LAB-2

Code:

import random

import math

function\_set = ['+', '-', '\*', '/']

terminal\_set = ['x', 1, 2, 3, 4, 5]

population\_size = 50

chromosome\_length = 10

mutation\_rate = 0.1

crossover\_rate = 0.7

generations = 100

# f(x) = x^2 + 2x + 1

def fitness\_function(expr):

    error = 0.0

    for x in range(-10, 11):

        try:

            val = eval(expr.replace('x', str(x)))

            target = x\*\*2 + 2\*x + 1

            error += abs(val - target)

        except (ZeroDivisionError, SyntaxError, NameError):

            # Penalize invalid expressions with a large error

            error += 1000

    return -error  # Negative error as fitness (higher is better)

def generate\_chromosome():

    chromosome = []

    for \_ in range(chromosome\_length):

        if random.random() < 0.5:

            chromosome.append(str(random.choice(terminal\_set)))

        else:

            chromosome.append(random.choice(function\_set))

    return chromosome

# Expression from chromosome: join as string (simple concatenation)

def express(chromosome):

    # Join elements into string expression - simplistic, can produce invalid exprs

    return "".join(chromosome)

# Selection by tournament

def select(population, fitnesses):

    tournament\_size = 3

    selected = []

    for \_ in range(population\_size):

        competitors = random.sample(list(zip(population, fitnesses)), tournament\_size)

        winner = max(competitors, key=lambda x: x[1])

        selected.append(winner[0])

    return selected

# Crossover between two chromosomes

def crossover(parent1, parent2):

    if random.random() > crossover\_rate:

        return parent1[:], parent2[:]

    point = random.randint(1, chromosome\_length - 2)

    child1 = parent1[:point] + parent2[point:]

    child2 = parent2[:point] + parent1[point:]

    return child1, child2

# Mutation on chromosome

def mutate(chromosome):

    for i in range(chromosome\_length):

        if random.random() < mutation\_rate:

            if chromosome[i] in function\_set:

                chromosome[i] = random.choice(function\_set)

            else:

                chromosome[i] = str(random.choice(terminal\_set))

    return chromosome

# Main GEA loop

def gene\_expression\_algorithm():

    population = [generate\_chromosome() for \_ in range(population\_size)]

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

    for gen in range(generations):

        # Expression and fitness evaluation

        fitnesses = []

        for chrom in population:

            expr = express(chrom)

            fit = fitness\_function(expr)

            fitnesses.append(fit)

            if fit > best\_fitness:

                best\_fitness = fit

                best\_solution = expr

        # Selection

        selected = select(population, fitnesses)

        # Create next generation

        next\_gen = []

        while len(next\_gen) < population\_size:

            parent1 = random.choice(selected)

            parent2 = random.choice(selected)

            child1, child2 = crossover(parent1, parent2)

            next\_gen.append(mutate(child1))

            if len(next\_gen) < population\_size:

                next\_gen.append(mutate(child2))

        population = next\_gen

        print(f"Generation {gen+1}: Best Fitness = {best\_fitness}")

    print(f"\nBest solution found: {best\_solution} with fitness {best\_fitness}")

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

    gene\_expression\_algorithm()

Output:



