Introduction:

In numerical analysis, solving nonlinear equations is a fundamental requirement in various scientific and engineering applications. Many nonlinear equations cannot be solved analytically, bringing the use of numerical techniques. Alongside bracketing methods such as the Bisection and False Position techniques, another major class of algorithms called Open Methods exists.

Open Methods do not require the initial interval to bracket a root . Instead, they rely on interpolation or derivative-based extrapolation to iteratively approach the root. These methods often converge faster than bracketing techniques but may fail if the initial guesses are not chosen properly.

There are two types of open methods:

1.Newton-Raphson
2.MethodSecant Method

Newton-Raphson Method:

The Newton-Raphson method uses the tangent line at the current approximation to estimate a better root approximation.Given a guess (xn), the tangent line at that point is:

$[y = f(xn) + f'(xn)(x - xn)]$

Setting $(y = 0)$: $[0 = f(xn) + f'(xn)(x - xn)]$

Solving for (x): $[x(n+1) = xn - f(xn)/f'(xn)]$

This is the Newton-Raphson formula.

Algorithm of Newton-Raphson Method:

Step 1: Choosing an initial guess x0.
Step 2: Computing derivative f'(xn).
Step 3: Computing next approximation:
$[x(n+1)= xn -f(xn)/f'(xn)]$

Step 4: Checking stopping criterion:

$|x(n+1)-xn| < E$

If satisfied => Stop.
Else => setting (xn = x(n+1)) and repeating.

Pseudocode:

```
NewtonRaphson(f, df, x0, tolerance, max_iter):
    for i = 1 to max_iter:
        fx = f(x0)
        dfx = df(x0)
        if dfx == 0:
            break
        x1 = x0 - fx/dfx
        if abs(x1 - x0) < tolerance:
            return x1
        x0 = x1
    return x1
```

Worked Example:

Find the root of [f(x) = 3x - cos x - 1] near x = 0 using Newton-Raphson Method.

Derivative:[f'(x) = 3 + sin x]

Iteration Table :

Iteration	(xn)	f(xn)	f'(xn)	x(n+1)=xn - f(xn)/f'(xn)
0	0.00	-2.0000	3.0000	0.6667
1	0.6667	0.2307	3.6184	0.6020
2	0.6020	0.0105	3.5667	0.5991
3	0.5991	-0.00001	3.5643	0.5991

Approximate root: 0.5991

Importance:

Extremely fast convergence.Widely used in engineering and computational systems.Efficient for large-scale problems.

Limitations:

Fails if derivative becomes zero.Diverges if starting guess is too far.Requires derivative $f'(x)$, which may be complicated.

Applications:

1. Machine learning optimization
2. Electrical circuit analysis
3. Mechanical system design
4. Robotics control
5. Computational physics

Secant Method:

The Secant Method approximates the derivative numerically using two points instead of requiring an analytical derivative.Given two initial points $x(n-1)$ and xn :

$$f'(xn) = (f(xn) - f(x(n-1)))/(xn - x(n-1))$$

Substituting this into Newton's formula:

$$[x(n+1) = xn - f(xn) * (xn - x(n-1)) / (f(xn) - f(x(n-1)))]$$

Algorithm of Secant Method:

Step 1: Choosing two initial approximations x0 and x1.
Step 2: Applying:
$[x(n+1) = xn - f(xn) * (xn - x(n-1)) / (f(xn) - f(x(n-1)))]$
Step 3: Checking:
$| x(n+1) - xn | < E$

If yes ->Stop.
Else -> set $x(n-1) = xn$, $xn = x(n+1)$ and repeat.

Pseudocode:

```
SecantMethod(f, x0, x1, tolerance, max_iter):
    for i = 1 to max_iter:
        f0 = f(x0)
        f1 = f(x1)
        x2 = x1 - f1*(x1 - x0)/(f1 - f0)
        if abs(x2 - x1) < tolerance:
            return x2
        x0 = x1
        x1 = x2
    return x2
```

Worked Example:

Find root of[f(x) = 3x - cos x - 1] Choosing:
(x0 = 0)
(x1 = 1)

Iteration Table

Iteration	x(n-1)	xn	x(n+1)	f(x(n+1))
1	0.000	1.000	0.3333	-0.667
2	1.000	0.3333	0.3668	-0.534
3	0.3333	0.3668	0.3679	0.0001

Approximate Root: 0.3679

Importance

Faster than Bisection/False Position.Does not require derivative.Useful when derivative computation is expensive.

Limitations:

May diverge if guesses are poor.Requires two initial approximations.Slower than Newton-Raphson.

Applications

1. Nonlinear dynamics
2. Heat transfer equations
3. Circuit simulations
4. Engineering optimization