Name: Nguyễn Đình Khánh Ngân

ID: ITCSIU22236

Lab 2

Part 1: Bisection Method Objective

Objective: Implement the bisection method to find the root of f(x). Task: Write a program to solve f(x)=x3−x−2=0 using the bisection method. Use [1,2] as the initial interval.

Implement the bisection method function

```python
import math as m

def f(x):
    return x**3 - x - 2

def bisection_method(f, a, b, tol=1e-6, max_iteration=100):
    if f(a) * f(b) > 0:
        raise ValueError("f(a) and f(b) must have opposite signs for the bisection method")
    t_data = []
    c_old = None
    for i in range(1, max_iteration + 1):
        c = (a + b) / 2.0
        fc = f(c)
        if c_old is not None:
            rel_error = abs((c - c_old) / c)
        else:
            rel_error = None
        t_data.append((i, a, b, c, fc, rel_error))
        if fc == 0 or abs(b - a) / 2 < tol or (rel_error is not None and rel_error < tol):
            return c, t_data
        if f(a) * fc < 0:
            b = c
        else:
            a = c
        c_old = c
    # If without meeting tolerance
    return (a + b) / 2.0, t_data
```

```python
root, iteration_table = bisection_method(f, 1, 2, tol=1e-6, max_iteration=100)
print("Bisection Method")
print("--------------------------------------------------------------------------------")
print("Iteration |     a      |     b      |     c      |    f(c)    |    Error    ")
print("----------+------------+------------+------------+------------+-------------")
for r in iteration_table:
    (iter_num, a_val, b_val, c_val, fc_val, rel_err) = r
    if rel_err is None:
        rel_err_str = "N/A"
    else:
        rel_err_str = f"{rel_err:.2e}"
    print(f"  {iter_num:4d}    | {a_val:10.6f} | {b_val:10.6f} | {c_val:10.6f} | {fc_val:10.6f} | {rel_err_str:>11}")
```

```
Bisection Method
--------------------------------------------------------------------------------
Iteration |     a      |     b      |     c      |    f(c)    |    Error    
----------+------------+------------+------------+------------+-------------
     1    |   1.000000 |   2.000000 |   1.500000 |   0.250000 |         N/A
     2    |   1.000000 |   1.500000 |   1.250000 |  -0.437500 |    2.00e-01
     3    |   1.250000 |   1.500000 |   1.375000 |  -0.109375 |    9.09e-02
     4    |   1.375000 |   1.500000 |   1.437500 |   0.066406 |    4.35e-02
     5    |   1.375000 |   1.437500 |   1.406250 |  -0.022461 |    2.22e-02
     6    |   1.406250 |   1.437500 |   1.421875 |   0.021729 |    1.10e-02
     7    |   1.406250 |   1.421875 |   1.414062 |  -0.000427 |    5.52e-03
     8    |   1.414062 |   1.421875 |   1.417969 |   0.010635 |    2.75e-03
     9    |   1.414062 |   1.417969 |   1.416016 |   0.005100 |    1.38e-03
    10    |   1.414062 |   1.416016 |   1.415039 |   0.002336 |    6.90e-04
    11    |   1.414062 |   1.415039 |   1.414551 |   0.000954 |    3.45e-04
    12    |   1.414062 |   1.414551 |   1.414307 |   0.000263 |    1.73e-04
    13    |   1.414062 |   1.414307 |   1.414185 |  -0.000082 |    8.63e-05
    14    |   1.414185 |   1.414307 |   1.414246 |   0.000091 |    4.32e-05
    15    |   1.414185 |   1.414246 |   1.414215 |   0.000004 |    2.16e-05
    16    |   1.414185 |   1.414215 |   1.414200 |  -0.000039 |    1.08e-05
    17    |   1.414200 |   1.414215 |   1.414207 |  -0.000017 |    5.39e-06
    18    |   1.414207 |   1.414215 |   1.414211 |  -0.000006 |    2.70e-06
    19    |   1.414211 |   1.414215 |   1.414213 |  -0.000001 |    1.35e-06
    20    |   1.414213 |   1.414215 |   1.414214 |   0.000002 |    6.74e-07
```

```
print("\nFinal Approximate Root =", root)
print("f(root) =", f(root))
```

```
Final Approximate Root = 1.5213804244995117
f(root) = 4.265829404825894e-06
```

Part 2: Secant Method

Objective: Implement the secant method for the equation f(x)=x2−2=0. Task: Solve the equation starting with initial guesses x0=1 and x1=2.

