

Uniwersytet Jagielloński
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

MODELOWANIE ROZMYTE

ŚRODOWISKO PROGRAMISTYCZNE

Grzegorz Głowaty

Praca magisterska napisana pod kierunkiem
dr Andrzeja Łachwy

Kraków, maj 2003

SPIS TREŚCI

Wstęp.....	3
I. Teoretyczne podstawy modelowania opartego na logice rozmytej.....	4
1. Reguły sterownika rozmytego.....	4
2. Proces wnioskowania.....	6
2.1. Blok rozmywania.....	6
2.2. Blok wnioskowania.....	7
2.3. Blok wyostrzania.....	8
3. Przykładowe wnioskowanie oparte na logice rozmytej.....	9
II. Modelowanie rozmyte.....	12
1. Modelowanie systemu z pomocą eksperta.....	12
2. Automatyczna generacja modelu rozmytego.....	13
2.1. Idea zastosowania klasteryzacji do ekstrakcji bazy reguł sterownika rozmytego.....	14
2.2. Klasteryzacja rozmyta.....	16
2.2.1. Algorytm c-środków (fuzzy c-means, FCM).....	17
2.2.2. Algorytm Gustafsona-Kessela (GK) – wersja zmodyfikowana.....	20
2.3. Przetwarzanie wyników klasteryzacji.....	23
III. Szczegóły implementacji oraz stworzone narzędzia.....	27
1. Szczegóły implementacji klas sterownika.....	28
2. Język opisu sterownika.....	31
2.1. Składnia języka FML (Fuzzy Modelling Language).....	32
2.1.1. Definicja typu dokumentu FML (DTD).....	37
2.2. Proces translacji FML na C++.....	38
3. Programy narzędziowe.....	39
3.1. Program ClusterIt.....	39
3.2. Program fuzzyShape.....	41
3.3. Program fml2cpp.....	42
IV. Eksperymenty i wnioski.....	43
1. Klasteryzacja i modelowanie funkcji $f(x,y)=\sin(x)*\cos(y)$	43
2. Problem balansowania kuli na pochylni (ang. Ball and Beam).....	46
3. Podsumowanie.....	50
Bibliografia:.....	51

WSTĘP

Zbiory rozmyte, po raz pierwszy zaproponowane przez Zadeha w 1965 r. mają za zadanie matematyczną reprezentację niejednoznaczności i niepewności.

Logika rozmyta, której zadaniem jest naśladowanie „ludzkiego” sposobu wnioskowania stanowi podstawę teorii zbiorów rozmytych i rozmytego modelowania.

Systemy rozmyte znajdują zastosowanie głównie tam, gdzie nie posiadamy dostatecznej wiedzy o dokładnym matematycznym modelu tworzonego systemu lub istnieje potrzeba podejmowania decyzji w okolicznościach niepewnej lub niekompletnej informacji o otoczeniu. Przybliżenie systemu łatwo wyrażalnymi regułami lingwistycznymi jest w wielu przypadkach dużo łatwiejsze (i co zaskakujące – wystarczające) niż odtworzenie dokładnych wzorów matematycznych nim rządzących.

W ostatnich latach teoria zbiorów rozmytych zaczęła cieszyć się coraz większą popularnością, szczególnie w zakresie jej zastosowań w bazach danych, sterowaniu oraz w dziedzinach zajmujących się przetwarzaniem języka naturalnego.

Najwięcej praktycznych zastosowań znajduje sterownie rozmyte. Produkowane są obecnie mikroprocesory, których zestawy rozkazów optymalizowane są pod względem zastosowania ich jako rozmytych sterowników.

W niniejszej pracy przedstawiłem podstawy dotyczące modelowania rozmytego. Skupiłem się na metodach pozyskiwania wiedzy opartych na klasteryzacji rozmytej. Zilustrowane zostały problemy wiążące się z opisanymi metodami gromadzenia wiedzy. Przedstawiona została propozycja interfejsu programistycznego wspomagającego tworzenie aplikacji wykorzystujących logikę rozmytą oraz próba zaproponowania przeze mnie opartego na XML języka opisu sterowników rozmytych. Niejako na marginesie tworzenia tej pracy powstał szereg opisanych w rozdziale III narzędzi ułatwiających automatyczne pozyskiwanie wiedzy oraz programowanie sterowników rozmytych. Aby zilustrować opisane w pracy metody przeprowadzone zostały eksperymenty zastosowania sterowników rozmytych w konkretnych zadaniach. Próby te zilustrowane zostały w rozdziale IV niniejszej pracy.

I. TEORETYCZNE PODSTAWY MODELOWANIA OPARTEGO NA LOGICE ROZMYTEJ.

Jedną z dziedzin, w których doskonale zastosowanie znajduje logika rozmyta jest sterowanie. Sterownik rozmyty (ang. fuzzy controller) jest modelem matematycznym opisanym za pomocą zmiennych lingwistycznych oraz reguł operujących na tych zmiennych. Z matematycznego punktu widzenia sterownik rozmyty aproksymuje pewną funkcję realizowaną przez rzeczywisty system.

Działanie sterownika rozmytego (wspomniana aproksymacja) odbywa się w kilku opisanych poniżej fazach. Każdy z tych etapów może być realizowany według jednego z wielu z wielu schematów teoretycznych. Sama konstrukcja sterownika rozmytego jest przedmiotem szerokich dyskusji i trwają ciągle próby znalezienia modelu najdoskonalszego. Wymienić można tutaj chociażby model Takagi-Sugeno czy nieco starszy model Mamdaniego. W niniejszej pracy skupię się na realizacji modelu Mamdaniego, z regułami wzbogaconymi o wagi (model relacyjny).

1. REGUŁY STEROWNIKA ROZMYTEGO.

Reguły opisujące sterownik rozmyty są regułami typu IF ... THEN

Ich poprzedniki składają się z serii predykatów połączonych spójnikami logicznymi, a następni to predykaty definiujące konkluzję.

Predykaty te są postaci

$A \text{ is } B$, gdzie:

A – nazwa pewnej zmiennej lingwistycznej,

B – jeden z termów definiujących zmienną A .

W modelu Takagi-Sugeno, w odróżnieniu do opisywanego tutaj modelu Mamdaniego konkluzje reguł mają postać $A \text{ is } f(x_1, x_2, x_3, \dots, x_n)$, gdzie f jest pewną funkcją wartości ostrych kolejnych zmiennych wejściowych.

W teorii zbiorów rozmytych terminy określające zmienną lingwistyczną mają postać zbiorów rozmytych. Na zmienną lingwistyczną możemy zatem spojrzeć jak na zbiór nazwanych zbiorów rozmytych określających pewne pojęcie z języka naturalnego. Na przykład zmienna lingwistyczna „temperatura wody” może zostać scharakteryzowana przez trzy terminy: „mała”, „średnia” oraz „wysoka”, którym zostaną przypisane odpowiednie zbiory rozmyte opisujące kolejno małą, średnią oraz wysoką temperaturę wody.

Zmienne lingwistyczne pojawiające się po lewej stronie reguł rozmytych nazywać będziemy zmiennymi wejściowymi sterownika. Zmienne z konkluzji reguł to zmienne wyjściowe.

Jako podstawę wnioskowania rozmytego stosuje się uogólnioną na teorię zbiorów rozmytych regułę modus ponens. Tak jak klasyczna reguła modus ponens, tak i uogólniona jej wersja, pozwala na podstawie prawdziwości poprzednika implikacji wnioskować o prawdziwości jej następnika. Zasadniczą różnicą jest pojęcie stopnia prawdziwości, czy też stopnia spełnienia, zarówno poprzednika jak i następnika implikacji jako pewnego predykatu. Sama implikacja jest traktowana tutaj jako dwuargumentowa relacja rozmyta, dzięki czemu projektant sterownika rozmytego ma możliwość doboru odpowiedniej funkcji realizującej rozmytą implikację w zależności od potrzeb systemu. W projektowanym systemie zastosowałem dwa rodzaje operatora implikacji rozmytej. Implikację Mamdaniego oraz implikację wykorzystującą algebraiczny operator PROD [Łachwa 01], [Piegat 99].

Implikacja Mamdaniego ma postać:

$$\mu_{A \rightarrow B}(x, y) = \min(\mu_A(x), \mu_B(y)) \quad .$$

Implikacja typu PROD z kolei:

$$\mu_{A \rightarrow B}(x, y) = \mu_A(x) \mu_B(y)$$

Wynikiem działania reguły rozmytej typu

IF x is A THEN y is B

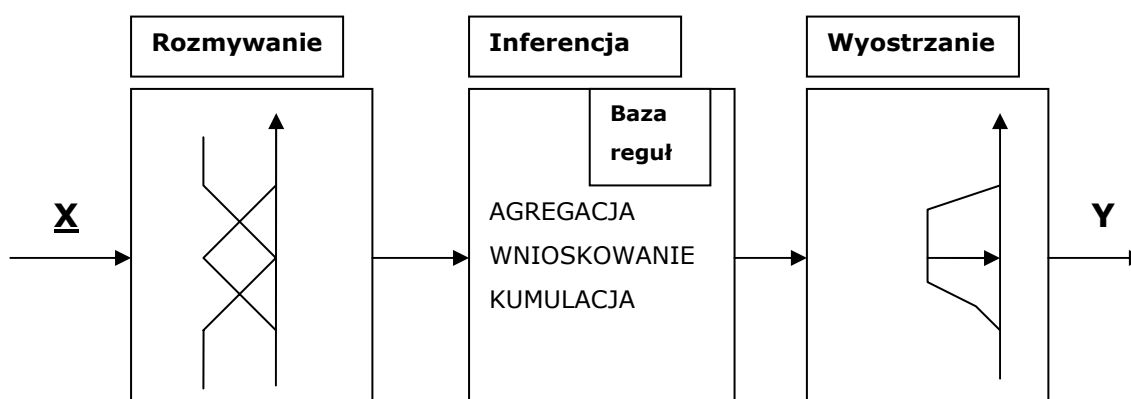
używającej pewnego ustalonego operatora implikacji dla pewnej ustalonej wartości ostrej $x = x_0$ jest zbiór rozmyty będący przekrojem zbioru definiującego relację

$\mu_{A \rightarrow B}(x, y)$ wykonanym dla x_0 . Mówiąc dokładnie jest to zbiór o funkcji przynależności $\mu(y) = \mu_{A \rightarrow B}(x_0, y)$.

Przedstawiony wyżej model wnioskowania rozmytego jest podstawą działania sterowników rozmytych Mamdaniego. Dokładna procedura funkcjonowania takiego sterownika została opisana poniżej.

2. PROCES WNIOSKOWANIA.

W sterownikach rozmytych możemy wyróżnić trzy, odgrywające najważniejszą rolę w procesie wnioskowania, bloki przedstawione na poniższym schemacie. [IEC 97]



Rys. I-1 Schemat działania sterownika rozmytego.

2.1. Blok rozmywania.

Zadaniem bloku rozmywania jest przekształcenie otrzymanej na wejściu wartości liczbowej skojarzonej z daną zmienną lingwistyczną na stopień spełnienia predykatów w poprzednikach reguł angażujących tą właśnie zmienną. Najczęściej stosowaną w praktyce metodą rozmywania jest metoda typu singleton, w której wartości ostrej x skojarzonej ze zmienną lingwistyczną A podanej na wejściu przypisuje się zbiór rozmyty X o stopniu przynależności 1 dla x , a 0 dla wszystkich innych elementów dziedziny, a jako wartość spełnienia predykatu logicznego A *is* B uznaje się zbiór powstały w wyniku przecięcia (z użyciem wybranej t-normy) zbioru X ze zbiorem skojarzonym z termem B .

Metoda singleton została przeze mnie wybrana z uwagi na niski nakład obliczeniowy potrzebny do dokonania rozmycia oraz późniejszego wnioskowania oraz na fakt, iż metoda ta sprawdza się w praktyce. Istnieje wiele innych, bardziej skomplikowanych, metod rozmywania, które zostały opisane m. in. w [Łachwa 01] czy [Piegat 97].

2.2. Blok wnioskowania.

Wartości zmiennych wejściowych zamienione zostały na stopnie spełnienia odpowiednich predykatów w przesłankach reguł. W bloku wnioskowania ma miejsce uruchomienie każdej reguły, której przesłanki są spełnione, wyliczenie zbioru rozmytego będącego wynikiem jej działania oraz kumulacja wyników w obrębie bloku reguł.

W procesie **agregacji** stopień spełnienia każdej z reguł obliczany jest na podstawie stopnia spełnienia jej przesłanek. Wykorzystywane w tym celu są logiczne operacje rozmyte AND, OR oraz NOT. Jak wyjaśniono w rozdziale III punkt 2 w realizowanym przeze mnie modelu pod uwagę brane są jedynie reguły z przesłankami połączonymi spójnikiem AND.

Przyjęte przez mnie możliwości realizacji operatora AND (t-normy) to

- MIN: $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$,
- PROD: $\mu_{A \cap B}(x) = \mu_A(x)\mu_B(x)$,
- parametryzowana t-norma Hammachera:

$$\mu_{A \cap B}^{\gamma}(x) = \frac{\mu_A(x)\mu_B(x)}{\gamma + (1 - \gamma)[\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)]}, \gamma \geq 0.$$

Reguły, których stopień spełnienia uzyskany w procesie agregacji jest niezerowy są **aktywowane**, w wyniku czego zgodnie z przyjętym dla reguły modelem implikacji rozmytej oraz metodą wnioskowania opisaną wyżej obliczany jest zbiór rozmyty reprezentujący konkluzję danej reguły.

Możliwe do zastosowania w tworzonym przeze mnie systemie operatory implikacji rozmytej to MIN (operator Mamdaniego) oraz PROD (iloczyn algebraiczny).

Wynikowe zbiory rozmyte **kumulują się** w pojedynczy zbiór będący wynikiem wnioskowania, który następnie poddawany jest procesowi wyostrzania. Kumulacja polega na zastosowaniu odpowiedniej s-normy (operatora sumowania) do zbiorów

otrzymanych w kolejnych regułach. Możliwymi do użycia w tworzonym systemie operatorami kumulacji są MAX,BSUM oraz SUM.

Operatory te działają następująco:

- MAX: $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$,
 - BSUM: $\mu_{A \cup B}(x) = \min(1, \mu_A(x) + \mu_B(x))$,
 - SUM: $\mu_{A_1 \cup A_2 \cup \dots \cup A_n}(x_0) = f(x_0) / \max(f(x))$,
- gdzie $f(x) = \mu_{A_1}(x) + \mu_{A_2}(x) + \dots + \mu_{A_n}(x)$.

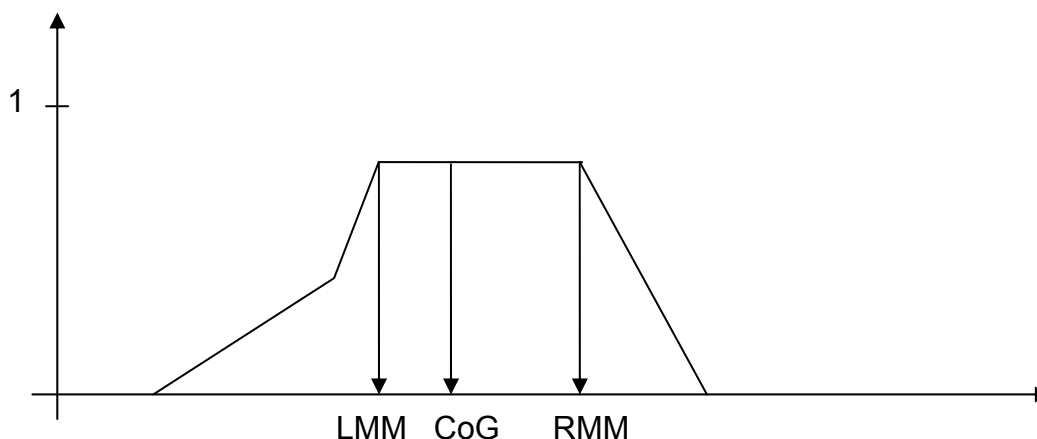
Jak widać zaproponowany przeze mnie operator SUM jest operatorem sumy nieograniczonej, która po wyliczeniu jest normalizowana tak, aby jej maksimum była liczba 1. Operator SUM nie jest s-normą (por. definicję s-normy [Łachwa 01] [Piegat 99]) ponieważ nie jest łączny, jednak stosowany globalnie – normalizowany dopiero na końcu sumowania wszystkich zbiorów, tak jak zapisano powyżej – sprawdza się w procesie kumulacji. Stosując ten operator można uzyskać metodę wyostrzani środka sum przedstawioną m.in. w [Piegat 99]. Wystarczy zrealizować kumulację operatorem SUM, a wyostrzanie operatorem środka ciężkości (patrz niżej).

