Laura IVANCIU

Gabriel OLTEAN

SISTEME FUZZY

Îndrumător de laborator

FUZZY LOGIC SYSTEMS

Laboratory manual

UTPRESS
Cluj-Napoca, 2024
ISBN 978-606-737-711-8

Laura IVANCIU **Gabriel OLTEAN**

SISTEME FUZZY
Îndrumător de laborator

FUZZY LOGIC SYSTEMS
Laboratory manual

UTPRESS
Cluj-Napoca, 2024
ISBN 978-606-737-711-8

Editura UTPRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.: 0264-401.999
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Recenzia: Conf.dr.ing. Emilia Șipoș
Conf.dr.ing. Botond Kirei

Pregătire format electronic on-line: Gabriela Groza

Copyright © 2024 Editura UTPRESS
Reproducerea integrală sau parţială a textului sau ilustraţiilor din această carte este posibilă numai cu acordul prealabil scris al editurii UTPRESS.

ISBN 978-606-737-711-8

Cuprins

Prefață

Acest îndrumător de laborator se adresează în principal studenților din anul IV ai Facultății de Electronică, Telecomunicații și Tehnologia Informației, specializarea Electronică Aplicată (română și engleză). De asemenea, alte categorii studenți care încep să descopere universul logicii fuzzy pot utiliza acest material.

La începutul fiecărei lucrări, sunt prezentate obiectivele de învățare și noțiunile teoretice de care studenții au nevoie, pentru realizarea exercițiilor propuse. Toate lucrările se rezolvă utilizând mediul de dezvoltare MATLAB/Simulink, accesat online.

Nivelul de dificultate al lucrărilor crește gradual, de la aspecte generale despre mulțimi și sisteme fuzzy în primele lucrări, până la controlere fuzzy, respectiv modelare fuzzy pe baza datelor numerice, spre final. Se pune accent pe modurile de integrare a sistemelor fuzzy în aplicații practice, concrete, cum ar fi segmentarea imaginilor color, determinarea timpului de spălare al unei mașini automate, controlul temperaturii într-o incintă termică.

Prin rezolvarea tuturor exercițiilor propuse, studenții dobândesc abilitățile necesare pentru a crea, implementa și analiza sisteme cu logică fuzzy și a le integra în aplicații practice, concrete.

Cluj – Napoca, iunie 2024

Autorii

1. Introducere în Fuzzy Logic Toolbox

Obiectiv: familiarizarea cu funcțiile și interfețele grafice din Fuzzy Logic Toolbox, parte a MATLAB/Simulink

Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *SLF, FIS, mulțime fuzzy, grad de apartenență, suprafață de control*

○ Funcții și interfețe grafice

Mediul de dezvoltare MATLAB/Simulink pune la dispoziție 6 categorii de funcții și interfețe grafice, pentru lucrul cu mulțimi fuzzy (fuzzy sets), respective sisteme cu logică fuzzy SLF (fuzzy inference systems - FIS):

- *GUI editors* - editoare cu interfață grafică
- *Membership functions* - funcții de apartenență
- *Command line FIS functions* - funcții pentru crearea/modificarea/explorarea unui SLF de la linia de comandă
- *Advanced techniques* - tehnici avansate
- *Miscellaneous functions* - funcții diverse
- *GUI helper files* - fișiere auxiliare utilizate de editoarele cu interfață grafică

Pentru a vizualiza în MATLAB funcțiile și interfețele grafice disponibile, în linia de comandă se scrie:

```
help fuzzy
```

○ Utilizarea funcțiilor

Pentru a vedea modul în care poate fi utilizată fiecare funcție, se scrie în linia de comandă "*help* [*nume_functie*]". De exemplu, pentru a vedea cum se utilizează funcția "*trimf*", scrieți la linia de comandă:

```
help trimf
```

Exercițiul 1
Rulați în MATLAB exemplul oferit de help-ul functiei "*trimf*". Analizați semnificația fiecărei linii de cod.

Exercițiul 2
Reluați exercițiul 1 pentru o altă funcție din grupul "*Membership functions*". Analizați semnificația fiecărei linii de cod.

- ## Lansarea în execuție a editorului grafic pentru crearea unui SLF

  Pentru a lansa editorul grafic, la linia de comandă MATLAB se scrie:

```
fuzzyLogicDesigner
```

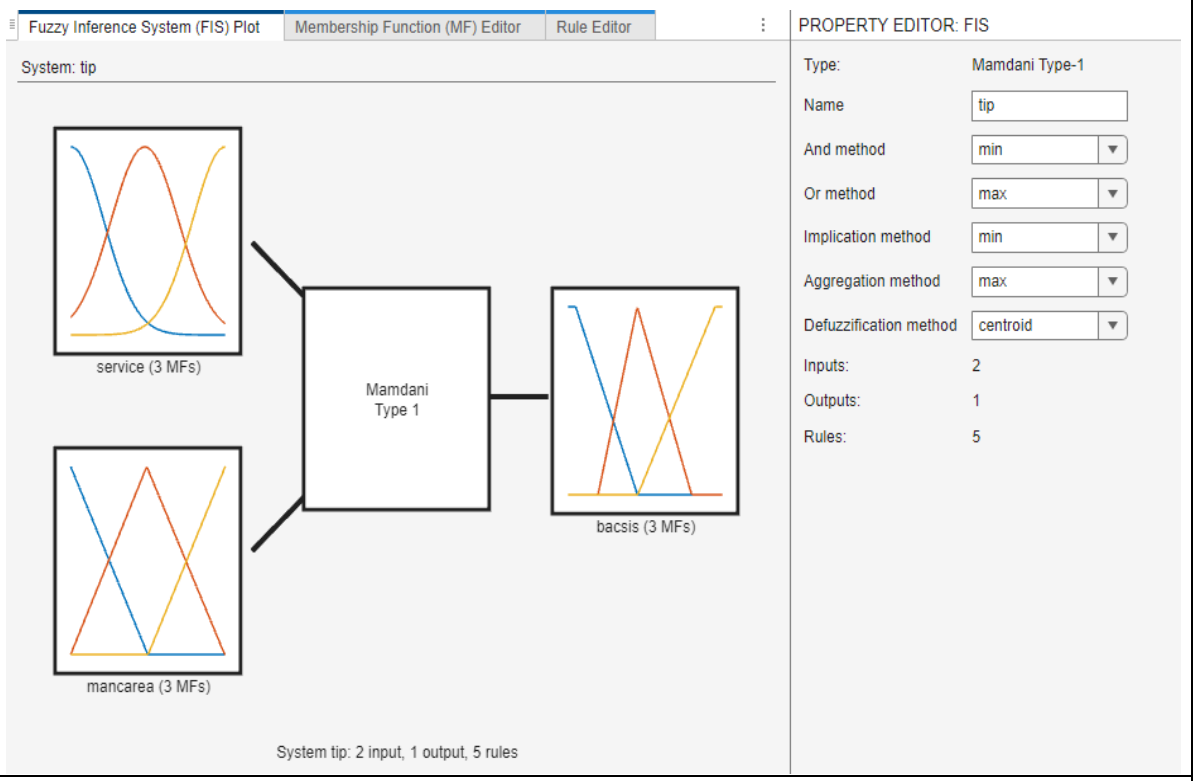
Exercițiul 3

Încărcați in editorul grafic un SLF creat anterior - SLF pentru determinarea mărimii bacșișului în cazul servirii unei mese la restaurant, disponibil la

https://drive.google.com/file/d/10Pvwc1vBSoRagxAzjKvE_mJi3D3V8Mip/view?usp=sharing

Descărcați arhiva "*tip.zip*" și plasați conținutul acesteia (cu *drag-and-drop*) în directorul curent în care lucrează MATLAB. Dezarhivarea se face cu dublu click pe fișier.

Din editorul grafic, alegeți: "*Import -> From file...*" și selectați fisierul "*tip.fis*".



Analizați proprietățile sistemului fuzzy.

Analizați funcțiile de apartenență definite pe fiecare dintre variabilele sistemului fuzzy.

Vizualizați suprafața de control.

Analizați modul de definire a regulilor.

Analizați activarea regulilor, din meniul *Rule Inference*. Care este impactul modificării conectivului într-o regulă?

- Funcții pentru crearea/modificarea/explorarea unui SLF din linia de comandă

Exercițiul 4

Încărcați în workspace fișierul "*tip.fis*" în variabila "*tip*" utilizând:

```
tip = readfis ('tip.fis');
```

Pentru a vizualiza proprietățile sistemului fuzzy:

```
getfis(tip)
```

Pentru a vizualiza diagrama sistemului fuzzy:

```
plotfis(tip)
```

Pentru a vizualiza suprafața de control a sistemului fuzzy:

```
gensurf(tip)
```

2. Mulțimi fuzzy. Operații cu mulțimi fuzzy aplicate în segmentarea imaginilor color.

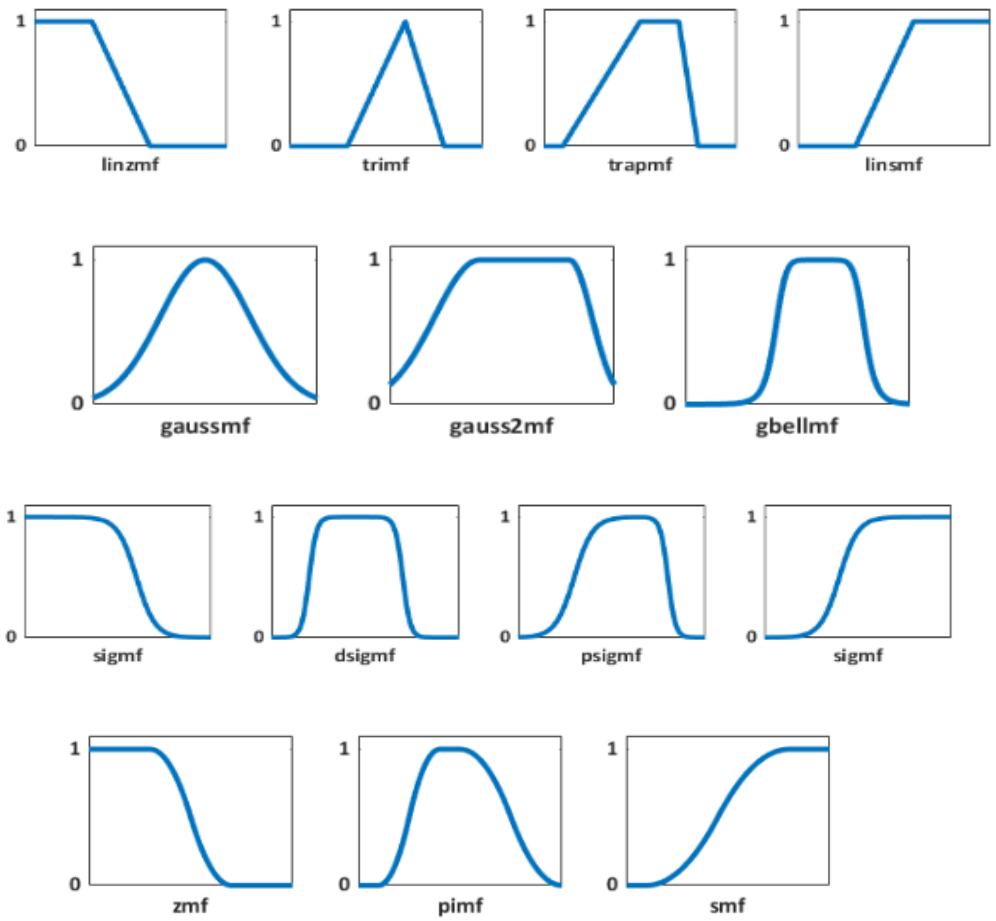
Obiective: utilizarea funcțiilor specifice pentru definirea mulțimilor fuzzy, înțelegerea operațiilor simple cu mulțimi fuzzy, familiarizarea cu modul de reprezentare a culorilor în MATLAB, înțelegerea operațiilor de bază pentru prelucrarea imaginilor și a conceptului de segmentare a unei imagini

Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *MF, parametri ai unei mulțimi fuzzy, reuniune, intersecție, complement, segmentare*

○ Galeria de mulțimi fuzzy. Utilizarea și editarea unei mulțimi fuzzy.

MATLAB pune la dispoziție 13 tipuri de mulțimi fuzzy (MF – membership function):



Detalii despre orice mulțime fuzzy, împreună cu o mică secvență de cod, se accesează cu "*help nume_funcție*", astfel:

```
help trapmf
```

Exercițiul 1

Rulați secvența următoare, ce are ca scop afișarea unei mulțimi fuzzy trapezoidale. Analizați fiecare linie de cod.

```
close all % inchide figura anterioara
clear all % sterge toate variabilele existente in workspace
clc       % sterge consola (fereastra de comenzi)
x = (0:0.1:10)'; % Universul discutiei este [0,10], in care se definesc
%puncte cu pasul de 0.1
params=[2 3 7 9];% parametrii multimii fuzzy trapezoidale
y = trapmf(x,params); % calculeaza valorile functiei de apartenenta
plot(x, y,'linewidth',2);
axis([0 10 -0.1 1.1]);
xlabel('universul discutiei');  % numele variabilei pe axa orizontala
ylabel ('grad de apartenenta'); % numnele variabilei pe axa verticala
set(gcf, 'name', 'Multime fuzzy trapezoidala', 'numbertitle', 'off'); %
%numele ferestrei
```

Exercițiul 2

Pe graficul afișat anterior, se adaugă reprezentarea gradului de apartenență al unui punct ales arbitrar la mulțimea fuzzy.

```
hold on
x1=2.75;
u1=evalmf(x1,params,'trapmf');
sprintf('Valoarea x1=%1.2f are gradul de apartenenta u1=%1.2f',x1,u1);
plot (x1,u1,'r*') % marcheaza punctul pe grafic
plot ([x1,x1],[0,u1],...
    'linestyle','-','color','r')
plot ([0,x1],[u1,u1],...
    'linestyle','-','color','r')
hold off
```

Exercițiul 3

Considerați variabila lingvistică "Viteza", cu universul discuției [0, 140] km/h. Definiți cinci valori lingvistice și reprezentați grafic cele cinci mf corespunzătoare, pe aceeași axă (folosiți "*hold on*"). Mulțimile fuzzy trebuie să formeze o partiție fuzzy în intervalul [0,140]. Afișați la linia de comandă valorile gradelor de apartenență la fiecare mulțime a punctelor 10 km/h, 52 km/h, 85 km/h și 100 km/h.

○ Operații cu mulțimi fuzzy

Reuniunea a două mulțimi fuzzy A și B este definită cu ajutorul funcțiilor de apartenență:

$$\mu_{AUB}(x) = \max\left(\mu_A(x), \mu_B(x)\right)$$

Intersecția a două mulțimi fuzzy A și B este definită astfel:

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

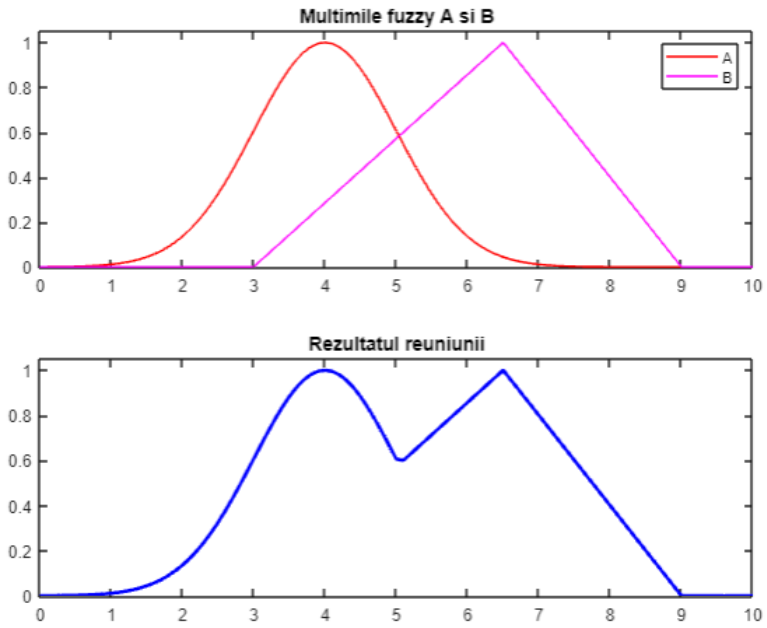
Complementul unei mulțimi fuzzy A este definit astfel:

$$\overline{\mu_A}(x) = 1 - \mu_A(x)$$

Exercițiul 4

Rulați secvența următoare, pentru afișarea reuniunii a două mulțimi fuzzy. Analizați fiecare linie de cod.

```
close all
x = (0:0.1:10)'; % Universul discutiei este [0,10], in care se definesc
%puncte cu pasul de 0.1
u1=gaussmf(x,[1,4]); % prima mf, gaussiana
u2=trimf(x,[3 6.5 9]);   % a doua mf, triunghiulara
u_reuniune=max(u1,u2);  % calculeaza gradele de apartenenta ale reuniunii
%prin operatorul "MAX"
hold on
subplot(2,1,1); % imparte fereastra de reprezentare a unei figuri in doua
%regiunii= verticale
% graficul se afiseaza in fereastra de sus
plot (x,u1,'r');hold on
plot (x,u2,'m'); hold off
axis([0 10 0 1.05]);
legend ('A','B');
title('Multimile fuzzy A si B')
subplot (2,1,2) % se va afisa in fereastra de jos
plot(x, u_reuniune,'color','b','linewidth',2)
axis([0 10 0 1.05]);
title('Rezultatul reuniunii')
set(gcf, 'name', 'Reuniunea multimilor fuzzy A si B - operatorul "max"',
'numbertitle', 'off'); % numele ferestrei
```



Exercițiul 5

Modificați secvența de cod anterioară, astfel încât să afișați într-o singură fereastră, pe grafice diferite, în ordine: mulțimile fuzzy A și B, rezultatul reuniunii, rezultatul intersecției, rezultatul complementului.

○ Reprezentarea culorilor în MATLAB. Spațiile RGB și HSV.

Spațiul **RGB** (red, green, blue) este definit de cele trei culori primare, roșu, verde și albastru, din care se pot obține, prin combinații aditive, toate celelalte culori ale spectrului. Valorile R, G și B sunt cuprinse în domeniul [0;1]. Pentru a reprezenta în MATLAB o culoare în spațiul RGB, se poate folosi numele (întreg sau prescurtat) al culorii, pentru culorile predefinite, sau un vector ce combină valorile componentelor primare.

Pentru a defini culoarea roz ca fond pentru fereastra curentă, scrieți în linia de comandă:

```
figure
hold on
set(gcf,'Color',[1,0.4,0.6]);
```

Exercițiul 6

Modificați culoarea de fond a ferestrei curente la cyan, respectiv verde, utilizând atât numele prescurtat al culorii, cât și reprezentarea vectorială.

În spațiul **HSV** (hue, saturation, value), culorile sunt reprezentate prin atributele perceptuale de nuanță (hue), saturație (saturation) și luminozitate sau valoare (value). Valorile H, S și V sunt reprezentate în domeniul [0;1]. În MATLAB, la deschiderea unei imagini color, reprezentarea sa implicită este în spațiul RGB. Conversia reprezentării în spațiul HSV se face cu funcția "r*gb2hsv*".

