Nirbhay Sharma (B19CSE114)

Optimization for Machine Learning

Python Code

```
import numpy as np

# def calculate_g(x:np.array, n:int) -> int:
#     """
#     x:(n,)
#     """
#     coeff = 9 / (n-1)
#     sum_x = 0;
#     for i in range(1,n):
#         sum_x += x[i]

#     return 1 + coeff * sum_x

# def calculate_f(x:np.array, n:int) -> int:
#     """
#     x:(n,)
#     """
#     g_value = calculate_g(x,n)
#     f = g_value - (x[0] ** 2 / g_value)
#     return f


def calculate_f(x:np.array) -> int:
    return 4 * x[0] ** 2 - 3 * x[0] * x[1] + 2 * x[1] ** 2 - x[0] + 2 *
x[1]

def grad_f(x:np.array, n:int) -> np.array:
    h = 1e-5
    grad = []
    f_value = calculate_f(x)
    for i in range(n):
        x[i] += h;
        f_x_h = calculate_f(x)
        grad.append((f_x_h - f_value)/h)
        x[i] -= h
    return np.array(grad)

# def grad_f(x:np.array, n:int) -> np.array:
#     h = 1e-7
#     grad = []
#     f_value = calculate_f(x,n)
#     for i in range(n):
#         new_x = [x[i] + h if j == i else x[i] for j in range(n)]
#         f_x_h = calculate_f(new_x,n)
#         grad.append((f_x_h - f_value)/h)
```

```
#     return np.array(grad)


def check_armijo_wolf_cond(x_k: np.array, d_k: np.array, alpha:float,
beta1:float, beta2:float, n:int) -> bool:
    xk_alp_dk = x_k + alpha * d_k;
    f_grad = grad_f(x_k, n);
    f_grad_alph = grad_f(xk_alp_dk,n)
    armijo_left = calculate_f(xk_alp_dk);
    armijo_right = calculate_f(x_k) + alpha * beta1 *np.dot(f_grad.T,d_k);

    wolf_left = np.dot(f_grad_alph.T, d_k)
    wolf_right = beta2 * np.dot(f_grad.T,d_k);


    return (armijo_left <= armijo_right) and (wolf_left >= wolf_right)


def steepest_direction(x_k:np.array, d_k:np.array, beta1: float, beta2:
float, n:int) -> float:
    alpha = 1;
    while not check_armijo_wolf_cond(x_k, d_k, alpha, beta1, beta2, n):
        alpha = alpha * 0.5
        # print(f'current alpha: {alpha}')
    return alpha;

def find_x(x_0:np.array, beta1:float, beta2:float, n:int, epsilon:float) ->
np.array:
    x_k = x_0
    d_k = -grad_f(x_k, n)
    norm_grad = np.linalg.norm(-d_k)
    while norm_grad > epsilon:
        alpha = steepest_direction(x_k, d_k, beta1, beta2, n);
        x_k = x_k + alpha * d_k
        d_k = -grad_f(x_k,n)
        norm_grad = np.linalg.norm(-d_k)

    return x_k

n = 2;
beta1 = 1e-4;
beta2 = 0.9;
epsilon = 0.001
x_0 = np.array([14.0,15.0])

x = find_x(x_0, beta1, beta2, n, 0.001)
print(x)
```