LAB-07 Optimization via Gene Expression Algorithm

```
import numpy as np

# Step 1: Define the Problem (Objective Function)
def objective_function(x):
    return -x**2 + 4*x  # Example: Maximize f(x) = -x^2 + 4x

# Step 2: Initialize Parameters
population_size = 10
num_genes = 1  # Single-variable problem (x)
mutation_rate = 0.1
crossover_rate = 0.8
num_generations = 50
lower_bound, upper_bound = 0, 5  # Search space for x

# Step 3: Initialize Population
def initialize_population(size, bounds, num_genes):
    return np.random.uniform(bounds[0], bounds[1], (size, num_genes))

# Step 4: Evaluate Fitness
def evaluate_fitness(population):
    return np.array([objective_function(ind[0]) for ind in population])

# Step 5: Selection (Roulette Wheel Selection with Non-Negative Fitness)
def select_parents(population, fitness):
    # Shift fitness to be non-negative
    fitness_shifted = fitness - fitness.min() + 1e-6  # Ensure all values
are positive
    probabilities = fitness_shifted / fitness_shifted.sum()  # Normalize
probabilities
    indices = np.random.choice(len(population), size=len(population),
p=probabilities)
    return population[indices]
```

```python
# Step 6: Crossover
def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        alpha = np.random.rand()  # Weighted average for crossover
        child1 = alpha * parent1 + (1 - alpha) * parent2
        child2 = alpha * parent2 + (1 - alpha) * parent1
        return child1, child2
    return parent1, parent2

# Step 7: Mutation
def mutate(individual, bounds):
    if np.random.rand() < mutation_rate:
        gene = np.random.randint(len(individual))  # Randomly select gene
to mutate
        individual[gene] = np.random.uniform(bounds[0], bounds[1])
    return individual

# Step 8: Gene Expression (Direct Mapping of Genes to Solutions)

# Step 9: Iterate
def gene_expression_algorithm():
    population = initialize_population(population_size, (lower_bound,
upper_bound), num_genes)
    best_solution = None
    best_fitness = -np.inf

    for generation in range(num_generations):
        fitness = evaluate_fitness(population)
        best_idx = np.argmax(fitness)

        # Update the best solution
        if fitness[best_idx] > best_fitness:
            best_fitness = fitness[best_idx]
            best_solution = population[best_idx].copy()

        # Selection
        parents = select_parents(population, fitness)

        # Crossover and Mutation
```

```
        offspring = []
        for i in range(0, len(parents), 2):
            parent1 = parents[i]
            parent2 = parents[(i + 1) % len(parents)]  # Wrap-around for
last parent
            child1, child2 = crossover(parent1, parent2)
            offspring.append(mutate(child1, (lower_bound, upper_bound)))
            offspring.append(mutate(child2, (lower_bound, upper_bound)))

        # Update the population
        population = np.array(offspring)

    return best_solution, best_fitness

# Step 10: Output the Best Solution
best_solution, best_fitness = gene_expression_algorithm()
print(f"Best Solution: {best_solution[0]:.4f}, Best Fitness:
{best_fitness:.4f}")
```

Output

```
Best Solution: 1.9999, Best Fitness: 4.0000
```