**Lab-7**-Optimization via Gene Expression Algorithms

**Code:**

#7

import numpy as np

# Step 1: Define the Problem

# Define your mathematical optimization function here

# Example: Minimize the function f(x) = x^2 - 4x + 4 (a simple quadratic function)

def objective\_function(x):

return x\*\*2 - 4\*x + 4

# Step 2: Initialize Parameters

def initialize\_parameters():

population\_size = int(input("Enter population size: ")) # Number of individuals in the population

num\_genes = int(input("Enter number of genes: ")) # Number of genes in each individual

mutation\_rate = float(input("Enter mutation rate (0 to 1): ")) # Probability of mutation

crossover\_rate = float(input("Enter crossover rate (0 to 1): ")) # Probability of crossover

num\_generations = int(input("Enter number of generations: ")) # Number of generations

return population\_size, num\_genes, mutation\_rate, crossover\_rate, num\_generations

# Step 3: Initialize Population

def initialize\_population(population\_size, num\_genes):

population = np.random.uniform(low=-5, high=5, size=(population\_size, num\_genes))

return population

# Step 4: Evaluate Fitness

def evaluate\_fitness(population, objective\_function):

fitness = np.apply\_along\_axis(objective\_function, 1, population) # Apply the objective function to each individual

return fitness

# Step 5: Selection (Tournament Selection)

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

parents = []

for \_ in range(num\_parents):

tournament = np.random.choice(population.shape[0], size=3, replace=False) # Select 3 individuals for tournament

tournament\_fitness = fitness[tournament]

winner = tournament[np.argmin(tournament\_fitness)] # Select the individual with the best fitness

parents.append(population[winner])

return np.array(parents)

# Step 6: Crossover (Single-point crossover)

def crossover(parents, crossover\_rate):

num\_parents = parents.shape[0]

offspring = []

for i in range(0, num\_parents, 2):

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

# Ensure that we have more than 1 gene to perform crossover

if parents.shape[1] > 1:

crossover\_point = np.random.randint(1, parents.shape[1])

offspring1 = np.concatenate([parents[i, :crossover\_point], parents[i+1, crossover\_point:]])

offspring2 = np.concatenate([parents[i+1, :crossover\_point], parents[i, crossover\_point:]])

offspring.append(offspring1)

offspring.append(offspring2)

else:

# No crossover if there's only 1 gene

offspring.append(parents[i])

if i + 1 < num\_parents:

offspring.append(parents[i + 1])

else:

offspring.append(parents[i])

if i + 1 < num\_parents:

offspring.append(parents[i + 1])

return np.array(offspring)

# Step 7: Mutation

def mutation(offspring, mutation\_rate):

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

for j in range(offspring.shape[1]):

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

offspring[i, j] += np.random.normal(0, 0.1) # Apply Gaussian mutation

return offspring

# Step 8: Gene Expression (Translate Genes into Solutions)

def gene\_expression(offspring):

# In this case, the genes are directly the solutions

return offspring

# Step 9: Iterate (Repeat the selection, crossover, mutation, and gene expression processes)

def run\_ge\_algorithm(objective\_function):

population\_size, num\_genes, mutation\_rate, crossover\_rate, num\_generations = initialize\_parameters()

# Initialize population

population = initialize\_population(population\_size, num\_genes)

# Start optimization process

best\_solution = None

best\_fitness = float('inf')

for generation in range(num\_generations):

# Evaluate fitness

fitness = evaluate\_fitness(population, objective\_function)

# Track the best solution

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]

# Selection

parents = selection(population, fitness, population\_size // 2)

# Crossover

offspring = crossover(parents, crossover\_rate)

# Mutation

offspring = mutation(offspring, mutation\_rate)

# Gene Expression (Directly apply the offspring as solutions)

population = gene\_expression(offspring)

# Print the progress

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

return best\_solution, best\_fitness

# Step 10: Output the Best Solution

best\_solution, best\_fitness = run\_ge\_algorithm(objective\_function)

print(f"Best solution found: {best\_solution}")

print(f"Best fitness value: {best\_fitness}")

print("Pooja M")

print("1BM22CS195”)  
