Name: Phan Tran Thanh Huy

ID: ITCSIU22056

Lab 4

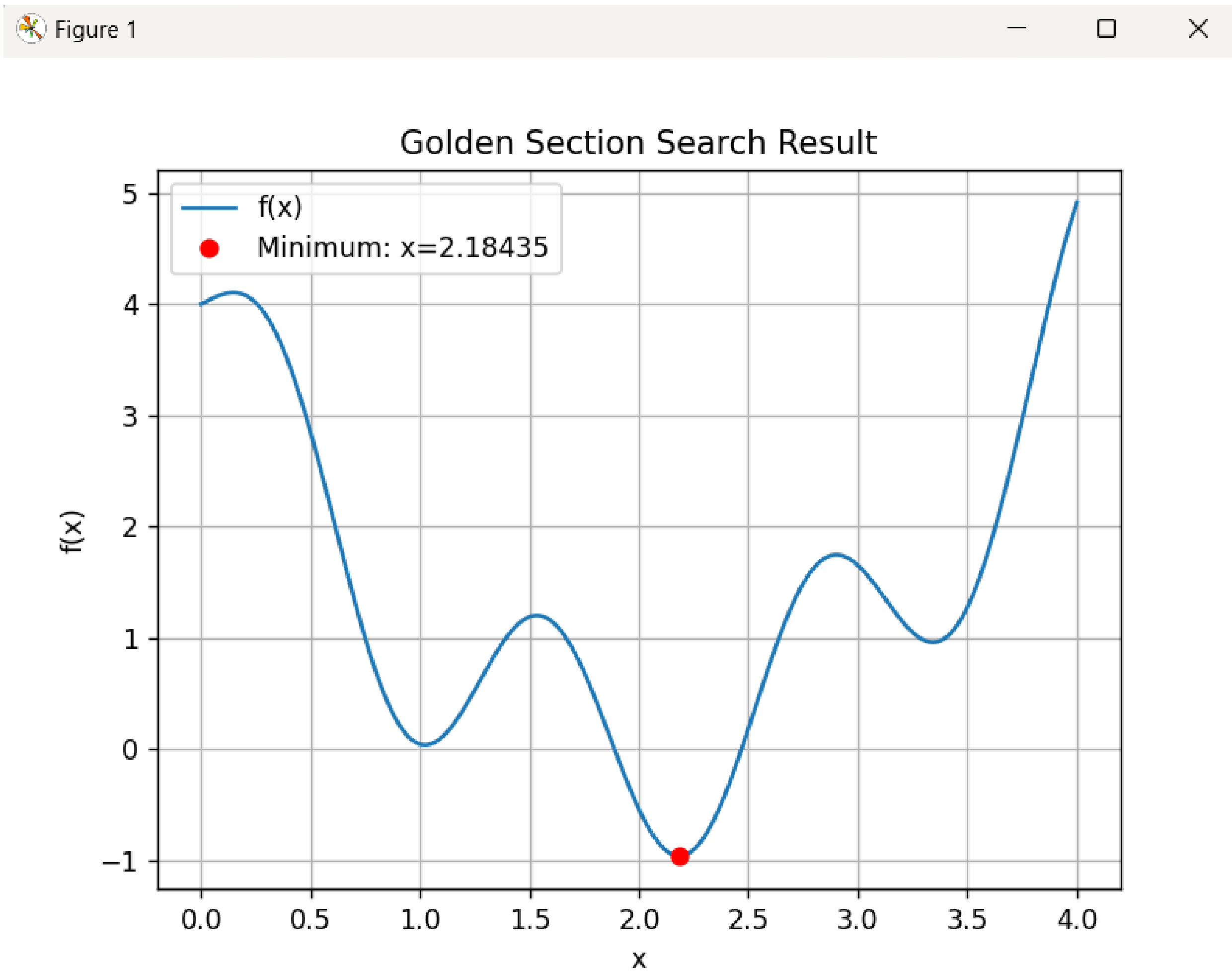
Activity 1: Golden Section Search

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x):
    return (x - 2)**2 + np.sin(5 * x)

