Name: Nguyễn Đình Khánh Ngân

ID: ITCSIU22236

Lab 4

Part 1: Golden Section Search

Objective: Minimize the function

$$f(x) = (x-2)^2 + \sin(5x), \quad x \in [0,4]$$

Task:

Implement Golden Section Search

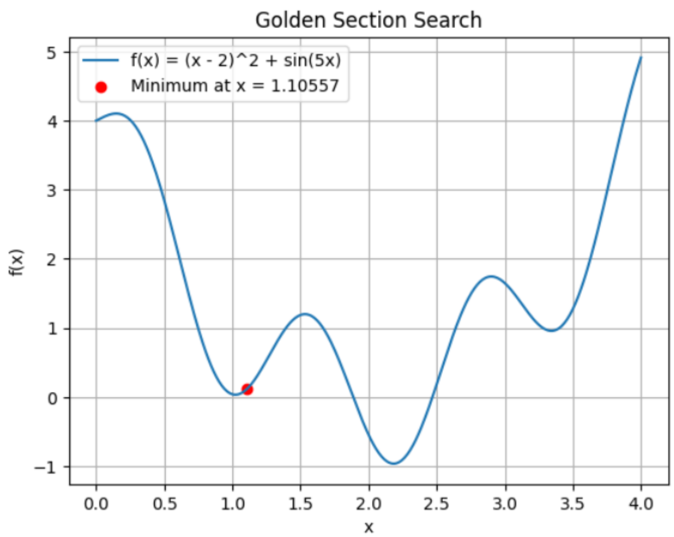
```
# Golden Section Search
def gs_search(a, b, tol):
    t = (np.sqrt(5) - 1) / 2  # (τ)
    x1 = a * (1 - t) + b * t
    x2 = a * t + b * (1 - t)
    while abs(b - a) > tol:
        if f(x1) > f(x2):
            a = x1
            x1 = x2
            x2 = a * t + b * (1 - t)
        else:
            b = x2
            x2 = x1
            x1 = a * (1 - t) + b * t
    min = (a + b) / 2
    return min
```

Use a tolerance of $\varepsilon = 10^{-5}$

```
a, b = 0, 4
tolerance = 1e-5
minimum = golden_section_search(a, b, tol=tolerance)
```

Plot the function and mark the minimum found

```
a, b = 0, 4
tolerance = 1e-5
min = gs_search(a, b, tol=tolerance)
x_vals = np.linspace(a, b, 400)
y_vals = f(x_vals)
plt.plot(x_vals, y_vals, label="f(x) = (x - 2)^2 + sin(5x)")
plt.scatter(min, f(min), color='red', label=f'Minimum at x = {min:.5f}')
plt.title('Golden Section Search')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
print(f"Minimum value at {min:.5f}")
```




```
Minimum value at 1.10557
```

Part 2: Gradient Descent in 1D

Objective: Minimize the function

$$f(x) = x^4 - 3x^3 + 2, \quad \text{starting at } x_0 = 0.5$$

Task:

Implement gradient descent

$$x_{k+1} = x_k - \alpha f'(x_k)$$

```
def g_d(l_r, x0, iteration):
    x_values = [x0]
    x = x0
    for i in range(iteration):
        x = x - l_r * df(x)
        if abs(x) > 1e5:  # Limit
            print(f"Overflow iteration {i} with x = {x}")
            break
        x_values.append(x)
    return x_values
```

