Apply the Joint Optimization in the given BRB Problem

1) Apply structure optimization in the given BRB tree by using any of the algorithms.
2) Apply parameter optimization by using fmincon function, Genetic Algorithm Function and also by using PSO which are available in the MATLAB.
3) Apply the joint optimization. You need to consider both conjunctive BRB and Disjunctive BRB in each of the cases.

Dataset:

Input No	Input			Output
	X_1	X_2	X_3	Y_M
1	0.8	0.6	0.2	0.9
2	0.8	0.2	0.5	0.7
3	0.4	0.4	0.3	0.5
4	0.2	0.8	0.7	0.4
5	0.4	0.0	0.2	0.3
6	1.0	0.1	0.7	0.8
7	0.1	0.4	0.3	0.2
8	0.5	0.6	0.2	0.7
9	0.3	0.9	0.2	0.7
10	0.4	0.5	0.2	0.6

Number of Input-Output Pair, M = 10

Number of Attributes, T = 3

Answer to the question no 01

The heuristics method is used to find the optimal value for the structure of the system. We try to find the optimal number of referential values which will lead to least error.

To satisfy this we have taken the same seed to generate the belief degrees and checked for minimum mean squared error taking 3 to 8 as candidate solution for number of referential values.

For disjunctive method, we get that for number of referential values, N = 4, we get the minimum mean squared error, that is 0.0311

RMS error for other candidates were:

```
Number of Reference Value - Error Pair:
    3.0000    0.0756
    4.0000    0.0311
    5.0000    0.0555
    6.0000    0.0570
    7.0000    0.0600
    8.0000    0.0491
```

The belief degree when the error was minimum is (One Row for Each Rule):

```
0.7754    0.0542    0.0770    0.0935
0.3122    0.3688    0.2674    0.0517
0.4542    0.0693    0.0738    0.4026
0.1420    0.2824    0.3236    0.2519
```

For conjunctive method, we get that for number of referential values, N = 3, we get the minimum mean squared error, that is 0.0468

RMS error for other candidates were:

```
Number of Reference Value - Error Pair:
    3.0000    0.0468
    4.0000    0.0543
    5.0000    0.0582
    6.0000    0.0480
    7.0000    0.0597
    8.0000    0.0606
```

The belief degree when the error was minimum is (One Row for Each Rule):

```
    0.6235    0.1584    0.2181
    0.2039    0.3235    0.4726
    0.1606    0.3051    0.5343
    0.1574    0.7454    0.0972
    0.3713    0.2194    0.4093
    0.5104    0.4878    0.0018
    0.3766    0.4769    0.1465
```

0.3867	0.4327	0.1806
0.0834	0.4079	0.5087
0.5652	0.1170	0.3178
0.3123	0.2375	0.4502
0.1775	0.5747	0.2478
0.1426	0.0863	0.7711
0.3528	0.3608	0.2865
0.2894	0.2553	0.4554
0.4265	0.1663	0.4072
0.0904	0.5010	0.4086
0.5469	0.3846	0.0685
0.3901	0.5606	0.0493
0.0171	0.2332	0.7497
0.2908	0.4952	0.2139
0.3826	0.2313	0.3861
0.1011	0.2086	0.6903
0.4837	0.0019	0.5144
0.3578	0.4396	0.2026
0.1498	0.4248	0.4254
0.1485	0.8031	0.0484

<u>Answer to the question no 02</u>

For demonstrating the parameter optimization for both conjunctive and disjunctive BRB using the MATLAB functions *fmincon* and *ga*, the codes were developed which is attached in the code section under the names

“Parameter Optimization Using *fmincon* (Disjunctive)”,
“Parameter Optimization Using *fmincon* (Conjunctive)”,
“Parameter Optimization Using *ga* (Disjunctive)”
“Parameter Optimization Using *ga* (Conjunctive)”

The code using MATLAB function *paerticleswarm* was not developed, because this function for particle swarm does not support linear equality constraints, which upholds very important characteristics of BRB.

