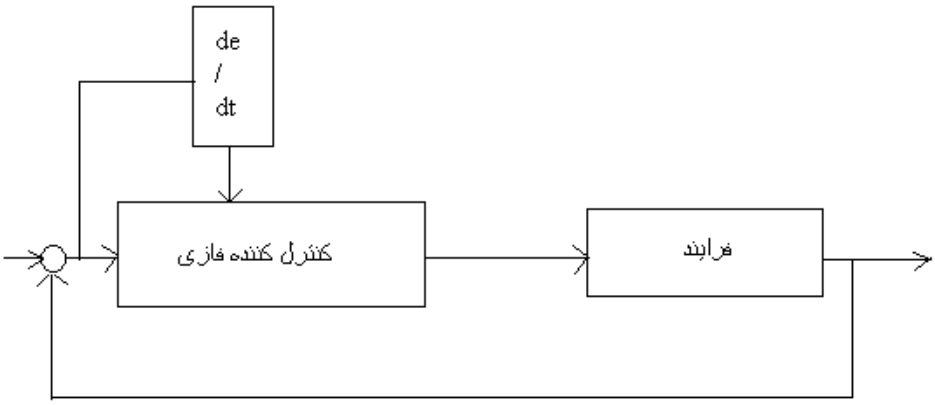
تمرینات Take Home درس سیستم های فازی

وحید مواجی 83205947

تمرین شماره 1:

برای کنترل سطح آب از یک سیستم کنترل کننده بصورت زیر استفاده می کنیم:



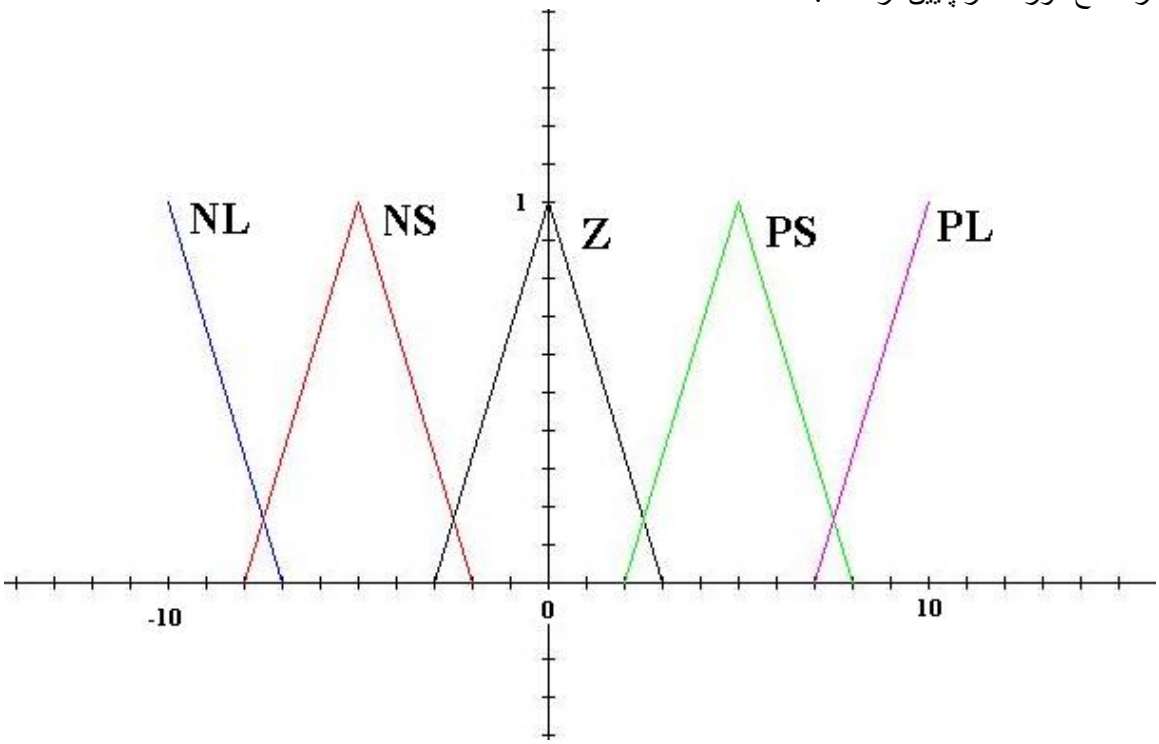
در این کنترل کننده سه قانون کلی زیر را داریم:

1. اگر خطا(e) و تغییرات خطا (Δe) هر دو صفر باشند، کنترل قبلی مساعد بوده و باید حفظ شود.
2. اگر خطا با نرخ مناسبی به سمت صفر شدن می رود، کنترل قبلی مساعد بوده و باید حفظ شود.
3. اگر خطا در جهتی غیر از صفر شدن تغییر می کند، میزان کنترل را با علامت و مقدار خطا و نرخ تغییرات آن در نظر می گیریم.

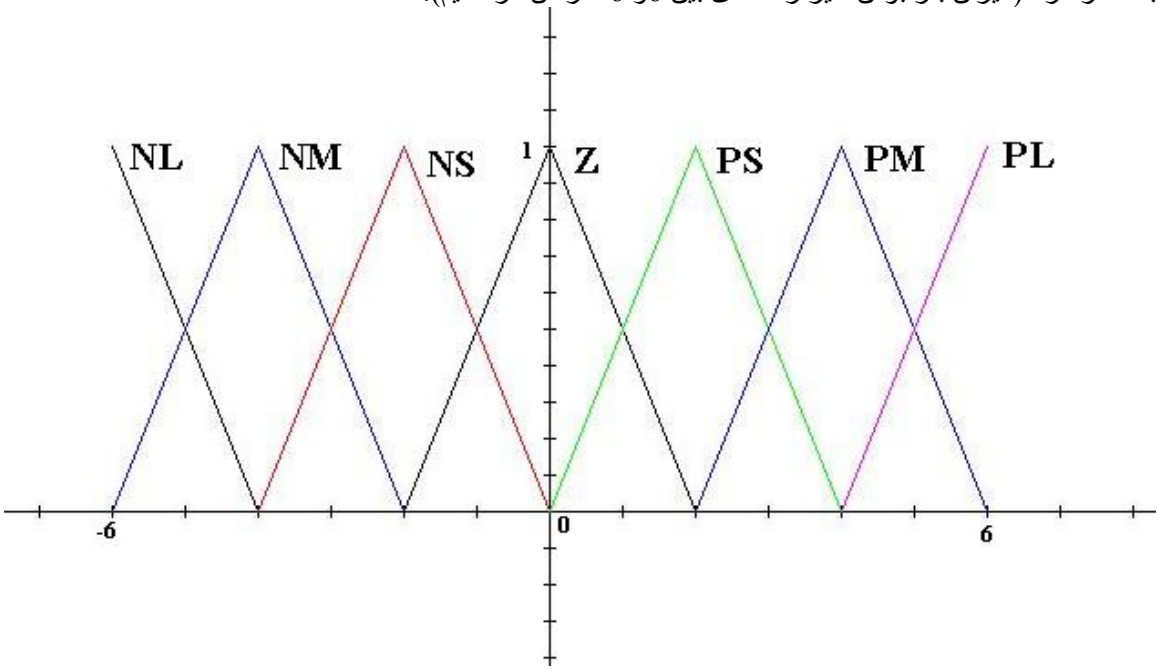
با توجه به این سه قانون کلی، می توان یک پایگاه قواعد تشکیل داد. اگر خطا و تغییرات آن را هر کدام به پنج مقدار فازی {NL, NS, Z, PS, PL} به معنی {زیاد مثبت، کم مثبت، صفر، کم منفی، زیاد منفی} و خروجی کنترل کننده به هفت مقدار فازی {NL, NM, NS, Z, PS, PM, PL} به معنی {زیاد مثبت، متوسط مثبت، کم مثبت، صفر، کم منفی، متوسط منفی، زیاد منفی} افراز کنیم، پایگاه قواعد زیر را می توان در نظر گرفت:

e / Δe	**NL**	**NS**	**Z**	**PS**	**PL**
NL	PL	PL	PM	PS	Z
NS	PL	PM	PS	Z	NS
Z	PM	PS	Z	NS	NM
PS	PS	Z	NS	NM	NL
PL	Z	NS	NM	NL	NL

نمودار مجموعه های فازی برای مقادیر خطا و تغییرات خطا که بین 10و 10- تغییر می کنند می تواند بصورت زیر باشد که برای خطا، مقادیر مثبت یعنی میزان آب از سطح مورد نظر بالاتر است و مقادیر منفی یعنی میزان آب از سطح مورد نظر پایین تر است:



نمودار مجموعه های فازی برای کنترل شیر A (میزان باز بودن شیر A) را می توان به صورت زیر نشان داد که صفر یعنی در باز بودن کنونی شیر تغییر حاصل نشود، مقادیر مثبت یعنی شیر بازتر شود و مقادیر منفی یعنی شیر بسته تر گردد(میزان باز بودن شیر را عددی بین 6و 6- فرض کرده ایم):



برای استنتاج از مدل ممدانی استفاده می کنیم: در مدل ممدانی قواعد به فرم زیر می باشند:

If x is A_1 and y is B_1 then z is C_1
If x is A_2 and y is B_2 then z is C_2

حال اگر متغیر ورودی x, y با مجموعه های مقدم هر دو قاعده فوق همپوشانی نسبی داشته باشند، خروجی z باید از مستنتج از هر دو قاعده باشد که بدین منظور از ترکیب ماکزیمم – مینیمم یا ماکزیمم – ضرب استفاده می شود. بنابراین می توانیم قدرت هر یک از قاعده های فوق را اینگونه بدست آوریم:

$\alpha_1 = \min(\mu_{A1}(x_0), \mu_{B1}(y_0))$
$\alpha_2 = \min(\mu_{A2}(x_0), \mu_{B2}(y_0))$

بعد از بدست آوردن قدرت هریک از قاعده های شرکت کننده در رقابت می توان خروجی قاعده ها را به صورت زیر بدست آورد:

$\mu_{C1}(z) = \min(\alpha_1 , \mu_{C1})$
$\mu_{C2}(z) = \min(\alpha_2 , \mu_{C2})$

نتیجه کلی ترکیب ماکزیمم – مینیمم بصورت زیر می باشد:

$\mu_C(z) = \max (\min(\alpha_1 , \mu_{C1}) , \min(\alpha_2 , \mu_{C2}))$

در مرحله غیر فازی سازی از چهار روش استفاده کرده ایم:

1. روش مرکز ثقل:

$Z = (\sum z_i . \mu_C(z_i)) \ / (\sum \mu_C(z_i))$

2. روش مرکز مجموع ها:

$Z = (\sum z_i . \sum \mu_{C'}(z_i)) \ / (\sum \sum \mu_{C'}(z_i))$

3. روش ارتفاع:

$Z = (\sum \alpha_k . z^{(k)}) \ / (\sum \alpha_k)$

4. روش متوسط ماکزیمم:

Z = (INF M + SUP M) / 2
Where M = $\{z \mid \mu_C(z) = hgt(Z)\}$

در این مسأله برای $0 \le t \le 2$ داریم $e = 0, \Delta e = 0$ و برای $2 \le t \le 10$ داریم $e = 1.25(t - 2), \Delta e = 1.25$.
برای همه مقادیر t با استفاده از روش ممدانی و چهار روش غیرفازی کردن که ذکر شد مقادیر بازبودن شیر را بدین صورت بدست آوردیم:

t	e	Δe	شیر مرکز ثقل	شیر مرکز مجموع	شیر ارتفاع	شیر متوسط ماکزیمم
0.0	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.1	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.2	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.3	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.4	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.5	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.6	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5

0.7	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.8	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
0.9	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.0	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.1	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.2	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.3	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.4	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.5	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.6	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.7	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.8	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
1.9	0.0	0.0	1.3784961E-7	1.3784961E-7	0.0	6.67927E-5
2.0	2.9802322E-7	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.1	0.12500018	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.2	0.25000006	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.3	0.37499994	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.4	0.49999982	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.5	0.6249997	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.6	0.7499996	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.7	0.87499946	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.8	0.99999934	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
2.9	0.99999934	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
3.0	1.249999	1.25	1.2064923E-7	1.2064923E-7	0.0	1.3357401E-4
3.1	1.374999	1.25	-1.635334E-7	-1.635334E-7	0.0	1.3357401E-4
3.2	1.4999988	1.25	-1.9650892E-8	-1.9650892E-8	0.0	-0.004866421
3.3	1.6249988	1.25	2.0858725E-7	2.0858725E-7	0.0	1.3357401E-4
3.4	1.7499986	1.25	-2.247216E-7	-2.247216E-7	0.0	1.3357401E-4
3.5	1.8749986	1.25	7.114343E-9	7.114343E-9	0.0	-0.004866421
3.6	1.9999983	1.25	2.1379867E-7	2.1379867E-7	0.0	1.3357401E-4
3.7	2.1249983	1.25	-0.28142256	-0.28142256	-0.24999665	1.3357401E-4
3.8	2.249998	1.25	-0.5348676	-0.5348676	-0.4999962	-0.004866421
3.9	2.374998	1.25	-0.77142984	-0.77142984	-0.7499962	1.3357401E-4
4.0	2.4999979	1.25	-1.000001	-1.000001	-0.9999957	1.3357401E-4
4.1	2.624998	1.25	-1.2285614	-1.2285614	-1.2499962	-1.9998648
4.2	2.749998	1.25	-1.4651177	-1.4651177	-1.4999962	-2.0048647
4.3	2.8749979	1.25	-1.7185794	-1.7185794	-1.7499957	-1.9998648
4.4	2.9999976	1.25	-1.9999951	-1.9999951	-1.9999952	-1.9998648
4.5	3.1249976	1.25	-1.999997	-1.999997	-2.0	-2.0048647
4.6	3.2499976	1.25	-2.0000062	-2.0000062	-2.0	-1.9998648
4.7	3.3749974	1.25	-1.9999955	-1.9999955	-2.0	-1.9998648
4.8	3.4999971	1.25	-1.9999988	-1.9999988	-2.0	-2.0048647
4.9	3.6249971	1.25	-2.0000055	-2.0000055	-2.0	-1.9998648
5.0	3.7499971	1.25	-1.9999967	-1.9999967	-2.0	-1.9998648
5.1	3.874997	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.2	3.9999967	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.3	4.1249967	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.4	4.2499967	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648

