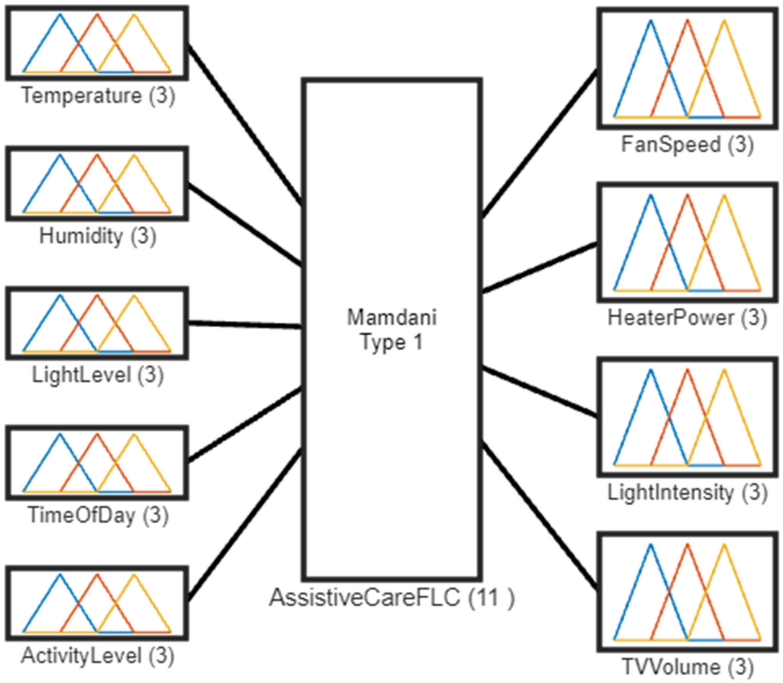
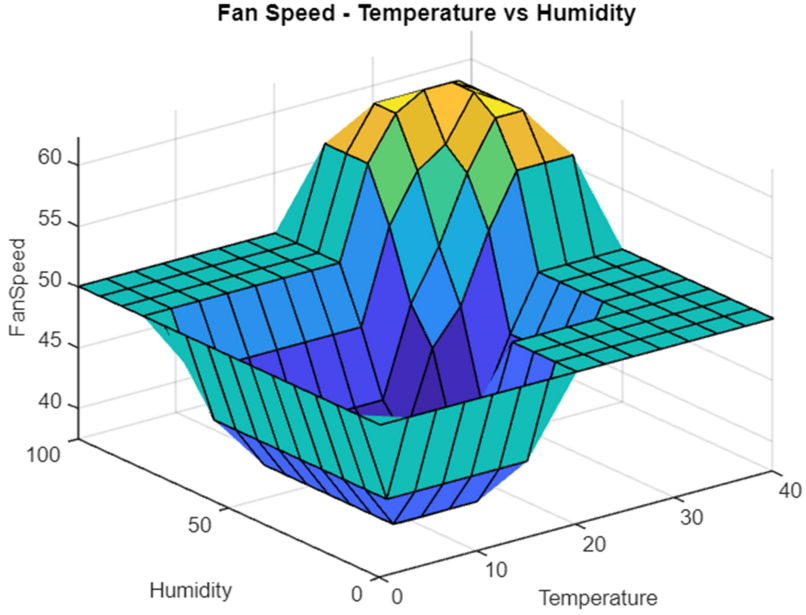
```
Part-1 : Design and Implementation of the FLC

% Create a new FIS
fis = mamfis('Name', 'AssistiveCareFLC');
% Define input variables with sufficient range and overlap
% Temperature
fis = addInput(fis, [0 40], 'Name', 'Temperature');
fis = addMF(fis, 'Temperature', 'trimf', [0 10 20], 'Name', 'Cold');
fis = addMF(fis, 'Temperature', 'trimf', [10 20 30], 'Name', 'Comfortable');
fis = addMF(fis, 'Temperature', 'trimf', [20 30 40], 'Name', 'Hot');


% Humidity
fis = addInput(fis, [0 100], 'Name', 'Humidity');
fis = addMF(fis, 'Humidity', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'Humidity', 'trimf', [25 50 75], 'Name', 'Medium');
fis = addMF(fis, 'Humidity', 'trimf', [50 75 100], 'Name', 'High');


% Light Level
fis = addInput(fis, [0 100], 'Name', 'LightLevel');
fis = addMF(fis, 'LightLevel', 'trimf', [0 25 50], 'Name', 'Dark');
fis = addMF(fis, 'LightLevel', 'trimf', [25 50 75], 'Name', 'Moderate');
fis = addMF(fis, 'LightLevel', 'trimf', [50 75 100], 'Name', 'Bright');


% Time of Day
fis = addInput(fis, [0 24], 'Name', 'TimeOfDay');
fis = addMF(fis, 'TimeOfDay', 'trimf', [0 6 12], 'Name', 'Morning');
fis = addMF(fis, 'TimeOfDay', 'trimf', [6 12 18], 'Name', 'Afternoon');
fis = addMF(fis, 'TimeOfDay', 'trimf', [12 18 24], 'Name', 'Night');


% User Activity Level
fis = addInput(fis, [0 100], 'Name', 'ActivityLevel');
fis = addMF(fis, 'ActivityLevel', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'ActivityLevel', 'trimf', [25 50 75], 'Name', 'Medium');
fis = addMF(fis, 'ActivityLevel', 'trimf', [50 75 100], 'Name', 'High');


% Define output variables
% Fan Speed
fis = addOutput(fis, [0 100], 'Name', 'FanSpeed');
fis = addMF(fis, 'FanSpeed', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'FanSpeed', 'trimf', [25 50 75], 'Name', 'Medium');
fis = addMF(fis, 'FanSpeed', 'trimf', [50 75 100], 'Name', 'High');


% Heater Power
fis = addOutput(fis, [0 100], 'Name', 'HeaterPower');
fis = addMF(fis, 'HeaterPower', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'HeaterPower', 'trimf', [25 50 75], 'Name', 'Medium');
