TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Bài tập môn Cơ Sở Tri Thức

HIỆN THỰC CÁC PHÉP TOÁN TRÊN SỐ MỜ

GVHD
GS. TS. Cao Hoàng Trụ

SVTH
Nguyễn Thanh Hải (7140230)

Tp. Hồ Chí Minh, Tháng 12/2015

MỤC LỤC

1. Yêu cầu bài tập

Hiện thực các phép toán số học (cộng, trừ, nhân, chia) trên số mờ theo nguyên lý mở rộng và nguyên lý số học khoảng. Vẽ đồ thị hiển thị các số mờ là các đối số và kết quả của các phép toán.

2. Cơ sở lý thuyết

2.1. Dẫn nhập

Để giải quyết bài tập này cần có những kiến thức cơ bản về tập mờ, số mờ và các phép toán trên số mờ. Trong phần này, học viên sẽ tóm tắt lại lý thuyết mà học viên đã được học trên lớp và đọc thêm các tài liệu. Việc này giúp học viên vừa cũng cố lại để nắm vững kiến thức, vừa làm nền tảng cho phần hiện thực phần bài tập được giao.

Để tránh các nội dung dư thừa, trong phần này học viên chỉ nêu ra những phần lý thuyết cơ bản nhất và thật sự cần thiết trong quá trình hiện thực thuật giải của học viên.

2.2. Tập mờ

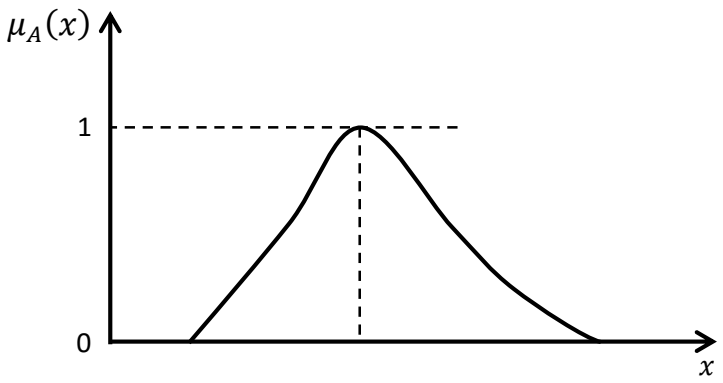
2.2.1. Khái niệm

Lý thuyết *tập mờ* được Lofti Zadeh đưa ra năm 1965. Các *tập mờ* hay *tập hợp mờ (fuzzy set)* là một mở rộng của lý thuyết tập hợp cổ điển và được dùng trong lô-gic mờ. Trong lý thuyết tập hợp cổ điển, quan hệ thành viên của các phần tử trong một tập hợp được đánh giá theo kiểu nhị phân theo một điều kiện rõ ràng, một phần tử hoặc thuộc hoặc không thuộc về tập hợp. Ngược lại, lý thuyết tập mờ cho phép đánh giá về quan hệ thành viên giữa một phần tử và một tập hợp, quan hệ này được mô tả bằng một hàm thành viên *(membership function)* $\mu(x): U \rightarrow [0,1]$. Lý thuyết tập mờ được coi là một mở rộng của lý thuyết tập hợp cổ điển. Với một *vũ trụ (universe)* nhất định, khi hàm thành viên suy biến, ánh xạ mỗi phần tử tới một giá trị 0 hoặc 1 thì tập mờ trở thành tập rõ như trong khái niệm cổ điển. Trong trường hợp này hàm thành viên có thể giữ vai trò của một hàm đặc trưng *(indicator function).*

Định nghĩa: Một tập mờ A trên một không gian χ được định nghĩa như sau:

$$A = \{(x, \mu_A(x)) | x \in \chi\}$$

Hàm thành viên $\mu_A(x)$ lượng hóa mức độ mà các phần tử x thuộc về tập cơ sở χ. Nếu hàm cho kết quả 0 đối với một phần tử thì phần tử đó không có trong tập đã cho, kết quả 1 mô tả một thành viên toàn phần của tập hợp. Các giá trị trong khoảng mở từ 0 đến 1 đặc trưng cho các thành viên mờ.



Hình 1. Tập mờ

Hàm thành viên $\mu_A(x)$ thỏa mãn các điều kiện sau:

$$\mu_A(x) \geq 0, \forall x \in \chi$$

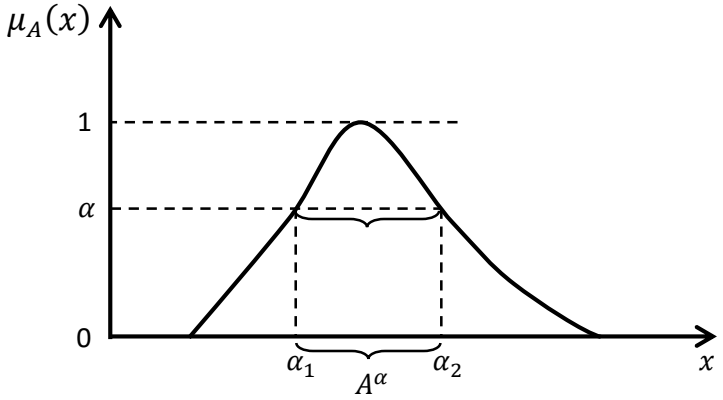
$$sup_{x\in\chi}[\mu_A(x)] = 1$$

Như vậy tập mờ A là một bộ đôi gồm một tập hợp cổ điển U và một hàm thành viên $\mu(x): U \rightarrow [0,1]$ xác định mức độ thuộc của từng phần tử trong U đối với A.