5.5	4.374996	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.6	4.499996	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.7	4.624996	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.8	4.749996	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
5.9	4.874996	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.0	4.9999957	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.1	5.1249957	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.2	5.2499957	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.3	5.374995	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.4	5.499995	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.5	5.624995	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.6	5.749995	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.7	5.874995	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.8	5.9999948	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
6.9	6.1249948	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
7.0	6.2499948	1.25	-1.9999986	-1.9999986	-2.0	-1.9998648
7.1	6.3749943	1.25	-1.999996	-1.999996	-2.0	-1.9998648
7.2	6.4999943	1.25	-2.0000055	-2.0000055	-2.0	-2.0048647
7.3	6.6249943	1.25	-1.9999989	-1.9999989	-2.0	-1.9998648
7.4	6.7499943	1.25	-1.9999965	-1.9999965	-2.0	-1.9998648
7.5	6.8749943	1.25	-2.0000064	-2.0000064	-2.0	-2.0048647
7.6	6.999994	1.25	-1.999998	-1.999998	-2.0	-1.9998648
7.7	7.124994	1.25	-2.281401	-2.281401	-2.2499876	-1.9998648
7.8	7.249994	1.25	-2.5348601	-2.5348601	-2.4999878	-2.0048647
7.9	7.3749933	1.25	-2.7714121	-2.7714121	-2.7499866	-1.9998648
8.0	7.4999933	1.25	-2.999965	-2.999965	-2.9999866	-1.9998648
8.1	7.6249933	1.25	-3.2285454	-3.2285454	-3.2499866	-3.9998817
8.2	7.749994	1.25	-3.4650993	-3.4650993	-3.4999876	-4.004881
8.3	7.8749943	1.25	-3.7185278	-3.7185278	-3.7499888	-3.9998796
8.4	7.9999948	1.25	-3.9999762	-3.9999762	-3.9999893	-3.9998786
8.5	8.124995	1.25	-3.999981	-3.999981	-4.0	-4.0048776
8.6	8.249996	1.25	-3.9999976	-3.9999976	-4.0	-3.9998765
8.7	8.374996	1.25	-4.000004	-4.000004	-4.0	-3.9998755
8.8	8.499996	1.25	-3.9999855	-3.9999855	-4.0	-4.0048747
8.9	8.624997	1.25	-3.9999974	-3.9999974	-4.0	-3.9998736
9.0	8.749998	1.25	-3.999984	-3.999984	-4.0	-3.9998727
9.1	8.874998	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.2	8.999998	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.3	9.124999	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.4	9.25	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.5	9.375	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.6	9.5	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.7	9.625001	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.8	9.750002	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727
9.9	9.875002	1.25	-3.9999857	-3.9999857	-4.0	-3.9998727

تمرین شماره 2:

برای محاسبه حاصلضرب 2 عدد فازی A, B می توانیم از مقاطع آلفای آنها استفاده کنیم برای این منظور فرض می کنیم مقطع α – A, B به ترتیب [c, d] و [a, b] باشد، آنگاه داریم:

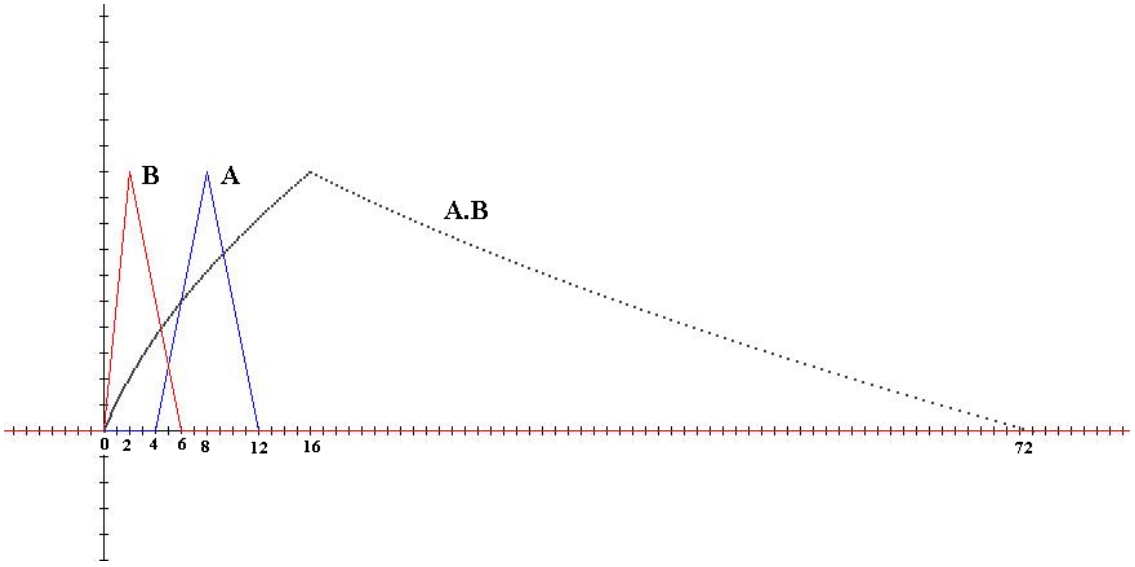
[a, b] * [c, d] = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]