```

```
fis = addMF(fis, 'HeaterPower', 'trimf', [50 75 100], 'Name', 'High');


% Light Intensity
fis = addOutput(fis, [0 100], 'Name', 'LightIntensity');
fis = addMF(fis, 'LightIntensity', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'LightIntensity', 'trimf', [25 50 75], 'Name', 'Medium');
fis = addMF(fis, 'LightIntensity', 'trimf', [50 75 100], 'Name', 'High');


% TV Volume
fis = addOutput(fis, [0 100], 'Name', 'TVVolume');
fis = addMF(fis, 'TVVolume', 'trimf', [0 25 50], 'Name', 'Low');
fis = addMF(fis, 'TVVolume', 'trimf', [25 50 75], 'Name', 'Medium');
fis = addMF(fis, 'TVVolume', 'trimf', [50 75 100], 'Name', 'High');


% Define rules
rules = [
    "Temperature==Cold & Humidity==Low => FanSpeed=Low, HeaterPower=High,
LightIntensity=Low, TVVolume=Low"
    "Temperature==Cold & Humidity==Medium => FanSpeed=Low, HeaterPower=High,
LightIntensity=Medium, TVVolume=Medium"
    "Temperature==Comfortable & Humidity==Medium => FanSpeed=Low, HeaterPower=Low,
LightIntensity=High, TVVolume=High"
    "Temperature==Hot & Humidity==High => FanSpeed=High, HeaterPower=Low,
LightIntensity=Low, TVVolume=Low"
    "Temperature==Hot & Humidity==Low => FanSpeed=Medium, HeaterPower=Low,
LightIntensity=High, TVVolume=Medium"
    "Temperature==Comfortable & Humidity==High => FanSpeed=Medium, HeaterPower=Low,
LightIntensity=Medium, TVVolume=Medium"
    "Temperature==Cold & LightLevel==Dark => FanSpeed=Low, HeaterPower=High,
LightIntensity=Medium, TVVolume=Medium"
    "Temperature==Hot & LightLevel==Bright => FanSpeed=High, HeaterPower=Low,
LightIntensity=High, TVVolume=Low"
    "TimeOfDay==Night & ActivityLevel==Low => FanSpeed=Low, HeaterPower=Low,
LightIntensity=Low, TVVolume=Low"
    "TimeOfDay==Morning & ActivityLevel==High => FanSpeed=Medium, HeaterPower=High,
LightIntensity=High, TVVolume=Medium"
    "TimeOfDay==Afternoon & ActivityLevel==Medium => FanSpeed=Medium, HeaterPower=Low,
LightIntensity=High, TVVolume=Medium"
];


% Add rules to the FIS
fis = addRule(fis, rules);


% Display FIS structure
figure;
plotfis(fis);
```