○ Transpunerea culorilor în mulțimi fuzzy

Diferitele valori, între 0 și 1, ale nuanței, H, definesc culorile importante din spectrul vizibil: roșu, orange, galben, verde, cyan, albastru, indigo (purpuriu), violet (magenta), roz și (la capătul domeniului), din nou roșu. Pentru reprezentarea culorilor ca mulțimi fuzzy, universul discuției este domeniul valorilor posibile pentru nuanță, variabila din universul discuției fiind nuanță H. Ca domeniu de valori, se va utiliza același domeniu cu cel utilizat în prelucrarea digitală a imaginilor, intervalul H=[0; 255] (reprezentare pe 8 biți). Pentru fiecare culoare, se definește câte o mulțime fuzzy peste X. Rezultă un total de 9 mulțimi fuzzy, cu valorile lingvistice: Red, Orange, Yellow, Green, Cyan, Blue, Purple, Magenta, Pink, definite peste H (intervalul [0; 255]), cu valori în intervalul [0; 1].

Exercițiul 7

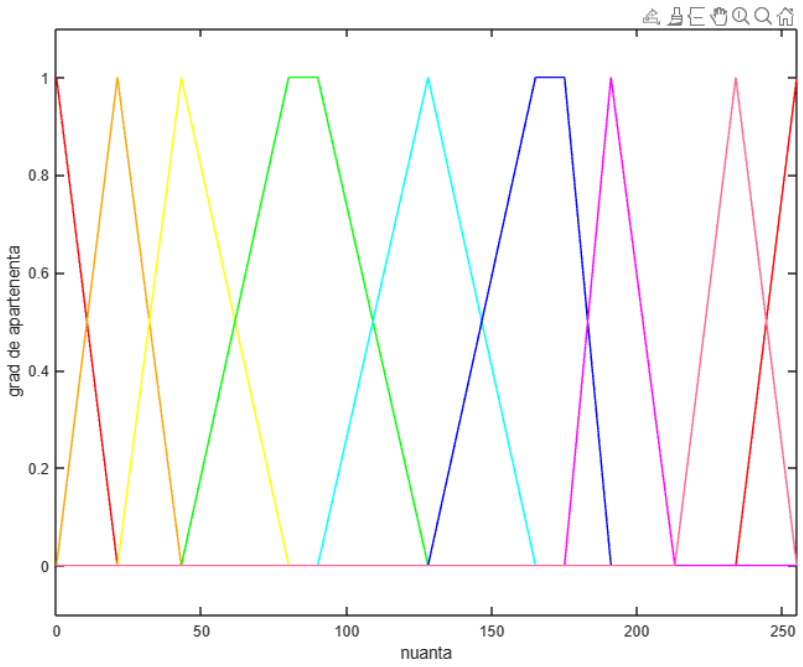
Rulați secvența de mai jos pentru a vizualiza reprezentarea cu mulțimi fuzzy a culorilor. Analizați fiecare linie de cod.

```
close all;
clear all;
clc;
% definirea multimilor fuzzy pentru nuanta:
H=(0:.5:255)'; %universul discutiei
Red_l=trimf(H, [0 0 21]); % rosu, la stanga
Red_r=trimf(H, [234 255 255]); %rosu, la dreapta
```

```
   Red=max(Red_l, Red_r);
   Orange=trimf(H, [0 21 43]); % portocaliu
   Yellow=trimf(H, [21 43 80]); % galben
   Green=trapmf(H, [43 80 90 128]); % verde
   Cyan=trimf(H, [90 128 165]); % turcoaz
   Blue=trapmf(H, [128 165 175 191]); % albastru
   Purple=trimf(H, [175 191 213]); % violet
   Pink=trimf(H, [213 234 255]); %roz

   plot(H, Red_l, 'color', 'r');
   axis ([0 255 -.1 1.1]);
   hold on;
   xlabel('nuanta');  % numele variabilei pe axa orizontala
   ylabel ('grad de apartenenta'); % numele variabilei pe axa verticala
   set(gcf,  'name',  'Multimi  fuzzy  pentru  reprezentarea  culorilor',
'numbertitle', 'off'); % numele ferestrei

   plot(H, Red_r, 'color', 'r');
   hold on;
   plot(H, Orange, 'color', [1 0.64 0]);
   hold on;
   plot(H, Yellow, 'color', 'y');
   hold on;
   plot(H, Green, 'color', 'g');
   hold on;
   plot(H, Cyan, 'color', 'c');
   hold on;
   plot(H, Blue, 'color', 'b');
   hold on;
   plot(H, Purple, 'color', 'm' );
   hold on;
   plot(H, Pink, 'color', [1,0.4,0.6]);
   hold off;
```



○ Segmentarea unei imagini color

Segmentarea unei imagini color presupune izolarea zonelor de o anumită culoare, față de celelalte zone. Pentru exemplificare, se va analiza o imagine din care se dorește separarea zonelor (obiectelor) de culoare dominant roșie, adică reprezentarea lor prin niveluri de gri cât mai apropiate de alb, spre deosebire de orice alte culori din imagine, care se vor reprezenta prin negru sau niveluri de gri cât mai apropiate de negru.

Pentru segmentare, se va forma mulțimea fuzzy *CloseToRed*, prin care se dorește capturarea zonelor din imagine care sunt nu doar roșu pur, ci și nuanțe apropiate de roșu. Mulțimea fuzzy *CloseToRed* este formată aplicând operațiile de bază (reuniune, intersecție, complement):

CloseToRed = *Red* SAU (*Orange* SI NU *Yellow*) SAU *Pink*

$$CloseToRed(H) = \max(Red(H), \min(Orange(H), 1 - Yellow(H)), Pink(H))$$

Exercițiul 8

Rulați secvența de mai jos pentru afișarea mulțimii *CloseToRed*. Analizați fiecare linie de cod.

```
figure;
NOT_Yellow=1-Yellow;
OrangeNotYellow=min(Orange, NOT_Yellow);
CloseRed=max(Red, OrangeNotYellow);
CloseToRed=max(CloseRed, Pink);
plot(H, CloseToRed, 'Linewidth', 2, 'color','r');
axis ([0 255 -.1 1.1]);
xlabel('nuanta');  % numele variabilei pe axa orizontala
ylabel ('grad de apartenenta'); % numnele variabilei pe axa verticala
set(gcf, 'name', 'Multimi fuzzy CuloareAproapeRosu', 'numbertitle',
'off'); % numele ferestrei
```

Cu cât obiectul sau zona din imagine are nuanța mai apropiată de roșu, cu atât gradul său de apartenență la mulțimea *CloseToRed* este mai mare. Reprezentarea vizuală a rezultatului obținut se face prin scalarea gradului de apartenență rezultat în domeniul [0; 255], pentru fiecare pixel. Imaginea segmentată va conține zone albe sau aproape albe în locul obiectelor roșii sau aproape roșii.

Exercițiul 9

Descărcați arhiva "*imagini.zip*" și plasați conținutul acesteia (cu *drag-and-drop*) în directorul curent în care lucrează MATLAB. Dezarhivarea se face cu dublu click pe fișier.

https://drive.google.com/file/d/1LvgcTts3SRZHJAH0gAOtzejvt-oUgdp1/view?usp=sharing

Rulați secvența de mai jos pentru segmentarea unei imagini color. Analizați efectul segmentării asupra fiecărei imagini din arhivă.

```
close all; clc; clear all
% imagini disponibile; la un moment dat, se va selecta doar una dintre ele
  img='Tiffany24.bmp';
% img='Cuticle.bmp';
% img='Lenna.bmp';
% img='Mouse.bmp';
% img='Peppers2.bmp';

OrigImg=imread(img); % citirea imaginii
```

```
   HSVimg=(rgb2hsv(OrigImg)); % conversie RGB->HSV
   Himg=(HSVimg(:,:,1));    % extragerea componentei nuanta Hue, cuprinsa in
%intervalul [0,1]
   H=(Himg*255);         % normalizare la [0, 255]

   % calculul gradului de apartenenta al nuantei pentru fiecare pixel, la
   % fiecare multime
   RedH_l=trimf(H, [0 0 21]);
   RedH_r=trimf(H, [234 255 255]);
   RedH=max(RedH_l, RedH_r);
   OrangeH=trimf(H, [0 21 43]);
   YellowH=trimf(H, [21 43 80]);
   GreenH=trapmf(H, [43 80 90 128]);
   CyanH=trimf(H, [90 128 165]);
   BlueH=trapmf(H, [128 165 175 191]);
   PurpleH=trimf(H, [175 191 213]);
   PinkH=trimf(H, [213 234 255]);

   % normalizare la [0, 255]
   RedObj=uint8(255*RedH);
   % compunerea multimii "CloseToRed"
   NOT_YellowH=1-YellowH;
   OrangeNotYellowH=min(OrangeH, NOT_YellowH);
   CloseRedH=max(RedH, OrangeNotYellowH);
   CloseToRedH=max(CloseRedH, PinkH);
   % afisare
   imshow(OrigImg); set(gcf, 'name', 'Original image'); % imagine originala
   figure
   imshow(RedObj); set(gcf, 'name', 'Red segments'); % evidentiere obiecte
%rosii
   figure
   imshow   (uint8(CloseToRedH*255));   set(gcf,   'name',   'Close   to   red
segments'); % evidentiere obiecte aproape rosii
```

Imagine originală

Imagine segmentată după roșu

Imagine segmentată după roșu și aproape roșu

Exercițiul 10

Creați un script care să definească o mulțime fuzzy pentru aproximarea unei alte nuanțe. Afișați mulțimea fuzzy rezultată. Realizați segmentarea după nuanța și după aproximarea nuanței alese.

Pentru testare, puteți utiliza imaginile din arhivă sau alte imagini reprezentative.

3. Simularea sistemelor cu logică fuzzy în MATLAB. Aplicație – mașina de spălat cu logică fuzzy.

Obiective: construirea unui sistem cu logică fuzzy utilizând Fuzzy Logic Toolbox, analizarea diferitelor metode pentru implicație și defuzzificare

Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *fuzzificare, defuzzificare, implicație, centroid, bisector, MOM, SOM, LOM*

○ Controlul proceselor prin sisteme cu logică fuzzy

Una dintre aplicațiile practice importante ale sistemelor cu logică fuzzy este utilizarea lor ca sisteme de control a proceselor. Controlul proceselor prin sisteme cu logică fuzzy are la bază un suport teoretic solid și este validat prin aplicații comerciale. Pentru exemplificare, se va prezenta o aplicație de control al timpului de spălare pentru o mașină de spălat.

Când se folosește o mașină de spălat, utilizatorul selectează de obicei durata de spălare, în funcție de cantitatea de haine și de tipul și gradul de murdărie al acestora. Pentru automatizarea procesului de spălare, se pot utiliza senzori de detecție pentru volumul hainelor, respectiv tipul și gradul de murdărie. Pe baza acestor date, se va alege un anumit timp de spălare.

Deoarece nu se poate defini o relație matematică precisă între mărimile de intrare (volumul hainelor, tipul și gradul de murdărie) și mărimea de ieșire (timpul de spălare). Astfel, timpul de spălare se setează manual, de către utilizator, pe baza experienței proprii și a încercărilor repetate.

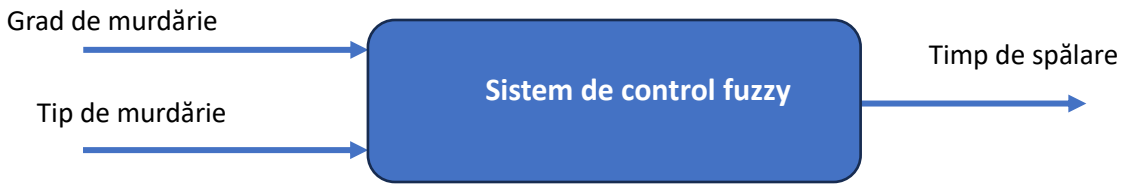
Realizarea unei mașini de spălat cu timp de spălare autodeterminat presupune construirea a două subsisteme:

- sistemul de senzori - furnizează mărimile de intrare ale mașinii de spălat, preluate din mediul exterior (hainele din mașină)
- unitatea de control - pe baza informațiilor primite de la sistemul de senzori, va lua decizia asupra timpului de spălare, sub forma unei ieșiri de comandă.

Exercițiul 1
Ce tip de senzori pot fi utilizați pentru determinarea volumului hainelor, gradului și tipului de murdărie, culorii, materialului hainelor?

○ Definirea SLF pentru controlul timpului de spălare

Se dorește proiectarea unui sistem de control cu logică fuzzy pentru o mașină de spălat, care să furnizeze timpul de spălare corect, în funcție de anumite informații despre hainele care trebuie spălate. Schema bloc a sistemului este:

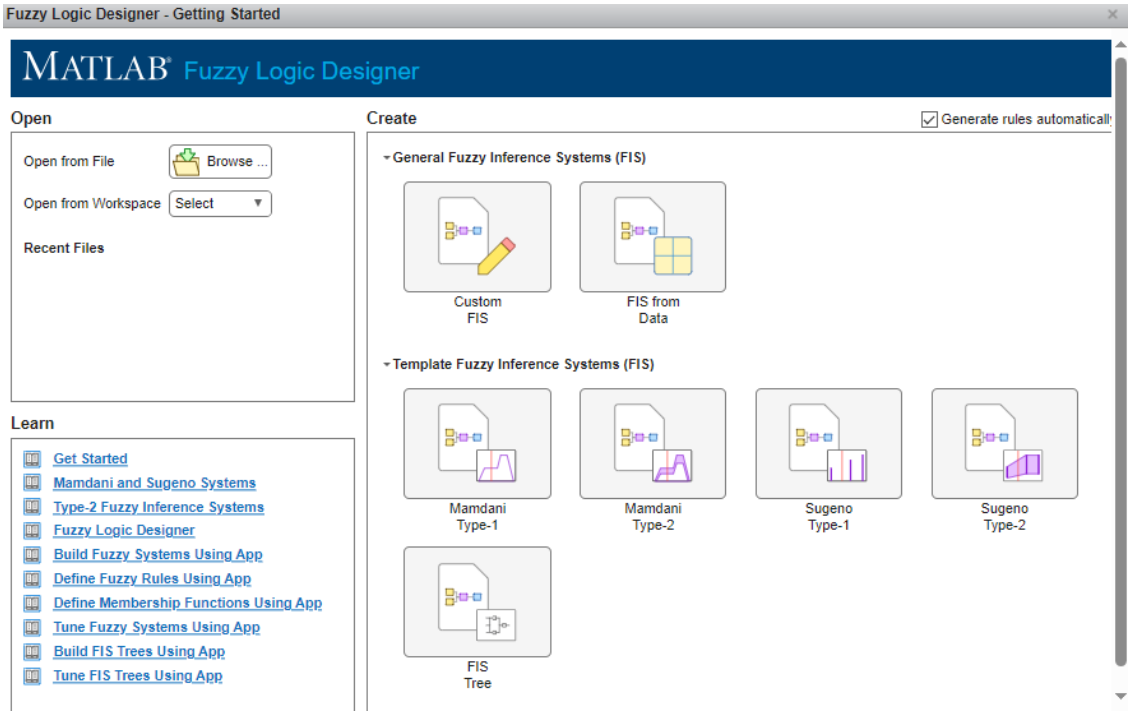


Sistemul are două intrări:

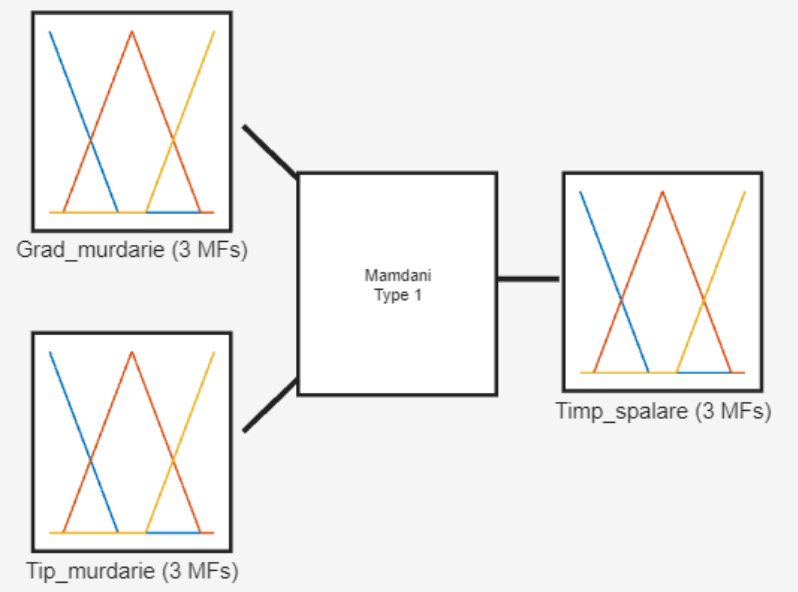
- gradul de murdărie - poate fi determinat din transparența apei
- tipul de murdărie - poate fi determinat din timpul necesar apei în care se înmoaie hainele să ajungă cu transparența la saturație (fără modificări vizibile ale transparenței). De exemplu, pentru hainele cu pete de grăsime, acest timp va fi mai lung, deoarece grăsimea este mai greu solubilă în apa decât alte tipuri de murdărie.

Sistemul cu logică fuzzy va fi construit utilizând *Fuzzy Logic Designer*, a cărui lansare în execuție se face scriind la linia de comandă:

```
FuzzyLogicDesigner
```



Se selectează opțiunea *Mamdani Type-1* (sistem fuzzy Mamdani predefinit) și se continuă cu particularizarea numelor pentru variabilele de intrare, respectiv variabila de ieșire.



Mulțimile fuzzy pentru cele 3 variabile au denumirile, tipul și parametrii din tabelul alăturat. Mulțimile definite peste variabilele de intrare formează o partiție fuzzy.

Variabilă	**Universul discuției**	**Denumire**	**Tip**	**Parametri**
Grad_murdarie	[0…100] %	*Mic* *Mediu* *Mare*	*trimf*	[-40 0 50] [0 50 100] [50 100 140]
Tip_murdarie	[0...100] %	*NeGras* *Mediu* *Gras*	*trimf*	[-40 0 50] [0 50 100] [50 100 140]
Timp_spalare	[0...60] minute	*FoarteScurt* *Scurt* *Mediu* *Lung* *FoarteLung*	*trimf*	[0 8 12] [8 12 20] [12 20 40] [20 40 60] [40 60 60]

Baza de reguli a sistemului este:

Grad_murdarie / *Tip_murdarie*	***Mic***	***Mediu***	***Mare***
NeGras	FoarteScurt	Scurt	Mediu
Mediu	Mediu	Mediu	Lung
Gras	Lung	Lung	FoarteLung

Toate regulile sunt alcătuite utilizând conectivul AND.