Answer to the question no 03

For demonstrating the joint optimization, we consider the belief degrees and optimal number of reference values achieved from structure optimization and use this to tune the structure of the BRB. Then we use the MATLAB function *fmincon* to further tune the parameters of the system.

For disjunctive BRB the least root mean square error becomes: 0.0311 to 0.0214

Computation time reduced from 13.219 seconds to 10.175 seconds.

And Optimal Belief degrees for BRB are:

```
0.9996    0.0002    0.0001    0.0000
0.7666    0.0006    0.0007    0.2321
0.6472    0.0003    0.0025    0.3500
0.0002    0.0003    0.0005    0.9991
```

For conjunctive BRB the least root mean square error becomes: 0.0468 to 0.0029

Computation time reduced from 13.340 seconds to 9.910 seconds.

And Optimal Belief degrees for BRB are:

```
0.3295    0.3393    0.3311
0.5997    0.2776    0.1227
0.4054    0.3207    0.2739
0.3817    0.3140    0.3043
0.9437    0.0394    0.0169
0.8401    0.1028    0.0572
0.5730    0.2686    0.1583
0.6413    0.2192    0.1394
0.4481    0.3104    0.2415
0.2271    0.3406    0.4323
0.5849    0.2589    0.1562
0.7058    0.2196    0.0746
0.2685    0.3417    0.3898
0.6435    0.1737    0.1828
0.4471    0.2782    0.2747
0.3281    0.3442    0.3277
0.1214    0.2943    0.5843
0.1976    0.2587    0.5437
0.1692    0.3028    0.5280
0.2203    0.3141    0.4656
0.5341    0.2879    0.1780
0.2268    0.3304    0.4428
0.0764    0.2958    0.6277
0.2251    0.3677    0.4072
0.3295    0.3304    0.3401
0.2937    0.3218    0.3845
0.2931    0.3120    0.3949
```

Codes:

Structure Optimization (Disjunctive):

```
% BRB Structure Optimization (Disjunctive Approach)

% Finding Optimal Number of Reference Values
% Number of Rules Depends on Number of Reference Values.
% So, Finding optimal Number of Reference Values will lead to Optimal
Number of Rules

% Training Dataset
M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
train_input = [0.8 0.6 0.2;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.1 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];

train_output = [0.9;
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.2;
    0.7;
    0.7;
    0.6];

no_of_solution_candidate_considered = 6; % Number of solution candidate
solution_candidates = 3:(3 + no_of_solution_candidate_considered - 1); %
Solution Candidates

seed = rng; % Preserve seed of random generator, so that we can generate
the same belief degree later

% Initialize Variables
calculated_output = zeros(M,1);
differences = zeros(M,1);
result = zeros(no_of_solution_candidate_considered, 2);

for x = 1:no_of_solution_candidate_considered

    N = solution_candidates(1,x); % No of referential values
    L = N; % Number of rules (Because, We Are Considering Disjunctive
Inference)

    % Referential Values
    ref_val = generate_ref_val(N,1,0);

    % Initial Belief Degree
```

```
    belief_degree = generate_belief_degree(N,L,seed);  % Initial Belief
Degrees of size (N,L)

    % Inference Methodology
    for i = 1:M
        % Get Rule Weights
        weights = get_rule_weights(train_input,i,T,N,ref_val);

        % Calculate Aggregated Belief Degree and Compute Y
        aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degree, N, L);

        % Calculate Difference Between Outputs
        calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
        differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
    end
    result(x,:) = [N, sum(differences,'all')];   % Number of ref val -
total_difference result pair
end

disp("Number of Reference Value - Error Pair:")
disp(result)

optimal_ref_val_number = find_optimal_ref_val_number(result,
no_of_solution_candidate_considered)
belief_degree_used = generate_belief_degree(optimal_ref_val_number,
optimal_ref_val_number,seed)


function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L,seed)
    rng(seed);
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
        end
    end

    for row = 1:N
        for col = 1:L
```

```
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
    end
end

function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val); % Calculate
Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
              arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
              arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    arr = zeros(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_attributes
            arr(i,1) = arr(i,1) + individual_matching_degree(j,i);
        end

    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
       val = val + matching_degrees(i,1);
    end
end
```

```
function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end

function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
    for i = 1:no_of_rules
```

```
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end

function no_of_ref_val = find_optimal_ref_val_number(result_pair_array,
no_of_solution_candidate_considered)
    value = result_pair_array(1,2);
    ref_no = result_pair_array(1,1);
    for i = 2:no_of_solution_candidate_considered
        if result_pair_array(i,2) <= value
            ref_no = result_pair_array(i,1);
            value = result_pair_array(i,2);
        end
    end
    no_of_ref_val = ref_no;
end
```

