Numerical Algorithms

Fall 2019

# Assignment 1: Newton’s Method

September 24th 2019

By:

Simone Eandi

Jose Castro

Abstract:

Sheila is a students who oftenly drives an old and rusty car, reason why, one day the speedometer’s needle fell apart, fostering sheila to glue it back again. Sheila glued the needle on an incorrect position, thus, she wish to know how big the difference between the displayed and the real speed is. Sheila made some test taking note of the speed the needle was showing and the total amount time all the tests took.

This assignment pretends to demonstrate how the Newton’s method for finding the roots of a function is useful for solving Sheila’s problem. The solution will be implemented on Python and explained here.

## Table of contents

Input data (Given data): 3

Mathematical development: 3

Newton’s Method 6

Solving the problem with Python 6

References: 9

## Input data (Given data):

Number of Sheila’s journey segments:

n = 4

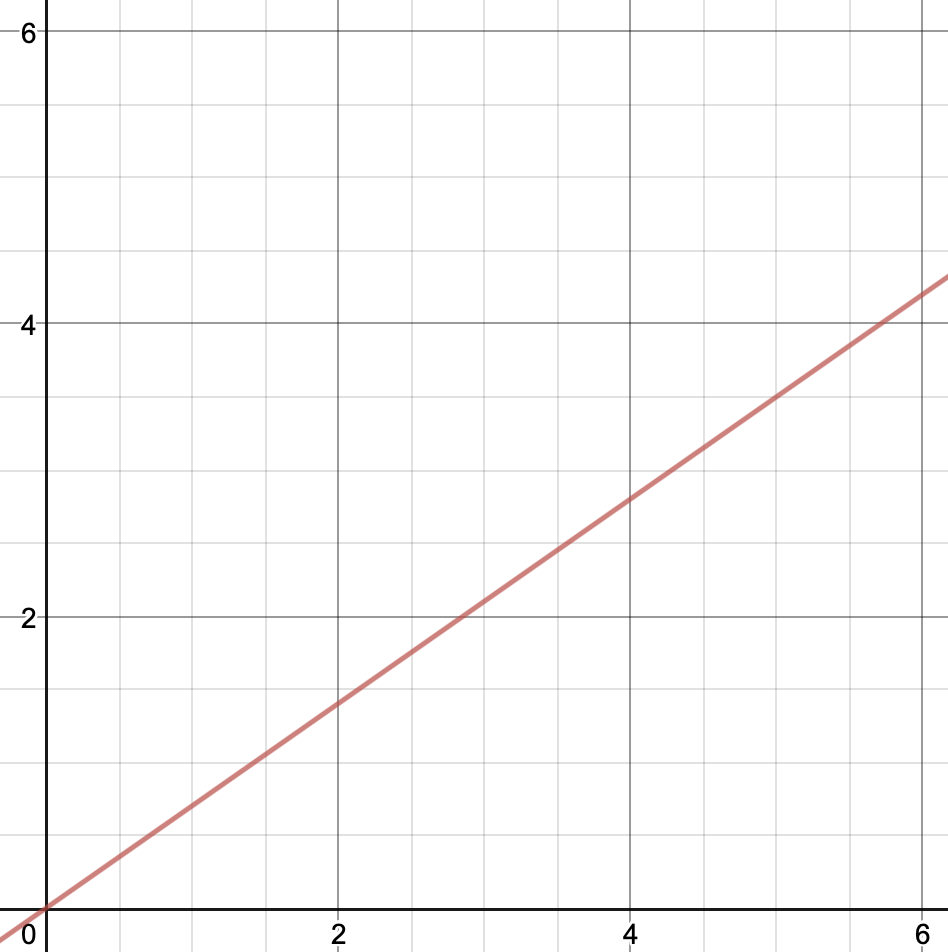
Total of time of her journey (in hours):

t=10

Distance traveled (di) and speedometer reading (si) per segment:

| **Segment** | **Distance (miles)**  **di** | **Speed (miles/hour)**  **Si** |
| --- | --- | --- |
| 1 | 5 | 3 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 3 | 1 |