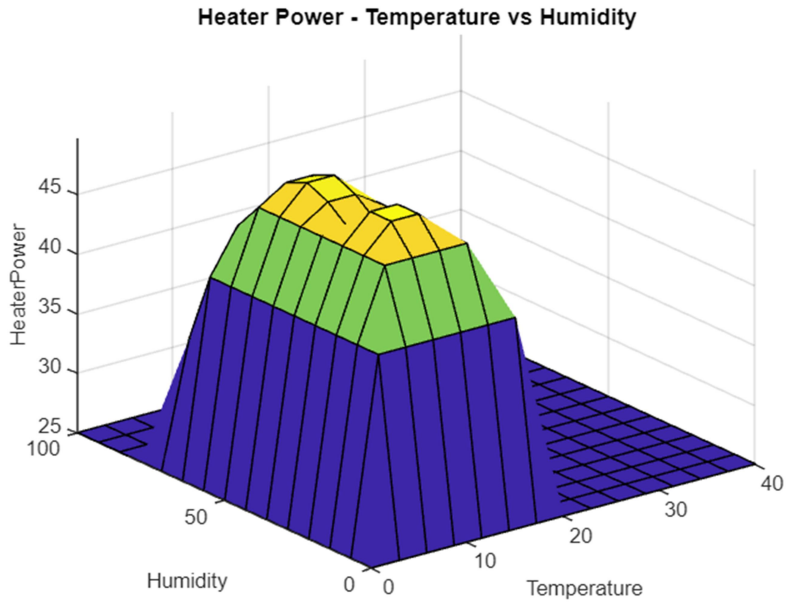


System AssistiveCareFLC: 5 inputs, 4 outputs, 11 rules

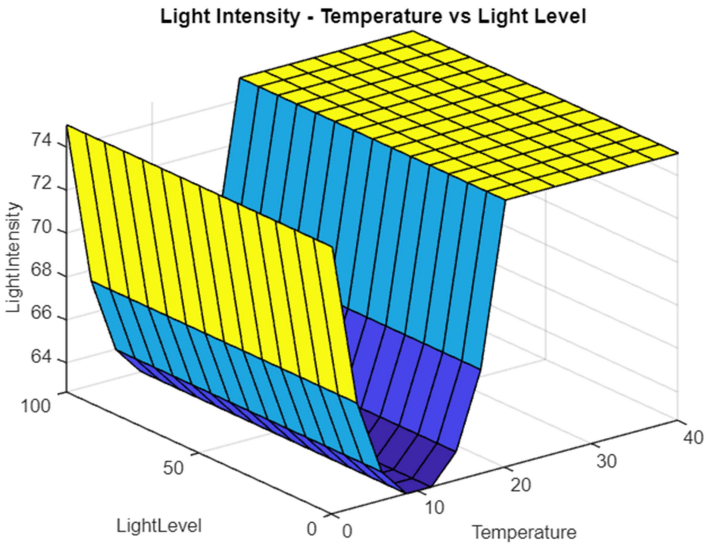
```
% View the control surface plot for Fan Speed output
figure;
gensurf(fis, [1 2], 1);
title('Fan Speed - Temperature vs Humidity');
```



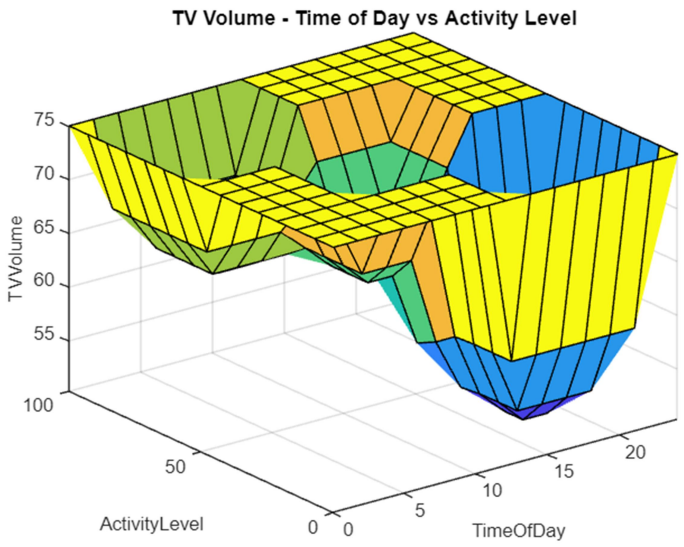
```
% View the control surface plot for Heater Power output
figure;
gensurf(fis, [1 2], 2);
title('Heater Power - Temperature vs Humidity');
```




```
% View the control surface plot for Light Intensity output
figure;
gensurf(fis, [1 3], 3);
title('Light Intensity - Temperature vs Light Level');
```



```
% View the control surface plot for TV Volume output
figure;
gensurf(fis, [4 5], 4);
title('TV Volume - Time of Day vs Activity Level');
```




```
% Test FIS with an example input
input = [20 50 40 14 50]; % Example input: [Temp, Humidity, Light Level, Time of Day,
Activity Level]
output = evalfis(fis, input);
disp(['Fan Speed: ', num2str(output(1))]);
```

```
Fan Speed: 36.6499
```

```
disp(['Heater Power: ', num2str(output(2))]);
```

```
Heater Power: 25
```

```
disp(['Light Intensity: ', num2str(output(3))]);
```

```
Light Intensity: 75
```

```
disp(['TV Volume: ', num2str(output(4))]);
```

```
TV Volume: 63.3501
```

```
% To view the rules, use the Fuzzy Logic Designer app:
fuzzyLogicDesigner(fis);
```