Structure Optimization (Conjunctive):

```
% BRB Structure Optimization (Conjunctive Approach)

% Finding Optimal Number of Reference Values
% Number of Rules Depends on Number of Reference Values.
% So, Finding optimal Number of Reference Values will lead to Optimal
Number of Rules

% Training Dataset
M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
train_input = [0.8 0.6 0.2;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.1 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];

train_output = [0.9;
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.2;
    0.7;
    0.7;
    0.6];

no_of_solution_candidate_considered = 6; % Number of solution candidate
solution_candidates = 3:(3 + no_of_solution_candidate_considered - 1); %
Solution Candidates

seed = rng; % Preserve seed of random generator, so that we can generate
the same belief degree later

% Initialize Variables
calculated_output = zeros(M,1);
differences = zeros(M,1);
result = zeros(no_of_solution_candidate_considered, 2);

for x = 1:no_of_solution_candidate_considered

    N = solution_candidates(1,x); % No of referential values
    L = N^T; % Number of rules (Because, We Are Considering Conjunctive
Inference)

    % Referential Values
    ref_val = generate_ref_val(N,1,0);

    % Initial Belief Degree
    belief_degree = generate_belief_degree(N,L,seed);  % Initial Belief
Degrees of size (N,L)
```

```
    % Inference Methodology
    for i = 1:M
        % Get Rule Weights
        weights = get_rule_weights(train_input,i,T,N,ref_val);

        % Calculate Aggregated Belief Degree and Compute Y
        aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degree, N, L);

        % Calculate Difference Between Outputs
        calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
        differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
    end

    result(x,:) = [N, sum(differences,'all')];   % Number of ref val -
total_difference result pair
end

disp("Number of Reference Value - Error Pair:")
disp(result)

optimal_ref_val_number = find_optimal_ref_val_number(result,
no_of_solution_candidate_considered)
belief_degree_used = generate_belief_degree(optimal_ref_val_number,
optimal_ref_val_number^T,seed)

function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L,seed)
    rng(seed);
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
        end
    end

    for row = 1:N
        for col = 1:L
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
```

```
    end
end

function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val^no_of_attribu
tes); % Calculate Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
                arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
                arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    no_of_matching_degree = no_of_ref_val^no_of_attributes;
    arr = ones(no_of_matching_degree, 1);
    counter = 0;
    for i = 1:no_of_ref_val
        for j = 1:no_of_ref_val
            for k = 1:no_of_ref_val
                counter = counter + 1;
                arr(counter,1) = individual_matching_degree(1,i) *
individual_matching_degree(2,j) * individual_matching_degree(3,k);
            end
        end
    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
       val = val + matching_degrees(i,1);
    end
end
```

```
function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end

function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
```

```
    for i = 1:no_of_rules
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end

function no_of_ref_val = find_optimal_ref_val_number(result_pair_array,
no_of_solution_candidate_considered)
    value = result_pair_array(1,2);
    ref_no = result_pair_array(1,1);
    for i = 2:no_of_solution_candidate_considered
        if result_pair_array(i,2) <= value
            ref_no = result_pair_array(i,1);
            value = result_pair_array(i,2);
        end
    end
    no_of_ref_val = ref_no;
end
```

