Cojocariu Sebastian
Group 407

Project 2

Exercise 1 – SLD (Forward/Backward Chaining)

The Knowledge Base

Rules:

- If the student has graduated first in his class and the student is olympic (at some subject), then he gets a trip to Europe.
- If the student has passed the Baccalaureate and he has the highest grades in his class, then he has graduated first in his class.
- If the student gets at least 5 in each of the subjects Math, Romanian and Informatics and an overall of at least 6, then he has passed the Baccalaureate.

Questions:

- Which grade did you get at Math? (number from 1 to 10)
- Which grade did you get Romanian? (number from 1 to 10)
- Which grade did you get at Informatics? (number from 1 to 10)
- Do you have the highest grades among your collegues? (yes or no)
- Are you olympic at some subject? (yes or no)

Goal: Whether the student gets a trip to Europe or not.

Implementation details

For this task we made used of the Horn representation power (at most one positive atom, there rest, in exist, being negative ones) to compare two resolution strategies: **Backward-Chaining** and **Forward-Chaining**.

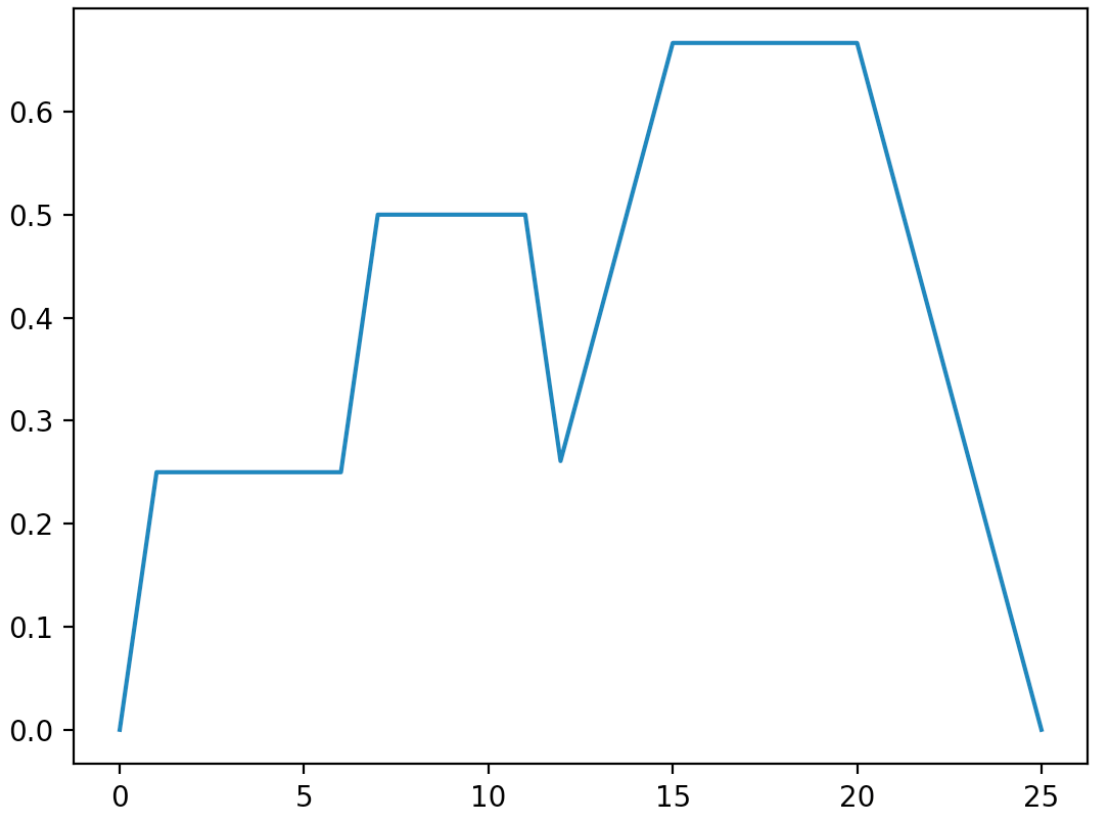
- ***remove_negations****: predicate that strips negation from a list of atoms.*
- ***backward_chaining****: implementation following the algorithm from C5, Slide 20.*
- ***forward_chaining****: implementation following the algorithm from C5, Slide 25.*
- ***read_file****: read the content of a file in a list.*
- ***read_KB****: read the KB from file.*
- ***ask_questions****: predicate that ask all the necessary questions to provide a useful prediction.*
- ***several other predicates*** *for use interaction: math_question, romanian_question, informatics_question, olympic_question, highest_grades_in_class_question*

Observation! The program starts by running the following command: *main().*

Cojocariu Sebastian
Group 407

Exercise 2 – Vagueness

Run on the course example. Service=3, Food=8, Tip = 14.58%



Story

Suppose someone wants to know his or her chances to enter a particular University (say it University of Bucharest) based on his/her grade at the entrance exam, the difficulty of the problems and the competition level.

- If own grade is bad or the competition level is high or subject difficulty is high, then the chances of entering are low.
- If own grade is good or the subject difficulty is moderate or the competition level is medium, then the chances of entering are medium.
- If own grade is very good or the competition level is low or the subject difficulty is low, then the chances of entering are high.

Rules as given in file:

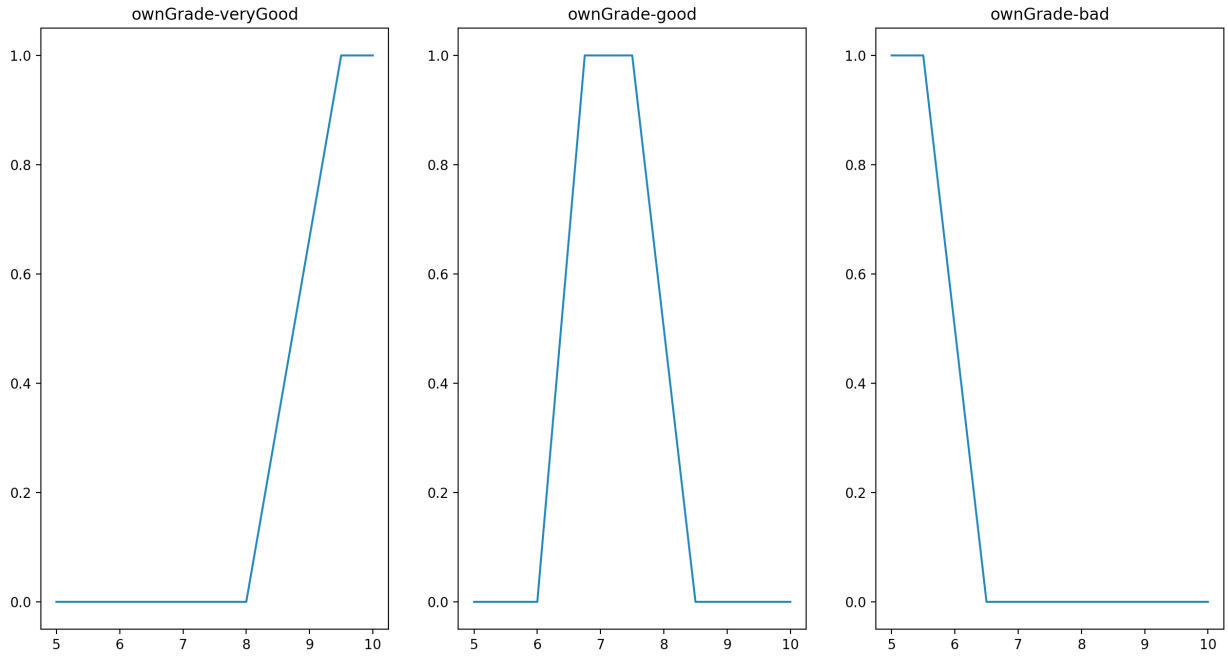
[or(ownGrade/bad, or(subjectDifficulty/high, competitionLevel/high)), chanceOfEntering/low].
[or(ownGrade/good, or(subjectDifficulty/moderate, competitionLevel/medium)), chanceOfEntering/medium].
[or(ownGrade/veryGood, or(subjectDifficulty/low, competitionLevel/low)), chanceOfEntering/high].

