Numerical Methods HW 3

Adam Kit, 3707437

4 May 2020

1 Task 1

Calculate the 32-bit floating point (binary) representation for π and e.

1.1 32-bit (single precision) Floating Point Single Precision

A single precision floating point number has 3 sections of bits. The first section is 1 bit long, and represents the sign of the number; 0 if the decimal number is positive, 1 negative. The second section is 8 bits long and is the "exponent" of the number, which thus has a bias of 127 that makes sure the number fills the 8 bits (since 127 can be represented by 7 bits in binary). The last section is 23 bits long and represents the *mantissa*. This can be described as what is left over after finding the "exponent" part of the decimal number. The steps of converting a decimal to binary can be seen in the calculations of π and e below:

$\pi = 3.1415$ to binary $\rightarrow 11.00100100001 = 1.100100100001 * 2^1$ where the one in 2^1 is added to the bias to get the "exponent" $\rightarrow 1 + 127 = 128_{10} = 10000000_2$ and the mantissa is then what is left over in $1.100100100001 * 2^1 \rightarrow 0.100100100001$. The sign is 0 since π is positive, thus the final combination is: 01000000010010010000100000000000

$e = 2.71828_{10} = 10.1011011111_2 = 1.0101101111 * 2^1$ so the exponent is: $127 + 1 = 128_{10} = 10000000_2$ and the mantissa: 0101101111 and the sign: 0. All together: 01000000001011011110000000000000

2 Task 2

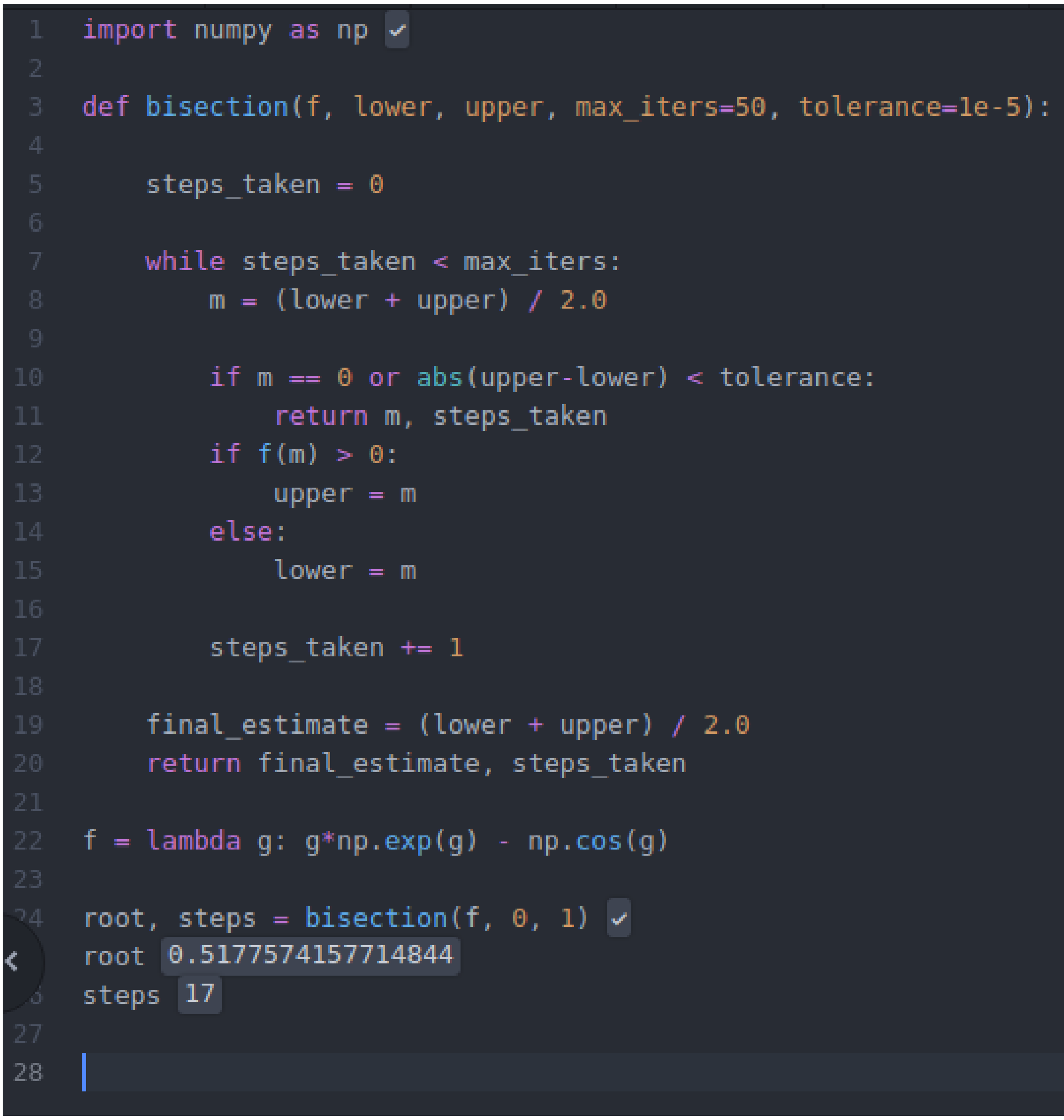
Estimate a root of $f(x) = xe^x - cos(x)$ correct upto 4 decimal places using bisection method. You can use the initial approx. for bracketing as 0and 1. You can write a program to implement the iteration.

2.1 Bisection Method

The bisection method is in practical, a binary search method. One starts with two sides or guesses that define the interval in which the root exists, and at each step take the midpoint of the two guesses. If the midpoint is zero (the guesses are the same) or if the guesses are sufficiently close to one another, return the midpoint. Otherwise, if the f(midpoint) is negative, then replace the lower bound with the midpoint, and if f(midpoint) is positive, then replace the upper bound with the midpoint.

In the end I found that the root to 4 decimal points is 0.5178.

A snapshot of the code can be seen in Figure 1, as my Latex editor in Atom is having issues with the listings package :(. In addition one can find the github repository for all of the homeworks can be found here, as well as a tutorial on popular root finders for the other classmates who are not so good at python here.



```
import numpy as np

def bisection(f, lower, upper, max_iters=50, tolerance=1e-5):

    steps_taken = 0

    while steps_taken < max_iters:
        m = (lower + upper) / 2.0

        if m == 0 or abs(upper-lower) < tolerance:
            return m, steps_taken
        if f(m) > 0:
            upper = m
        else:
            lower = m

        steps_taken += 1

    final_estimate = (lower + upper) / 2.0
    return final_estimate, steps_taken

f = lambda g: g*np.exp(g) - np.cos(g)

root, steps = bisection(f, 0, 1)
root 0.5177574157714844
steps 17
```

Figure 1: Screenshot of code