du_logo.png

University of Dhaka

Department of Computer Science and Engineering

Implementation and Comparative Analysis of the Bisection, False Position, Newton–Raphson, and Secant Methods for Finding Roots of Nonlinear Equations

Course: CSE-3212

Submitted By:

Ashesh Bar
Roll: 18
Session: 2021–22

1 Introduction

Many engineering and scientific problems involve equations that cannot be solved analytically, requiring numerical techniques to find approximate solutions. Root-finding methods are iterative numerical algorithms used to find the real roots of nonlinear equations of the form $f(x) = 0$.

Among the widely used approaches are:

- **Bisection Method:** A bracketing method that repeatedly divides an interval in half and selects a subinterval where the sign of the function changes. It is simple and always convergent if the initial interval is valid, but converges slowly.
- **False Position (Regula Falsi) Method:** Another bracketing method that uses a linear interpolation to estimate the root. It generally converges faster than Bisection but can stagnate in some cases.
- **Newton–Raphson Method:** An open method that uses the derivative of the function to predict the next approximation. It offers quadratic convergence near the root but requires a good initial guess and a differentiable function.
- **Secant Method:** A derivative-free alternative to Newton–Raphson, using two prior approximations to estimate the slope. It usually converges faster than bracketing methods and is less computationally expensive than Newton–Raphson.

These methods are fundamental in numerical analysis, forming the basis for solving nonlinear systems and optimization problems in computational mathematics, physics, and engineering.

2 Objectives

The primary objectives of this experiment are:

1. To implement four numerical root-finding methods — Bisection, False Position, Newton–Raphson, and Secant — using Python.
2. To determine the root of the nonlinear equation:

$$f(h) = h^3 - 10h + 5e^{-h/2} - 2 = 0$$

3. To achieve a convergence criterion of approximate relative error $e_a \leq 0.001\%$.
4. To compare the convergence behavior and accuracy of all four methods using a graphical representation of error versus iteration.

5. To analyze the efficiency and convergence rate of each method, identifying their advantages and limitations.

3 Algorithms

3.1 Bisection Method

1. Choose an interval $[a, b]$ such that $f(a) \cdot f(b) < 0$.
2. Compute midpoint $c = \frac{a+b}{2}$.
3. Evaluate $f(c)$.
4. If $f(a) \cdot f(c) < 0$, set $b = c$; else, set $a = c$.
5. Repeat until $e_a \leq 0.001\%$.

3.2 False Position Method

1. Choose an interval $[a, b]$ such that $f(a) \cdot f(b) < 0$.
2. Compute
$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$
3. Evaluate $f(c)$.
4. If $f(a) \cdot f(c) < 0$, set $b = c$; else, set $a = c$.
5. Repeat until $|f(c)|$ or $|b - a|$ is less than the tolerance.

3.3 Newton–Raphson Method

1. Choose an initial guess x_0.
2. Update the root approximation using
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$
3. Repeat until $|x_{i+1} - x_i|$ or $|f(x_{i+1})|$ is less than the tolerance.

3.4 Secant Method

1. Choose two initial guesses x_0 and x_1.

2. Update the root approximation using

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

3. Repeat until $|x_{i+1} - x_i| \leq$ tolerance.

4 Implementation

4.1 Bisection Method

import math import pandas as pd import matplotlib.pyplot as plt

def f(h): return h**3 - 10*h + 5*math.exp(-h/2) - 2

Bisection Method def bisection$_verbose(a, b, tol = 0.001, N = 200, round_n = 5) : Checkifrootisguaranteediff(a)*f(b) >= 0 : print(f"Norootguaranteedintheinterval[a, b].")retu$

rows = [] ea$_plot = []print("Iter|xl|xu|xr|f(xr)|ea(print("---")$

for k in range(1, N + 1): midpoint xr = round((a + b) / 2, round$_n)fxr = round(f(xr), round_n)$

Absolute error ea = abs(b - a) * 100 if k ¿ 1 else None

Print current iteration if ea is not None: print(f'k:4d — a:8.5f — b:8.5f — xr:9.5f — fxr:11.5f — ea:8.5f') else: print(f'k:4d — a:8.5f — b:8.5f — xr:9.5f — fxr:11.5f — ' —-'")