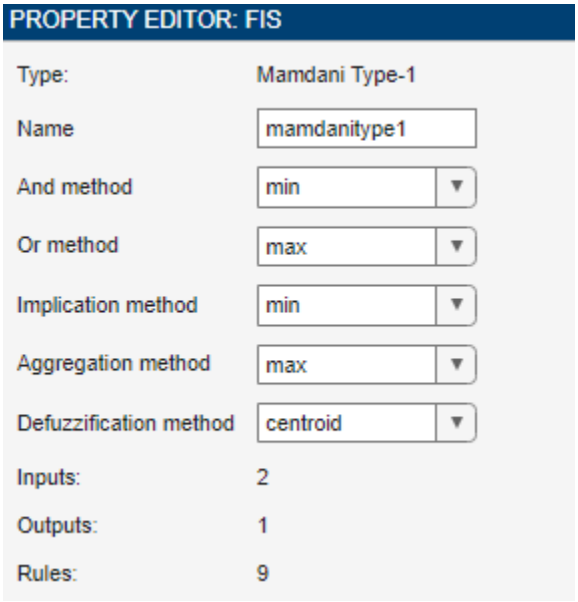
Exercițiul 2

Definiți mulțimile fuzzy pentru cele 3 variabile și baza de reguli, conform valorilor specificate anterior.

Operațiile sistemului cu logică fuzzy pentru controlul mașinii de spălat sunt:

- fuzzificare - mărimea de intrare se transformă în mulțime fuzzy *singleton*
- inferență - de tip *max-min* (Mamdani)
- defuzzificare - se va folosi defuzzificare de tip *centroid* (*COA - center of area*). Succesiunea răspunsurilor tranșante obținute prin acest tip de defuzzificare este suficient de lină pentru a asigura o suprafață de control fără variații bruște, cerință importantă a unui controller de proces. Formula după care se calculează valoarea tranșanta a ieșirii, pe baza mulțimii fuzzy agregată de ieșire, în cazul folosirii metodei de defuzzificare COA (centroid) este:

$$t_{0-continuu} = \frac{\int_{t=0}^{60} t * \mu_{MFO}(t)}{\int_{t=0}^{60} \mu_{MFO}(t)} \qquad t_{0-discret} = \frac{\sum_{t=0}^{60} t * \mu_{MFO}(t)}{\sum_{t=0}^{60} \mu_{MFO}(t)}$$



Exercițiul 3

Analizați funcționarea SLF vizualizând suprafața de control (Control surface) și operațiile realizate în sistemul cu logică fuzzy.

Exercițiul 4

Realizați un document în care să plasați suprafețele de control și valorile numerice obținute la ieșire, pentru 5 perechi de valori relevante ale variabilelor de intrare, în următoarele cazuri:

- `Implication: min; Defuzzification: centroid`
- `Implication: prod; Defuzzification: centroid`
- `Implication: min; Defuzzification: bisector`

- `Implication: min; Defuzzification: MOM`

De ce suprafața de control a unui sistem de control în general trebuie să fie cât mai netedă? Care metodă de defuzzificare este cea mai potrivită și care este cea mai defavorabilă pentru implementarea sistemului cu logică fuzzy de control din aplicația studiată?

4. Sisteme fuzzy de control. Controler de temperatură.

Obiective: înțelegerea conceptului de controler clasic, înțelegerea diferențelor dintre un controler clasic și un controler fuzzy, vizualizarea răspunsului unui controler fuzzy.

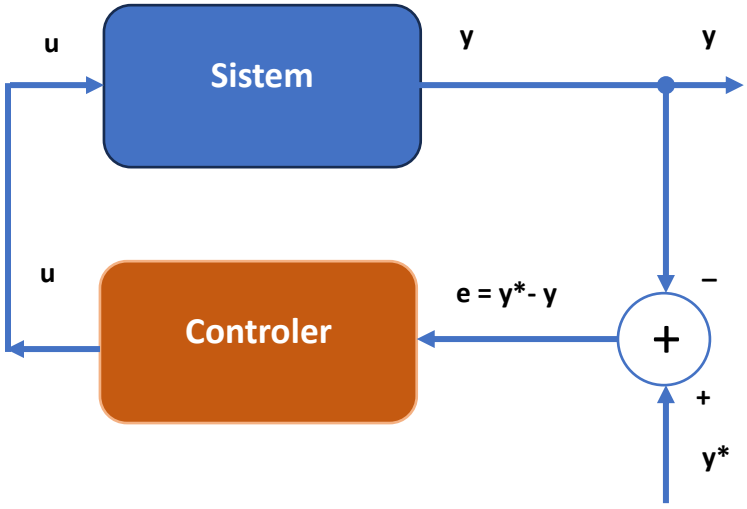
Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *controler clasic, controler fuzzy, PID.*

○ Sisteme de control clasice

Fie un sistem în buclă deschisă, având o intrare de comandă **u** și o ieșire **y**:

Scopul sistemului este să asigure o anumită valoare dorită la ieșire, **y***. Sistemul este astfel proiectat încât, în absența perturbațiilor și a variațiilor parametrilor sistemului, **y = y*** pentru o anumită intrare **u = u***. Lipsa perturbațiilor nu apare niciodată în practică, adică **y*** va fi diferit de **y**, dacă sistemul este lăsat să funcționeze în buclă deschisă, pentru intrarea de comandă **u = u***. Ca urmare, pentru a asigura **y = y*** în prezenta perturbațiilor, este nevoie să se modifice **u** față de **u***, astfel încât să se compenseze acțiunea perturbațiilor. Modificarea **u** depinde de modificarea **y** față de **y***, și este realizată de un alt sistem, conectat între ieșirea și intrarea sistemului inițial, numit ***controler***:



Noul sistem se numește *sistem în buclă închisă* , sau *sistem cu reacție*. Ieșirea **u** a controlerului reprezintă intrarea de comandă a sistemului și depinde, în general, de erorile la momentele de timp

anterioare: diferențele dintre ieșirea **y** și ieșirea dorită **y***, dar și de comenzile **u** la momentele de timp anterioare:

$$u(k) = f(e(k), e(k-1), \dots, e(k-t), u(k-1), \dots, u(k-t))$$

unde *f* definește legea de control, iar *t* este ordinul controlerului. Pentru t > 0, controlerul este cu *memorie*.

Eroarea e se definește ca:

$$e(k) = y^* - y$$

În general, legea de control *f* este neliniară. În teoria clasică a controlului, legea de control *f* este dedusă pe baza modelului matematic al procesului în buclă deschisă.

Legile clasice de control sunt:

a) legea de control proporțională (P):

$$u = K_p * e \Rightarrow u(k) = K_p * e(k)$$

b) legea de control integrală (I):

$$u = K_i * \int e\, dt$$

sau, în domeniul discret:

$$u(k) = K_i * \sum_{j=0}^{\tau} e(k-j)$$

c) legea de control derivativă (D):

$$u = K_d * \frac{d^\tau e}{dt^\tau}$$

d) combinații ale acestor legi - de exemplu ,controlerul PI:

$$u(k) = K_p * e(k) + K_i * \sum_{j=0}^{\tau} e(k-j)$$

○ Controlere fuzzy

Pentru un sistem fuzzy cu intrările **e(k), e(k-1),..., e(k-t), u(k-1),..., u(k-t)**, se poate formula o dependență lingvistică între ieșirea **u(k)** și aceste intrări. Controlere fuzzy utilizate în practică sunt de ordin t = 1, astfel:

$$u(k) = f(e(k), e(k-1), \dots, u(k-1))$$

Controlerele fuzzy tipice au o formă și mai simplificată, în sensul că nu țin cont de ieșirea anterioară **u(k-1)**; în regulile fuzzy sunt luate în considerare doar **e(k)** și **e(k-1)**, iar ieșirea controlerului este variația ieșirii **u**, definită ca:

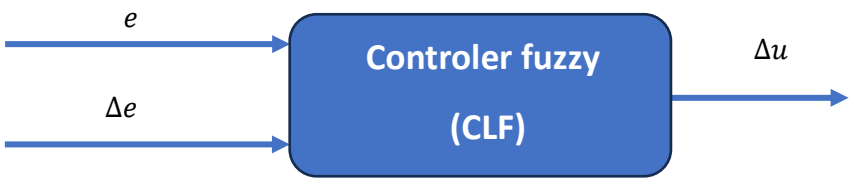
$$\Delta u(k) = u(k) - u(k-1) \Rightarrow u(k) = \Delta u(k) + u(k-1)$$

$$\Delta u(k) = F\big(e(k), e(k-1)\big)$$

unde **F** este funcția de transfer a sistemului fuzzy, determinată de:

- mulțimile fuzzy definite peste intrări și ieșire
- baza de reguli
- mecanismul de inferență
- metoda de defuzzificare utilizată.

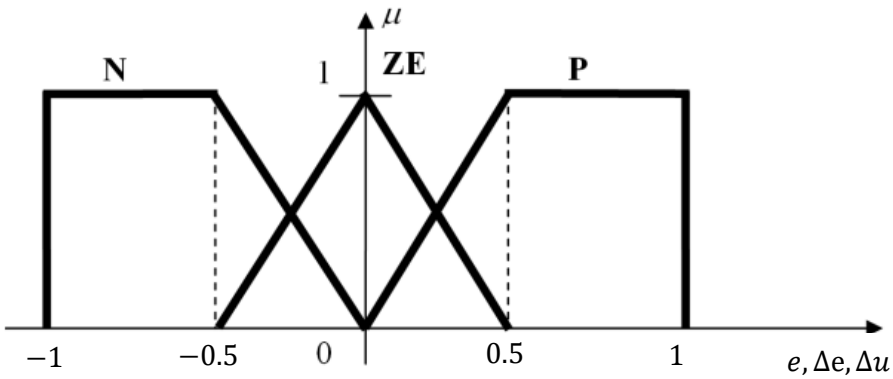
Tipic, intrările tranșante ale controllerului fuzzy sunt eroarea $e(k)$, derivata erorii $\Delta e(k) = e(k) - e(k-1)$, iar ieșirea este $\Delta u(k) = F\big(e(k), e(k-1)\big)$.



Acest tip de controller fuzzy de ordin 1 (t = 1) a fost propus în 1975 de către Mamdani și Assilian, și se numește CLF de tip Mamdani.

○ Controler fuzzy de tip Mamdani - exemplu

Cel mai simplu model de controler fuzzy Mamdani prevede trei mulțimi fuzzy (**Negativ N**, **Zero ZE**, **Pozitiv P**), identic definite pentru cele două intrări, respectiv pentru ieșire:



Baza de reguli se deduce ținând cont de obiectivul dorit, **y = y***, adică **e = y* - y = 0**. Privind nuanțat această valoare nulă a erorii, obiectivul dorit este ***"e este ZE"***. În plus, se consideră că ieșirea **y** a sistemului variază în același sens cu comanda **u**:

- dacă **u** crește, **y** crește;
- dacă **u** este constant, **y** este constant;
- dacă **u** scade, **y** scade;

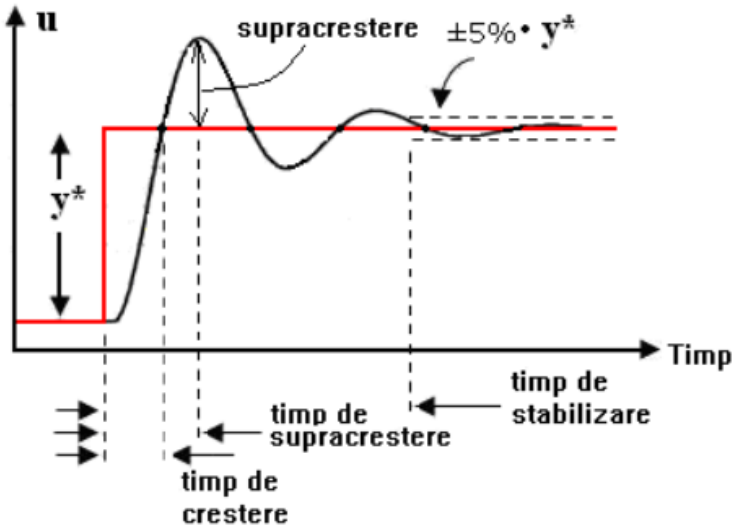
Baza de reguli conține toate combinațiile posibile ale valorilor lingvistice ale intrărilor (bază de reguli completă):

Δe \ e	***N***	***ZE***	***P***
N	N	N	ZE
ZE	N	ZE	P
P	ZE	P	P

Mecanismul de inferență utilizat este Mamdani, adică inferență max-min, iar metoda de defuzzificare este COA (centroid).

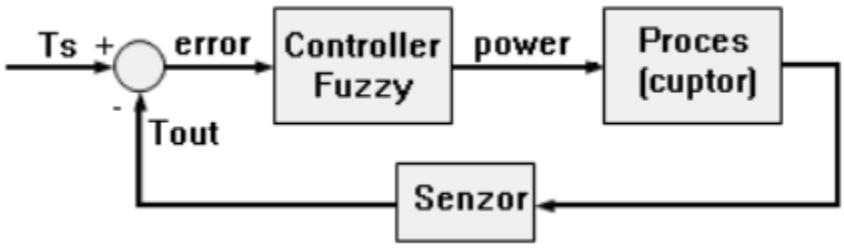
Răspunsul tipic al unui sistem automat de reglare la un semnal treaptă poate fi caracterizat de o serie de parametri:

- timpi de creștere/supracreștere/stabilizare
- supracreștere (mărimea primei supracreșteri peste valoarea stabilită).



○ Controler fuzzy de temperatură

Sistemul de control al temperaturii se compune dintr-un proces (cuptorul) și controlerul fuzzy:

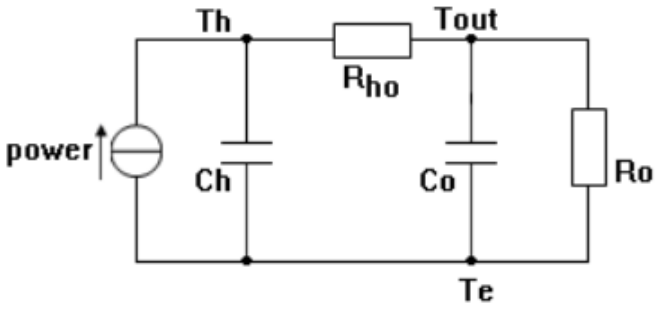


unde:

- *Ts* – temperatura dorită în incintă (*Set Point Temperature*)
- *Tout* – temperatura măsurată (temperatura provenită de la senzorul din incinta cuptorului)

- *error* – eroarea (*error* = *Ts* – *Tout*)
- *power* - puterea necesară încălzirii/răcirii (ieșirea controlerului de temperatură, respectiv intrarea de comandă în proces).

Rolul controlerului este de a menține temperaturii din cuptor egală cu temperatura dorită (*Ts*). Pentru modelarea generării și transferului de căldură, este folosit circuitul echivalent din figură:



Sursa de curent *power* (putere termică) reprezintă puterea furnizată elementului de încălzire/răcire. Sistemul cuprinde un element de încălzire/răcire electric, având capacitatea Ch=500[J/°C] conectat printr-o rezistență Rho=0.143[°C/W] la un cuptor cu capacitatea de încălzire Co=1000[J/°C]. Cuptorul degajă căldură în mediul exterior având temperatura Te, prin rezistența termică de transfer Ro=0.1[°C/W]. Controllerul de temperatură ajustează puterea disipată elementului de încălzire *power*, comparând temperatura cuptorului Tout cu temperatura de referință Ts.

Descărcați arhiva "*TempControl.zip*" și plasați conținutul acesteia (cu *drag-and-drop*) în directorul curent în care lucrează MATLAB. Dezarhivarea se face cu dublu click pe fișier.

https://drive.google.com/file/d/1c_XcAD8KaaSrx37ixkdDm-1x_JAHES0n/view?usp=sharing

Schema bloc Simulink este prezentată în figură. Pentru a asigura "generalitatea" controlerului fuzzy, au fost utilizate două blocuri de conversie liniară la intrare (*Scale error*, *Scale delta_error*), și un bloc de conversie la ieșire (*Scale_power*). Pentru limitarea în interval [-1, 1] a valorilor furnizate la intrarea controlerului, se utilizează blocurile de limitare dublă (*Saturation*, *Saturation1*).

Exercițiul 1

Implementați sistemul fuzzy Mamdani de control a temperaturii, conform specificațiilor definite anterior. Vizualizați suprafața de control. Salvați sistemul fuzzy creat cu numele *„TempControlM.fis".* Citiți sistemul fuzzy creat într-o variabilă denumită *fls*, utilizând instrucțiunea *readfis*.

Exercițiul 2

Încărcați modelul Simulink al sistemului de control a temperaturii, "*TempControlM.mdl*". Inițializați parametri cuptorului, prin rularea script-ului *„HeaterOven_params.m"*.

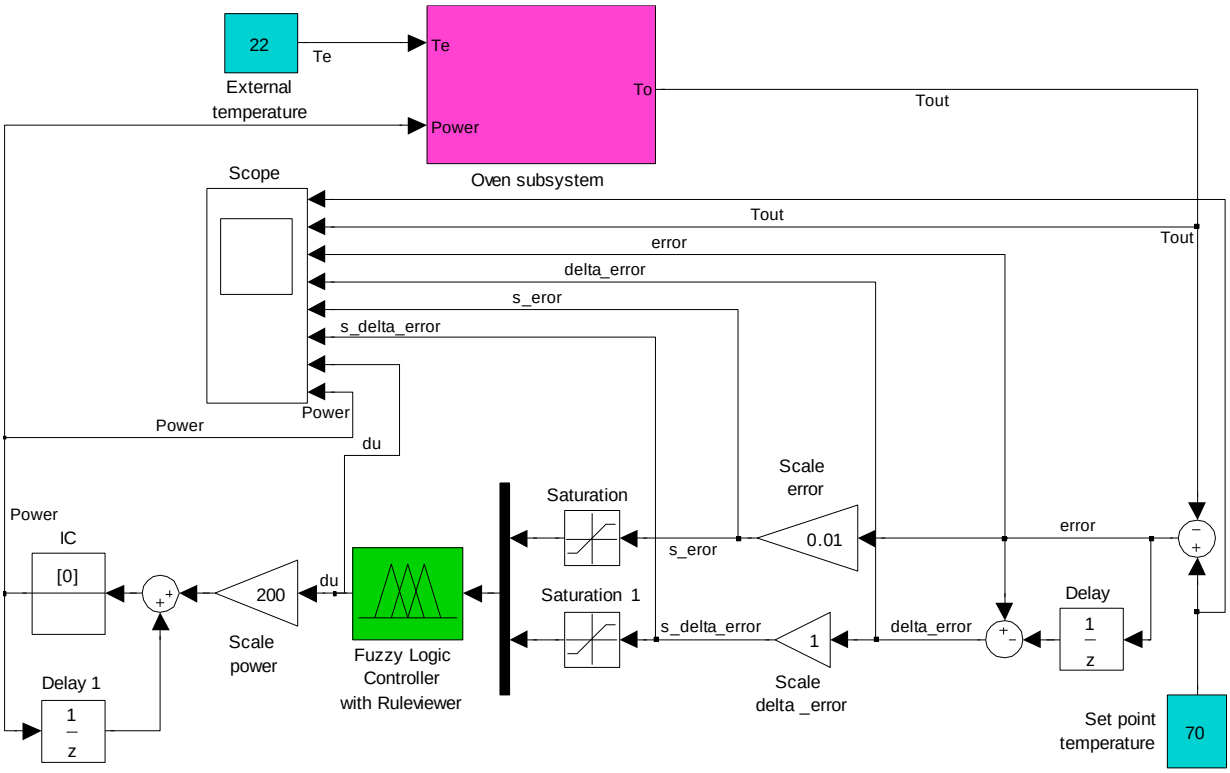
Porniți simularea și activați vizualizarea formelor de undă pe osciloscop (dublu-click pe blocul *Scope*). Măsurați parametrii sistemului de control: timpul de creștere, timpul de stabilizare, supracreșterea. Notați valorile obținute.