Observation! The program starts by running the following command: *main().*

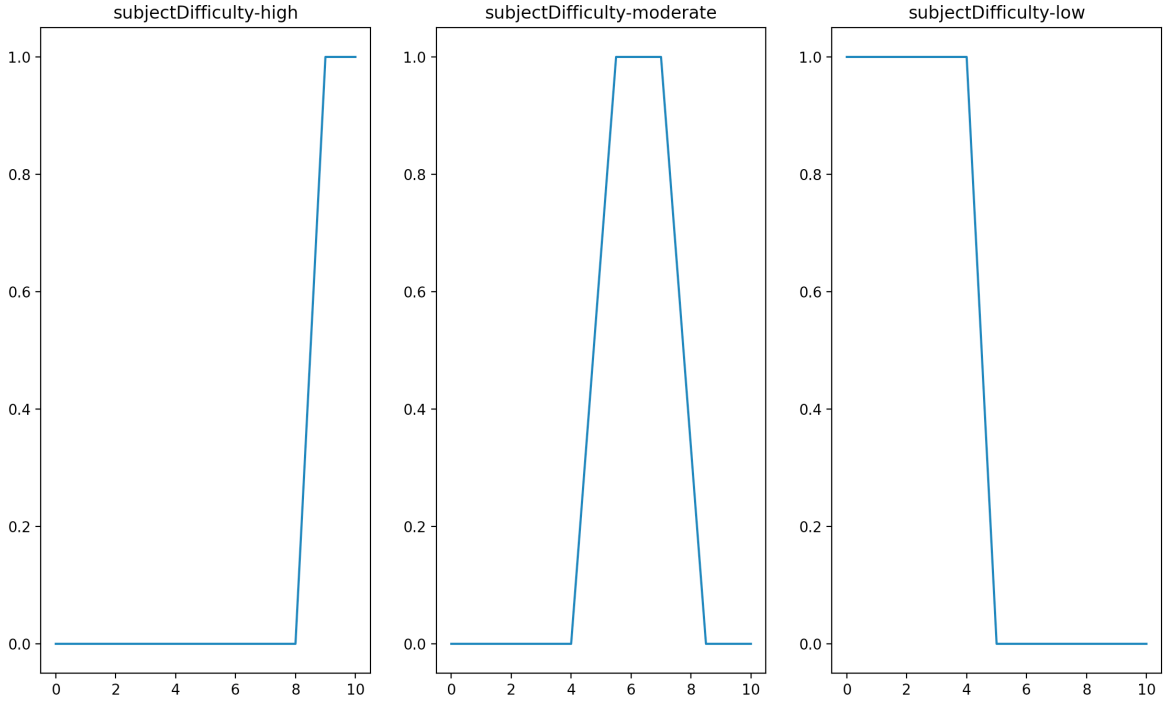
Cojocariu Sebastian
Group 407

Degree Curves

<u>OwnGrade</u> – on a scale form 5 to 10 (as 5 is the threshold for most admissions)

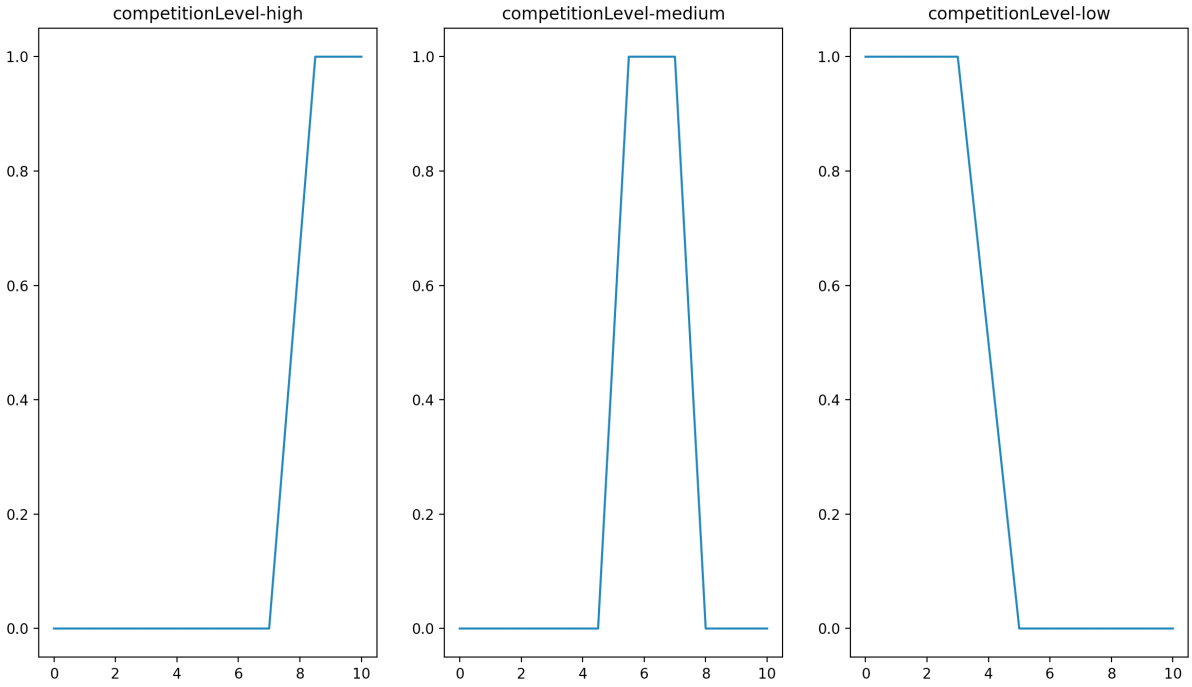


<u>Subject Difficulty</u> – on a scale from 0 to 10.

