18/10/24

PAGE :
DATE : / /

Genetic Algorithm:

```
import random
def fitness_function(x):
    return x**2
population_size = 100
mutation_rate = 0.1
crossover_rate = 0.8
num_generations = 50
lower_bound = -10
upper_bound = 10

def generate_population(size, lower_bound, upper_bound):
    return [random.uniform(lower_bound, upper_bound) for _ in range(size)]

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

def select_parents(population, fitnesses):
    total_fitness = sum(fitnesses)
    selection_probs = [f / total_fitness for f in fitnesses]
    parent1 = random.choices(population, weights=selection_probs, k=1)[0]
    parent2 = random.choices(population, weights=selection_probs, k=1)[0]
    return parent1, parent2
def crossover(parent1, parent2, crossover_rate):
    if random.random() < crossover_rate:
        crossover_point = random.random()
        child1 = crossover_point * parent1 + (1 - crossover_point) * parent2
        child2 = crossover_point * parent2 + (1 - crossover_point) * parent1
        return child1, child2
    else:
        return parent1, parent2
```

```
def mutate(individual, mutation_rate, lower_bound, upper_bound):
    if random.random() < mutation_rate:
        individual = random.uniform(lower_bound, upper_bound)
    return individual
def genetic_algorithm():
    population = generate_population(population_size, lower_bound,
                                     upper_bound)
    for generation in range(num_generations):
        fitness = evaluate_population(population)
        new_population = []
        while len(new_population) < population_size:
            parent1, parent2 = select_parents(population, fitness)
            child1, child2 = Crossover(parent1, parent2, crossover_rate)

            child1 = mutate(child1, mutation_rate, lower_bound, upper_bound)
            child2 = mutate(child2, mutation_rate, lower_bound, upper_bound)
            new_population.extend([child1, child2])
        population = new_population[:population_size]
    final_fitness = evaluate_population(population)
    best_individual = population[final_fitness.index(max(final_fitness))]
    best_fitness = max(final_fitness)
    return best_individual, best_fitness
best_solution, best_fitness = genetic_algorithm()
print(f"Best solution found: {best_solution}")
print(f"Fitness of the best solution: {best_fitness}")
```

O/P :-

```
Best Solution found:- 4.939656417265594
Fitness of the best solution: 98.79676932...
```

st) * parent 2

st) * parent 1