2.3. Blok wyostrzania.

Wyostrzanie ma na celu przekształcenie wynikowego zbioru rozmytego na określoną wartość rzeczywistą stanowiącą wartość wyjścia modelu.

W tworzonym systemie możliwymi operatorami wyostrzania są pierwsze z lewej maksimum (LMM – Left Most Maximum), pierwsze z prawej maksimum (RMM – Right Most Maximum) oraz środek ciężkości (CoG -center of gravity).

Działanie powyższych operatorów pokazane zostało na rysunku:



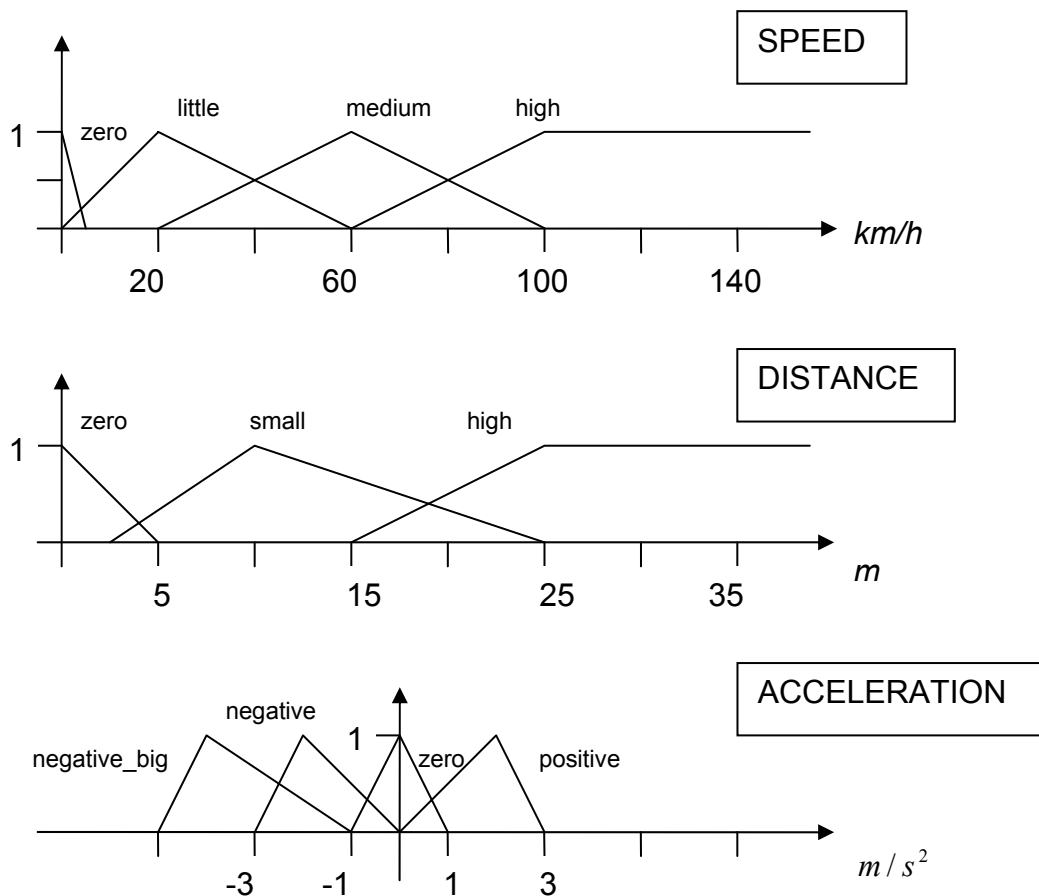
Rys. I-2 Różne metody wyostrzania.

W zależności od zastosowanej metody otrzymujemy następujące wyniki wyostrzania:

- LMM: $x_0 : \mu(x_0) = \max(\mu(x)) \wedge \forall x < x_0 \mu(x) < \mu(x_0)$
- RMM: $x_0 : \mu(x_0) = \max(\mu(x)) \wedge \forall x > x_0 \mu(x) > \mu(x_0)$
- CoG: $x_0 = \frac{\int x\mu(x)dx}{\int \mu(x)dx}$

3. PRZYKŁADOWE WNIOSKOWANIE OPARTE NA LOGICE ROZMYTEJ.

Założmy, że mamy dane dwie zmienne wejściowe charakteryzujące aktualną prędkość i odległość od ewentualnej przeszkody stojącej na drodze hipotetycznego samochodu (SPEED, DISTANCE) oraz jedna zmienna wyjściowa mówiąca o sugerowanym przyspieszeniu (opóźnieniu) tegoż samochodu (ACCELERATION) scharakteryzowane terminami jak na Rys. I-3.



Rys. I-3 Zaproponowane zmienne lingwistyczne.

Sterowanie przyspieszaniem i zwalnianiem samochodu zdefiniowane jest następującymi regułami:

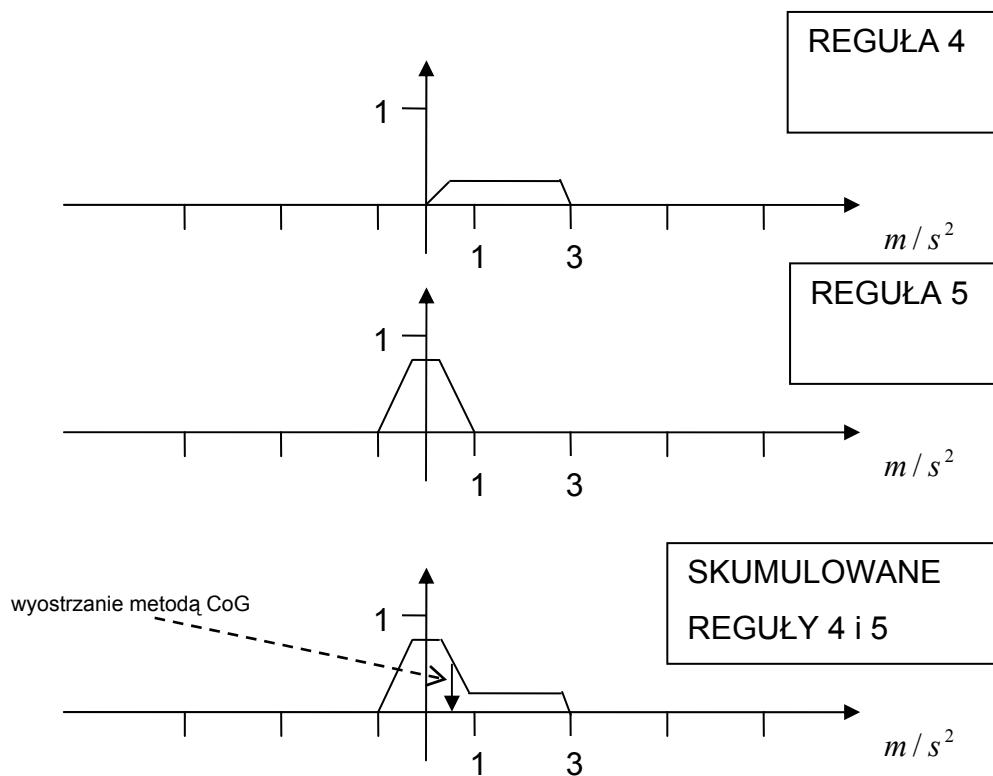
1. IF SPEED IS ZERO AND DISTANCE IS ZERO THEN ACCELERATION IS ZERO
2. IF SPEED IS ZERO AND DISTANCE IS SMALL THEN ACCELERATION IS POSITIVE
3. IF SPEED IS LITTLE AND DISTANCE IS ZERO THEN ACCELERATION IS NEGATIVE
4. IF SPEED IS LITTLE AND DISTANCE IS SMALL THEN ACCELERATION IS POSITIVE
5. IF SPEED IS MEDIUM AND DISTANCE IS ZERO THEN ACCELERATION IS NEGATIVE_BIG
6. IF SPEED IS MEDIUM AND DISTANCE IS SMALL THEN ACCELERATION IS ZERO
7. IF SPEED IS HIGH AND DISTANCE IS ZERO THEN ACCELERATION IS NEGATIVE_BIG
8. IF SPEED IS HIGH AND DISTANCE IS SMALL THEN ACCELERATION IS NEGATIVE
9. IF DISTANCE IS HIGH THEN ACCELERATION IS POSITIVE

Dyskusja poprawności działania sterownika w oparciu o powyższą bazę reguł nie jest przedmiotem tej pracy. Powyższe reguły zostały zaproponowane w celu ilustracji działania procesu wnioskowania.

Założmy, że zmierzone wartości prędkości oraz odległości to SPEED = 50, DISTANCE = 9. Agregację realizować będziemy przy pomocy t-normy MIN, implikacja zrealizowana zostanie przy pomocy operatora MIN, kumulacja przy pomocy operatora MAX, wyostrzanie przy pomocy operatora CoG.

W wyniku procesu agregacji przesłanek powyższych reguł dwie z nich (reguła 4 i reguła 5) otrzymają niezerowe stopnie spełnienia (równe odpowiednio 0,25 i 0,75).

Reguły 4 i 5 zostaną aktywowane. Wyniki ich aktywacji pokazane zostały na rysunku niżej.



Rys. I-4 Wynik aktywacji, agregacji i wyostrzania.

W procesie wyostrzania zbioru wynikowego przedstawionego na rysunku metodą środka ciężkości stosując opisany wyżej wzór otrzymano wynikowe przyspieszenie równe $0,652 \frac{m}{s^2}$. Wynik ten można zinterpretować następująco: podjęta została decyzja o powolnym zwiększaniu prędkości w związku z faktem iż aktualna prędkość jest średnia i jest pewien zapas odległości do poprzedzającego samochodu. Oczywiście sterownik ten działa dobrze pod warunkiem, iż przeszkoda (samochód poprzedzający) porusza się w tym samym kierunku co samochód sterowany z podobną prędkością. Właściwy sterownik powinien brać pod uwagę nie tylko prędkość sterowanego samochodu, lecz przede wszystkim prędkość zbliżania się do przeszkody. Jak jednak wcześniej wspomniano przykład ten ma na celu jedynie zilustrowanie mechanizmu wnioskowania.

II. MODELOWANIE ROZMYTE.

1. MODELOWANIE SYSTEMU Z POMOCĄ EKSPERTA.

Istnieje wiele sposobów tworzenia modeli rozmytych istniejących systemów. Najprostszym z nich jest stworzenie zestawu odpowiednich zmiennych lingwistycznych wraz z określającymi je termami oraz grupą reguł typu **IF...THEN...** przez eksperta posiadającego wiedzę o modelowanym systemie.

Podejście to napotyka jednak na pewne oczywiste problemy związane ze stopniem złożoności modelowanego procesu oraz charakterystyką wiedzy eksperta, która najczęściej jest bardziej „intuicyjna” niż ściśle określona. Mimo iż istotą modelowania rozmytego jest oparcie reguł inferencyjnych na takiej właśnie intuicyjnej wiedzy, samo wyspecyfikowanie parametrów poszczególnych termów lingwistycznych może być problemem, podczas, gdy ekspert doskonale poradzi sobie ze zdefiniowaniem relewantnej bazy reguł.

Nasuwa się tutaj następujące rozwiązanie powyższego problemu związanego z niedokładnym określeniem termów.

Wykorzystajmy wiedzę eksperta do zdefiniowania reguł rządzących zachowaniem systemu oraz do określenia zmiennych lingwistycznych i wstępnego kształtu odpowiednich zbiorów rozmytych. Następnie poddamy tak utworzony sterownik procesowi dostrajania, dzięki czemu możliwe będzie doprecyzowanie pozyskanej od eksperta wiedzy.

Proces dostrajania sterownika rozmytego realizowany może być w oparciu o algorytmy genetyczne lub przekształcenie sterownika w rozmytą sieć neuronową [Piegat 97]. Strojenie nie jest głównym tematem niniejszej pracy, wobec czego opiszę pokrótce jedną z możliwości jego realizacji przy pomocy algorytmów genetycznych. Uzyskaną wstępną wersję sterownika przekształcamy w jednego z członków populacji, na której będzie operował algorytm genetyczny. Jako parametry reprezentacji możemy wybrać kształt termów, rodzaj implikacji, rodzaje operatorów AND oraz OR a także wagi poszczególnych reguł w modelu relacyjnym. Następnie poprzez cykliczne powtarzanie operacji mutacji, krzyżowania oraz selekcji dostajemy coraz to nowe, bardziej przystosowane, wersje modelu. Ciekawym sposobem

dokonania selekcji jest testowanie modelu na szeregu losowo dobranych zadań i odrzucanie tych modeli, które radzą sobie najgorzej.

Dostrajanie sterownika odgrywa także rolę w przypadku opisanej poniżej automatycznej generacji sterownika, gdzie uzyskany wynik jest często dopiero wstępną wersją modelu systemu.

2. AUTOMATYCZNA GENERACJA MODELU ROZMYTEGO.

Automatyczna generacja modelu rozmytego to proces jego tworzenia na podstawie szeregu próbek pomiarowych uzyskanych na podstawie pomiaru działającego (modelowanego) systemu.

Zastosowanie w dziedzinie automatycznej generacji modelu rozmytego znajdują wszelkie metody optymalizacyjne. Od wspomnianych wcześniej algorytmów genetycznych i metody polegającej na przekształceniu sterownika rozmytego w rozmytą sieć neuronową aż do metod klasteryzacyjnych.

Metody klasteryzacyjne polegają na automatycznym wykrywaniu skupisk (klasterów) próbek pomiarowych w przestrzeni działania systemu oraz wyciąganiu na tej podstawie wniosków o modelu. W dalszej części przedstawię kilka interesujących z punktu widzenia modelowania rozmytego algorytmów klasteryzacyjnych wraz z przykładami ich działania.

Konstruując metody klasteryzacyjne musimy mieć na uwadze wymaganą od nich odporność na zaburzenia pomiarowe zgromadzonych próbek oraz fakt, iż próbki wejściowe mogą być rozłożone nierównomiernie w przestrzeni działania systemu. Jak zauważono w [Piegat 99], nierównomierne rozłożenie próbek pomiarowych w przestrzeni nie jest problemem z punktu widzenia praktyki, o ile mamy gwarancję iż w większości (o ile nie we wszystkich) przypadków modelowany system będzie znajdował się w stanach, które były (choć w przybliżeniu) określone przez zgromadzone próbki pomiarowe. Jeżeli modelowany system pracuje tylko w pewnych ściśle określonych obszarach przestrzeni wejść nie jest koniecznym wymaganiem odpowiedniego zachowania się modelu w całości tej przestrzeni.

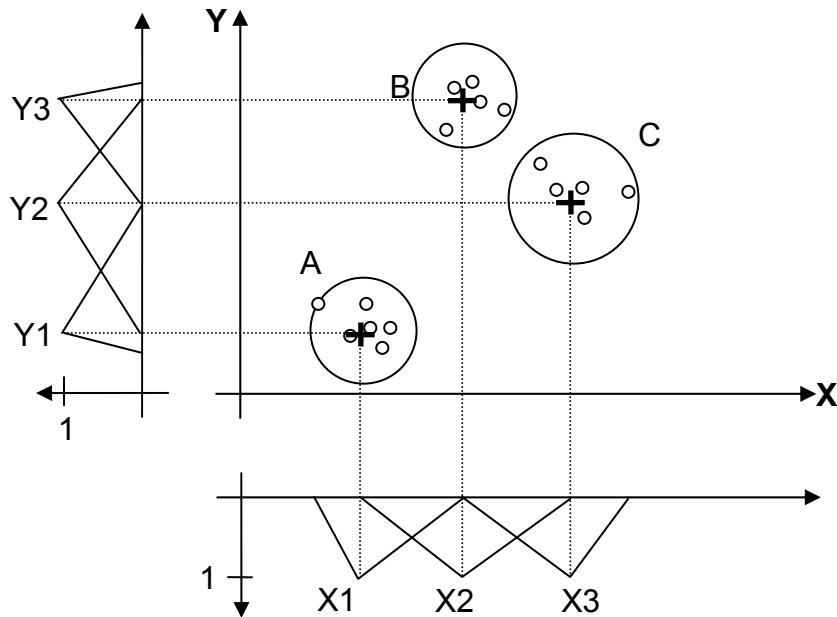
W niniejszej pracy szczególny nacisk położę na metody poszukiwania klastrów w przestrzeni $X_1 \times \dots \times X_n \times Y$, gdzie funkcja modelowanego systemu jest określona jako $f : X_1 \times \dots \times X_n \rightarrow Y$.