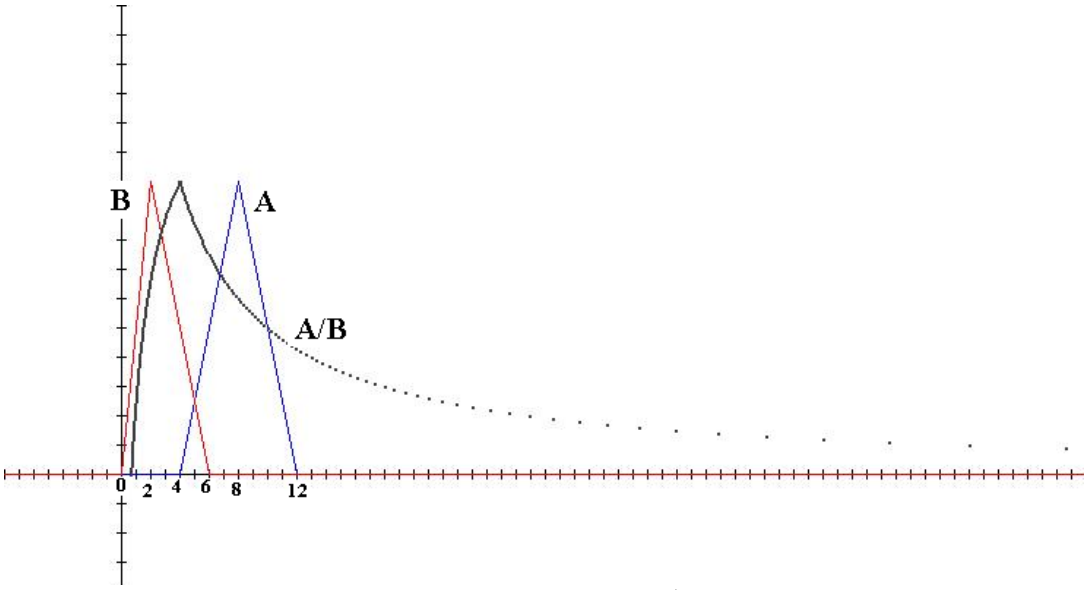
با یک برنامه ساده می توان به ازای $0 \leq \alpha \leq 1$ ، مقاطع آلفا را بدست آورده و به ازای هر کدام از آنها دو نقطه یعنی نقاط انتهایی بازه حاصلضرب که از روش بالا بدست می آید را حساب کرد. با وصل کردن این نقاط به یکدیگر، نمودار حاصلضرب فازی بدست می آید.

عمل تقسیم هم مشابه عمل ضرب است با این تفاوت:

[a, b] / [c, d] = [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)]

نتیجه این دو عمل با برنامه ای که نوشته شد، بدین صورت می باشد:

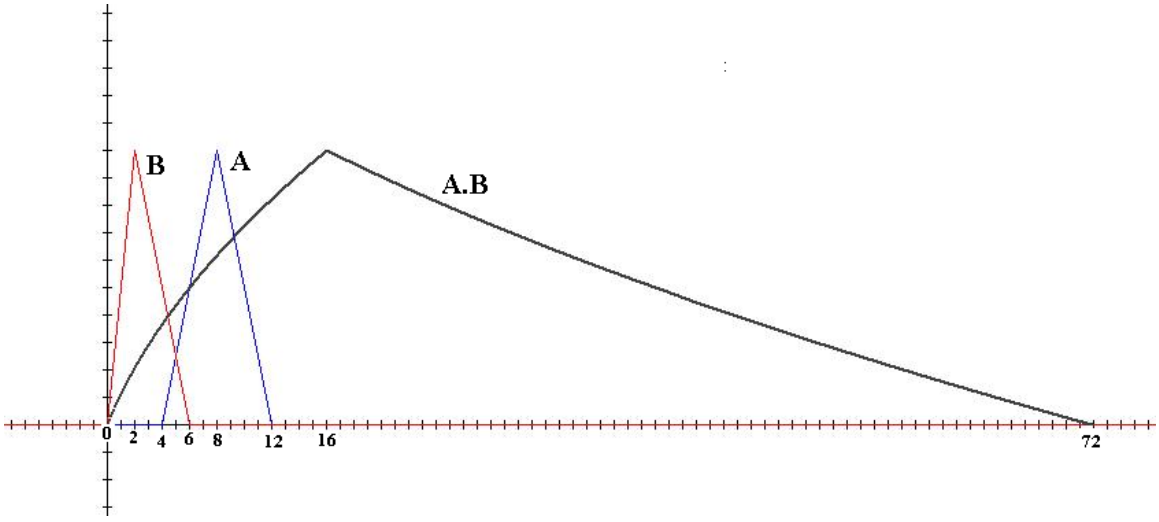


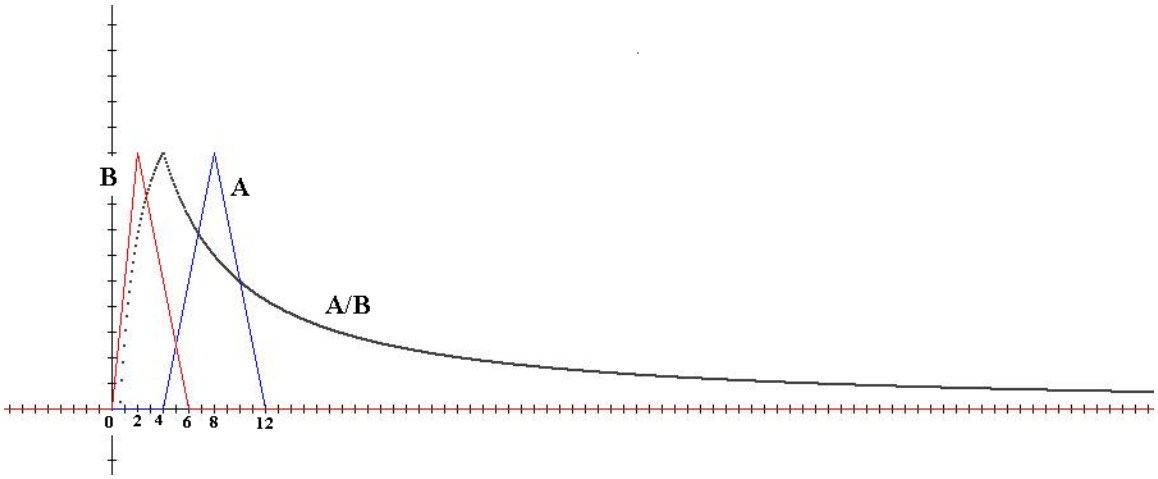


همچنین اعداد فازی را می توان با روش اصل توسعه در هم ضرب یا بر هم تقسیم کرد. در این روش اعداد بصورت زیر در هم ضرب یا بر هم تقسیم می شوند:

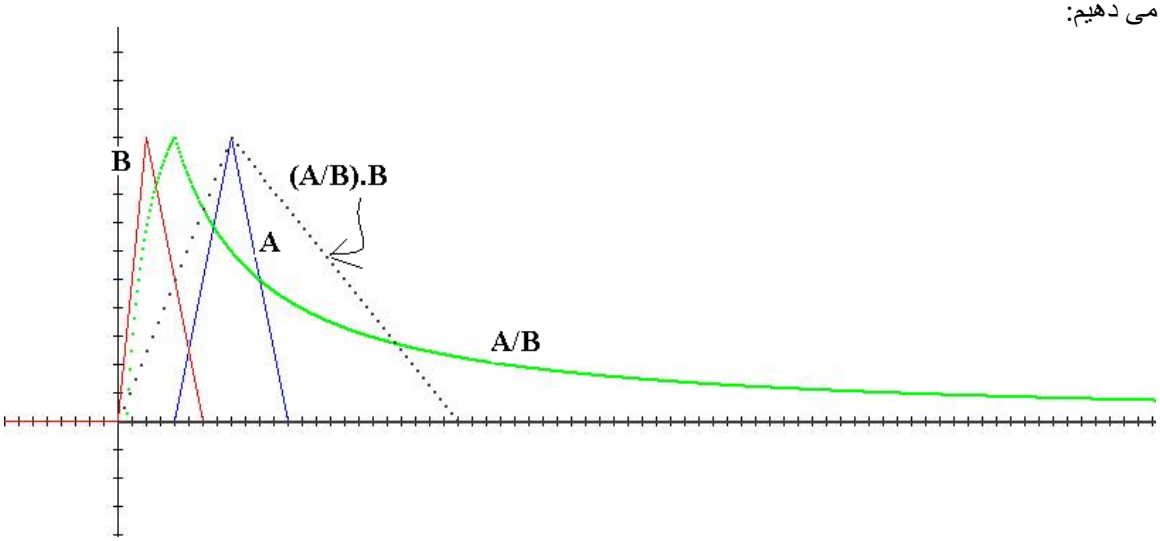
$\mu_{A.B}(z) = \sup \min[\mu_A(z), \mu_B(z)]$ for all x, y that x.y = z
$\mu_{A/B}(z) = \sup \min[\mu_A(z), \mu_B(z)]$ for all x, y that x/y = z

باز هم با یک برنامه ساده می توان این ضرب و تقسیم را انجام داد دراین قسمت به جای اینکه روی α حرکت کنیم، روی محور x یا y حرکت می کنیم و از روی آنها مقادیر تابع عضویت حاصلضرب یا حاصل تقسیم را بدست می آوریم.





حال می خواهیم مقایسه کنیم که آیا (A/B).B = A می باشد یا نه؟ با استفاده از همان برنامه قبلی این تست را انجام می دهیم:



ملاحظه می کنیم که این تساوی برقرار نمی باشد.

