Program 5 - Grey Wolf Optimizer for Mathematical Function Optimization:

Problem Statement:Design and implement the Grey Wolf Optimizer (GWO) algorithm in Python to maximize a given mathematical function. The algorithm will simulate the social hierarchy and hunting behavior of grey wolves, using alpha, beta, and delta wolves to guide the search for the optimal solution.

Algorithm :

1. **Define the Problem**: Use f(x)=−x2+10x+15f(x) = -x^2 + 10x + 15f(x)=−x2+10x+15 as the objective function.
2. **Initialize Parameters**:
    - Number of wolves NNN
    - Number of iterations III
    - Search space limits
3. **Initialize Population**:
    - Randomly generate the initial positions of NNN wolves within the search space.
4. **Evaluate Fitness**:
    - Compute the fitness of each wolf using f(x)f(x)f(x).
    - Identify the alpha (best), beta (second best), and delta (third best) wolves based on their fitness.
5. **Update Positions**:
    - Use the positions of the alpha, beta, and delta wolves to guide the position updates:
6. **Iterate**:
    - Repeat the evaluation and position update steps for III iterations.
7. **Output the Best Solution**:
    - Track and output the position of the alpha wolf and its fitness.

code:

```
import numpy as np

# Define the fitness function
def fitness_function(x):
    return -x**2 + 10*x + 15

# Grey Wolf Optimizer (GWO) implementation
def grey_wolf_optimizer():
    # Input parameters
    num_wolves = int(input("Enter number of wolves: "))
    num_iterations = int(input("Enter number of iterations: "))
    x_min = float(input("Enter minimum value of x: "))
    x_max = float(input("Enter maximum value of x: "))

    # Initialize positions of wolves
    wolves = np.random.uniform(x_min, x_max, num_wolves)
```

```
    fitness = np.array([fitness_function(x) for x in wolves])

    # Identify alpha, beta, and delta wolves
    alpha, beta, delta = np.argsort(fitness)[-3:]
    alpha_pos, beta_pos, delta_pos = wolves[alpha], wolves[beta],
wolves[delta]

    # Main optimization loop
    for t in range(num_iterations):
        a = 2 - t * (2 / num_iterations)  # Linearly decreasing factor

        for i in range(num_wolves):
            # Update position based on alpha, beta, and delta wolves
            A1, A2, A3 = a * (2 * np.random.rand() - 1), a * (2 *
np.random.rand() - 1), a * (2 * np.random.rand() - 1)
            C1, C2, C3 = 2 * np.random.rand(), 2 * np.random.rand(), 2 *
np.random.rand()

            D_alpha = abs(C1 * alpha_pos - wolves[i])
            D_beta = abs(C2 * beta_pos - wolves[i])
            D_delta = abs(C3 * delta_pos - wolves[i])

            X1 = alpha_pos - A1 * D_alpha
            X2 = beta_pos - A2 * D_beta
            X3 = delta_pos - A3 * D_delta

            wolves[i] = (X1 + X2 + X3) / 3
            wolves[i] = np.clip(wolves[i], x_min, x_max)  # Keep within
bounds

        # Recalculate fitness and update alpha, beta, delta
        fitness = np.array([fitness_function(x) for x in wolves])
        alpha, beta, delta = np.argsort(fitness)[-3:]
        alpha_pos, beta_pos, delta_pos = wolves[alpha], wolves[beta],
wolves[delta]

    return alpha_pos, fitness_function(alpha_pos)

# Run the Grey Wolf Optimizer
best_solution, best_value = grey_wolf_optimizer()
print(f"Best Solution: {best_solution}, Fitness: {best_value}")
```

```
print("Name-pooja Gaikwad(1BM22CS194)")
```

Output:

```
Enter number of wolves: 5
Enter number of iterations: 50
Enter minimum value of x: 0
Enter maximum value of x: 10
Best Solution: 5.029433041367449, Fitness: 39.99913369607586
Name-pooja Gaikwad(1BM22CS194)
```

Observation:

⑥ Grey Wolf Optimization -

1. Start

2. Initialize :
   N ← Population size (no. of wolves)
   max-iter ← max. no. of iter.

3. Calculate 3 top solution :

   α (alpha) : best solution (leader)
   β (beta) : second best
   δ (delta) : third best

4. For iteration 1 to max :
   for each wolf 1 to N :
   update position
   1. Calculate distance to α, β, δ
   D-α = | C-1 * X-α - Xi |
   D-β = | C-2 * X-β - Xi |
   D-δ = | C-3 * X-δ - Xi |
   2. Update pos.
   $X_i = (X_\alpha - A_1 * D_{-\alpha}) + (X_\beta - A_2 * D_{-\beta}) + (X_\delta - A_3 * D_\delta)$

5. Calculate fitness of updated pos.
   fitness (X-i) = evaluate fitness (X-i)

6. Update α, β, δ pos. based on fitness.

7. END

8. Return α as best solution