# Golden Section Search implementation
def golden_section_search(f, a, b, tol=1e-5):
    gr = (np.sqrt(5) + 1) / 2  # Golden ratio

    c = b - (b - a) / gr
    d = a + (b - a) / gr

    while abs(b - a) > tol:
        if f(c) < f(d):
            b = d
        else:
            a = c

        # Recalculate points
        c = b - (b - a) / gr
        d = a + (b - a) / gr

    x_min = (b + a) / 2
    return x_min, f(x_min)

# Run the search
x_min, y_min = golden_section_search(f, 0, 4)

# Print the results
print(f"Minimum value found at x = {x_min:.5f}")
print(f"Minimum value of f(x) = {y_min:.5f}")

# Plot the function and minimum
x_vals = np.linspace(0, 4, 500)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label="f(x)")
plt.plot(x_min, y_min, 'ro', label=f"Minimum: x={x_min:.5f}")
plt.title("Golden Section Search Result")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()
```

Result:

```
Minimum value found at x = 2.18435
Minimum value of f(x) = -0.96329
```



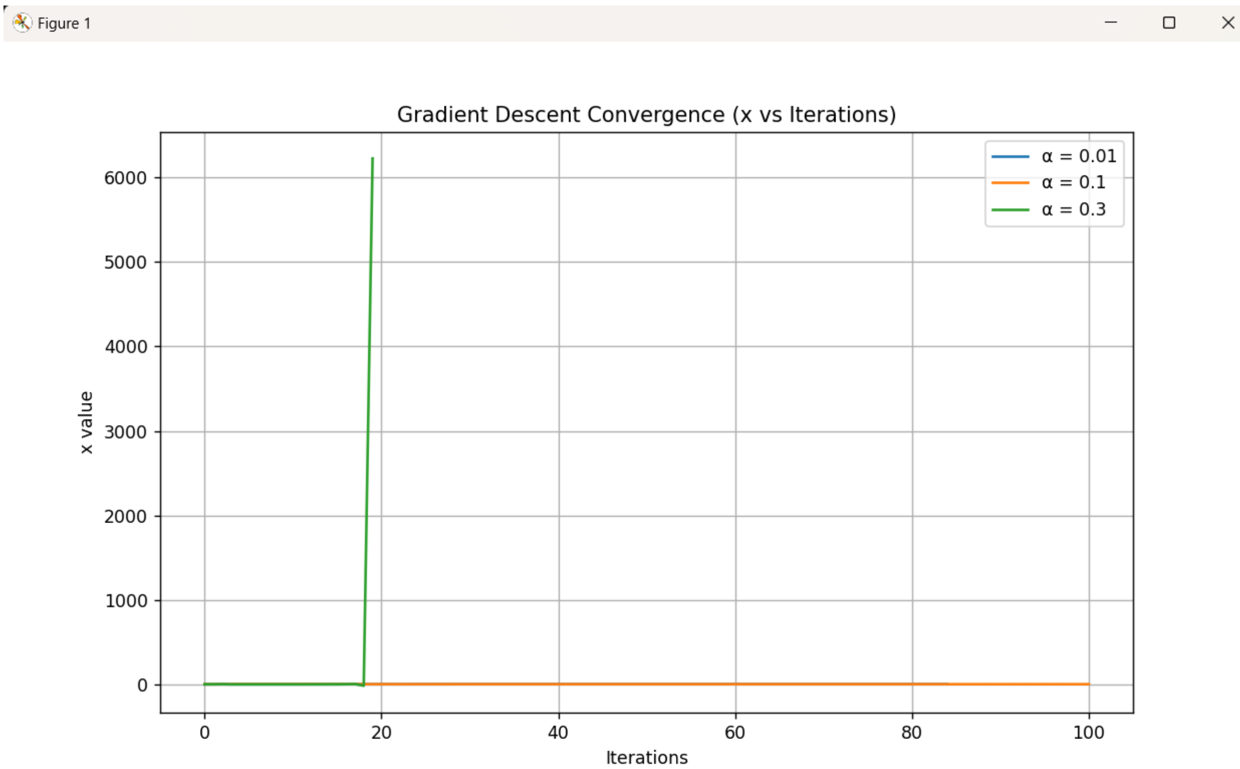
Activity 2: Gradient Descent in 1D

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Define the function and its derivative
def f(x):
    return x**4 - 3*x**3 + 2

def df(x):
    return 4*x**3 - 9*x**2

# Gradient descent implementation
def gradient_descent(df, x0, alpha, max_iter=100, tol=1e-6):