Implement the secant method function

```
def f(x):
    return x**2 - 2
def secant_method(f, x0, x1, tol=1e-6, max_iteration=100):
    iteration_data = []
    for i in range(1, max_iteration+1):
        fx0 = f(x0)
        fx1 = f(x1)
        if fx1 - fx0 == 0:
            raise ZeroDivisionError("Denominator zero in secant method")
        x2 = x1 - fx1 * ((x1 - x0)/(fx1 - fx0))
        error_pct = abs((x2 - x1)/x2) * 100
        iteration_data.append((i, x0, x1, x2, f(x2), error_pct))
        if error_pct < (tol*100):
            return x2, iteration_data
        x0, x1 = x1, x2
    return x1, iteration_data
```

```
root_s, table_s = secant_method(f, 1, 2, tol=1e-6)
print("Part 2: Secant Method")
print("Iteration |   x_old    |   x_new    |   x_next   |   f(x_next)   |   Error%")
print("----------+------------+------------+------------+---------------+-----------")

for r in table_s:
    i, x_old, x_new, x_next, fx_next, err_p = r
    print(f"   {i:4d}   | {x_old:10.6f} | {x_new:10.6f} | {x_next:10.6f} | {fx_next:13.6e} | {err_p:9.4e}")
```

```
Part 2: Secant Method
Iteration |   x_old    |   x_new    |   x_next   |   f(x_next)   |   Error%
----------+------------+------------+------------+---------------+-----------
      1   |   1.000000 |   2.000000 |   0.765035 | -4.367634e-02 | 1.6143e+02
      2   |   2.000000 |   0.765035 |   0.742299 | -5.383261e-03 | 3.0628e+00
      3   |   0.765035 |   0.742299 |   0.739103 | -3.035433e-05 | 4.3243e-01
      4   |   0.742299 |   0.739103 |   0.739085 | -2.150675e-08 | 2.4522e-03
      5   |   0.739103 |   0.739085 |   0.739085 | -8.604228e-14 | 1.7387e-06
```

```
print(f"\nApproximate root after {len(table_s)} iterations: {root_s:.8f}")
print(f"f(root) = {f(root_s):.4e}\n")
```

```
Approximate root after 6 iterations: 1.41421356
f(root) = 8.8818e-16
```

Part 3: Newton-Raphson Method

Objective: Apply the Newton-Raphson method to find the root of f(x)=cos(x)−x=0. Task: Write a program to implement the method, starting from x0=0.5.

Implement the Newton-Raphson method function

```
def f(x):
    return m.cos(x) - x
def df_newton(x):
    return -m.sin(x) - 1
def newton_raphson_method(f, df, x0, tol=1e-6, max_iteration=100):
    iteration_data = []
    x = x0
    for i in range(1, max_iteration+1):
        fx = f(x)
        dfx = df(x)
        if dfx == 0:
            raise ZeroDivisionError("Derivative is zero")
        x_new = x - fx/dfx
        error_pct = abs((x_new - x)/x_new) * 100
        iteration_data.append((i, x, x_new, f(x_new), error_pct))
        if error_pct < (tol*100):
            return x_new, iteration_data
        x = x_new
    return x, iteration_data
```

```
root_nt, table_nt = newton_raphson_method(f, df_newton, x0=0.5, tol=1e-6)
print("Part 3: Newton-Raphson Method")
print("Iteration |   x_old    |   x_new    |   f(x_new)    |   Error%")
print("----------+------------+------------+---------------+-----------")
for r in table_nt:
    i, x_old, x_new, fx_new, err_p = r
    print(f"   {i:4d}   | {x_old:10.6f} | {x_new:10.6f} | {fx_new:13.6e} | {err_p:9.4e}")
```

```
print(f"\nApproximate root after {len(table_nt)} iterations: {root_nt:.8f}")
print(f"f(root) = {f(root_nt):.4e}\n")
```

```
Approximate root after 4 iterations: 0.73908513
f(root) = 0.0000e+00
```

