Lab. 5

Date __/__/__
Page _____

# Simulated Annealing Algorithm

Input:

- Objective function $f(x)$ to minimize.
- Initial solution $x_0$
- Initial Temperature $T_0$.
- Cooling rate $\alpha$ $(0 < \alpha < 1)$
- Step size $\Delta x$ to control size of neighbor.
- Maximum number of iterations max_iter.
- Minimum Temperature $T_{min}$

1. Initialise:

Set $x_{current} = x_0$ (initial solution)
Set $T = T_0$ (initial temp)
Set $x_{best} = x_0$
iteration counter $k = 0$

2. Repeat until stopping criteria is met (max iteration or $T$ reaches $T_{min}$):

a) Generate a neighbor:

Generate random neighbor $x_{new}$ of the current $x_0$ by modifying $x_0$ slightly

$x_{new} = x_{current} + \Delta x$, $\Delta x$ is a random perturbation

b) Evaluate Objective function

Objective function: Rastrigin function

$$f(x) = A \cdot n + \sum_{i=1}^{n} [x_i^2 - A\cos(2\pi x_i)]$$

A is typically 10
n is no. of dimensions
$x_i$ is each element of solution vector

. compute $f(x_{new})$ and $f(x_{current})$

c) Acceptance Probability:

If $f(x_{new}) < f(x_{current})$, accept $x_{new}$ as new solution

d) Update current solution

If $x_{new}$ is accepted:
$x_{current} = x_{new}$

e) increment iteration

3) Termination:

Stop when temperature T is lower than $T_{min}$ or max iterations are reached

Return $x_{best}$

code:

Date __/__/__
Page _____

```
import math
import random

def objective_function(x):
    A=10
    return A * len(x) + sum([x[i]**2 - A * math.cos(2*math.pi*x[i]) for i in range(len(x))])

def get_neighbor(x, step_size):
    neighbor = x[:]
    i = random.randint(0, len(x) - 1)
    neighbor[i] += random.uniform(-step_size, step_size)
    return neighbor

def simulated_annealing(objective, bounds, temp, cooling_rate, step_size,
                        max_iter, temp_min):
    current_solution = [random.uniform(b[0], b[1]) for b in bounds]
    current_value = objective(current_solution)

    best_solution = current_solution[:]
    best_value = current_value

    iteration = 0

    while temp > temp_min and iteration < max_iter:
        new_solution = get_neighbor(current_solution, step_size)
        new_value = objective(new_solution)

        delta = new_value - current_value
        if delta < 0:
            current_solution = new_solution
            current_value = new_value
```

```
        if new_value < best_value:
            best_solution = new_solution
            best_value = new_value

        else:
            acceptance_prob = math.exp(-delta / temp)
            if random.random() < acceptance_prob:
                current_solution = new_solution
                current_value = new_value

        temp = temp * cooling_rate

        iteration += 1

        print(f"Iteration {iteration}, Current Solution: {current_solution}, "
              f"Current Value: {current_value}, Temperature: {temp}")

    return best_solution, best_value

if __name__ == "__main__":

    bounds = [(-5.12, 5.12) for _ in range(2)]

    initial_temp = 1000
    cooling_rate = 0.99
    step_size = 0.1
    max_iterations = 1000
    temp_min = 1e-3

    best_solution, best_value = simulated_annealing(objective_function,
        bounds, initial_temp, cooling_rate, step_size, max_iterations,
        temp_min)
```

Date __/__/__
Page _____

```
print("Best Solution:", best_solution)
print("Best Objective Value:", best_value)
```

Output:

Best Solution: [-3.97347601, 0.99335 9981664]
Best Objective Value: 16.922545902851

22+10=24