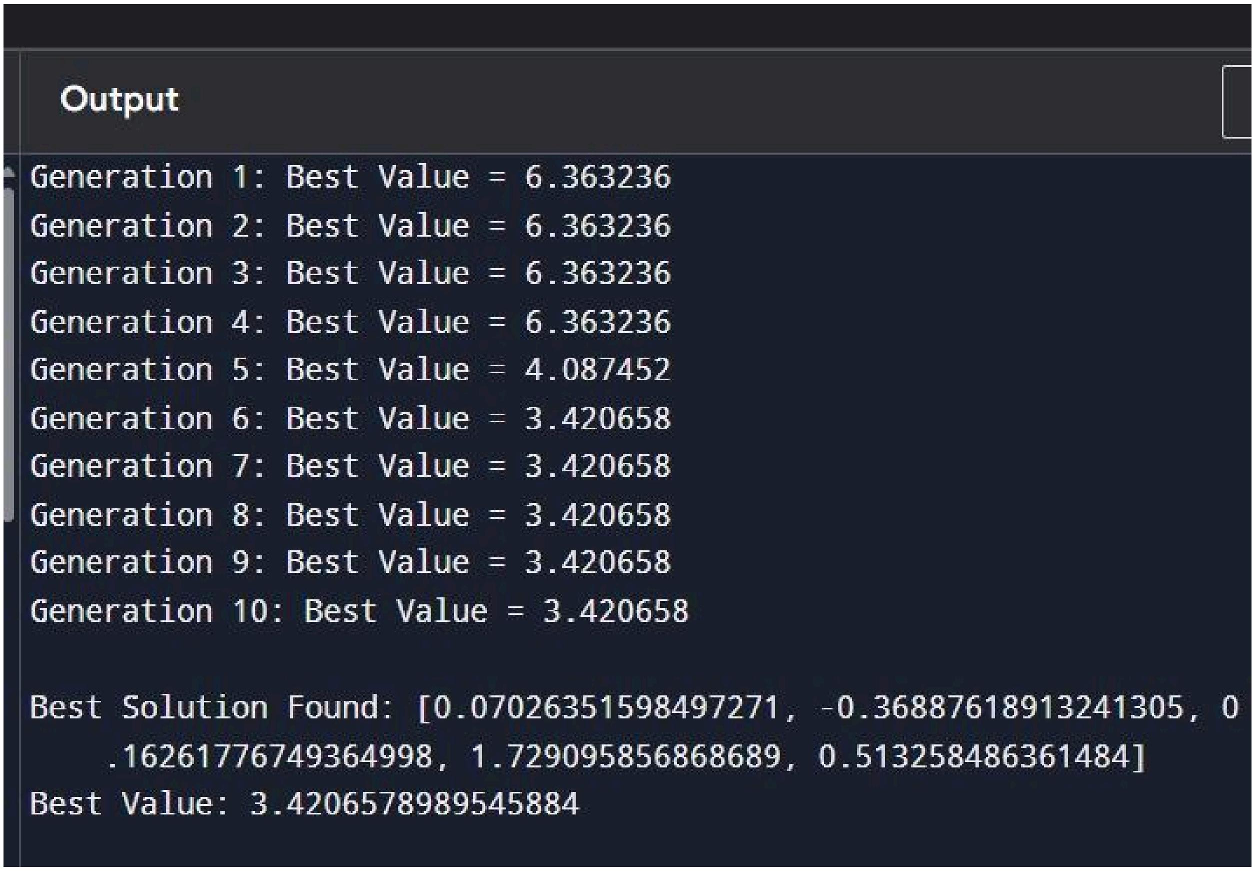
# Optimization via Gene Expression Algorithm

Code :

```
import random
import numpy as np

def sphere_function(x):
    """Optimization function: minimize sum of squares"""
    return sum(xi ** 2 for xi in x)


POP_SIZE = 50        # population size
NUM_GENES = 5        # dimension of solution
MUTATION_RATE = 0.05  # mutation probability
CROSSOVER_RATE = 0.08# crossover probability
GENERATIONS = 10   # number of generations
GENE_RANGE = (-5, 5) # range of values for each gene


def initialize_population():
    return [[random.uniform(*GENE_RANGE) for _ in range(NUM_GENES)]
            for _ in range(POP_SIZE)]

def evaluate_fitness(population):
    return [1 / (1 + sphere_function(ind)) for ind in population]  # smaller value -> higher fitness

def select(population, fitness):
    """Roulette wheel selection"""
    total_fitness = sum(fitness)
    pick = random.uniform(0, total_fitness)
    current = 0
    for ind, fit in zip(population, fitness):
        current += fit
        if current > pick:
            return ind
    return population[-1]

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(individual):
    for i in range(NUM_GENES):
        if random.random() < MUTATION_RATE:
            individual[i] = random.uniform(*GENE_RANGE)
    return individual

Here gene expression = interpreting list of genes as solution vector
def express(individual):
    return individual


def gene_expression_algorithm():
    population = initialize_population()
    best_solution = None
    best_value = float("inf")

    for gen in range(GENERATIONS):
        fitness = evaluate_fitness(population)
        new_population = []

        while len(new_population) < POP_SIZE:
            parent1 = select(population, fitness)
            parent2 = select(population, fitness)

            child1, child2 = crossover(parent1, parent2)

            child1 = mutate(child1)
            child2 = mutate(child2)


            new_population.append(express(child1))
            if len(new_population) < POP_SIZE:
                new_population.append(express(child2))

        population = new_population


        for ind in population:
            value = sphere_function(ind)
            if value < best_value:
                best_value = value
                best_solution = ind

        print(f"Generation {gen+1}: Best Value = {best_value:.6f}")

    return best_solution, best_value



if __name__ == "__main__":
    best_sol, best_val = gene_expression_algorithm()
    print("\nBest Solution Found:", best_sol)
    print("Best Value:", best_val)
```

Output :



```
Generation 1: Best Value = 6.363236
Generation 2: Best Value = 6.363236
Generation 3: Best Value = 6.363236
Generation 4: Best Value = 6.363236
Generation 5: Best Value = 4.087452
Generation 6: Best Value = 3.420658
Generation 7: Best Value = 3.420658
Generation 8: Best Value = 3.420658
Generation 9: Best Value = 3.420658
Generation 10: Best Value = 3.420658

Best Solution Found: [0.07026351598497271, -0.36887618913241305, 0
    .16261776749364998, 1.729095856868689, 0.513258486361484]
Best Value: 3.4206578989545884
```