**Genetic Algorithm**

**Assignment**

**Submitted By:**

Rishav Acharya

Roll no: 01

Semester: Fourth/2077 Batch

Q. Find the maximum solution of f(x,y,z) =x2-xy+z with the given criteria -5<x<5 , 0<y<2, -2<z<2 set following parameter in advance ,

* population size 100
* probability of crossover 0.9
* probability of mutation 0.15
* fitness function is f(x,y,z)

**Solution:**

<https://colab.research.google.com/drive/1pnvnQDu-2HfKHz5fZQ1X6sCefgZjNL1O>

Code:

import numpy as np

def fitness(x, y, z):

"""Calculates the fitness of an individual (x, y, z)."""

return x\*\*2 - x\*y + z

def initialize\_population(n, x\_min=-5, x\_max=5, y\_min=0, y\_max=2, z\_min=-2, z\_max=2):

"""Initializes a population of size n."""

population = []

for \_ in range(n):

x = np.random.uniform(x\_min, x\_max)

y = np.random.uniform(y\_min, y\_max)

z = np.random.uniform(z\_min, z\_max)

population.append((x, y, z))

return population

def roulette\_wheel(population, fitness\_values, tournament\_size=4):

"""Selects the best individuals from the population using tournament selection."""

selected\_indices = []

for \_ in range(len(population)):

tournament = np.random.choice(len(population), tournament\_size, replace=False)

tournament\_fitness = [fitness\_values[i] for i in tournament]

selected\_indices.append(tournament[tournament\_fitness.index(max(tournament\_fitness))])

return selected\_indices

def crossover(parent1, parent2):

"""Performs single-point crossover between two parent individuals."""

crossover\_point = np.random.randint(1, len(parent1))

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

def mutate(individual, mutation\_prob=0.15, x\_min=-5, x\_max=5, y\_min=0, y\_max=2, z\_min=-2, z\_max=2):

"""Mutates an individual with a given probability."""

mutated\_individual = list(individual)

for i in range(3):

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

if i == 0:

mutated\_individual[i] = np.random.uniform(x\_min, x\_max)

elif i == 1:

mutated\_individual[i] = np.random.uniform(y\_min, y\_max)

elif i == 2:

mutated\_individual[i] = np.random.uniform(z\_min, z\_max)

return tuple(mutated\_individual)

def genetic\_algorithm(population\_size=100, num\_generations=100, crossover\_prob=0.9, mutation\_prob=0.10):

"""Runs the genetic algorithm."""

# Initialize the population.

population = initialize\_population(population\_size)

# Iterate over the generations.

for generation in range(num\_generations):

# Calculate the fitness of each individual.

fitness\_values = [fitness(\*individual) for individual in population]

# Select the best individuals to produce the next generation.

selected\_indices = roulette\_wheel(population, fitness\_values)

# Produce the next generation through crossover and mutation.

new\_population = []

for i in range(0, population\_size, 2):

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

child1, child2 = crossover(population[selected\_indices[i]], population[selected\_indices[i + 1]])

new\_population.extend([child1, child2])

else:

new\_population.extend([population[selected\_indices[i]], population[selected\_indices[i + 1]]])

new\_population = [mutate(ind) for ind in new\_population]

# Replace the old population with the new population.

population = new\_population

fitness\_values = [fitness(\*individual) for individual in population]

# Find the maximum fitness and corresponding individual

max\_fitness = max(fitness\_values)

max\_individual = population[fitness\_values.index(max\_fitness)]

return max\_individual, max\_fitness

# Run the genetic algorithm

best\_individual, best\_fitness = genetic\_algorithm()

print("Best Individual:", best\_individual)

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

**Output:**