Store for table and plotting rows.append("iter": k, "xl": round(a, round$_n), "xu" : round(b, round_n), "xr" : xr, "f(xr)" : fxr, "ea()ifeaisnotNone : ea_plot.append((k, ea))$

check convergence if fxr == 0 or abs(b - a) ¡ tol: print("———————————————————-") print(f'Approximate root after k iterations: xr = xr:.5f') df = pd.DataFrame(rows) return df, ea$_plot, xr$

update interval if f(a) * f(xr) ¡ 0: b = xr else: a = xr

reached max iterations print("———————————————————-") print(f'Stopped after N iterations. Approximate root xr:.5f') df = pd.DataFrame(rows) return df, ea$_plot, xr$

Run the Bisection Method a, b = 0.1, 0.4 tol = 0.001 N = 50

df$_bi, ea_plot_bi, root_bi = bisection_verbose(a, b, tol, N, round_n = 5)$

Save iteration table df$_bi.to_csv("bisection_iterations.csv", index = False)$

Plot Error vs Iteration if len(ea$_plot_bi) > 0 : its = [itforit, einea_plot_bi]eas = [eforit, einea_plot_bi]$

Linear scale plt.figure(figsize=(7,4.5)) plt.plot(its, eas, marker='o') plt.xlabel("Iteration") plt.ylabel("Absolute Error ea (plt.title("Bisection Method: ea(plt.grid(True) plt.savefig("bisection$_ea_linear.png")plt.show()$

Final Output Summary print("Summary:") print(f"Approximate root = root$_bi : .5f")print(f"Iterationdatasavedin : bisection_iterations.csv")print("Errorplotssavedas : bisection_ea_linear.png")$

4.2 False Position Method

import math, pandas as pd, matplotlib.pyplot as plt

def f(h): return h**3 - 10*h + 5*math.exp(-h/2) - 2

def false$_position_verbose(a_init, b_init, tol_percent = 0.001, round_n = 5, max_iter =$ 500) : $a = round(a_init, round_n)b = round(b_init, round_n)fa = round(f(a), round_n)fb = round(f(b), round_n)iffa*fb > 0 : raiseValueError(f"Initialinterval[a, b]doesnotbracketaroot.")$

rows = [] prev$_c = Noneea_plot = []final = None$

for itr in range(1, max$_iter + 1) : denom = (fb - fa)ifdenom == 0 : raiseZeroDivisionError("Denominatorzeroinfalsepositionformula.")c_raw = (a * fb - b * fa)/denomc = round(c_raw, round_n)fc = round(f(c), round_n)$

if prev$_cisNone : ea = Noneelse : ea = abs((c - prev_c)/c) * 100.0ifc! = 0elsefloat('inf')$

rows.append("iter": itr, "xl": f"a:.5f", "xu": f"b:.5f", "xr": f"c:.5f", "f(xr)": f"fc:.5f", "ea() if ea is not None: ea$_plot.append((itr, ea))$

if fc == 0 or (ea is not None and ea ¡= tol$_percent) : final = "iterations" : itr, "root" : c, "f_root" : fc,$

if fa * fc ¡ 0: b, fb = c, fc else: a, fa = c, fc

a = round(a, round$_n)b = round(b, round_n)prev_c = celse : final = "iterations" : max_iter, "root" : c, "f_root" : fc, "final_ea" : (ea_plot[-1][1]ifea_plotelseNone)$

df = pd.DataFrame(rows) return df, ea$_plot, final$

Run df$_{fp}, ea_plot_{fp}, final_{fp} = false_position_verbose(0.1, 0.4, tol_percent = 0.001, round_n = 5, max_iter = 500)df_{fp}.to_csv("false_position_iterations.csv", index = False)$

Print table print(df$_{fp}.to_string(index = False))$

Plot (linear) if ea$_plot_{fp} : its = [itforit, eineaplot_{fp}]; eas = [eforit, eineaplot_{fp}]plt.figure(figsize = (7, 4.5))plt.plot(its, eas, marker =' o')plt.xlabel("Iteration"); plt.ylabel("Approx.relativeerrorea(plt.ti$ $ea(plt.savefig("falsepos_ea_linear.png"); plt.show()$

4.3 Newton–Raphson Method

import math import pandas as pd import matplotlib.pyplot as plt

Define the function and its derivative def f(h): return h**3 - 10*h + 5*math.exp(-h/2) - 2

def f$_prime(h) : return3 * h * *2 - 10 - (5/2) * math.exp(-h/2)$

def newton$_raphson_verbose(x0, tol_percent = 0.001, round_n = 5, max_iter =$ 200) : $xi = round(x0, round_n)rows = []ea_plot = []prev_x = Nonefinal = None$

for itr in range(1, max$_iter + 1) : fxi = round(f(xi), round_n)fpxi = round(f_prime(xi), round_n)$

if fpxi == 0: raise ZeroDivisionError(f"Derivative zero at iteration itr, x=xi")

Newton-Raphson formula x$_next_raw = xi - fxi/fpxix_next = round(x_next_raw, round_n)fx_next = round(f(x_next), round_n)$

if prev$_x isNone: ea = Noneelse: ea = abs((x_next - prev_x)/x_next) * 100 if x_next != 0 else float('inf')$

rows.append("iter": itr, "xi": f'xi:.5f', "f(xi)": f'fxi:.5f', "f'(xi)": f'fpxi:.5f', "xi+1": f'x$_next : .5f", "f(xi+1)" : f" fx_next : .5f", "ea()$

if ea is not None: ea$_plot.append((itr, ea))$

if fx$_next == 0 or (ea is not None and ea <= tol_percent) : final = "iterations" : itr, "root" : x_next, "f_r$

prev$_x = x_next xi = x_next$

else: final = "iterations": max$_iter, "root" : x_next, "f_root" : fx_next, "final_ea" : (ea_plot[-1][1] if ea_plot else None)$

df$_table = pd.DataFrame(rows) return df_table, ea_plot, final$

df$_nr, ea_plot_nr, final_nr = newton_raphson_verbose(x0 = 1.5, tol_percent = 0.001, round_n = 5, max_iter = 200)$

df$_nr.to_csv("newton_raphson_iterations.csv", index = False)$

print(df$_nr.to_string(index = False))$

Plot Error vs Iteration if len(ea$_plot_nr) > 0 : its = [it for it, e in ea_plot_nr] eas = [e for it, e in ea_plot_nr]$

Linear scale plt.figure(figsize=(7,4.5)) plt.plot(its, eas, marker='o') plt.xlabel("Iteration") plt.ylabel("Approx. relative error ea (plt.title("Newton-Raphson: ea(plt.grid(True) plt.savefig("newton$_ea_linear.png") plt.show()$

print("Summary:") print(f'Converged in final$_nr['iterations'] iterations.") print(f" Approximateroot$ $final_nr['root'] : .5f") print(f" f(root) = final_nr['f_root'] : .5f") if final_nr['final_ea'] is not None :$ $print(f" Final approximate relative error ea(else : print(" Final approximate relative error ea($

4.4 Secant Method

import math import pandas as pd import matplotlib.pyplot as plt

Define the function def f(h): return h**3 - 10*h + 5*math.exp(-h/2) - 2

def secant$_verbose(x0, x1, tol_percent = 0.001, round_n = 5, max_iter = 200) : x_prev = round(x0, round_n) x_curr = round(x1, round_n) rows = [] ea_plot = [] final = None$

for itr in range(1, max$_iter+1) : f_prev = round(f(x_prev), round_n) f_curr = round(f(x_curr), round_n)$

Avoid division by zero if f$_curr - f_prev == 0 : raise ZeroDivisionError(f" Division by zero at iteratio$

Secant formula x$_next_raw = x_curr - f_curr * (x_curr - x_prev)/(f_curr - f_prev) x_next = round(x_next_raw, round_n) f_next = round(f(x_next), round_n)$

Approximate relative error if itr == 1: ea = None else: ea = abs((x$_next - x_curr)/x_next) * 100 if x_next != 0 else float('inf')$

rows.append("iter": itr, "x(i-1)": f'x$_prev : .5f", "x(i)" : f" x_curr : .5f", "f(x(i-1))" : f" f_prev : .5f", "f(x(i))" : f" f_curr : .5f", "x(i+1)" : f" x_next : .5f", "f(x(i+1))" : f" f_next : .5f", "ea()$

if ea is not None: ea$_plot.append((itr, ea))$

Stop condition if f$_next == 0 or (ea is not None and ea <= tol_percent) : final = "iterations" : itr, "root" : x_next, "f_root" : f_next, "final_ea" : ea break$

Update for next iteration x$_prev, x_curr = x_curr, x_next$

else: final = "iterations": max$_iter, "root" : x_next, "f_root" : f_next, "final_ea" : (ea_plot[-1][1] if ea_plot else None)$

df$_table = pd.DataFrame(rows) return df_table, ea_plot, final$

df$_sc, ea_plot_sc, final_sc = secant_verbose(x0 = 1.5, x1 = 2.0, tol_percent = 0.001, round_n = 5, max_iter = 200)$

df$_sc.to_csv("secant_iterations.csv", index = False)$

print(df$_sc.to_string(index = False))$

Plot Error vs Iteration if len(ea$_plot_sc) > 0 : its = [itforit, eineaplot_sc]eas = [eforit, eineaplot_sc]$

Linear scale plt.figure(figsize=(7,4.5)) plt.plot(its, eas, marker='o') plt.xlabel("Iteration") plt.ylabel("Approx. relative error ea (plt.title("Secant Method: ea(plt.grid(True) plt.savefig("secant$_ea_linear.png")plt.show()$

print("Summary:") print(f"Converged in final$_sc['iterations']iterations.")print(f"Approximateroot final_sc['root'] : .5f")print(f"f(root) = final_sc['f_root'] : .5f")iffinal_sc['final_ea']isnotNone : print(f"Finalapproximaterelativeerrorea(else : print("Finalapproximaterelativeerrorea($

4.5 Combined Convergence Plot

import math import matplotlib.pyplot as plt

Define the function and derivative def f(x): return x**3 - 10*x + 5*math.exp(-x/2) - 2

def f$_prime(x) : return3 * x ** 2 - 10 - (5/2) * math.exp(-x/2)$

1 Bisection Method def bisection(a, b, tol=0.001, max$_iter = 100) : ea_list = []iff(a) * f(b) >= 0 : returnea_listforiinrange(1, max_iter + 1) : c = (a+b)/2ea = abs(b-a)*100ifi > 1elseNoneifeaisnotNone : ea_list.append((i, ea))iff(c) == 0orabs(b - a) < tol : breakiff(a) * f(c) < 0 : b = celse : a = creturnea_list$

2 False Position Method def false$_position(a, b, tol = 0.001, max_iter = 100) : ea_list = []fa, fb = f(a), f(b)iffa * fb >= 0 : returnea_listprev_c = Noneforiinrange(1, max_iter + 1) : c = (a * fb - b * fa)/(fb - fa)fc = f(c)ifprev_cisnotNone : ea = abs((c-prev_c)/c)*100ea_list.append((i, ea))ifea <= tol : breakiffa*fc < 0 : b, fb = c, fcelse : a, fa = c, fcprev_c = creturnea_list$

3 Newton–Raphson Method def newton$_raphson(x0, tol = 0.001, max_iter = 100) : ea_list = []x = x0foriinrange(1, max_iter+1) : fx, dfx = f(x), f_prime(x)ifdfx == 0 : breakx_new = x-fx/dfxea = abs((x_new-x)/x_new)*100ifi > 1elseNoneifeaisnotNone : ea_list.append((i, ea))ifea <= tol : breakx = x_newreturnea_list$