Parameter Optimization Using *fmincon* (Disjunctive):

```
% Initial Inputs
x0 = generate_belief_degree(3,3);  % Belief Degrees of size (N,L)

% Bound Constraints
lb = zeros(3,3); % Lower Bound
ub = ones(3,3); % Upper Bound

% Equality Constraints
% Equality Constraints are taken as row vector "Aeq" sized m*n.
% m = number of equality constraints
% n = number of elements in x0/solution
% Solver converts x0/solution into x0(:)/solution(:) to impose constraints
% beq is a column vector of m elements
aeq = zeros(3,9);
aeq(1,1:3) = [1 1 1];
aeq(2,4:6) = [1 1 1];
aeq(3,7:9) = [1 1 1];
beq = ones(3,1);



% Set nondefault solver options
options = optimoptions('fmincon','PlotFcn','optimplotfvalconstr');

% Solve
[solution,objectiveValue] =
fmincon(@objectiveFcn,x0,[],[],aeq,beq,lb,ub,[],...
    options);

% Clear variables
clearvars options

% Display Optimized Values
disp("Least Mean Square Error = ")
disp(objectiveValue)
disp("Optimized Belief Degrees (Left to Right One Row, Represents For One
Rule) = ")
disp(solution')

function f = objectiveFcn(optimInput)
% Training Dataset

M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
N = 3; % No of referencial values
L = N; % Number of rules (Disjunctive Assumption)
train_input = [0.8 0.6 0.4;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.8 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];

train_output = [1.0;
```

```
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.9;
    0.7;
    0.7;
    0.6];


% Define the variables
belief_degrees = optimInput;

ref_val = generate_ref_val(N,1,0);

calculated_output = zeros(M,1);
differences = zeros(M,1);

for i = 1:M
    weights = get_rule_weights(train_input,i,T,N,ref_val); % Rule Weights

    % Calculate Aggregated Belief Degree and Compute Y
    aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degrees, N, L);

    calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
    differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
end


% Define Objective Function
f = sum((differences).^2) / M;
end

function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L)
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
```

```
        end
    end

    for row = 1:N
        for col = 1:L
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
    end
end

function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val); % Calculate
Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
                arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
                arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    arr = zeros(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_attributes
            arr(i,1) = arr(i,1) + individual_matching_degree(j,i);
        end

    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
```

```
        val = val + matching_degrees(i,1);
    end
end

function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end
```

```
function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
    for i = 1:no_of_rules
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end
```

Parameter Optimization Using *fmincon* (Conjunctive):

```
% Initial Inputs
x0 = generate_belief_degree(4,4^3);  % Belief Degrees of size (N,L)

% Bound Constraints
lb = zeros(4,4^3); % Lower Bound of size (N,L)
ub = ones(4,4^3); % Upper Bound of size (N,L)

% Equality Constraints
% Equality Constraints are taken as row vector "Aeq" sized m*n.
% m = number of equality constraints
% n = number of elements in x0/solution
% Solver converts x0/solution into x0(:)/solution(:) to impose constraints
% beq is a column vector of m elements
aeq = zeros(4^3,4*(4^3)); % of size (L,L*N)
for i = 1:4^3
    temp1 = (i-1)*4 + 1; % (i-1)*N + 1
    temp2 = (i-1)*4 + 4; % (i-1)*N + 4
    aeq(i,temp1:temp2) = ones(1,4); % ones(1,N)
end
beq = ones(4^3,1); % of size (L,1)


% Set nondefault solver options
options = optimoptions('fmincon','PlotFcn','optimplotfvalconstr');

% Solve
[solution,objectiveValue] =
fmincon(@objectiveFcn,x0,[],[],aeq,beq,lb,ub,[],...
    options);

% Clear variables
clearvars options

% Display Optimized Values
disp("Least Mean Square Error = ")
disp(objectiveValue)
disp("Optimized Belief Degrees (Left to Right One Row, Represents For One
Rule) = ")
disp(solution')

function f = objectiveFcn(optimInput)
% Training Dataset

M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
N = 4; % No of referencial values
L = N^T; % Number of rules (Disjunctive Assumption)
train_input = [0.8 0.6 0.4;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.8 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];
```

```
train_output = [1.0;
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.9;
    0.7;
    0.7;
    0.6];


% Define the variables
belief_degrees = optimInput;

ref_val = generate_ref_val(N,1,0);

calculated_output = zeros(M,1);
differences = zeros(M,1);

for i = 1:M
    weights = get_rule_weights(train_input,i,T,N,ref_val); % Rule Weights

    % Calculate Aggregated Belief Degree and Compute Y
    aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degrees, N, L);

    calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
    differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
end


% Define Objective Function
f = sum((differences).^2) / M;
end

function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L)
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
```

```
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
        end
    end

    for row = 1:N
        for col = 1:L
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
    end
end

function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val^no_of_attribu
tes); % Calculate Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
              arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
              arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    no_of_matching_degree = no_of_ref_val^no_of_attributes;
    arr = ones(no_of_matching_degree, 1);
    counter = 0;
    for i = 1:no_of_ref_val
        for j = 1:no_of_ref_val
            for k = 1:no_of_ref_val
                counter = counter + 1;
                arr(counter,1) = individual_matching_degree(1,i) *
individual_matching_degree(2,j) * individual_matching_degree(3,k);
            end
        end
```

```
    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
       val = val + matching_degrees(i,1);
    end
end

function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
```

```
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end

function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
    for i = 1:no_of_rules
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end
```

