Solutions to Non-Linear Equations

1 Bisection Method

This method is based on the Intermediate Value Theorem which states that $f(x)$ is continuous on the interval $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs then there exists at least 1 root of $f(x)$ in the interval $[a, b]$.

Implementation

In this example, I used the following function :

$$f(x) = 6x^{19} - 5x^8 + 7x^5 - 9x^4 + 3x^3 + 6x^2 + 11x + 23 \tag{1}$$

Within the interval of : $[-3, 3]$ and a tolerance of : $1 - ef$
And Maximum Iterations of: 100

The Python code

```
import matplotlib.pyplot as plt
import numpy as np
import timeit

def f(x):
    return 6*x**19-5*x**8+7*x**5-9*x**4+3*x**3+6*x**2+11*x+23

a=-3
b=3
tolerance = 1e-6
max_iterations = 100
starttime = timeit.default_timer()

for i in range(max_iterations):
  c=(a+b)/2
  print(i)
  print(c)
  if abs(f(c)) < tolerance :
    print(f"Root found at x= {c:.6f}")
    break
  elif f(c)*f(a) < 0 :
    b=c
  else:
    a=c

print(f"BISECTION:{timeit.default_timer()-starttime:.6f} milliseconds");
```

Results

The Root obtained is : x=−0.926857 and it made 23 *iterations* with a time span of 19.523312 *milliseconds*

2 Newton Raphson

This method is used to find the roots of a differentiable function as it uses the tangent line to iteractively approach the root of the function.

Implementation

In this example, I used the following function :

$$f(x) = 6x^{19} - 5x^8 + 7x^5 - 9x^4 + 3x^3 + 6x^2 + 11x + 23 \tag{2}$$

and its derivative is :

$$df(x) = 114x^{18} - 40x^7 + 35x^4 - 36x^3 + 9x^2 + 12x^1 + 11 \tag{3}$$

My initial guess was : $x_o = 1$ and a tolerance of : $1 - ef$
And Maximum Iterations of: 100

The Python code

```
import matplotlib.pyplot as plt
import numpy as np
import timeit

def f(x):
return 6*x**19-5*x**8+7*x**5-9*x**4+3*x**3+6*x**2+11*x+23

def df(x):
return 114*x**18-40*x**7+35*x**4-36*x**3+9*x**2+12*x**1+11

x0 = 1
tolerance = 1e-6
max_iterations = 100
starttime = timeit.default_timer()

for i in range (max_iterations):
    fx=f(x0)
    dfx=df(x0)
    x1=x0-fx/dfx

    if abs(f(x1)) < tolerance :
         print(f"Root found at x={x1:.6f}")
```

```
            break
        else:
            x0=x1

    print(f"NEWTON-RAPHSON:{timeit.default_timer()-starttime:.6f} milliseconds")
```

Results

The Root obtained is : x=−0.926857 and it made 9 with a time span of 6.784470 *milliseconds*

Conclusion

It is seen that the Newton Raphson Method is faster as compared to Bisect Method and does fewer iterations