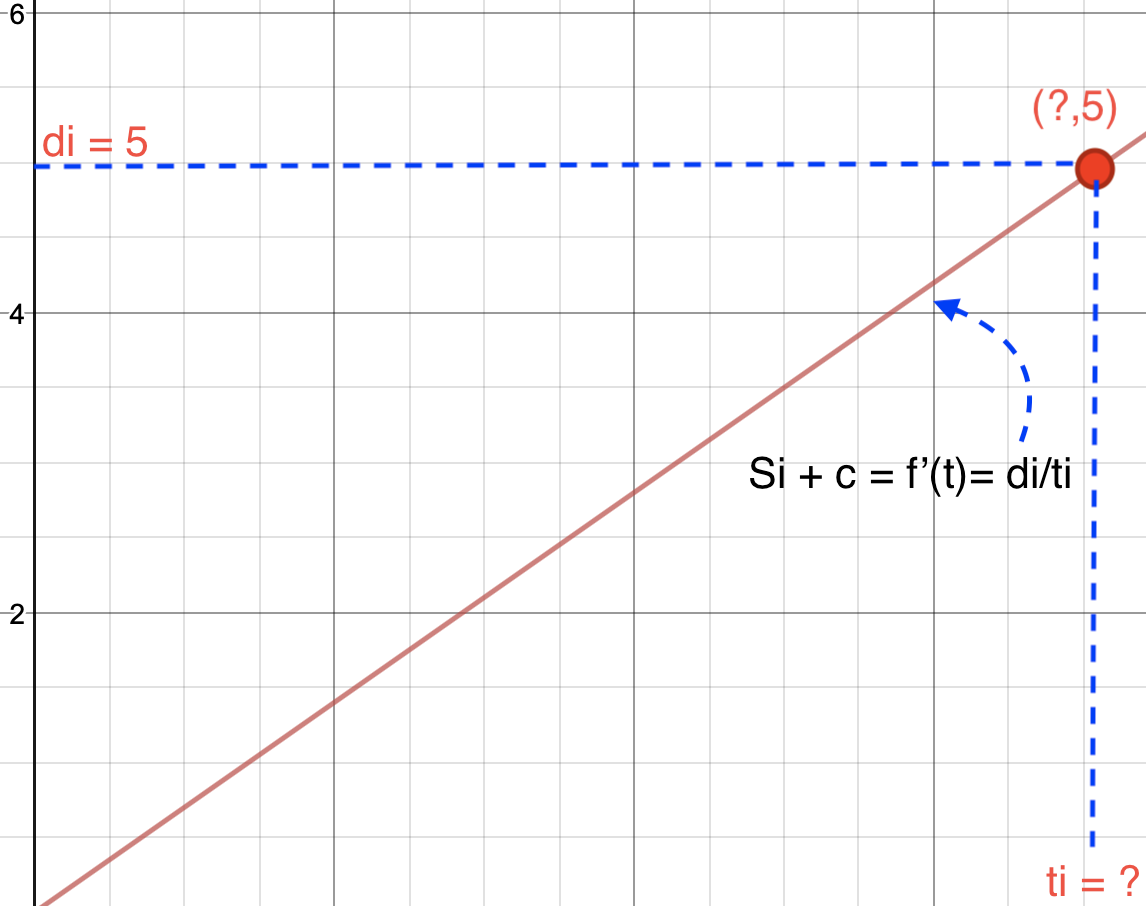
## Mathematical development:

We know that Sheila’s speed was constant in each segment, but different among them, meaning that the speed at segment 1 was always the same during the time the segment lasted. We can sketch a graph that represents any of the segments traveled by Sheila as shown on figure 1.1.

**Figure 1.1:** Sketch of a graph representing any of the segments traveled by the student. Note that since the speed is constant for every segment, the graph has a linear behavior.

Since the reported speed is not correct and since we ignore what was the time that each segment lasted, we cannot determine what the real speed is. We only know that the reported speed needs to be corrected by implementing a constant “c”, meaning that, since the slope of the graph represents the speed, we can conclude that for every segment:

**f’(s) = si + c.**

****

**Figure 1.2:** The slope of the function represent the velocity at which Sheila was traveling in a segment “i”. Note that the “t axis” has been ignored since we do not know how long each segment lasts.

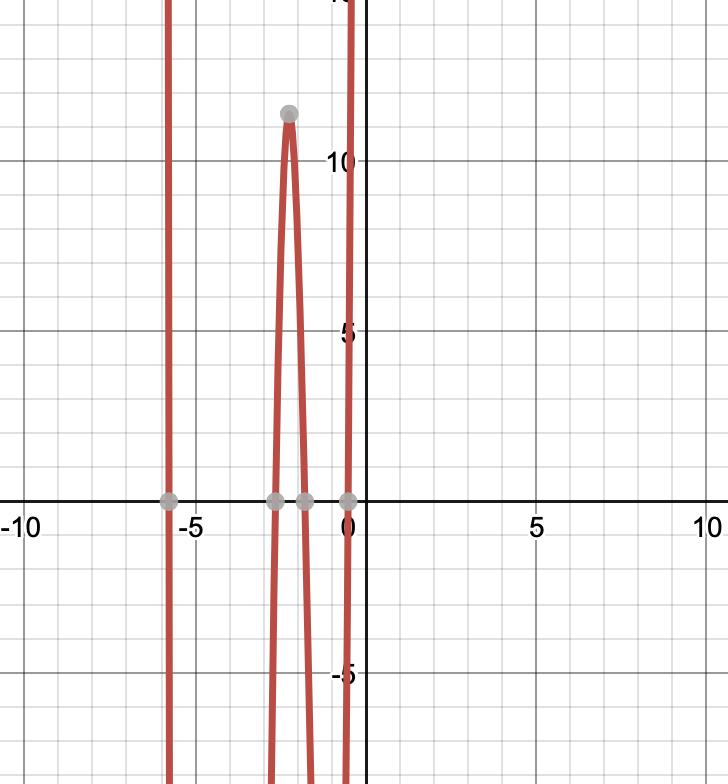
Since we know that f’(t) = di/ti = Si + c, we can solve for ti as follows: **,** and since we know that the total time of all the segments is t = 10 we conclude that:

F(c) = (∑(from i=1 to i=n) di/(Si + c)) - 10

And its derivative:

f’(c) = ∑(from i=1 to i=n) -di/(Si + c)²

By plotting the given function we obtain the new graph that represents the system in function of “c”.