Exercițiul 3

Modificați factorii de scalare pentru intrări și ieșire, astfel încât performanțele sistemului să fie îmbunătățite.

Exercițiul 4

Reluați exercițiul 2, în ipoteza utilizării unui controler fuzzy de tip Takagi-Sugeno. Conversia între Mamdani și Takagi-Sugeno se face automat, cu opțiunea *Mamdani to Sugeno*. În acest caz, modelul Simulink utilizat va fi "*TempControlTS.mdl*". Care dintre cele două tipuri de sisteme de control are performanțe mai bune, pentru valorile inițiale ale factorilor de scalare?



5. Aproximarea caracteristicii curent-tensiune a diodei semiconductoare utilizând un SLF

Obiective: înțelegerea motivației utilizării SLF ca aproximatori universali, experimentarea legăturii între mulțimile fuzzy de intrare și de ieșire și baza de reguli, înțelegerea operațiilor de fuzzificare/inferență/defuzzificare, realizarea unui SLF complet pentru aproximarea unei funcții date.

Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *aproximare de funcții, modelare fuzzy.*

○ Sisteme cu logică fuzzy ca aproximatori universali

Una dintre aplicațiile importante ale sistemelor cu logică fuzzy este folosirea lor ca aproximatori de funcții. Fiind dată o funcție grafică **y = f(x)**, care poate fi reprezentată doar aproximativ sub formă analitică, se poate proiecta un sistem cu logică fuzzy având **x** ca intrare și **y** ca ieșire, astfel încât între valoarea estimată a lui **y**, **y_est**, și **f(x)**, există o diferență oricât de mică se dorește, notată **e**.

$$|y\_est\ - f(x)|\ < e$$

Cu alte cuvinte, sistemul cu logică fuzzy cu intrarea **x** și ieșirea **y** este un ***aproximator universal de funcții*** grafice **y = f(x)**.

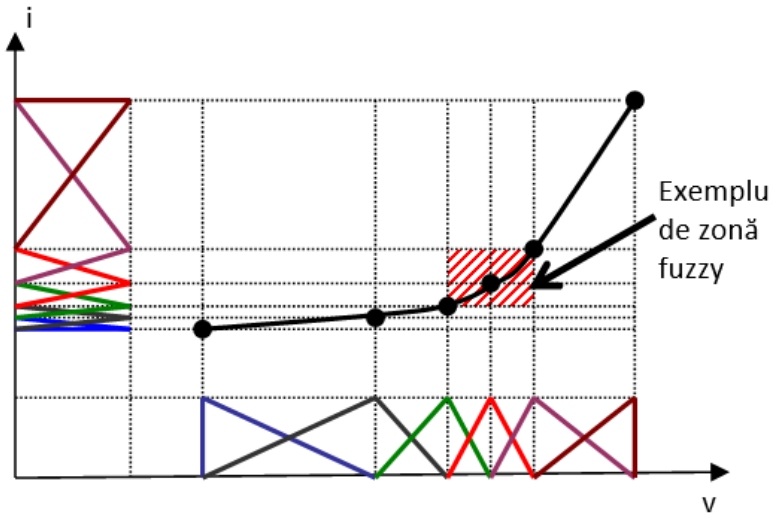
○ Modelarea fuzzy a caracteristicii i-v a diodei semiconductoare

Se dorește aproximarea regiunii directe a caracteristicii statice curent-tensiune (i-v) a unei diode semiconductoare, pe baza curbei i-v a componentei determinate experimental (sau luate din catalog). Sistemul cu logică fuzzy este descris astfel:

- **variabila de intrare** - valorile tensiunii pe diodă la polarizare directă, v = v_AK, în domeniul [0 mV, 860 mV]
- **variabila de ieșire** - valoarea curentului anodic prin diodă, i = i_A, în domeniul [0 mA, 450 mA]

SLF astfel definit va caracteriza complet comportamentul electric al diodei în polarizare directă, pentru v = v_AK, în domeniul [0 mV, 860 mV], ignorând variațiile cu temperatura, umiditatea și abaterile de proces tehnologic de fabricație a diodei. Deci, sistemul cu logica fuzzy va putea fi utilizat ca un model electric al diodei.

Principiul modelării caracteristicii i-v a diodei se bazează pe acoperirea graficului neliniar de variație a curentului în funcție de tensiune prin “zone fuzzy”:



Fiecare "zonă fuzzy" este descrisă prin:

- o mulțime fuzzy peste universul discuției variabilei de intrare v, notată *MFVk*
- o mulțime fuzzy corespunzătoare peste universul discuției variabilei de ieșire i, notată *MFIk*
- o regulă fuzzy Rk, care indică faptul că între cele două mulțimi fuzzy *MFVk* și *MFIk* există o relație fuzzy care descrie "zona fuzzy"
  Rk: Dacă v este *MFVk*, atunci i este *MFIk*

Observație: Pentru zonele cu neliniarități pronunțate ale graficului, sunt necesare mai multe mulțimi fuzzy de intrare și respectiv de ieșire.

○ SLF pentru modelarea caracteristicii i-v a diodei semiconductoare

SLF pentru modelarea caracteristicii i-v a diodei semiconductoare este compus din:

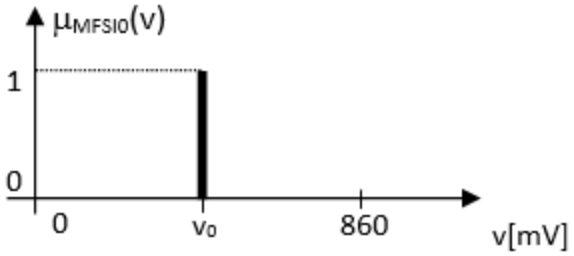
- mulțimile fuzzy definite peste universul discuției variabilei de intrare v, notate *MFVk*
- mulțimile fuzzy definite peste universul discuției variabilei de ieșire i, notate *MFIk*
- baza de reguli fuzzy, ce conține *k* reguli de forma: Rk: Dacă v este *MFVk*, atunci i este *MFIk*

Componentele SLF și operațiile prin care o valoare tranșantă a variabilei de intrare (v = v_0 - tensiunea pe diodă) este prelucrată, astfel încât la ieșire să se furnizeze o valoare tranșantă a variabilei de ieșire (i = i_0 - curentul prin diodă) sunt ilustrate în figură.

Operațiile unui SLF sunt:

- **fuzzificare** - transformarea mărimii tranșante de intrare într-o mulțime fuzzy singleton, cu funcția de apartenență:

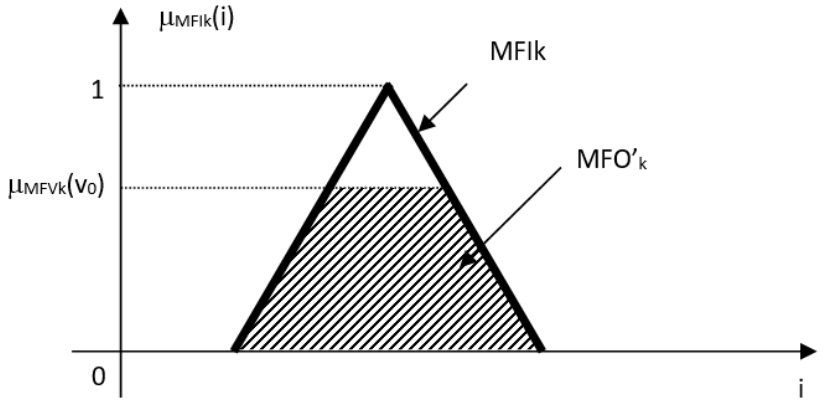
$$\mu_{MSFI_0}(v) = \begin{cases} 1, v = v_0 \\ 0, v \neq v_0 \end{cases}$$



- **inferență** - pe baza valorii fuzzy de intrare și a fiecărei reguli Rk, deduce rezultatul Yk a regulii Rk, folosind implicație de tip *min* (Mamdani):

$$\mu_{MFO'_k}(i) = \min\left(\mu_{MFV_k}(v_0), \mu_{MFI_k}(i)\right)$$

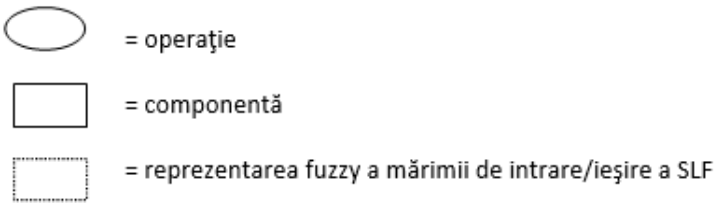
Mulțimea fuzzy de ieșire a regulii Rk (concluzie parțială) este:

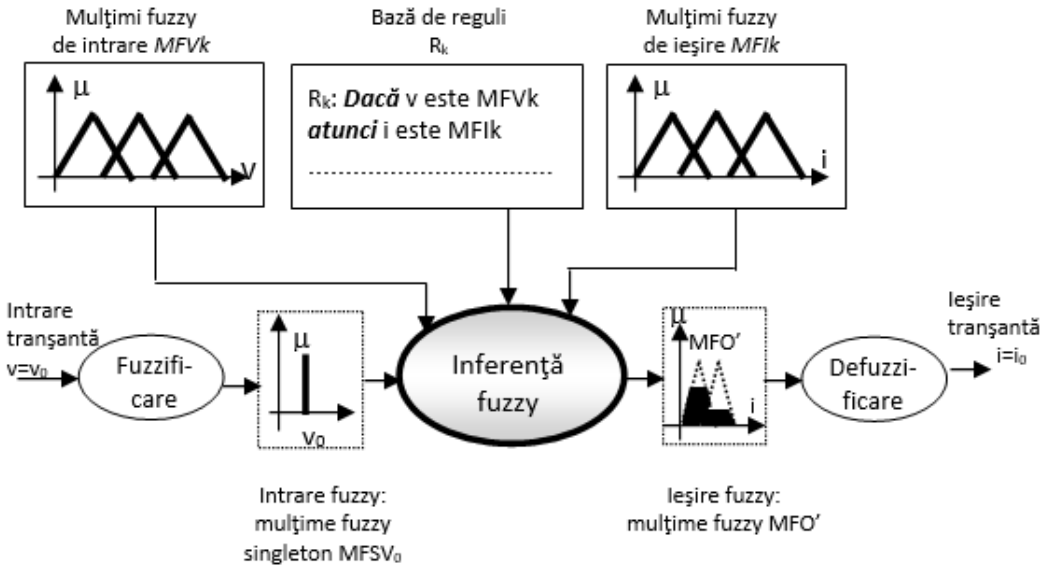


Agregarea concluziilor parțiale se face folosind operatorul *max* (reuniune): $\mu_{MFO'}(i) = \max_k \mu_{MFO'_k}(i)$ echivalent cu $MFO' = \bigcup_k MFO'_k$.

- **defuzzificare –** operația inversă fuzzificării, prin care se extrage o valoare tranșantă din mulțimea fuzzy de ieșire, rezultată în urma inferenței.

Notații:





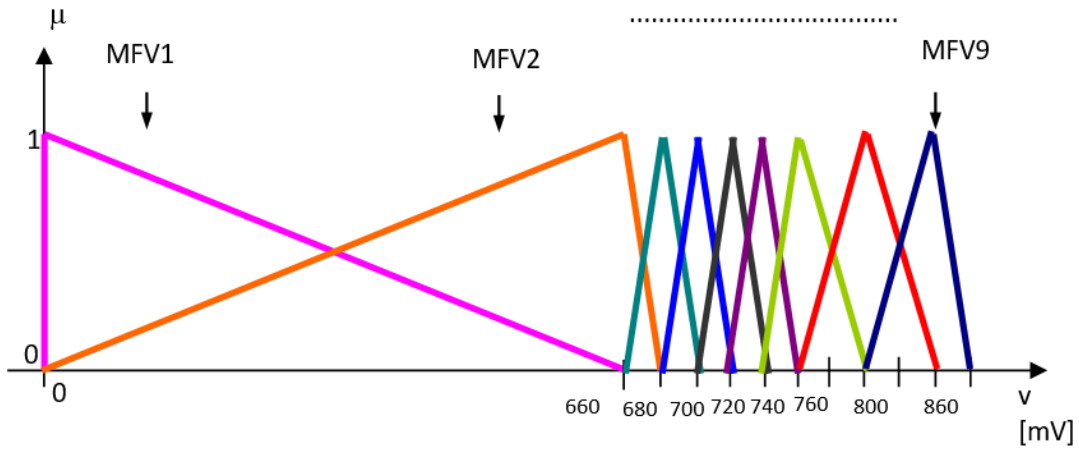
Crearea SLF

Exercițiul 1

Implementați sistemul fuzzy cu o intrare și o ieșire, utilizând *Fuzzy Logic Designer.* Specificațiile sistemului sunt:

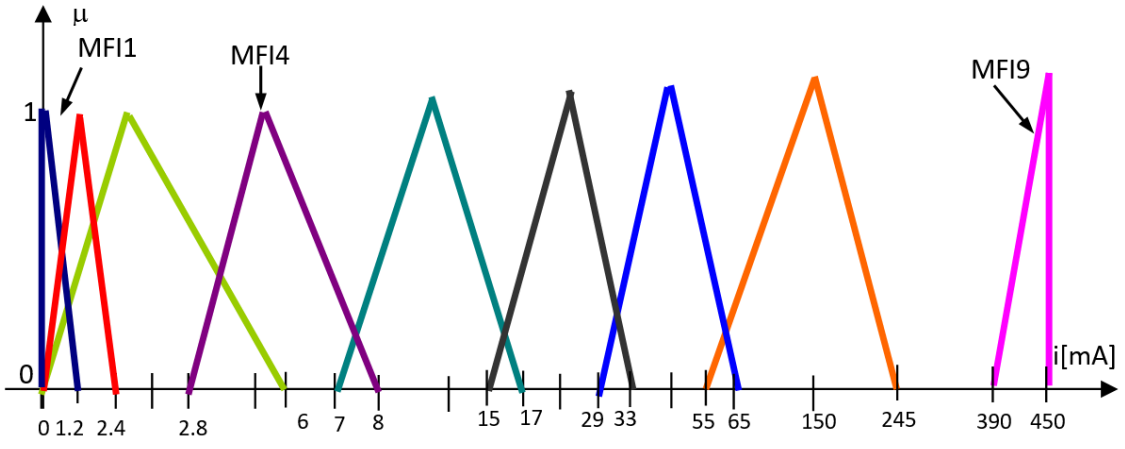
- variabila de intrare v, cu 9 mulțimi triunghiulare, notate *MFVk, k* = 1, 2, ..., 9, având parametrii:

MFV1 - [0 0 680] MFV2 - [0 680 690] MFV3 - [680 690 700]
MFV4 - [690 700 720] MFV5 - [700 720 740] MFV6 - [720 740 760]
MFV7 - [740 760 800] MFV8 - [760 800 860] MFV9 - [800 860 860]



- variabila de ieșire i, cu 9 mulțimi triunghiulare, notate *MFIk, k* = 1, 2, ..., 9, având parametrii:

MFI1 - [0 0 1.2] MFI2 - [0 1.2 2.4] MFI3 - [0 2.4 6]
MFI4 - [2.8 6 8] MFI5 - [7 12 17] MFI6 - [15 24 33]
MFI7 - [29 47 65] MFI8 - [55 150 245] MFI9 - [390 450 450]



- baza de reguli, cu 9 reguli de forma: *Dacă v este MFVk atunci i este MFIk, k* = 1, 2, ..., 9

Salvați sistemul creat cu numele *"dioda.fis".*

Vizualizați curba intrare-ieșire (tensiune-curent). Setați numărul de puncte pentru afișare la 100. Comparați curba obținută cu caracteristica inițială.

Exercițiul 2

Testați sistemul creat pentru un set de valori tranșante ale variabilei de intrare:
v = {0 mV; 650 mV; 675 mV; 700 mV; 725 mV; 750 mV; 775 mV; 800 mV; 850 mV}. Notați valorile obținute la ieșire.

Exercițiul 3

Scrieți un script care să realizeze afișarea graficului curent-tensiune, utilizând funcțiile *readfis*, *evalfis*, *plot*.
Observație: la *evalfis* utilizati si parametrul *NPts* = 10000

Exercițiul 4

Identificați zonele de pe grafic în care sunt prezente erori de aproximare mari. Modificați sistemul fuzzy creat inițial, pentru a reduce aceste erori. Salvați noul sistem fuzzy. Afișați pe același grafic cele două variante ale curbei caracteristicii curent-tensiune.

6. Clasificare substractivă. Modelarea unei funcții neliniare de două variabile pe bază de date numerice.

Obiective: înțelegerea conceptului și metodelor de clasificare a datelor, înțelegerea noțiunilor de clasă și centru al clasei, familiarizarea cu modul de determinare a preciziei unui model.

Observație: MATLAB/Simulink se accesează online (https://matlab.mathworks.com/), prin logare cu credențialele MS Teams (cele de tip nume.prenume@student.utcluj.ro).

Termeni și acronime: *clasificare tranșantă, clasificare substractivă, Fuzzy C-Means.*

○ Clasificarea datelor

Clasificarea este o metodă fundamentală de analiză a datelor, ce are ca scop identificarea grupării naturale a datelor dintr-un set de date de dimensiuni mari. Clasificarea datelor este o metodă de învățare nesupervizată, deoarece valoarea dorită a ieșirii (număr de clase, apartenența fiecărui obiect la o anumită clasă) nu este cunoscută a priori. Aplicații tipice ale clasificării sunt: recunoașterea formelor, extragerea de caracteristici, cuantizarea vectorilor, segmentarea imaginilor, aproximarea funcțiilor, data mining.

Împărțirea în clase se realizează pe baza unei mulțimi de caracteristici ce descriu fiecare obiect. Rezultatul clasificării este o structură fixă a partiționării datelor, adică pentru fiecare clasă se va cunoaște:

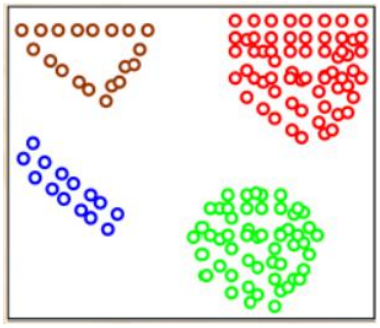
- localizarea - centrul clasei
- forma clasei
- gradul de apartenență al fiecărui obiect la clasa respectivă.

Partiționarea datelor trebuie să respecte:

- omogenitatea în clase - obiectele din aceeași clasă să fie cât se poate de asemănătoare unele cu altele

eterogeneitate între clase - obiectele din clase diferite să fie cât se poate de diferite unele de altele

Un exemplu de măsură a similitudinii dintre obiecte este distanța euclidiană, ilustrată în imaginea de mai jos, unde obiectele care sunt apropiate între ele au fost colorate utilizând aceeași culoare.



