BIS LAB

LAB 2

Optimization via Gene Expression Algorithms:

Gene Expression Algorithms (GEA) are inspired by the biological process of gene expression in living organisms. This process involves the translation of genetic information encoded in DNA into functional proteins. In GEA, solutions to optimization problems are encoded in a manner similar to genetic sequences. The algorithm evolves these solutions through selection, crossover, mutation, and gene expression to find optimal or near-optimal solutions. GEA is effective for solving complex optimization problems in various domains, including engineering, data analysis, and machine learning.

PYTHON CODE:

Gene Expression Algorithm (GEA) in Python for a simple **function optimization problem** — say maximizing f(x)=xsin(10πx)+1.0f(x) = x \sin(10\pi x) + 1.0f(x)=xsin(10πx)+1.0.

```
import random

import math

def objective_function(x):

    """Example mathematical function to maximize."""

    return x * math.sin(10 * math.pi * x) + 1.0

POP_SIZE = 20            # Population size

NUM_GENES = 16           # Number of genes per chromosome

MUTATION_RATE = 0.05     # Mutation probability

CROSSOVER_RATE = 0.7     # Crossover probability

GENERATIONS = 50         # Number of generations

X_MIN, X_MAX = 0, 1      # Search range



def create_gene():

    """Each gene is a binary value (0 or 1)."""

    return random.choice(['0', '1'])


def create_chromosome():

    """A chromosome is a list of genes."""

    return [create_gene() for _ in range(NUM_GENES)]
```

```python
def create_population():
    """Generate initial population."""
    return [create_chromosome() for _ in range(POP_SIZE)]
def decode(chromosome):
    """Translate genes into a real number (phenotype)."""
    value = int(''.join(chromosome), 2)
    return X_MIN + (value / (2**NUM_GENES - 1)) * (X_MAX - X_MIN)

def fitness(chromosome):
    """Compute the fitness value of an individual."""
    x = decode(chromosome)
    return objective_function(x)
def select_parent(population, fitnesses):
    total_fit = sum(fitnesses)
    pick = random.uniform(0, total_fit)
    current = 0
    for i, f in enumerate(fitnesses):
        current += f
        if current > pick:
            return population[i]
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        point = random.randint(1, NUM_GENES - 1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
        return child1, child2
    return parent1[:], parent2[:]

def mutate(chromosome):
    for i in range(NUM_GENES):
        if random.random() < MUTATION_RATE:
            chromosome[i] = '1' if chromosome[i] == '0' else '0'
```

```
    return chromosome

# 8. Gene Expression (Translate Genotype → Phenotype)
def express(chromosome):
    """Expression phase maps genetic code to actual function value."""
    x = decode(chromosome)
    return x, objective_function(x)

# 9. Iterate through Generations
population = create_population()
best_solution = None
best_fitness = float('-inf')

for gen in range(GENERATIONS):
    fitnesses = [fitness(individual) for individual in population]

    #  for i, f in enumerate(fitnesses):
        if f > best_fitness:
            best_fitness = f
            best_solution = population[i]

    print(f"Generation {gen + 1} | Best Fitness: {best_fitness:.5f}")

    new_population = []
    while len(new_population) < POP_SIZE:
        parent1 = select_parent(population, fitnesses)
        parent2 = select_parent(population, fitnesses)
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1)
        child2 = mutate(child2)
        new_population.extend([child1, child2])

    population = new_population[:POP_SIZE]
```

```
# 10. Output the Best Solution
best_x, best_y = express(best_solution)
print("\n Best Solution Found:")
print(f"x = {best_x:.5f}")
print(f"Fitness = {best_y:.5f}")
```

OUTPUT:

```
Generation 1 | Best Fitness: 1.62960
Generation 2 | Best Fitness: 1.62960
Generation 3 | Best Fitness: 1.84872
Generation 4 | Best Fitness: 1.84913
Generation 5 | Best Fitness: 1.85005
Generation 6 | Best Fitness: 1.85005
Generation 7 | Best Fitness: 1.85005
Generation 8 | Best Fitness: 1.85005
Generation 9 | Best Fitness: 1.85005
Generation 10 | Best Fitness: 1.85005
Generation 11 | Best Fitness: 1.85059
Generation 12 | Best Fitness: 1.85059
Generation 13 | Best Fitness: 1.85059
Generation 14 | Best Fitness: 1.85059
Generation 15 | Best Fitness: 1.85059
Generation 16 | Best Fitness: 1.85059
Generation 17 | Best Fitness: 1.85059
Generation 18 | Best Fitness: 1.85059
Generation 19 | Best Fitness: 1.85059
Generation 20 | Best Fitness: 1.85059
Generation 21 | Best Fitness: 1.85059
Generation 22 | Best Fitness: 1.85059
Generation 23 | Best Fitness: 1.85059
Generation 24 | Best Fitness: 1.85059
Generation 25 | Best Fitness: 1.85059
Generation 26 | Best Fitness: 1.85059
Generation 27 | Best Fitness: 1.85059
Generation 28 | Best Fitness: 1.85059
Generation 29 | Best Fitness: 1.85059
Generation 30 | Best Fitness: 1.85059
Generation 31 | Best Fitness: 1.85059
Generation 32 | Best Fitness: 1.85059
Generation 33 | Best Fitness: 1.85059
Generation 34 | Best Fitness: 1.85059
Generation 35 | Best Fitness: 1.85059
Generation 36 | Best Fitness: 1.85059
Generation 37 | Best Fitness: 1.85059
Generation 38 | Best Fitness: 1.85059
Generation 39 | Best Fitness: 1.85059
Generation 40 | Best Fitness: 1.85059
Generation 41 | Best Fitness: 1.85059
Generation 42 | Best Fitness: 1.85059
Generation 43 | Best Fitness: 1.85059
Generation 44 | Best Fitness: 1.85059
Generation 45 | Best Fitness: 1.85059
Generation 46 | Best Fitness: 1.85059
Generation 47 | Best Fitness: 1.85059
Generation 48 | Best Fitness: 1.85059
Generation 49 | Best Fitness: 1.85059
Generation 50 | Best Fitness: 1.85059

🧠 Best Solution Found:
x = 0.85122
Fitness = 1.85059
```