Lab 2

optimization via Gene Expression Algorithm

1) Initialization
Define constants & parameters:
Set population size P, no of generations G, mutation rate M, crossover rate C and maximum tree depth D
• Define the function set F(eg; +, *, /); and terminal set T.

Initialize the population:
Generate P random individuals, each represented by a mathematical expression of depth upto D

2. Evaluate Fitness
1) For each individual in the population
Replace the variable x in the individuals expression with a specific value e.g. x = 5
2) Evaluate the mathematical expression to calculate fitness
3) If the expression is invalid, assign a high fitness value.

3) Selection
1) Identify the best individual in the population.

e) Store & output the best individuals fitness for the current generation

4) create new population with crossover and Mutation

5) find best individual & its fitness

⟹ code:

```
function fitness_function(x)
    return (Integer(binary_str,2))/
        (2^length(binary_str)-1))*60
```

```
Initialize population_size=20
Initialize_num_genes=10
Initialize mutation_rate=0.1.
Initialize Crossover_rate=0.7
Initialize generations=100
```

Initialize population as an array of population_size with random binary strings of length num_genes

for each generation from 1 to generation DO:

    Initialize fitness as an empty array
    for each individual in population
    fitness[individual_index]=

fitness = array (binary) of individuals
total_fitness = sum(fitness)

selected = random.choice(population, size
= population, probabilities)

crossover

Initialize offspring as an empty array
for i from 0 to population_size - 1
if random value < crossover_rate
then:
point = random.integer
(1, num_genes)
offspring.append(selected[i][:
point] + selected[i+1][point:])
offspring.append(selected[i+1]
[:point] + selected[i][point:])

else:
offspring.append(selected[i])
offspring.append(selected[i+1])

Mutation
for i from 0 to population size = 1 do:
if random.value(mutation_rate) then
point = random.integer(0, num_genes-1)
offspring[i] = offspring

Best individual = Individual in population with the minimum fitness value
best_fitness = fitness_function(binary_decimal(best_individual))

printf("Best solution found:" + binary_decimal(best_individual))

print("Fitness:" + best_fitness)

output:

Best solution found:- 45 + 2992
fitness = 23.7453.

Sol.