Statistik un Data Science für die Informatik

Humboldt-Universität zu Berlin, SS 22, Prof. Dr. Alan Akbik

Übungsblatt 5

Task 2

a)

Gegeben: $f_1(x) = \sqrt{x+1} + \frac{sin(x)}{10} - 2$ with a starting point $x_0 = 8$.
Gesucht: zeros of the function $f_1(x)$.

$$x_{m+1} = x_m - \frac{f(x_m)}{f'(x_m)}$$

$$x_{m+1} = x_m - \frac{\sqrt{x+1} + \frac{sin(x)}{10} - 2}{\frac{1}{2\sqrt{x+1}} + \frac{1}{10}cos(x)}$$

$$x_1 = x_0 - \frac{f_1(x_0)}{f_1'(x_0)} = x_0 - \frac{\sqrt{x_0+1} + \frac{sin(x_0)}{10} - 2}{\frac{1}{2\sqrt{x_0+1}} + \frac{1}{10}cos(x_0)} = 8 - \frac{\sqrt{8+1} + \frac{sin(8)}{10} - 2}{\frac{1}{2\sqrt{8+1}} + \frac{1}{10}cos(8)} = 8 - \frac{1.09893582466}{0.15211666328} = 0.77570385147$$

$$f_1(x_1) = f_1(0.77570385147) = -0.5974227411562083$$

$$x_2 = x_1 - \frac{f_1(x_1)}{f_1'(x_1)} = 0.77570385147 - \frac{-0.59742274115}{0.44661176207} = 2.11338187943$$

$$f_1(x_2) = f_1(2.11338187943) = -0.14988455870028083$$

$$x_3 = x_2 - \frac{f_1(x_2)}{f_1'(x_2)} = 2.11338187943 - \frac{-0.14988455870028083}{0.23173476703} = 2.76017545564$$

$$f_1(x_3) = f_1(2.76017545564) = -0.02365919558228402$$

$$x_4 = x_3 - \frac{f_1(x_3)}{f_1'(x_3)} = 2.76017545564 - \frac{-0.02365919558228402}{0.16503549198} = 2.903533683$$

$$f_1(x_4) = f_1(2.903533683) = -0.0006820833588669917$$

$$x_5 = x_4 - \frac{f_1(x_4)}{f_1'(x_4)} = 2.903533683 - \frac{-0.0006820833588669917}{0.15589046448} = 2.90790908441$$

$$f_1(x_5) = f_1(2.90790908441) = -5.344790601213845e-07$$

$$x_6 = x_5 - \frac{f_1(x_5)}{f_1'(x_5)} = 2.90790908441 - \frac{-5.344790601213845e-07}{0.15564650388} = 2.90791251834$$

$$f_1(x_6) = f_1(2.90791251834) = -1.8851586958135158e-13$$

$$x_7 = x_6 - \frac{f_1(x_6)}{f_1'(x_6)} = 2.90791251834 - \frac{-1.8851586958135158e-13}{0.15564631324} = 2.90791251834$$

$$f_1(x_7) = f_1(2.90791251834) = -1.8851586958135158e-13$$

At this point, $f_1(x_6) = f_1(x_7) \approx -1.885e-13$, which means with the following iterations, the function values will be very similar to each other $\Longrightarrow$ the value of $x_6 = x_7 \approx 2.90791$ delivers the best approximation.

b)

Gegeben: $f_2(x) = -x^3 + 3x^2 - x - 1$ with a starting point $x_0 = 2.5$.
Gesucht: zeros of the function $f_2(x)$.

$$x_{m+1} = x_m - \frac{f_2(x_m)}{f_2'(x_m)} = x_m - \frac{-x^3 + 3x^2 - x - 1}{-3x^2 + 6x - 1}$$

$$x_1 = x_0 - \frac{f_2(x_0)}{f_2'(x_0)} = 2.5 - \frac{-0.375}{-4.75} = 2.42$$

$$f_2(x_1) = f_2(2.42) = -0.02$$

$$x_2 = x_1 - \frac{f_2(x_1)}{f_2'(x_1)} = 2.42 - \frac{-0.02}{-4.05} = 2.42$$

$$f_2(x_2) = f_2(2.42) = -0.02$$

At this point, $x1 = x2 \approx 2.42 \in [2, 2.5]$ and $f_2(x_1) = f_2(x_2) \approx -0.02$, which means that with the following iterations, the function value will stay the same and the value of $x_1 = x_2 \approx 2.42$ (limited to two digits after the comma) delivers the best approximation.

P.S.: The Newton's method was implemented with Python to make sure the numbers are correct. Unfortunately, due to the limitation on the number of possible files that could be submitted, we could not attach the Python script, too. For this reason, we provide the code here:

```
from typing import Callable, Union
import numpy as np
from math import pow, isclose
from scipy.misc import derivative
```

```
def derivative_(func: Callable, x: float) -> float:
    return derivative(func, x, dx=1e-5)


def approximation(x: float, func: Callable):
    return x - func(x) / derivative_(func, x)


def func1(x: float) -> float:
    return np.sqrt(x + 1) + np.sin(x) / 10 - 2


def func2(x: float) -> float:
    return -pow(x, 3) + 3*pow(x, 2) - x - 1


def newton(start: Union[int, float], func: Callable):
    x = float(start)
    y = func(x)
    y_prev = -9  # random number not equal 0
    iteration = 0

    while (y != 0):
        if (y == y_prev):
            break
        x_prev = x
        x = approximation(x_prev, func)
        y_prev = y
        y = func(x)
        print(f"x={x}, f(x)={y}")
        iteration += 1


    print(f"\nFinished in {iteration} iterations")
    print(f"y={y} for x={x}")
    return x


def main():
    print("Task a)\n")
    start = 8
    func = func1
    nullstelle_x = newton(start, func)
    assert func(nullstelle_x) == 0

    print("\nTask b)\n")
```

```
    start = 2.5
    func = func2
    nullstelle_x = newton(start, func)
    assert isclose(func(nullstelle_x), 0, abs_tol=1e-15)


if __name__ == "__main__":
    main()
```

The output of the file:

```
Task a)

x=0.7757038518335664, f(x)=-0.5974227409938351
x=2.113381879603041, f(x)=-0.1498845586601809
x=2.7601754557021967, f(x)=-0.023659195572019343
x=2.9035336830002274, f(x)=-0.0006820833588314645
x=2.90790908441493, f(x)=-5.344782929572744e-07
x=2.907912518339108, f(x)=-3.2729374765949615e-13
x=2.907912518341211, f(x)=0.0

Finished in 7 iterations
y=0.0 for x=2.907912518341211

Task b)

x=2.421052631579649, f(x)=-0.027555037180278408
x=2.414262619005194, f(x)=-0.00019623673865210023
x=2.4142135649254266, f(x)=-1.0209325829180216e-08
x=2.4142135623730954, f(x)=-1.3322676295501878e-15
x=2.414213562373095, f(x)=-8.881784197001252e-16
x=2.414213562373095, f(x)=-8.881784197001252e-16

Finished in 6 iterations
y=-8.881784197001252e-16 for x=2.414213562373095
```

Note that the number of iterations clearly depends on the number of digits after the comma (i.e., the fewer digits after comma allowed, the fewer iterations needed).