2.1. Idea zastosowania klasteryzacji do ekstrakcji bazy reguł sterownika rozmytego.

Podstawą stosowania klasteryzacji przy automatycznym tworzeniu sterownika jest obserwacja następującego faktu: dane pomiarowe obserwowanego systemu grupują się w charakterystycznych dla systemu punktach w obszary (klastery). Dla każdego z takich obszarów można w łatwy sposób ustalić determinującą wyjście w zakresie tego obszaru regułę.

Aby naświetlić metodologię stosowania klasteryzacji w pozyskiwaniu reguł przeanalizujemy następujący przykład wykorzystujący klasteryzację ostrą. Przestrzeń próbek zostanie podzielona na 3 klastery. Do klasteryzacji można wykorzystać dowolny algorytm grupowania nierozmytego (na przykład najbliższego sąsiedztwa). Rozważmy przykład przedstawiony na rysunku Rys. II-1. W przestrzeni rozważań $X \times Y$ wyodrębniono trzy zgrupowania próbek pomiarowych (klastery) zaznaczone okręgami.

Najprostszym sposobem wytworzenia zestawu reguł sterownika rozmytego jest skonstruowanie w przestrzeni X i Y zestawu zbiorów rozmytych o funkcjach przynależności jak na wykresach na Rys. II-1 ($X_1, X_2, X_3, Y_1, Y_2, Y_3$).



Rys. II-1. Przykładowe klasterzy z zaznaczonymi środkami w przypadku dwuwymiarowym.

Niech x_A, x_B, x_C będą odpowiednio środkami klasterów A, B, C .

Niech $MIN_X = \min_{i=1 \dots N}(x_{i,1})$, $MIN_Y = \min_{i=1 \dots N}(x_{i,2})$, $MAX_X = \max_{i=1 \dots N}(x_{i,1})$, $MAX_Y = \max_{i=1 \dots N}(x_{i,2})$.

Przykładowe funkcje przynależności zbiorów rozmytych $X1$ i $X2$ wyrażają się wzorami:

$$\mu_{X1}(x) = \begin{cases} \frac{x - MIN_X}{x_{A,1} - MIN_X}, & MIN_X \leq x < x_{A,1} \\ \frac{x_{B,1} - x}{x_{B,1} - x_{A,1}}, & x_{A,1} \leq x < x_{B,1} \end{cases}$$

$$\mu_{X2}(x) = \begin{cases} \frac{x - x_{B,1}}{x_{C,1} - x_{B,1}}, & x_{B,1} \leq x < x_{C,1} \\ \frac{x_{B,1} - x}{x_{B,1} - x_{A,1}}, & x_{A,1} \leq x < x_{B,1} \end{cases}$$

Możemy napisać reguły sterownika:

IF x is X1 THEN y is Y1

IF x is X2 THEN y is Y3

IF x is X3 THEN y is Y2

Przedstawione w powyższym przykładzie podejście do zagadnienia tworzenia sterownika jest zaskakująco proste i szybkie w implementacji (z dokładnością do zastosowanego algorytmu klasteryzacji), jednak, co za tym idzie, nie jest pozbawione wad.

Najłatwiej zauważyć, że arbitralne ustawienie funkcji przynależności wygenerowanych zbiorów rozmytych wynika między innymi z tego, iż przyporządkowanie punktów do klastrów jest jednoznaczne. Punkt albo należy, albo nie należy do klastra. Nie można uwzględnić potencjalnych stopni przynależności punktów do klastrów w procesie uzyskiwania zbiorów rozmytych, bo stopni tych nie uzyskano w procesie klasteryzacji. Tak utworzone zbiory rozmyte są nieco sztuczne i nie odzwierciedlają dokładnie systemu. Ich zaletą jednak jest fakt, iż są proste pod względem złożoności obliczeniowej przyszłego sterownika oraz, iż suma stopni przynależności poszczególnych punktów do wszystkich zbiorów wynosi 1.

W powyższych rozważaniach przedstawiono ideę pozyskiwania bazy reguł z danych oraz przykładowe problemy, które można napotkać podczas klasteryzacji. W kolejnych punktach przedstawione zostaną algorytmy klasteryzacji rozmytej, która pozwoli nam skonstruować klaster i zbiory rozmyte o bardziej adekwatnych kształtach, bardziej odzwierciedlające modelowany system.

2.2. Klasteryzacja rozmyta.

W procesie klasteryzacji rozmytej przestrzeni S na c klastrów każdemu z N poddawanych klasteryzacji jej elementów $x_i \in S, i = 1 \dots N$, przyporządkowuje się każdemu elementowi x_i stopień przynależności do każdego z c klastrów, taki, że:

$0 \leq \mu_i^{(k)} \leq 1, \forall i = 1 \dots N \quad \forall k = 1 \dots c$, gdzie $\mu_i^{(k)}$ oznacza stopień przynależności punktu x_i do klastra k .

Jak widać, klaster staje się zbiorem rozmytym określonym w przestrzeni S .

Stopień przynależności do klastra zależy zwykle od odległości punktu od uzyskanego w procesie klasteryzacji środka klastra.

Często postuluje się [np. Babuška 01], by były spełnione następujące warunki:

$$\sum_{k=1}^c \mu_j^{(k)} = 1 \quad \forall j = 1 \dots N \quad (2.1)$$

$$\forall k \exists i \mu_i^{(k)} > 0 \quad (2.2)$$

Warunki te oznaczają odpowiednio, iż łączny stopień przynależności każdej próbki do wszystkich klasterów jest całkowity (równy jeden) oraz, iż żaden z klasterów nie pozostanie po klasteryzacji pusty.

Najczęściej opisywanym w literaturze ze względu na stosunkowo prostą implementację i dobre wyniki algorytmem pozwalającym na zbudowanie klasterów spełniających powyższe warunki jest algorytm rozmytych c-środków (ang. Fuzzy c-means, FCM) przedstawiony w [Piegat 99].

2.2.1. Algorytm c-środków (fuzzy c-means, FCM).

Założmy, iż klasteryzacji poddajemy zbiór N punktów z przestrzeni R^n . W procesie klasteryzacji chcemy uzyskać c klasterów, oraz cN stopni przynależności każdego z punktów do każdego z klasterów.

Chcemy także, aby spełnione były warunki (2.1) oraz (2.2).

Istotą algorytmu c-środków jest przedstawienie problemu klasteryzacji, jako problemu minimalizacji wartości następującej funkcji:

$$f_X(U, M) = \sum_{i=1}^c \sum_{j=1}^N (\mu_j^{(i)})^q \sum_{k=1}^n (m_k^{(i)} - x_k^{(j)})^2 \quad (2.3)$$

Gdzie:

$X = \{x^{(j)} \in R^n : j = 1, 2, \dots, N\}$ jest zbiorem punktów poddanych klasteryzacji,

$M = \{m^{(i)} \in R^n : i = 1, 2, \dots, c\}$ jest zbiorem środków klasterów,

$U = \{\mu_j^{(i)} \in [0, 1] : i = 1, 2, \dots, c; j = 1, 2, \dots, N\}$ jest zbiorem stopni przynależności punktu $x^{(j)}$ do klastera o środku w $m^{(i)}$,

$q > 1$ jest współczynnikiem rozmytości klasterów.

Jako metrykę w przestrzeni przyjąłem metrykę euklidesową.

Funkcja (2.3) przyjmuje tylko wartości nieujemne. Widać teraz, jak ważne jest spełnienie warunku (2.1). Gdyby tego warunku nie było minimalizacja wartości funkcji (2.3) sprowadziłaby się do położenia $\mu_j^{(i)} = 0 \ \forall i = 1 \dots c, j = 1 \dots N$.

Ponieważ nie udało mi się dotrzeć do oryginalnej pracy Bezdeka opisującej sposób dojścia do wzorów definiujących algorytm c-środków poniżej przedstawię moją propozycję obliczenia warunków koniecznych na istnienie ekstremum funkcji (2.3) przy warunkach (2.1).

Zastosuję metodę mnożników Lagrange'a na znajdowanie ekstremum warunkowego opisaną między innymi w [Maurin 73]. Stosując w/w metodę w wyniku różniczkowania następującej funkcji, której postać wynika z tw. Lagrange'a.

$$f_X(U, M) = \sum_{i=1}^c \sum_{j=1}^N (\mu_j^{(i)})^q \sum_{k=1}^n (m_k^{(i)} - x_k^{(j)})^2 - \sum_{j=1}^N \lambda_j [(\sum_{k=1}^c \mu_j^{(k)}) - 1]$$

dostajemy następujący układ równań postaci:

$$q(\mu_j^{(i)})^{q-1} \sum_{k=1}^n (m_k^{(i)} - x_k^{(j)})^2 = \lambda_j, \quad i = 1 \dots c, j = 1 \dots N \quad (2.4)$$

(różniczkując po $\mu_j^{(i)}$) oraz dodatkowe cn równań

$$\sum_{j=1}^N (\mu_j^{(i)})^q (2m_k^{(i)} - 2x_k^{(j)}) = 0, \quad i = 1 \dots c, k = 1 \dots n \quad (2.5)$$

(różniczkując po $m_k^{(i)}$).

Dodatkowo mamy na uwadze N równań (2.1).

Otrzymaliśmy układ $cN + cn + N$ równań

z niewiadomymi $m_k^{(i)}, \mu_j^{(i)}, \lambda_j, i = 1 \dots c, j = 1 \dots N, k = 1 \dots n$ ($cN + cn + N$ niewiadomych).

Przekształcając równania typu (2.5) otrzymujemy:

$$m_k^{(i)} = \frac{\sum_{j=1}^N (\mu_j^{(i)})^q x_k^{(j)}}{\sum_{j=1}^N (\mu_j^{(i)})^q} \quad \forall i = 1 \dots c, \forall k = 1 \dots n \quad (2.6)$$

Jak wynika ze wzoru (2.6), kolejne współrzędne klasterów są średnią ważoną wszystkich punktów z uwzględnieniem stopnia przynależności, jako wagi. Stąd właśnie nazwa algorytmu.

Przekształcając z kolei równania typu (2.4) należy zauważyć, iż składnik λ_j występuje po prawej stronie c równań. Przyrównując, zatem, do siebie lewe strony tych równań otrzymujemy:

$$(\mu_j^{(u)})^{q-1} \sum_{k=1}^n (m_k^{(u)} - x_k^{(j)})^2 = (\mu_j^{(v)})^{q-1} \sum_{k=1}^n (m_k^{(v)} - x_k^{(j)})^2, u, v = 1 \dots c, j = 1 \dots N$$

Co za tym idzie dla pewnego $i = 1 \dots c$

$$\mu_j^{(v)} = \mu_j^{(i)} \left(\frac{\sum_{k=1}^n (m_k^{(i)} - x_k^{(j)})^2}{\sum_{k=1}^n (m_k^{(v)} - x_k^{(j)})^2} \right)^{\frac{1}{q-1}} \quad \forall v = 1 \dots c, j = 1 \dots N \quad (2.7)$$

Chcąc wyliczyć $\mu_j^{(i)}$ możemy podstawić w j-tym równaniu (2.1) za wszystkie wartości $\mu_j^{(v)}, v \neq i$ wyrażenia uzyskane z (2.7).

Otrzymujemy ostatecznie:

$$\mu_j^{(i)} = \frac{1}{\sum_{l=1}^c \left(\frac{\sum_{k=1}^n (m_k^{(i)} - x_k^{(j)})^2}{\sum_{k=1}^n (m_k^{(l)} - x_k^{(j)})^2} \right)^{\frac{1}{q-1}}} \quad (2.8)$$

Warunki (2.6) i (2.8) muszą zajść, aby funkcja mogła mieć ekstremum warunkowe przy warunkach (2.1).

Algorytm c-środków działa według następującego schematu:

1. Ustal $\varepsilon > 0$: wartość progową determinującą zakończenie algorytmu.
Wyznacz losowo macierz U_0 stopni przynależności ${}^0\mu_j^{(i)}$, tak, aby spełnione były warunki (2.1). Ustaw $k = 0$.
2. Ustaw $k := k + 1$. Na podstawie wzoru (2.6) oblicz potencjalne środki klasterów.
3. Oblicz macierz U_k stopni przynależności ${}^k\mu_j^{(i)}$ przy pomocy wzoru (2.8).

4. Jeżeli $\|U_k - U_{k-1}\| < \varepsilon$ to zakończ poszukiwanie klasterów, jeśli nie to idź do kroku 2.

Z punktem 4 powyższego algorytmu można wiązać następujące intuicje. Jeżeli macierz stopni przynależności nie zmieniła się (znacząco) w kolejnym kroku algorytmu oznacza to, iż wyliczone we wzorach (2.6) oraz (2.8) parametry klasterów odpowiadają sobie nawzajem, zatem funkcja (2.3) posiada w tym punkcie ekstremum warunkowe.

Opisany wyżej algorytm jest efektywną metodą znajdowania klasterów, jednak posiada także znaczącą wadę, którą jest fakt, iż odszukane klasterzy mają kształt kulisty. Kształt klasterów jest zupełnie niezależny od rozmieszczenia poddawanych grupowaniu danych w przestrzeni. Co za tym idzie tak uzyskane klasterzy często nie korespondują z wyobrażeniem na temat konkretnej klasteryzacji danego zbioru punktów. Nasuwa się tutaj następujące spostrzeżenie, którego dokonali Gustafson i Kessel. Ich modyfikacja algorytmu FCM opisana między innymi w [Klawonn 97]. Ideą algorytmu Gustafsona-Kessela (GK) jest poddanie procesowi optymalizacji także kształtu docelowego klasterów poprzez optymalizowanie odległości, która w FCM była odległością Euklidesową, tak, aby punkty o równej odległości od środka klasterów tworzyły nie sfery, lecz powierzchnie hiperelipsoid.

Oryginalny algorytm GK pozwala na uzyskanie hiperelipsoid o osiach symetrii zorientowanych pod dowolnym kątem w stosunku do osi układu współrzędnych rozważanej przestrzeni. Ponieważ, jak zauważyłem na początku niniejszego rozdziału po procesie klasteryzacji następuje rzutowanie klastera na poszczególne osie szczególnie interesować będą nas hiperelipsoidy o osiach równoległych do osi układu współrzędnych. Podobna teza została postawiona także w [Klawonn 97]. Jednak w w/w artykule oprócz wspomnianego algorytmu GK do uzyskania klasteryzacji autorzy stosują także algorytm, którego autorami są Gath i Geva, opublikowany 10 lat po Gustafsonie i Kesselu. W niniejszej pracy ograniczę się do zastosowania algorytmu GK oraz do uzasadnienia, iż istotnie klasterzy o osiach równoległych do osi przestrzeni powinny wystarczyć.

2.2.2. Algorytm Gustafsona-Kessela (GK) – wersja zmodyfikowana.

Oryginalny algorytm Gustafsona-Kessela polega na zmianie funkcji definiującej odległość punktu od środka klastera z euklidesowej na następującą, co istotne, inną dla każdego z klasterów.

$$d_i(x_j) = (m_i - x_j)^T A_i (m_i - x_j) \quad (2.9)$$

Gdzie A_i jest macierzą $n \times n$ przekształcenia przestrzeni odpowiednią dla każdego z klasterów $i = 1, \dots, c$.

Jak opisano w [Babuška 01] (za [Gustafson 79]) elementy macierzy A_i oblicza się na podstawie rozmytych macierzy kowariancji dla każdego z klasterów. Macierze te oddają odchylenie danych od środka klasterów, jako wagę biorąc stopień przynależności do klastera.

