Netrokona University

Department of Computer Science and Engineering

Laboratory Report - 03

Newton-Raphson Method Implementation

Course: CSE-3212 (Numerical Methods Lab)

Submitted By:

Name: Eyasir Ahamed
Class Roll: 15
Exam Roll: 413
Reg. No: 202004017

Submitted To:

Dr. A F M Shahab Uddin
Assistant Professor
Dept. of CSE
Jashore University of Science & Technology

July 4, 2025

Contents

1 Introduction

The Newton–Raphson method is a root-finding technique that uses tangent-line approximations to rapidly converge on a solution. It typically exhibits quadratic convergence when conditions are met.

2 Theory

2.1 Formula and Derivation

Starting with the Taylor series approximation at x_n:

$$0 \approx f(x_n) + (x_{n+1} - x_n)f'(x_n) \Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

2.2 Convergence Requirements

- **Derivative non-zero:** Requires $f'(x_n) \neq 0$ at each iteration to avoid division by zero and ensure a valid update step.
- **Quadratic convergence:** Exhibits quadratic convergence—meaning the error roughly squares each step—when the initial guess x_0 is sufficiently close to a simple root.

3 Algorithm Design

Algorithm 1 Newton-Raphson Method

Require: Initial guess x_0, function f, derivative f', tolerance ϵ, maximum iterations N
Ensure: Approximate root x or failure message

for $n \leftarrow 0$ to $N - 1$ **do**
  Compute $f_n \leftarrow f(x_n)$ and $f'_n \leftarrow f'(x_n)$
  Update $x_{n+1} \leftarrow x_n - \frac{f_n}{f'_n}$
  Calculate error: $err \leftarrow \left|\frac{x_{n+1} - x_n}{x_{n+1}}\right|$
  if $err < \epsilon$ or $|f(x_{n+1})| < \epsilon$ **then**
    return x_{n+1}
  end if
end for
return "Method did not converge within the maximum iterations"

4 Worked Example

4.1 Problem Definition

$$f(x) = x^3 - 2x^2 + x - 3, \quad f'(x) = 3x^2 - 4x + 1$$

Use initial guess $x_0 = 2$, tolerance $\epsilon = 10^{-6}$, and maximum 20 iterations.

5 C++ Implementation

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

double f(double x) { return x * x * x - 2 * x * x + x - 3; }
double df(double x) { return 3 * x * x - 4 * x + 1; }

int main() {
    double x = 2.0, tol = 1e-6, err;
    int maxIter = 20;
    cout << fixed << setprecision(7);
    cout << "Iter  x_n        f(x_n)        x_{n+1}     Error\n";
    for (int i = 1; i <= maxIter; ++i) {
        double fx = f(x), dfx = df(x);
        if (fabs(dfx) < 1e-12) {
            cout << "Zero derivative. Stop.\n";
            return 1;
        }
        double x1 = x - fx / dfx;
        err = fabs((x1 - x) / x1);
        cout << i << "      " << x << "   " << fx << "   " << x1 << "   " << err << "\n";
        if (err < tol) {
            cout << "\nConverged to " << x1 << " in " << i << " iterations.\n";
            return 0;
        }
        x = x1;
    }
    cout << "Did not converge within " << maxIter << " iterations.\n";
    return 1;
}
```

Listing 1: Newton–Raphson for $x^3 - 2x^2 + x - 3$

6 Results and Analysis

6.1 Execution Output

7 Discussion

The method converged in just 4 iterations—showing the expected **quadratic convergence**. The derivative stayed well-behaved throughout, avoiding numerical pitfalls.

```
Iter   x_n         f(x_n)          x_{n+1}       Error
1      2.0000000    -1.0000000    2.2000000     0.0909091
2      2.2000000   0.1680000    2.1750000    0.0114943
3      2.1750000   0.0028594    2.1745595    0.0002025
4      2.1745595   0.0000009    2.1745594    0.0000001

Converged to 2.1745594 in 4 iterations.
```

Figure: Program Output

8 Conclusion

Newton–Raphson effectively solved the equation $x^3 - 2x^2 + x - 3 = 0$, finding the root $x \approx 2.1745595$ in four iterations. Its speed makes it ideal for smooth functions with accessible derivatives, though care is needed with initial guesses and potential flat derivatives.