**CODE:**

import numpy as np

def objective\_function(x):

return sum(x\*\*2)

def genetic\_algorithm(population\_size, num\_generations, mutation\_rate):

population = np.random.rand(population\_size, 2)

best\_solution, best\_fitness = None, float('inf')

for \_ in range(num\_generations):

fitness = np.apply\_along\_axis(objective\_function, 1, population)

min\_fitness\_idx = np.argmin(fitness)

if fitness[min\_fitness\_idx] < best\_fitness:

best\_fitness = fitness[min\_fitness\_idx]

best\_solution = population[min\_fitness\_idx]

parents\_idx = np.random.choice(population\_size, size=(population\_size // 2, 2), replace=False)

parents = population[parents\_idx]

crossover\_point = np.random.randint(1, 2)

offspring = np.empty\_like(parents)

for i in range(len(parents)):

crossover\_mask = np.zeros\_like(parents[i][0], dtype=bool)

crossover\_mask[:crossover\_point] = True

offspring[i][0] = np.where(crossover\_mask, parents[i][0], parents[i][1])

offspring[i][1] = np.where(~crossover\_mask, parents[i][0], parents[i][1])

mutation\_mask = np.random.rand(population\_size // 2, 2) < mutation\_rate

mutation\_values = np.random.rand(population\_size // 2, 2)

offspring[mutation\_mask] = mutation\_values[mutation\_mask]

population = offspring

return best\_solution, best\_fitness

best\_solution, best\_fitness = genetic\_algorithm(population\_size=100, num\_generations=100, mutation\_rate=0.01)

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

print("Best fitness:", best\_fitness)