Datele de clasificat se reprezintă sub forma vectorilor *N*-dimensionali:

$$X_i = [x_{i1}, x_{i2}, \dots, x_{iN}], X_i \in \mathbb{R}^N, i = \overline{1, \dots, M}$$

unde *N* – numărul caracteristicilor fiecărui obiect, *M* – numărul de obiecte (dimensiunea setului de date).

$$X = \begin{bmatrix} x_{11} & x_{12} \dots & x_{1N} \\ x_{21} & x_{22} \dots & x_{2N} \\ \dots & \dots & \dots \\ x_{M1} & x_{M2} \dots & x_{MN} \end{bmatrix}$$

Obiectivul clasificării este de a găsi cei *K* vectori, care constituie centrele celor *K* clase în care se realizează partiționarea datelor:

$$c_k = [x_{k1}, x_{k2}, \dots, x_{kN}], k = \overline{1, \dots, C}$$

○ Tipuri de clasificare

În ***clasificarea tranșantă***, un obiect aparține în totalitate unei clase, sau nu aparține deloc clasei respective. Astfel, gradul de apartenență al unui obiect la o clasă poate fi 0 sau 1. Fiecare obiect aparține unei clase și nu există clase vide, nici clase care conțin toate obiectele.

În situații reale, clasificarea tranșantă este rareori întâlnită, astfel că se preferă o partiționare a obiectelor în care un obiect să poată aparține mai multor clase în același timp, cu diferite grade de apartenență, cuprinse între 0 și 1. Există două astfel de tipuri de partiționări:

- clasificare fuzzy – Fuzzy C-Means, clasificare substractivă
- clasificare probabilistică.

○ Clasificarea substractivă

Clasificarea substractivă determină numărul de clase și centrele claselor dintr-un set de date. Trebuie precizată valoarea unei raze ce specifică domeniul de influență al centrului clasei, în fiecare dimensiune a datelor.

Etapele algoritmului de clasificare substractivă sunt:

1. Presupune că fiecare punct de date este un potențial centru de clasă și calculează probabilitatea ca acesta să definească un centru, pe baza densității punctelor înconjurătoare.
2. Selectează punctul cu cel mai mare potențial ca fiind primul centru de clasă.
3. Înlătură toate punctele din vecinătatea centrului determinat anterior (în conformitate cu raza precizată), în scopul determinării următoarei clase și a centrului ei.
4. Repetă acest proces până când toate punctele se află în raza de influență a unui centru de clasă.

○ Modelarea unei funcții neliniare de două variabile. Aplicație la modelarea funcțională a unui amplificator transconductanță

Descărcați arhiva "*Date_Machete.zip*" și plasați conținutul acesteia (cu *drag-and-drop*) în directorul curent în care lucrează MATLAB. Dezarhivarea se face cu dublu click pe fișier.

https://drive.google.com/file/d/1C8PpZT3KuE3I-Zm8adnTfMPkSvqjywSt/view?usp=sharing

Arhiva conține 6 fișiere: 3 fișiere de tip *.mat*, în care se regăsesc seturile de date, respectiv 3 script-uri cu extensia *.m*.

Pentru construirea modelului care implementează funcția *amplificare = f(frecvență, temperatură)* se utilizează trei seturi de date:

- "*gendata*" cu dimensiunea 179, pentru generarea sistemului fuzzy inițial
- "*antdata*" cu dimensiunea 35717, pentru antrenarea sistemului fuzzy inițial
- "*verdata*" cu dimensiunea 497, pentru verificarea sistemului fuzzy pe durata antrenării (în scopul detectării suprapotrivirii, dacă este cazul)

Domeniul inițial de variație al mărimilor de intrare este:

frecventa_ini: [1 Hz, 10 MHz]

temperatura_ini: [-55, +125]℃

Pentru a reduce gama dinamică a datelor de intrare și pentru a lucra doar cu valori pozitive, se efectuează următoarele transformări:

frecventa = *log(frecventa_ini)*, astfel frecvența va fi cuprinsă în intervalul [0, 7]

temperatura = *temperatura_ini* + 60, astfel temperatura va fi cuprinsa in intervalul [5, 185]℃

Seturile de date au o structură matricială, fiecare rând al matricii conținând un tuplu de date [*frecvență, temperatură, amplificare*].

Exercițiul 1

Examinați structura celor 3 seturi de date, prin încărcare în workspace, cu dublu-click pe fiecare fișier *.mat*.

Exercițiul 2

Pentru **generarea sistemului fuzzy inițial**, se utilizează script-ul "*generare_aft.m*", în care se va completa cu secvențe de cod, conform instrucțiunilor din fișier.

Generați sistemul fuzzy inițial, utilizând funcția *genfis2*. Examinați proprietățile acestuia, din *Fuzzy Logic Designer*.

Ce valoare are parametrul *radii*?

În câte clase a fost împărțit setul de date?

Câte reguli are sistemul fuzzy generat?

Care este legătura dintre numărul de clase, numărul de reguli și structura fiecărei reguli?

Ce tip au mulțimile fuzzy de intrare? Care este expresia mulțimilor fuzzy de ieșire?

Care sunt centrele claselor, pentru variabilele de intrare? Vizualizați suprafața de control a sistemului fuzzy inițial.

Exercițiul 3

Pentru **determinarea preciziei de aproximare a modelului fuzzy inițial**, se utilizează script-ul "*erori.m*", în care se va completa cu secvențe de cod, conform instrucțiunilor din fișier.

Calculați și salvați într-un fișier:

- valoarea maximă a erorii absolute

$$eroare\_absolută_i = |amplificare\_referință_i - amplificare\_fuzzy_i|$$

- valoarea maximă a erorii relative

$$eroare\_relativă_i = \frac{|amplificare\_referință_i - amplificare\_fuzzy_i|}{amplificare\_referință_i}$$

- eroarea medie procentuală

$$eroare\_medie\_procentuală = \frac{1}{N}\sum_{i=1}^{N}\frac{|amplificare\_referință_i - amplificare\_fuzzy_i|}{amplificare\_referință_i}$$

Exercițiul 4

Pentru **antrenarea sistemului fuzzy inițial**, se utilizează script-ul "*antrenare_aft.m*", în care se va completa cu secvențe de cod, conform instrucțiunilor din fișier.

Antrenați sistemul fuzzy inițial, utilizând funcția *anfis*.

Ce tip de eroare utilizează funcția *anfis*?

Comentați evoluția erorilor pentru seturile de date de antrenare și de verificare. Apare fenomenul de suprapotrivire?

Considerați că ar fi utilă creșterea numărului de epoci de antrenare? Justificați.

Vizualizați suprafața de control a sistemului fuzzy antrenat. Comparați-o cu cea a sistemului fuzzy inițial.

Identificați modificările apărute în mulțimile fuzzy de intrare și de ieșire, pe durata antrenării.

Exercițiul 5

Pentru **determinarea preciziei de aproximare a modelului fuzzy final**, se utilizează script-ul "*erori.m*", în care se va completa cu secvențe de cod, conform instrucțiunilor din fișier.

Calculați și salvați într-un fișier:

- valoarea maximă a erorii absolute

$$eroare\_absolută_i = |amplificare\_referință_i - amplificare\_fuzzy_i|$$

- valoarea maximă a erorii relative

$$eroare\_relativă_i = \frac{|amplificare\_referință_i - amplificare\_fuzzy_i|}{amplificare\_referință_i}$$

- eroarea medie procentuală

$$eroare\_medie\_procentuală = \frac{1}{N}\sum_{i=1}^{N}\frac{|amplificare\_referință_i - amplificare\_fuzzy_i|}{amplificare\_referință_i}$$

Comparați valorile obținute pentru sistemul fuzzy final cu cele obținute pentru sistemul fuzzy inițial.

1. Introduction to Fuzzy Logic Toolbox

Objective: getting used to the functions and graphical interfaces of Fuzzy Logic Toolbox, part of MATLAB/Simulink

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

Terms and abbreviations: *FLS, FIS, fuzzy set, membership degree, control surface*

o Functions and graphical interfaces

MATLAB/Simulink offers 6 categories of functions and graphical interfaces dedicated to fuzzy sets and systems (fuzzy inference system – FIS, fuzzy logic system - FLS):

- *GUI editors* – graphical interface editors
- *Membership functions*
- *Command line FIS functions* - functions for creating/modifying/exploring a FLS
- *Advanced techniques*
- *Miscellaneous functions*
- *GUI helper files* - additional files used by the GUI editors

To visualize the available functions and graphical interfaces in MATLAB, type the following in the command line:

```
help fuzzy
```

o Working with the functions

To explore how the functions work, type "*help* [*function_name*]" in the command line. As an example, for the "*trimf*" function, the code is:

```
help trimf
```

Exercise 1
Run the example presented in the "*trimf*" function help. Analyse what happens, line by line.

Exercise 2
Rerun exercise 1 for another function in the "*Membership functions*" category. Analyse what happens, line by line.

o Launching the graphical editor for creating a FLS

Type the following in the command line:

```
fuzzyLogicDesigner
```

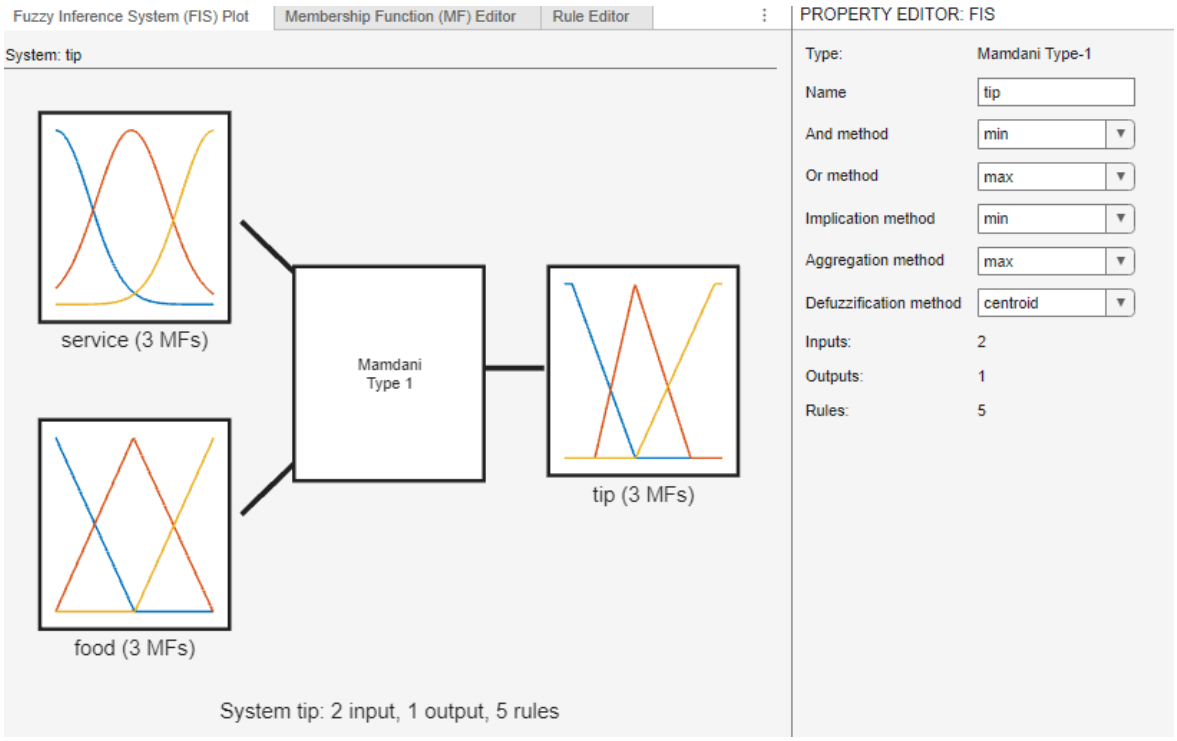
Exercise 3

Load a previously created FLS in the graphical editor – the FLS for computing the tip for a meal at the restaurant, available here:

https://drive.google.com/file/d/10Pvwc1vBSoRagxAzjKvE_mJi3D3V8Mip/view?usp=sharing

Download "*tip.zip*" and place the archive (using *drag-and-drop*) in the current directory of MATLAB. Double click to unzip and view the contents of the folder.

From the graphical editor, using "*Import -> From file...*" select "*tip.fis*".



Analyse the properties of the fuzzy system.

Analyse the membership functions for each variable of the fuzzy system.

Visualise the control surface.

Analyse the rule base and the activation of the rules, from the *Rule Inference* menu. What happens when the connection of a rule is changed?

- Functions for creating/modifying/exploring a FLS from the command line

Exercise 4

Load "*tip.fis*" in the workspace variable "*tip*" using:

```
tip = readfis ('tip.fis');
```

To view the properties of the FLS:

```
getfis(tip)
```

To view the diagram of the FLS:

```
plotfis(tip)
```

To view the control surface of the FLS:

```
gensurf(tip)
```

2. Fuzzy sets. Operations with fuzzy sets applied to colour image segmentation.

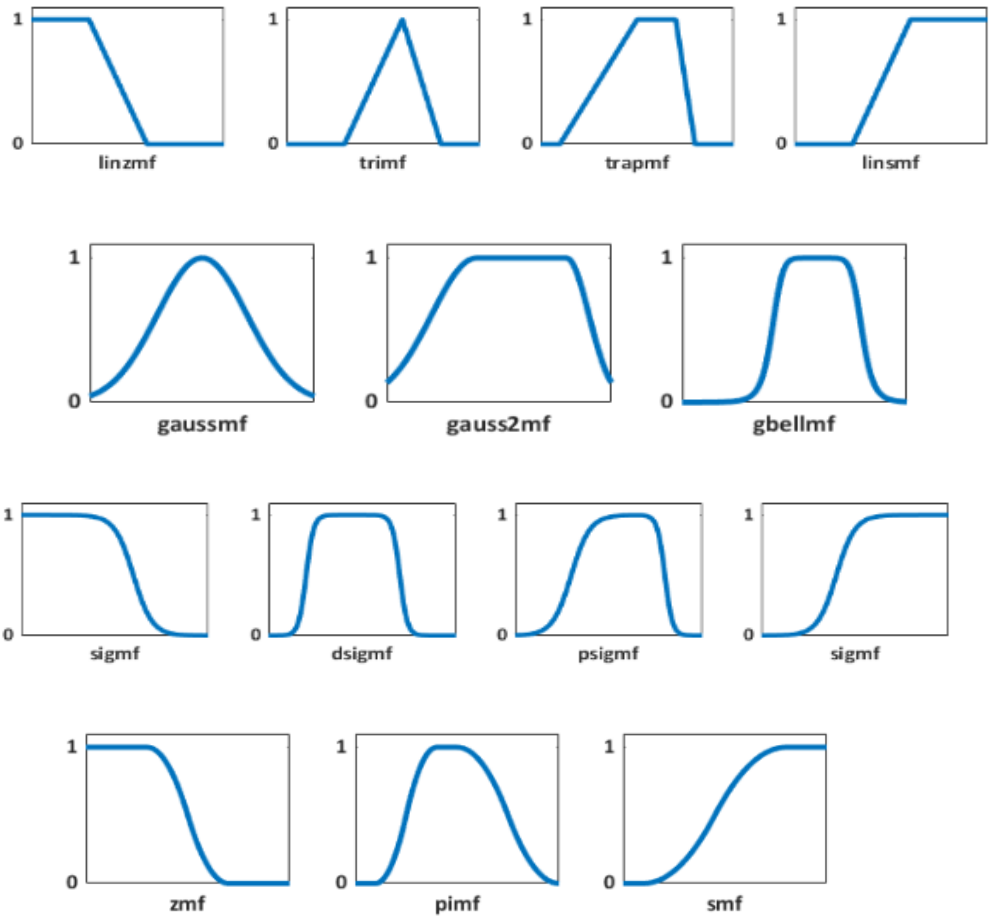
Objectives: getting used to the functions for defining fuzzy sets, understanding basic fuzzy sets operations, understanding the way colours are represented in MATLAB, understanding the basic operations for image processing and image segmentation

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

Terms and abbreviations: *fuzzy set, membership function, union, intersection, complement, segmentation*

o Types of membership functions. Using and editing a membership function.

MATLAB offers 13 types of membership functions (MF – membership function):



To view details about any function, as well as a brief usage example, use "*help function_name*":

```
help trapmf
```

Exercise 1

Copy and paste the following code snippet in the command line. Analyse the code, line by line.

```
close all % closes all the open figure windows
clear all % removes all variables in the workspace
clc       % clears the command window
x = (0:0.1:10)'; % The universe of discourse is [0,10]; the points are
%defined with a step of 0.1
params=[2 3 7 9];% parameters for a trapezoidal membership function
y = trapmf(x,params); % compute the membership function values
plot(x, y,'linewidth',2);
axis([0 10 -0.1 1.1]);
xlabel('universe of discourse'); % horizontal axis variable name
ylabel ('membership degree'); % vertical axis variable name
set(gcf, 'name', 'Trapezoidal membership function', 'numbertitle', 'off');
% figure name
```

Exercise 2

Run the next sequence to find the membership degree of the point x1=2.75, to the above defined trapezoidal fuzzy set.

```
hold on
x1=2.75;
u1=evalmf(x1,params,'trapmf');
sprintf( 'x1=%1.2f has the membership degree u1=%1.2f',x1,u1);
plot (x1,u1,'r*') % place the point on the graph
plot ([x1,x1],[0,u1],...
   'linestyle','-','color','r')
plot ([0,x1],[u1,u1],...
   'linestyle','-','color','r')
hold off
```

Exercise 3

Let "Speed" be a linguistic variable, with the universe of discourse [0, 140] km/h. Define five linguistic values and plot the corresponding membership functions on the same axis (using "*hold on*"). The fuzzy sets must form a fuzzy partition for the [0,140] range. Display at the command line the values of the membership degrees of points 10 km/h, 52 km/h, 85 km/h and 100 km/h to each fuzzy set.

o Fuzzy sets operations

The union of two fuzzy sets A and B is defined using the membership functions:

$$\mu_{AUB}(x) = \max\left(\mu_A(x), \mu_B(x)\right)$$

The intersection of two fuzzy sets A and B is defined as:

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

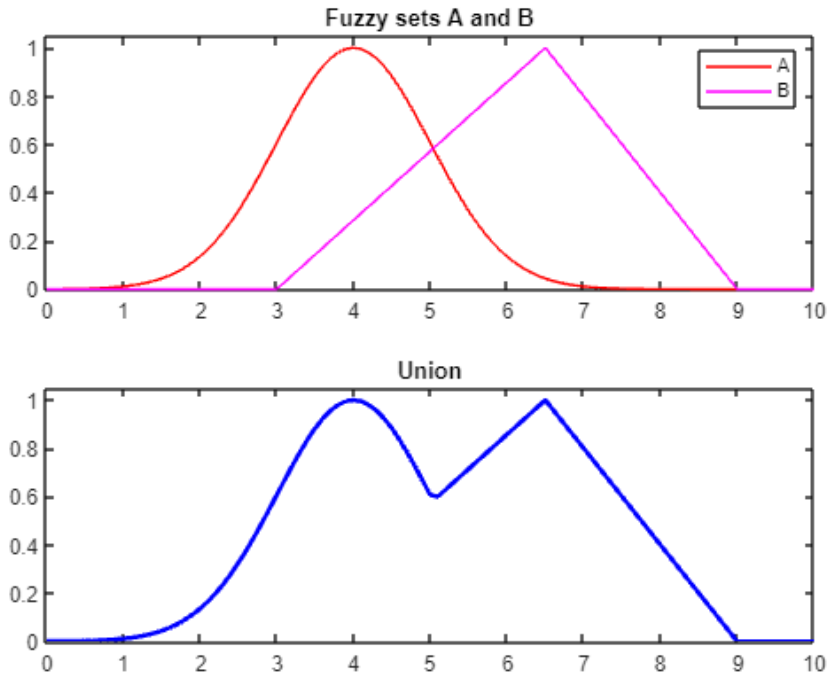
The complement of a fuzzy set A is defined as:

$$\overline{\mu_A}(x) = 1 - \mu_A(x)$$

Exercise 4

Run the following code snippet, to plot the union of two fuzzy sets. Analyse the code, line by line.

```
close all
x = (0:0.1:10)'; % % The universe of discourse is [0,10]; the points are
%defined with a step of 0.1
u1=gaussmf(x,[1,4]); % first membership function, gaussian type
u2=trimf(x,[3 6.5 9]);   % second membership function, triangular type
u_union=max(u1,u2);  % compute the membership degrees for the union using
%the "MAX" operator
hold on
subplot(2,1,1); % breaks the figure in two windows
                % the current plot appears in the upper window
plot (x,u1,'r');hold on
plot (x,u2,'m'); hold off
axis([0 10 0 1.05]);
legend ('A','B');
title('Fuzzy sets A and B')
subplot (2,1,2) % the current plot appears in the lower window
plot(x, u_union,'color','b','linewidth',2)
axis([0 10 0 1.05]);
title('Union')
set(gcf, 'name', 'Union of A and B fuzzy sets - "max" operator',
'numbertitle', 'off'); %figure name
```



Exercise 5

Change the previous code snippet, to plot in a single figure, on different subplots, the following: fuzzy sets A and B, union, intersection, complement.

o Colour representation in MATLAB. RGB and HSV colour spaces.