2.2.2. Lát cắt α

Định nghĩa: Lát cắt α *(α-cut)* của tập mờ A xác định bởi hàm thành viên $\mu_A(x)$ được định nghĩa như sau:

$$A^\alpha = \{x|\mu_A(x) \geq \alpha\}$$



Hình 2. Lát cắt α

Trong trường hợp tập mờ như ở *Hình 2* thì $A^\alpha = [\alpha_1, \alpha_2]$

Ký hiệu $A^{\alpha+}$ là lát cắt α mạnh *(strong α-cut)*, được định nghĩa như sau:

$$A^{\alpha+} = \{x|\mu_A(x) > \alpha\}$$

2.2.3. Các khái niệm liên quan

Để xây dựng khái niệm số mờ cần thêm một số khái niệm khác như: *nền* (*support*), *lõi* (*core*) và *chiều cao* (*height*) trên tập mờ.

Nền của tập mờ: Với mỗi tập mờ A trên miền U, tập các phần tử trên U có mức độ thành viên lớn hơn 0 gọi là nền của A, ký hiệu là $supp(A)$. Nếu xem xét thông qua khái niệm lát cắt α thì nền của tập mờ A chính là lát cắt α mạnh với $\alpha = 0$.

Tức là

$$supp(A) = \{u|A(u) > 0\} = A^{0+}$$

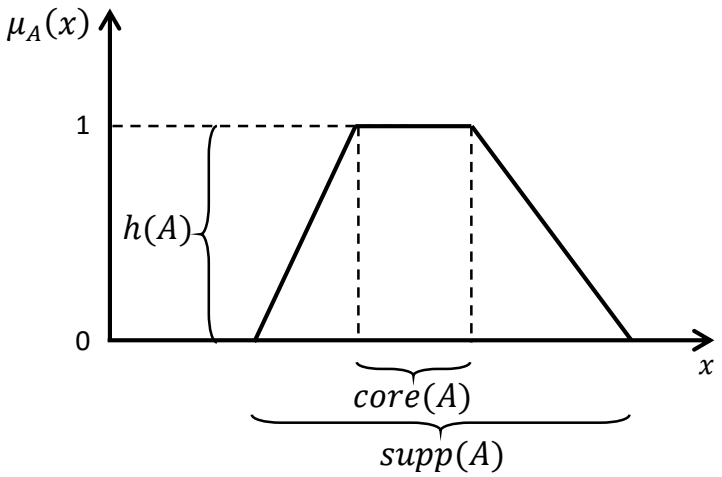
Lõi của tập mờ: Với mỗi tập mờ A trên miền U, tập các phần tử trên U có mức độ thành viên bằng 1 gọi là lõi của A, ký hiệu là $core(A)$. Đây cũng chính là lát cắt α của A với $\alpha = 1$ (vì $A(u) \leq 1$).

Tức là

$$core(A) = \{u|A(u) = 1\} = A^{1}$$

Chiều cao của tập mờ: Chiều cao của tập mờ A trên miền U, ký hiệu là $h(A)$, được định nghĩa như sau:

$$h(A) = \sup\{A(u)|\mathrm{u} \in \mathrm{U}\}$$



Hình 3. Nền, lõi và chiều cao của tập mờ

2.2.4. Phân loại tập mờ

Dựa vào khái niệm độ cao của tập mờ, có thể phân tập mờ thành hai loại:

- Tập mờ bình thường (normal fuzzy set)
- Tập mờ bất thường (sub-normal fuzzy set)

Tập mờ A trên miền U được gọi là *tập mờ bình thường* nếu và chỉ nếu $h(A) = 1$, ngược lại thì gọi là *tập mờ bất thường*. Về mặt ý nghĩa, tập mờ bình thường phải có ít nhất một phần tử hoàn toàn thuộc vào nó, tức là có mức độ thành viên bằng 1.

2.3. Số học mờ

Các giá trị định lượng mờ và việc tính toán trên chúng đòi hỏi phải có một lý thuyết về *số học mờ* (*fuzzy arithmetic*).

2.3.1. Định nghĩa số mờ

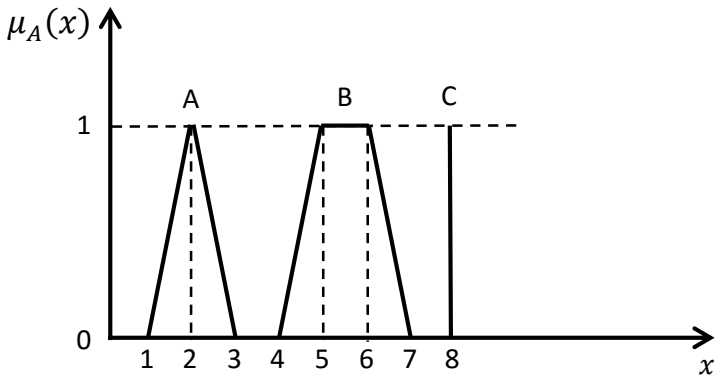
Một số mờ A là một tập mờ trên miền số thực $\mathbb{R}$ thỏa:

- A là một tập mờ, có hàm thành viên $\mu_A(x)$ liên tục trên từng đoạn.
- A^α là khoảng đóng, $\forall \alpha \in (0,1]$
- $supp(A) = A^{0+}$ phải được bao, tức là có giới hạn biên trái và phải.

