

**UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**(BE-CSE/IT-7th Sem)**



**Subject Name:** NUMERICAL METHODS AND OPTIMIZATION USING PYTHON

**Subject Code:** 21CSH-459

**Submitted to: Submitted by:**

Er. Kanika Rana (E16532) Name: Vinita Maddheshiya

UID: 21BCS10540

Section: 21BCS\_11

Group: A

**Topic:** *Performance Analysis of Root-Finding Algorithms in Python*

***Introduction:***

Root finding algorithm is a computational method used to determine the roots of a mathematical function. The root of a function is the value of x that makes the function equal to zero, i.e., f(x) = 0. These algorithms are essential in various fields of science and engineering because they help solve equations that cannot be easily rearranged or solved analytically. Examples of root-finding algorithms include the Bisection Method, Newton-Raphson Method, and Secant Method.

***Procedure and Code:***

1. **Bisection Method:**

The Bisection Method is a straightforward root-finding technique that works by repeatedly halving the interval in which the root lies. The method assumes that the function changes sign over the interval [a,b][a, b][a,b], which implies that a root exists between aaa and bbb. At each iteration, the interval is halved by computing the midpoint, and the function's value at the midpoint is used to determine the new interval. The process continues until the interval is sufficiently small, meaning the root has been approximated to a desired tolerance.

***Code:***

def bisection\_method(f, a, b, tol=1e-6):

if f(a) \* f(b) >= 0:

raise ValueError("The function must have different signs at a and b.")

while (b - a) / 2 > tol:

midpoint = (a + b) / 2.0

if f(midpoint) == 0:

return midpoint

elif f(a) \* f(midpoint) < 0:

b = midpoint

else:

a = midpoint

return (a + b) / 2.0

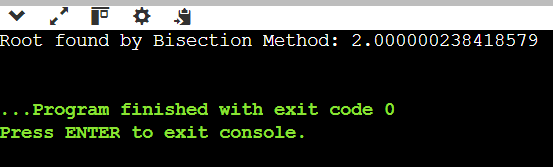
# Example usage:

f = lambda x: x\*\*2 - 4

root = bisection\_method(f, 0, 3)

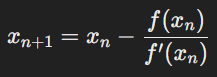
print(f"Root found by Bisection Method: {root}")

***Output:***

****

1. **Newton-Raphson Method:**

The Newton-Raphson Method is an iterative root-finding algorithm that uses the function's derivative to find successively better approximations to the root. Starting from an initial guess x0x\_0x0​, the method uses the formula:



This process is repeated until the difference between consecutive approximations is less than a specified tolerance. The Newton-Raphson method is known for its fast convergence when the initial guess is close to the actual root. However, it requires the function to be differentiable and can fail or converge slowly if the initial guess is poor or if the derivative is zero at any iteration.

***Code:***

def newton\_raphson\_method(f, df, x0, tol=1e-6, max\_iter=1000):

x = x0

for i in range(max\_iter):

fx = f(x)

dfx = df(x)

if abs(fx) < tol:

return x

x = x - fx / dfx

raise ValueError("Maximum iterations exceeded without finding root.")

# Example usage:

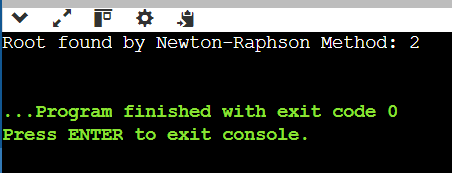
f = lambda x: x\*\*2 - 4

df = lambda x: 2 \* x

root = newton\_raphson\_method(f, df, 2)

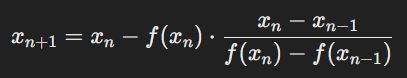
print(f"Root found by Newton-Raphson Method: {root}")

***Output:***

****

1. **Secant Method:**

The Secant Method is similar to the Newton-Raphson Method but does not require the computation of the derivative. Instead, it approximates the derivative using two previous points. The update formula is:



This method is particularly useful when the derivative of the function is difficult or impossible to calculate. The Secant Method generally converges faster than the Bisection Method but slower than the Newton-Raphson Method.

***Code:***

def secant\_method(f, x0, x1, tol=1e-6, max\_iter=1000):

for i in range(max\_iter):

f\_x0 = f(x0)

f\_x1 = f(x1)

if abs(f\_x1) < tol:

return x1

denominator = f\_x1 - f\_x0

if denominator == 0:

raise ValueError("Zero denominator encountered in Secant Method.")

x2 = x1 - f\_x1 \* (x1 - x0) / denominator

x0, x1 = x1, x2

raise ValueError("Maximum iterations exceeded without finding root.")

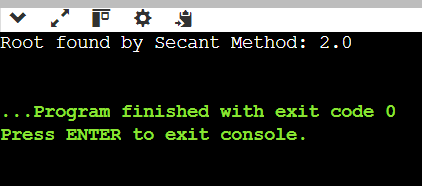
# Example usage:

f = lambda x: x\*\*2 - 4

root = secant\_method(f, 2, 3)

print(f"Root found by Secant Method: {root}")

***Output:***

****

Performance analysis of all the mentioned three algorithms are:

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Bisection Method | Newton-Raphson Method | Secant Method |
| Convergence Speed | Slow | Fast (Quadratic convergence) | Faster than Bisection, slower than Newton |
| Order of Convergence | Linear (order 1) | Quadratic (order 2) | Superlinear (≈1.618) |
| Initial Guess Requirement | Two initial guesses | One initial guess and its derivative | Two initial guesses |
| Robustness | Very robust, always converges | Can fail if the derivative is zero or changes sign | Less robust, may fail if initial guesses are not close |
| Function Requirements | Only the function needs to be continuous | Function must be differentiable | Function needs to be continuous, but derivative not required |
| Error Bound | Error reduces by half each iteration | Error reduces quadratically | Error bound not well-defined |
| Computational Cost per Iteration | Low, requires two function evaluations | High, requires one function and one derivative evaluation | Moderate, requires two function evaluations |
| Ease of Implementation | Easy to implement | More complex due to derivative calculation | Moderate, simpler than Newton but more complex than Bisection |

***Conclusion:***

The analysis shows that each root-finding algorithm has its strengths. The Bisection Method is robust and easy to implement but converges slowly. The Newton-Raphson Method converges quickly but requires the function's derivative, making it more complex. The Secant Method, which doesn't need the derivative, offers faster convergence than Bisection but is less reliable than Newton-Raphson. The choice of method depends on the problem's specific needs, such as speed, complexity, and the availability of derivatives.