برای انجام این تمرین ها از زبان جاوا برای برنامه نویسی استفاده کرده ایم که کد برنامه نوشته شده در زیر می آید. البته می شد از matlab هم استفاده کرد ولی ما ترجیح دادیم از اول خودمان یک کد بنویسیم.

```
import java.awt.Color;
import java.awt.Graphics;
import java.util.HashMap;
import java.util.Iterator;

/**
 * Date: Jun 9, 2005
 * Time: 11:52:17 PM
 *
 * @author Vahid Mavaji
 */
public class FuzzySetTri {

    public int a;
    public int b;
    public int c;
    public Color color;
    Graphics g;

    final Color COLOR = Color.black;
    final int DELTA = 3;
    final int WIDTH = 1000;
    final int HEIGTH = 700;
    final int X_CENTER = 480;
    final int Y_CENTER = 500;
    final int DELTA_X = 40;
    final int DELTA_Y = 20;

    public FuzzySetTri(int a, int b, int c, Graphics g) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.g = g;
    }
```

```
    public FuzzySetTri(int a, int b, int c, Color color, Graphics g) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.color = color;
        this.g = g;
    }

    float getMu(float x) {
        float result = 0;
        if (x >= a && x <= b) {
            result = (x - a) / (b - a);
        } else if (x > b && x <= c) {
            result = (x - c) / (b - c);
        }
        return result;
    }

    void drawAxis() {
        g.setColor(COLOR);
        g.drawLine(0, Y_CENTER, WIDTH, Y_CENTER);
        for (int i = 0; i < WIDTH; i += DELTA_X) {
            g.drawLine(i, Y_CENTER - DELTA, i, Y_CENTER + DELTA);
        }

        g.drawLine(X_CENTER, 0, X_CENTER, HEIGTH);
        for (int i = 0; i < HEIGTH; i += DELTA_Y) {
            g.drawLine(X_CENTER - DELTA, i, X_CENTER + DELTA, i);
        }
    }

    void drawSet() {
        g.setColor(color);
//        g.drawLine(0, Y_CENTER,
//                a * DELTA_X + X_CENTER, Y_CENTER);
        if (a != b) {
            g.drawLine(a * DELTA_X + X_CENTER, Y_CENTER,
                    b * DELTA_X + X_CENTER, Y_CENTER - 10 * DELTA_Y);
        }
        if (b != c) {
            g.drawLine(b * DELTA_X + X_CENTER, Y_CENTER - 10 * DELTA_Y,
                    c * DELTA_X + X_CENTER, Y_CENTER);
        }
//        g.drawLine(c * DELTA_X + X_CENTER, Y_CENTER,
//                WIDTH, Y_CENTER);
    }

    void add_aCut(FuzzySetTri fuzzySet2, Color color) {
        float alpha;
        for (alpha = 0; alpha <= 1; alpha += 0.01) {
            float A_alpha1 = (b - a) * alpha + a;
            float A_alpha2 = (alpha - 1) * (b - c) + b;

            float B_alpha1 = (fuzzySet2.b - fuzzySet2.a) * alpha + fuzzySet2.a;
            float B_alpha2 = (alpha - 1) * (fuzzySet2.b - fuzzySet2.c) + fuzzySet2.b;

            float x1 = A_alpha1 + B_alpha1;
            float x2 = A_alpha2 + B_alpha2;

            g.setColor(color);
            g.drawOval((int) (x1 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
            g.drawOval((int) (x2 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
        }
    }

    HashMap ext_princ(FuzzySetTri fuzzySet2, Color color, int op) {
        g.setColor(color);
        HashMap map = new HashMap();
```

```
float s = 0;
Float S;
float min, mi, mj;
for (float i = a; i <= c; i += 0.01) {
    if (i < b) {
        mi = (i - a) / (b - a);
    } else {
        mi = (i - b) / (b - c) + 1;
    }

    for (float j = fuzzySet2.a; j <= fuzzySet2.c; j += 0.01) {
        if (j < fuzzySet2.b) {
            mj = (j - fuzzySet2.a) / (fuzzySet2.b - fuzzySet2.a);
        } else {
            mj = (j - fuzzySet2.b) / (fuzzySet2.b - fuzzySet2.c) + 1;
        }

        min = (mi < mj) ? mi : mj;

        switch (op) {
            case 0:
                s = i + j;
                break;
            case 1:
                s = i - j;
                break;
            case 2:
                s = i * j;
                break;
            case 3:
                s = i / j;
                break;
        }

        S = new Float(s);

        if (!map.containsKey(S)) {
            map.put(S, new Float(min));
        } else {
            float min1 = ((Float) map.get(S)).floatValue();
            if (min > min1) {
                map.put(S, new Float(min));
            }
        }
    }
}

HashMap result = new HashMap();
HashMap xymap = new HashMap();
for (Iterator iter = map.keySet().iterator(); iter.hasNext();) {
    Float f = (Float) iter.next();
    float x = f.floatValue();
    float y = ((Float) map.get(f)).floatValue();


    Integer xx = new Integer((int) (x * DELTA_X + X_CENTER));
    Integer yy = new Integer((int) (Y_CENTER - y * 10 * DELTA_Y));
    if (!xymap.containsKey(xx)) {
        xymap.put(xx, yy);
        result.put(new Float((xx.intValue() - X_CENTER) / DELTA_X),
                new Float((Y_CENTER - yy.intValue()) / (10 * DELTA_Y)));
    } else {
        if (yy.intValue() < ((Integer) xymap.get(xx)).intValue()) {
            xymap.put(xx, yy);
            result.put(new Float((xx.intValue() - X_CENTER) / DELTA_X),
                    new Float((Y_CENTER - yy.intValue()) / (10 * DELTA_Y)));
        }
    }
}

for (Iterator iter = xymap.keySet().iterator(); iter.hasNext();) {
```

```
        Integer x = (Integer) iter.next();
        Integer y = (Integer) xymap.get(x);
        g.drawOval(x.intValue(), y.intValue(), 1, 1);
    }
    return result;
}

void ext_princ(HashMap xyMap1, Color color, int op) {
    g.setColor(color);
    HashMap map = new HashMap();
    float s = 0;
    Float S;
    float min, mi, mj;
    float j;

    for (float i = a; i <= c; i += 0.1) {
        if (i < b) {
            mi = (i - a) / (b - a);
        } else {
            mi = (i - b) / (b - c) + 1;
        }

        for (Iterator iter = xyMap1.keySet().iterator(); iter.hasNext();) {
            Float x = (Float) iter.next();
            mj = ((Float) xyMap1.get(x)).floatValue();
            j = x.floatValue();


            min = (mi < mj) ? mi : mj;

            switch (op) {
                case 0:
                    s = i + j;
                    break;
                case 1:
                    s = i - j;
                    break;
                case 2:
                    s = i * j;
                    break;
                case 3:
                    s = i / j;
                    break;
            }

            S = new Float(s);

            if (!map.containsKey(S)) {
                map.put(S, new Float(min));
            } else {
                float min1 = ((Float) map.get(S)).floatValue();
                if (min > min1) {
                    map.put(S, new Float(min));
                }
            }
        }
    }

    HashMap xymap = new HashMap();
    for (Iterator iter = map.keySet().iterator(); iter.hasNext();) {
        Float f = (Float) iter.next();
        float x = f.floatValue();
        float y = ((Float) map.get(f)).floatValue();

        Integer xx = new Integer((int) (x * DELTA_X + X_CENTER));
        Integer yy = new Integer((int) (Y_CENTER - y * 10 * DELTA_Y));
        if (!xymap.containsKey(xx)) {
            xymap.put(xx, yy);
        } else {
            if (yy.intValue() < ((Integer) xymap.get(xx)).intValue()) {
                xymap.put(xx, yy);
```

```
                }
            }
        }

        for (Iterator iter = xymap.keySet().iterator(); iter.hasNext();) {
            Integer x = (Integer) iter.next();
            Integer y = (Integer) xymap.get(x);
            g.drawOval(x.intValue(), y.intValue(), 1, 1);
        }
    }

    void sub_aCut(FuzzySetTri fuzzySet2, Color color) {
        float alpha;
        g.setColor(color);
        for (alpha = 0; alpha <= 1; alpha += 0.01) {
            float A_alpha1 = (b - a) * alpha + a;
