Numerical Analysis Series

Newton-Raphson Method

(Theory, Derivation & Code)

Author
Shahaduddin
shahaduddin.com

Date
February 2, 2026
Version 1.0

1. Theoretical Background

Introduction

The **Newton-Raphson method** is one of the most powerful and well-known numerical methods for solving a root-finding problem $f(x) = 0$.
Unlike the Bisection method which uses interval halving, Newton's method uses the **derivative** $f'(x)$ to find the root. It converges much faster (Quadratically) provided the initial guess is close to the true root.

Derivation from Taylor Series

Suppose that $f \in C^2[a, b]$ and let p_0 be an approximation to the root p. Consider the first Taylor polynomial expanded about p_0:

$$f(p) = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2}f''(\xi) \tag{1}$$

Since p is a root, $f(p) = 0$. Assuming $|p - p_0|$ is small, we can neglect the second-order term:

$$0 \approx f(p_0) + (p - p_0)f'(p_0) \tag{2}$$

Solving for p:

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \tag{3}$$

This gives us the iterative formula for generating the sequence $\{p_n\}$:

$$\boxed{p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{for } n \geq 1} \tag{4}$$

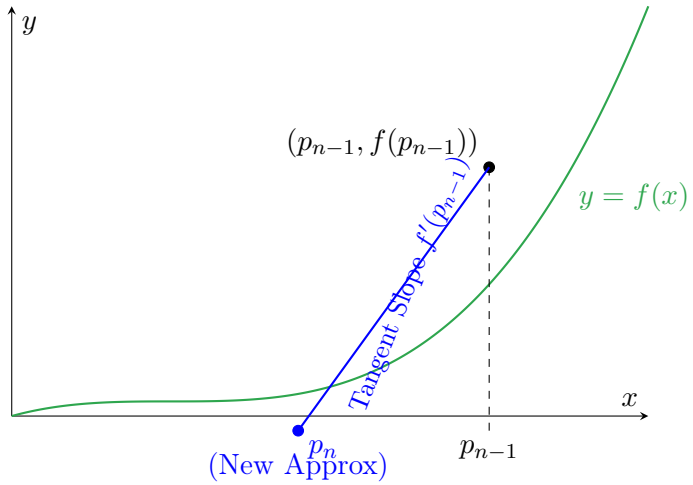

Fig 1. Geometric Interpretation: p_n is the x-intercept of the tangent line at p_{n-1}.

2. Algorithm Convergence

Newton's Method Algorithm

To find a solution to $f(x) = 0$ given an initial approximation p_0:

1. **Input:** initial approximation p_0; tolerance TOL; max iterations N_0.
2. Set $i = 1$.
3. While $i \leq N_0$ do:
   - Step 3: Set $p = p_0 - f(p_0)/f'(p_0)$.
   - Step 4: If $|p - p_0| < TOL$, then **OUTPUT** p; **STOP**.
   - Step 5: Set $i = i + 1$.
   - Step 6: Set $p_0 = p$ (Update).
4. **Output:** Method failed after N_0 iterations.

Convergence Note: Newton's method is a functional iteration technique $p_n = g(p_{n-1})$ where:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

If $f \in C^2[a, b]$, $f(p) = 0$ and $f'(p) \neq 0$, then Newton's method exhibits **Quadratic Convergence**. This means the number of correct decimal places roughly doubles with each iteration.

3. Code Implementations

Solving for root of $x^2 - 4 = 0$ (Root $= 2.0$).

>PythonImplementation (*.py*)

```python
def newton_raphson(f, df, x0, tol=1e-6, max_iter=100):
    """
    Newton-Raphson Method.
    Quadratic convergence for smooth functions.
    """
    x = x0
    for i in range(max_iter):
        fx = f(x)
        dfx = df(x)

        # Prevent division by zero
        if dfx == 0:
            raise ValueError("Derivative is zero. No solution found.")

        x_new = x - fx / dfx

```

```
        # Check convergence
        if abs(x_new - x) < tol:
            return x_new

        x = x_new

    return x

# Example Usage
f = lambda x: x**2 - 4
df = lambda x: 2*x
root = newton_raphson(f, df, 10)
print(f"Root: {root}")
```

>FortranImplementation *(.f90)*

```fortran
program newton_raphson
    implicit none
    real :: x, x_new, tol
    integer :: i

    x = 10.0
    tol = 1e-6

    do i = 1, 100
        if (df(x) == 0.0) stop "Derivative zero"

        x_new = x - f(x) / df(x)

        if (abs(x_new - x) < tol) exit

        x = x_new
    end do

    print *, "Root: ", x_new

contains
    real function f(v)
        real, intent(in) :: v
        f = v**2 - 4.0
    end function f

    real function df(v)
        real, intent(in) :: v
        df = 2.0 * v
    end function df
end program newton_raphson
```

>C++Implementation *(.cpp)*

```cpp
#include <iostream>
#include <cmath>

double newton(double (*f)(double), double (*df)(double), double x0, double
    tol = 1e-6) {
    double x = x0;
    for(int i = 0; i < 100; ++i) {
        double fx = f(x);
        double dfx = df(x);

        if(std::abs(dfx) < 1e-12) break; // Avoid division by zero

        double x_new = x - fx / dfx;

        if(std::abs(x_new - x) < tol) return x_new;

        x = x_new;
    }
    return x;
}

```

```cpp
// Function definitions for example
double f(double x) { return x*x - 4; }
double df(double x) { return 2*x; }

int main() {
    std::cout << "Root: " << newton(f, df, 10.0) << std::endl;
    return 0;
}
```

For more numerical analysis resources, visit shahaduddin.com/PyNum