Bisection method and Newton method to find solution to non-linear equations

Ashraf Mohammed Hassan Anil SCT211-0255/2021

March, 18, 2023

ICS2207: Scientific Computing

The non-linear equation used by both methods to find its roots was:

$$f(x) = x^6 + x - 1$$

timeit() in python was used to measure time taken by both methods

1 Bisection Method for Finding Roots of a Function

1.1 Implementation of Bisection method in Python

Here's an example Python code for implementing the Bisection method:

```
# import math
import timeit


def f(x):
    return x ** 6 + x - 1


def bisection_method():
    a, b = 0, 2
    tolerance = 1e-6
    max_iteration = 1000
    count_iteration = 0
    if f(a) * f(b) >= 0:
        print("Root cannot be found; range[a, b] should generate f(a) and f(b) with opposite
    for i in range(max_iteration):
        count_iteration += 1
```

```
        c = (a + b) / 2
        if abs(f(c)) < tolerance:
            print(f"Number of iterations taken are {count_iteration}")
            print(f"Root found at x={c:.6f}")
            break

        elif f(c) * f(a) < 0:
            b = c
        else:
            a = c


bisection_method()

time_taken = timeit.timeit(bisection_method, number=10000)

print(f"Bisection method took an average of {time_taken / 10000:.6f} seconds per run over 1(
```

1.2 Explanation of the code

The interval $a = 0$ and $b = 2$ was chosen since $f(a) = 0$ while $f(b) = 65$ thus $f(a)$ and $f(b)$ have opposite signs thus it holds

The program returns the number of iterations required to reach the root and the average time the bisection method takes in 1000 runs

2 Newton-Raphson Method for Finding Roots of a Function

2.1 Implementation of Newton-Raphson method in Python

Here's an example Python code for implementing the Newton-Raphson method:
$f(x) = x^6 + x - 1$

```
import math
import timeit


def f(x):
    return x**6 + x - 1


def df(x):
    return 6*x**5 + 1
```

```
def newton():
    x0 = 3
    tolerance = 1e-6
    maxiterr = 1000
    count_iterations = 0

    for i in range(maxiterr):
        count_iterations += 1
        fx = f(x0)
        dfx = df(x0)

        x1 = x0 - (fx / dfx)

        if abs(f(x1)) < tolerance:
            print(f"Number of iterations taken = {count_iterations}")

            print(f"Root found at x = {x1:.6f}")

            break
        else:
            x0 = x1


time_taken = timeit.timeit(newton, number=10000)

print(f"Newton method took an average of {time_taken/10000:.6f} seconds per run over 10000 r
```

2.2 Explanation of the code

From the above code x_n was taken to be 2 since from the graph of $f(x)$ plotted against x the initial guess is to be obtained from the range of about -7.5 to 7.5 so that the method generates the roots.

However, the closer to zero the value is obtained from the better hence 2 was taken and not values like 4, 5, 6 etc.

The program returns the number of iterations required to reach the root and the average time the bisection method takes in 1000 runs

Figure below is the plot of $f(x)$ against x

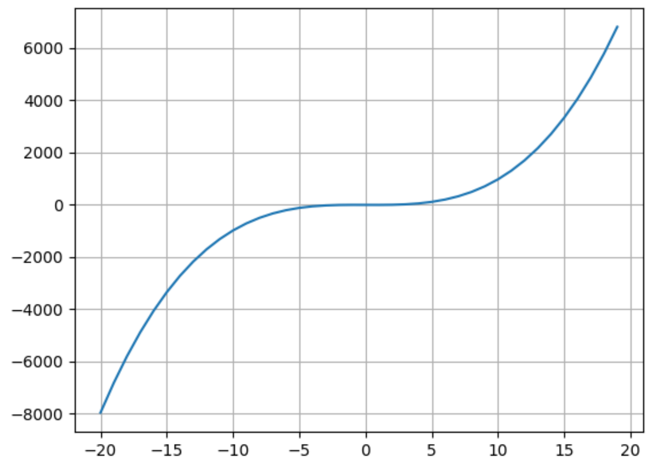

Figure 1: Plot of $f(x) = x^6 + x - 1$

2.3 Comparison between Bisection method and Newton method

The Newton-Raphson method converges faster than the bisection method. This is because the Newton-Raphson method used less number of iterations: 4 under 0.000021s while Bisection took: 19 iteration in 0.000030s

However, the Newton-Raphson method can be sensitive to the initial guess, and can converge to a different root or fail to converge altogether if the initial guess is not close enough to the true root. On the other hand, the bisection method always converges to a root, but it does so more slowly and may require more iterations to achieve the same level of accuracy.

2.4 Conclusion

Root obtained by both methods was 0.778090 which when substituted in $f(x) = x^6 + x - 1$ results to $f(x) = 0.00000108806$ which is a valid solution as it is close to 0 within the given tolerance