18/10/24

LAB - 8:

Genetic Algorithm:

Algorithm:

Step 1: Set population_size, mutation_rate, crossover_rate, num_generations, gene_length

Step 2: Generate population_size, individually each individual is random binary string of length (gene_length)

Step 3: Repeat step 2 for each generation (0 to no. of generations)

Step 4 Evaluate fitness of each generation individually

Step 5: Select parents based on fitness by performing roulette wheel selection

Step 6: Create new population based on
→ Crossover
→ Mutation
→ Replace old population

Step 7: Track the best fit solution

Step 8: Generate all the best solution.

Code:

```
import random
import numpy as np

def fitness_function(x):
    return x ** 2

population_size = 10
mutation_rate = 0.01
crossover_rate = 0.8
num_generations = 10
```

```
gene_length = 10
def create_population(size, gene_length):
    return [np.random.randint(0, 2, gene_length).tolist() for
            _ in range(size)]

def binary_to_decimal(binary):
    binary_str = ''.join(str(bit) for bit in binary)
    return int(binary_str, 2) / (2**gene_length - 1) * 20 - 10

def evaluate_population(population):
    return [fitness_function(binary_to_decimal(individual))
            for individual in population]

def select(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    selection_probs = [fitness / total_fitness for fitness in
                       fitness_scores]
    return population[np.random.choice(range(len(population)),
                      p=selection_probs)]
def crossover(parent1, parent2):
    if random.random() < crossover_rate:
        crossover_point = random.randint(1, gene_length-1)
        child1 = parent1[:crossover_point] + parent2[crossover_pt:]
        child2 = parent2[:crossover_pt] + parent1[crossover_pt:]
        return [child1, child2]
    return [parent1, parent2]

def mutate(individual):
    for i in range(gene_length):
        if random.random() < mutation_rate:
            individual[i] = 1 - individual[i]

return individual
```

```
def genetic_algorithm():
    population = create_pop (pop_size, gene_length)
    for generation in range (num_gen):
        fitness_score = evaluate_pop(pop)
        best_fitness = max(fitness_score)
        best_individual = population [fitness_score.index(best)]
        print(f"Generation {gen}: Best Fitness = {best:.4f}")
        new_population = []
        while len (new_pop) < pop_size:
            parent1 = select (population, fitness_scores)
            parent2 = select (population, fitness_scores)
            offspring = crossover (parent1, parent2)
        population = new_pop [:pop_size]
best_fitness = max(fitness_scores)
best_individual = population [fitness_scores.index(best_fit)]
best_solution = binary_to_decimal (best_individual)
print(f"\n Best solution found: x = {best_solution:.4f}
      f(x) = {fitness_function (best_solution):.4f}")

genetic_algorithm()
```

Output:

```
Generation 0: Best Fitness = 72.4912
Generation 1: Best Fitness = 92.3327
Generation 2: Best Fitness = 97.6677
Generation 3: Best Fitness = 92.3327
Generation 4: Best Fitness = 92.3327
```

Generation 5: Best Fitness = 92.3327
Generation 6: Best Fitness = 92.3327
Generation 7: Best Fitness = 93.8417
Generation 8: Best Fitness = 93.8417
Generation 9: Best Fitness = 92.3327

Best solution found: $x = -9.4525$, $f(x) = 89.3515$