Part 4: Comparative Analysis

Objective: Compare the performance (iterations, accuracy) of the three methods for the same problem f(x)=x3−x−2=0. Task: Solve the equation using all three methods. Tabulate the number of iterations and the accuracy

```
def f_compare(x):
    return x**3 - x - 2
def df_compare(x):
    return 3*x**2 - 1
# Bisection
root_bi, table_bi = bisection_method(f_compare, 1, 2, tol=1e-6)

# Secant
root_sc, table_sc = secant_method(f_compare, 1, 2, tol=1e-6)

# Newton-Raphson
def newton_raphson_compare(x0, tol=1e-6):
    x = x0
    iteration_data = []
    for i in range(1, 101):
        fx = f_compare(x)
        dfx = df_compare(x)
        if dfx == 0:
            raise ZeroDivisionError("Derivative = 0. No convergence")
        x_new = x - fx/dfx
        error_pct = abs((x_new - x)/x_new) * 100
        iteration_data.append((i, x, x_new, f_compare(x_new), error_pct))
        if error_pct < tol*100:
            return x_new, iteration_data
        x = x_new
    return x, iteration_data
```

```
root_nr, table_nr = newton_raphson_compare(1.5, tol=1e-6)
bisection_iter = len(table_bi)
secant_iter    = len(table_sc)
nr_iter        = len(table_nr)
f_bi = f_compare(root_bi)
f_sc = f_compare(root_sc)
f_nr = f_compare(root_nr)
```

```
print("Part 4: Comparative Analysis")
print("-------------------------------------------------------------------")
print("     Method      | Iteration  | Approxiate Root |   f(root)     ")
print("-----------------+------------+-----------------+---------------")
print(f"Bisection        | {bisection_iter:10d} | {root_bi:14.8f} | {f_bi:14.5e}")
print(f"Secant           | {secant_iter:10d} | {root_sc:14.8f} | {f_sc:14.5e}")
print(f"Newton-Raphson   | {nr_iter:10d} | {root_nr:14.8f} | {f_nr:14.5e}")
```

```
Part 4: Comparative Analysis
-------------------------------------------------------------------
     Method      | Iteration  | Approxiate Root |   f(root)
-----------------+------------+-----------------+---------------
Bisection        |         20 |     1.52138042 |    4.26583e-06
Secant           |          7 |     1.52137971 |   -1.84297e-14
Newton-Raphson   |          3 |     1.52137971 |    4.52971e-14
```

Part 5: Problem

Solve the equation f(x)=ln(x)+x2−4=0 using any two methods of your choice. Compare the results and discuss the best method for this problem.

Ln(x) is defined for x>0, so it need to be positive initial guesses or intervals.

Define the function

```
def f_part(x):
    return math.log(x) + x**2 - 4

def df_part(x):
    return 1/x + 2*x
```

Bisection Method

- $f(1) = \ln(1) + 1^2 - 4 = -3 < 0$
- $f(2) = \ln(2) + 2^2 - 4 = \ln(2) + 4 - 4 \approx 0.6931 > 0$

  So [1, 2] works for bisection

```
root_bi, table_bi = bisection_method(f_part, 1, 2, tol=1e-6)

print("Part 5: Solve ln(x) + x^2 - 4 = 0")
print("Using Bisection on [1,2]:")
print(f"Root approx: {root_bi:.8f}")
print(f"f(root)    : {f_part(root_bi):.4e}\n")
```

```
Part 5: Solve ln(x) + x^2 - 4 = 0
Using Bisection on [1,2]:
Root approx: 1.84109783
f(root)    : 3.2674e-06
```

```
def newton_part(x0, tol=1e-6, max_iteration=100):
    x = x0
    iteration_data = []
    for i in range(1, max_iteration+1):
        fx = f_part(x)
        dfx = df_part(x)
        if dfx == 0:
            raise ZeroDivisionError("Derivative is zero")
        x_new = x - fx/dfx
        error_pct = abs((x_new - x)/x_new)*100
        iteration_data.append((i, x, x_new, f_part(x_new), error_pct))
        if error_pct < tol*100:
            return x_new, iteration_data
        x = x_new
    return x, iteration_data

root_nr, table_nr = newton_part(1.5, tol=1e-6)
print("Using Newton-Raphson, x0=1.5:")
print("Iterations:", len(table_nr))
print(f"Root approx: {root_nr:.8f}")
print(f"f(root)    : {f_part(root_nr):.4e}\n")
```

```
Using Newton-Raphson, x0=1.5:
Iterations: 4
Root approx: 1.84109706
f(root)    : -4.4409e-16
```

- Bisection is guaranteed to converge once we have a valid bracket [1,2]
- Newton-Raphson typically converges faster if the initial guess is good and the function is well-behaved
- For $\ln(x) + x^2 - 4$, there is only one real root (in (1,2)). Newton often converges quickly

Newton usually has fewer iterations than Bisection