Parameter Optimization Using *ga* (Disjunctive):

```
% This is the initial input to the ga
% Represents total number of elements
% Must be equal to n of the aeq
nVars = 3*3; % N*L

% Bound Constraints
lb = zeros(3,3); % Lower Bound
ub = ones(3,3); % Upper Bound

% Equality Constraints
% Equality Constraints are taken as row vector "Aeq" sized m*n.
% m = number of equality constraints
% n = number of elements in x0/solution
% Solver converts x0/solution into x0(:)/solution(:) to impose constraints
% beq is a column vector of m elements
aeq = zeros(3,9);
aeq(1,1:3) = [1 1 1];
aeq(2,4:6) = [1 1 1];
aeq(3,7:9) = [1 1 1];
beq = ones(3,1);

% Solve
[solution,objectiveValue] = ga(@objectiveFcn,nVars,[],[],aeq,beq,lb,ub);

optimized_belief_degree = structure_belief_degrees(solution, 3, 3)
optimal_value = objectiveValue

function f = objectiveFcn(~)
% Training Dataset

M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
N = 3; % No of referencial values
L = N; % Number of rules (Disjunctive Assumption)
train_input = [0.8 0.6 0.4;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.8 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];

train_output = [1.0;
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.9;
    0.7;
    0.7;
    0.6];
```

```
% Define the variables
belief_degrees = generate_belief_degree(N,L);

ref_val = generate_ref_val(N,1,0);

calculated_output = zeros(M,1);
differences = zeros(M,1);

for i = 1:M
    weights = get_rule_weights(train_input,i,T,N,ref_val); % Rule Weights

    % Calculate Aggregated Belief Degree and Compute Y
    aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degrees, N, L);

    calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
    differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
end


% Define Objective Function
f = sum((differences).^2) / M;
end

function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L)
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
        end
    end

    for row = 1:N
        for col = 1:L
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
    end
end
```

```
function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val); % Calculate
Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
              arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
              arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    arr = zeros(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_attributes
            arr(i,1) = arr(i,1) + individual_matching_degree(j,i);
        end

    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
       val = val + matching_degrees(i,1);
    end
end

function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
```

```
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end

function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
    for i = 1:no_of_rules
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
```

```
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end

function arr = structure_belief_degrees(solution, N, L)
    temp = zeros(L, N);
    for i = 1:L
        temp(i, 1:N) = solution((((i-1)*N) + 1) : (((i-1)*N) + N));
    end
    arr = temp;
end
```

