28-11-24
Grey Wolf Algorithm
Algorithm:-

```
import numpy as np

# Sphere function: minimize f(x) = sum(x_i^2)
def sphere_function(position):
    return sum(x**2 for x in position)

# Grey Wolf Optimizer
def grey_wolf_optimizer(dim, wolves_count, iterations, lb, ub):
    """
    GWO implementation to minimize a function.
    dim: Number of dimensions.
    wolves_count: Number of wolves.
    iterations: Number of iterations.
    lb: Lower bound of search space.
    ub: Upper bound of search space.
    """
    # Initialize the positions of wolves randomly
    wolves_positions = np.random.uniform(lb, ub, (wolves_count, dim))

    # Initialize alpha, beta, and delta (best, second-best, and third-best wolves)
    alpha, beta, delta = np.zeros(dim), np.zeros(dim), np.zeros(dim)
    alpha_score, beta_score, delta_score = float("inf"), float("inf"), float("inf")

    # Iterate through generations
    for t in range(iterations):
        for i in range(wolves_count):
            fitness = sphere_function(wolves_positions[i])

            # Update alpha, beta, and delta based on fitness
            if fitness < alpha_score:
                alpha_score, beta_score, delta_score = fitness, alpha_score, beta_score
                alpha, beta, delta = wolves_positions[i], alpha, beta
            elif fitness < beta_score:
                beta_score, delta_score = fitness, beta_score
                beta, delta = wolves_positions[i], beta
            elif fitness < delta_score:
                delta_score = fitness
                delta = wolves_positions[i]

        # Update positions of wolves
```

```
        for i in range(wolves_count):
            for j in range(dim):
                r1, r2 = np.random.rand(), np.random.rand()
                A1, C1 = 2 * r1 - 1, 2 * r2
                D_alpha = abs(C1 * alpha[j] - wolves_positions[i][j])
                X1 = alpha[j] - A1 * D_alpha

                r1, r2 = np.random.rand(), np.random.rand()
                A2, C2 = 2 * r1 - 1, 2 * r2
                D_beta = abs(C2 * beta[j] - wolves_positions[i][j])
                X2 = beta[j] - A2 * D_beta

                r1, r2 = np.random.rand(), np.random.rand()
                A3, C3 = 2 * r1 - 1, 2 * r2
                D_delta = abs(C3 * delta[j] - wolves_positions[i][j])
                X3 = delta[j] - A3 * D_delta

                # Update the wolf's position
                wolves_positions[i][j] = (X1 + X2 + X3) / 3

                # Boundary control
                wolves_positions[i][j] = np.clip(wolves_positions[i][j], lb, ub)

    print("\nOptimization by Devanshi Slathia: Grey Wolf Optimizer completed!")
    return alpha, alpha_score

# Main function
def main():
    print("Grey Wolf Optimizer")
    dim = int(input("Enter the number of dimensions (e.g., 2, 3): "))
    wolves_count = int(input("Enter the number of wolves (e.g., 5, 10): "))
    iterations = int(input("Enter the number of iterations (e.g., 50): "))
    lb, ub = map(float, input("Enter the lower and upper bounds of the search space (e.g., -10
10): ").split())

    # Run GWO
    best_position, best_score = grey_wolf_optimizer(dim, wolves_count, iterations, lb, ub)

    print("\nBest Position Found:", best_position)
    print("Best Score (Minimum Value of Function):", best_score)

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

Output:-

```
Grey Wolf Optimizer
Enter the number of dimensions (e.g., 2, 3): 2
Enter the number of wolves (e.g., 5, 10): 5
Enter the number of iterations (e.g., 50): 50
Enter the lower and upper bounds of the search space (e.g., -10 10): -10 10

Optimization by Devanshi Slathia: Grey Wolf Optimizer completed!

Best Position Found: [-0.02475322  0.0023002 ]
Best Score (Minimum Value of Function): 0.0007324328554199868
```