2.3.2. Các dạng số mờ đặc biệt

Số mờ hình tam giác là số mờ có hàm thành viên hình tam giác, biểu diễn một giá trị mờ gần khoảng một giá trị chính xác.

Số mờ hình thang là số mờ có hàm thành viên hình thang, biểu diễn một khoảng mờ mà hai biên không xác định.



Hình 4. Các dạng số mờ đặc biệt

Trong ví dụ ở hình trên:

- A là số mờ tam giác *khoảng 2*
- B là số mờ hình thang *khoảng từ 5 đến 6*
- C là số 8, khi số mờ suy biến thành số rõ.

2.4. Tính toán trên số mờ

Có hai phương pháp phổ biến trong việc tính toán trên các số mờ.

2.4.1. Phương pháp dựa trên nguyên lý lát cắt α

Theo nguyên lý này, để thực hiện một phép toán, được ký hiệu chung là ∘, trên hai số mờ, trước hết ta áp dụng phép toán này trên từng cặp lát cắt α của hai số mờ đó, sau đó sử dụng các kết quả như là các lát cắt α của số mờ kết quả để xây dựng lại nó.

Tức là:

$$(A \circ B)^{\alpha} = A^{\alpha} \circ B^{\alpha}$$

với mọi $\alpha \in [0,1]$ và ∘ là một phép toán trên hai số mờ A và B.

Vì các lát cắt α là các khoảng, nên việc thực hiện các phép toán trên các lát cắt α thực chất là thực hiện các phép toán trên các khoảng. Việc thực hiện một phép toán ∘ trên hai khoảng $[a,b]$ và $[d,e]$ có thể xem như việc thực hiện phép toán này trên mỗi cặp số $f \in [a,b]$ và $g \in [d,e]$. Tức là:

$$[a,b] \circ [d,e] = \{\, f \circ g \mid a \le f \le b, d \le g \le e \,\}$$

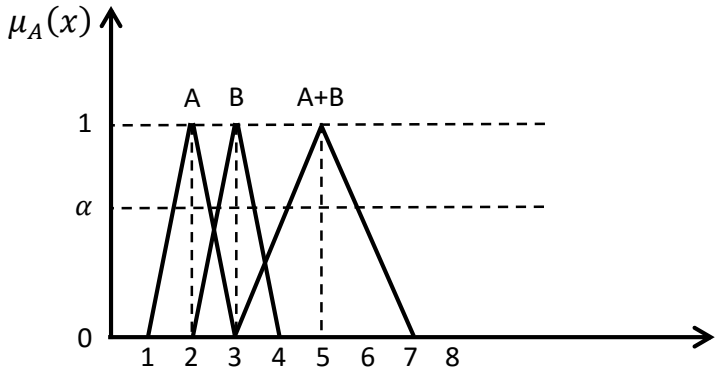
Với ∘ là một trong các phép toán số học cộng, trừ, nhân và chia thông thường, các kết quả như trên là các khoảng đóng. Cụ thể là:

$$[a,b] + [d,e] = [a+d, b+e]$$

$$[a,b] - [d,e] = [a-e, b-d]$$

$$[a,b] \times [d,e] = [\min(ad, ae, bd, be), \max(ad, ae, bd, be)]$$

$$[a,b] \div [d,e] = [a,b] \times \left[\frac{1}{e}, \frac{1}{d}\right], \qquad 0 \notin [d,e]$$



Hình 5. Phép cộng hai số mờ theo phương pháp lát cắt α

2.4.2. Phương pháp dựa trên nguyên lý mở rộng

Phương pháp này tính toán trên các số mờ dựa trên nguyên lý mở rộng (extension principle).

Mỗi hàm

$$f: U_1 \times U_2 \times \ldots \times U_n \to V$$

dẫn ra hàm mờ tương ứng

$$g: \widetilde{U}_1 \times \widetilde{U}_2 \times \ldots \times \widetilde{U}_n \to \widetilde{V}$$

với $\widetilde{V}$ là tập các tập mờ trên V và mỗi $\widetilde{U}_i$ là tập các tập mờ trên U_i, sao cho

$$[g(A_1, A_2, \ldots, A_n)](v) = sup_{\{(u_1,u_2,\ldots,u_n)|v=f(u_1,u_2,\ldots,u_n)\}} min\{A_1(u_1), A_2(u_2), \ldots, A_n(u_n)\}$$