Aby dostosować kształty klasterów do aktualnego rozłożenia poddawanych klasteryzacji danych dodatkowymi parametrami minimalizacji funkcji (2.3), która po zamianie funkcji odległości przybierze postać

$$f_X(U, M, A) = \sum_{i=1}^c \sum_{j=1}^N \mu_{i,j}^q d_i(x_j) \quad (2.10)$$

jest także rodzina macierzy $A = \{A_i : i = 1, \dots, c\}$.

Jak wynika ze wzoru (2.9) funkcja (2.10) może przyjąć dowolnie małe wartości, gdy nie założymy żadnych dodatkowych warunków odnośnie macierzy A_i .

Dlatego, aby dokonać „sensownej” minimalizacji zakłada się, że wyznacznik każdej z macierzy A_i powinien przybrać pewną ściśle określoną wartość σ_i . Jak wcześniej wspomniałem algorytm dzięki zastosowaniu metryk (2.9) algorytm Gustafsona-Kessela pozwala na uzyskanie klasterów o dowolnej orientacji względem osi przestrzeni. Ponieważ po fazie klasteryzacji wykonane ma zostać rzutowanie na poszczególne wejścia, najbardziej interesujące (z punktu widzenia dokładności przenoszonej przez rzutowania informacji) będą klustery elipsoidalne o osiach równoległych do osi przestrzeni. Dzięki temu spostrzeżeniu możemy ograniczyć się w naszych rozważaniach do wybierania macierzy A_i spośród klasy macierzy diagonalnych. Spowoduje to istotne zmniejszenie złożoności na poziomie tworzenia macierzy A_i .

Jak wspomniałem minimalizacja będzie odbywać się przy założeniu, że wyznacznik z A_i jest równy pewnej ustalonej liczbie. Dla macierzy diagonalnej warunek ten przybierze postać:

$$\prod_{j=1}^n \delta_j^{(i)} = \sigma_i \quad \forall i = 1 \dots c \quad (2.11)$$

Gdzie:

$$A_i = \begin{bmatrix} \delta_1^{(i)} & 0 & \dots & 0 & 0 \\ 0 & \delta_2^{(i)} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \delta_{n-1}^{(i)} & 0 \\ 0 & 0 & \dots & 0 & \delta_n^{(i)} \end{bmatrix}$$

Ponownie korzystając z metody mnożników Lagrange'a (różniczkując (2.10) oraz (2.11) po $\delta_k^{(i)}$) otrzymujemy układ równań postaci:

$$\sum_{j=1}^N \mu_{i,j}^q (m_k^{(i)} - x_k^{(j)})^2 - \lambda_i \prod_{l=1, l \neq k}^n \delta_l^{(i)} = 0 \quad \forall i = 1, \dots, c; \forall k = 1 \dots n \quad (2.12)$$

Dodatkowo mamy na uwadze c równań (2.11).

Korzystając z (2.11) dostajemy

$$\sum_{j=1}^N \mu_{i,j}^q (m_k^{(i)} - x_k^{(j)})^2 = \lambda_i \frac{\sigma_i}{\delta_k^{(i)}} \quad \forall i = 1, \dots, c; \forall k = 1 \dots n \quad (2.13)$$

Dla pewnego i mnożąc stronami n odpowiednich równań postaci (2.13) dostajemy

$$\prod_{k=1}^n \left(\sum_{j=1}^N \mu_{i,j}^q (m_k^{(i)} - x_k^{(j)})^2 \right) = \lambda_i^n \sigma_i^{n-1} \quad \forall i = 1, \dots, c$$

A co za tym idzie:

$$\lambda_i = \left(\frac{\prod_{k=1}^n \left(\sum_{j=1}^N \mu_{i,j}^q (m_k^{(i)} - x_k^{(j)})^2 \right)}{\sigma_i^{n-1}} \right)^{\frac{1}{n}} \quad \forall i = 1, \dots, c \quad (2.14)$$

Mając wyliczone poszczególne λ_i możemy podstawić je do (2.13) uzyskując ostatecznie:

$$\delta_k^{(i)} = \frac{\left(\sigma_i \prod_{u=1}^n \left(\sum_{j=1}^N \mu_{i,j}^q (m_u^{(i)} - x_u^{(j)})^2 \right) \right)^{\frac{1}{n}}}{\sum_{j=1}^N \mu_{i,j}^q (m_k^{(i)} - x_k^{(j)})^2} \quad \forall i = 1, \dots, c; \forall k = 1 \dots n \quad (2.15)$$

Wzór (2.15) opisuje sposób wyliczania macierzy w kolejnych iteracjach.

Do algorytmu FCM pomiędzy kroki 2 a 3 dodajemy nowy krok 2a: na podstawie wzoru (2.15) oblicz rodzinę macierzy A_i .

Łatwo zauważać, że wzory wyprowadzone dla FCM zachowują ważność przy tak zmienionej metryce.

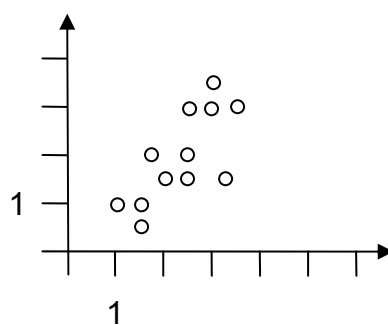
2.3. Przetwarzanie wyników klasteryzacji.

Dane uzyskane w wyniku wykonania algorytmu klasteryzacyjnego na próbkach pomiarowych muszą zostać poddane dalszemu przetwarzaniu w celu uzyskania modelu rozmytego.

Proponowane przeze mnie podejście polega na rzutowaniu uzyskanych klasterów na kolejne wejścia oraz wyjście. W wyniku rzutowania klastera $C_i \in X_1 \times \dots \times X_n$ (który de facto jest zbiorem rozmytym o stopniach przynależności uzyskanych w wyniku przeprowadzenia klasteryzacji) na kolejne X_k (dla uproszczenia oznaczeń przyjmujemy, że $Y = X_n$) dostajemy rodzinę zbiorów rozmytych o następującej postaci: $D_k^i = \{(\mu(x), x) : x \in X_k, \exists x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n : (x_1, \dots, x_{k-1}, x, x_{k+1}, \dots, x_n) \in C_i\}$.

Gdzie $\mu(x)$ jest supremum z wszystkich stopni przynależności próbek pomiarowych mających wartość x na pozycji k .

Opisana metoda napotyka na następujący problem, który zobrazuje poniższym przykładem.



Rys. II-2. Przykładowe dane do rzutowania.

W wyniku klasteryzacji danych przedstawionych na rysunku wyżej przy użyciu zmodyfikowanego algorytmu Gustafsona-Kessela z zadaną liczbą trzech klasterów uzyskano następujące wyniki:¹

Środki klasterów

$C1 = (1.36, 0.86)$

$C2 = (2.99, 3.11)$

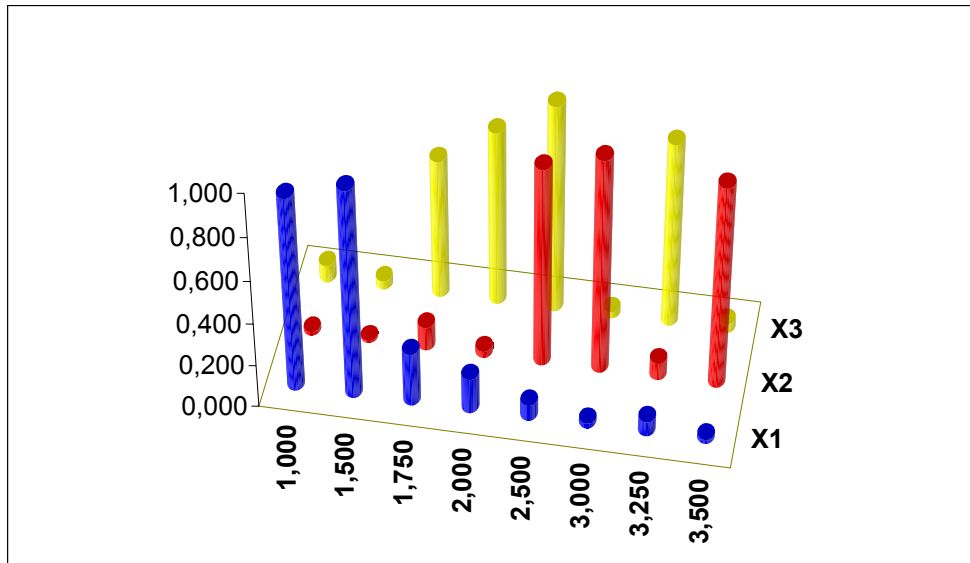
$C3 = (2.47, 1.66)$

Stopnie przynależności poszczególnych punktów do klasterów zobrazowano w poniższej tabeli:

x	Y	μ_{C1}	μ_{C2}	μ_{C3}
1,000	1,000	0,903	0,017	0,080
1,500	1,000	0,971	0,004	0,025
1,500	0,500	0,948	0,011	0,041
2,000	1,500	0,163	0,029	0,808
2,500	1,500	0,028	0,013	0,959
2,500	2,000	0,074	0,097	0,830
1,750	2,000	0,248	0,109	0,643
3,250	1,500	0,068	0,079	0,853
2,500	3,000	0,033	0,919	0,049
3,000	3,000	0,003	0,992	0,005
3,500	3,000	0,020	0,937	0,044
3,000	3,500	0,023	0,948	0,029

W wyniku opisanego wcześniej rzutowania na oś X uzyskano punkty wyznaczające przeszłe zbiory rozmyte. Punkty rozłożyły się jak pokazano poniżej:

¹ Wyniki pochodzą z programu ClusterIt opisanego w dalszej części pracy. Ustawienia programu: minimum difference: 0.001, cluster capacity: 1, number of clusters: 3, fuzziness factor: 2.



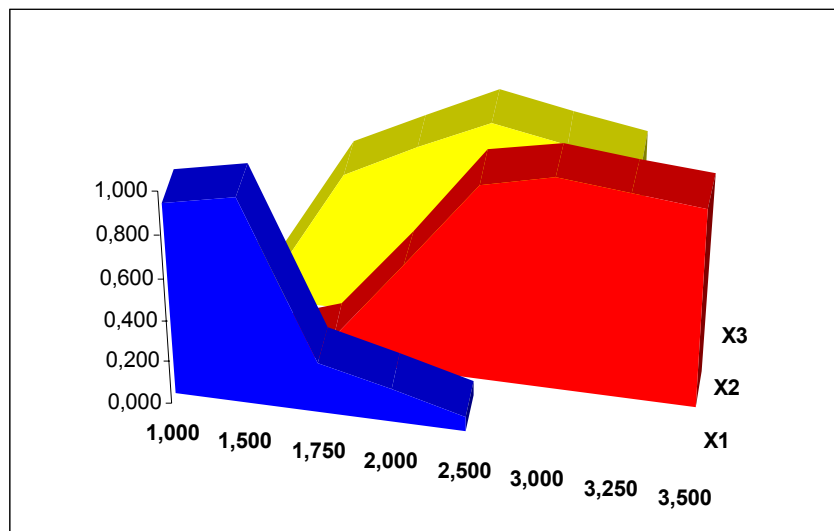
Rys. II-3 Wyniki działania programu CluterIt!

Jak widać sytuacja nie odpowiada intuicjom. Wartości 3 oraz 3,5 należą w znacznym stopniu do zbioru X2 podczas gdy punkt 3,25 nie należy do niego prawie wcale (por. tabela). Z uwagi na charakter klasteryzacji rozmytej prawie każdy z punktów należy w stopniu niezerowym do prawie wszystkich zbiorów. Z punktu widzenia sterowania sytuacja ta nie jest także pożądana ze względu na jednoczesną aktywację zbyt wielu reguł. Przyjmuje się podejście, aby skrajne wartości funkcji przynależności do danego zbioru, które są mniejsze od pewnej przyjętej wartości progowej (np. 0,05) przyrównać do zera.

Pozostaje jednak jeszcze problem kształtu zbiorów. Aby model odpowiadał intuicjom i nie był podatny na nierównomierne rozmieszczenie próbek pomiarowych proponuję jako końcowy zbiór rozmyty przyjąć zbiór uzyskany w następujący sposób.

Dla rodziny $\{x_k : k = 1, \dots, N\}$ rzutów próbek pomiarowych uporządkowanych tak, że $x_k < x_l$, gdy $k < l$ jeżeli $\mu_X(x_k) < \mu_X(x_{k-1})$ oraz $\mu_X(x_k) < \mu_X(x_{k+1})$ to przyjmujemy $\mu_X(x_k)$ jako wartość wynikającą z interpolacji prostą przechodzącą przez punkty $(x_{k-1}, \mu_X(x_{k-1})), (x_{k+1}, \mu_X(x_{k+1}))$.

Po zastosowaniu opisanych wyżej przekształceń do przykładowych danych otrzymamy następujący zestaw zbiorów rozmytych:



Rys. II-4 Proponowane kształty zbiorów rozmytych.

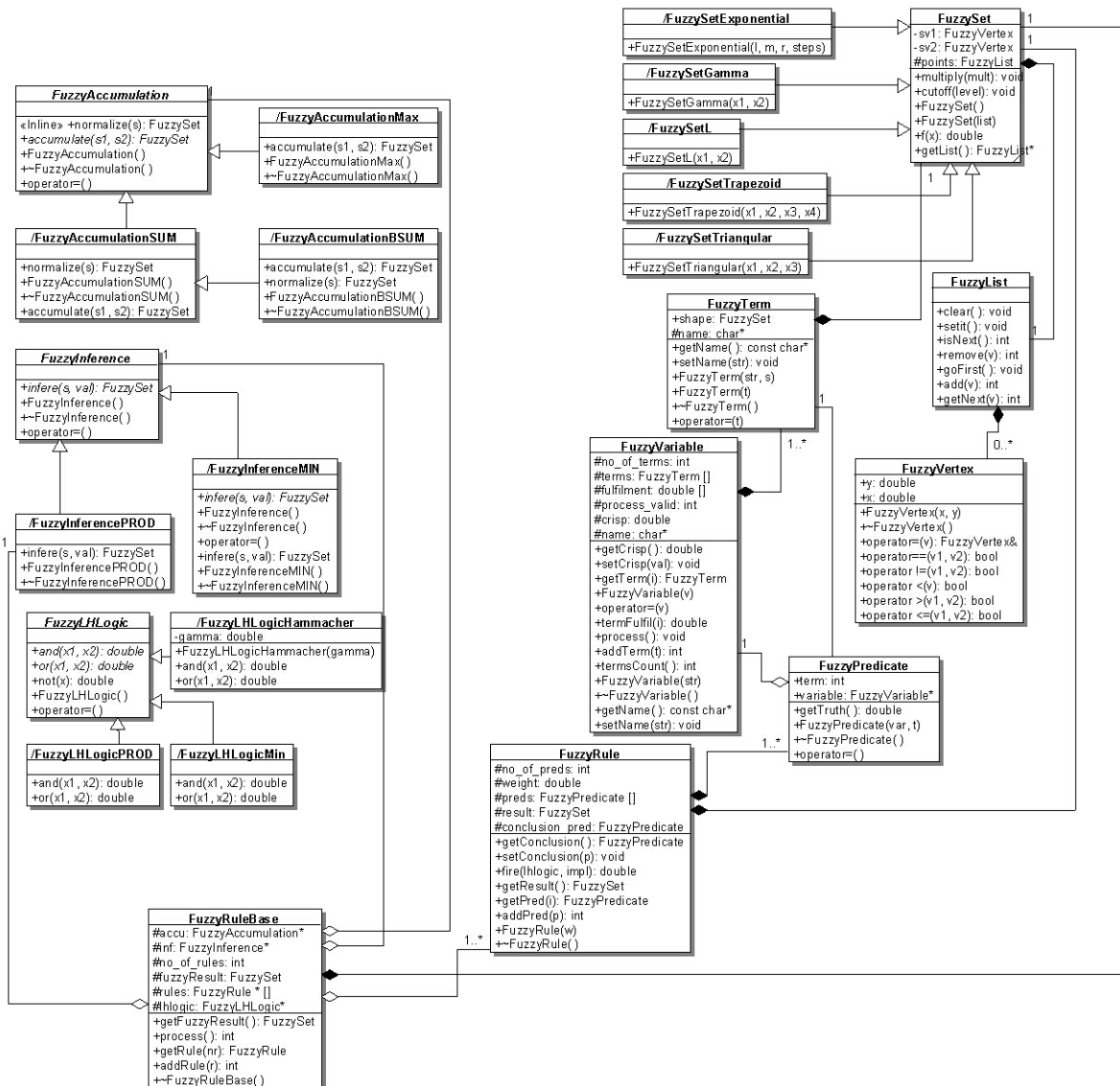
Innym podejściem (proponowanym między innymi w [Klawonn 97]) jest przybliżenie końcowych zbiorów rozmytych poprzez zbiory o kształtach trapezoidalnych lub trójkątnych. Zaletą tego podejścia jest otrzymanie stosunkowo prostych (z punktu widzenia późniejszych obliczeń) zbiorów, wadą natomiast utrata pewnej ilości informacji w procesie przybliżania.