4 Secant Method def secant(x0, x1, tol=0.001, max$_iter = 100) : ea_list = []prev, curr = x0, x1foriinrange(1, max_iter+1) : f_prev, f_curr = f(prev), f(curr)iff_curr-f_prev == 0 : breaknext_x = curr - f_curr * (curr - prev)/(f_curr - f_prev)ea = abs((next_x-curr)/next_x)*100ifi > 1elseNoneifeaisnotNone : ea_list.append((i, ea))ifea <= tol : breakprev, curr = curr, next_xreturnea_list$

bisection$_errors = bisection(0.1, 0.4)falsepos_errors = false_position(0.1, 0.4)newton_errors = newton_raphson(1.5)secant_errors = secant(1.5, 2.0)$

Plot Comparison (Linear Scale) plt.figure(figsize=(8,6))

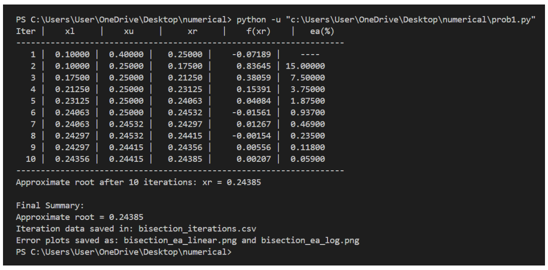
plt.plot([i for i, e in bisection$_errors], [efori, einbisection_errors], marker =' o', label = "Bisection", linewidth = 2)plt.plot([ifori, einfalsepos_errors], [efori, einfalsepos_errors], m s', label = "FalsePosition", linewidth = 2)plt.plot([ifori, einnewton_errors], [efori, einnewton_errors], , label = "Newton–Raphson", linewidth = 2)plt.plot([ifori, einsecant_errors], [efori, einsecant_errors], d', label = "Secant", linewidth = 2)$

plt.xlabel("Iteration") plt.ylabel("Approx. Relative Error ea (plt.title("Convergence Rate Comparison of Root-Finding Methods (Linear Scale)") plt.legend() plt.grid(True) plt.tight$_layout()plt.savefig("convergence_comparison_linear.png")plt.show()$

print(" Linear-scale comparison plot saved as: convergence$_{c}omparison_{l}inear.png$")

5 Output

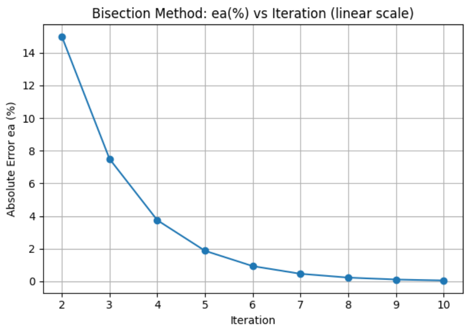
5.1 Bisection Method



```
PS C:\Users\User\OneDrive\Desktop\numerical> python -u "c:\Users\User\OneDrive\Desktop\numerical\prob1.py"
Iter |    xl     |    xu     |    xr     |   f(xr)    |   ea(%)
---------------------------------------------------------------------
   1 |  0.10000 |  0.40000 |   0.25000 |   -0.07189 |    ----
   2 |  0.10000 |  0.25000 |   0.17500 |    0.83645 | 15.00000
   3 |  0.17500 |  0.25000 |   0.21250 |    0.38059 |  7.50000
   4 |  0.21250 |  0.25000 |   0.23125 |    0.15391 |  3.75000
   5 |  0.23125 |  0.25000 |   0.24063 |    0.04084 |  1.87500
   6 |  0.24063 |  0.25000 |   0.24532 |   -0.01561 |  0.93700
   7 |  0.24063 |  0.24532 |   0.24297 |    0.01267 |  0.46900
   8 |  0.24297 |  0.24532 |   0.24415 |   -0.00154 |  0.23500
   9 |  0.24297 |  0.24415 |   0.24356 |    0.00556 |  0.11800
  10 |  0.24356 |  0.24415 |   0.24385 |    0.00207 |  0.05900
---------------------------------------------------------------------
Approximate root after 10 iterations: xr = 0.24385

Final Summary:
Approximate root = 0.24385
Iteration data saved in: bisection_iterations.csv
Error plots saved as: bisection_ea_linear.png and bisection_ea_log.png
PS C:\Users\User\OneDrive\Desktop\numerical>
```

