Program 7 - Optimization via Gene Expression Algorithms (GEA) :

The goal is to implement a Gene Expression Algorithm (GEA) to maximize the mathematical function f(x)=−x2+10x+15f(x) = -x^2 + 10x + 15f(x)=−x2+10x+15 within the range [0,10][0, 10][0,10]. GEA mimics biological processes to evolve solutions through genetic sequences, selection, crossover, mutation, and gene expression to optimize the given function.

Algorithm:

Define the Problem: The objective is to maximize f(x)=−x2+10x+15f(x) = -x^2 + 10x + 15f(x)=−x2+10x+15.

Initialize Parameters:

- Population size: PPP
- Number of genes per individual: GGG
- Mutation rate: μ\muμ
- Crossover rate: λ\lambdaλ
- Number of generations: TTT
- Search space

Initialize Population:

- Generate an initial population of random genetic sequences, each encoding a potential solution (gene).

Evaluate Fitness:

- Evaluate the fitness of each individual by applying the objective function to its decoded value.

Selection:

- Select individuals based on their fitness using techniques like tournament selection or roulette-wheel selection for reproduction.

Crossover:

- Apply crossover between selected pairs of individuals to create offspring (e.g., single-point crossover).

Mutation:

- Mutate the offspring with a probability determined by the mutation rate.

Gene Expression:

- The genetic sequence is translated into a functional solution (real number in the domain).

Iterate:

- Repeat the selection, crossover, mutation, and gene expression steps for a fixed number of generations.

Output the Best Solution:

- Track and output the best solution found during the iterations.

Code:

```
import numpy as np


# Define the fitness function

def fitness_function(x):

    return -x**2 + 10*x + 15


# Gene Expression Algorithm (GEA) implementation

def gene_expression_algorithm():

    # Input parameters

    population_size = int(input("Enter population size: "))

    num_genes = int(input("Enter number of genes per individual: "))

    mutation_rate = float(input("Enter mutation rate (0-1): "))

    crossover_rate = float(input("Enter crossover rate (0-1): "))

    num_generations = int(input("Enter number of generations: "))

    x_min = float(input("Enter minimum value of x: "))

    x_max = float(input("Enter maximum value of x: "))


    # Initialize the population with random genetic sequences

    population = np.random.uniform(x_min, x_max, population_size)
```

```
    fitness = np.array([fitness_function(individual) for individual in
population])  # Evaluate fitness

    # Main optimization loop
    for generation in range(num_generations):
        new_population = []

        # Selection (tournament selection)
        for _ in range(population_size):
            # Select two random parents
            parent_indices = np.random.choice(population_size, 2,
replace=False)
            parent1, parent2 = population[parent_indices]
            fitness1, fitness2 = fitness[parent_indices]

            # Select the best parent based on fitness
            selected_parent = parent1 if fitness1 > fitness2 else parent2

            # Crossover (single-point crossover)
            if np.random.rand() < crossover_rate:
                # As we have only one gene per individual, we just pick one
parent
                offspring = selected_parent
            else:
                offspring = selected_parent
```

```python
            # Mutation
            if np.random.rand() < mutation_rate:
                offspring = np.random.uniform(x_min, x_max)

            # Append the offspring to the new population
            new_population.append(offspring)

        # Update population and fitness
        population = np.array(new_population)
        fitness = np.array([fitness_function(individual) for individual in population])

    # Find the best solution
    best_index = np.argmax(fitness)
    best_solution = population[best_index]
    best_fitness = fitness[best_index]

    return best_solution, best_fitness  # Return the decoded solution (best gene)

# Run the Gene Expression Algorithm
best_solution, best_fitness = gene_expression_algorithm()
print(f"Best Solution: {best_solution}, Fitness: {best_fitness}")
```

```
print("Name-pooja Gaikwad(1BM22CS194)")
```

Ouput:

```
Enter population size: 20
Enter number of genes per individual: 1
Enter mutation rate (0-1): 0.5
Enter crossover rate (0-1): 0.9
Enter number of generations: 100
Enter minimum value of x: 0
Enter maximum value of x: 10
Best Solution: 4.97870106795995, Fitness: 39.999546355493955
Name-pooja Gaikwad(1BM22CS194)
```

Observation:

① Gene Expression Programming

1) Initialize population P of individual i.

2) Genes are represented as expression tree.

3) Fitness = evaluate fitness (i)

4) Solution (Population)

$P_1 = P$ (fitness, i)

P_2 = Select (Population)

* Based on fitness

for generation 1 to max :

Evaluate fitness (Population)
Selected Parent = solution (Population)
New-Population = Reproduction (selected-P , crossover-rate, Mutation-rate)

population = new-pop.

Return but individual (sol.)