**CODE :**

```
import numpy as np

# Traffic congestion function (Objective Function)
def traffic_objective(signal_timings):
    """
    Simulates traffic congestion based on signal timings.
    Lower values represent better performance (less congestion).
    Example: The function is simplified and can be replaced with real traffic data.
    """
    total_waiting_time = 0
    for timing in signal_timings:
        total_waiting_time += abs(50 - timing) + np.random.uniform(0, 5)  # Example
penalty model
    return total_waiting_time

class GreyWolfOptimizer:
    def __init__(self, num_wolves, num_iterations, bounds):
        self.num_wolves = num_wolves
        self.num_iterations = num_iterations
        self.bounds = bounds
        self.dimensions = len(bounds)  # Number of traffic signals
        self.wolves = np.random.uniform(
            [b[0] for b in bounds], [b[1] for b in bounds], (num_wolves,
self.dimensions)
        )
        self.alpha = None
        self.beta = None
        self.delta = None
        self.alpha_score = float("inf")
        self.beta_score = float("inf")
        self.delta_score = float("inf")

    def evaluate_fitness(self, wolf):
        return traffic_objective(wolf)
```

```
def update_leadership(self):
    for wolf in self.wolves:
        fitness = self.evaluate_fitness(wolf)
        if fitness < self.alpha_score:
            self.delta_score, self.delta = self.beta_score, self.beta
            self.beta_score, self.beta = self.alpha_score, self.alpha
            self.alpha_score, self.alpha = fitness, wolf
        elif fitness < self.beta_score:
            self.delta_score, self.delta = self.beta_score, self.beta
            self.beta_score, self.beta = fitness, wolf
        elif fitness < self.delta_score:
            self.delta_score, self.delta = fitness, wolf

def update_positions(self, iteration):
    a = 2 - iteration * (2 / self.num_iterations)  # Linearly decreases from 2 to 0

    for i in range(self.num_wolves):
        for j in range(self.dimensions):
            r1, r2 = np.random.random(), np.random.random()
            A1, C1 = 2 * a * r1 - a, 2 * r2
            D_alpha = abs(C1 * self.alpha[j] - self.wolves[i][j])
            X1 = self.alpha[j] - A1 * D_alpha

            r1, r2 = np.random.random(), np.random.random()
            A2, C2 = 2 * a * r1 - a, 2 * r2
            D_beta = abs(C2 * self.beta[j] - self.wolves[i][j])
            X2 = self.beta[j] - A2 * D_beta

            r1, r2 = np.random.random(), np.random.random()
            A3, C3 = 2 * a * r1 - a, 2 * r2
            D_delta = abs(C3 * self.delta[j] - self.wolves[i][j])
            X3 = self.delta[j] - A3 * D_delta

            # Update position
```

```
            self.wolves[i][j] = (X1 + X2 + X3) / 3

        # Ensure wolves stay within bounds
        self.wolves[i] = np.clip(self.wolves[i], [b[0] for b in self.bounds], [b[1] for
b in self.bounds])

  def optimize(self):
     self.update_leadership()  # Initialize alpha, beta, and delta

     for iteration in range(self.num_iterations):
        self.update_positions(iteration)
        self.update_leadership()
        print(f"Iteration {iteration + 1} Error: {self.alpha_score:.4f}")

     return self.alpha, self.alpha_score


# Parameters
num_signals = 5  # Number of traffic signals
bounds = [(20, 100)] * num_signals  # Traffic light timings (min, max) in seconds
num_wolves = 20
num_iterations = 5

# Run Grey Wolf Optimizer
gwo = GreyWolfOptimizer(num_wolves, num_iterations, bounds)
best_solution, best_fitness = gwo.optimize()

# Output final results
print("\nOptimal: ", best_solution)
```

**OUTPUT:**

```
Iteration 1 Error: 63.8191
Iteration 2 Error: 63.8191
Iteration 3 Error: 63.8191
Iteration 4 Error: 63.8191
Iteration 5 Error: 63.8191

Optimal:  [90.1193826  97.82522127 56.86934156 36.4752775  64.60604421]
```