Task 2: Compare different optimization techniques on CEC’2005 functions.

```
% Benchmark Functions Comparison

% Define test functions
```

```
functions = {@(x) sum(x.^2), ... % Sphere function
             @(x) sum(100*(x(2:end)-x(1:end-1).^2).^2 + (1-x(1:end-1)).^2), ... %
Rosenbrock function
             @(x) -20*exp(-0.2*sqrt(sum(x.^2)/numel(x))) -
exp(sum(cos(2*pi*x))/numel(x)) + 20 + exp(1)}; % Ackley function

function_names = {'Sphere', 'Rosenbrock', 'Ackley'};

% Dimensions to test
dimensions = [2, 10];

% Number of runs
num_runs = 15;

% Optimization algorithms to compare
algorithms = {@simple_ga, @simple_pso, @simple_sa};
alg_names = {'Simple Genetic Algorithm', 'Simple Particle Swarm Optimization',
'Simple Simulated Annealing'};

% Results storage
results = struct();

% Convergence history storage
convergence_history = cell(length(dimensions), length(functions),
length(algorithms));

% Create 3D surface plots for each function
create_3d_plots(functions, function_names);

for d = 1:length(dimensions)
    D = dimensions(d);

    for f = 1:length(functions)
        func = functions{f};
        func_name = function_names{f};

        % Set bounds for optimization
        lb = -100 * ones(1, D);
        ub = 100 * ones(1, D);

        for a = 1:length(algorithms)
            alg = algorithms{a};
            alg_name = alg_names{a};

            fprintf('Testing %s on %s (D=%d)\n', alg_name, func_name, D);

            % Run optimization multiple times
            fitness_values = zeros(1, num_runs);
            for run = 1:num_runs
                [~, fval, history] = alg(func, D, lb, ub);
                fitness_values(run) = fval;

                % Store convergence history for the first run
                if run == 1
                    convergence_history{d, f, a} = history;
                end
            end

            % Store results
```

```
            results(d,f,a).algorithm = alg_name;
            results(d,f,a).function = func_name;
            results(d,f,a).dimension = D;
            results(d,f,a).mean = mean(fitness_values);
            results(d,f,a).std = std(fitness_values);
            results(d,f,a).best = min(fitness_values);
            results(d,f,a).worst = max(fitness_values);
        end
    end
end

% Display results
for d = 1:length(dimensions)
    for f = 1:length(functions)
        fprintf('\nResults for %s (D=%d):\n', function_names{f}, dimensions(d));
        for a = 1:length(algorithms)
            r = results(d,f,a);
            fprintf('%s:\n', r.algorithm);
            fprintf('  Mean: %.4e (±%.4e)\n', r.mean, r.std);
            fprintf('  Best: %.4e\n', r.best);
            fprintf('  Worst: %.4e\n', r.worst);
        end
    end
end

% Plot convergence
for d = 1:length(dimensions)
    for f = 1:length(functions)
        figure;
        hold on;
        for a = 1:length(algorithms)
            history = convergence_history{d, f, a};
            plot(1:length(history), history, 'DisplayName', alg_names{a});
        end
        xlabel('Iterations');
        ylabel('Fitness Value');
        title(sprintf('Convergence for %s (D=%d)', function_names{f},
dimensions(d)));
        legend('show');
        set(gca, 'YScale', 'log');  % Use log scale for y-axis
        hold off;
    end
end

% Function to create 3D surface plots
function create_3d_plots(functions, function_names)
    [X, Y] = meshgrid(-5:0.1:5, -5:0.1:5);
    for i = 1:length(functions)
        Z = zeros(size(X));
        for j = 1:size(X, 1)
            for k = 1:size(X, 2)
                Z(j, k) = functions{i}([X(j, k), Y(j, k)]);
            end
        end
        figure;
        surf(X, Y, Z);
        title(['3D Surface Plot of ', function_names{i}, ' Function']);
        xlabel('x');
        ylabel('y');
```

```
        zlabel('z');
    end
end

% Simple Genetic Algorithm