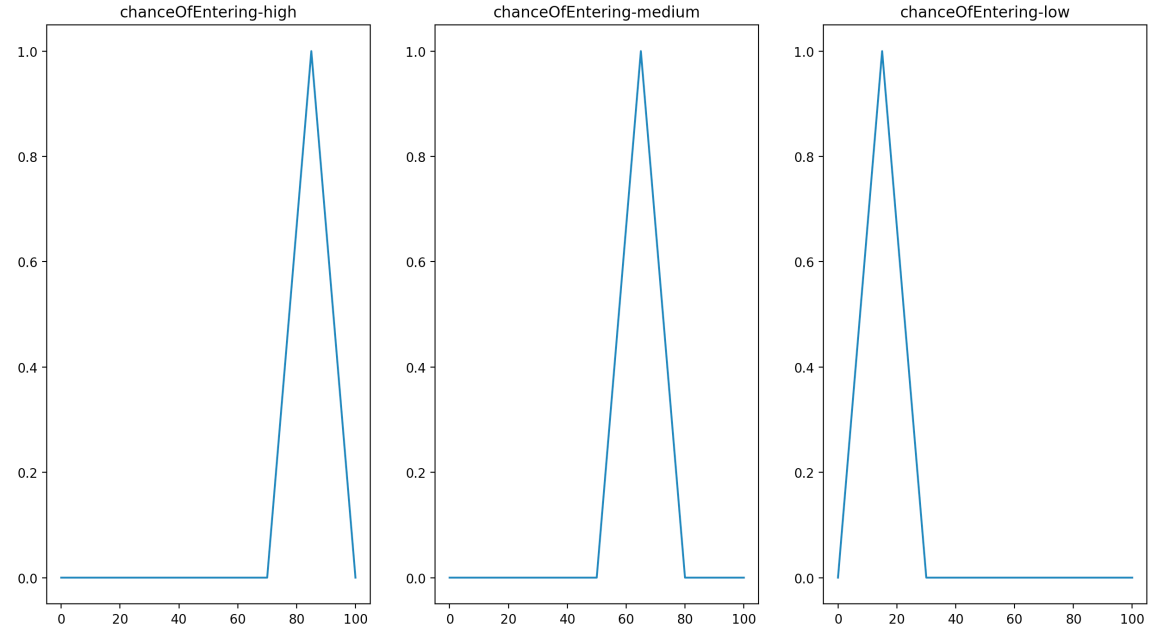


Cojocariu Sebastian
Group 407

Competition Level – on a scale from 0 to 10



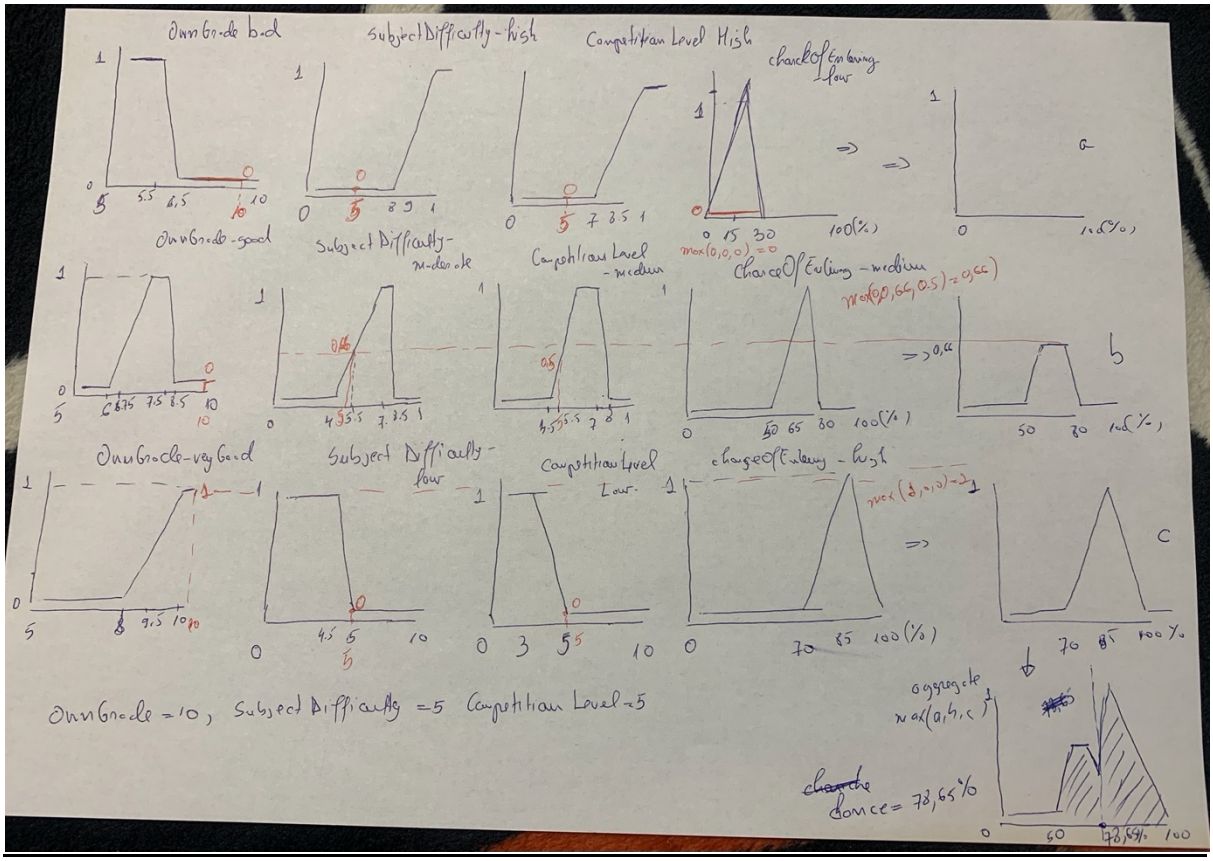
ChanceOfEntering: on a scale from 0 – 100 (%)