W zaproponowanym przeze mnie podejściu na ostateczny kształt zbiorów powinien mieć wpływ człowiek, którego zadaniem będzie aproksymacja kształtu na podstawie dostarczonego mu zestawu punktów rzutowanego na daną oś. Jest to zadanie o wiele łatwiejsze niż zadanie podobne, jednak nie poprzedzone etapem klasteryzacji.

III. SZCZEGÓŁY IMPLEMENTACJI ORAZ STWORZONE NARZĘDZIA.

W czasie implementacji środowiska programistycznego ułatwiającego tworzenie sterowników rozmytych przyświecała mi idea programowania obiektowego, a w szczególności polimorfizmu. Chciałem także utworzyć język wyższego poziomu ideologicznie zbliżony do FCL [IEC 97], w którym użytkownik tworzyłby niezależne od implementacji specyfikacje sterowników rozmytych. Tak zrodził się FML – oparty na XML język opisu sterownika, tłumaczony następnie na C++.

1. SZCZEGÓŁY IMPLEMENTACJI KLAS STEROWNIKA.

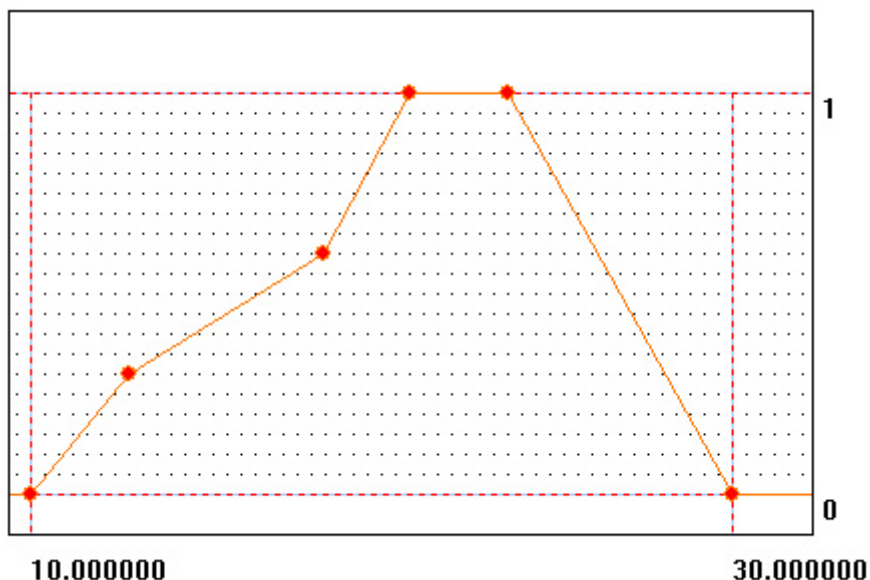


Rys. III-1 Diagram klas użytych do konstrukcji sterownika.

Na rysunku przedstawiono schemat w UML przedstawiający klasy użyte w konstrukcji sterownika rozmytego oraz zależności między nimi. Klasy powyższe stanowią bibliotekę którą nazwałem **fuzzylib**.

Podstawową klasą implementacji jest klasa *FuzzySet*, która jest odpowiednikiem zbioru rozmytego. Klasa *FuzzySet* opiera się na klasie *FuzzyList*, która została zaimplementowana z wykorzystaniem klasy *list* ze standardowej biblioteki wzorców C++ (STL). Zbiór rozmyty w rozumieniu *FuzzySet* jest uporządkowanym zbiorem odcinków łączących punkty określone na nośniku rzeczywistym. W ciągu punktów nie może być dwóch punktów o tej samej współrzędnej x.

W związku z powyższym zbiór rozmyty określany przez *FuzzySet* (i wszystkie klasy pochodne) posiada ciągłą funkcję charakterystyczną. Funkcja charakterystyczna na lewo od pierwszego (x_0, y_0) jest funkcją stałą o wartości y_0 . Podobnie na prawo od ostatniego punktu (x_n, y_n) funkcja jest funkcją stałą o wartości y_n .



Rys. III-2 Przykładowy zbiór definiowany przez obiekt typu *FuzzySet*.

Klasy dziedziczące z *FuzzySet* mają ułatwić tworzenie zbiorów rozmytych o standardowych kształtach. Dostępne kształty zostały opisane przy okazji omówienia języka FML w dalszej części tego rozdziału.

Klasa *FuzzySet* została opakowana w klasę *FuzzyTerm*. *FuzzyTerm* jest niczym innym jak tylko nazwanym zbiorem rozmytym. Zbiór obiektów typu *FuzzyTerm* tworzy zmienną lingwistyczną reprezentowaną w systemie przez klasę *FuzzyVariable*. Ta ostatnia oprócz charakteryzujących ją termów posiada także nazwę. Zmienna lingwistyczna wraz z określeniem jednego z jej termów tworzy predykat rozmyty (egzemplarz klasy *FuzzyPredicate*).

Główne klasy sterownika to *FuzzyRule* oraz *FuzzyRuleBase*.

Pierwsza z nich zawiera zestaw predykatów (*FuzzyPredicate*) określających poprzednik reguły, oraz jeden predykat określający jej następnik. Taki model pozwala konstruować reguły z jedną zmienną w wyniku. Predykaty w poprzedniku związane są logicznym operatorem *AND*. W obrębie pojedynczej bazy reguł *FuzzyRuleBase* znajduje się zestaw reguł *FuzzyRule* oraz zdefiniowane są 3 obiekty wspomagające

typu *FuzzyLHLogic*, *FuzzyAccumulation* oraz *FuzzyInference* określające algorytm wnioskowania dla bazy reguł.

Wydaje się, iż takie podejście mocno ogranicza uzyskiwane sterowniki do takich, które w poprzednikach używają tylko operacji *AND* oraz posiadają jedno wyjście. Nie jest to jednak prawdą ponieważ każdy zbiór reguł używający zarówno operatorów *AND* jak i *OR* można przekształcić do zbioru używającego w poprzednikach tylko operatora *AND*.

Problem ilości wyjść pozwala konstruować sterowniki typu MISO (multiple input – single output), które, jak to opisano w literaturze [Łachwa 01] pozwalają poprzez połączenie n takich sterowników stworzyć sterownik typu MIMO (multiple input – multiple output).

Zaimplementowane rozwiązanie pozwala realizować pojedynczy sterownik typu MISO w obrębie danej bazy reguł, co za tym idzie dzięki stworzeniu wielu baz reguł można w łatwy sposób uzyskać sterownik typu MIMO. Jak więc wykazano powyżej narzucone ograniczenia nie zawężają zakresu zastosowań proponowanego rozwiązania, jednocześnie jednak zmniejszają znacznie poniesiony nakład implementacyjny związany z brakiem konieczności nadmiernego rozrostu proponowanego modelu obiektowego.

Klasy pochodne *FuzzyLHLogic* odpowiedzialne są za realizację operacji *AND* (t-normy) po lewej stronie reguł. Zastosowane przeze mnie warianty to opisane w rozdziale I operatory *MIN*, *PROD* oraz t-norma Hammachera.

Potomkowie *FuzzyInference* to klasy odpowiedzialne za realizację implikacji. Najczęściej używanymi w praktyce są opisane w rozdziale I: *MIN* oraz *PROD*.

Klasy pochodne od *FuzzyAccumulation* realizują kumulację wyników działania poszczególnych reguł. Zaimplementowane sposoby to *MAX* (maksimum), *SUM* (suma nieograniczona, normalizowana na końcu) oraz *BSUM* (suma ograniczona, obcinana przy wartości 1).

Algorytm realizujący wnioskowanie wywołuje kolejno metodę *fire* dla wszystkich reguł należących do danej bazy reguł. Zauważmy (co opisano już w rozdziale I), iż odpalenie każdej reguły wymaga sprawdzenia stopnia spełnienia wszystkich predykatów znajdujących się w jej poprzedniku. Sprawdzenie takie odbywa się w

czasie liniowym w zależności od ilości punktów charakteryzujących występujący w predykcji term. Łatwo zauważyć, że sprawdzenie takie powinno odbywać się raz dla predykatu, a nie raz dla predykatu dla danej reguły, jeżeli dany predykat występuje w kilku regułach. Dlatego, aby zmniejszyć nakład obliczeniowy, jest to realizowane w taki właśnie sposób (predykat raz sprawdzony przy odpalaniu jednej reguły nie zostaje sprawdzany przy odpalaniu innej reguły mającej ten sam predykat w poprzedniku). Rezultat działania każdej reguły jest kumulowany w obrębie każdej bazy reguł w pojedynczy rezultat dla danej bazy. Następnie używając adekwatnej dla danej bazy reguł metody następuje wyostrenie rezultatu i wartość ostra zmiennej jest przypisywana odpowiedniej zmiennej lingwistycznej. Jest to wynik wnioskowania w obrębie danej bazy reguł.

2. JĘZYK OPISU STEROWNIKA.

Używając powyższego zestawu klas można z powodzeniem tworzyć sterowniki rozmyte w C++. Ponieważ jednak wygodne byłoby wyabstrahowanie powyższego zestawu tak, aby można było tworzyć sterownik rozmyty niezależnie od docelowego języka programowania zaproponowałem następujący (oparty na XML), język opisu sterownika, który nazwałem FML (Fuzzy Modelling Language). Użycie języka (będącego poziom wyżej niż C++), który tłumaczony zostanie docelowo na C++ pozwala uniezależnić się od konkretnego języka programowania, a w przyszłości rozszerzyć implementację powyższego zestawu klas na inne języki programowania jednocześnie dostarczając odpowiedni translator nie zmieniając specyfikacji utworzonych sterowników czyniąc całość bardziej skalowalną. Od 1997 roku istnieje przemysłowy standard określający język FCL (Fuzzy Control Language) dla sterowników rozmytych [IEC 97]. Sterownik opisany w FCL (na poziomie zgodności Basic) można łatwo przełożyć na program w FML. FML wykracza w niektórych dziedzinach poza poziom Basic FCL (np. dowolna ilość punktów definiująca zbiór, kaskadowe łączenie bloków reguł). FML nie implementuje jednak spójnika OR ani modyfikatora NOT. Oczywiście jak to opisano w rozdziale I nie jest to konieczne, jednak może stanowić podstawę do rozszerzenia zaproponowanego przeze mnie modelu.

2.1. Składnia języka FML (Fuzzy Modelling Language).

Składnia została opisana podobnie do opisu przyjętego dla gramatyk bezkontekstowych.

Znak + oznacza jedno lub więcej powtórzeń danego nieterminala lub wyrażenia ujętego w nawiasy, znak * zero lub więcej. Znak ? oznacza wystąpienie danego nieterminala lub wyrażenia ujętego w nawiasy co najwyżej raz.

Nieterminale oznaczone zostały w poniższej specyfikacji pogrubioną kursywą.

```
fuzzy_controller:
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE fuzzy SYSTEM "fml.dtd" >
<FUZZY name="controllername">
    <VARIABLES>
        linguistic_variable+
    </VARIABLES>
    rule_block+
</FUZZY>
```

```
linguistic_variable:
    <VAR name="varname">
        term+
    </VAR>
```

Parametrem zmiennej lingwistycznej jest jej nazwa *varname*. Nazwa powinna składać się z liter lub cyfr. Dla zmiennej dostarczony musi zostać zestaw termów.

```
term:
    <TERM name="termname" type="typestring">
        definition
    </TERM>
```

definition:	point+	(gdy type="points")
	(a,b,c,d)	(gdy type="exp")
	(a,b)	(gdy type="gamma")
	(a,b)	(gdy type="l")
	(a,b,c)	(gdy type="triangular") (domyślne)
	(a,b,c,d)	(gdy type="trapezoid")


```
point:
    (x, y)
```

Term zdefiniowany jest poprzez podanie jego nazwy *termname* (z obostrzeniami podobnymi jak dla nazwy zmiennej) oraz typu *typestring*.

Możliwe typy *typestring* to points, exp, gamma, l, triangular (ustawiony domyślnie), trapezoid.

Typ points:

Zbiór rozmyty definiujący znaczenie termu opisuje się używając ciągu punktów. Każdy z punktów powinien posiadać dwie współrzędne rzeczywiste, y powinno być z przedziału $[0,1]$. W związku z konstrukcją opisaną wyżej klasy *FuzzySet* w zestawie punktów może istnieć tylko jeden punkt o danej współrzędnej $x=x_0$. Każdy kolejny punkt o współrzędnej $x=x_0$ zostanie zignorowany. Ten typ zbioru rozmytego pozwala zdefiniować zbiór każdego z poniższych typów. Inne typy zostały wprowadzone dla uproszczenia procesu tworzenia zbiorów o „popularnych” kształtach.

Typ exp:

Definiuje zbiór rozmyty o funkcji przynależności aproksymującej niesymetryczną funkcję Gaussa [Piegat 99].

Do zdefiniowania go wymagane są 4 liczby.

a – lewy koniec zbioru $\mu(x) = 0$ dla $x \leq a$,

b – punkt szczytowy $\mu(x) = 1$ dla $x = b$,

c – prawy koniec zbioru $\mu(x) = 0$ dla $x \geq c$,

d – dokładność aproksymacji. Funkcja w przedziale $[a,b]$ oraz w przedziale $[b,c]$ aproksymowana będzie d odcinkami na przedział. Powinna przyjmować wartość naturalną od 1 do 10. Zalecane 4.

Typ gamma:

Definiuje zbiór typu gamma, poprzez podanie punktu początku wzrostu (a) oraz punktu zakończenia wzrostu (b) funkcji przynależności.

Typ l:

Zbiór symetryczny do zbioru typu gamma.

Typ traingular:

Zbiór typu trójkątny, a,b,c definiują współrzędne x wierzchołków trójkąta.

Typ trapezoid:

Zbiór o kształcie trapezu. Liczby a, b, c, d definiują kolejne współrzędne x wierzchołków trapezu.

Dla prawidłowej definicji zbioru pomiędzy wymienionymi wyżej liczbami a, b, c, d musi zachodzić nierówność $a < b < c < d$.

```
rule_block:
<RULEBLOCK (tnorm="t-norm_type") (gamma="gamma_val") (inference="implication_type")
(accumulation="accumulation_type") (defuzzify="defuzzification_type")
outvar="varname">
    <INPUTS>
        <INPUT name="inputvarname"/>+
    </INPUTS>
    rule+
</RULEBLOCK>
```

Zdefiniowanie bazy reguł wymaga wyspecyfikowania t-normy, sposobu realizacji implikacji, kumulacji oraz wyostrzania. Podać należy także zbiór reguł zawarty w danym bloku oraz nazwę zmiennej wyjściowej *varname*, której dotyczy dany blok. Listę zmiennych wejściowych dla danego bloku reguł specyfikujemy podając nazwy zmiennych w klauzuli **INPUTS**. W przypadku pominięcia atrybutów *tnorm*, *inference*, *accumulation* oraz *defuzzify* przyjęte zostaną odpowiednio następujące wartości domyślne: min, min, max, lmm.

```
t-norm_type:      min
                  |      prod
                  |      hammacher

gamma_val:        liczba_rzeczywista>=0 (domyślnie 0)
```

Można wybrać kolejno normę minimum (min), produkt (prod) oraz normę Hammachera (hammacher). W przypadku tej ostatniej w parametrze *gamma* należy podać liczbę definiującą daną wersję normy Hammachera. Gdy wybrana zostanie norma min lub prod parametr *gamma* jest ignorowany.

```
implication_type: min (domyślnie)
                  |      prod
```

Implementowane rodzaje realizacji implikacji to minimum (min) oraz produkt (prod).

```
accumulation_type:  max (domyślnie)
                   |      sum
                   |      bsum
```

Wyniki działania poszczególnych reguł można kumulować używając do tego celu kumulacji typu maksimum (max), suma nieograniczona normalizowana ostatecznie do 1 (sum) oraz suma ograniczona obcinana każdorazowo na poziomie 1 (bsum).

```
defuzzification_type:  cog (domyślnie)
                      |      lmm
                      |      rmm
```

Możemy wybrać następującą metodę wyostrażania wyniku: środek ciężkości – cog, pierwsze z lewej maksimum (lmm) lub pierwsze z prawej maksimum (rmm). Używając kumulacji typu sum oraz metody wyostrażania cog dostaniemy wyostrażenie typu środek sum (Center of Sums).

```
rule:
    <RULE out="outtermname" (weight="value")>
    predicate+
    </RULE>
```

```
predicate:
    <IS var="varname" term="termname"/>
```

Regułę rozmytą specyfikujemy podając nazwę termu *outtermname*, wagę reguły *value* oraz oddzieloną przecinkami listę predykatów. Ponieważ dana reguła zawarta jest docelowo w bloku reguł o określonej zmiennej wyjściowej wystarczy podać tylko *outtermname* miast podawać nazwę zmiennej wyjściowej oraz nazwę termu.