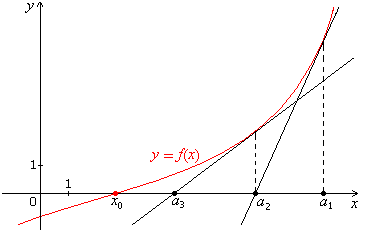
**Figure 1.3:** Graphic representation of the function F(c) where the first root of the graph represents the desired value of c.

## Newton’s Method

Newton’s method converges to the nearest root of a function in a fast way, decreasing the error in each calculation in a quadratic form. This method, calculates the tangent line of the function and iterates each time that this tangent intercepts with 0 on the y axis, adopting this new value as xi+1.

The resulting formula for this process is:

**Xi+1 = Xi - F(Xi)/f’(Xi)**



**Figure 2.1:** Newton’s method approaches the zero of the function by iterating a tangent line in a given point and assigning the new point where the tangent equals 0. Image retrieved from: Ving, P. K. *Calculus of one real value*

## Solving the problem with Python

Input.txt

4

10

5 2 3 3

3 2 6 1

Assigment1.py

if \_\_name\_\_ == “\_\_main\_\_":

    solver = SpeedometerSolver('input.txt') #instantiate the class takin the data inputs from the "input.txt"

    solver.run(-0.1) #initial value for Newton's method is -0.1

\_\_init\_\_(self, filename): instantiates an object by receiving the name of the input file and decompose it in different variables.

            n= number of segments

            T = duration of all the segments (in hours)

            distances = list of distances traveled (in miles)

            speeds = list of speeds reported by Sheila (in miles/hrs)

def \_\_init\_\_(self, filename):

        file = open(filename, "r")

        self.n = int(file.readline())

        self.T = int(file.readline())

        self.distances = list(map(int, file.readline().rstrip('\n').split()))

        self.speeds = list(map(int, file.readline().strip('\n').split()))

        file.close()

run(self, c0): invoke the Newton's method and print the resulting error on time calculation

def run(self, c0):

        self.newtons\_method(c0)

        print("Time error: {0}".format(self.test\_solution(self.c)))

newtons\_method(self, c0): looks for the closest root to the initial value

def newtons\_method(self, c0):

   self.c = c0

   for i in range(0, 10):#Iterative formula: ci+1 = ci - f(si)/f'(si)

self.c = (self.solve\_equation(self.c) / self.solve\_derivative(self.c))

   print(self.c)

solve\_equation(self, c): Solves the main equation of the problem:

F(c) = (∑(from i=1 to i=n) di/(Si + c)) - 10

def solve\_equation(self, c):

        s = 0

        for di, si in zip(self.distances, self.speeds):

            s += di / (c + si)          #f(s)= di/(c+si)

        return s - self.T

solve\_derivative(self, c): Solve the differentiated equation:

f’(c) = ∑(from i=1 to i=n) -di/(Si + c)²

def solve\_derivative(self, c):

   s = 0

   for di, si in zip(self.distances, self.speeds):

       s -= di / ((c + si)\*(c + si)) #f'(s) = -di/(c+si)^2

   return s

test\_solution(self, c):Calculates the error of the solution. If the solution is correct, this would return a value that is close to 0

def test\_solution(self, c):

    time = 0

    for di, si in zip(self.distances, self.speeds):

        time += di / (si + c) #Iterate to sum the time of each segment traveled by Sheila

    return self.T - time

#If the answer is correct, this should return a valuea close to 0

Running the program will result on the following output:

-0.7847157252570663

-0.6595438875467998

-0.5523861134391509

-0.5122222295578926

-0.5086768807929416

-0.5086533778138455

-0.508653376795394

-0.508653376795394

-0.508653376795394

**-0.508653376795394**

**Time error: 0.0**

Meaning that the constant c = -0.5 and using this value to evaluate

∑(from i=1 to i=n) di/(Si + c)

Will result in t = 10, meaning that there’s no error in the calculation.

We can compare this result to the plot obtained in **figure 1.3,** where the interception of the graph with respect to the “y-axis” is shown to be at **(-0.5,0)**.

## References:

*Hormann, K. (2018). Numerical Algorithms 1 Root Finding.*

*Ving, P. K. (n.d.). Calculus of one real value. Retrieved from http://www.phengkimving.com/calc\_of\_one\_real\_var/08\_app\_of\_the\_der\_part\_2/08\_05\_approx\_of\_roots\_of\_func\_newtons\_meth.htm*