Derive

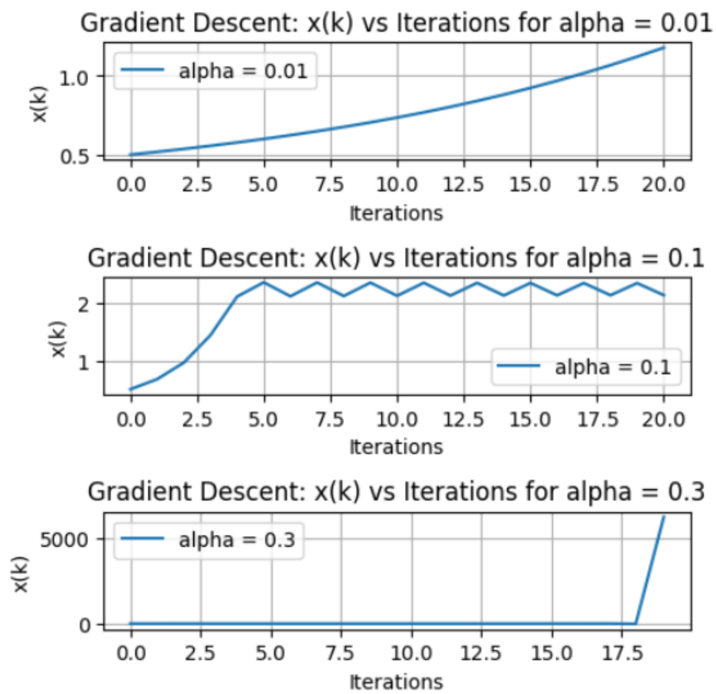
$$f'(x) = 4x^3 - 9x^2.$$

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

Try different learning rates α = 0.01, 0.1 , 0.3

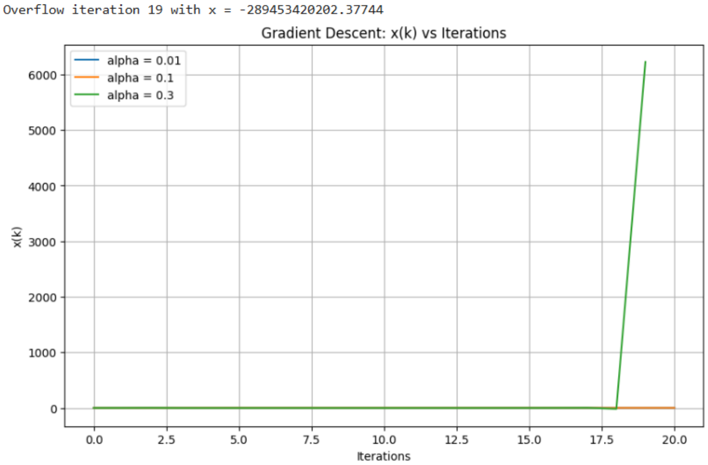
```
x0 = 0.5
iteration = 20
l_r = [0.01, 0.1, 0.3]
fig, axes = plt.subplots(3, 1, figsize=(5, 5))
for i, a in enumerate(l_r):
    x_values = g_d(a, x0, iteration)
    axes[i].plot(range(len(x_values)), x_values, label=f'alpha = {a}')
    axes[i].set_title(f'Gradient Descent: x(k) vs Iterations for alpha = {a}')
    axes[i].set_xlabel('Iterations')
    axes[i].set_ylabel('x(k)')
    axes[i].legend()
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

```
Overflow iteration 19 with x = -289453420202.37744
```



Plot x_k vs iterations. Analyze convergence

```
Overflow iteration 19 with x = -289453420202.37744
```



For small learning rates like α=0.01, we will see gradual convergence with minimal overshooting

For moderate learning rates like α=0.05, we will see a faster convergence with a stable trajectory

For higher learning rates like α=0.1, the algorithm may fail to converge properly, showing oscillations or divergence

Part 3: Newton's Method

Objective: Minimize the function

$$f(x) = \ln(x) + x^2, \quad x > 0, \quad x_0 = 2$$

Task:

Compute derivative

$$f'(x) = \frac{1}{x} + 2x$$
$$f''(x) = -\frac{1}{x^2} + 2$$

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