Với các phép toán số học thông thường, mỗi phép toán số học trên các số thực xác định phép toán số học tương ứng trên các số mờ, cụ thể cho từng phép toán cộng, trừ, nhân, chia trên hai số mờ A và B như sau:

$$(A + B)(z) = sup_{\{(x,y)|z=x+y)\}} min\{A(x), B(y)\}$$

$$(A - B)(z) = sup_{\{(x,y)|z=x-y)\}} min\{A(x), B(y)\}$$

$$(A \times B)(z) = sup_{\{(x,y)|z=x\times y)\}} min\{A(x), B(y)\}$$

$$(A \div B)(z) = sup_{\{(x,y)|z=x\div y)\}} min\{A(x), B(y)\}$$

3. Hiện thực

3.1. Dẫn nhập

Trong phần này học viên sẽ trình bày ý tưởng, giải thuật hiện thực chương trình tính toán và vẽ đồ thị hiển thị kết quả. Học viên cũng sẽ trình bày các kết quả trung gian, các vấn đề phát sinh trong quá trình hiện thực và các giải pháp mà học viên đưa ra để khắc phục. Kết quả đạt được cuối cùng sẽ được học viên trình bày trong phần *Kết quả*.

3.2. Hiện thực chung

3.2.1. Môi trường hiện thực chương trình

- Hệ điều hành Windows 10 64-bit
- Trình biên dịch Visual Studio 2013
- Ngôn ngữ C#
- .NET 4.5

3.2.2. Ý tưởng chung về chương trình

Chương trình là hiện thực là một chương trình phần mềm có giao diện đồ họa trên máy tính (windows form application), cho phép thực hiện các phép toán cộng, trừ, nhân, chia trên các số mờ theo nguyên lý mở rộng và nguyên lý lát cắt α.

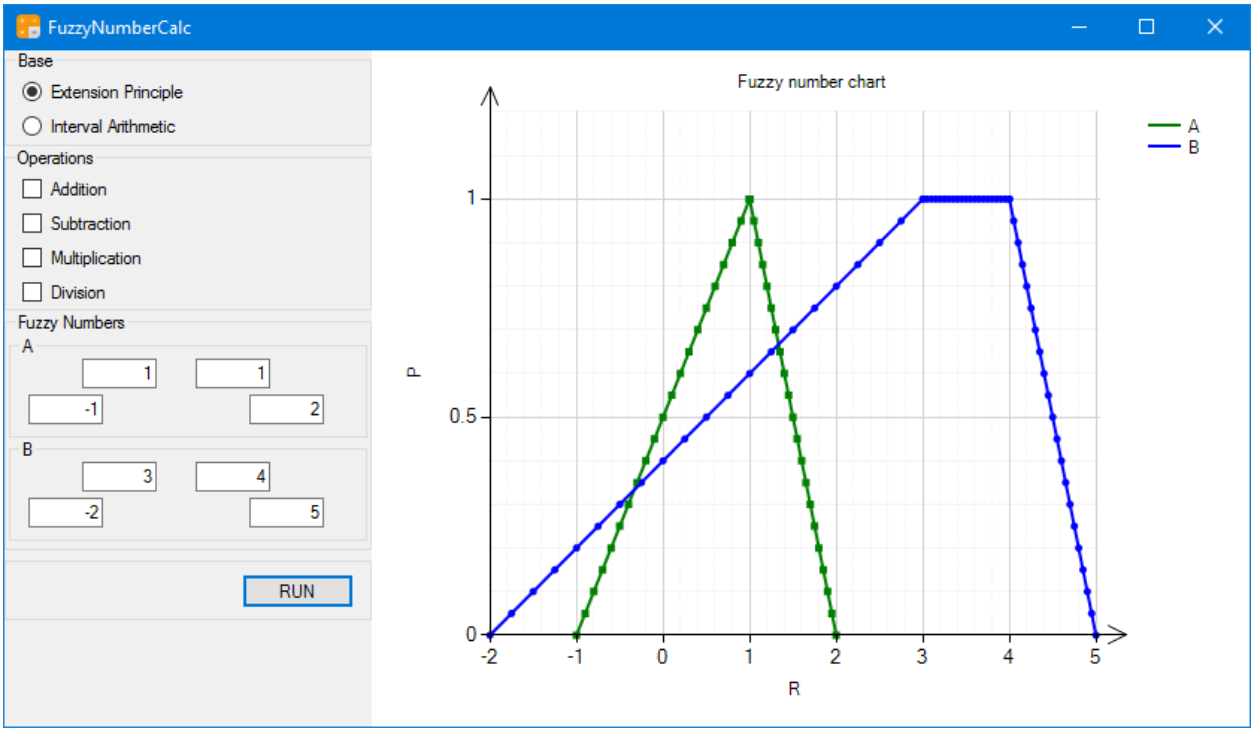
Các giới hạn:

- Chỉ tính theo theo hai phương pháp: theo nguyên lý mở rộng và nguyên lý lát cắt α
- Chỉ tính toán trên hai toán tử A và B
- Chỉ tính bốn phép toán số học cơ bản là: cộng, trừ, nhân và chia
- Chỉ cho phép các toán tử là số mờ có dạng hình tam giác hoặc hình thang

3.2.3. Giao diện chương trình

Như đã đề cập ở phần trên, giao diện chương trình cần đảm bảo các phần chính yếu sau:

- Cho phép nhập các số mờ A và B
- Cho phép chọn phương pháp tính: theo nguyên lý lát cắt α hay theo nguyên lý mở rộng
- Cho phép chọn phép toán cần tính
- Cho phép vẽ đồ thị các toán tử và kết quả



Hình 6. Các thành phần chính trong giao diện chương trình

Như đã đề cập ở phần trên, chương trình chỉ hỗ trợ tính toán trên các số mờ có dạng hình tam giác hoặc hình thang. Ở mỗi khung nhập số mờ *A* hoặc *B* có 4 ô, tương ứng với hoành độ của bốn điểm trên hình thang của số mờ. Khi hai điểm đỉnh trên của hình thang có giá trị bằng nhau thì hình thang suy biến thành hình tam giác.