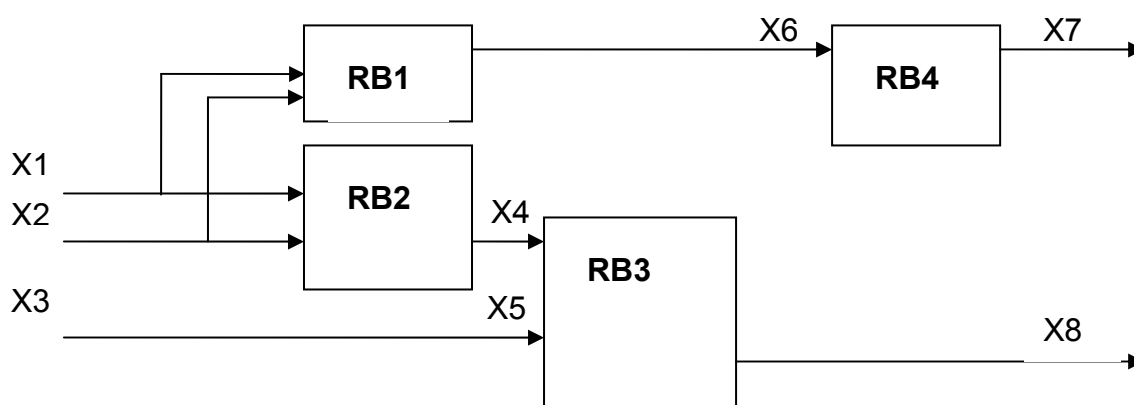
Niepodanie *weight="value"* powoduje ustawienie wagi reguły na 1.

Specyfikując predykat należy podać nazwę zmiennej *varname* oraz nazwę termu skojarzonego z tą zmienną *termname*. *varname* powinna być różna od wartości *outvar* podanej dla bloku reguł, w którym zawarta jest reguła. Zapewni to odróżnienie w obrębie danego bloku zmiennych wejściowych i wyjściowych.

Wnioskowanie odbywa się w kolejnych blokach reguł w kolejności ich specyfikowania w dokumencie FML. Co za tym idzie możliwe jest uzyskanie efektu kaskadowego

połączenia bloków reguł, tak aby kolejne bloki spełniały rolę podsystemów wnioskujących. Przykładowe połączenie kaskadowe bloków reguł przedstawione zostało na rysunku niżej.

W przedstawionym przykładzie wejścia systemu to X1,X2,X3. System posiada 2 wyjścia: X7 oraz X8. Kolejność specyfikowania bloków reguł w kodzie FML powinna spełniać następujący warunek. Dla każdej zmiennej wejściowej X bloku R, która na schemacie jest zmienną wyjściową innego bloku B definicja bloku B powinna znaleźć się przed definicją bloku R. Graf zależności między blokami nie może posiadać cykli. Jedną z odpowiednich kolejności specyfikowania bloków reguł dla poniższego przykładu jest RB1, RB2, RB3, RB4.



Rys. III-3 Kaskadowe łączenie bloków reguł.

Przykładowy fragment bloku reguł, używający jako metody kumulacji metody maksimum, środka ciężkości do wyostrzania, a jako t-normy i implikacji minimum:

IF speed **IS** high **AND** distance **IS** small **THEN** acceleration **IS** negative

IF distance **IS** high **THEN** acceleration **IS** positive

Może zostać przepisany jako:

```

<RULEBLOCK tnorm="min" inference="min" accumulation="max" defuzzify="cog"
outvar="acceleration">
<INPUTS><INPUT var="speed"/><INPUT var="distance"/></INPUTS>
<RULE out="negative"><IS var="speed" term="high"/><IS var="distance" term="small"/>
</RULE>
<RULE out="positive"><IS var="distance" term="high"/>
</RULE>

