**Genetic Algorithm:**

import random

# Step 1: Determine the number of chromosomes, generation, mutation rate, and crossover rate value

def genetic\_algorithm(num\_chromosomes, mutation\_rate, crossover\_rate, num\_variables):

# Step 2: Generate chromosome population with random initialization

population = generate\_population(num\_chromosomes, num\_variables)

i = int(1)

while True:

# Step 3: Evaluation of fitness value of chromosomes

fitness\_values = [fitness\_function(chromosome) for chromosome in population]

# Step 8: Solution (Best Chromosomes)

best\_chromosome = population[fitness\_values.index(min(fitness\_values))]

# print(f"Generation {i}: solution: {best\_chromosome}, Fitness: {min(fitness\_values)}")

if min(fitness\_values) == 0:

print(f"\nGeneration {i}: Best solution: {best\_chromosome}, Fitness: {min(fitness\_values)}")

print(f"Optimal Solution after {i} Generations")

break

i = i + 1

next\_generation = []

# Step 5: Chromosomes selection

selected\_population = selection(population, fitness\_values)

for \_ in range(num\_chromosomes // 2):

parent1, parent2 = selected\_population[random.randint(0, len(selected\_population) - 1)], selected\_population[random.randint(0, len(selected\_population) - 1)]

# Step 6: Crossover

child1, child2 = crossover(parent1, parent2, crossover\_rate)

# Step 7: Mutation

child1 = mutation(child1, mutation\_rate)

child2 = mutation(child2, mutation\_rate)

next\_generation.extend([child1, child2])

population = next\_generation

return best\_chromosome

# Step 2: Generate initial population

def generate\_population(size, num\_variables):

population = []

for \_ in range(size):

individual = [random.randint(0, 10) for \_ in range(num\_variables)]

population.append(individual)

return population

# Step 4: Fitness function evaluation

def fitness\_function(variables):

x, y, z, w = variables

return abs((a + 2\*b + 3\*c + 4\*d) - 30)

# Step 5: Chromosome selection using roulette wheel selection

def selection(population, fitness\_values):

selected\_population = []

total\_fitness = sum(fitness\_values)

probabilities = [fitness / total\_fitness for fitness in fitness\_values]

for \_ in range(len(population)):

selected = random.choices(population, probabilities)[0]

selected\_population.append(selected)

return selected\_population

# Step 6: Crossover

def crossover(parent1, parent2, crossover\_rate):

if random.random() < crossover\_rate:

crossover\_point = random.randint(1, len(parent1) - 1)

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

return parent1, parent2

# Step 7: Mutation

def mutation(individual, mutation\_rate):

if random.random() < mutation\_rate:

index = random.randint(0, len(individual) - 1)

new\_value = random.randint(0, 10)

individual[index] = new\_value

return individual

def get\_input():

num\_chromosomes = int(input("Enter the number of chromosomes: "))

mutation\_rate = float(input("Enter the mutation rate: "))

crossover\_rate = float(input("Enter the crossover rate: "))

num\_variables = 4 # Assuming 4 variables (a,b,c,d) for the given equation

return num\_chromosomes, mutation\_rate, crossover\_rate, num\_variables

def main():

num\_chromosomes, mutation\_rate, crossover\_rate, num\_variables = get\_input()

best\_chromosome = genetic\_algorithm(num\_chromosomes, mutation\_rate, crossover\_rate, num\_variables)

print("Best solution:", best\_chromosome)

if \_\_name\_\_ == "\_\_main\_\_":

main()