            float A_alpha2 = (alpha - 1) * (b - c) + b;

            float B_alpha1 = (fuzzySet2.b - fuzzySet2.a) * alpha + fuzzySet2.a;
            float B_alpha2 = (alpha - 1) * (fuzzySet2.b - fuzzySet2.c) + fuzzySet2.b;

            float x1 = A_alpha1 - B_alpha2;
            float x2 = A_alpha2 - B_alpha1;

            g.drawOval((int) (x1 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
            g.drawOval((int) (x2 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
        }
    }

    void mul_aCut(FuzzySetTri fuzzySet2, Color color) {
        float alpha;
        g.setColor(color);
        for (alpha = 0; alpha <= 1; alpha += 0.01) {
            float A_alpha1 = (b - a) * alpha + a;
            float A_alpha2 = (alpha - 1) * (b - c) + b;

            float B_alpha1 = (fuzzySet2.b - fuzzySet2.a) * alpha + fuzzySet2.a;
            float B_alpha2 = (alpha - 1) * (fuzzySet2.b - fuzzySet2.c) + fuzzySet2.b;

            float x1 = A_alpha1 * B_alpha1;
            if (A_alpha1 * B_alpha2 < x1) {
                x1 = A_alpha1 * B_alpha2;
            }
            if (A_alpha2 * B_alpha1 < x1) {
                x1 = A_alpha2 * B_alpha1;
            }
            if (A_alpha2 * B_alpha2 < x1) {
                x1 = A_alpha2 * B_alpha2;
            }

            float x2 = A_alpha1 * B_alpha1;
            if (A_alpha1 * B_alpha2 > x2) {
                x2 = A_alpha1 * B_alpha2;
            }
            if (A_alpha2 * B_alpha1 > x2) {
                x2 = A_alpha2 * B_alpha1;
            }
            if (A_alpha2 * B_alpha2 > x2) {
                x2 = A_alpha2 * B_alpha2;
            }

            g.drawOval((int) (x1 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
            g.drawOval((int) (x2 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
        }
    }

    void div_aCut(FuzzySetTri fuzzySet2, Color color) {
```

```
        float alpha;
        g.setColor(color);
        for (alpha = 0; alpha <= 1; alpha += 0.01) {
            float A_alpha1 = (b - a) * alpha + a;
            float A_alpha2 = (alpha - 1) * (b - c) + b;

            float B_alpha1 = (fuzzySet2.b - fuzzySet2.a) * alpha + fuzzySet2.a;
            float B_alpha2 = (alpha - 1) * (fuzzySet2.b - fuzzySet2.c) + fuzzySet2.b;

            float x1 = A_alpha1 / B_alpha1;
            if (A_alpha1 / B_alpha2 < x1) {
                x1 = A_alpha1 / B_alpha2;
            }
            if (A_alpha2 / B_alpha1 < x1) {
                x1 = A_alpha2 / B_alpha1;
            }
            if (A_alpha2 / B_alpha2 < x1) {
                x1 = A_alpha2 / B_alpha2;
            }

            float x2 = A_alpha1 / B_alpha1;
            if (A_alpha1 / B_alpha2 > x2) {
                x2 = A_alpha1 / B_alpha2;
            }
            if (A_alpha2 / B_alpha1 > x2) {
                x2 = A_alpha2 / B_alpha1;
            }
            if (A_alpha2 / B_alpha2 > x2) {
                x2 = A_alpha2 / B_alpha2;
            }

            g.drawOval((int) (x1 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
            g.drawOval((int) (x2 * DELTA_X + X_CENTER), (int) (Y_CENTER - alpha * 10 *
DELTA_Y), 1, 1);
        }
    }

}
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.AWTEvent;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.WindowEvent;
import java.util.HashMap;

/**
 * Date: Jun 9, 2005
 * Time: 1:41:54 PM
 *
 * @author Vahid Mavaji
 */
public class Frame1 extends JFrame {
    JPanel contentPane;


    /**
     * Construct the frame
     */
    public Frame1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Component initialization
     */
    private void jbInit() throws Exception {

//setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.getResource("[Your
Icon]")));
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(null);
        this.setSize(new Dimension(1000, 700));
        this.setTitle("Frame Title");
        this.setBackground(Color.white);
    }

    /**
     * Overridden so we can exit when window is closed
     */
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }

    public void paint(Graphics g) {
        takeHome1(g);
//        takeHome2(g);
    }

    public void takeHome1(Graphics g) {
        FuzzySetTri NL = new FuzzySetTri(-10, -10, -7, Color.blue, g);
        FuzzySetTri NS = new FuzzySetTri(-8, -5, -2, Color.red, g);
        FuzzySetTri Z = new FuzzySetTri(-3, 0, 3, Color.BLACK, g);
        FuzzySetTri PS = new FuzzySetTri(2, 5, 8, Color.green, g);
        FuzzySetTri PL = new FuzzySetTri(7, 10, 10, Color.magenta, g);

//        NL.drawAxis();
//
//        NL.drawSet();
//        NS.drawSet();
//        Z.drawSet();
//        PS.drawSet();
//        PL.drawSet();

        FuzzySetTri NLv = new FuzzySetTri(-6, -6, -4, Color.black, g);
        FuzzySetTri NMv = new FuzzySetTri(-6, -4, -2, Color.blue, g);
        FuzzySetTri NSv = new FuzzySetTri(-4, -2, 0, Color.red, g);
        FuzzySetTri Zv = new FuzzySetTri(-2, 0, 2, Color.black, g);
        FuzzySetTri PSv = new FuzzySetTri(0, 2, 4, Color.green, g);
        FuzzySetTri PMv = new FuzzySetTri(2, 4, 6, Color.blue, g);
        FuzzySetTri PLv = new FuzzySetTri(4, 6, 6, Color.magenta, g);

//        NLv.drawAxis();
//
//        NLv.drawSet();
//        NMv.drawSet();
//        NSv.drawSet();
//        Zv.drawSet();
//        PSv.drawSet();
//        PMv.drawSet();
//        PLv.drawSet();


        float dE = 0;
        float E = 0;
        for (float t = 0f; t <= 10; t += 0.1) {
```

```
            if (t <= 2) {
                E = 0;
                dE = 0;
            } else {
                E = 1.25f * (t - 2);
                dE = 1.25f;
            }
            float[][] alpha = new float[5][5];
            alpha[0][0] = Math.min(NL.getMu(E), NL.getMu(dE));
            alpha[0][1] = Math.min(NL.getMu(E), NS.getMu(dE));
            alpha[0][2] = Math.min(NL.getMu(E), Z.getMu(dE));
            alpha[0][3] = Math.min(NL.getMu(E), PS.getMu(dE));
            alpha[0][4] = Math.min(NL.getMu(E), PL.getMu(dE));

            alpha[1][0] = Math.min(NS.getMu(E), NL.getMu(dE));
            alpha[1][1] = Math.min(NS.getMu(E), NS.getMu(dE));
            alpha[1][2] = Math.min(NS.getMu(E), Z.getMu(dE));
            alpha[1][3] = Math.min(NS.getMu(E), PS.getMu(dE));
            alpha[1][4] = Math.min(NS.getMu(E), PL.getMu(dE));

            alpha[2][0] = Math.min(Z.getMu(E), NL.getMu(dE));
            alpha[2][1] = Math.min(Z.getMu(E), NS.getMu(dE));
            alpha[2][2] = Math.min(Z.getMu(E), Z.getMu(dE));
            alpha[2][3] = Math.min(Z.getMu(E), PS.getMu(dE));
            alpha[2][4] = Math.min(Z.getMu(E), PL.getMu(dE));

            alpha[3][0] = Math.min(PS.getMu(E), NL.getMu(dE));
            alpha[3][1] = Math.min(PS.getMu(E), PS.getMu(dE));
            alpha[3][2] = Math.min(PS.getMu(E), Z.getMu(dE));
            alpha[3][3] = Math.min(PS.getMu(E), PS.getMu(dE));
            alpha[3][4] = Math.min(PS.getMu(E), PL.getMu(dE));

            alpha[4][0] = Math.min(PL.getMu(E), NL.getMu(dE));
            alpha[4][1] = Math.min(PL.getMu(E), NS.getMu(dE));
            alpha[4][2] = Math.min(PL.getMu(E), Z.getMu(dE));
            alpha[4][3] = Math.min(PL.getMu(E), PS.getMu(dE));
            alpha[4][4] = Math.min(PL.getMu(E), PL.getMu(dE));

            float s1 = 0, s2 = 0;
            float maxak = 0;
//            for (float z = -10; z <= 10; z += 0.01) {
//                float[][] mu = new float[5][5];
//                mu[0][0] = Math.min(alpha[0][0], PLv.getMu(z));
//                mu[0][1] = Math.min(alpha[0][1], PLv.getMu(z));
//                mu[0][2] = Math.min(alpha[0][2], PMv.getMu(z));
