**Practical No : 11**

**Problem Statement:**

Solve quadratic equations using Genetic Algorithm.

**Code:**

import random

# Coefficients of the quadratic equation (e.g., ax^2 + bx + c = 0)

A = 4

B = -12

C = 8

class Individual:

def \_\_init\_\_(self, x\_value):

self.x\_value = x\_value

self.fitness = self.calculate\_fitness()

def calculate\_fitness(self):

"""

Fitness score: how close the quadratic equation is to zero

"""

value\_at\_x = A \* (self.x\_value \*\* 2) + B \* self.x\_value + C

return abs(value\_at\_x)

def mate(self, partner):

"""

Perform mating (crossover) and produce new offspring

"""

# Simple average of parent values

child\_x\_value = (self.x\_value + partner.x\_value) / 2

# Mutate the child with a certain probability

if random.random() < 0.1: # 10% mutation chance

child\_x\_value += random.uniform(-1, 1) # Small mutation

return Individual(child\_x\_value)

def create\_initial\_population(size):

"""

Create a population of individuals with random x values

"""

return [Individual(random.uniform(-10, 10)) for \_ in range(size)]

def selection(population):

"""

Select two individuals based on fitness (tournament selection)

"""

tournament\_size = 5

selected = random.sample(population, tournament\_size)

return sorted(selected, key=lambda ind: ind.fitness)[:2]

def genetic\_algorithm(population\_size, max\_generations):

print("Quadratic Equation to Solve using Generic Algorithm: ")

print(f"{A}x^2 + {B}x + {C} = 0")

population = create\_initial\_population(population\_size)

found = False

for generation in range(max\_generations):

population = sorted(population, key=lambda ind: ind.fitness)

# Check for the best fitness

best\_individual = population[0]

print(f"Generation: {generation + 1}, Best X: {best\_individual.x\_value:.4f}, Fitness: {best\_individual.fitness:.4f}")

# Check for a solution

if best\_individual.fitness <= 0.00: # tolerance

print(f"A\*({best\_individual.x\_value:.4f}^2) + B\*({best\_individual.x\_value:4f}) + C = 0")

print("Found a solution!")

found = True

break

# Create the next generation

next\_generation = []

for \_ in range(population\_size // 2): # Create pairs

parent1, parent2 = selection(population)

child1 = parent1.mate(parent2)

child2 = parent2.mate(parent1)

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

population = next\_generation

if not found:

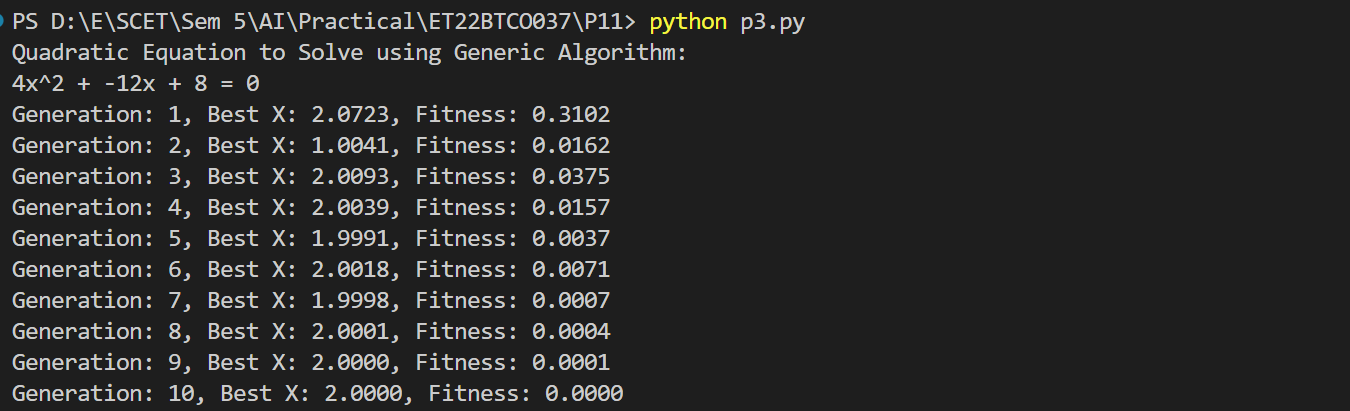
print("Solution not found")

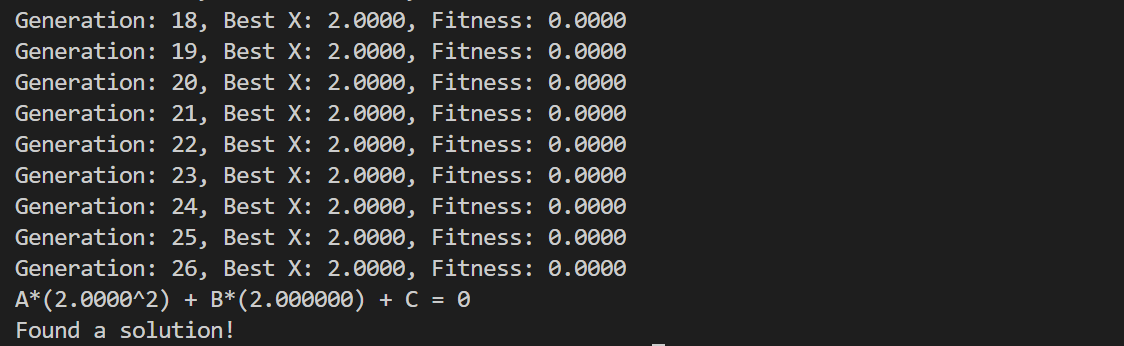
# Driver Code

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

genetic\_algorithm(population\_size=100, max\_generations=1000)

**Output:**

****

****