructed. Perhaps rounding errors from millions of iterations, as required for convergence when 100 segments are used, lead to a significant error over time, though this would not have affected the straight line travel time. More likely then is that the function for calculating and summing the travel time for each segment to give the total line travel time had rounding errors which compounded when many segments were used. The balance, therefore, is to use enough segments to construct a curve which closely resembles the true shape of the Brachistochrone, whilst using as few segments as possible to minimise rounding errors.

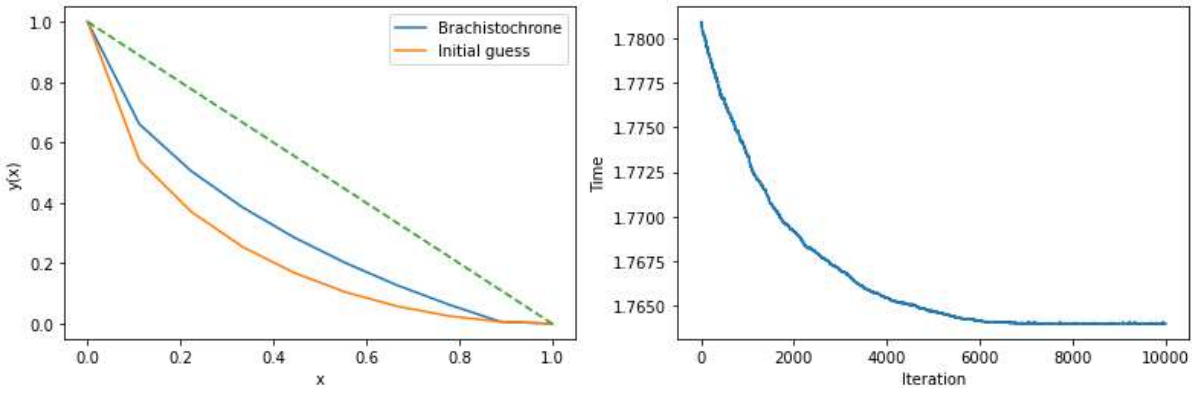

Figure 4. Estimated Brachistrochrone when only 10 segments were used.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 21:04:41 2021

@author: tom_m
"""

from __future__ import division
import numpy as np
from random import randrange, random
import matplotlib.pyplot as plt
from numba import jit

@jit
def time(x, y):
    # Returns the time it would take for a ball to travel down the total
    # length of the trajectory

    t = 0
    v = 0
    g = 1
    n = len(x)-1

    for i in range(1, n):
        dx = x[i] - x[i-1]
        dy = y[i] - y[i-1]
        v_f = np.sqrt(v**2 - 2*g*dy)
        v_avg = (v + v_f)/2
        dist = np.sqrt(dx**2 + dy**2)
        dt = dist / v_avg
        v = v_f
        t = t + dt

    return t


def brachistochrone(A, B, N):

    n_opt = 100000 # Number of optimisation trials per segment
    n_trials = N * n_opt # Total number of optimisation trials run
    times = np.empty(n_trials, float) # Array to store the times for each
    # trajectory
    x = np.linspace(A[0], B[0], N) # x-values for the segment boundaries
    y_line = np.linspace(A1[1], B1[1], N) # y-values if a straight line
were
    # drawn between A and B
    y = np.zeros(len(x), float) # Array to store the calculated y-values

    t_line = time(x, y_line) # Time taken for ball to travel down a
straight
    # line from A to B

    if A[1] == B[0]:
        for j in range(0, N):
            y[j] = -np.sqrt((A[1]-A[0])**2 - (x[j] - A[1])**2) + B[0]
            # Sets the condition that the initial shape of the curve
joining
            # A and B can be the quarter sector of a circle if A[y]=B[x]
```

```
    else:
        y = y_line
        # Otherwise the initial guess is simply a straight line, for ease

    x_start = np.copy(x) # Copy of x and y to store as initial guess
    y_start = np.copy(y)

    t_start = time(x_start, y_start)
    times[0] = t_start # First element in times array is time for initial
    # shape guess

    for k in range(1, n_trials):

        ran = randrange(1, N-2) # Selects a random number which will be
the
        # index of the y-value to change. Location of A and B are
constant
        # so index must be within the inner coordinates

        if y[ran] != 0:
            maxchange = y[ran] * 0.01
            # If the y-value is non-zero then the max it can change from
its
            # current position is 1% of its currrent value
        else:
            maxchange = random() * 0.01
            # If it is zero, then it can change up to 1% of a random
number
            # between 0 and 1

        changey = (random()-0.5)*maxchange
        # Amount that y-value changes is between -0.5 and 0.5 multiplied
by
        # the maximum amount its allowed to change

        yt = np.copy(y) # Stores copy of the new y-values
        yt[ran] = yt[ran] + changey # Adds that change to the random y-
value


        times[k] = time(x, yt) # Calculates time for new coordinates
        if times[k] < times[k-1]:
            y = yt
            t_start = times[k]
            # If the current time is less than the time at the previous
            # iteration, then the current set of y-values is made the new
set



    print('t for straight line:', t_line, 'Brachistochrone t: ', t_start,
          't for initial guess', times[0])

    plt.plot(x, y, label = 'Brachistochrone')
    plt.plot(x_start, y_start, label = 'Initial guess')
    plt.plot((A[0],B[0]), (A[1],B[1]), '--')
    plt.xlabel('x')
    plt.ylabel('y(x)')
```

```
    plt.legend()
    plt.show()
    plt.figure()
    x_axis = np.linspace(1, n_trials, n_trials)
    plt.plot(x_axis, times)
    plt.ylabel('Time')
    plt.xlabel('Iteration')


A1 = [0, 1]
B1 = [1, 0]
N = 100


brachistochrone(A1, B1, N)
```