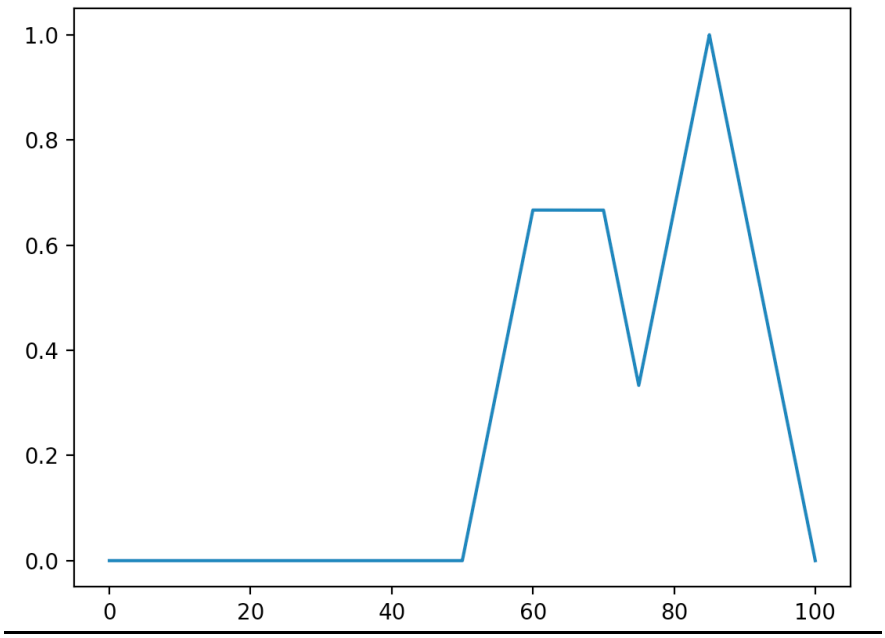


Cojocariu Sebastian
Group 407

Examples

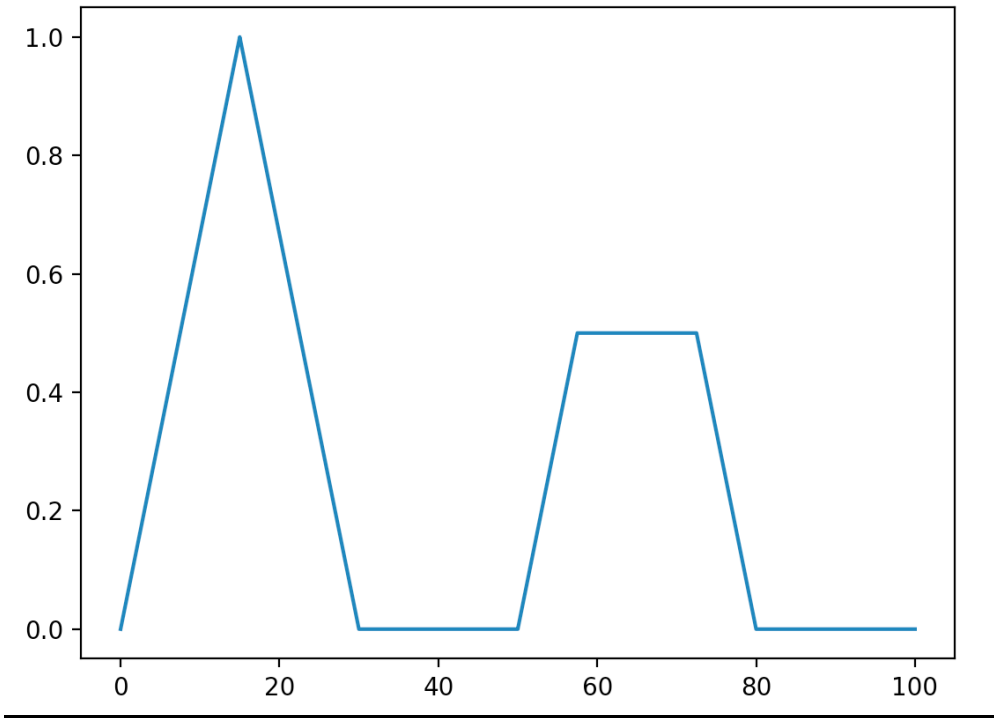
OwnGrade = 10, Competition Level = 5, Subject Difficulty = 5, Chances =78.65%