Đây là giao diện dự kiến ban đầu của chương trình, chứa tất cả các yếu cần thiết của chương trình. Trong quá trình thực hiện ở phần sau có bổ sung thêm một số yếu tố để đáp ứng tốt hơn trong từng giải thuật tính toán.

Việc vẽ đồ thị sẽ sử dụng *Component Chart*, đây là một thành phần chuẩn được hãng *Microsoft* hỗ trợ và được tích hợp sẵn trong *Visual Studio 2013*.

3.2.4. Khung chương trình

Class FormMain: chứa chương trình chính để tính toán và vẽ đồ thị kết quả. Trong class này có 3 hàm chính là:

```
private void drawTrapezoidFuzzyNumberGraph(…)
```

Hàm này sẽ vẽ đồ thị các số mờ *A* và *B* nhập vào. Khi có yêu cầu tính toán (nhấn nút *RUN*) hàm này sẽ được gọi 2 lần để lần lược vẽ hai số mờ *A* và *B*. Tham số chính của hàm là số mờ cần vẽ và màu sắc hiển thị.

```
private void drawResultIntervalBasedGraph(…)
```

Hàm này sẽ tính toán và vẽ đồ thị số mờ theo nguyên lý lát cắt α (số học khoảng). Giải thuật hiện thực hàm này sẽ được trình bày chi tiết trong phần sau.

```
private void drawResultEPBasedGraph(…)
```

Hàm này sẽ tính toán và vẽ đồ thị số mờ theo nguyên lý mở rộng. Giải thuật hiện thực hàm này sẽ được trình bày chi tiết trong phần sau.

Ngoài ra, trong class này cũng có một số hàm phụ trợ khác:

```
private void buttonRun_Click(…)
```

Hàm xử lý sự kiện khi nhấp chuột vào nút *RUN*. Các tác vụ chính của hàm này là:

- Kiểm tra tính hợp lệ của các số mờ *A* và *B* nhập vào, nếu hợp lệ mới thực hiện tiếp.
- Vẽ đồ thị hiển thị hai số mờ *A* và *B*
- Tính toán và vẽ đồ thị kết quả theo các điều kiện lựa chọn.

```
private bool validateAndGetInput(…)
```

Hàm để kiểm tra tính hợp lệ của các số mờ *A* và *B* nhập vào. Nếu hợp lệ thì lấy giá trị trả về cho các hàm tính toán. Một số mờ hợp lệ khi nó đúng dạng hình thang hoặc tam khác, chấp nhập suy biến thành hình chữ nhật hoặc suy biến về số rõ.

```
private void resetGraphCoordinate(…)
```

Hàm đặt lại giá trị góc tọa độ cho đồ thị, mục đích là giúp đồ thị vẽ ra dễ nhìn hơn. Việc này chỉ liên quan đến phần hiển thị đồ thị, không ảnh hưởng tới các giải thuật khác.

Class FuzzyNumberTrapezoid: đây là class định nghĩa số mờ dạng hình thang.

```
public class TrapezoidFuzzyNumber
{
    public decimal BottomLeft { get; set; }
    public decimal BottomRight { get; set; }
    public decimal TopLeft { get; set; }
    public decimal TopRight { get; set; }

    public bool IsValid()
    public bool IsZero()
    public decimal Probability(decimal x)
    public Interval AlphaCut(decimal y)
}
```

Class này có 4 thuộc tính *BottomLeft*, *BottomRight*, *TopLeft*, *TopRight* tương ứng với bốn giá trị hoành độ của bốn đỉnh của số mờ hình thang.

Class này cũng có 4 phương thức xử lý.

- Hàm *IsValid*: kiểm tra sự hợp lệ của số mờ. Hàm trả về *true* nếu số mờ là hợp lệ, ngược lại thì trả về *false*. Hàm này được sử dụng trong hàm *validateAndGetInput* của class FormMain đã trình bày ở trên.

- Hàm *IsZero*: kiểm tra xem số mờ có chứa số 0 không. Hàm trả về *true* nếu số mờ chứa số 0, ngược lại thì trả về *false*. Hàm này để kiểm tra khi thực hiện phép chia.
- Hàm *Probability*: hàm này trả về giá trị hàm thành viên của số mờ tại một điểm có hoành độ x bất kỳ.
- Hàm *AlphaCut*: hàm tính lát cắt α của số mờ tại một tung độ y (tức α) bất kỳ. Kết quả trả về là một khoảng (*Interval*) với giá trị chặng trên và dưới. Kiểu Interval là một class tự định nghĩa trong chương trình.

