BIS LAB

LAB 1

Genetic Algorithm for Optimization Problems:

Genetic Algorithms (GA) are inspired by the process of natural selection and genetics, where the fittest individuals are selected for reproduction to produce the next generation. GAs are widely used for solving optimization and search problems. Implement a Genetic Algorithm using Python to solve a basic optimization problem, such as finding the maximum value of a mathematical function.

PYTHON CODE:

Genetic Algorithm to **find the maximum value of a function** (for example, f(x)=xsin⁡(10πx)+1.0f(x) = x \sin(10\pi x) + 1.0f(x)=xsin(10πx)+1.0 in the range [0, 1]).

```
import random
import math

# 1. Define the problem — function to optimize
def fitness_function(x):
    # Example function: f(x) = x * sin(10πx) + 1.0 (maximize this)
    return x * math.sin(10 * math.pi * x) + 1.0

# 2. Initialize Parameters
POP_SIZE = 20          # Number of individuals in population
GENES = 16             # Number of bits to represent a chromosome
MUTATION_RATE = 0.01   # Probability of mutation
CROSSOVER_RATE = 0.7   # Probability of crossover
GENERATIONS = 50       # Number of generations
X_MIN, X_MAX = 0, 1    # Range of x values

# Helper functions
def decode(chromosome):
    """Convert binary chromosome to a real value in range [X_MIN,
X_MAX]"""
    decimal_value = int(chromosome, 2)
    return X_MIN + (decimal_value / (2**GENES - 1)) * (X_MAX - X_MIN)

def create_individual():
    """Generate a random binary string (chromosome)"""
    return ''.join(random.choice('01') for _ in range(GENES))

def crossover(parent1, parent2):
    """Single-point crossover"""
    if random.random() < CROSSOVER_RATE:
        point = random.randint(1, GENES - 1)
```

```
        return parent1[:point] + parent2[point:], parent2[:point] +
parent1[point:]
    return parent1, parent2

def mutate(chromosome):
    """Random bit flip mutation"""
    chromosome = list(chromosome)
    for i in range(GENES):
        if random.random() < MUTATION_RATE:
            chromosome[i] = '1' if chromosome[i] == '0' else '0'
    return ''.join(chromosome)

# 3. Create initial population
population = [create_individual() for _ in range(POP_SIZE)]

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

# 4-8. Main GA loop
for generation in range(GENERATIONS):
    # Evaluate fitness for each individual
    fitness_scores = []
    for individual in population:
        x = decode(individual)
        fitness = fitness_function(x)
        fitness_scores.append(fitness)

        if fitness > best_fitness:
            best_fitness = fitness
            best_solution = individual

    # Print progress
    print(f"Generation {generation + 1} | Best Fitness:
{best_fitness:.5f}")

    # 5. Selection (Roulette Wheel Selection)
    total_fitness = sum(fitness_scores)
    probabilities = [f / total_fitness for f in fitness_scores]

    def select_parent():
        r = random.random()
        cumulative = 0
        for i, prob in enumerate(probabilities):
            cumulative += prob
            if r <= cumulative:
                return population[i]

    # 6. Crossover and 7. Mutation
```

```
    new_population = []
    while len(new_population) < POP_SIZE:
        parent1 = select_parent()
        parent2 = select_parent()
        offspring1, offspring2 = crossover(parent1, parent2)
        new_population.append(mutate(offspring1))
        if len(new_population) < POP_SIZE:
            new_population.append(mutate(offspring2))

    population = new_population

# 9. Output the best solution
best_x = decode(best_solution)
print("\n Best Solution Found:")
print(f"x = {best_x:.5f}")
print(f"Fitness = {best_fitness:.5f}")
```

```
Generation 1 | Best Fitness: 1.67253
Generation 2 | Best Fitness: 1.67253
Generation 3 | Best Fitness: 1.78018
Generation 4 | Best Fitness: 1.80517
Generation 5 | Best Fitness: 1.80530
Generation 6 | Best Fitness: 1.80530
Generation 7 | Best Fitness: 1.80530
Generation 8 | Best Fitness: 1.85035
Generation 9 | Best Fitness: 1.85035
Generation 10 | Best Fitness: 1.85035
Generation 11 | Best Fitness: 1.85035
Generation 12 | Best Fitness: 1.85035
Generation 13 | Best Fitness: 1.85035
Generation 14 | Best Fitness: 1.85035
Generation 15 | Best Fitness: 1.85035
Generation 16 | Best Fitness: 1.85035
Generation 17 | Best Fitness: 1.85035
Generation 18 | Best Fitness: 1.85035
Generation 19 | Best Fitness: 1.85035
Generation 20 | Best Fitness: 1.85035
Generation 21 | Best Fitness: 1.85035
Generation 22 | Best Fitness: 1.85035
Generation 23 | Best Fitness: 1.85035
Generation 24 | Best Fitness: 1.85035
Generation 25 | Best Fitness: 1.85035
Generation 26 | Best Fitness: 1.85035
Generation 27 | Best Fitness: 1.85035
Generation 28 | Best Fitness: 1.85035
Generation 29 | Best Fitness: 1.85035
Generation 30 | Best Fitness: 1.85035
Generation 31 | Best Fitness: 1.85035
Generation 32 | Best Fitness: 1.85035
Generation 33 | Best Fitness: 1.85035
Generation 34 | Best Fitness: 1.85035
Generation 35 | Best Fitness: 1.85035
Generation 36 | Best Fitness: 1.85035
Generation 37 | Best Fitness: 1.85035
Generation 38 | Best Fitness: 1.85035
Generation 39 | Best Fitness: 1.85035
Generation 40 | Best Fitness: 1.85035
Generation 41 | Best Fitness: 1.85035
Generation 42 | Best Fitness: 1.85035
Generation 43 | Best Fitness: 1.85035
Generation 44 | Best Fitness: 1.85035
Generation 45 | Best Fitness: 1.85035
Generation 46 | Best Fitness: 1.85035
Generation 47 | Best Fitness: 1.85035
Generation 48 | Best Fitness: 1.85035
Generation 49 | Best Fitness: 1.85035
Generation 50 | Best Fitness: 1.85035

 Best Solution Found:
x = 0.85196
Fitness = 1.85035
```