(a) Output



(b) Convergence Graph

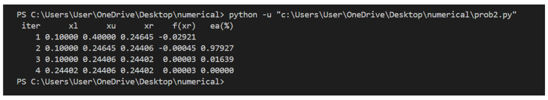
Figure 2: Bisection Method Results

5.2 False Position Method

5.3 Newton–Raphson Method

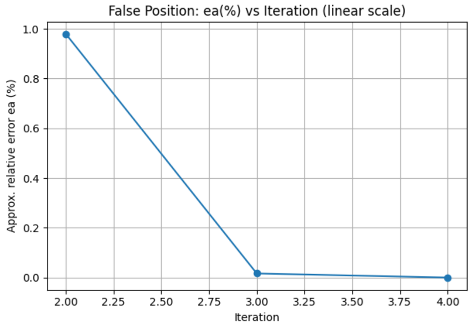
5.4 Secant Method

5.5 Combined Convergence Plot



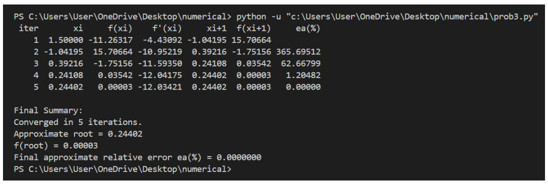
```
PS C:\Users\User\OneDrive\Desktop\numerical> python -u "c:\Users\User\OneDrive\Desktop\numerical\prob2.py"
 iter      xl      xu      xr    f(xr)   ea(%)
    1 0.10000 0.40000 0.24645 -0.02921
    2 0.10000 0.24645 0.24406 -0.00045 0.97927
    3 0.10000 0.24406 0.24402  0.00003 0.01639
    4 0.24402 0.24406 0.24402  0.00003 0.00000
PS C:\Users\User\OneDrive\Desktop\numerical>
```

(a) Output



(b) Convergence Graph

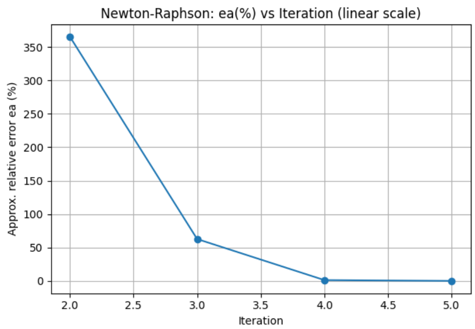
Figure 3: False Position Method Results



```
PS C:\Users\User\OneDrive\Desktop\numerical> python -u "c:\Users\User\OneDrive\Desktop\numerical\prob3.py"
 iter       xi     f(xi)    f'(xi)     xi+1  f(xi+1)     ea(%)
    1  1.50000 -11.26317  -4.43092 -1.04195 15.70664
    2 -1.04195  15.70664 -10.95219  0.39216 -1.75156 365.69512
    3  0.39216  -1.75156 -11.59350  0.24108  0.03542  62.66799
    4  0.24108   0.03542 -12.04175  0.24402  0.00003   1.20482
    5  0.24402   0.00003 -12.03421  0.24402  0.00003   0.00000

Final Summary:
Converged in 5 iterations.
Approximate root = 0.24402
f(root) = 0.00003
Final approximate relative error ea(%) = 0.0000000
PS C:\Users\User\OneDrive\Desktop\numerical>
```

(a) Output



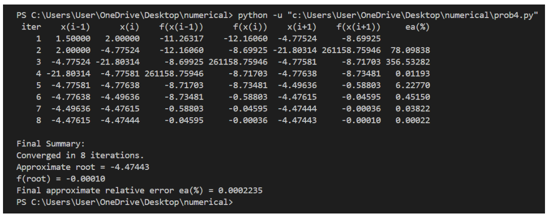
(b) Convergence Graph

Figure 4: Newton–Raphson Method Results

6 Summary

In this experiment, four root-finding algorithms — Bisection, False Position, Newton–Raphson, and Secant — were successfully implemented to solve a nonlinear equation. The numerical results demonstrated the following trends:

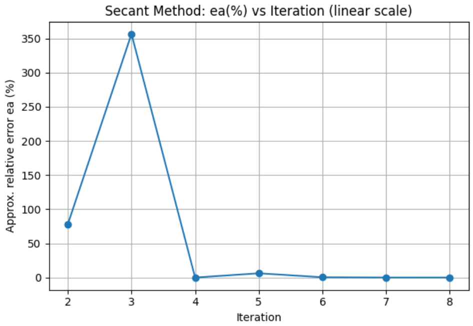
- **Bisection Method:** Slow but guaranteed convergence.
- **False Position Method:** Faster than Bisection but can stagnate near certain root configurations.
- **Newton–Raphson Method:** Fastest convergence due to the use of derivative information but sensitive to initial guess.



```
PS C:\Users\User\OneDrive\Desktop\numerical> python -u "c:\Users\User\OneDrive\Desktop\numerical\prob4.py"
 iter    x(i-1)      x(i)    f(x(i-1))      f(x(i))    x(i+1)    f(x(i+1))     ea(%)
    1   1.50000   2.00000    -11.26317    -12.16060  -4.77524     -8.69925
    2   2.00000  -4.77524    -12.16060     -8.69925 -21.80314 261158.75946  78.09838
    3  -4.77524 -21.80314     -8.69925 261158.75946  -4.77581     -8.71703 356.53282
    4 -21.80314  -4.77581 261158.75946     -8.71703  -4.77638     -8.73481   0.01193
    5  -4.77581  -4.77638     -8.71703     -8.73481  -4.49636     -0.58803   6.22770
    6  -4.77638  -4.49636     -8.73481     -0.58803  -4.47615     -0.04595   0.45150
    7  -4.49636  -4.47615     -0.58803     -0.04595  -4.47444     -0.00036   0.03822
    8  -4.47615  -4.47444     -0.04595     -0.00036  -4.47443     -0.00010   0.00022

Final Summary:
Converged in 8 iterations.
Approximate root = -4.47443
f(root) = -0.00010
Final approximate relative error ea(%) = 0.0002235
PS C:\Users\User\OneDrive\Desktop\numerical>
```

(a) Output



(b) Convergence Graph

Figure 5: Secant Method Results

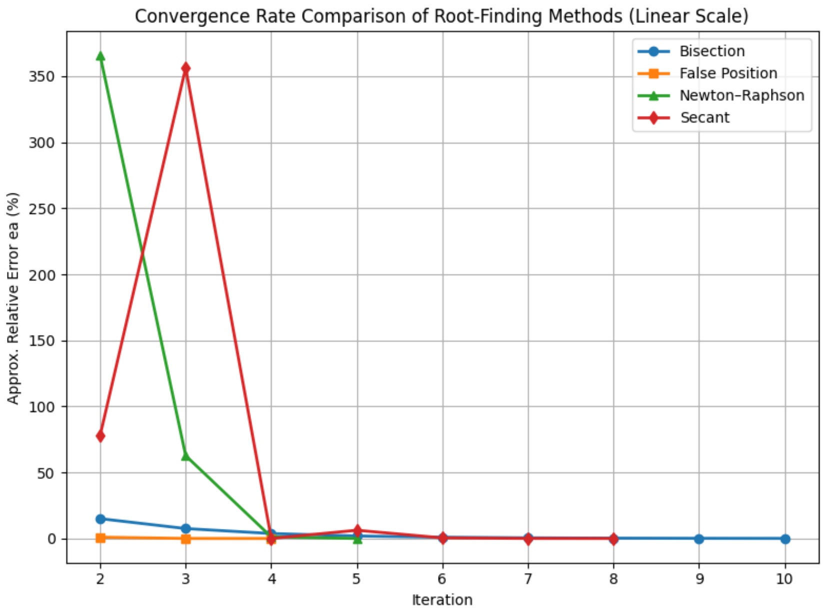


Figure 6: Error vs Iteration for All Methods

- **Secant Method:** Nearly as fast as Newton–Raphson, derivative-free, and more computationally efficient.

The Newton–Raphson method provided the most rapid and accurate convergence, while the Bisection method served as the most reliable fallback approach. This comparative analysis highlights that the optimal root-finding method depends on the function characteristics, desired accuracy, and available derivative information.