Class Interval: đây là class định nghĩa một khảng số học. Class có hai thuộc tính chính là cận trên và cận dưới của khoảng số học.

```
public class Interval
{
    public decimal LowerBound { get; set; }
    public decimal UpperBound { get; set; }
}
```

Class CPoint: đây là class định nghĩa một điểm trong không gian hai chiều. Hai thuộc tính chính của class là hoành độ và tung độ của điểm.

```
public class CPoint
{
    public decimal x { get; set; }
    public decimal y { get; set; }
}
```

Class ArithmeticInterval (trong file *Utils.cs*): đây là một class khá quan trọng, nó giúp tính toán các phép toán trong số học khoảng. Class này có một hàm chính là *Calculate*, hàm này nhận tham số là hai khoảng và phép toán cần tính, kết quả trả về là một khoảng kết quả.

```
public static class ArithmeticInterval
    {
        public static Interval Calculate(Operation operation, Interval a, Interval b)
        {
            Interval result = new Interval();
            switch (operation)
            {
                case Operation.Add:
                    result.UpperBound = a.UpperBound + b.UpperBound;
                    result.LowerBound = a.LowerBound + b.LowerBound;
                    break;
                case Operation.Sub:
                    result.UpperBound = a.UpperBound - b.LowerBound;
                    result.LowerBound = a.LowerBound - b.UpperBound;
                    break;
                case Operation.Mul:
                    result.UpperBound = Math.Max(Math.Max(a.UpperBound * b.UpperBound, a.UpperBound *
b.LowerBound), Math.Max(a.LowerBound * b.UpperBound, a.LowerBound * b.LowerBound));
                    result.LowerBound = Math.Min(Math.Min(a.UpperBound * b.UpperBound, a.UpperBound *
b.LowerBound), Math.Min(a.LowerBound * b.UpperBound, a.LowerBound * b.LowerBound));
                    break;
                case Operation.Div:
                    if (b.UpperBound != 0 && b.LowerBound != 0)
                    {
                        result.UpperBound = Math.Max(Math.Max(a.UpperBound / b.UpperBound, a.UpperBoun
d / b.LowerBound), Math.Max(a.LowerBound / b.UpperBound, a.LowerBound / b.LowerBound));
                        result.LowerBound = Math.Min(Math.Min(a.UpperBound / b.UpperBound, a.UpperBoun
d / b.LowerBound), Math.Min(a.LowerBound / b.UpperBound, a.LowerBound / b.LowerBound));
                    }
                    break;
                default:
                    break;
            }
            return result;
        }
    }
```

Class ExtensionPrinciple (trong file *Utils.cs*): tương tự như *Class ArithmeticInterval* ở trên, nhưng class này sẽ được sử dụng cho việc tính toán theo phương pháp nguyên lý mở rộng.

```
public static class ExtensionPrinciple
    {
        public static decimal Calculate(Operation operation, decimal a, decimal b)
        {
            decimal result = 0;
            switch (operation)
            {
                case Operation.Add:
                    result = a + b;
                    break;
                case Operation.Sub:
                    result = a - b;
                    break;
                case Operation.Mul:
                    result = a * b;
                    break;
                case Operation.Div:
                    if (b != 0)
                        result = a / b;
                    break;
                default:
                    break;
            }
            return result;
        }
    }
```

Ngoài các class chính trình bày ở trên, chương trình còn có một số class phụ khác.

Như vậy với các class đã xây dựng, chúng ta có thể hiện thực việc tính toán và vẽ đồ thị kết quả. Việc hiện thực chi tiết theo từng phương pháp sẽ được trình bày trong phần sau.

3.3. Hiện thực phương pháp dựa trên nguyên lý lát cắt α

Giải thuật chính được viết trong hàm *drawResultIntervalBasedGraph* của class *FormMain*. Ý tưởng chính là rời rạc hóa, cho giá trị α chạy từ 0 đến 1 với một bước nhảy nhất định. Ở mỗi giá trị α tính lát cắt α của kết quả và thêm vào đồ thị. Bước nhảy càng nhỏ thì đồ thị kết quả vẽ ra càng mịn, càng thực hơn.