The **RGB** (red, green, blue) colour space is defined by the three primary colours, red, green and blue, out of which any other colour can be obtained, by weighted additive combinations. The values of R, G and B are in [0;1]. To represent any colour in MATLAB in the RGB space, one can use either the name (full or abbreviated) of the colour, for predefined colours, or a vector that contains the values of the primary components.

To set a pink background for the current figure, use the following code in the command line:

```
figure
hold on
set(gcf,'Color',[1,0.4,0.6]);
```

Exercise 6

Change the background colour of the current figure to cyan, then to green, using both the abbreviated name and the vector representation.

In the **HSV** (hue, saturation, value) colour space, colours are represented by the perceptual attributes – hue, saturation, and value. The values for H, S and V are in [0;1]. The default colour representation in MATLAB is in the RGB space. To convert from RGB to HSV, the dedicated function is "r*gb2hsv*".

o Representation of colours using fuzzy sets

The values between 0 and 1 of the hue component, H, define the colours of the visible spectrum: red, orange, yellow, green, cyan, blue, purple, magenta, pink, and red again, at the end of the spectrum. To represent colours as fuzzy sets, the universe of discourse is the range of possible values for hue; when using an 8-bit representation, the range of values for H is [0; 255]. A different fuzzy set is defined for each colour, and the linguistic values are: Red, Orange, Yellow, Green, Cyan, Blue, Purple, Magenta, Pink, over the linguistic variable H (range [0; 255]), with values in [0; 1].

Exercise 7

Run the following code snippet. Analyse the code, line by line.

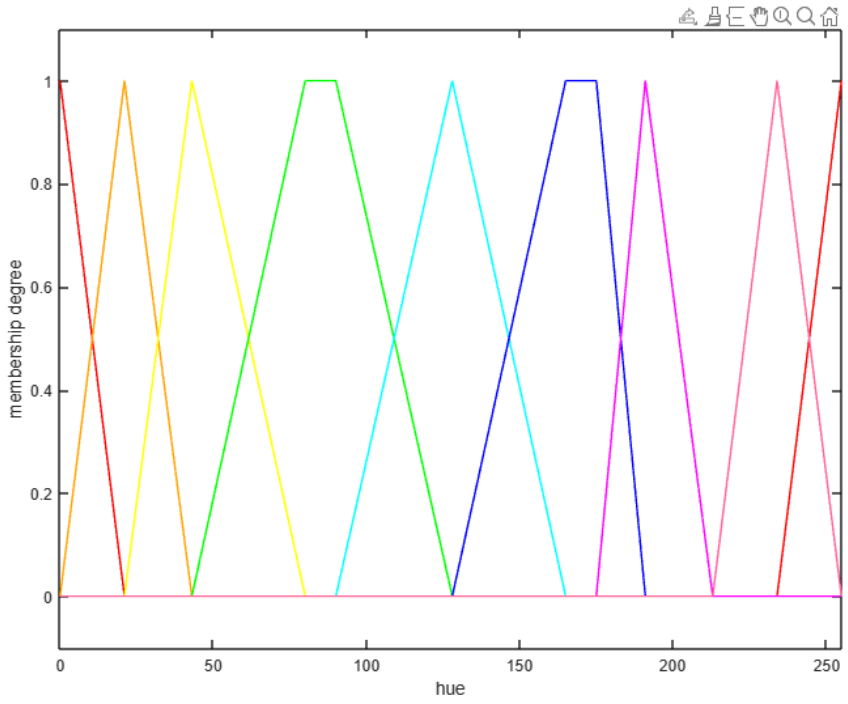
```
close all;
clear all;
clc;

% fuzzy sets definition
x=(0:.5:255)'; % universe of discourse
Red_l=trimf(x, [0 0 21]); % red, left
Red_r=trimf(x, [234 255 255]); % red, right
Red=max(Red_l, Red_r);
Orange=trimf(x, [0 21 43]); % orange
Yellow=trimf(x, [21 43 80]); % yellow
Green=trapmf(x, [43 80 90 128]); % green
Cyan=trimf(x, [90 128 165]); % cyan
```

```
Blue=trapmf(x, [128 165 175 191]); % blue
Purple=trimf(x, [175 191 213]); % purple
Pink=trimf(x, [213 234 255]); % pink

plot(x, Red_l, 'color', 'r');
axis ([0 255 -.1 1.1]);
hold on;
xlabel('hue'); % x axis variable name
ylabel ('membership degree'); % y axis variable name
set(gcf, 'name', 'Representation of colours as fuzzy sets', 'numbertitle',
'off'); % figure name

plot(x, Red_r, 'color', 'r');
hold on;
plot(x, Orange, 'color', [1 0.64 0]);
hold on;
plot(x, Yellow, 'color', 'y');
hold on;
plot(x, Green, 'color', 'g');
hold on;
plot(x, Cyan, 'color', 'c');
hold on;
plot(x, Blue, 'color', 'b');
hold on;
plot(x, Purple, 'color', 'm' );
hold on;
plot(x, Pink, 'color', [1,0.4,0.6]);
hold off;
```



o Segmentation of a colour image

Segmentation of a colour image means isolating areas of a certain colour in an image. Let's assume the red colour segmentation is desired: the areas or objects coloured in red will be separated, by representing them using shades of grey. The shades of grey will be very close to white, if the object is almost red, and darker (even black), if the object is any other colour.

Using the previously defined fuzzy sets for colours, a new fuzzy set is created, called *CloseToRed*, that expresses the concept of objects with red dominant colour or objects that are very close to red. The *CloseToRed* fuzzy set is defined using the basic operations:

CloseToRed = *Red* OR (*Orange* NOT *Yellow*) OR *Pink*

$$CloseToRed(H) = \max(Red(H), \min(Orange(H), 1 - Yellow(H)), Pink(H))$$

Exercise 8

Run the following code snippet to plot the *CloseToRed* fuzzy set. Analyse the code, line by line.

```
figure;
NOT_Yellow=1-Yellow;
OrangeNotYellow=min(Orange, NOT_Yellow);
CloseRed=max(Red, OrangeNotYellow);
CloseToRed=max(CloseRed, Pink);
plot(x, CloseToRed, 'Linewidth', 2, 'color','r');
axis ([0 255 -.1 1.1]);
xlabel('hue');  % x axis variable name
ylabel ('membership degree'); % y axis variable name
set(gcf, 'name', 'CloseToRed fuzzy set', 'numbertitle', 'off'); % figure
%name
```

The membership degree of a pixel to the *CloseToRed* fuzzy set is proportional to its shade of red. The visual representation of the result is done by scaling the resulting membership degree in the [0;255], for each pixel. In the segmented image, the red or almost red areas will be replaced by white or almost white areas.

Exercise 9

Download "*images.zip*" and place the archive (using *drag-and-drop*) in the current directory of MATLAB. Double click to unzip and view the contents of the folder.

https://drive.google.com/file/d/1LvgcTts3SRZHJAH0gAOtzejvt-oUgdp1/view?usp=sharing

Run the following code snippet to segment a colour image. Analyse the result of the segmentation for each image in the folder.

```
close all; clc; clear all
% available images; use only one at once
  img='Tiffany24.bmp';
% img='Cuticle.bmp';
% img='Lenna.bmp';
% img='Mouse.bmp';
% img='Peppers2.bmp';
% img='Boats24.bmp';

OrigImg=imread(img); % reading the image
```

```
HSVimg=(rgb2hsv(OrigImg)); % RGB->HSV conversion
Himg=(HSVimg(:,:,1));   % extracting the hue component, between [0,1]
H=(Himg*255);        % normalizing to [0, 255]

% computation of the membership degree of each pixel, to each fuzzy set
RedH_l=trimf(H, [0 0 21]);
RedH_r=trimf(H, [234 255 255]);
RedH=max(RedH_l, RedH_r);
OrangeH=trimf(H, [0 21 43]);
YellowH=trimf(H, [21 43 80]);
GreenH=trapmf(H, [43 80 90 128]);
CyanH=trimf(H, [90 128 165]);
BlueH=trapmf(H, [128 165 175 191]);
PurpleH=trimf(H, [175 191 213]);
PinkH=trimf(H, [213 234 255]);

% normalizing to [0, 255]
RedObj=uint8(255*RedH);
% creating the "CloseToRed" fuzzy set
NOT_YellowH=1-YellowH;
OrangeNotYellowH=min(OrangeH, NOT_YellowH);
CloseRedH=max(RedH, OrangeNotYellowH);
CloseToRedH=max(CloseRedH, PinkH);
% displaying the images
imshow(OrigImg); set(gcf, 'name', 'Original image'); % original image
figure
imshow(RedObj); set(gcf, 'name', 'Red segmentation); % highlighting the red
%segments
figure
imshow (uint8(CloseToRedH*255)); set(gcf, 'name', 'Close to red
segmentation); % highlighting the close to red segments
```

Original image

Red segmentation

Close to red segmentation

Exercise 10

Create a script to define a fuzzy set that approximates a different colour (other than red). Plot the fuzzy set. Apply the segmentation using the pure colour and the approximated colour.

Use the images in the folder to test the segmentation or download other relevant images.

3. Simulating Fuzzy Logic Systems in MATLAB. Application - fuzzy logic washing machine.

Objectives: implementing a fuzzy logic system using the Fuzzy Logic Toolbox, analysing various methods for implication and defuzzification.

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

Terms and abbreviations: *fuzzification, defuzzification, implication, centroid, bisector, MOM, SOM, LOM*

o Process control using fuzzy logic systems

One of the most important practical applications of FLSs is using them as "process control systems". FLSs as control systems rely on a solid theoretical background and are integrated in many commercial applications. The example chosen in this lab aims to show how the washing time of an automatic washing machine can be controlled, using a fuzzy logic system.

When using a washing machine, the user manually sets the washing time, based on the volume of clothes and the type and degree of dirtiness. To automate the washing process, sensors can be used to detect the volume of clothes and their dirtiness. A certain washing time will be chosen, based on this data.

Unfortunately, a precise mathematical relation between the inputs (volume of clothes, dirtiness) and the output (washing time) cannot be defined. Thus, the washing time is manually set by the user, based on previous experience and trials.

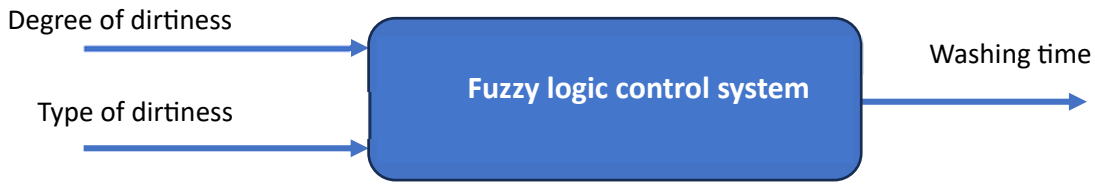
Building a washing machine with automatic washing time determination means building the following two subsystems:

- the sensor system - collects data from the outer environment (the clothes) and send them to the controller
- the controller system - sets the washing time, based on the information received from the sensor system.

Exercise 1
What type of sensors can be used to determine the volume, color, material, degree and type of dirtiness of the clothes?

o Creating the FLS for the control of the washing time

The goal is to create a fuzzy logic controller for a washing machine, that will compute the appropriate washing time, based on certain information about the clothes. A schematic view of this system is deployed in the figure below:

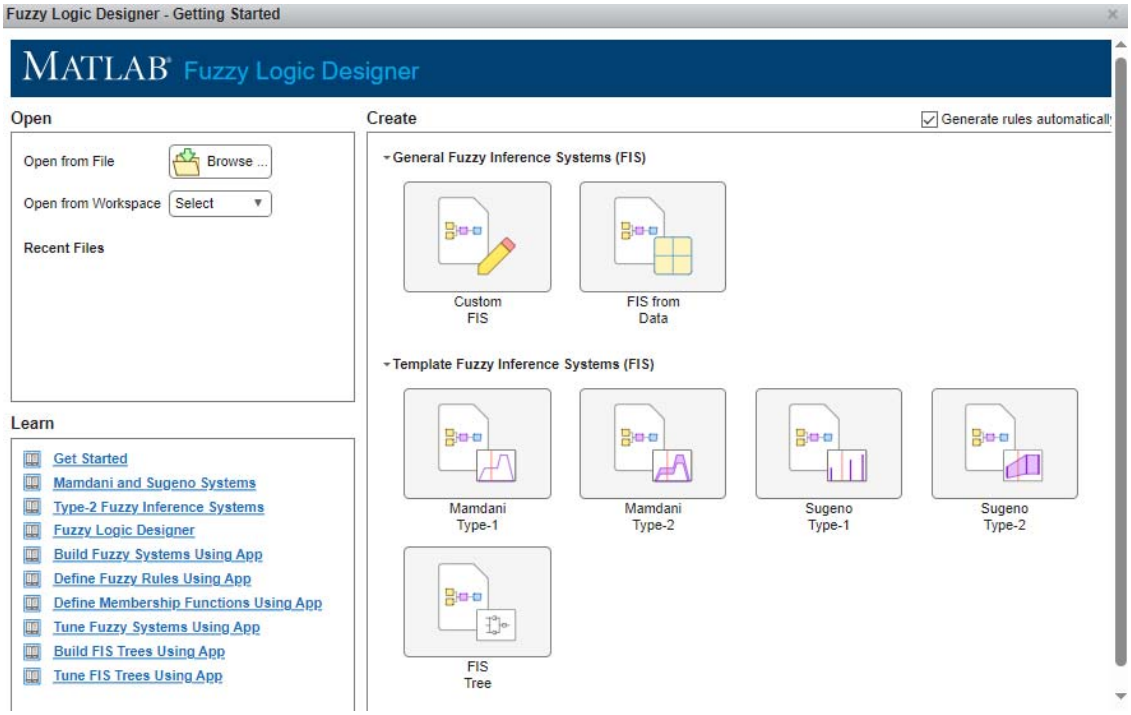


The two inputs of the system are:

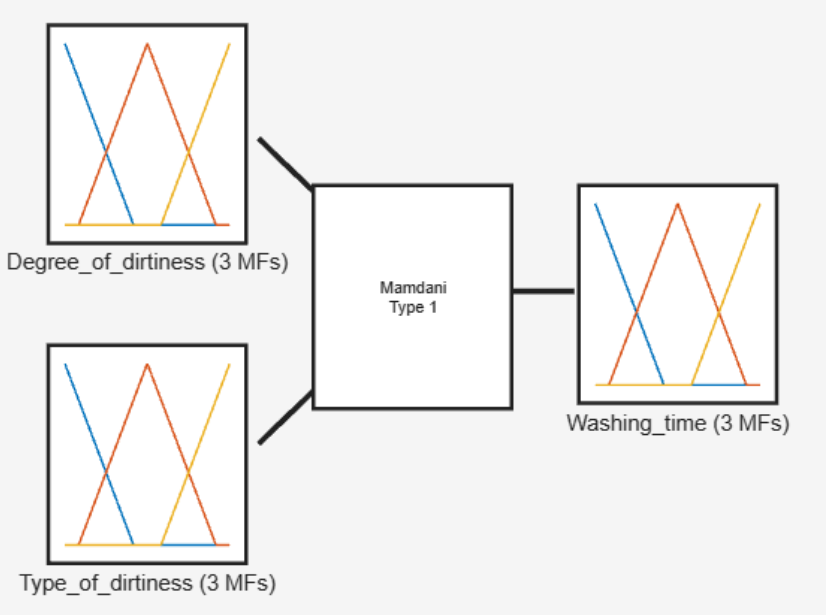
- degree of dirtiness - can be determined by examining the transparency of the water
- type of dirtiness - can be determined by examining the soaking time needed for the water to get to a constant transparency - transparency saturation. For example, for the clothes with fat stains, the time is longer, because the fat dissolves slower in water than other types of dirt.

The FLS is built using *Fuzzy Logic Designer*; to launch the tool, type the following in the command line:

```
FuzzyLogicDesigner
```



Select *Mamdani Type-1* (predefined Mamdani fuzzy system) and set the names for the input and output variables.



The fuzzy sets defined over the three variables have the names, types and parameters in the table below. The fuzzy sets define a fuzzy partition over the universe of discourse, for all three variables.

Variable name	**Universe of discourse**	**Name**	**Type**	**Parameters**
Degree_of_dirtiness	[0…100] %	*Low* *Medium* *High*	*trimf*	[-40 0 50] [0 50 100] [50 100 140]
Type_of_dirtiness	[0...100] %	*NonFat* *Medium* *Fat*	*trimf*	[-40 0 50] [0 50 100] [50 100 140]
Washing_time	[0...60] minutes	*VeryShort* *Short* *Medium* *Long* *VeryLong*	*trimf*	[0 8 12] [8 12 20] [12 20 40] [20 40 60] [40 60 60]

The rule base of the system is:

Type_of_dirtiness \ *Degree_of_dirtiness*	***Low***	***Medium***	***High***
NonFat	VeryShort	Short	Medium
Medium	Medium	Medium	Long
Fat	Long	Long	VeryLong

All rules use the AND connector.

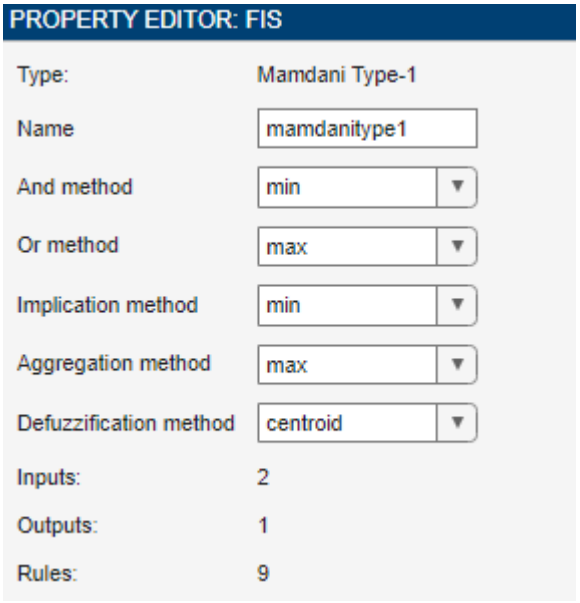
Exercise 2

Define the fuzzy sets for all three variables and the rule base, as specified in the previous tables.