    x_vals = [x0]
    x = x0
    for _ in range(max_iter):
        try:
            grad = df(x)
            x_new = x - alpha * grad
        except OverflowError:
            print("Overflow detected. Gradient too large.")
            break

        if abs(x_new) > 1e6:  # Arbitrary large value to detect divergence
            print("Divergence detected. x became too large.")
            break

        x_vals.append(x_new)
        if abs(x_new - x) < tol:
            break
        x = x_new
    return x_vals

# Plotting function
def plot_convergence(x_lists, alphas):
    plt.figure(figsize=(10, 6))
    for x_vals, alpha in zip(x_lists, alphas):
        plt.plot(range(len(x_vals)), x_vals, label=f'α = {alpha}')

    plt.title('Gradient Descent Convergence (x vs Iterations)')
    plt.xlabel('Iterations')
    plt.ylabel('x value')
    plt.grid(True)
    plt.legend()
    plt.show()

# Main function
def run_experiment():
    x0 = 0.5
    alphas = [0.01, 0.1, 0.3]
    results = []

    for alpha in alphas:
        x_vals = gradient_descent(df, x0, alpha)
        results.append(x_vals)
        print(f"\nα = {alpha} converged to x = {x_vals[-1]:.5f} in {len(x_vals)-1} iterations")

    plot_convergence(results, alphas)

if __name__ == "__main__":
    run_experiment()
```