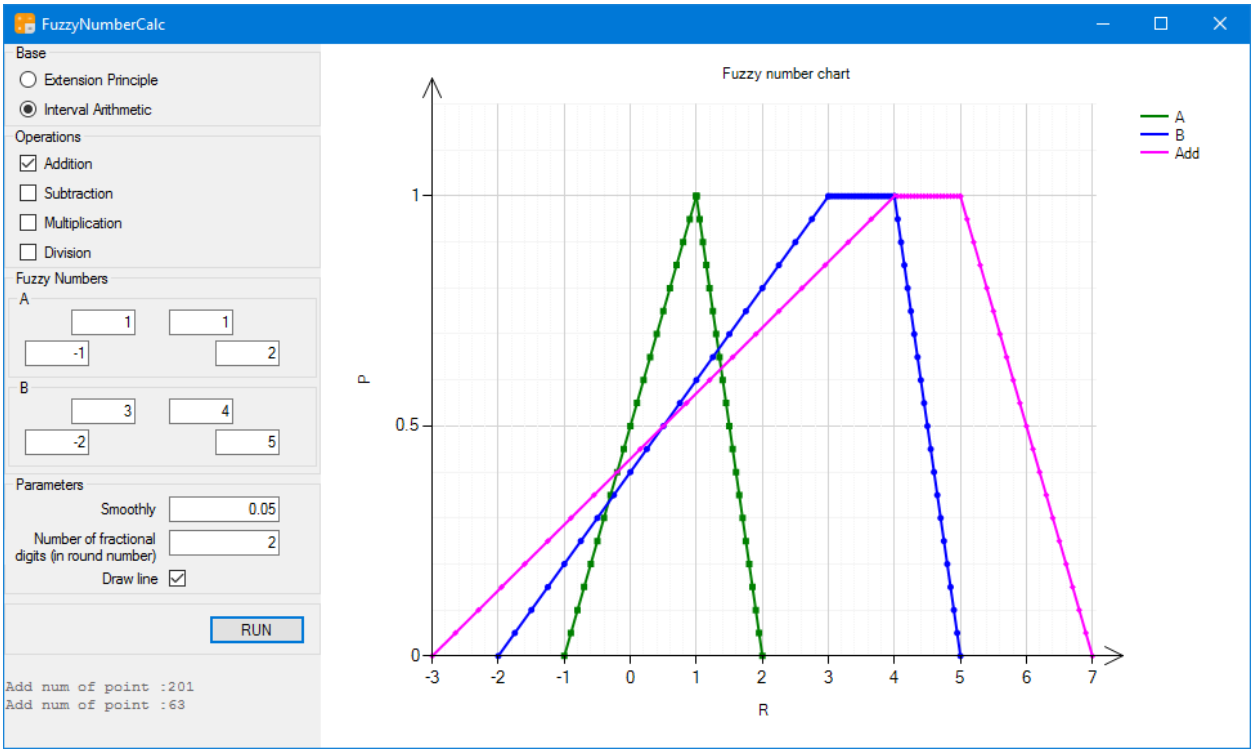
Có thể trình bày tóm tắt giải thuật theo mã giả sau:

```
y = 0;
while (y <= 1)
{
    a = fnumA.AlphaCut(y);
    b = fnumB.AlphaCut(y);
    acut = ArithmeticInterval.Calculate(operation, a, b);

    chart.Add(acut.LowerBound, y);
    chart.Add(acut.UpperBound, y);
    y += smooth;
}
```

Giải thuật:

- Cho y (tức α) chạy từ 0 đến 1, bước nhảy là smooth. Giá trị *smooth* do người dùng nhập vào, được xem như là độ mịn của đồ thị.
- Ở mỗi bước, ta tính lát cắt α của A và B, từ đó tính ra lát cắt α của kết quả tại đó.

Như vậy, để giao diện chương trình cần thêm một ô để nhập vào giá trị *smooth*.



Hình 7. Phép cộng số mờ theo nguyên lý lát cắt α

3.4. Hiện thực phương pháp dựa trên nguyên lý mở rộng

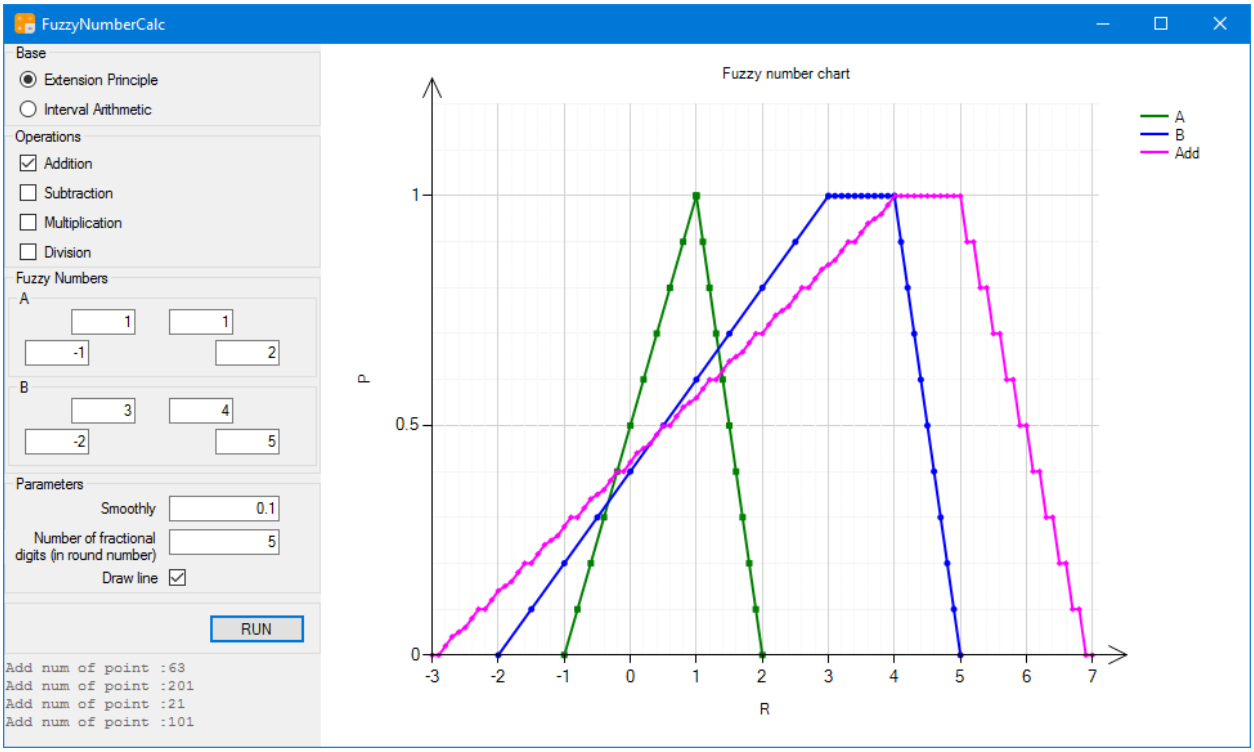
Giải thuật chính được viết trong hàm *drawResultEPBasedGraph* của class *FormMain*. Ý tưởng chính là rời rạc hóa. Có thể trình bày tóm tắt giải thuật theo mã giả sau:

```
xA = fnumA.BottomLeft;
while (xA <= fnumA.BottomRight)
{
    xB = fnumB.BottomLeft;
    while (xB <= fnumB.BottomRight)
    {
        xResult = ExtensionPrinciple.Calculate(operation, xA, xB);

        aProbability = fnumA.Probability(xA);
        bProbability = fnumB.Probability(xB);
        minProbability = Math.Min(aProbability, bProbability);

        tempPoint = resultPointList.FirstOrDefault(p => p.x == xResult);
        if (tempPoint == null)
        {
            resultPointList.Add(new CPoint(xResult, minProbability));
        }
        else
        {
            tempPoint.y = Math.Max(tempPoint.y, minProbability);
        }
        xB += smooth;
    }
    xA += smooth;
}
```