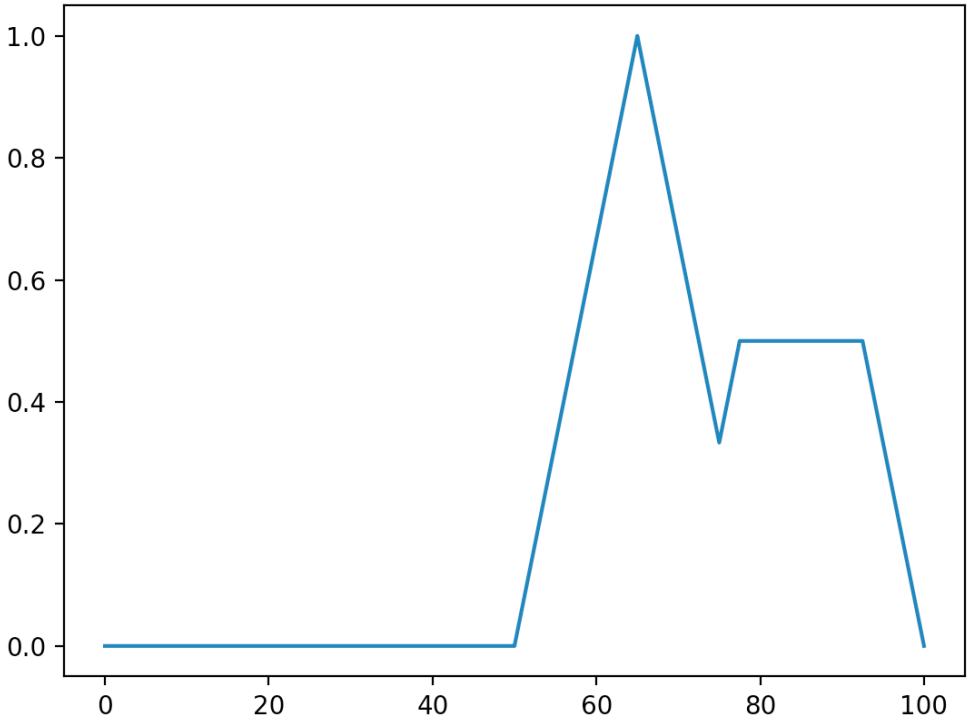


Cojocariu Sebastian
Group 407

OwnGrade = 8, Competition Level = 10, Subject Difficulty = 9, Chances = 17,01%

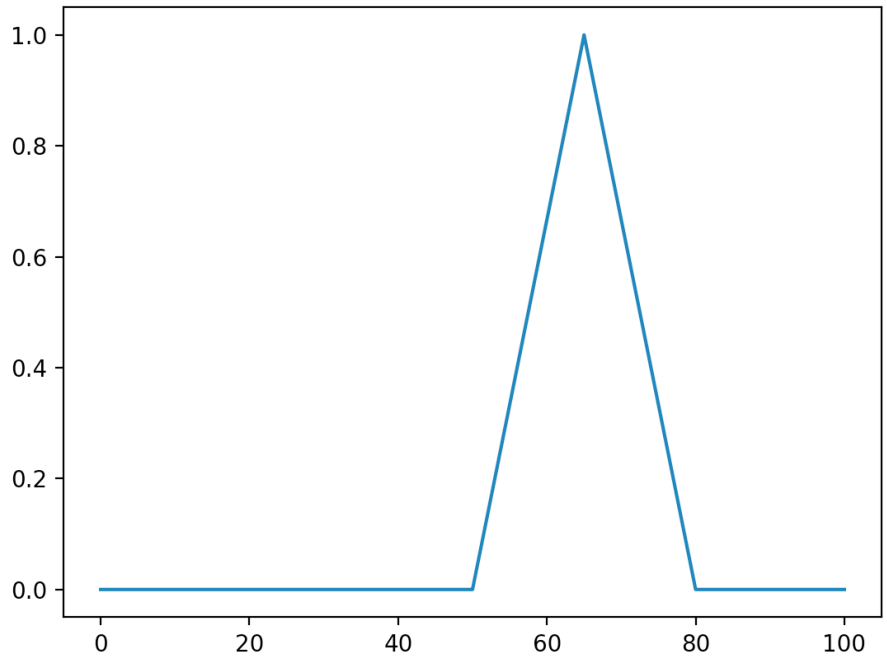


OwnGrade = 8, Competition Level = 4, Subject Difficulty = 6, Chances = 73,38%

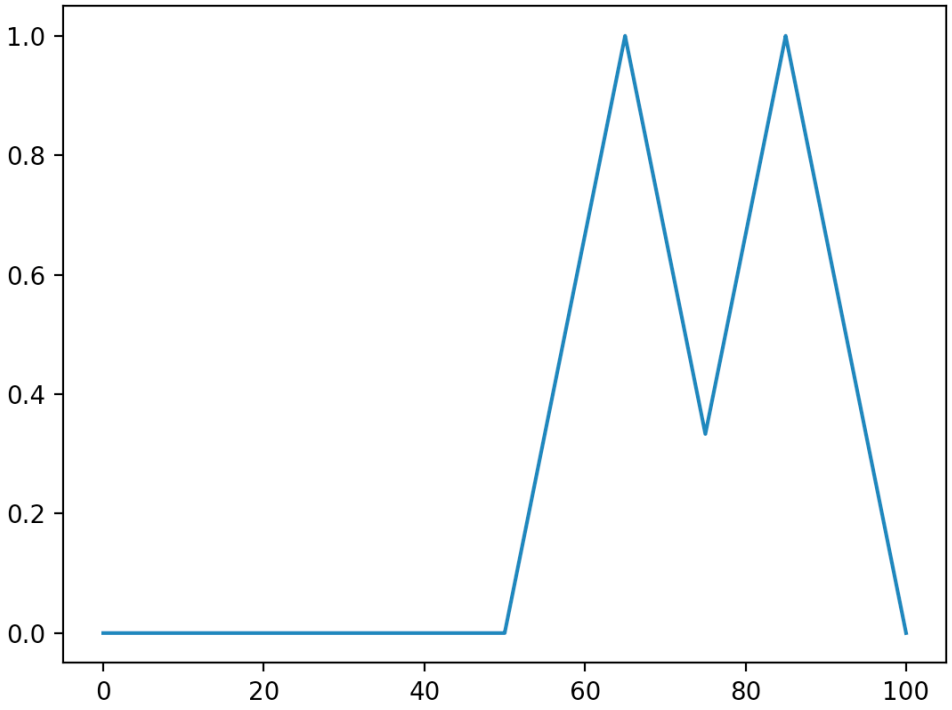


Cojocariu Sebastian
Group 407

OwnGrade = 7, Competition Level = 5, Subject Difficulty = 8, Chances = 64,99%



OwnGrade = 9.5, Competition Level = 4, Subject Difficulty = 6.5, Chances = 74,99%



Cojocariu Sebastian
Group 407

Implementation details

In order to implement a the requirements, I have implemented the following predicates:

- ***check_almost_equal****: used to asses a comparison between 2 numbers at some threshold (0.00001).*
- ***get_line_parameters****: given two points, it computes the line's parameters, namely m and n (y = mx + n).*
- ***construct_function_from_points****: construct a function (which is a collection of lines, from a set of points, by interpolation).*
- ***check_point_in_line_support:*** *check if a coordinate is in the support of a segment.*
- ***intersect_intervals****: intersection of two interval [A,B], [C,D].*
- ***get_line_intersection****: computes the intersection of 2 segments.*
- ***get_probability_per_function****: computes a function on same value.*
- ***trim_almost_same_lines****: removes the degenerated segments.*
- ***apply_clip_per_line****: limit the line to some y-threshold. It splits the segment if needed.*
- ***apply_clip_per_function****: limit the function to some y-threshold.*
- ***get_line_intersection_with_function****: get the interestion of a line/segment with a function (collection of segments).*
- ***get_intersection_points_of_two_functions****: get the intersection points of two functions.*
- ***lines_defined_in_given_interval****: collect all the lines/segments that are part of a given interval.*
- ***apply_input_on_lines_params****: apply an input to each lines from a list of lines.*
- ***select_best_lines****: select the best lines/segments for multiple intervals.*
- ***aggregate****: aggregate multiple functions based on max/min approach.*
- ***getArea_per_line****: computes area under a segment.*
- ***getAreaPerFunction****: computes are of a function.*
- ***getPartialAreas****: computes partial sum of areas (starting from the end).*
- ***solveQuadraticEquation****: solve a quadratic equation (or a line equation if first coefficient is 0).*
- ***bisectLineAtThresholdArea****: tries to find the x-coordinate from the segment support that generates 2 trapeziums, one having the area equal to Threshold.*
- ***defuzzify****: find the median point of a function (that splits the function into two equal parts).*
- ***apply_func****: convert a boolean expression (and – min, or – max, not – 1-p).*
- ***read_file****: read the content of a file.*
- ***read_degree_curves****: read degree curves from file.*
- ***convert_degree_curves****: create degree curves from points.*
- ***grade_question***
- ***competition_question,***
- ***difficulty_question,***
- ***main:*** *all the logic.*

