Lab-1-Genetic Algorithm for Optimization Problems

Code:

```
import random
def fitness_function(x):
    return x**2
population_size = 500
mutation_rate = 0.01
crossover_rate = 0.7
generations = 5
lower_bound = -10
upper_bound = 10
def create_population():
    population = [random.uniform(lower_bound, upper_bound) for _ in range(population_size)]
    return population
def evaluate_fitness(population):
    fitness_scores = [fitness_function(individual) for individual in population]
    return fitness_scores
def select(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    selection_probs = [score / total_fitness for score in fitness_scores]
    selected = random.choices(population, weights=selection_probs, k=2)
    return selected
def crossover(parent1, parent2):
    if random.random() < crossover_rate:
        # Crossover at a single point (since we're dealing with floats, we'll use averaging)
        child1 = (parent1 + parent2) / 2
        child2 = (parent1 + parent2) / 2
    else:
        child1, child2 = parent1, parent2
    return child1, child2

def mutate(individual):
    if random.random() < mutation_rate:
        mutation_value = random.uniform(-1, 1)
        individual += mutation_value
    return max(min(individual, upper_bound), lower_bound)
def genetic_algorithm():
    population = create_population()
```

```
    for generation in range(generations):
        fitness_scores = evaluate_fitness(population)
        best_fitness = max(fitness_scores)
        best_individual = population[fitness_scores.index(best_fitness)]
        new_population = []
        while len(new_population) < population_size:
            parent1, parent2 = select(population, fitness_scores)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1)
            child2 = mutate(child2)
            new_population.append(child1)
            if len(new_population) < population_size:
                new_population.append(child2)
        population = new_population
        print(f"Generation {generation + 1}: Best Fitness = {best_fitness}, Best Individual =
{best_individual}")
    return best_individual, best_fitness
best_solution, best_fitness = genetic_algorithm()
print(f"\nBest solution found: {best_solution} with fitness: {best_fitness}")
```

Output:

```
Generation 1: Best Fitness = 99.82898901507124, Best Individual = 9.991445792029863
Generation 2: Best Fitness = 99.82898901507124, Best Individual = 9.991445792029863
Generation 3: Best Fitness = 91.2968572320205, Best Individual = 9.554938892113361
Generation 4: Best Fitness = 81.45529573357616, Best Individual = 9.02525876269352
Generation 5: Best Fitness = 75.10066143216397, Best Individual = 8.666063779603977

Best solution found: 8.666063779603977 with fitness: 75.10066143216397
```