Newtons Raphson Method for Finding Roots

In order to determine the function's root within a given tolerance and the maximum number of iterations, this function uses the Newton-Raphson approach. Initial guess x0, tolerance, and maximum iterations are provided as inputs. If the root cannot be located within the provided number of iterations, the function emits an error message instead of the root when it is discovered. Moreover, the function needs the derivative, which is represented by the symbol df (x).

Consider the function $f(x) = 2x^3 - 3^x - 12x$ and its derivative $f'(x) = 6x^2 - 6x - 12$.

Let $x_0 = 1$ be the initial guess for the root of $f(x)$.

The Newton-Raphson method is used to find the root of the function. The algorithm iteratively updates the value of x_0 using the following equation:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

If the absolute value of $f(x_1)$ is less than the tolerance level, then the algorithm terminates and the root is found.

Otherwise, the value of x_0 is updated to x_1 and the process continues.

Python code's implementation of Newton-Raphson method

```
import timeit

def f(x):
    return 2*x**3 - 3**x - 12*x

def df(x):
    return 6*x**2 - 6*x-12

x0 = 1
tolerance = 1e-6
max_iteration = 100

for i in range(max_iteration):
    fx = f(x0)
    dfx = df(x0)

    x1 = x0 - (fx / dfx)
    print("Iterations " + str(i) + ": x = " + str(x0) + " f(x) = " +
          str(f(x0)))

    if abs(f(x1)) < tolerance:
        print(f"Root found at x = {x1:.2f}")
        print("Time taken is ", timeit.timeit())
        break
```

```
    else:
        x0 = x1
```

The output of the Python code is printed to the console

This includes the iteration number, the root found, and the time taken to find the root.

Bisection Method for Finding Roots

Within a given tolerance and maximum number of iterations, this function employs the bisection method to locate the function's root. The root's location is in the range $[a, b]$. The tolerance and the maximum number of iterations are specified in the function. We start by defining a function $f(x)$ and its derivative $f'(x)$, as well as an initial guess for the root:

Consider a function $f(x)$ that is continuous on the interval $[a, b]$ and let $f(a)$ and $f(b)$ have opposite signs, i.e., $f(a) \cdot f(b) < 0$. Then, by the Intermediate Value Theorem, there exists at least one root of $f(x)$ in the interval $[a, b]$.

The bisection method is a simple numerical algorithm for finding this root. It works by repeatedly dividing the interval $[a, b]$ in half and testing which half contains the root, until a sufficiently accurate estimate of the root is found. Specifically, the bisection method involves the following iterative equation for updating the guesses for the root:

$$c_k = \frac{a_k + b_k}{2}$$

where c_k is the midpoint of the interval $[a_k, b_k]$. At each iteration k, the function $f(x)$ is evaluated at c_k, and the sign of $f(c_k)$ is used to determine which half of the interval contains the root. The interval is then updated accordingly, and the process is repeated until the desired level of accuracy is achieved.

Here's an implementation of the bisection method in Python:

```
def f(x):
    return 2*x**3 - 3**x - 12*x

a = 2
b = -1
tolerance = 1e-6
max_iterations = 100

for i in range(max_iterations):
    c = (a + b) / 2
    print(f"Iteration number is: {i}")

    if abs(f(c)) < tolerance:
        print(f"Root found at x = {c:6f}")
        print("Time taken is: ", timeit.timeit())
```

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

In this code, we define the function $f(x)$ and the initial guesses for the root, a and b. We also set the tolerance and maximum number of iterations. The main loop then implements the bisection method by updating the interval $[a, b]$ using the midpoint c, and testing the sign of $f(c)$ to determine which half of the interval contains the root. The process continues until a root within the specified tolerance is found, or the maximum number of iterations is reached.