Numerical analysis: Assignment 1

Niccolo Zuppichini

September 22, 2020

Exercise 1

My first implementation of the newton method implementation diverged to -inf, therefore I worked on a different implementation before coming up with the correct one.
This second implementation I've done for the assignment doesn't involve the newton method and can be found in *bisection.py*. As the name suggests, it is based on the bisection method, we have two pointers which bound the solution from above and below (i.e. the initial values for these two pointers should include the solution between them). Let's call the pointer bounding from below "low" and the one from above "high". Then, iteratively, we compute a "feedback" on the value of the midpoint (i.e. (high+low)/2) and based on this feedback we choose to discard the left interval [low, mid] or the right interval [mid, high] and then repeat the iteration until we get convergence.
The feedback on the current midpoint is given by following the indications in the problem:

1. the real speed S = c+s[i] is always positive. Therefore if c+s[i] is less than zero, increase c (i.e. discard left interval)
2. If the sum of the time to travel each segment is greater than the total time for the whole journey, increase c. In other words, if $\sum_{i=1}^{N} \frac{d_i}{s_i+c} > T$ then increase c. Otherwise, if $\sum_{i=1}^{N} \frac{d_i}{s_i+c} > T$ then decrease c.

The code is in *code/bisection.py* and can be run by, with the given parameters as follows:

```
python3 bisection.py 4  --time 10  --distances 5 2 3 3 --speed 3 2 6 1
```

and it obtains an (approximated) solution of -0.5086.

By using the same function I've used on my *bisection.py* method implementation I've fixed my newton method.
The total time for the whole journey is given by

$$f(c) = \frac{\sum_{i=0}^{N} d}{(\sum_{i=0}^{N} s + c)} - t$$

by differentiate $f(c)$ we obtain:

$$f'(c) = -\frac{\sum_{i=0}^{N} d}{(\sum_{i=0}^{N} s + c)^2}$$

Then computing the root of $f(c)$ using the newton method leads to the solution.

The code is in *newton.py* and can be run by, with the given parameters as follows:

```
python3 newton\_method.py 4  --time 10  --distances 5 2 3 3 --speed 3 2 6 1
```

and it obtains an (approximated) solution of -0.5086, as my bisection method. However, the rate of convergence of the newton method is much faster than the bisection method. Infact, to get the same decimal accuracy, newton methods takes 7 iteration and my bisection method takes 23.

One last consideration I feel to make is about the condition $s + c > 0 \forall s$. On my bisection method I check this condition at each iteration; on the other hand, I'm not sure if there's any direct way to check this condition on the newton method (other than checking it in the end and starting the newton method again from another initial guess).