The operations of the FLS are:

- fuzzification – turns the crisp input into a *singleton* fuzzy set
- inference - max-min (Mamdani) type
- defuzzification - *centroid* (COA - center of area) defuzzification is used. The output values obtained by using this type of defuzzification guarantee a smooth control surface, which is a very important demand in control processes. The crisp output of a centroid defuzzification is computed using:

$$t_{0-continuous} = \frac{\int_{t=0}^{60} t * \mu_{MFO}(t)}{\int_{t=0}^{60} \mu_{MFO}(t)} \qquad t_{0-discrete} = \frac{\sum_{t=0}^{60} t * \mu_{MFO}(t)}{\sum_{t=0}^{60} \mu_{MFO}(t)}$$



Exercise 3

Analyse the behaviour of the FLS by viewing the control surface and the operations.

Exercise 4

Collect the control surfaces and the crisp output values in a document, for 5 relevant pairs of input values, in the following cases:

- `Implication: min; Defuzzification: centroid`
- `Implication: prod; Defuzzification: centroid`
- `Implication: min; Defuzzification: bisector`
- `Implication: min; Defuzzification: MOM`

Why does the surface of a control system generally need to be as smooth as possible? Which defuzzification method is the best, and which is the worst, for the current application?

4. Fuzzy logic control systems. Fuzzy temperature controller.

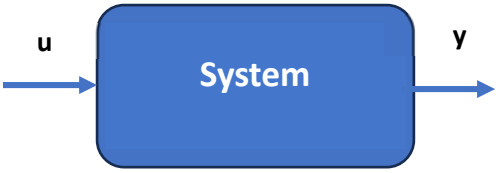
Objectives: understanding how a classical controller works, understanding the differences between classical and fuzzy logic controllers, visualising the output of a fuzzy controller.

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

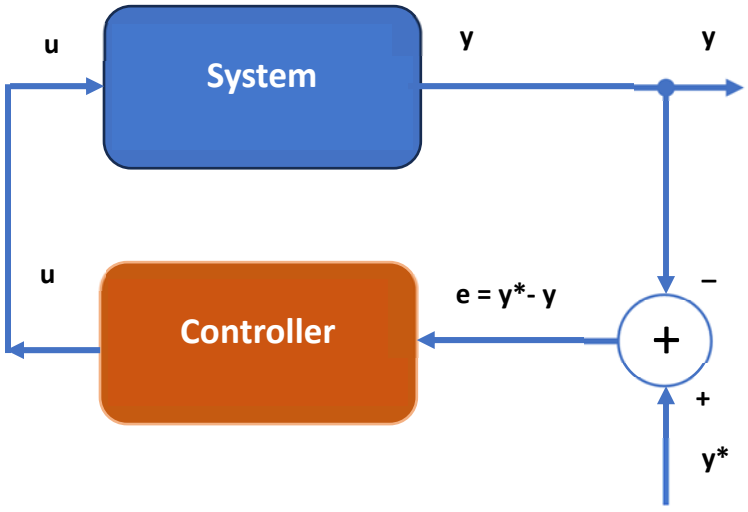
Terms and abbreviations: *classical controller, fuzzy controller, PID.*

o Classical control systems

Consider an open loop system, with a control input **u** and an output **y**:



The goal is to provide a desired output, **y***. The system is designed so that, in the absence of disturbances and variations of system parameters, **y = y*** for a certain input **u = u***. The absence of disturbances is impossible in real life situations, so **y*** is different from **y**, if the system works in an open loop, for an input **u = u***. Thus, to ensure **y = y*** when disturbances are present, **u** needs to be different from **u***, to compensate the disturbances. The change in **u** depends on the change of **y** with respect to **y***, and is achieved by connecting another system, called ***controller***, between the output and the input of the initial system:



The new system is called *closed loop system*, or *feedback system*. The output **u** of the controller represents the command input of the system and generally depends on the errors computed at previous times: differences between output **y** and desired output **y***, but also on the previous commands **u**:

$$u(k) = f(e(k), e(k-1), \dots, e(k-t), u(k-1), \dots, u(k-t))$$

where *f* is the control law, and *t* is the order of the controller. For t > 0, the controller has a memory of *t*.

The error *e* is computed as:

$$e(k) = y^* - y$$

Generally, the control law *f* is nonlinear. In classical control theory, the control law *f* is inferred based on the mathematical model of the open loop process.

The classical control laws are:

a) the proportional control law (P):

$$u = K_p * e \Rightarrow u(k) = K_p * e(k)$$

b) the integral control law (I):

$$u = K_i * \int e\, dt$$

or the discrete version:

$$u(k) = K_i * \sum_{j=0}^{\tau} e(k-j)$$

c) the derivative control law (D):

$$u = K_d * \frac{d^\tau e}{dt^\tau}$$

d) combinations of these laws, such as the PI controller:

$$u(k) = K_p * e(k) + K_i * \sum_{j=0}^{\tau} e(k-j)$$

o Fuzzy controllers

Given a fuzzy logic system with the inputs **e(k), e(k-1),..., e(k-t), u(k-1),..., u(k-t)**, a linguistic dependency between the output **u(k)** and these inputs can be found. Most commonly used fuzzy controllers are for t = 1:

$$u(k) = f(e(k), e(k-1), \dots, u(k-1))$$

Typical fuzzy controllers have a even more compact form, where the previous output **u(k-1)** is not taken into account. The inputs are only **e(k)** and **e(k-1)**, and the output of the controller is the variation of **u**, defined as:

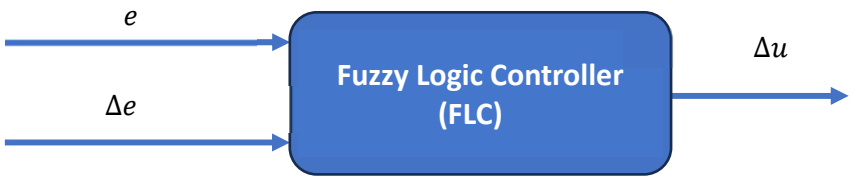
$$\Delta u(k) = u(k) - u(k-1) \Rightarrow u(k) = \Delta u(k) + u(k-1)$$

$$\Delta u(k) = F\big(e(k), e(k-1)\big)$$

where **F** is the transfer function of the control system, given by:

- the fuzzy sets for the inputs and output
- the fuzzy rule base
- the inference mechanism
- the defuzzification method.

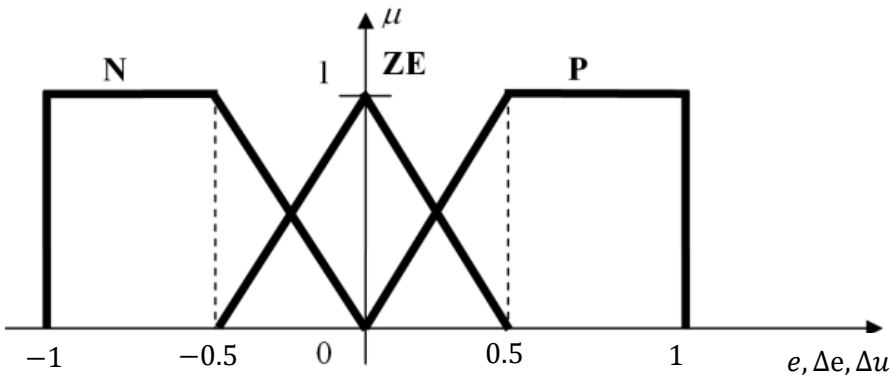
Typically, the crisp inputs of the FLS are the error $e(k)$, the variation of the error $\Delta e(k) = e(k) - e(k-1)$, and the output is $\Delta u(k) = F\big(e(k), e(k-1)\big)$.



This fuzzy logic controller with t = 1 was proposed in 1975 Mamdani and Assilian and is called Mamdani-type FLC.

o Mamdani-type fuzzy logic controller - example

The easiest way to define the fuzzy sets for inputs and output is by using three fuzzy sets (**Negative N**, **Zero ZE**, **Positive P**), identical for the two inputs and the output.



The rule base is deduced knowing that the goal is **y = y***, meaning **e = y* - y = 0**. In other words, the desired output is ***"e is ZE"***. Additionally, it is assumed that the output **y** and the command **u** have the same type of variation:

- if **u** increases, **y** increases;
- if **u** is constant, **y** is constant;
- if **u** decreases, **y** decreases;

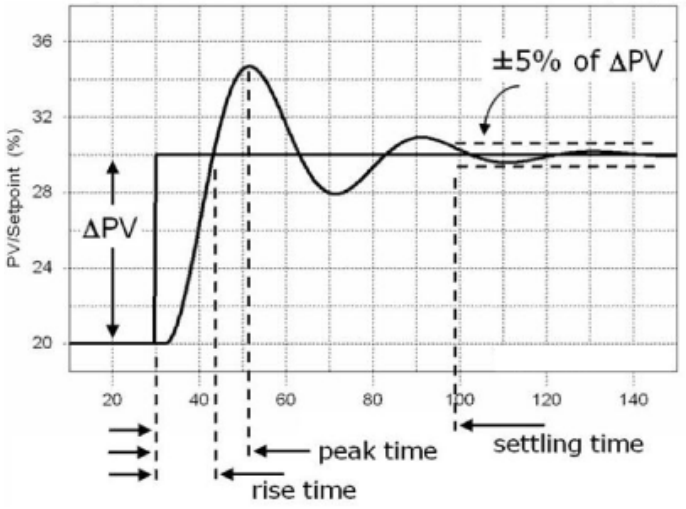
The complete rule base takes into account all possible combinations of the two outputs:

Δe \ e	***N***	***ZE***	***P***
N	N	N	ZE
ZE	N	ZE	P
P	ZE	P	P

The inference mechanism is usually Mamdani, that is max-min inference, and the defuzzification method is COA (centroid).

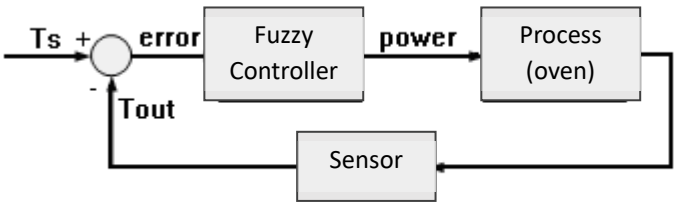
The typical response of a system for automatically adjusting to a step signal is presented in the next figure and can be characterized by several parameters:

- rise time, peak time, settling time
- overshoot.



o Fuzzy logic temperature controller

The control system consists of a process (oven) and the fuzzy controller:

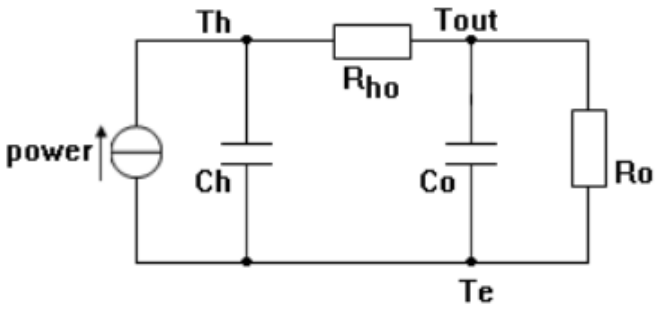


where:

- *Ts* – desired temperature (*Set Point Temperature*)
- *Tout* measured temperature (inside the oven)
- *error* = *Ts* – *Tout*

- *power* - power required for heating/cooling (temperature controller output, and the command input of the process).

The purpose of the controller is to maintain the temperature inside the oven at a constant value, equal to the desired temperature (*Ts*). To model the heat generation and transfer, the following equivalent circuit is used:



The *power* current source (thermal power) represents the power supplied to the heating/cooling element. The system has an electrical heating/cooling element with a capacity of Ch=500[J/°C], connected through a resistor Rho=0.143[°C/W], for a heating capacity of Co=1000[J/°C]. The oven dissipates heat into the exterior (external temperature Te), through the thermal resistor Ro=0.1[°C/W]. The temperature controller adjusts the power dissipated to the heating element *power*, by comparing the temperature inside the oven Tout against the set point (desired) temperature Ts.

Download "*TempControl.zip*" and place the archive (using *drag-and-drop*) in the current directory of MATLAB. Double click to unzip and view the contents of the folder.

https://drive.google.com/file/d/1c_XcAD8KaaSrx37ixkdDm-1x_JAHES0n/view?usp=sharing

The Simulink block diagram is shown below. To make sure that the controller is universal, linear conversion blocks were used at the two inputs (*Scale error*, *Scale delta_error*) and at the output (*Scale_power*). The values at the inputs of the controller are limited to [-1, 1] by means of saturation blocks (*Saturation*, *Saturation1*).

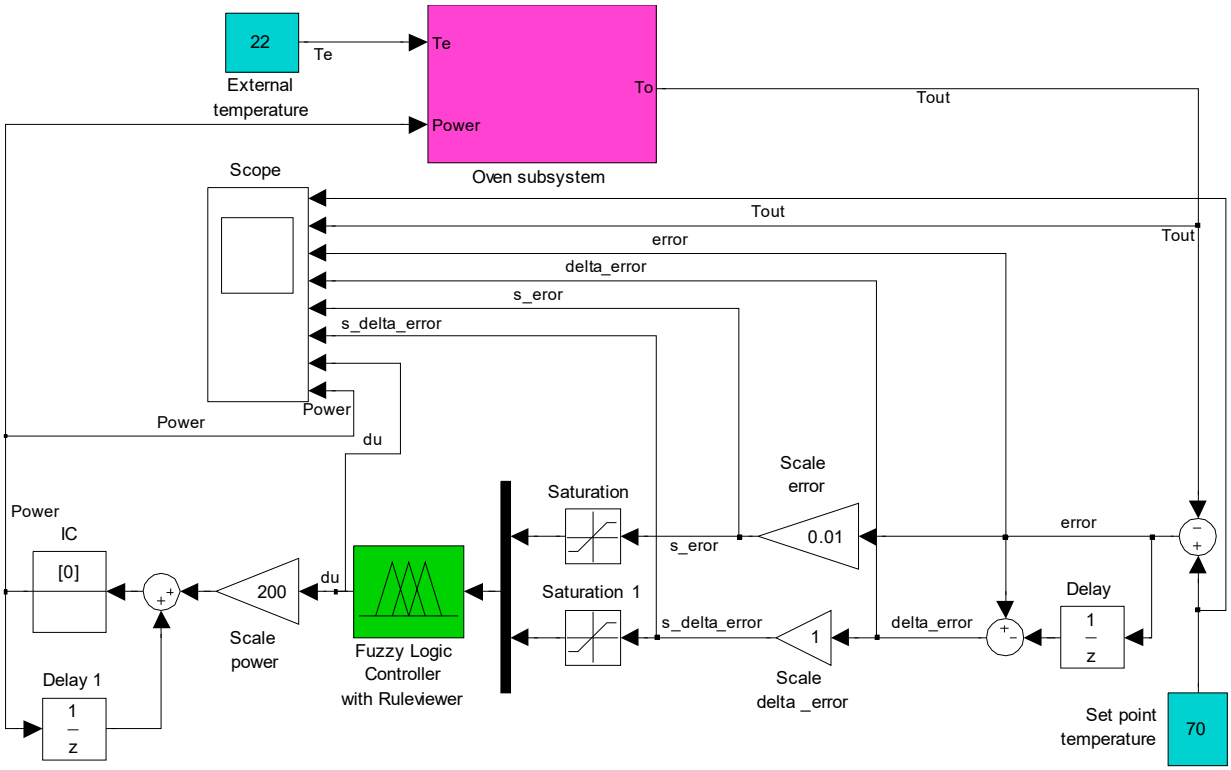
Exercise 1

Implement the Mamdani-type fuzzy logic controller, based on the previous specifications. Visualise the control surface. Save the system as *„TempControlM.fis".* Read the system into a variable called *fls*, using *readfis*.

Exercise 2

Open the Simulink model of the temperature control process, "*TempControlM.mdl*". Initialize the parameters of the oven, by running the *„HeaterOven_params.m"* script.

Start the simulation and visualize the waveforms on the oscilloscope (double-click on *Scope*). Measure the parameters of the control system: rise time, settling time, overshoot. Save the measured values.



Exercise 3

Adjust the scaling factors for inputs and output, so that the performance of the control system is improved.

Exercise 4

Repeat *Exercise 2*, this time using a Takagi-Sugeno type controller. The conversion from Mamdani to Takagi-Sugeno is done by using the *Mamdani to Sugeno* option. In this case, the Simulink model is "*TempControlTS.mdl*". Which of the two control systems performs better, for the initial values of the scaling factors?

5. Approximation of the current-voltage diode characteristic using a Fuzzy Logic System

Objectives: understanding the motivation behind using FLSs as universal approximators, experimenting the connection between input and output fuzzy sets and the rule base, understanding fuzzification/inference/defuzzification, creating a complete FLS for the approximation of a given function.

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

Terms and abbreviations: *function approximation, function modelling.*

o Fuzzy logic systems as universal approximators

One of the most important applications of fuzzy logic systems is function approximation. Let **y = f(x)** be a graphical function, with an approximate analytic representation. One can design a FLS with **x** as input and **y** as output, so that the difference **e** between the estimated value of **y**, **y_est**, and **f(x)**, is as small as required.

$$|y\_est - f(x)| < e$$

The fuzzy logic system with the input **x** and the output **y** is a ***universal approximator for graphical functions*** **y=f(x)**.

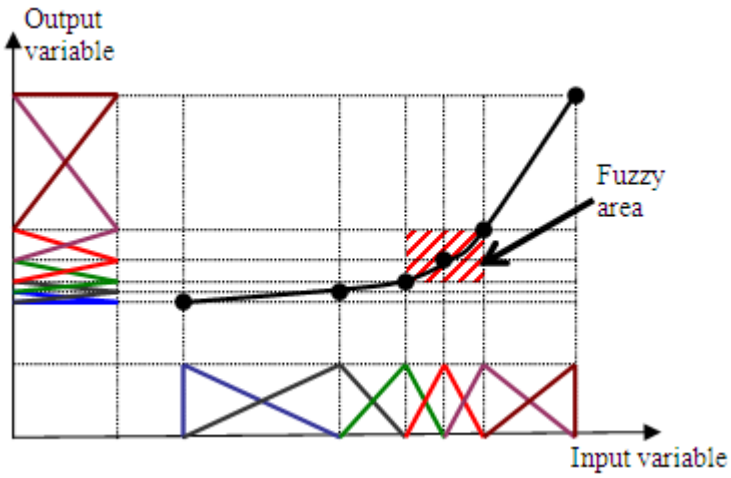
o Fuzzy modelling of the current-voltage diode characteristic

The goal is to approximate the forward bias region of the current-voltage (i-v) characteristic of a semiconductor diode, based on the i-v plot, obtained either through experiments or from the datasheet. The FLS is described as follows:

- **Input variable** – values of the voltage drop across the diode in forward bias, v = v_AK, between [0 mV, 860 mV]
- **Output variable** – values of the forward current through the diode, i = i_A, between [0 mA, 450 mA]

The FLS completely describes the electrical behaviour of the diode in forward bias, for v = v_AK, between [0 mV, 860 mV], ignoring temperature, humidity and technological deviations variations. Thus, the FLS can be used as an electriocal model of the diode.