```

</RULEBLOCK>

2.1.1. Definicja typu dokumentu FML (DTD).

Poprawny syntaktycznie dokument FML powinien być zgodny z poniższą definicją typu dokumentu XML. Zgodność z tą definicją nie świadczy jednak o pełnej poprawności dokumentu FML. DTD nie daje możliwości sprawdzania wartości elementów oraz atrybutów pod względem ich poprawności. Definicja ta może być jednak wykorzystana w procesie tworzenia dokumentów FML przy użyciu edytorów ułatwiających pracę z XML. Opisany później fml2cpp podczas konwersji sprawdza nie tylko zgodność z DTD ale poprawność dokumentu FML w ogóle.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT FUZZY (VARIABLES, RULEBLOCK+)>
<!ELEMENT VARIABLES (VAR+)>
<!ELEMENT VAR (TERM+)>
<!ELEMENT TERM (#PCDATA)>
<!ELEMENT RULEBLOCK (INPUT+, RULE+)>
<!ELEMENT INPUT EMPTY>
<!ELEMENT RULE (IS+)>
<!ELEMENT IS EMPTY>
<!ATTLIST FUZZY name CDATA "controller">
<!ATTLIST VAR name CDATA #REQUIRED>
<!ATTLIST TERM
    name CDATA #REQUIRED
    type (points | exp | gamma | l | triangular | trapezoid) "triangular"
>
<!ATTLIST RULEBLOCK
    tnorm (min | prod | hammacher) "hammacher"
    gamma CDATA "0"
    inference (min | prod) "min"
    accumulation (max | sum | bsum) "max"
    defuzzify (cog | lmm | rmm) "cog"
    outvar CDATA #REQUIRED
>
<!ATTLIST INPUT name CDATA #REQUIRED>
<!ATTLIST RULE
    out CDATA #REQUIRED
    weight CDATA "1"
>
<!ATTLIST IS
    var CDATA #REQUIRED
    term CDATA #REQUIRED
>
```

2.2. Proces translacji FML na C++.

Kod źródłowy w FML jest pseudo kodem, który nie podlega bezpośrednio wykonaniu. Specyfikacja FML tłumaczona jest na program operujący na klasach opisanych w punkcie 1. Nazwy używane w specyfikacji FML nie mogą być słowami kluczowymi języka C++, muszą zaczynać się od litery, mogą zawierać znaki alfanumeryczne oraz znak podkreślenia `_` a ich długość nie powinna przekraczać 16 znaków. Wielkość liter w nazwach używanych w dokumencie FML jest istotna. Ważna jest także wielkość liter w słowach kluczowych (nazwy elementów pisane dużymi literami, nazwy atrybutów pisane małymi literami, wartości atrybutów o charakterze wyliczeniowym pisane małymi literami).

Proces translacji przebiega w trzech fazach:

1. Sprawdzenie syntaktycznej zgodności dokumentu FML z definicją typu DTD zawartą w dołączonym do tłumacza pliku, opisującą gramatykę poprawnego dokumentu FML.
2. Sprawdzenie semantycznej poprawności dokumentu FML (sprawdzane zostają tutaj rzeczy takie jak przynależność używanych w predykatach termów do używanej zmiennej lingwistycznej, należenie liczb do odpowiednich zakresów itp.)
3. Wygenerowanie szeregu plików C++, w tym klasy *FuzzyController_name* posiadającej publiczną metodę *process()*, która oczekuje na podanie zmiennych wejściowych, a wyniki przekazuje poprzez referencje. Dokładny sposób użycia metody *process()* zależy od specyfikacji podanej w dokumencie FML (tłumacz samodzielnie determinuje ilość wejść i wyjść sterownika na podstawie specyfikacji). Wartości zmiennych przekazuje się przez referencję. Wartości wyjść można odczytać odczytując odpowiednie zmienny po wykonaniu metody. Metoda *process()* zwraca liczbę całkowitą oznaczającą ilość reguł, które zadziałały. Jeżeli zwrócone zostanie 0 oznacza to, że sterownik nie został zdefiniowany dla podanych wartości wejść, a co za tym idzie nie zadziałała żadna reguła. Program korzystający z metody *process()* może w takim wypadku zdecydować jaką wartość podać na wyjście (czy pozostawić poprzednią, czy podać zero, czy zwrócić błąd).

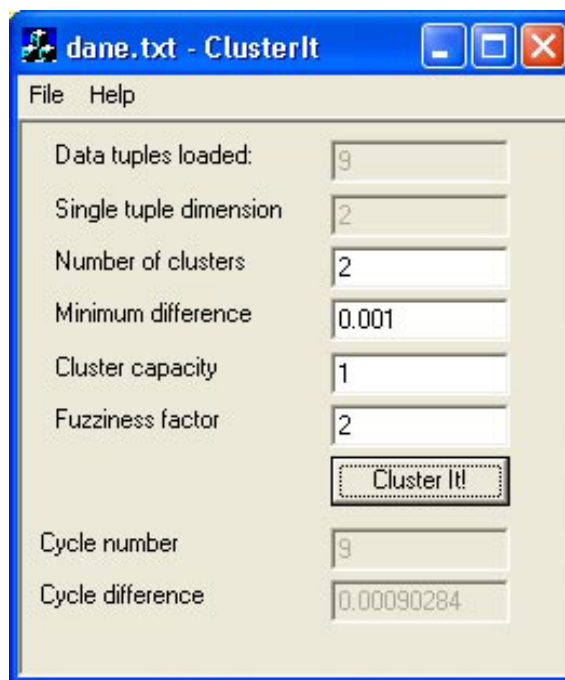
Proces translacji realizowany jest przez program **fml2cpp** (opisany w rozdziale III) napisany w Visual C++ korzystający z parsera języka XML - MSXML firmy Microsoft.

3. PROGRAMY NARZĘDZIOWE.

Dla wspomagania tworzenia sterowników rozmytych poza zestawem klas opisanych w punkcie 1 niniejszego rozdziału, stanowiącym niejako proste API projektanta systemów rozmytych stworzony został przez mnie szereg narzędzi wspomagających pracę z systemami rozmytymi. Są to narzędzia do klasteryzacji (**ClusterIt**), interpretacji jej wyników (**fuzzyShape**) oraz automatycznego tłumaczenia kodu FML na C++ (**fml2cpp**).

3.1. Program ClusterIt

Program ClusterIt jest programem realizującym opisany w rozdziale II zmodyfikowany algorytm Gusafsona-Kessela.



Rys. III-4 Program ClusterIt

Program stworzony został w Visual C++. Przy realizacji programu ClusterIt napotkałem wiele problemów związanych z błędami zaokrągleń obliczeń pojawiających się w przypadku znajdowania klasterów o bardzo płaskich kształtach (poziomych, bądź pionowych), kiedy współczynniki macierzy definiującej normę były diametralnie różne (podczas gdy jedne dążyły do nieskończoności, drugie malały do zera). Sposobem zapobieżenia temu zjawisku było ograniczenie możliwości malenia i wzrostu wspomnianych współczynników poniżej oraz powyżej pewnego z góry założonego poziomu. Dzięki temu udało się opanować błędy związane z niedokładnym mnożeniem, kosztem ograniczenia nieco kształtów otrzymywanych klasterów poprzez wykluczenie tych najbardziej „płaskich”. W praktyce nie ma to jednak wpływu na jakość uzyskiwanych wyników.

Wejściem programu ClusterIt jest plik z danymi do klasteryzacji. Format pliku wejściowego jest następujący:

W pierwszej linii pojedyncza liczba n oznaczająca wymiarowość przestrzeni, z której pochodzą klasteryzowane dane.

W każdej z kolejnych m linii znajduje się n liczb oddzielonych przecinkiem opisujących kolejne współrzędne danej.

Podczas klasteryzacji w programie można ustalić szereg parametrów opisanych przy okazji opisu algorytmu Gustafsona-Kessela, takich jak ilość klasterów, objętość pojedynczego klastra, próg zakończenia obliczeń.

Po zakończeniu klasteryzacji program pozwala wyeksportować wyniki klasteryzacji do pliku z rozszerzeniem „res”.

Tak uzyskane wyniki mogą posłużyć za dane wejściowe dla programu **fuzzyShape**.

W przypadku, gdy chcemy wykorzystać wyniki programu ClusterIt jako wejście dla programu fuzzyShape n -wymiarowy wektor danych poddawanych klasteryzacji powinien zawierać $(n-1)$ wejść na pierwszych pozycjach oraz wartość wyjścia na pozycji ostatniej. Program fuzzyShape tak właśnie traktuje dane otrzymane na wejściu.

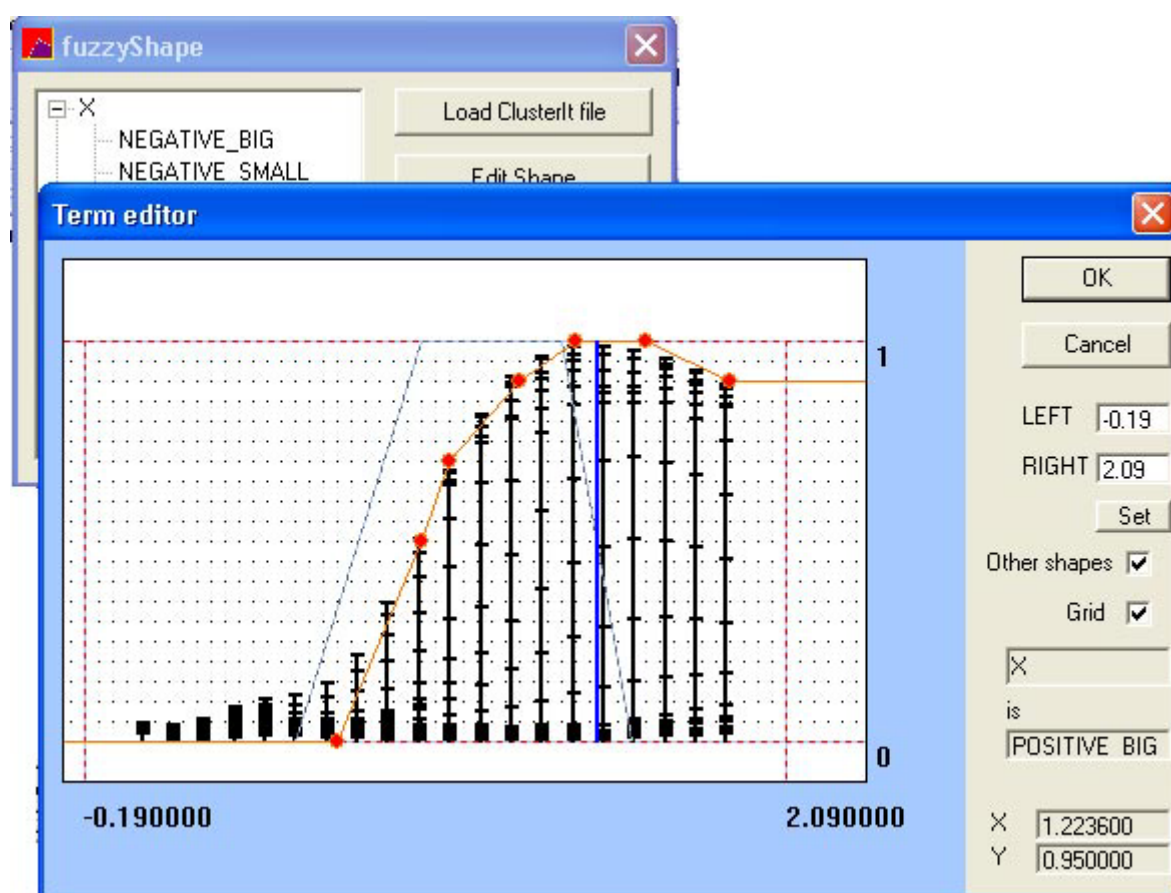
3.2. Program fuzzyShape.

Program fuzzyShape dostarcza interfejs użytkownika pozwalający w łatwy sposób narzucić interpretację wynikom działania ClusterIt.

Program ten generuje gotową do użycia w dokumencie FML definicję zbiorów rozmytych oraz reguły sterownika.

Wejściem dla programu fuzzyShape jest zestaw próbek pomiarowych wraz z ich stopniami przynależności wygenerowany przez ClusterIt.

Po wczytaniu poprawnego zestawu program generuje zmienne lingwistyczne oraz reguły sterownika na podstawie ilości klasterów i wymiarowości przestrzeni.



Rys. III-5 Program fuzzyShape.

Program traktuje podane mu dane tak, jakby było to $n-1$ wejść oraz 1 wyjście dla n -wymiarowej przestrzeni danych wejściowych. Wyjściem jest zawsze ostatnia współrzędna wektora danych. Dla każdego klastera uzyskanego w ClusterIt generowana jest jedna reguła sterownika (jak to opisano w rozdziale II). Użytkownik programu ma możliwość nadania kształtów zbiorom rozmytym reprezentującym odpowiednie termy zmiennych lingwistycznych. Po nadaniu kształtów wszystkich zbiorów oraz po odpowiednim nazwaniu termów możliwe jest wygenerowanie

gotowego kodu sterownika w języku FML. Program fuzzyShape nie służy jak widać do wizualnej kreacji sterowników, jest on raczej narzędziem umożliwiającym łatwą interpretację wyników klasteryzacji rozmytej.

Podczas dobierania kształtu terma wyświetlane na ekranie są zrzutowane na odpowiednią oś próbki oraz ich stopnie przynależności (reprezentowane są one pionowymi słupkami). Każdy słupek zakończony jest poziomą kreską, tak, aby w przypadku nałożenia się na siebie kilku próbek o jednakowej współrzędnej możliwe było odróżnienie ich stopni przynależności. Użytkownik aproksymując kształty zbiorów rozmytych może zastosować metodę opisaną na końcu poprzedniego rozdziału.

3.3. Program fml2cpp.

Narzędzie fml2cpp jest translatorem potrafiącym przekształcić dokument FML ze specyfikacją sterownika na klasę C++ używającą API opisanego w części początkowej tego rozdziału. Klasa wygenerowana przez fml2cpp jest gotowa do użycia w dowolnym programie Visual C++.

Program fml2cpp wymaga podania jako parametrów pliku FML, nazwy plików wynikowych oraz ścieżki do pliku fl.h dostarczonego z biblioteką fuzzylib. Ścieżka zostanie umieszczona w klauzuli #include w wygenerowanych plikach. Generowane pliki to plik nagłówkowy .h oraz plik .cpp definiujące klasę sterownika.

Program fml2cpp sprawdza poprawność pliku FML zarówno pod względem składni jak i sensu.

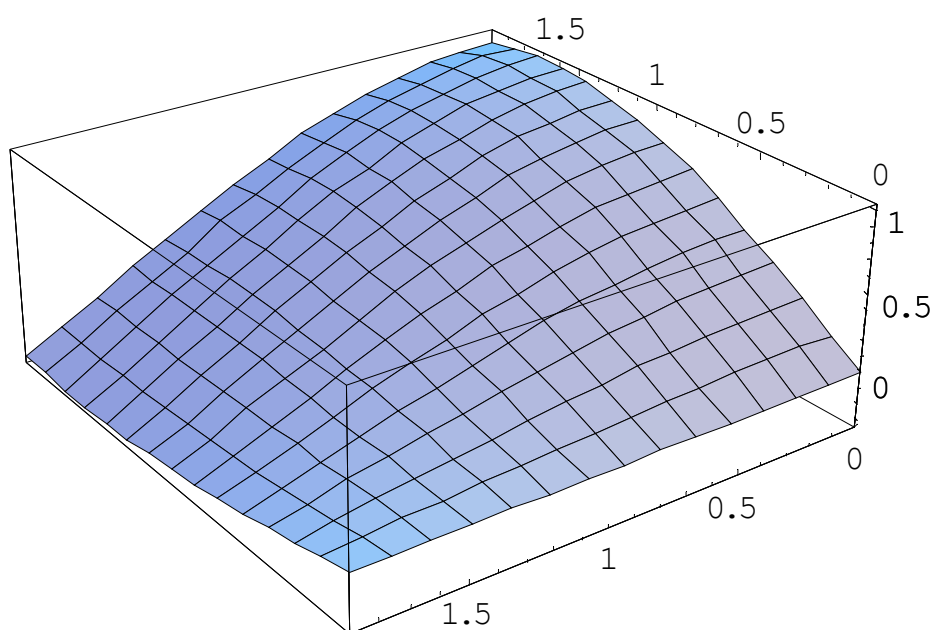
Użycie wygenerowanego sterownika w programie Visual C++ 6.0 używającym MFC :

- we właściwościach tworzonego projektu na zakładce linkera dodanie fuzzylib.lib w polu Object/library modules,
- dodanie do projektu wygenerowanych plików,
- włączenie poprzez #include wygenerowanego pliku nagłówkowego,
- zadeklarowanie zmiennej typu Fuzzy_nazwa_sterownika,
- użycie metody process z odpowiednimi parametrami.

IV. EKSPERYMENTY I WNIOSKI.

W celu ilustracji działania sterownika opartego na fuzzylib oraz zilustrowania działania algorytmów z rozdziału II zrealizowano następujące próby:

