Lab 8

Optimization via Gene Expression

```
Function(x):
    return sum(x**2)

Initialize-Population(pop_size, num_genes, lower, upper)
    pop ← random(lower, upper, (pop_size, num_genes))
    return pop

Evaluate-fitness(pop, function):
    fitness ← zeros(pop.shape[0])
    for i in range pop.shape[0]:
        fitness[i] ← function(pop[i])
    return fitness

selection(pop, fitness, num_selected):
    prob ← fitness / fitness.sum()
    indices ← random.choice(range(len(pop)), size=num_selected, p=prob)
    selected_pop ← pop[indices]
    return selected_pop

Crossover(selected_pop, crossover):
    new_pop ← []
    num_individuals ← len(selected_pop)
```

```
for i in range 0 to num_individuals-1, step2:
    p1, p2 ← selected_pop[i], selected_pop[i+1]
    if len(p1) > 1 and random < crossover:
        crossover_point ← random(1, len(p1)-1)
        child1 ← concatenate(p1[:crossover_point], p2[crossover_point:])
        child2 ← concatenate(p2[:crossover_point], p1[crossover_point:])
        new_pop.extend([p1, p2])
    if num_individuals % 2 == 1:
        new_pop.append(selected_pop[-1])
return new_pop

Mutation(pop, rate, lower, upper):
    for i in range(pop.shape[0]):
        if random < rate:
            gene ← random 1:0, pop.shape[1]-1)
            pop[i, gene] ← random.uniform
                           (lower, upper)
return pop

Gene_expression(individual, function):
    return function(individual)
```

```
Gene_Expression_Algorithm(pop_size, num_genes, lower, upper, max_gen, mutation_rate, crossover_rate, function):

    pop ← initialize_population(pop_size, num_genes, lower, upper)

    best_sol ← NIL
    best_fit ← ∞

    for gen in 0 to max_gen:

        fitness ← evaluate_fitness(pop, function)
        min_fitness ← fitness.min()

        if min_fitness < best_fit:
            best_fit ← min_fitness
            best_sol ← pop[min(fitness)]

        selected_pop ← selection(pop, fitness, pop_size//2)

        offspring_pop ← crossover(selected_pop, crossover_rate)

        pop ← mutation(offspring_pop, mutation_rate, lower, upper)

        print(Generation, fitness)

    return best_sol, best_fit
```

Inputs :

pop_size = 50
num_genes = 1
lower = -5
upper = 5
max_gen = 100
mutation_rate = 0.1
crossover_rate = 0.7