<u>Parameter Optimization Using *ga* (Conjunctive):</u>

```
% This is the initial input to the ga
% Represents total number of elements
% Must be equal to n of the aeq
nVars = 4*(4^3); % N*L

% Bound Constraints
lb = zeros(4,4^3); % Lower Bound of size (N,L)
ub = ones(4,4^3); % Upper Bound of size (N,L)

% Equality Constraints
% Equality Constraints are taken as row vector "Aeq" sized m*n.
% m = number of equality constraints
% n = number of elements in x0/solution
% Solver converts x0/solution into x0(:)/solution(:) to impose constraints
% beq is a column vector of m elements
aeq = zeros(4^3,4*(4^3)); % of size (L,L*N)
for i = 1:4^3
    temp1 = (i-1)*4 + 1; % (i-1)*N + 1
    temp2 = (i-1)*4 + 4; % (i-1)*N + 4
    aeq(i,temp1:temp2) = ones(1,4); % ones(1,N)
end
beq = ones(4^3,1); % of size (L,1)

% Solve
[solution,objectiveValue] = ga(@objectiveFcn,nVars,[],[],aeq,beq,lb,ub);

optimized_belief_degree = structure_belief_degrees(solution, 4, (4^3))
optimal_value = objectiveValue

function f = objectiveFcn(~)
% Training Dataset

M = 10; % No of Inputs-Output Pair
T = 3; % No of Attributes
N = 3; % No of referencial values
L = N^T; % Number of rules (Disjunctive Assumption)
train_input = [0.8 0.6 0.4;
    0.8 0.2 0.5;
    0.4 0.4 0.3;
    0.2 0.8 0.7;
    0.4 0.0 0.2;
    1.0 0.1 0.7;
    0.8 0.4 0.3;
    0.5 0.6 0.2;
    0.3 0.9 0.2;
    0.4 0.5 0.2];

train_output = [1.0;
    0.7;
    0.5;
    0.4;
    0.3;
    0.8;
    0.9;
    0.7;
    0.7;
    0.6];
```

```
% Define the variables
belief_degrees = generate_belief_degree(N,L);

ref_val = generate_ref_val(N,1,0);

calculated_output = zeros(M,1);
differences = zeros(M,1);

for i = 1:M
    weights = get_rule_weights(train_input,i,T,N,ref_val); % Rule Weights

    % Calculate Aggregated Belief Degree and Compute Y
    aggregated_belief_degree = calc_aggregated_belief_degree(weights,
belief_degrees, N, L);

    calculated_output(i,1) =
calculateY(aggregated_belief_degree,ref_val,N);
    differences(i,1) = abs(calculated_output(i,1) - train_output(i,1));
end


% Define Objective Function
f = sum((differences).^2) / M;
end

function arr = generate_ref_val(no_of_ref_val, upper, lower)
    arr = zeros(1, no_of_ref_val);
    value_difference = (upper - lower)/(no_of_ref_val - 1);
    for i = 1:no_of_ref_val
        if i == 1
            arr(1,i) = upper;
        elseif i == no_of_ref_val
            arr(1,i) = lower;
        else
            arr(1,i) = (upper - (value_difference * (i-1)));
        end

    end
end

function arr = generate_belief_degree(N, L)
    belief_generator = rand(N,L);
    temp_gen_col_total = zeros(L,1);
    arr = zeros(N,L);
    for col = 1:L
        for row = 1:N
            temp_gen_col_total(col,1) = temp_gen_col_total(col,1) +
belief_generator(row, col);
        end
    end

    for row = 1:N
        for col = 1:L
            arr(row,col)  = belief_generator(row,col) ./
temp_gen_col_total(col,1);
        end
    end
end
```

```
function arr =
get_rule_weights(train_input,input_no,no_of_attributes,no_of_ref_val,
ref_vals)
    % Input Transformation
    transformed_input =
transform_input(train_input(input_no,:),no_of_attributes, no_of_ref_val,
ref_vals);

    % Rule Activation Weight Calculation
    matching_degrees = calc_matching_degrees(transformed_input,
no_of_attributes, no_of_ref_val); % Calculate Matching Degree
    combined_matching_degree =
calc_combined_matching_degrees(matching_degrees,no_of_ref_val^no_of_attribu
