Fitness Function

November 28, 2024

[1]:
```
import numpy as np
import random

x1_bits = 13
x2_bits = 9
total_bits = x1_bits + x2_bits
population_size = 20
generations = 100
crossover_prob = 0.8
mutation_prob = 0.01
penalty_value = 1e6
```

[2]:
```
def decode(binary, range_min, range_max, precision, bits):
    decimal = int(binary, 2)
    real_value = range_min + decimal * (range_max - range_min) / (2 ** bits - 1)
    return round(real_value, precision)
```

[3]:
```
def fitness_function(x1, x2):
    objective = (x1 - 2.5)**2 + (x2 - 5)**2
    constraint1 = 5.5 * x2 + x1 - 18 <= 0

    if constraint1:
        penalty = 0
    else:
        penalty = penalty_value

    return -objective - penalty
```

[4]:
```
def generate_population(population_size, total_bits):
    return [''.join(random.choices('01', k=total_bits)) for _ in␣
 ↪range(population_size)]
```

[5]:
```
def evaluate_population(population):
    fitness_scores = []
    for individual in population:
        x1_binary = individual[:x1_bits]
        x2_binary = individual[x1_bits:]
        x1 = decode(x1_binary, 0, 5, 3, x1_bits)
```

```
        x2 = decode(x2_binary, 0, 5, 2, x2_bits)
        fitness_scores.append(fitness_function(x1, x2))
    return fitness_scores
```

[6]:
```
def select_parents(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    probabilities = [f / total_fitness for f in fitness_scores]
    selected_indices = np.random.choice(len(population), size=2,␣
↪p=probabilities)
    return population[selected_indices[0]], population[selected_indices[1]]
```

[7]:
```
def crossover(parent1, parent2):
    if random.random() < crossover_prob:
        point = random.randint(1, total_bits - 1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
        return child1, child2
    return parent1, parent2
```

[8]:
```
def mutate(individual):
    mutated = list(individual)
    for i in range(len(mutated)):
        if random.random() < mutation_prob:
            mutated[i] = '0' if mutated[i] == '1' else '1'
    return ''.join(mutated)
```

[9]:
```
def genetic_algorithm():
    population = generate_population(population_size, total_bits)
    for generation in range(generations):
        fitness_scores = evaluate_population(population)
        new_population = []
        for _ in range(population_size // 2):
            parent1, parent2 = select_parents(population, fitness_scores)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1)
            child2 = mutate(child2)
            new_population.extend([child1, child2])
        population = new_population

    # Evaluate final population
    final_fitness = evaluate_population(population)
    best_index = np.argmax(final_fitness)
    best_individual = population[best_index]

    # Decode best solution
    x1_binary = best_individual[:x1_bits]
    x2_binary = best_individual[x1_bits:]
```

```python
    x1 = decode(x1_binary, 0, 5, 3, x1_bits)
    x2 = decode(x2_binary, 0, 5, 2, x2_bits)
    best_fitness = final_fitness[best_index]

    return x1, x2, -best_fitness

# Run the Genetic Algorithm
best_x1, best_x2, best_objective = genetic_algorithm()
print(f"Best solution: x1 = {best_x1}, x2 = {best_x2}, Objective = {best_objective}")
```

```
Best solution: x1 = 2.939, x2 = 4.91, Objective = 1000000.200821
```