Program 5: Grey Wolf Optimizer (GWO)

Code:

```
import numpy as np

def initialize_wolves(search_space, num_wolves):
    dimensions = len(search_space)
    wolves = np.zeros((num_wolves, dimensions))
    for i in range(num_wolves):
        wolves[i] = np.random.uniform(search_space[:, 0], search_space[:, 1])
    return wolves

def fitness_function(x):
    # Define your fitness function to evaluate the quality of a solution
    # Example: Sphere function (minimize the sum of squares)
    return np.sum(x**2)

def gwo_algorithm(search_space, num_wolves, max_iterations):
    dimensions = len(search_space)

    # Initialize wolves
    wolves = initialize_wolves(search_space, num_wolves)

    # Initialize alpha, beta, and gamma wolves
    alpha_wolf = np.zeros(dimensions)
    beta_wolf = np.zeros(dimensions)
    gamma_wolf = np.zeros(dimensions)

    # Initialize the fitness of alpha, beta, gamma wolves
    alpha_fitness = float('inf')
    beta_fitness = float('inf')
    gamma_fitness = float('inf')

    # Store the best fitness found
    best_fitness = float('inf')

    for iteration in range(max_iterations):
        a = 2 - (iteration / max_iterations) * 2  # Parameter a decreases linearly from 2 to 0

        #print(f"Iteration {iteration + 1}/{max_iterations}")
```

```
# Evaluate the fitness of all wolves
for i in range(num_wolves):
    fitness = fitness_function(wolves[i])

    # Print the fitness of the current wolf
   # print(f"Wolf {i+1} Fitness: {fitness}")

    # Update alpha, beta, gamma wolves based on fitness
    if fitness < alpha_fitness:
        gamma_wolf = beta_wolf.copy()
        gamma_fitness = beta_fitness
        beta_wolf = alpha_wolf.copy()
        beta_fitness = alpha_fitness
        alpha_wolf = wolves[i].copy()
        alpha_fitness = fitness
    elif fitness < beta_fitness:
        gamma_wolf = beta_wolf.copy()
        gamma_fitness = beta_fitness
        beta_wolf = wolves[i].copy()
        beta_fitness = fitness
    elif fitness < gamma_fitness:
        gamma_wolf = wolves[i].copy()
        gamma_fitness = fitness

# Print the best fitness for this iteration
#print(f"Best Fitness in this Iteration: {alpha_fitness}")

# Store the best overall fitness found so far
if alpha_fitness < best_fitness:
    best_fitness = alpha_fitness

# Update positions of wolves
for i in range(num_wolves):
    for j in range(dimensions):
        r1 = np.random.random()
        r2 = np.random.random()

        A1 = 2 * a * r1 - a
        C1 = 2 * r2
```

```
                D_alpha = np.abs(C1 * alpha_wolf[j] - wolves[i, j])
                X1 = alpha_wolf[j] - A1 * D_alpha

                r1 = np.random.random()
                r2 = np.random.random()

                A2 = 2 * a * r1 - a
                C2 = 2 * r2

                D_beta = np.abs(C2 * beta_wolf[j] - wolves[i, j])
                X2 = beta_wolf[j] - A2 * D_beta

                r1 = np.random.random()
                r2 = np.random.random()

                A3 = 2 * a * r1 - a
                C3 = 2 * r2

                D_gamma = np.abs(C3 * gamma_wolf[j] - wolves[i, j])
                X3 = gamma_wolf[j] - A3 * D_gamma

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

                # Ensure the new position is within the search space bounds
                wolves[i, j] = np.clip(wolves[i, j], search_space[j, 0], search_space[j, 1])

    print(f"Optimal Solution Found: {alpha_wolf}")
    print(f"Optimal Fitness: {best_fitness}")
    return alpha_wolf  # Return the best solution found

# Example usage
search_space = np.array([[-5, 5], [-5, 5]])  # Define the search space for the optimization problem
num_wolves = 10  # Number of wolves in the pack
max_iterations = 100  # Maximum number of iterations

# Run the GWO algorithm
optimal_solution = gwo_algorithm(search_space, num_wolves, max_iterations)
```

# Print the optimal solution
print("Optimal Solution:", optimal_solution)

Output:

```
Optimal Solution Found: [ 1.51778516e-13 -1.31752029e-13]
Optimal Fitness: 4.0395315250402290e-26
Optimal Solution: [ 1.51778516e-13 -1.31752029e-13]
```