//                mu[0][3] = Math.min(alpha[0][3], PSv.getMu(z));
//                mu[0][4] = Math.min(alpha[0][4], Zv.getMu(z));
//
//                mu[1][0] = Math.min(alpha[1][0], PLv.getMu(z));
//                mu[1][1] = Math.min(alpha[1][1], PMv.getMu(z));
//                mu[1][2] = Math.min(alpha[1][2], PSv.getMu(z));
//                mu[1][3] = Math.min(alpha[1][3], Zv.getMu(z));
//                mu[1][4] = Math.min(alpha[1][4], NSv.getMu(z));
//
//                mu[2][0] = Math.min(alpha[2][0], PMv.getMu(z));
//                mu[2][1] = Math.min(alpha[2][1], PSv.getMu(z));
//                mu[2][2] = Math.min(alpha[2][2], Zv.getMu(z));
//                mu[2][3] = Math.min(alpha[2][3], NSv.getMu(z));
//                mu[2][4] = Math.min(alpha[2][4], NMv.getMu(z));
//
//                mu[3][0] = Math.min(alpha[3][0], PSv.getMu(z));
//                mu[3][1] = Math.min(alpha[3][1], Zv.getMu(z));
//                mu[3][2] = Math.min(alpha[3][2], NSv.getMu(z));
//                mu[3][3] = Math.min(alpha[3][3], NMv.getMu(z));
//                mu[3][4] = Math.min(alpha[3][4], NLv.getMu(z));
//
//                mu[4][0] = Math.min(alpha[4][0], Zv.getMu(z));
//                mu[4][1] = Math.min(alpha[4][1], NSv.getMu(z));
//                mu[4][2] = Math.min(alpha[4][2], NMv.getMu(z));
//                mu[4][3] = Math.min(alpha[4][3], NLv.getMu(z));
//                mu[4][4] = Math.min(alpha[4][4], NLv.getMu(z));
```

```
//
//                float max = 0;
//                for (int i = 0; i < 5; i++) {
//                    for (int j = 0; j < 5; j++) {
//                        if (mu[i][j] > max) {
//                            max = mu[i][j];
//                        }
//                    }
//                }
//                if (max > maxak) {
//                    maxak = max;
//                }
//                float sum = 0;
//                for (int i = 0; i < 5; i++) {
//                    for (int j = 0; j < 5; j++) {
//                        sum += mu[i][j];
//                    }
//                }
//                float sum = 0;
//                for (int i = 0; i < 5; i++) {
//                    for (int j = 0; j < 5; j++) {
//                        s2 += alpha[i][j];
//                    }
//                }
//
//            }

            for (int i = 0; i < 5; i++) {
                for (int j = 0; j < 5; j++) {
                    s2 += alpha[i][j];
                }
            }
            s1 = alpha[0][0] * PLv.b
                    + alpha[0][1] * PLv.b
                    + alpha[0][2] * PMv.b
                    + alpha[0][3] * PSv.b
                    + alpha[0][4] * Zv.b

                    + alpha[1][0] * PLv.b
                    + alpha[1][1] * PMv.b
                    + alpha[1][2] * PSv.b
                    + alpha[1][3] * Zv.b
                    + alpha[1][4] * NSv.b

                    + alpha[2][0] * PMv.b
                    + alpha[2][1] * PSv.b
                    + alpha[2][2] * Zv.b
                    + alpha[2][3] * NSv.b
                    + alpha[2][4] * NMv.b

                    + alpha[3][0] * PSv.b
                    + alpha[3][1] * Zv.b
                    + alpha[3][2] * NSv.b
                    + alpha[3][3] * NMv.b
                    + alpha[3][4] * NL.b

                    + alpha[4][0] * Zv.b
                    + alpha[4][1] * NSv.b
                    + alpha[4][2] * NMv.b
                    + alpha[4][3] * NLv.b
                    + alpha[4][4] * NLv.b;
//
//
                float result = s1 / s2;
                System.out.println("t = " + t + " E = " + E + " dE = " + dE + " valve = "
+ result);
//            }
      /*      float INF = 0, SUP = 0;
            boolean flag = false;
            for (float z = -10; z <= 10; z += 0.01) {
                float[][] mu = new float[5][5];
```

```
                mu[0][0] = Math.min(alpha[0][0], PLv.getMu(z));
                mu[0][1] = Math.min(alpha[0][1], PLv.getMu(z));
                mu[0][2] = Math.min(alpha[0][2], PMv.getMu(z));
                mu[0][3] = Math.min(alpha[0][3], PSv.getMu(z));
                mu[0][4] = Math.min(alpha[0][4], Zv.getMu(z));

                mu[1][0] = Math.min(alpha[1][0], PLv.getMu(z));
                mu[1][1] = Math.min(alpha[1][1], PMv.getMu(z));
                mu[1][2] = Math.min(alpha[1][2], PSv.getMu(z));
                mu[1][3] = Math.min(alpha[1][3], Zv.getMu(z));
                mu[1][4] = Math.min(alpha[1][4], NSv.getMu(z));

                mu[2][0] = Math.min(alpha[2][0], PMv.getMu(z));
                mu[2][1] = Math.min(alpha[2][1], PSv.getMu(z));
                mu[2][2] = Math.min(alpha[2][2], Zv.getMu(z));
                mu[2][3] = Math.min(alpha[2][3], NSv.getMu(z));
                mu[2][4] = Math.min(alpha[2][4], NMv.getMu(z));

                mu[3][0] = Math.min(alpha[3][0], PSv.getMu(z));
                mu[3][1] = Math.min(alpha[3][1], Zv.getMu(z));
                mu[3][2] = Math.min(alpha[3][2], NSv.getMu(z));
                mu[3][3] = Math.min(alpha[3][3], NMv.getMu(z));
                mu[3][4] = Math.min(alpha[3][4], NLv.getMu(z));

                mu[4][0] = Math.min(alpha[4][0], Zv.getMu(z));
                mu[4][1] = Math.min(alpha[4][1], NSv.getMu(z));
                mu[4][2] = Math.min(alpha[4][2], NMv.getMu(z));
                mu[4][3] = Math.min(alpha[4][3], NLv.getMu(z));
                mu[4][4] = Math.min(alpha[4][4], NLv.getMu(z));

                float max = 0;
                for (int i = 0; i < 5; i++) {
                    for (int j = 0; j < 5; j++) {
                        if (mu[i][j] > max) {
                            max = mu[i][j];
                        }
                    }
                }

                if (max == maxak && flag == false) {
                    INF = z;
                    flag = true;
                } else if (max == maxak) {
                    SUP = z;
                }
            }
            float result = (INF + SUP) / 2;
            System.out.println("t = " + t + " E = " + E + " dE = " + dE + " valve = " +
result);*/
        }
    }


    public void takeHome2(Graphics g) {
        FuzzySetTri fuzzySetB = new FuzzySetTri(0, 2, 6, Color.red, g);
        FuzzySetTri fuzzySetA = new FuzzySetTri(4, 8, 12, Color.blue, g);

        fuzzySetA.drawAxis();

        fuzzySetA.drawSet();
        fuzzySetB.drawSet();

        fuzzySetA.mul_aCut(fuzzySetB, Color.darkGray);
        fuzzySetA.ext_princ(fuzzySetB, Color.darkGray, 2);
        fuzzySetA.div_aCut(fuzzySetB, Color.darkGray);
        fuzzySetA.ext_princ(fuzzySetB, Color.darkGray, 3);

        HashMap map = fuzzySetA.ext_princ(fuzzySetB, Color.green, 3);
        fuzzySetB.ext_princ(map, Color.darkGray, 2);
    }
}
```

```
--------------------------------------------------------------------------------

import javax.swing.UIManager;
import java.awt.Dimension;
import java.awt.Toolkit;

/**
 * Date: Jun 9, 2005
 * Time: 1:41:21 PM
 *
 * @author Vahid Mavaji
 */
public class Application1 {
    boolean packFrame = false;

    /**
     * Construct the application
     */
    public Application1() {
        Frame1 frame = new Frame1();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from their layout
        if (packFrame) {
            frame.pack();
        } else {
            frame.validate();
        }
        //Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
        frame.setVisible(true);
    }

    /**
     * Main method
     */
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace();
        }
        new Application1();
    }
}
```