LAB 1
GENETIC ALGORITHM FOR OPTIMIZATION PROBLEM

1BM22CS263
SHREE VARNA M

CODE:

```
import numpy as np
import random

# Define the fitness function
def fitness_function(x):
    return x ** 2

# Initialize parameters
population_size = 100
mutation_rate = 0.1
num_generations = 50
bounds = (-10, 10)

# Step 1: Create initial population
def create_initial_population(size, bounds):
    return [random.uniform(bounds[0], bounds[1]) for _ in
range(size)]

# Step 2: Evaluate fitness of the population
def evaluate_population(population):
    return [fitness_function(individual) for individual in
population]

# Step 3: Selection using roulette-wheel selection
def selection(population, fitness):
    total_fitness = sum(fitness)
    selection_probs = [f / total_fitness for f in fitness]
    return np.random.choice(population, size=2, p=selection_probs)

# Step 4: Crossover operation
def crossover(parent1, parent2):
    alpha = random.uniform(0, 1)
    offspring1 = alpha * parent1 + (1 - alpha) * parent2
    offspring2 = alpha * parent2 + (1 - alpha) * parent1
    return offspring1, offspring2

# Step 5: Mutation operation
```

```
def mutate(individual, bounds):
    if random.random() < mutation_rate:
        return random.uniform(bounds[0], bounds[1])
    return individual

# Main Genetic Algorithm loop
def genetic_algorithm(bounds):
    # Step 1: Create initial population
    population = create_initial_population(population_size, bounds)

    best_solution = None
    best_fitness = float('-inf')

    for generation in range(num_generations):
        # Step 2: Evaluate fitness
        fitness = evaluate_population(population)

        # Track the best solution
        current_best_fitness = max(fitness)
        if current_best_fitness > best_fitness:
            best_fitness = current_best_fitness
            best_solution =
population[fitness.index(current_best_fitness)]

        # Step 3: Create new population
        new_population = []

        while len(new_population) < population_size:
            parent1, parent2 = selection(population, fitness)
            offspring1, offspring2 = crossover(parent1, parent2)
            new_population.append(mutate(offspring1, bounds))
            new_population.append(mutate(offspring2, bounds))

        # Replace the old population with the new population
        population = new_population[:population_size]

    return best_solution, best_fitness

# Run the Genetic Algorithm
best_solution, best_fitness = genetic_algorithm(bounds)

print(f"Best Solution: x = {best_solution}")
print(f"Best Fitness: f(x) = {best_fitness}")
```

OUTPUT :

```
Best Solution: x = -9.992054162496341
Best Fitness: f(x) = 99.84114638626046
```