The principle behind modelling the i-v diode characteristic is based on covering the non-linear curve with “fuzzy areas”:



Each "fuzzy area" is described by:

- a fuzzy set over the universe of discourse for the input variable v, denoted *MFVk*
- a fuzzy set over the universe of discourse for the output variable i, denoted *MFIk*
- a fuzzy rule Rk, that indicates the fact that the two fuzzy sets *MFVk* and *MFIk* are connected through a fuzzy relation

Rk: If v is *MFVk*, then i is *MFIk*

Note: For the highly non-linear regions of the plot, it is necessary to use more fuzzy sets for input and output.

o FLS for modelling the i-v diode characteristic

The FLS for modelling the i-v diode characteristic consists of:

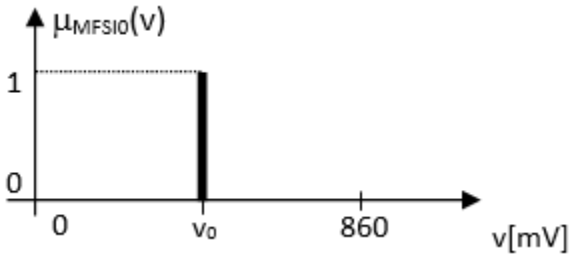
- the fuzzy sets over the universe of discourse for the input variable v, denoted *MFVk*
- the fuzzy sets over the universe of discourse for the input variable i, denoted *MFIk*
- the fuzzy rule base, with *k* rules: Rk: : If v is MFVk, then i is MFIk

The components of the FLS and the operations that turn a crisp input (v = v_0 – voltage drop across the diode) into a crisp output (i = i_0 – current through the diode) are shown in the figure.

The operation of the FLS are:

- **fuzzification** - transforming a crisp input value into a singleton input fuzzy set, with the following membership function:

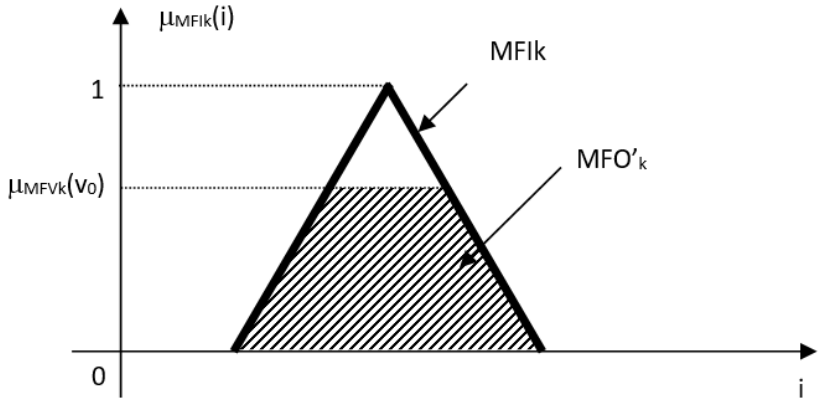
$$\mu_{MSFI_0}(v) = \begin{cases} 1, v = v_0 \\ 0, v \neq v_0 \end{cases}$$



- **inference** - computing the Yk result of each Rk rule, based on the fuzzy input value and on each Rk rule, and using the *min* (Mamdani) implication method:

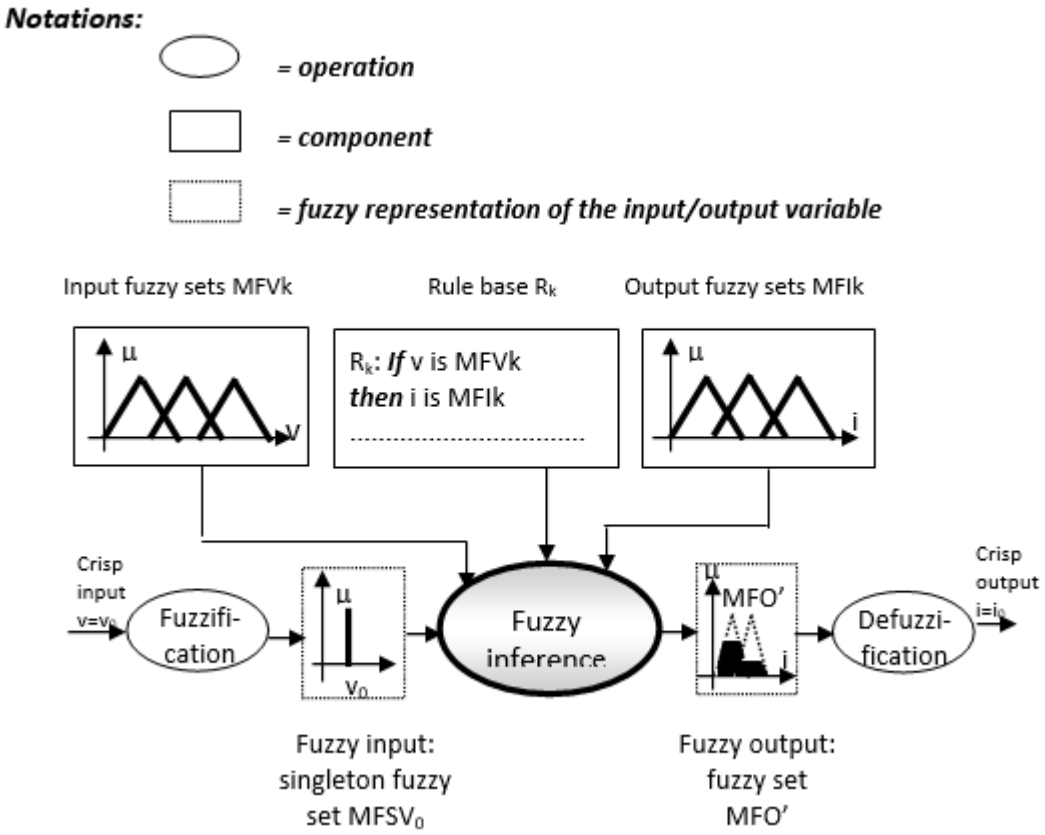
$$\mu_{MFO'_k}(i) = \min\left(\mu_{MFV_k}(v_0), \mu_{MFI_k}(i)\right)$$

The output fuzzy set for the rule Rk (partial conclusion) is:



Aggregating the partial results is done using *max* (union) operator: $\mu_{MFO'}(i) = \max_k \mu_{MFO'_k}(i)$, equivalent to $MFO' = \bigcup_k MFO'_k$.

- **defuzzification –** the opposite of fuzzification. It extracts a crisp value from the output fuzzy set, obtained in the inference process.



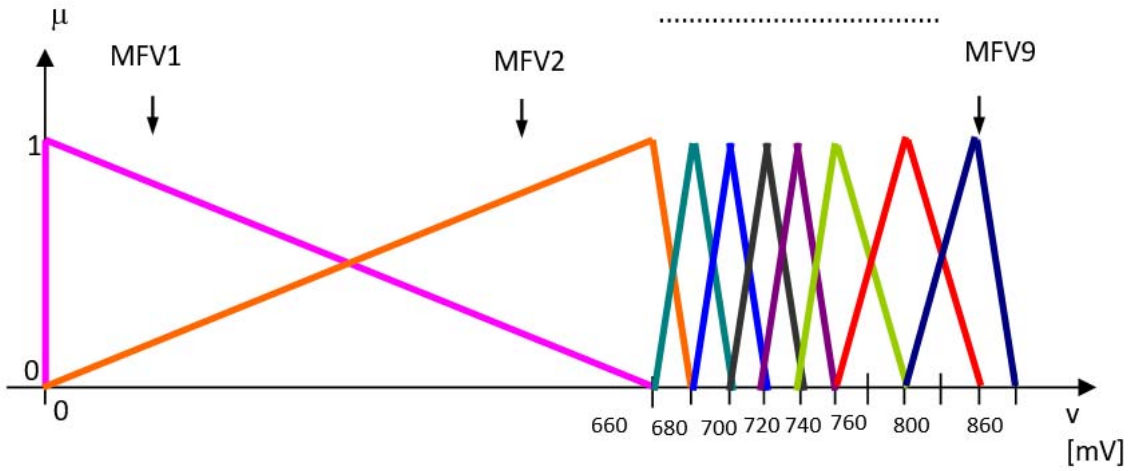
o Creating the FLS

Exercise 1

Implement the FLS with one input and one output, using *Fuzzy Logic Designer.* The specifications are:

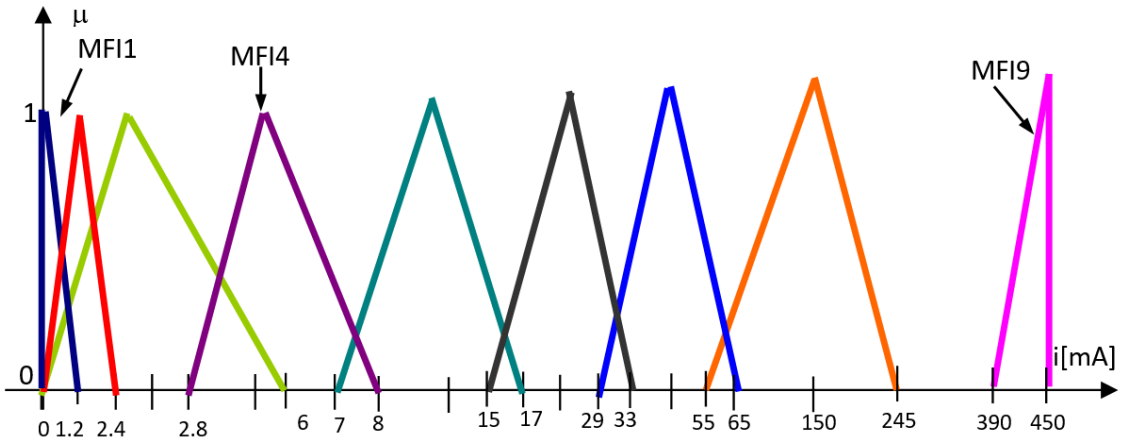
- Input variable v, 9 triangular fuzzy sets, denoted *MFVk, k* = 1, 2, ..., 9, with the following parameters:

MFV1 - [0 0 680] MFV2 - [0 680 690] MFV3 - [680 690 700]
MFV4 - [690 700 720] MFV5 - [700 720 740] MFV6 - [720 740 760]
MFV7 - [740 760 800] MFV8 - [760 800 860] MFV9 - [800 860 860]



- output variable i, 9 triangular fuzzy sets, denoted *MFIk*, k = 1, 2, ..., 9, with the following parameters:

MFI1 - [0 0 1.2] MFI2 - [0 1.2 2.4] MFI3 - [0 2.4 6]
MFI4 - [2.8 6 8] MFI5 - [7 12 17] MFI6 - [15 24 33]
MFI7 - [29 47 65] MFI8 - [55 150 245] MFI9 - [390 450 450]



- rule base with 9 rules as: *If v is MFVk then i is MFIk, k* = 1, 2, ..., 9

Save the FLS under the name *"diode.fis".*

Visualize the input-output curve (voltage-current). Set the number of plot points to 100. Compare it to the original characteristic.

Exercise 2

Test the fuzzy system for a series of crip input values:
v = {0 mV; 650 mV; 675 mV; 700 mV; 725 mV; 750 mV; 775 mV; 800 mV; 850 mV}. Write down the output values.

Exercise 3

Write a script that plots the current-voltage characteristic, using *readfis*, *evalfis*, *plot*.
Note: for *evalfis,* set *NPts* = 10000

Exercise 4

Identify the areas of the plot in which the approximation errors are high. Change the initial FLS to reduce these errors. Save the new FLS. Display the initial and improved current-voltage characteristic on the same plot.

6. Subtractive clustering. Modelling of a two-variable nonlinear function based on a data set.

Objectives: understanding the concept and methods for data clustering, understanding the meaning of class and class centre, understanding how the precision of a model is determined.

Note: MATLAB/Simulink is accessed online (https://matlab.mathworks.com/), by logging in with the MS Teams student credentials (surname.name@student.utcluj.ro).

Terms and abbreviations: *crisp clustering, subtractive clustering, Fuzzy C-Means.*

o Data clustering

Data clustering is a fundamental method for data analysis, where the purpose is to identify the natural data partitions, when having a large set of data. Data clustering is an unsupervised method, as the desired output (number of clusters, membership of an object to a certain cluster) is not known. Typical applications of data clustering include: shape recognition, vector quantization, image segmentation, function approximation, data mining.

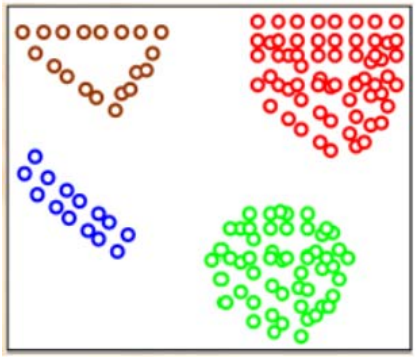
Data clustering is done using a set of features that describe each object of the data set. The result of data clustering is a fixed structure, made of clusters (classes). For each cluster, the following information is known:

- the cluster's location – the coordinates of the cluster's centre
- the cluster's shape
- the membership degree of each object to the cluster

The data partition has the following properties:

- homogeneity inside a cluster - the objects of a cluster should be as resembling as possible
- heterogeneity between clusters - objects belonging to different clusters should be as different as possible

A common measure for the resemblance between objects is the Euclidian distance, illustrated in the image below, where objects belonging to a certain cluster have the same colour.



The data to be clustered is represented as *N*-dimensional vectors:

$$X_i = [x_{i1}, x_{i2}, \dots, x_{iN}], X_i \in \mathbb{R}^N, i = \overline{1, \dots, M}$$

where *N* – number of features for each object, *M* – number of objects (size of dataset).

$$X = \begin{bmatrix} x_{11} & x_{12} \dots & x_{1N} \\ x_{21} & x_{22} \dots & x_{2N} \\ \dots & \dots & \dots \\ x_{M1} & x_{M2} \dots & x_{MN} \end{bmatrix}$$

The objective of clustering is to find *K* vectors that are the centres of the *K* clusters resulted after data clustering:

$$c_k = [x_{k1}, x_{k2}, \dots, x_{kN}], k = \overline{1, \dots, C}$$

o Types of data clustering

In ***crisp*** or ***hard*** or ***all-or-nothing clustering***, the membership degree of any object to a cluster is either 0 or 1. Each object belongs to a cluster, there are no empty clusters and no cluster can contain all the objects.

In real life, hard clustering is rarely suitable, thus a more "relaxed" clustering method is used, where each object can belong to several different clusters, with a membership degree between 0 and 1. This type of clustering can be:

- fuzzy – Fuzzy C-Means, subtractive clustering
- probabilistic.

o Subtractive clustering

Subtractive clustering finds the number of clusters and their centres in a set of data. The influence area of the centre of the cluster must be specified, for each dimension.

The steps for subtractive clustering are:

1. Assume that every data point is a potential cluster centre. Compute the probability of this assumption based on the density of the surrounding points.

2. Select the point with the highest potential as the first cluster centre.

3. Eliminate all the points around the centre found in the previous step (according to its influence area), to find the next cluster and its centre.

4. Repeat step 3 until all the points are part of at least one cluster.

o Modelling a two variable, non-linear function. Application - functional modelling of a transconductance amplifier

Download "*Data_Templates.zip*" and place the archive (using *drag-and-drop*) in the current directory of MATLAB. Double click to unzip and view the contents of the folder.

https://drive.google.com/file/d/1fIvyWwr6X3XWpspKAoqf_-oWoxgQg75s/view?usp=sharing

The archive contains 6 files: 3 files with the datasets (*.mat* file extension), and 3 scripts (*.m* file extension).

To build the model that implements the *gain = f(frequency, temperature)* function, 3 datasets are used:

- "*gendata*" – 179 samples, to generate the initial fuzzy system

- "*antdata*" – 35717 samples, to train the initial fuzzy system
- "*verdata*" – 497 samples, to verify the fuzzy system during training and detect overfitting

The initial ranges of values for the input variables are:

frequency_ini: [1 Hz, 10 MHz]

temperature_ini: [-55, +125]℃

To reduce the range of values and to work with positive values, two transformations are performed:

frequency = *log(frequency_ini)*, so the frequency is in [0, 7]

temperature = *temperature_ini* + 60, so the temperature is in [5, 185]℃

The datasets are matrices, each row represents the values for [*frequency, temperature, gain*].

Exercise 1

Analyse the structure of the 3 datasets, by loading them into the workspace (double-click on each *.mat* file).

Exercise 2

To **generate the initial fuzzy system,** use "*generate_aft.m*" template, in which you add code snippets, according with the instructions in the file.

Generate the initial fuzzy system, using the *genfis2* function. Examine the properties of the fuzzy system, using *Fuzzy Logic Designer*.

What is the value of *radii*?

How many clusters were determined?

How many rules are in the rule base?

What is the connection between the number of clusters, number of rules and the structure of each rule?

What type of fuzzy sets are used for the input variables? What are the expressions of the output fuzzy sets?

What are the centres of the clusters? Visualise the control surface of the initial fuzzy system.

Exercise 3

To **evaluate the precision of the initial fuzzy model,** use the "*errors.m*" template, in which you add code snippets, according with the instructions in the file.

Compute and save in a file:

- the maximum value of the absolute error:

$$absolute\_error_i = |reference\_gain_i - fuzzy\_gain_i|$$

- the maximum value of the relative error:

$$relative\_error_i = \frac{|reference\_gain_i - fuzzy\_gain_i|}{reference\_gain_i}$$

- mean percentual error

$$mean\_percentual\_error = \frac{1}{N}\sum_{i=1}^{N}\frac{|reference\_gain_i - fuzzy\_gain_i|}{refernce\_gain_i}$$

Exercise 4

To **train the initial fuzzy system**, use "*training_aft.m*" template, in which you add code snippets, according with the instructions in the file.

Train the initial fuzzy system, using the *anfis* function.

What type of error does *anfis* use?

Analyse the evolution of errors for the training and test datasets. Does overfitting occur?

Do you think increasing the number of training epochs is necessary? Justify your answer.

Visualise the control surface of the trained fuzzy system. Compare it against the control surface of the initial system.

Identify the changes in the input and output fuzzy sets.

Exercise 5

To **evaluate the precision of the final fuzzy model**, use the "*errors.m*" template, in which you add code snippets, according with the instructions in the file.

Compute and save in a file:

- the maximum value of the absolute error:

$$absolute\_error_i = |reference\_gain_i - fuzzy\_gain_i|$$

- the maximum value of the relative error:

$$relative\_error_i = \frac{|reference\_gain_i - fuzzy\_gain_i|}{reference\_gain_i}$$

- mean percentual error

$$mean\_percentual\_error = \frac{1}{N}\sum_{i=1}^{N} \frac{|reference\_gain_i - fuzzy\_gain_i|}{refernce\_gain_i}$$

Compares these values against the ones obtained for the initial fuzzy system.