**Genetics Algorithm**

**Mathematical Function Optimization:**

import numpy as np

def rosenbrock(x):

    return sum(100.0\*(x[1:]-x[:-1]\*\*2.0)\*\*2.0 + (1-x[:-1])\*\*2.0)

def initialize\_population(pop\_size, dim):

    return np.random.uniform(-5, 5, (pop\_size, dim))

def selection(population, fitness, num\_parents):

    parents = population[np.argsort(fitness)][:num\_parents]

    return parents

def crossover(parents, pop\_size):

    offspring = []

    crossover\_point = parents.shape[1] // 2

    for \_ in range(pop\_size - parents.shape[0]):

        parent1, parent2 = parents[np.random.choice(parents.shape[0], 2, replace=False)]

        offspring.append(np.concatenate((parent1[:crossover\_point], parent2[crossover\_point:])))

    return np.array(offspring)

def mutation(offspring, mutation\_rate):

    for i in range(offspring.shape[0]):

        if np.random.rand() < mutation\_rate:

            mutation\_point = np.random.randint(offspring.shape[1])

            offspring[i, mutation\_point] = np.random.uniform(-5, 5)

    return offspring

def genetic\_algorithm(pop\_size, dim, num\_generations, mutation\_rate, num\_parents):

    population = initialize\_population(pop\_size, dim)

    for generation in range(num\_generations):

        fitness = np.array([rosenbrock(ind) for ind in population])

        parents = selection(population, fitness, num\_parents)

        offspring = crossover(parents, pop\_size)

        offspring = mutation(offspring, mutation\_rate)

        population = np.vstack((parents, offspring))

    best\_solution = population[np.argmin(fitness)]

    return best\_solution, rosenbrock(best\_solution)

pop\_size = 50

dim = 10

num\_generations = 100

mutation\_rate = 0.1

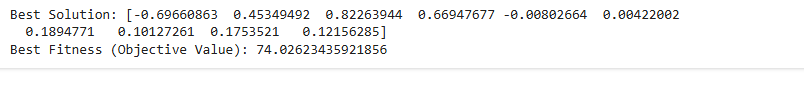
num\_parents = 20

best\_solution, best\_fitness = genetic\_algorithm(pop\_size, dim, num\_generations, mutation\_rate, num\_parents)

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

print("Best Fitness (Objective Value):", best\_fitness)

**OUTPUT:**

****