```
def d2_f(x):
    return -1/x**2 + 2
```

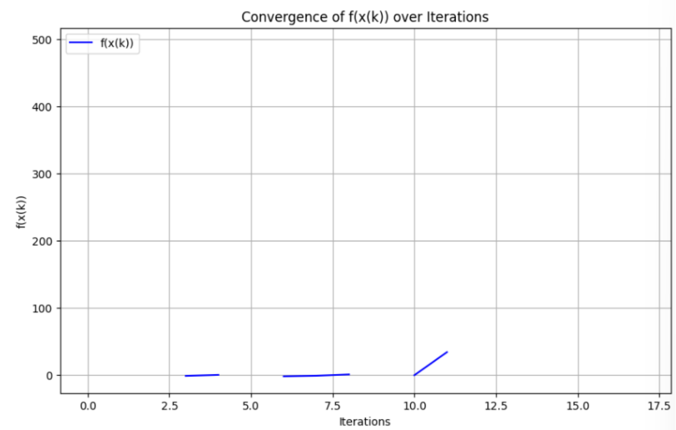
Use Newton's update

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

```
def newtons_method(x0, iteration, tolerance):
    x_vals = [x0]
    f_vals = [f(x0)]
    for _ in range(iteration):
        x_new = x_vals[-1] - df(x_vals[-1]) / d2_f(x_vals[-1])
        x_vals.append(x_new)
        f_vals.append(f(x_new))
        if abs(x_vals[-1] - x_vals[-2]) < tolerance:
            break
    return x_vals, f_vals
```

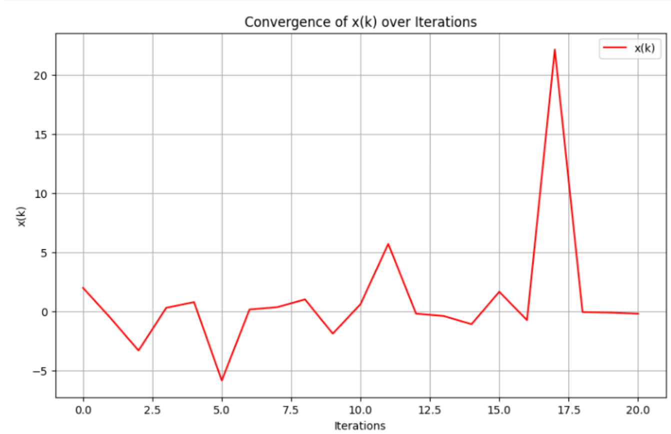
Plot convergence of f(x_k) and number of iterations

```
plt.figure(figsize=(10, 6))
plt.plot(range(len(f_vals)), f_vals, label="f(x(k))", color='b')
plt.title("Convergence of f(x(k)) over Iterations")
plt.xlabel("Iterations")
plt.ylabel("f(x(k))")
plt.grid(True)
plt.legend()
plt.show()
```



Discuss behavior near minimum

```
plt.figure(figsize=(10, 6))
plt.plot(range(len(x_vals)), x_vals, label="x(k)", color='r')
plt.title("Convergence of x(k) over Iterations")
plt.xlabel("Iterations")
plt.ylabel("x(k)")
plt.grid(True)
plt.legend()
plt.show()
```



Newton's method is highly sensitive to the initial guess. If the starting point is not close enough to the true root or if the function behaves non-linearly near the minimum, it can lead to large oscillations

Newton's method can converge quickly when the function is well-behaved, and the initial guess is close to the root. However, near the minimum, especially when the function flattens or has a small gradient

```python
print("Final value of x:", x_vals[-1])
print("f(x) at the final x:", f_vals[-1])
```

```
Final value of x: -0.18482065445840257
f(x) at the final x: nan
```

The peak at iteration 17 might suggest that the method is overshooting, potentially because the derivative values are causing large changes in x(k). This can happen if the second derivative f''(x) is very small or near zero, causing a large update step

As Newton's method involves division by the second derivative, when f''(x) becomes very small, the step size becomes large, which may cause such spikes or divergences