Lab-7-Optimization via Gene Expression Algorithms

Code:

```
import numpy as np

# Step 1: Define the Problem
# Define your mathematical optimization function here
# Example: Minimize the function f(x) = x^2 - 4x + 4 (a simple quadratic function)
def objective_function(x):
    return x**2 - 4*x + 4

# Step 2: Initialize Parameters
def initialize_parameters():
    population_size = int(input("Enter population size: "))  # Number of individuals in the
population
    num_genes = int(input("Enter number of genes: "))  # Number of genes in each individual
    mutation_rate = float(input("Enter mutation rate (0 to 1): "))  # Probability of mutation
    crossover_rate = float(input("Enter crossover rate (0 to 1): "))  # Probability of crossover
    num_generations = int(input("Enter number of generations: "))  # Number of generations
    return population_size, num_genes, mutation_rate, crossover_rate, num_generations

# Step 3: Initialize Population
def initialize_population(population_size, num_genes):
    population = np.random.uniform(low=-5, high=5, size=(population_size, num_genes))
    return population

# Step 4: Evaluate Fitness
def evaluate_fitness(population, objective_function):
    fitness = np.apply_along_axis(objective_function, 1, population)  # Apply the objective
function to each individual
    return fitness

# Step 5: Selection (Tournament Selection)
def selection(population, fitness, num_parents):
    parents = []
    for _ in range(num_parents):
        tournament = np.random.choice(population.shape[0], size=3, replace=False)  # Select 3
individuals for tournament
        tournament_fitness = fitness[tournament]
```

```
        winner = tournament[np.argmin(tournament_fitness)]  # Select the individual with the best
fitness
        parents.append(population[winner])
    return np.array(parents)

# Step 6: Crossover (Single-point crossover)
def crossover(parents, crossover_rate):
    num_parents = parents.shape[0]
    offspring = []
    for i in range(0, num_parents, 2):
        if np.random.rand() < crossover_rate:
            # Ensure that we have more than 1 gene to perform crossover
            if parents.shape[1] > 1:
                crossover_point = np.random.randint(1, parents.shape[1])
                offspring1 = np.concatenate([parents[i, :crossover_point], parents[i+1,
crossover_point:]])
                offspring2 = np.concatenate([parents[i+1, :crossover_point], parents[i,
crossover_point:]])
                offspring.append(offspring1)
                offspring.append(offspring2)
            else:
                # No crossover if there's only 1 gene
                offspring.append(parents[i])
                if i + 1 < num_parents:
                    offspring.append(parents[i + 1])
        else:
            offspring.append(parents[i])
            if i + 1 < num_parents:
                offspring.append(parents[i + 1])
    return np.array(offspring)

# Step 7: Mutation
def mutation(offspring, mutation_rate):
    for i in range(offspring.shape[0]):
        for j in range(offspring.shape[1]):
            if np.random.rand() < mutation_rate:
                offspring[i, j] += np.random.normal(0, 0.1)  # Apply Gaussian mutation
    return offspring

# Step 8: Gene Expression (Translate Genes into Solutions)
```

```
def gene_expression(offspring):
    # In this case, the genes are directly the solutions
    return offspring

# Step 9: Iterate (Repeat the selection, crossover, mutation, and gene expression processes)
def run_ge_algorithm(objective_function):
    population_size, num_genes, mutation_rate, crossover_rate, num_generations =
initialize_parameters()

    # Initialize population
    population = initialize_population(population_size, num_genes)

    # Start optimization process
    best_solution = None
    best_fitness = float('inf')
    for generation in range(num_generations):
        # Evaluate fitness
        fitness = evaluate_fitness(population, objective_function)

        # Track the best solution
        min_fitness_idx = np.argmin(fitness)
        if fitness[min_fitness_idx] < best_fitness:
            best_fitness = fitness[min_fitness_idx]
            best_solution = population[min_fitness_idx]

        # Selection
        parents = selection(population, fitness, population_size // 2)

        # Crossover
        offspring = crossover(parents, crossover_rate)

        # Mutation
        offspring = mutation(offspring, mutation_rate)

        # Gene Expression (Directly apply the offspring as solutions)
        population = gene_expression(offspring)

        # Print the progress
        print(f"Generation {generation + 1}: Best Fitness = {best_fitness}")
```

```
    return best_solution, best_fitness

# Step 10: Output the Best Solution
best_solution, best_fitness = run_ge_algorithm(objective_function)
print(f"Best solution found: {best_solution}")
print(f"Best fitness value: {best_fitness}")
```

Output:

```
Enter population size: 20
Enter number of genes: 1
Enter mutation rate (0 to 1): 0.1
Enter crossover rate (0 to 1): 0.9
Enter number of generations: 9
Generation 1: Best Fitness = [0.00134711]
Generation 2: Best Fitness = [0.00134711]
Generation 3: Best Fitness = [0.00134711]
Generation 4: Best Fitness = [0.00134711]
Generation 5: Best Fitness = [0.00134711]
Generation 6: Best Fitness = [0.00134711]
Generation 7: Best Fitness = [0.00134711]
Generation 8: Best Fitness = [0.00134711]
Generation 9: Best Fitness = [0.00134711]
Best solution found: [2.03670296]
Best fitness value: [0.00134711]
```