1. KLASTERYZACJA I MODELOWANIE FUNKCJI $F(x,y)=\sin(x)*\cos(y)$.



Rys. IV-1 Modelowana funkcja.

Funkcję $f(x,y) = \sin(x)\cos(y)$ poddano losowemu próbkowaniu na kwadratowym obszarze $[0,1.9] \times [0,1.9]$. Uzyskano 500 próbek pomiarowych.

Próbki poddano klasteryzacji z użyciem programu ClusterIt. W procesie wyróżniono 6 klasterów, stopień rozmytości dla zmodyfikowanego algorytmu Gustafsona-Kessela został ustawiony na 2.

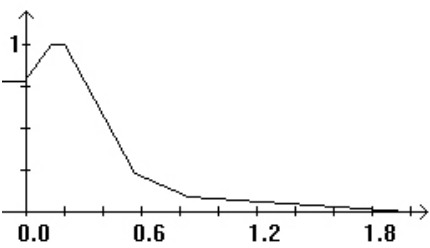
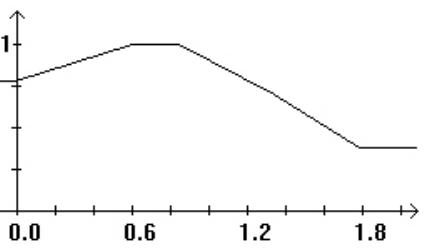
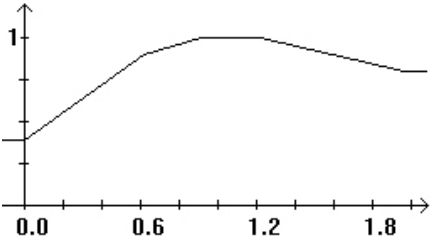
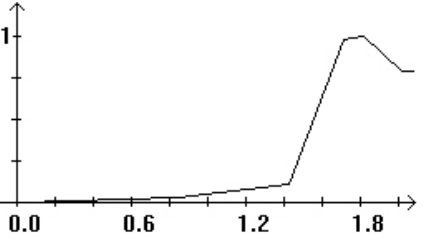
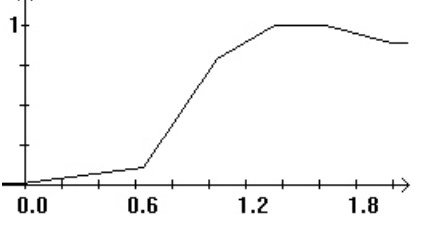
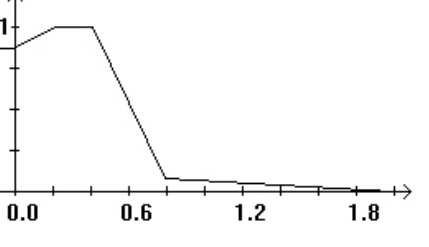
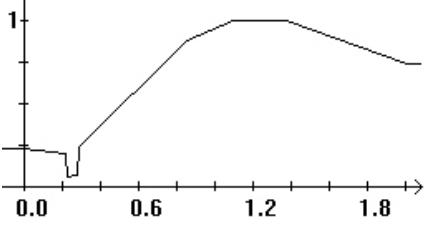
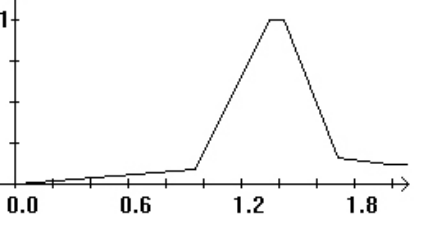
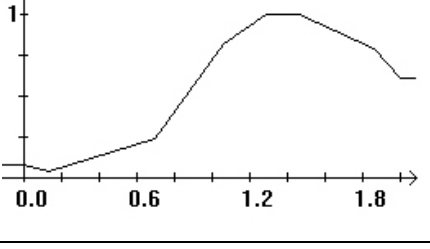
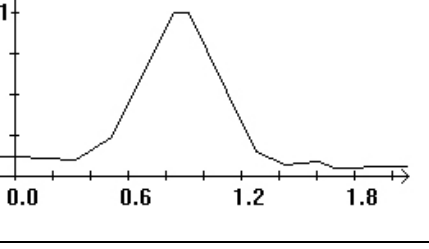
Wyodrębnione klasterzy poddano analizie programem fuzzyShape. Uzyskano w ten sposób 6 reguł modelujących zachowanie próbkowanej funkcji.

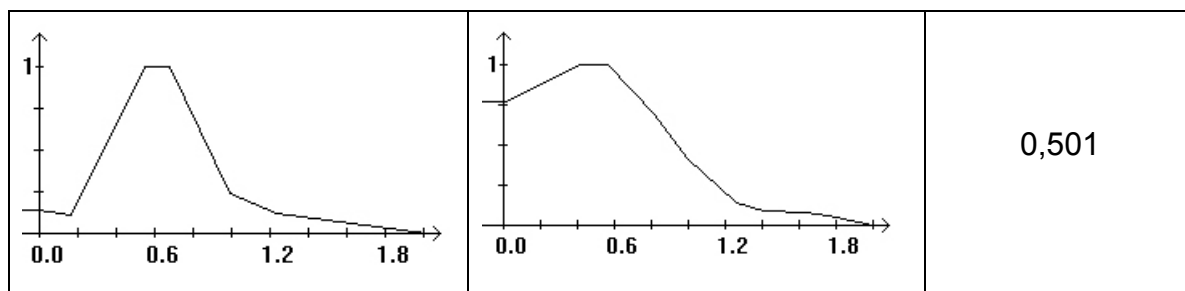
Najlepsze efekty uzyskano przy metodzie środka ciężkości użytej w wyostrzaniu. Zastosowano t-normę PROD jako operator AND dla reguł.

Przeprowadzono dwie próby.

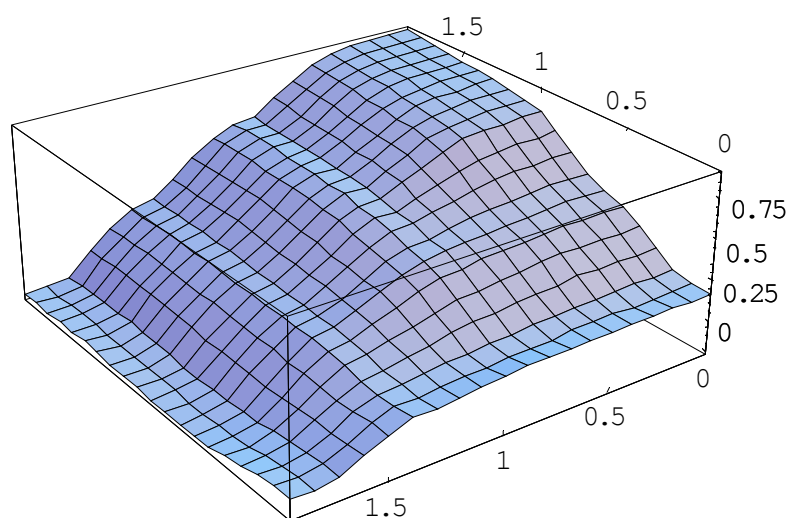
W pierwszej wynikowe zbiory rozmyte posiadały kształty sugerowane przez wynik klasteryzacji. W drugiej z kolei były to singletony (w przypadku fuzzylib symulowane przez zbiory trójkątne o minimalnym nośniku) w miejscach wyznaczonych przez środki ciężkości rzutów klastery. Drugi sposób dał nieco lepsze wyniki.

Uzyskane w procesie klasteryzacji reguły (w drugiej próbie) zaprezentowano w tabeli:

x	y	wyście (singleton)
		0,097
		-0,270
		0,938
		0,165
		0,602

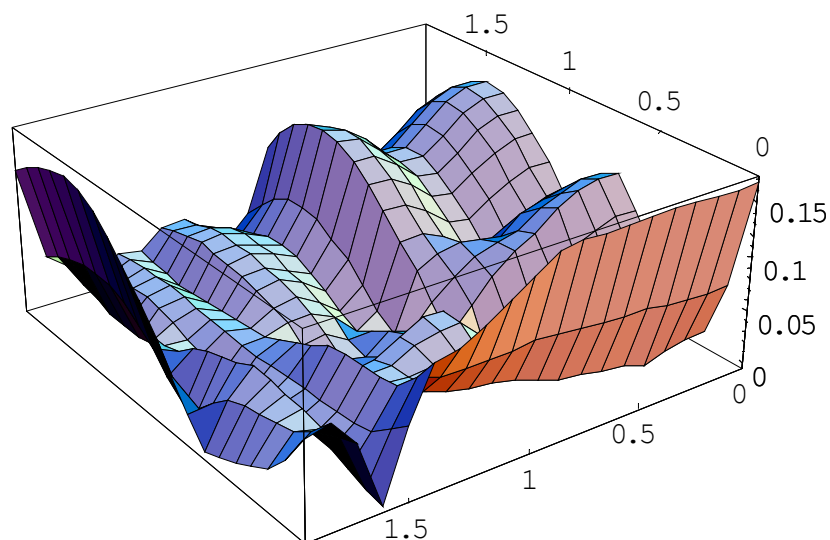


Model, który uzyskano stosując powyższe 6 reguł scharakteryzowany jest następującym wykresem:



Rys. IV-2 Wyjście sterownika modelującego $\sin(x)\cos(y)$.

Maksymalna wartość błędu modelu w stosunku do funkcji modelowanej nie przekracza 0,17. Rozkład błędu modelu w stosunku pokazano na rysunku poniżej:

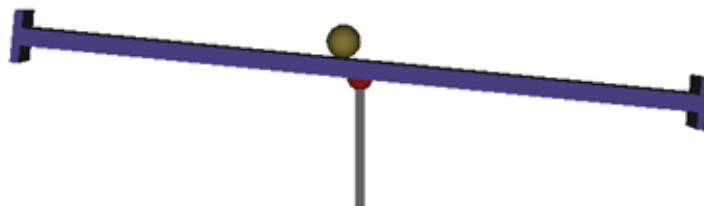


Rys. IV-3 Błąd modelowania.

Widać, że największe błędy pojawiają się na krawędziach modelowanej powierzchni, tam gdzie najczęściej działa istotnie pojedyncza reguła. Sposobem zapobieżenia temu zjawisku jest oddzielne modelowanie krawędzi. Innym powodem błędów może być niewielka ilość reguł. 6 reguł przy zastosowaniu klasycznej metodologii (pokryć cały obszar rozważań zbiorami na przykład trójkątnymi) pozwoliłoby nam podzielić zakres jednej ze zmiennych na 3, drugiej na 2 termy. Nie jest to wiele. Reguły uzyskane w wyniku klasteryzacji nie rozkładają się równomiernie, a mają tendencje do układania swych centrów się w „newralgicznych” punktach modelowanego systemu. Co może dawać lepsze efekty przy niewielkiej liczbie reguł. Nierównomierność może jednak być problemem, ponieważ niektóre z obszarów mogą zostać pokryte mniejszą ilością reguł niż inne (na przykład najbardziej oddalona od obserwatora ćwiartka na rysunku wyżej). Kolejnym powodem powstawania błędów może być losowy dobór próbek pomiarowych i ich nierównomierne rozłożenie w przestrzeni. Łatwo zauważyć, że istnieje wiele możliwości zwiększenia dokładności takiego modelu. Poczynając od zwiększenia liczby klasterów, a kończąc na lokalnym modelowaniu wycinków przestrzeni powodujących duży błąd poprzez klasteryzację próbek tylko z tych obszarów i tworzenie dodatkowych reguł dla nich właśnie.

2. PROBLEM BALANSOWANIA KULI NA POCHYLNI (ANG. BALL AND BEAM).

Dana jest podparta w centrum równia pochyła oraz poruszająca się po niej kula. Zadaniem systemu jest zbalansowanie kulki tak, aby zatrzymała się ona możliwie blisko centrum pochylni. System może zmieniać kąt nachylenia pochylni. System dokonuje próbkowania z częstotliwością 40Hz. Pomiarowi podlega aktualna prędkość kuli oraz aktualna odległość kuli od środka pochylni. Wynikiem działania sterownika jest pożądana wartość kąta nachylenia pochylni. System nie może zmienić podczas jednego taktu kąta nachylenia o więcej niż o $0,03\text{rad}$ ($\approx 1,72^\circ$). Daje to maksymalną prędkość kątową $1,2 \frac{\text{rad}}{\text{s}}$ ($\approx 68,79^\circ/\text{s}$). Długość pochylni to 10 m, zakres możliwych kątów nachylenia do poziomu od -45° do 45° . Siła tarcia na pochylni wynosi 0,02N. Masa kuli wynosi 0,2kg.



Rys. IV-4 Kula i pochylnia

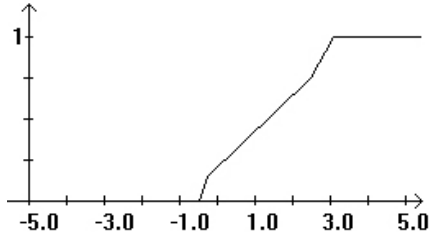
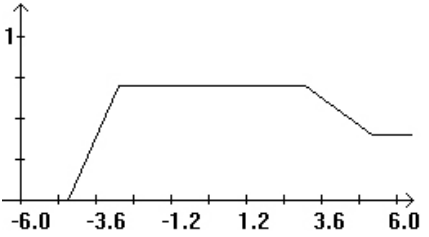
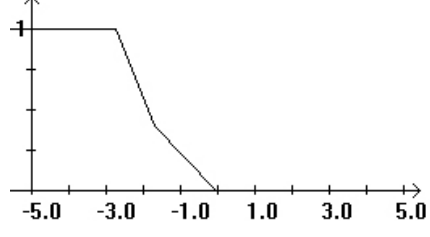
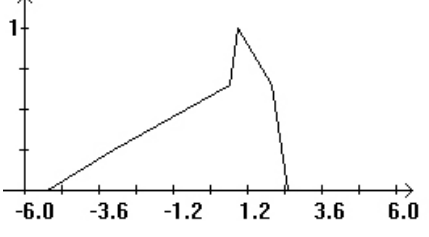
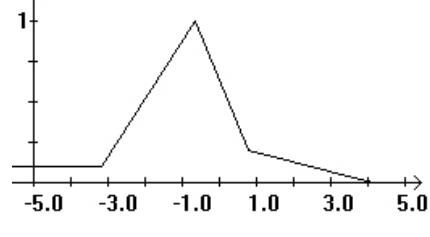
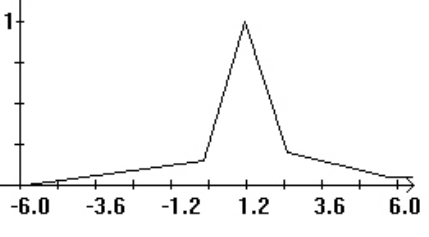
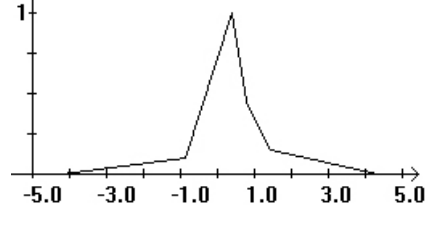
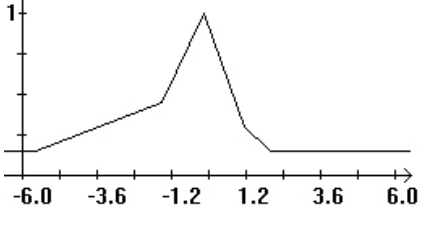
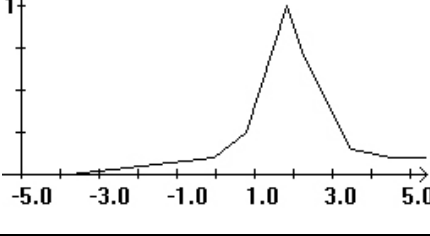
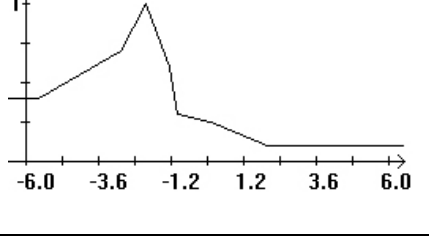
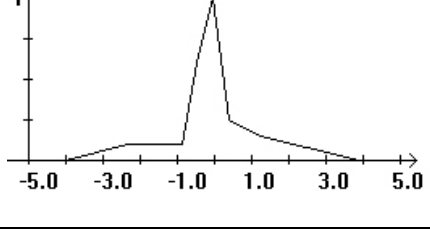
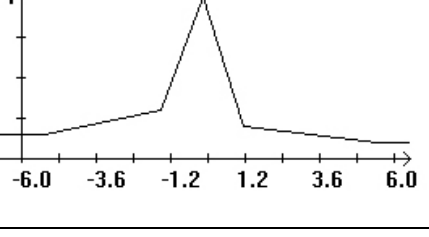
W celu przeprowadzenia modelowania i symulacji stworzony został program symulujący zachowanie pochylni.

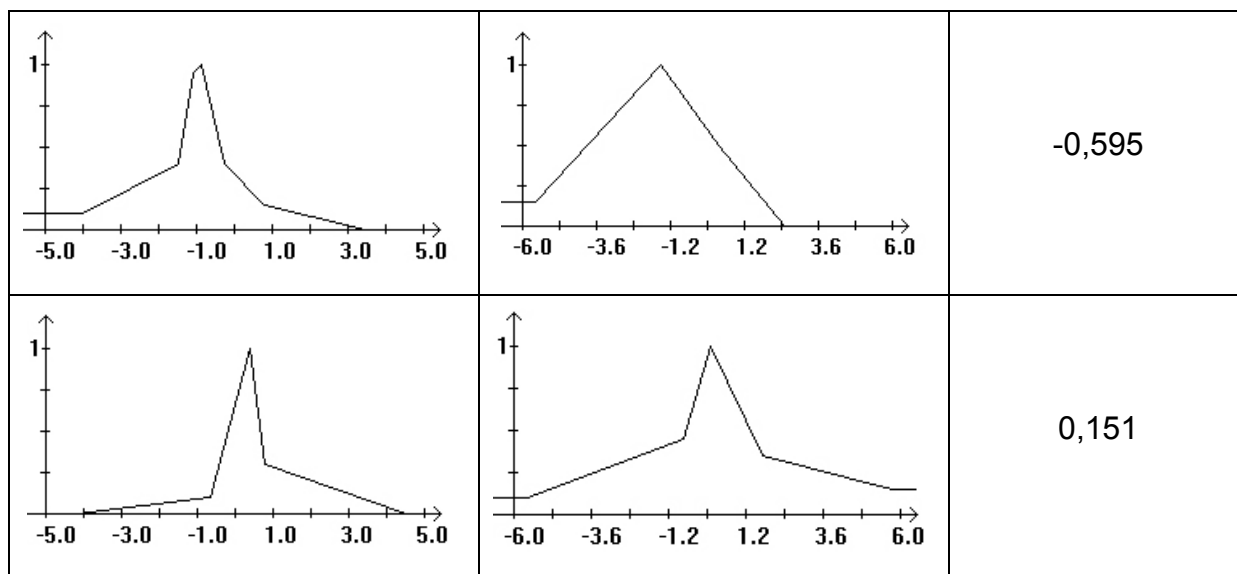
Dane do klasteryzacji pozyskano z obserwacji kilku przebiegów poprawnego balansowania wykonanych przez operatora. Należy zauważyć, że nie uproszczono systemu wykorzystując symetrię pochylni. Co za tym idzie system musiał zostać nauczony balansowania zarówno z lewej jak i prawej jej strony.

Klasteryzacji poddano 2822 próbki uzyskane z pomiarów działania operatora. Program ClusterIt miał za zadanie odnalezienie dziewięciu klasterów. Po odnalezieniu klasterów zostały poddane obróbce w celu uzyskania kształtów zbiorów rozmytych. Jako wartości wyjść poszczególnych reguł użyto (podobnie jak w przypadku modelowania funkcji rzeczywistej) singletonów.

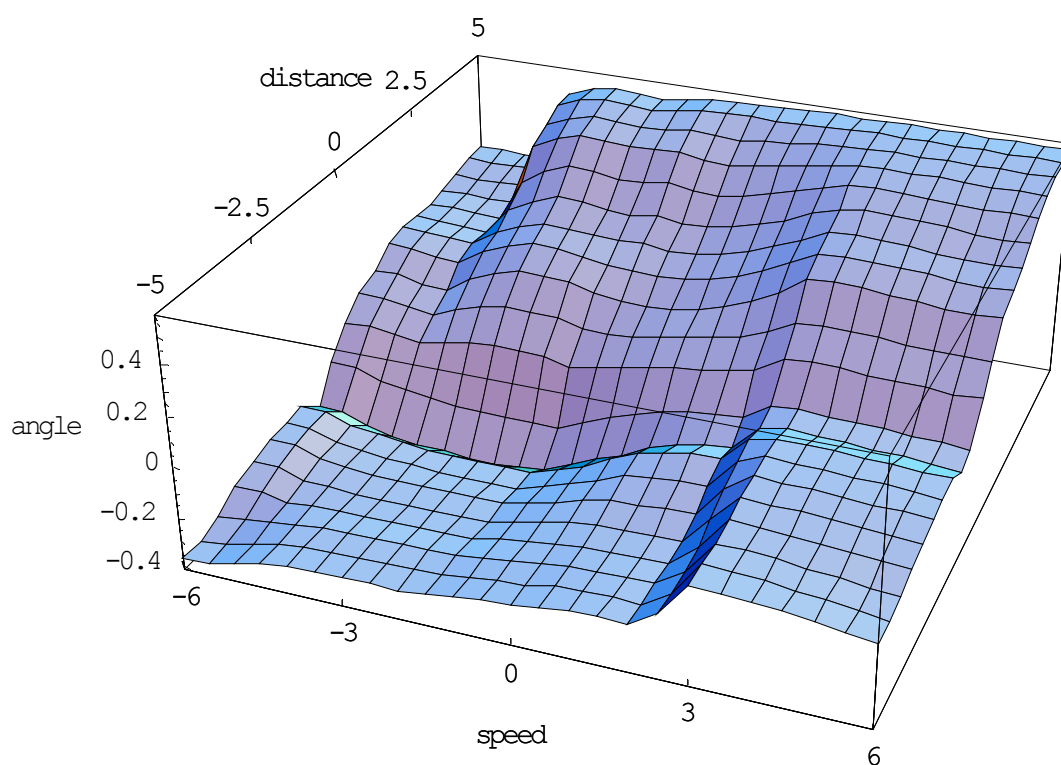
Poniższa tabela obrazuje zbiór reguł definiujących model:

<p>pozycja kuli</p> <p>(zero w środku pochylni, rośnie na prawo od środka, mierzona w metrach)</p>	<p>prędkość kuli</p> <p>(dodatnia, gdy kula przemieszcza się w prawo, mierzona w m/s)</p>	<p>wyjście (kąt)</p> <p>(mierzony w radianach, zero oznacza ustawienie poziome pochylni, rośnie gdy prawe ramie porusza się do góry)</p>
		<p>-0,141</p>

		0,572
		-0,238
		0,054
		0,021
		0,183
		-0,011



Docelowo uzyskano model scharakteryzowany następującym wykresem. Model oparty jest na dziewięciu regułach.



Rys. IV-5 Model kontrolujący pochylnię z kulą.

Na pierwszy rzut oka widać niesymetryczność powyższej funkcji. Wynika to z faktu, iż system nauczony został przez operatora nie biorąc pod uwagę symetrii pochylni. Dzięki temu możliwe jest nauczanie systemu w podobny sposób zachowań innych niż zatrzymywanie kuli na środku pochylni. Pozyskane reguły obrazują też różnice w

zachowaniu operatora gdy kula znajdowała się po lewej stronie w stosunku do zachowania z kulą po stronie prawej.

Tak stworzony model stanowi doskonałą podstawę do późniejszego jego dostrajania przy użyciu stosownych metod (na przykład rozmytych sieci neuronowych lub algorytmów genetycznych). W przypadku użycia algorytmów genetycznych miarą dopasowania osobnika w populacji mógłby być średni czas zbalansowania kuli na środku pochylni. Parametrami, które mogłyby być poddane strojeniu mogłyby być w najprostszym przypadku wagi reguł oraz singletony wyjściowe poszczególnych reguł.

3. PODSUMOWANIE.

Stworzone środowisko programistyczne w znacznym stopniu wspomaga implementację sterowników rozmytych i ich stosowanie we własnych rozwiązaniach programistycznych. Narzędzia wspomagające klasteryzację i przetwarzanie wyników znacznie przyspieszają czas tworzenia sterowników. Nie należy jednak zapominać iż, jak wynika z przeprowadzonych eksperymentów, sterowniki oparte tylko i wyłącznie na klasteryzacji są raczej pierwszą impresją końcowego modelu niż gotowym rozwiązaniem. Stanowią one dobrą podstawę do późniejszego dostrajania ich parametrów. Mimo tego posiadają ważną zaletę, którą jest stosunkowo mała liczba reguł (równa ilości klasterów).

Opisane narzędzia mogą zostać w przyszłości rozwinięte, tak aby tworzyły jeden spójny interfejs ułatwiający stosowanie logiki rozmytej w aplikacjach.

W ramach niniejszej pracy nie zostały rozwiązane wszelkie problemy związane z automatyczną kreacją sterowników, wiele zaproponowanych metod wciąż wymaga doprecyzowania czy też udoskonalenia.

Praca ta może stać się jednak pomocna przy rozwiązywaniu dużej ilości zadań z zakresu modelowania rozmytego. Opisane w niej metody są uniwersalne, a paleta ich zastosowań dość szeroka.

BIBLIOGRAFIA:

- [Babuška 01] – Robert Babuška; „Fuzzy and neural control. DISC Course Lecture Notes”; Delf University Of Technology 2001
- [Gustafson 79] – D. E. Gustafson, W. C. Kessel; „Fuzzy clustering with a fuzzy covariance matrix”; Protokoły konferencji IEEE CDC; San Diego 1979
- [IEC 97] INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC); „IEC 1131 - PROGRAMMABLE CONTROLLERS, Part 7 - Fuzzy Control Programming”; Committee Draft CD 1.0 1997
- [Maurin 73] – Franciszek Maurin; „Analiza, Część I, Elementy”; PWN, Warszawa 1973
- [Klawonn 97] – Frank Klawonn, Rudolf Kruse; „Constructing a fuzzy controller from data”; Fuzzy Sets And Systems 85; 1997.
- [Łachwa 01] – Andrzej Łachwa; „Rozmyty świat zbiorów, relacji i reguł”; Akademicka Oficyna Wydawnicza EXIT, Warszawa 2001
- [Piegat 99] – Andrzej Piegat; „Modelowanie i sterowanie rozmyte”; Akademicka Oficyna Wydawnicza EXIT, Warszawa 1999