USN: 1BM22CS253

LAB-7: Op miza on via Gene Expression Algorithms

CODE:

```
import numpy as np import
random


# 1. Define the Problem: Op miza on Func on (e.g., Sphere Func on) def
op miza on_func on(solu on):
    """Sphere Func on for minimiza on (fitness evalua on)."""
return sum(x**2 for x in solu on)


# 2. Ini alize Parameters
POPULATION_SIZE = 50  # Number of gene c sequences (solu ons)
GENES = 5  # Number of genes per solu on
MUTATION_RATE = 0.1  # Probability of muta on
CROSSOVER_RATE = 0.7  # Probability of crossover
GENERATIONS = 30  # Number of genera ons to evolve

# 3. Ini alize Popula on def ini
alize_popula on(pop_size, genes):
    """Generate ini al popula on of random gene c sequences."""     return
np.random.uniform(-10, 10, (pop_size, genes))


# 4. Evaluate Fitness def
evaluate_fitness(popula on):
    """Evaluate the fitness of each gene c sequence."""     fitness = [op
miza on_func on(solu on) for solu on in popula on]
```

```
    return np.array(fitness)


# 5. Selec on: Tournament Selec on
def select_parents(popula on, fitness, num_parents):
"""Select parents using tournament selec on."""
parents = []     for _ in range(num_parents):

        tournament = random.sample(range(len(popula on)), 3)  # Randomly select 3 candidates
best = min(tournament, key=lambda idx: fitness[idx])          parents.append(popula on[best])
return np.array(parents)



# 6. Crossover: Single-Point Crossover
def crossover(parents, crossover_rate):

    """Perform crossover between pairs of parents."""
offspring = []     for i in range(0, len(parents), 2):

        if i + 1 >= len(parents):
            break         parent1, parent2 = parents[i],
parents[i + 1]         if random.random() <
crossover_rate:

            point = random.randint(1, len(parent1) - 1)  # Single crossover
point             child1 = np.concatenate((parent1[:point], parent2[point:]))
child2 = np.concatenate((parent2[:point], parent1[point:]))

        else:
            child1, child2 = parent1, parent2  # No
crossover         offspring.extend([child1, child2])
return np.array(offspring)



# 7. Muta on def mutate(offspring, muta
on_rate):     """Apply muta on to introduce
```

```
variability."""     for i in range(len(offspring)):
for j in range(len(offspring[i])):
            if random.random() < muta on_rate:
                offspring[i][j] += np.random.uniform(-1, 1)  # Random small change
return offspring


# 8. Gene Expression: Func onal Solu on (No transforma on needed for this case) def
gene_expression(popula on):

    """Translate gene c sequences into func onal solu ons."""     return
popula on  # Gene c sequences directly represent solu ons here.


# 9. Main Func on: Gene Expression Algorithm
def gene_expression_algorithm():

    """Implementa on of Gene Expression Algorithm for op miza on."""
    #  Ini  alize  popula  on           popula  on  =  ini  alize_popula
on(POPULATION_SIZE, GENES)     best_solu on = None

    best_fitness = float('inf')


    for genera on in range(GENERATIONS):
        # Evaluate fitness         fitness =
evaluate_fitness(popula on)



        # Track the best solu on          min_fitness_idx
= np.argmin(fitness)       if fitness[min_fitness_idx]
<   best_fitness:                             best_fitness   =
fitness[min_fitness_idx]                   best_solu on  =
popula on[min_fitness_idx]
```

```
        # Selec on          parents = select_parents(popula on, fitness,
POPULATION_SIZE // 2)


        # Crossover
        offspring = crossover(parents, CROSSOVER_RATE)
offspring = mutate(offspring, MUTATION_RATE)

        # Gene Expression          popula on =
gene_expression(offspring)


        # Print progress          print(f"Genera on {genera on + 1}: Best
Fitness = {best_fitness}")


    # Output the best solu on          print("\nBest Solu on
Found:")          print(f"Posi on: {best_solu on}, Fitness:
{best_fitness}")


if __name__ == "__main__":
gene_expression_algorithm()
```

OUTPUT:

```
Generation 1: Best Fitness = 55.82997756903893
Generation 2: Best Fitness = 26.410565738143625
Generation 3: Best Fitness = 21.857647823851615
Generation 4: Best Fitness = 20.016914182036285
Generation 5: Best Fitness = 20.016914182036285
Generation 6: Best Fitness = 20.016914182036285
Generation 7: Best Fitness = 13.81760087982789
Generation 8: Best Fitness = 13.81760087982789
Generation 9: Best Fitness = 12.077725051361178
Generation 10: Best Fitness = 10.461698723345474
Generation 11: Best Fitness = 8.933105023570093
Generation 12: Best Fitness = 6.619449963941974
Generation 13: Best Fitness = 3.1567413435369454
Generation 14: Best Fitness = 3.1567413435369454
Generation 15: Best Fitness = 3.1567413435369454
Generation 16: Best Fitness = 2.74585545305795
Generation 17: Best Fitness = 2.7031453676198964
Generation 18: Best Fitness = 2.078188177116774
Generation 19: Best Fitness = 1.5193087227027497
Generation 20: Best Fitness = 1.4413606561895607
Generation 21: Best Fitness = 0.8501569187378994
Generation 22: Best Fitness = 0.4209372164676112
Generation 23: Best Fitness = 0.3893761873774093
Generation 24: Best Fitness = 0.3893761873774093
Generation 25: Best Fitness = 0.3893761873774093
Generation 26: Best Fitness = 0.3741053651316379
Generation 27: Best Fitness = 0.1381555631914642
Generation 28: Best Fitness = 0.12238160343023853
Generation 29: Best Fitness = 0.12238160343023853
Generation 30: Best Fitness = 0.12238160343023853

Best Solution Found:
Position: [-0.03614343 -0.00257499  0.02260677  0.31412563  0.14792784], Fitness: 0.12238160343023853
```