Cojocariu Sebastian
Group 407

Code

Exercise 1

```
check_list_contained(List1, List2):- intersection(List1, List2, Common),
Common=List1.

remove_negations([], []):-!.
remove_negations([n(P)|Q], [P|R]):- remove_negations(Q, R), !.


backward_chaining([], KB, 'yes'):-!.
backward_chaining([Positive_atom|Goals], KB, R):- member(Clause, KB),

member(Positive_atom, Clause),

not(n(_)=Positive_atom),

delete(Clause, Positive_atom, Clause_without_positive_atom),

remove_negations(Clause_without_positive_atom, Clause_without_negations),

append(Clause_without_negations, Goals, Concatenate),

backward_chaining(Concatenate, KB, R), !.
backward_chaining(Senteces, KB, 'no'):- !.


forward_chaining_helper(Goals, KB, Solved, 'yes'):-
check_list_contained(Goals, Solved), !.
forward_chaining_helper(Goals, KB, Solved, R):- member(Clause, KB),

      member(Positive_atom, Clause),

not(Positive_atom=n(_)),

not(member(Positive_atom, Solved)),

delete(Clause, Positive_atom, Clause_without_positive_atom),

remove_negations(Clause_without_positive_atom, Clause_without_negations),

check_list_contained(Clause_without_negations, Solved),

forward_chaining_helper(Goals, KB, [Positive_atom|Solved], R), !.
forward_chaining_helper(Senteces, KB, Solved, 'no'):- !.
forward_chaining(Sentences, KB, R):- forward_chaining_helper(Sentences, KB,
[], R).


read_file(Stream,[]) :- at_end_of_stream(Stream).
read_file(Stream,[L|R]) :- not(at_end_of_stream(Stream)), read(Stream, L),
read_file(Stream, R).

read_KB(File_content):- open('ex1_own_kb.txt', read, Stream),
read_file(Stream, File_content), close(Stream), !.

main():- read_KB(File),
```

```
            [Rules|_]=File,
            repeat,
            ask_questions(Goals),
            append(Rules, Goals, KB),
            forward_chaining([trip], KB, Response_forward), writef("The
output for Forward Chaining of whether you are going on a trip is: %w \n",
[Response_forward]),
            backward_chaining([trip], KB, Response_backward), writef("The
output for Backward Chaining of whether you are going on a trip is:
%w\n\n", [Response_backward]),
            writeln('Should we continue? Please type stop to end or any
other combination to continue!'),
            read(Stop_response), nl,
            (Stop_response = stop ->
                writef('You typed %w. Have a nice day! \n\n',
[Stop_response]), !;
                writef('You typed %w. Start again\n\n', [Stop_response]),
fail
                ).



ask_questions([[Q1],[Q2],[Q3],[Q4],[Q5],[Q6]]):- math_question(Q1, G1),

romanian_question(Q2, G2),

informatics_question(Q3, G3),

pass_overall_condition(G1, G2, G3, Q4),

highest_grades_in_class_question(Q5),

olympic_question(Q6).


at_least_5_condition(Grade, Subject_predicate, Subject_predicate):- Grade
>= 5, !.
at_least_5_condition(Grade, Subject_predicate, n(Subject_predicate)):- !.

pass_overall_condition(Math_grade, Romanian_grade, Informatics_grade,
pass_overall):- sum_list([Math_grade, Romanian_grade, Informatics_grade],
Total), Total >= 18, !.
pass_overall_condition(_, _, _, n(pass_overall)):-!.

math_question(Predicate, Grade):-
      repeat,
      writeln('Which grade did you get at Math? (number from 1 to 10)'),
      read(Grade), nl,
      (number(Grade), Grade >= 1, 10 >= Grade ->
            writef('Your response to the last question is %w\n\n',
[Grade]),
            at_least_5_condition(Grade, pass_math, Predicate), !;
            writeln('The input should be a number. Please try again.'),
fail).


romanian_question(Predicate, Grade):-
      repeat,
      writeln('Which grade did you get Romanian? (number from 1 to 10)'),
      read(Grade), nl,
      (number(Grade), Grade >= 1, 10 >= Grade->
```

```
            writef('Your response to the last question is %w\n\n',
[Grade]),
            at_least_5_condition(Grade, pass_romanian, Predicate), !;
            writeln('The input should be a number. Please try again.'),
fail).

informatics_question(Predicate, Grade):-
      repeat,
      writeln('Which grade did you get at Informatics? (number from 1 to
10)'),
      read(Grade), nl,
      (number(Grade), Grade >= 1, 10 >= Grade ->
            writef('Your response to the last question is %w\n\n',
[Grade]),
            at_least_5_condition(Grade, pass_informatics, Predicate), !;
            writeln('The input should be a number. Please try again.'),
fail).



highest_grades_condition(yes, highest_grades):-!.
highest_grades_condition(no, n(highest_grades)):-!.

highest_grades_in_class_question(Predicate):-
      repeat,
      writeln('Do you have the highest grades among your collegues? (yes or
no)'),
      read(Ans), nl,
      (member(Ans, [yes, no]) ->
            writef('Your response to the last question is %w\n\n', [Ans]),
            highest_grades_condition(Ans, Predicate), !;
            writeln('The input should be yes or no. Please try again.'),
fail).


olympic_condition(yes, olympic):-!.
olympic_condition(no, n(olympic)):-!.

olympic_question(Predicate):-
      repeat,
      writeln('Are you olympic at some subject? (yes or no)'),
      read(Ans), nl,
      (member(Ans, [yes, no]) ->
            writef('Your response to the last question is %w\n\n', [Ans]),
            olympic_condition(Ans, Predicate), !;
            writeln('The input should be yes or no. Please try again.'),
fail).
```

Cojocariu Sebastian
Group 407

Exercise 2

```
my_round(Input, No_decimals, Final_result) :- Result is Input *
10^No_decimals,

round(Result, Result_new),

Final_result is Result_new / (10^No_decimals).

check_almost_equal(X, Y):-abs(X - Y, R), 0.00001 >= R, !.
check_almost_equal(_, _):-fail, !.


get_line_parameters([[X1, Y1], [X2, Y2]], _, _):- check_almost_equal(X1,
X2), !.
get_line_parameters([[X1, Y1], [X2, Y2]], [[X1, X2], [M, N]]):- A_aux is
(Y2 - Y1), my_round(A_aux, 5, A),

                      B_aux is (X1 - X2), my_round(B_aux, 5, B),

                      C_aux is A*X1 + B*Y1, my_round(C_aux, 5, C),

                    M_aux is -A/B, my_round(M_aux, 5, M),

                    N_aux is C/B, my_round(N_aux, 5, N).

construct_function_from_points([], []):-!.
construct_function_from_points([_], []):-!.
construct_function_from_points([Point1, Point2| Points], [Line|Result]):-
get_line_parameters([Point1, Point2], Line),


construct_function_from_points([Point2| Points], Result),

                                    !.

parallel_lines(Line1, Line2):- [[_, _], [M1, N1]]=Line1, [[_, _], [M2,
N2]]=Line2, check_almost_equal(M1, M2), not(check_almost_equal(N1, N2)), !.
parallel_lines(Line1, Line2):- fail, !.

check_point_in_line_support(X, [[X1, X2], [_, _]]):- X >= X1, X2 >= X, !.
check_point_in_line_support(X, [[X1, X2], [_, _]]):- check_almost_equal(X,
X1), !.
check_point_in_line_support(X, [[X1, X2], [_, _]]):- check_almost_equal(X,
X2), !.
check_point_in_line_support(X, _):- fail, !.

intersect_intervals([X1, X2], [X3, X4], []):- X1 > X4 + 0.00001, !.
intersect_intervals([X1, X2], [X3, X4], []):- X3 > X2 + 0.00001, !.
intersect_intervals([_, X2], [X3, _], [Min_value, Max_value]):-
check_almost_equal(X2, X3), min_list([X2, X3], Min_value), max_list([X2,
X3], Max_value), !.
intersect_intervals([X1, _], [_, X4], [Min_value, Max_value]):-
check_almost_equal(X1, X4), min_list([X1, X4], Min_value), max_list([X1,
X4], Max_value), !.
intersect_intervals([X1, X2], [X3, X4], [Left, Right]):- max_list([X1, X3],
Left), min_list([X2, X4], Right),!.
```