function [best_solution, best_fitness, history] = simple_ga(func, D, lb, ub)
    pop_size = 50;
    num_generations = 100;
    mutation_rate = 0.1;

    population = lb + (ub - lb) .* rand(pop_size, D);
    fitness = zeros(pop_size, 1);
    history = zeros(1, num_generations);

    for gen = 1:num_generations
        for i = 1:pop_size
            fitness(i) = func(population(i, :));
        end

        [best_fitness, idx] = min(fitness);
        best_solution = population(idx(1), :);
        history(gen) = best_fitness;

        % Select top 2 individuals (or clone the best if only one unique fitness
value)
        [~, sorted_idx] = sort(fitness);
        if length(unique(fitness)) == 1
            new_pop = repmat(population(sorted_idx(1), :), 2, 1);
        else
            new_pop = population(sorted_idx(1:2), :);
        end

        while size(new_pop, 1) < pop_size
            parents = population(randperm(pop_size, 2), :);
            child = mean(parents, 1);
            if rand < mutation_rate
                child = child + randn(1, D) .* (ub - lb) * 0.1;
            end
            child = max(min(child, ub), lb);  % Ensure child is within bounds
            new_pop = [new_pop; child];
        end
        population = new_pop;
    end
end

% Simple Particle Swarm Optimization
function [best_solution, best_fitness, history] = simple_pso(func, D, lb, ub)
    num_particles = 50;
    num_iterations = 100;
    w = 0.7;
    c1 = 1.4;
    c2 = 1.4;

    positions = lb + (ub - lb) .* rand(num_particles, D);
    velocities = zeros(num_particles, D);
    personal_best_pos = positions;
    personal_best_fit = inf(num_particles, 1);
    global_best_pos = zeros(1, D);
    global_best_fit = inf;
```

```
    history = zeros(1, num_iterations);

    for iter = 1:num_iterations
        for i = 1:num_particles
            fitness = func(positions(i, :));
            if fitness < personal_best_fit(i)
                personal_best_fit(i) = fitness;
                personal_best_pos(i, :) = positions(i, :);
            end
            if fitness < global_best_fit
                global_best_fit = fitness;
                global_best_pos = positions(i, :);
            end
        end

        history(iter) = global_best_fit;

        for i = 1:num_particles
            r1 = rand(1, D);
            r2 = rand(1, D);
            velocities(i, :) = w * velocities(i, :) + ...
                c1 * r1 .* (personal_best_pos(i, :) - positions(i, :)) + ...
                c2 * r2 .* (global_best_pos - positions(i, :));
            positions(i, :) = positions(i, :) + velocities(i, :);
            positions(i, :) = max(min(positions(i, :), ub), lb);
        end
    end
    best_solution = global_best_pos;
    best_fitness = global_best_fit;
end

% Simple Simulated Annealing
function [best_solution, best_fitness, history] = simple_sa(func, D, lb, ub)
    max_iter = 1000;
    T0 = 100;
    Tf = 1e-8;
    alpha = 0.95;

    current_solution = lb + (ub - lb) .* rand(1, D);
    current_energy = func(current_solution);
    best_solution = current_solution;
    best_fitness = current_energy;
    T = T0;
    history = zeros(1, max_iter);

    for i = 1:max_iter
        neighbor = current_solution + randn(1, D) .* (ub - lb) * 0.1;
        neighbor = max(min(neighbor, ub), lb);
        neighbor_energy = func(neighbor);

        if neighbor_energy < current_energy || rand < exp((current_energy -
neighbor_energy) / T)
            current_solution = neighbor;
            current_energy = neighbor_energy;
        end

        if current_energy < best_fitness
            best_solution = current_solution;
            best_fitness = current_energy;
```

```
        end

        history(i) = best_fitness;

        T = T0 * alpha^i;
        if T < Tf
            break;
        end
    end
    history = history(1:i);  % Trim unused elements if SA terminates early
end
```

