Name: Phan Tran Thanh Huy

ID: ITCSIU22056

Lab 2

Part 1: Bisection Method

Code:

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

def bisection_method(a, b, tol=1e-6, max_iter=100):
    if f(a) * f(b) >= 0:
        print("Bisection method fails. f(a) and f(b) must have opposite signs.")
        return None

    iter_count = 0
    c_old = a

    print(f"{'Iter':<5}{'a':<10}{'b':<10}{'c':<10}{'f(c)':<15}{'Error (%)':<15}")
    print("-" * 60)

    while (b - a) / 2 > tol and iter_count < max_iter:
        c_new = (a + b) / 2
        error = abs((c_new - c_old) / c_new) * 100 if iter_count > 0 else None

        print(f"{iter_count:<5}{a:<10.6f}{b:<10.6f}{c_new:<10.6f}{f(c_new):<15.6f}{error if error is not None else 'N/A':<15}")

        if f(c_new) == 0:
            return c_new
        elif f(a) * f(c_new) < 0:
            b = c_new
        else:
            a = c_new

        c_old = c_new
        iter_count += 1

    return (a + b) / 2

root = bisection_method(1, 2)
print("\nApproximate root:", root)
```

Result:

```
PS E:\Homework\TMC\Lab2> & C:/Python312/python.exe e:/Homework/TMC/Lab2/ex1.py
Iter a         b         c         f(c)           Error (%)
----------------------------------------------------------
0    1.000000  2.000000  1.500000  -0.125000      N/A
1    1.500000  2.000000  1.750000  1.609375       14.285714285714285
2    1.500000  1.750000  1.625000  0.666016       7.6923076923076925
3    1.500000  1.625000  1.562500  0.252197       4.0
4    1.500000  1.562500  1.531250  0.059113       2.0408163265306123
5    1.500000  1.531250  1.515625  -0.034054      1.0309278350515463
6    1.515625  1.531250  1.523438  0.012250       0.5128205128205128
7    1.515625  1.523438  1.519531  -0.010971      0.2570694087403599
8    1.519531  1.523438  1.521484  0.000622       0.12836970474967907
9    1.519531  1.521484  1.520508  -0.005179      0.06422607578676942
10   1.520508  1.521484  1.520996  -0.002279      0.03210272873194221
11   1.520996  1.521484  1.521240  -0.000829      0.016048788316482106
12   1.521240  1.521484  1.521362  -0.000103      0.008023750300890637
13   1.521362  1.521484  1.521423  0.000259       0.0040117142054800015
14   1.521362  1.521423  1.521393  0.000078       0.002005897338174232
15   1.521362  1.521393  1.521378  -0.000013      0.0010029587282483325
16   1.521378  1.521393  1.521385  0.000033       0.0005014768493212511
17   1.521378  1.521385  1.521381  0.000010       0.00025073905335977795
18   1.521378  1.521381  1.521379  -0.000001      0.00012536968385526822

Approximate root: 1.5213804244995117
```

Part 2: Secant Method

Code:

```python
def f(x):
    return x**2 - 2

def secant_method(x0, x1, tol=1e-6, max_iter=100):
    iter_count = 0

    print(f"{'Iter':<5}{'x0':<15}{'x1':<15}{'x2':<15}{'f(x2)':<15}{'Error (%)':<15}")
    print("-" * 75)

    while abs(x1 - x0) > tol and iter_count < max_iter:
        if f(x1) - f(x0) == 0:
            print("Division by zero encountered in secant method.")
            return None

        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        error = abs((x2 - x1) / x2) * 100 if iter_count > 0 else None

        print(f"{iter_count:<5}{x0:<15.6f}{x1:<15.6f}{x2:<15.6f}{f(x2):<15.6f}{error if error is not None else 'N/A':<15}")

        x0, x1 = x1, x2
        iter_count += 1

    return x1

root = secant_method(1, 2)
print("\nApproximate root:", root)
```

Result:

```
PS E:\Homework\TMC\Lab2> & C:/Python312/python.exe e:/Homework/TMC/Lab2/ex2.py
Iter x0             x1             x2             f(x2)          Error (%)
---------------------------------------------------------------------------
0    1.000000       2.000000       1.333333       -0.222222      N/A
1    2.000000       1.333333       1.400000       -0.040000      4.761904761904762
2    1.333333       1.400000       1.414634       0.001190       1.0344827586206748
3    1.400000       1.414634       1.414211       -0.000006      0.029890004782391323
4    1.414634       1.414211       1.414214       -0.000000      0.00015015995513728908
5    1.414211       1.414214       1.414214       0.000000       2.2328661677132043e-08

Approximate root: 1.4142135623730954
```

Part 3: Newton-Raphson Method

Code:

```python
import math

def f(x):
    return math.cos(x) - x

def df(x):
    return -math.sin(x) - 1

def newton_raphson(x0, tol=1e-6, max_iter=100):
    iter_count = 0

    print(f"{'Iter':<5}{'x_old':<15}{'x_new':<15}{'f(x_new)':<15}{'Error (%)':<15}")
    print("-" * 75)

    while abs(f(x0)) > tol and iter_count < max_iter:
        if df(x0) == 0:
            print("Derivative is zero. Newton-Raphson method fails.")
            return None

        x_new = x0 - f(x0) / df(x0)
        error = abs((x_new - x0) / x_new) * 100 if iter_count > 0 else None

        print(f"{iter_count:<5}{x0:<15.6f}{x_new:<15.6f}{f(x_new):<15.6f}{error if error is not None else 'N/A':<15}")

        x0 = x_new
        iter_count += 1

    return x0

root = newton_raphson(0.5)
print("\nApproximate root:", root)
```

Result:

```
PS E:\Homework\TMC\Lab2> & C:/Python312/python.exe e:/Homework/TMC/Lab2/ex3.py
Iter x_old          x_new          f(x_new)       Error (%)
---------------------------------------------------------------------------
0    0.500000       0.755222       -0.027103      N/A
1    0.755222       0.739142       -0.000095      2.17559795262529
2    0.739142       0.739085       -0.000000      0.007648946850347983

Approximate root: 0.7390851339208068
```

Part 4: Comparative Analysis

Code:

```python
import math

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

def df(x):
    return 3*x**2 - 1

def bisection_method(a, b, tol=1e-6, max_iter=100):
    if f(a) * f(b) >= 0:
        print("Bisection method fails. f(a) and f(b) must have opposite signs.")
        return None, 0

    iter_count = 0
    while (b - a) / 2 > tol and iter_count < max_iter:
        c = (a + b) / 2
        if f(c) == 0:
            return c, iter_count
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iter_count += 1

    return (a + b) / 2, iter_count

def secant_method(x0, x1, tol=1e-6, max_iter=100):
    iter_count = 0
    while abs(x1 - x0) > tol and iter_count < max_iter:
        if f(x1) - f(x0) == 0:
            print("Division by zero encountered in secant method.")
            return None, 0

        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        x0, x1 = x1, x2
        iter_count += 1

    return x1, iter_count

def newton_raphson(x0, tol=1e-6, max_iter=100):
    iter_count = 0
    while abs(f(x0)) > tol and iter_count < max_iter:
        if df(x0) == 0:
            print("Derivative is zero. Newton-Raphson method fails.")
            return None, 0

        x0 = x0 - f(x0) / df(x0)
        iter_count += 1

    return x0, iter_count

bisection_root, bisection_iters = bisection_method(1, 2)
secant_root, secant_iters = secant_method(1, 2)
newton_root, newton_iters = newton_raphson(1.5)

print("\nComparison of Root Finding Methods:")
print("Method\t\tRoot\t\t\tIterations")
print(f"Bisection\t{bisection_root}\t{bisection_iters}")
print(f"Secant\t\t{secant_root}\t{secant_iters}")
print(f"Newton-Raphson\t{newton_root}\t{newton_iters}")
```

Result:

```
PS E:\Homework\TMC\Lab2> & C:/Python312/python.exe e:/Homework/TMC/Lab2/ex4.py
Method          Root                    Iterations
Bisection       1.5213804244995117      19
Secant          1.5213797068045645      7
Newton-Raphson  1.5213798059647863      2
```

Part 5: Problem

I choose Secant method and Newton-Raphson method to solve this problem.
Code:

```python
import math

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

def df(x):
    return 1/x + 2*x

def secant_method(x0, x1, tol=1e-6, max_iter=100):
    iter_count = 0
    while abs(x1 - x0) > tol and iter_count < max_iter:
        if f(x1) - f(x0) == 0:
            print("Division by zero encountered in secant method.")
            return None, 0

        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        x0, x1 = x1, x2
        iter_count += 1

    return x1, iter_count

def newton_raphson(x0, tol=1e-6, max_iter=100):
    iter_count = 0
    while abs(f(x0)) > tol and iter_count < max_iter:
        if df(x0) == 0:
            print("Derivative is zero. Newton-Raphson method fails.")
            return None, 0

        x0 = x0 - f(x0) / df(x0)
        iter_count += 1

    return x0, iter_count

secant_root, secant_iters = secant_method(1,2)
newton_root, newton_iters = newton_raphson(2)

print("Method\t\tRoot\t\t\tIterations")
print(f"Secant\t\t{secant_root}\t{secant_iters}")
print(f"Newton-Raphson\t{newton_root}\t{newton_iters}")
```

Result:

```
PS E:\Homework\TMC\Lab2> & C:/Python312/python.exe e:/Homework/TMC/Lab2/ex5.py
Method          Root                    Iterations
Secant          1.8410970584500779      5
Newton-Raphson  1.8410970584546869      3
```

Discussion: For solving the problem, the Newton-Raphson method is the better choice due to its faster convergence and fewer iterations. However, if the derivative were difficult to determine or the initial guess were poor, the Secant method would be a viable alternative. While the Secant method is more robust in some cases, it generally requires more iterations. Therefore, Newton-Raphson is preferred for this problem due to its efficiency.