Result:

```
PS E:\Homework\TMC\Lab4> & C:/Python312/python.exe e:/Homework/TMC/Lab4/p2.py

α = 0.01 converged to x ≈ 2.25000 in 84 iterations

α = 0.1 converged to x ≈ 2.14736 in 100 iterations
Divergence detected. x became too large.

α = 0.3 converged to x ≈ 6225.65158 in 19 iterations
```



Analyze converge:

- Lower α converges slower but more stable.
- Larger α (like 0.3) might overshoot or diverge depending on the function shape. For α = 0.3, it will **warn about divergence** instead of crashing.

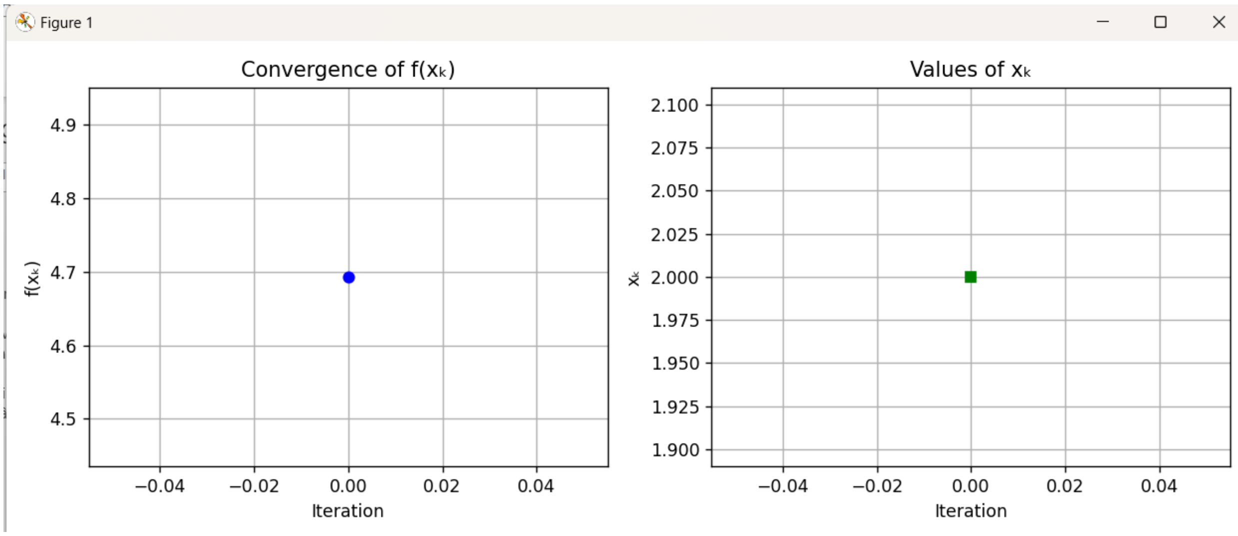
Activity 3: Newton's Method

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define function and its derivatives
def f(x):
    return np.log(x) + x**2

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

def d2f(x):
    return -1/(x**2) + 2

# Newton's Method implementation
def newtons_method(df, d2f, x0, tol=1e-6, max_iter=100):
    x_vals = [x0]
    f_vals = [f(x0)]
    x = x0

    for _ in range(max_iter):
        grad = df(x)
        hess = d2f(x)

        if hess == 0:
            print("Zero second derivative. Stopping.")
            break

        x_new = x - grad / hess

        if x_new <= 0:
            print("x out of domain (x <= 0). Stopping.")
            break

        x_vals.append(x_new)
        f_vals.append(f(x_new))

        if abs(x_new - x) < tol:
            break

        x = x_new

    return x_vals, f_vals

# Plotting functions
def plot_convergence(x_vals, f_vals):
    iterations = range(len(x_vals))

    # Plot 1: f(xk) over iterations
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(iterations, f_vals, marker='o', color='blue')
    plt.title("Convergence of f(xₖ)")
    plt.xlabel("Iteration")
    plt.ylabel("f(xₖ)")
    plt.grid(True)

    # Plot 2: xk over iterations
    plt.subplot(1, 2, 2)
    plt.plot(iterations, x_vals, marker='s', color='green')
    plt.title("Values of xₖ")
    plt.xlabel("Iteration")
    plt.ylabel("xₖ")
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Run experiment
def run_newton_experiment():
    x0 = 2
    x_vals, f_vals = newtons_method(df, d2f, x0)

    print(f"\nConverged to x ≈ {x_vals[-1]:.6f} with f(x) ≈ {f_vals[-1]:.6f} in {len(x_vals) - 1} iterations")
    plot_convergence(x_vals, f_vals)

if __name__ == "__main__":
    run_newton_experiment()
```

Result:

```
PS E:\Homework\TMC\Lab4> & C:/Python312/python.exe e:/Homework/TMC/Lab4/p3.py
x out of domain (x <= 0). Stopping.

Converged to x ≈ 2.000000 with f(x) ≈ 4.693147 in 0 iterations
```



Discuss:

- The method **converges very fast** due to **quadratic convergence** near the minimum.
- The exact minimum is at x=12≈0.7071x = \frac{1}{\sqrt{2}} \approx 0.7071x=21≈0.7071, where f'(x)=0f'(x) = 0f'(x)=0.
- Newton's Method performs well **as long as the starting point is in the domain** (here, x>0x > 0x>0).
- If started too close to zero, instability may occur due to ln(x)\ln(x)ln(x) and 1x2\frac{1}{x^2}x21.