tes); % Calculate Combined Matching Degree
    arr = (matching_degrees) ./ (combined_matching_degree); % Calculate
Activation Weight
end

function arr = transform_input (input,no_of_attr,no_of_ref_val,ref_vals)
    arr = zeros(no_of_attr,no_of_ref_val); % Initialize with row_number x
column_number dummy values
    % Calculate and Populate with original values
    for i = 1:no_of_attr
        for j = 1:(no_of_ref_val - 1)
            if (input(1,i)>= ref_vals(1,(j+1)) && input(1,i) <=
ref_vals(1,j))
               arr(i,(j+1)) = (ref_vals(1,j) - input(1,i))/(ref_vals(1,j) -
ref_vals(1,j+1));
               arr(i,j) = 1 - arr(i,(j+1));
            end
        end
    end
end

function arr = calc_matching_degrees(individual_matching_degree,
no_of_attributes, no_of_ref_val)
    no_of_matching_degree = no_of_ref_val^no_of_attributes;
    arr = ones(no_of_matching_degree, 1);
    counter = 0;
    for i = 1:no_of_ref_val
        for j = 1:no_of_ref_val
            for k = 1:no_of_ref_val
                counter = counter + 1;
                arr(counter,1) = individual_matching_degree(1,i) *
individual_matching_degree(2,j) * individual_matching_degree(3,k);
            end
        end
    end
end

function val = calc_combined_matching_degrees(matching_degrees,
no_of_rules)
    val = 0;
    for i = 1:no_of_rules
       val = val + matching_degrees(i,1);
    end
end
```

```
function arr = calc_aggregated_belief_degree(activation_weight,
belief_degree, no_of_ref_val, no_of_rules)
    arr = zeros(no_of_ref_val,1);

    partA = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partB = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules);
    partC = calc_Part_C(activation_weight, no_of_rules);

    combined_partA = 0;
    for i = 1:no_of_ref_val
        combined_partA = combined_partA + partA(i,1);
    end

    for j = 1:no_of_ref_val
        arr(j,1) = (partA(j,1) - partB)/((combined_partA - ((no_of_ref_val
- 1) * partB)) - partC);
    end
end

function arr = calc_Part_A(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    arr = ones(no_of_ref_val,1);
    for i = 1:no_of_ref_val
        for j = 1:no_of_rules
            part1 = activation_weight(j,1) * belief_degree(i,j);

            temp = 0;
            for k = 1:no_of_ref_val
                temp = temp + belief_degree(k,j);
            end
            part2 = (1 - (activation_weight(j,1)*temp));

            temp_val = part1 + part2;
            arr(i,1) = arr(i,1) * temp_val;
        end
    end
end

function val = calc_Part_B(activation_weight, belief_degree, no_of_ref_val,
no_of_rules)
    val = 1;
    for i = 1:no_of_rules

        temp_total_belief = 0;

        for j = 1:no_of_ref_val
            temp_total_belief = temp_total_belief + belief_degree(j,i);
        end

        temp = activation_weight(i,1) * temp_total_belief;
        val = val * (1 - temp);
    end
end

function val = calc_Part_C(activation_weight, no_of_rules)
    val = 1;
    for i = 1:no_of_rules
```

```
        val = val * (1 - activation_weight(i,1));
    end
end

function val = calculateY(agg_bel_val, ref_vals,no_ref_val)
    val = 0;
    for i = 1: no_ref_val
        val = val + (agg_bel_val(i,1)*ref_vals(1,i));
    end
end

function arr = structure_belief_degrees(solution, N, L)
    temp = zeros(L, N);
    for i = 1:L
        temp(i, 1:N) = solution((((i-1)*N) + 1) : (((i-1)*N) + N));
    end
    arr = temp;
end
```