Cojocariu Sebastian
Group 407

```
get_line_intersection(Line1, Line2, []):- [[X1, X2], _]=Line1,
not(check_point_in_line_support(X1, Line2)),
not(check_point_in_line_support(X2, Line2)), !.
get_line_intersection(Line1, Line2, []):- parallel_lines(Line1, Line2), !.
get_line_intersection(Line1, Line2, Intersection):- [Support1, [M1,
N1]]=Line1,

      [Support2, [M2, N2]]=Line2,

      check_almost_equal(M1, M2),

      check_almost_equal(N1, N2),

      intersect_intervals(Support1, Support2, Intersection),

      !.

get_line_intersection(Line1, Line2, [X_intersection, X_intersection]):- [_,
[M1, N1]]=Line1,

                                    [_, [M2, N2]]=Line2,

                                    X_intersection is (N2 - N1) / (M1 -
M2),


      check_point_in_line_support(X_intersection, Line1),


      check_point_in_line_support(X_intersection, Line2),

                                    !.
get_line_intersection(Line1, Line2, []):-!.


get_probability_per_function(Input, [], _):- fail, !.
get_probability_per_function(Input, [Line|Set_of_lines], Result):-
check_point_in_line_support(Input, Line),

                            [[_, _], [M, N]]=Line,

                            Result is M * Input + N,

                            !.
get_probability_per_function(Input, [Line|Set_of_lines], Result):-
get_probability_per_function(Input, Set_of_lines, Result), !.


trim_almost_same_lines([], []):-!.
trim_almost_same_lines([Line|Set_of_lines], Result):- [[X1, X2], [M,
N]]=Line, check_almost_equal(X1, X2), trim_almost_same_lines(Set_of_lines,
Result), !.
trim_almost_same_lines([Line|Set_of_lines], [Line|Result]):-
trim_almost_same_lines(Set_of_lines, Result), !.

apply_clip_per_line(Threshold, Line, [[[X1, X2], [0.0, Min_value]]]):-
[[X1, X2], [M, N]]=Line, check_almost_equal(M, 0.0), min_list([N,
Threshold], Min_value), !.
apply_clip_per_line(Threshold, Line, Final_result):- [[X1, X2], [M,
N]]=Line,
```

```
       X_intersection is (Threshold - N) / M,

       check_point_in_line_support(X_intersection, Line),

       (M > 0 ->

            Result = [[[X1, X_intersection], [M, N]], [[X_intersection,
X2],[0.0, Threshold]]];

            Result = [[[X1, X_intersection], [0.0, Threshold]],
[[X_intersection, X2],[M, N]]]

        ),

       trim_almost_same_lines(Result, Final_result),

       !.
apply_clip_per_line(Threshold, Line, Final_result):- [[X1, X2], [M,
N]]=Line,

       Y1 is M * X1 + N,

       Y2 is M * X2 + N,

       Y1 >= Threshold,

       Y2 >= Threshold,

       Result=[[[X1, X2], [0.0, Threshold]]],

       trim_almost_same_lines(Result, Final_result), !.
apply_clip_per_line(Threshold, Line, [Line]):-!.


apply_clip_per_function(Threshold, [], []):-!.
apply_clip_per_function(Threshold, [Line| Set_of_lines], Final_result):-
apply_clip_per_line(Threshold, Line, Clipped_line),

                                    apply_clip_per_function(Threshold,
Set_of_lines, Result),

                                    append(Clipped_line, Result,
Final_result).


get_line_intersection_with_function(Line, [], []):-!.
get_line_intersection_with_function(Line, [Curr_line|Set_of_lines],
Final_result):- get_line_intersection(Line, Curr_line, Intersection),


      get_line_intersection_with_function(Line, Set_of_lines, Result),

                                                    [[X1, X2], _] =
Curr_line,

                                                    [[X3, X4], _] = Line,

                                                    append(Result,
[X1,X2,X3,X4|Intersection], Intermediate),
```

Cojocariu Sebastian
Group 407

```
                                                        sort(Intermediate,
Sorted),

                                                        list_to_set(Sorted,
Final_result).
get_intersection_points_of_two_functions([], _, []):-!.
get_intersection_points_of_two_functions([Line|LinesFirstFunc],
LinesSecondFunc, Final_result):- get_line_intersection_with_function(Line,
LinesSecondFunc, Result1),


get_intersection_points_of_two_functions(LinesFirstFunc, LinesSecondFunc,
Result2),


append(Result1, Result2, Intermediate),


sort(Intermediate, Sorted),


list_to_set(Sorted, Final_result),

                                                                        !.
line_defined_in_given_interval(Interval, Line):- [X1, X2]=Interval,

[[Left, Right], [M,N]]=Line,
                                                                          X1
>= Left,

Right >= X2,
                                                                          !.

line_defined_in_given_interval(Interval, Line):-fail, !.

lines_defined_in_given_interval(Interval, [], []):-!.
lines_defined_in_given_interval(Interval, [Line|Lines], [Params|Result]):-
line_defined_in_given_interval(Interval, Line),


lines_defined_in_given_interval(Interval, Lines, Result),

                                        [Support, Params]=Line,

                                        !.
lines_defined_in_given_interval(Interval, [Line|Lines], Result):-
lines_defined_in_given_interval(Interval, Lines, Result).

group_by_intervals([], Lines, []):-!.
group_by_intervals([_], Lines, []):-!.
group_by_intervals([X1, X2| Cut_points], Lines, Final_result):-
check_almost_equal(X1, X2),

                        group_by_intervals([X2|Cut_points], Lines,
Final_result),

                        !.
```

Cojocariu Sebastian
Group 407

```
group_by_intervals([X1, X2| Cut_points], Lines, Final_result):-
lines_defined_in_given_interval([X1, X2], Lines, Result1),

                        group_by_intervals([X2|Cut_points], Lines,
Result2),

                        Final_result=[[[X1, X2], Result1] | Result2],

                        !.


apply_input_on_lines_params(Input, [], []):- !.
apply_input_on_lines_params(Input, [[M, N]|Set_of_lines_params],
[Value|Result]):- Value is M * Input + N,
apply_input_on_lines_params(Input, Set_of_lines_params, Result), !.

select_best_line_from_unintersected_lines(Type, [Interval, Lines_params],
[Interval, BestFunction]):- [X1, X2] = Interval,


Middle_of_interval is (X1 + X2) / 2,


apply_input_on_lines_params(Middle_of_interval, Lines_params, Values),


(Type = max ->


          max_list(Values, Value);


          min_list(Values, Value)),


nth0(Index, Values, Value),


nth0(Index, Lines_params, BestFunction), !.

select_best_lines(Type, [], []):-!.
select_best_lines(Type, [Current_interval_info| Intervals_info], Result):-
select_best_line_from_unintersected_lines(Type, Current_interval_info,
Result1),

                                    select_best_lines(Type,
Intervals_info, Result2),

                                    Result=[Result1| Result2].


flatten_one_level([], []):-!.
flatten_one_level([H|T], Final_result) :- is_list(H),

flatten_one_level(T, Result),
                                                        append(H,
Result, Final_result).

aggregate(Type, Functions, ChoosenLines):- flatten_one_level(Functions,
All_Lines),
```

Cojocariu Sebastian
Group 407

```
get_intersection_points_of_two_functions(All_Lines, All_Lines, Cut_points),

group_by_intervals(Cut_points, All_Lines, Grouped_lines_by_intervals),

select_best_lines(Type, Grouped_lines_by_intervals,
ChoosenLines_with_possible_duplicates),

trim_almost_same_lines(ChoosenLines_with_possible_duplicates,
ChoosenLines).

getArea_per_line(Line, Area):- [[X1, X2], [M, N]]=Line,
                                            Y1 is M * X1 + N,
                                            Y2 is M * X2 + N,
                                            Area is (Y1 + Y2) / 2 * (X2
- X1).

getAreaPerFunction([], 0):-!.
getAreaPerFunction([Line|Set_of_lines], Total_area):-
getArea_per_line(Line, Local_area),

getAreaPerFunction(Set_of_lines, Result),

Total_area is Result + Local_area, !.

getPartialAreas([], 0, []):-!.
getPartialAreas([Line|Set_of_lines], Partial_area, [Partial_area|List]):-
getPartialAreas(Set_of_lines, Current_Area, List),

                                    getArea_per_line(Line, Local_area),

                                    Partial_area is Current_Area +
Local_area,

                                    !.


solveQuadraticEquation(A, B, C, Result):- check_almost_equal(A, 0.0),
                                                        X is -C/B,
                                                        Result=[X],
                                                        !.

solveQuadraticEquation(A, B, C, _):- D is (B^2 - 4 * A * C), -0.00001 > D,
fail, !.
solveQuadraticEquation(A, B, C, Result):- D is (B^2 - 4 * A * C),

(check_almost_equal(D, 0.0) ->
                                                                    X
is B/(2 * A),

      Result=[X];

      sqrt(D, Squared),
                                                                    X1
is (-B + Squared)/2/A,
                                                                    X2
is (-B - Squared)/2/A,

      Result = [X1, X2]
                                                                    ),
                                                        !.
```

