Week 4 Hand-in

NumIntro 2019

Department of Computer Science
University of Copenhagen

Casper Lisager Frandsen <fsn483@alumni.ku.dk>

Version 1
Due: October 2nd, 08:00

Contents

1. Exercise **3**

2. Exercise **3**

3. Exercise **3**

4. Exercise **3**

5. Exercise **3**

1. Exercise

To formulate this as a root finding problem, we first have to find the formula for S, so we can rewrite:

$$S = \pi r l + \pi r^2$$

Where l is the slant height of the cone.

$$l = \sqrt{r^2 + h^2}$$

We rewrite this into one expression, and insert the known values:

$$\begin{aligned} 1200 &= \pi r \sqrt{r^2 + 400} + \pi r^2 \\ 0 &= \pi r \sqrt{r^2 + 400} + \pi r^2 - 1200 \end{aligned}$$

Now the expression can be solved as a root finding problem.

2. Exercise

This exercise has been completed in the Q0020.py hand-in.

3. Exercise

Using the secant algorithm implemented in Q0020.py, with starting values of $x_0 = 10$ and $x_1 = 12$, we find that the solution to the problem is approximately 11.196 with precision 10^{-9} after 4 iterations, not including x_0 and x_1.
This has been done by implementing the expression we found in exercise 1 as a function just like the "f" function in the file.

4. Exercise

I haven't been able to solve this exercise, and would appreciate some hints for how to solve it.

5. Exercise

I have chosen to solve this using the bisection method. I decided the easiest way to do this was by implementing it in python. It is included as a function named "bisect" in the Q0020.py file.
With this script we can clearly see that the bisection method converges on a solution significantly more slowly than the secant method. An advantage of both of these functions is that they do not need to calculate the derivative of the function in question. This is useful when finding that derivative is computationally expensive.

```python
def bisect(x0: float, x1: float, f, epsilon: float, iter_max: int) -> float:
    print("\nStarting Bisect algorithm with x0 = {0} and x1 = {1}".format(x0,x1))
    for i in range(iter_max):

        mid = (x0+x1)/2
        print(i,mid,f(mid))

        if (f(mid) == 0) or ((x1-x0)/2 < epsilon):
            return mid

        if f(x0) * f(mid) < 0:
            x1 = mid
        else:
            x0 = mid

    print("Reached max number of iterations")
    return(mid)
```

Figure 1: The code for implementing the bisection method