OUTPUT FOR THE TASK 2:

Testing Simple Genetic Algorithm on Sphere (D=2)

Testing Simple Particle Swarm Optimization on Sphere (D=2)

Testing Simple Simulated Annealing on Sphere (D=2)

Testing Simple Genetic Algorithm on Rosenbrock (D=2)

Testing Simple Particle Swarm Optimization on Rosenbrock (D=2)

Testing Simple Simulated Annealing on Rosenbrock (D=2)

Testing Simple Genetic Algorithm on Ackley (D=2)

Testing Simple Particle Swarm Optimization on Ackley (D=2)

Testing Simple Simulated Annealing on Ackley (D=2)

Testing Simple Genetic Algorithm on Sphere (D=10)

Testing Simple Particle Swarm Optimization on Sphere (D=10)

Testing Simple Simulated Annealing on Sphere (D=10)

Testing Simple Genetic Algorithm on Rosenbrock (D=10)

Testing Simple Particle Swarm Optimization on Rosenbrock (D=10)

Testing Simple Simulated Annealing on Rosenbrock (D=10)

Testing Simple Genetic Algorithm on Ackley (D=10)

Testing Simple Particle Swarm Optimization on Ackley (D=10)

Testing Simple Simulated Annealing on Ackley (D=10)

Results for Sphere (D=2):

Simple Genetic Algorithm:

Mean: 1.4624e-02 (±1.9850e-02)

Best: 1.2906e-04

Worst: 7.3739e-02

Simple Particle Swarm Optimization:

Mean: 5.4667e-12 (±6.9601e-12)

Best: 1.4738e-13

Worst: 2.4972e-11

Simple Simulated Annealing:

Mean: 1.3228e+00 (±1.2321e+00)

Best: 9.5640e-02

Worst: 4.1638e+00

Results for Rosenbrock (D=2):

Simple Genetic Algorithm:

Mean: 1.2138e+00 (±3.5358e+00)

Best: 1.9321e-04

Worst: 1.3904e+01

Simple Particle Swarm Optimization:

Mean: 8.6764e+00 (±2.3274e+01)

Best: 1.4238e-07

Worst: 7.9126e+01

Simple Simulated Annealing:

Mean: 4.0320e+01 (±3.8652e+01)

Best: 1.4679e-02

Worst: 1.2542e+02

Results for Ackley (D=2):

Simple Genetic Algorithm:

Mean: 5.2961e-01 (±6.7402e-01)

Best: 5.5399e-02

Worst: 2.4639e+00

Simple Particle Swarm Optimization:

Mean: 1.5310e-05 (±3.6214e-05)

Best: 6.7158e-07

Worst: 1.4464e-04

Simple Simulated Annealing:

Mean: 1.1164e+01 (±7.2547e+00)

Best: 1.5165e+00

Worst: 2.0000e+01

Results for Sphere (D=10):

Simple Genetic Algorithm:

Mean: 1.7861e+02 (±1.2570e+02)

Best: 3.8571e+01

Worst: 4.7625e+02

Simple Particle Swarm Optimization:

Mean: 2.6056e-04 (±2.2351e-04)

Best: 3.7404e-05

Worst: 8.7041e-04

Simple Simulated Annealing:

Mean: 8.9797e+02 (±1.9746e+02)

Best: 6.1791e+02

Worst: 1.2555e+03

Results for Rosenbrock (D=10):

Simple Genetic Algorithm:

Mean: 2.4031e+05 (±3.0497e+05)

Best: 2.1467e+04

Worst: 1.0296e+06

Simple Particle Swarm Optimization:

Mean: 2.0014e+05 (±4.1397e+05)

Best: 7.9673e+00

Worst: 1.0000e+06

Simple Simulated Annealing:

Mean: 1.3675e+07 (±6.5277e+06)

Best: 4.1877e+06

Worst: 3.0301e+07

Results for Ackley (D=10):

Simple Genetic Algorithm:

Mean: 1.1587e+01 (±1.8613e+00)

Best: 8.9594e+00

Worst: 1.5076e+01

Simple Particle Swarm Optimization:

Mean: 2.0017e+01 (±4.1512e-02)

Best: 2.0000e+01

Worst: 2.0155e+01

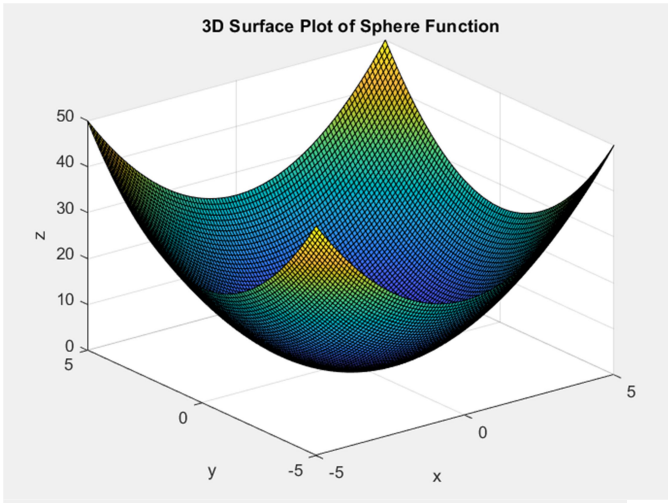
Simple Simulated Annealing:

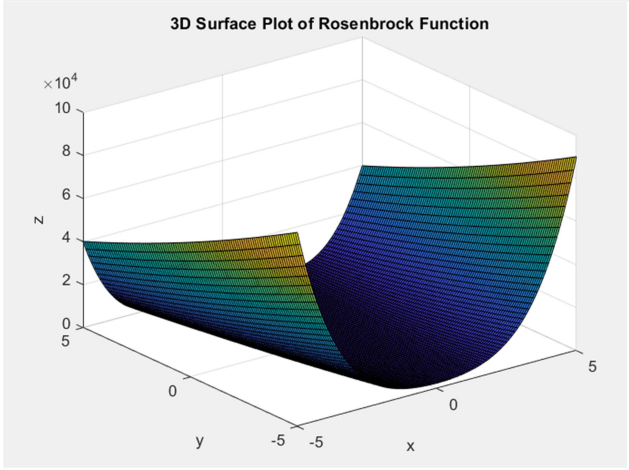
Mean: 2.0478e+01 (±1.5931e-01)

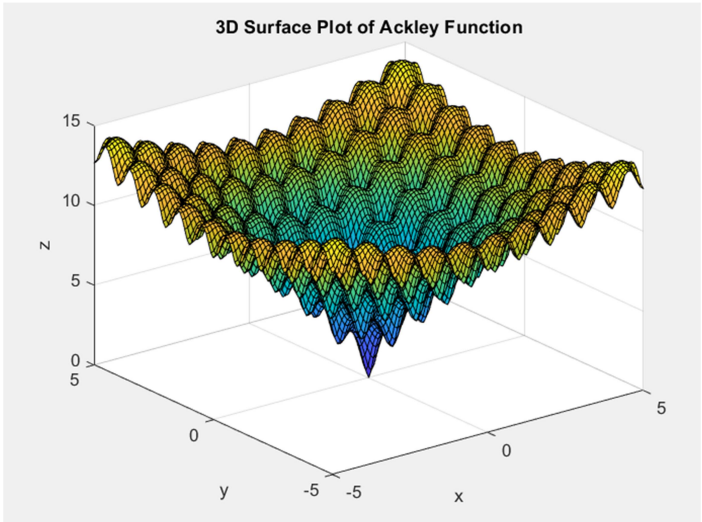
Best: 2.0237e+01

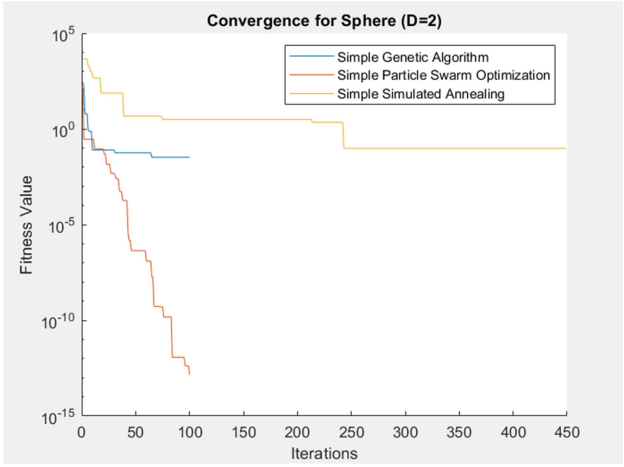
Worst: 2.0827e+01

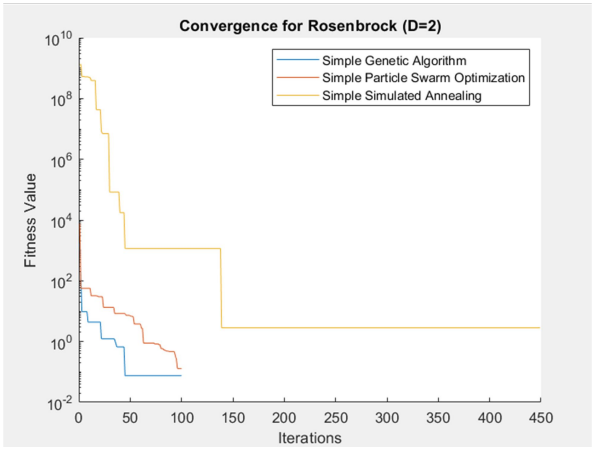
AND GRAPHS OF TASK 2:

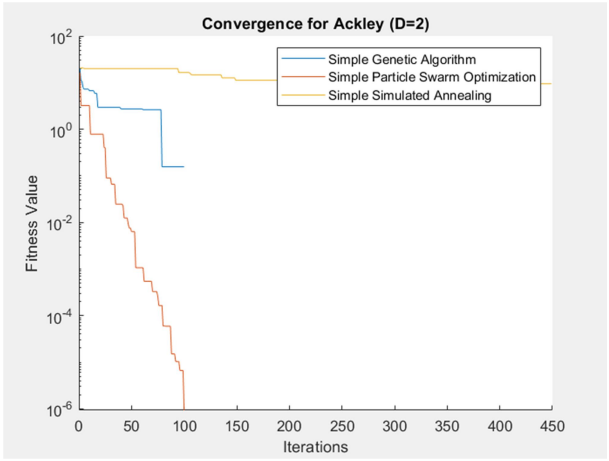

3D Surface Plot of Sphere Function
50
40
30
20
10
0
z
5
0
y
-5
-5
0
5
x

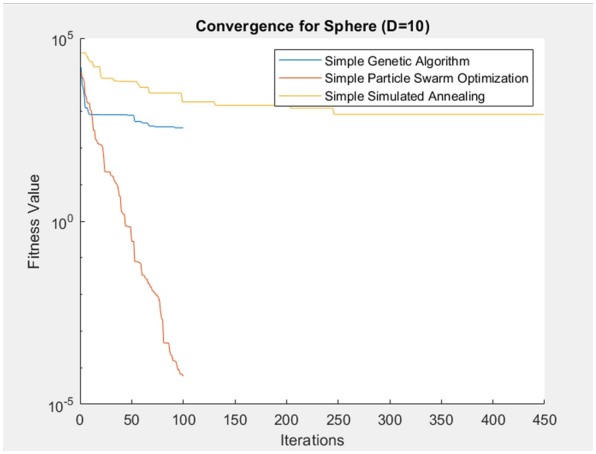

3D Surface Plot of Rosenbrock Function
×10^4
10
8
6
4
2
0
z
5
0
y
-5
-5
0
5
x

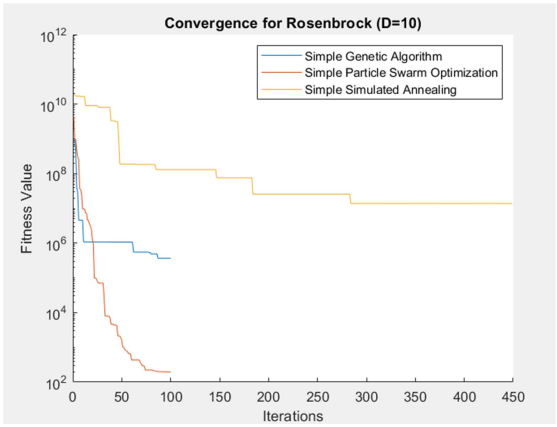

3D Surface Plot of Ackley Function
15
10
5
0
z
5
0
y
-5
-5
0
5
x

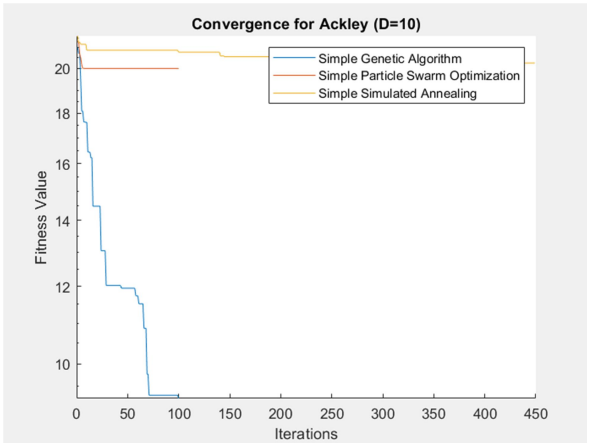

Convergence for Sphere (D=2)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations


Convergence for Rosenbrock (D=2)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations


Convergence for Ackley (D=2)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations


Convergence for Sphere (D=10)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations


Convergence for Rosenbrock (D=10)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations


Convergence for Ackley (D=10)
Simple Genetic Algorithm
Simple Particle Swarm Optimization
Simple Simulated Annealing
Fitness Value
Iterations