Cojocariu Sebastian
Group 407

```
trim_solutions([], Line, _):-fail, !.
trim_solutions([X|Solutions], Line, X):- check_point_in_line_support(X,
Line), !.
trim_solutions([X|Solutions], Line, Result):- trim_solutions(Solutions,
Line, Result), !.

bisectLineAtThresholdArea(Line, Threshold, Solution):- [[X1, X2], [M,
N]]=Line,

            Y2 is M * X2 + N,

            A is M,

            B is Y2 + N - M * X2,

            C is 2 * Threshold - X2 * Y2 - X2 * N,

            solveQuadraticEquation(A, B, C, Solutions),

            trim_solutions(Solutions, Line, Solution).


helper_to_find_index(Partial_areas, HalfArea, Index):- findall(Position,
(member(X, Partial_areas), X >= HalfArea - 0.00001, nth0(Position,
Partial_areas, X)), Indexes),

         max_list(Indexes, Index),

         !.


defuzzify(Partial_areas, HalfArea, Function, Solution):-
helper_to_find_index(Partial_areas, HalfArea, Index),

             nth0(Index, Partial_areas, Current_Area),

             nth0(Index, Function, LineToBisect),

             Remaining_area is Current_Area - HalfArea,

             (check_almost_equal(Remaining_area, 0.0) ->

                  [[Solution,_],_]=LineToBisect;

                  bisectLineAtThresholdArea(LineToBisect, Remaining_area,
Solution)

             ).



apply_func(Expression, Scores_from_user, Degrees_curves, Final_result):-
n(Expr)=Expression, apply_func(Expr, Scores_from_user, Degrees_curves,
Result), Final_result is 1 - Result, !.
apply_func(Expression, Scores_from_user, Degrees_curves, Final_result):-
and(Expr1, Expr2)=Expression,

                                    apply_func(Expr1, Scores_from_user,
Degrees_curves, Result1),
```

```
                                    apply_func(Expr2, Scores_from_user,
Degrees_curves, Result2),

                                    min_list([Result1, Result2],
Final_result),

                                    !.
apply_func(Expression, Scores_from_user, Degrees_curves, Final_result):-
or(Expr1, Expr2)=Expression,

                                   apply_func(Expr1, Scores_from_user,
Degrees_curves, Result1),

                                   apply_func(Expr2, Scores_from_user,
Degrees_curves, Result2),

                                   max_list([Result1, Result2],
Final_result),

                                   !.
apply_func(BaseFunction/Vague, Scores_from_user, Degrees_curves,
Final_result):- member([BaseFunction/Vague, Function], Degrees_curves),

                                                 member([BaseFunction,
Score], Scores_from_user),


get_probability_per_function(Score, Function, Final_result),

                                                 !.

evaluate_consequent([Expression, Consequent], Scores_from_user,
Degrees_curves, Evaluated_consequent):- apply_func(Expression,
Scores_from_user, Degrees_curves, Threshold),


            member([Consequent, Function_consequent], Degrees_curves),


          apply_clip_per_function(Threshold, Function_consequent,
Evaluated_consequent),


          !.

evaluate_list_of_consequents([], Scores_from_user, Degrees_curves, []):-!.
evaluate_list_of_consequents([H|T], Scores_from_user, Degrees_curves,
[Evaluated_consequent|Result]):- evaluate_consequent(H, Scores_from_user,
Degrees_curves, Evaluated_consequent),


         evaluate_list_of_consequents(T, Scores_from_user, Degrees_curves,
Result),


         !.


read_file(Stream,[]) :- at_end_of_stream(Stream).
```

Cojocariu Sebastian
Group 407

```
read_file(Stream,[L|R]) :- not(at_end_of_stream(Stream)), read(Stream, L),
read_file(Stream, R).


convert_degree_curves([], []):-!.
convert_degree_curves([[X/Y, Points] | Degrees_curves], [[X/Y,
Function]|Result]):- convert_degree_curves(Degrees_curves, Result),


      construct_function_from_points(Points, Function),

                                                    !.

read_degree_curves(Degrees_curves):- open('degrees_own.txt', read, Stream),
                                                     read_file(Stream,
File_content),
                                                     close(Stream),

convert_degree_curves(File_content, Degrees_curves),
                                                     !.

read_rules(Rules):-open('rules_own.txt', read, Stream),
                           read_file(Stream, Rules),
                           close(Stream),
                           !.

grade_question(Score):-
                                     repeat,
                                     writeln('Please enter your grade on a
scale from 5 to 10? (answer must be a number)'),
                                     read(Score), nl,
                                     (number(Score), Score >= 5, 10 >= Score
->
                                           writef('Your response to the last
question is %w\n\n', [Score]), !;
                                           writeln('The input should be a
number between 5 and 10. Please try again.'), fail).


competition_question(Score):-
                                repeat,
                                writeln('Please enter the competition level
on a scale from 0 to 10? (answer must be a number)'),
                                read(Score), nl,
                                (number(Score), Score >= 0, 10 >= Score ->
                                      writef('Your response to the last
question is %w\n\n', [Score]), !;
                                      writeln('The input should be a number
between 0 and 10. Please try again.'), fail).

difficulty_question(Score):-
                                repeat,
                                writeln('Please enter the difficulty of the
subjects on a scale from 0 to 10? (answer must be a number)'),
                                read(Score), nl,
                                (number(Score), Score >= 0, 10 >= Score ->
                                      writef('Your response to the last
question is %w\n\n', [Score]), !;
                                      writeln('The input should be a number
between 0 and 10. Please try again.'), fail).
```

Cojocariu Sebastian
Group 407

```
read_questions([[ownGrade, Grade_score], [competitionLevel,
Competition_Score], [subjectDifficulty, Difficulty_score]]):-
grade_question(Grade_score),



competition_question(Competition_Score),



difficulty_question(Difficulty_score),


                                        !.


main():-
             read_rules(Rules),
             read_degree_curves(Degrees_curves),
             repeat,
           read_questions(Scores_from_user),
             evaluate_list_of_consequents(Rules, Scores_from_user,
Degrees_curves, Consequents),
             aggregate(max, Consequents, Aggregated_consequents),
             getPartialAreas(Aggregated_consequents, _, Partial_areas),
             getAreaPerFunction(Aggregated_consequents, Total_area),
             writeFunctionToFile(Aggregated_consequents),
             HalfArea is Total_area / 2,
             defuzzify(Partial_areas, HalfArea, Aggregated_consequents,
Recommendation),
             writef("The chance of entering is: %w\n\n", [Recommendation]),
             writeln('Should we continue with another prediction? Please
type stop to end or any other combination to continue!'),
             read(Stop_response), nl,
             (Stop_response = stop ->
                  writef('You typed %w. Have a nice day! \n\n',
[Stop_response]), !;
                  writef('You typed %w. Start again\n\n', [Stop_response]),
fail
             ).

writeFunctionToFile(Function):- open('aggregate.txt',write, Out),
                                                write(Out, Function),
                                                close(Out).
```