RSM

1. Główna funkcja realizująca algorytm

```
function [result, rank] = RSM(U, A0, A1, A2, A3, min_max)
    [P_idx, P] = pareto(U, min_max);
    decisions = size(P, 1);     % Number of decisions

    % Checking the integrity of reference points sets
    if ~check_integrity(A0, min_max)
        disp("No integrity in A0 set")
        result = -404;
        return
    end
    if ~check_integrity(A1, min_max)
        disp("No integrity in A1 set")
        result = -404;
        return
    end
    if ~check_integrity(A2, min_max)
        disp("No integrity in A2 set")
        result = -404;
        return
    end
    if ~check_integrity(A3, min_max)
        disp("No integrity in A3 set")
        result = -404;
        return
    end

    distance0 = zeros(decisions, size(A0, 1));
    distance1 = zeros(decisions, size(A1, 1));
    distance2 = zeros(decisions, size(A2, 1));
    distance3 = zeros(decisions, size(A3, 1));

    for i = 1:decisions
        for j = 1:size(A0, 1)
            distance0(i, j) = norm(P(i, :) - A0(j, :));
        end
        for j = 1:size(A1, 1)
            distance1(i, j) = norm(P(i, :) - A1(j, :));
        end
        for j = 1:size(A2, 1)
            distance2(i, j) = norm(P(i, :) - A2(j, :));
        end
        for j = 1:size(A3, 1)
            distance3(i, j) = norm(P(i, :) - A3(j, :));
        end
    end
```

```
    scoring_fcn = scoring_function(distance0, distance1, decisions);
    rank = sortrows([P, scoring_fcn], 4);
    result = rank(1, 1:size(rank, 2)-1);

    figure;
    plot3(A0(:, 1), A0(:, 2), A0(:, 3), "bo-");          % A0
    hold on;
    grid on;
    plot3(A1(:, 1), A1(:, 2), A1(:, 3), "go-");          % A1
    plot3(A2(:, 1), A2(:, 2), A2(:, 3), "co-");          % A2
    plot3(A3(:, 1), A3(:, 2), A3(:, 3), "ro-");          % A3
    plot3(U(:, 1), U(:, 2), U(:, 3), "yo");              % U
    plot3(P(:, 1), P(:, 2), P(:, 3), "mo");             % Pareto
    plot3(result(:,1), result(:, 2), result(:, 3), "k*")      % result
    xlabel("X");
    ylabel("Y");
    zlabel("Z");
    legend("A0", "A1", "A2", "A3", "U", "PU", "Best value")

end
```

Na samym początku ze zbioru alternatyw wyznaczany jest zbiór pareto. Następnie sprawdzana jest spójność zbiorów A0-A3, w przypadku gdy któryś z nich nie spełni warunku spójności, funkcja zwróci błąd.

Kolejnym etapem jest wyznaczenie odległości punktów ze zbioru pareto do punktów ze zbiorów A0-A3. Następnie macierze odległości przekazywane są do funkcji scoringowej, która przypisuje każdemu z punktów jakąś wartość, która następnie jest używana do utworzenia rankingu punktów.

Funkcja RSM zwraca najbardziej optymalną alternatywę oraz ranking alternatyw ułożonych od tych najbardziej opłacalnych do tych najmniej opłacalnych.

2. **Wyznaczanie zbioru pareto.**

```
function [pareto_idx, pareto_set] = pareto(U, min_max)
    % min_max is a array of booleans - true means min, false means max
    % Determining pareto set
    U_size = size(U, 1);
    pareto_set = [];
    pareto_idx = [];

    for i = 1:U_size
        is_dominated = false;
        for j = 1:U_size
            if i ~= j
                control_sum = 0;
                for col = 1:size(U, 2);
                    if min_max(col)
                        if U(i, col) >= U(j, col)
                            % Checking if point is dominated
                            control_sum = control_sum + 1;
                        end
                    else
                        if U(i, col) <= U(j, col)
                            % Checking if point is dominated
                            control_sum = control_sum + 1;
                        end
                    end
                end


                if control_sum == size(U, 2)
                    is_dominated = true;
                    break;
                end
            end
        end


        if ~is_dominated
            pareto_set = [pareto_set; U(i, :)];
            pareto_idx = [pareto_idx i];
        end
    end
end
```

Funkcja ta służy do wyznaczenia zbioru pareto. Szukamy w niej pośród zbioru U zdominowanych alternatyw i je odrzucamy.

Problemem, z którym trzeba było się zmierzyć, było uwzględnienie tego, że niektóre kryteria przy procesie wyboru chcemy minimalizować, a inne maksymalizować. W tym celu jako dodatkowy parametr do funkcji należy podać wektor *min_max,* który określa które z kryteriów mają być minimalizowane, a które przeciwnie. Jest to wektor zmiennych logicznych - dla przykładu, jeśli rozważalibyśmy zbiór składający się z trzech kryteriów, pierwsze dwa z nich miałyby być minimalizowane, a ostatnie maksymalizowane, to wektor *min_max* dla tego przypadku wyglądałby następująco: *[1 1 0].*

3. Sprawdzanie spójności

```
function [integrity] = check_integrity(class, min_max)
    % The set that is integral must not contain dominated points
    class_size = size(class, 1);
    integrity = true;

    for i = 1:class_size
        for j = 1:class_size
            if i ~= j
                control_sum = 0;
                for col = 1:size(class, 2);
                    if min_max(col)
                        if class(i, col) >= class(j, col)
                            % Checking if point is dominated
                            control_sum = control_sum + 1;
                        end
                    else
                        if class(i, col) <= class(j, col)
                            % Checking if point is dominated
                            control_sum = control_sum + 1;
                        end
                    end
                end

                if control_sum == size(class, 2)
                    integrity = false;
                    return
                end
            end
        end
    end
end
```

Funkcja działa na takiej samej zasadzie jak ta do wyznaczania zbioru pareto. Badamy w niej, czy w zbiorze *class* występują alternatywy zdominowane. Jeśli tak, to znaczy, że zbiór jest niespójny i funkcja zwraca wartość false, jeśli nie, to zbiór jest spójny i funkcja zwraca true.

4. **Funkcja scoringowa**

```
function [scoring_result] = scoring_function(distance0, distance1, decisions)
    scoring_result = zeros(decisions, 1);

    for i = 1:decisions
        mean_A0 = mean(distance0(i, :));
        mean_A1 = mean(distance1(i, :));
        scoring_result(i, 1) = mean([mean_A0, mean_A1]);
    end
end
```

Funkcja scoringowa służy do określenia „korzystności” danej decyzji. Wykorzystujemy w niej odległości punktów ze zbioru pareto do punktów ze zbioru A0 i A1. W tej konkretnej implementacji punkty ze zbioru A0 i A1 są traktowane równorzędnie, a wartość funkcji scoringowej to po prostu średnia z uśrednionych odległości punktów pareto do zbiorów A0 i A1.