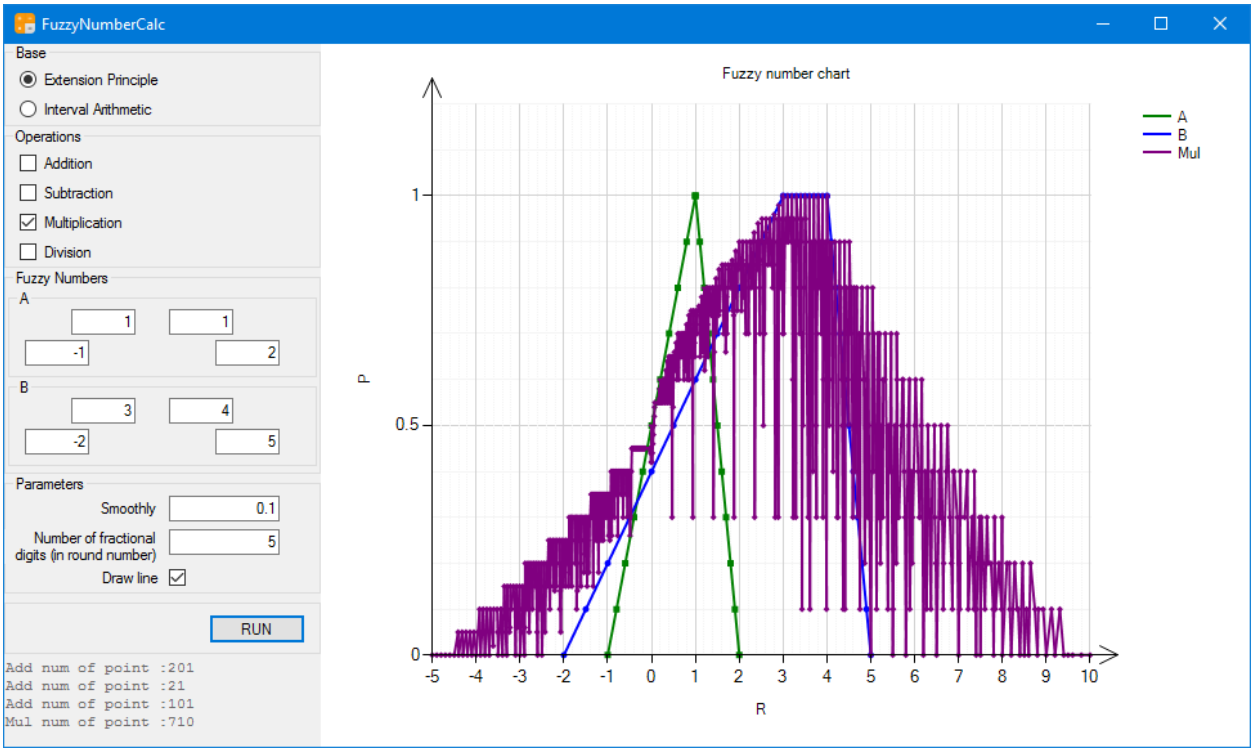
Giải thuật:

- Với hai số mờ A và B, *resultPointList* là danh sách các điểm sẽ thêm vào đồ thị.
- Cho xA và xB lần lược chạy trên khoảng
- Tại mỗi xA và xB ta tính giá trị hàm thành viên của chúng là $U(xA)$, $U(xB)$, $z = xA°xB$ và $U(z) = \min\{U(xA), U(xB)\}$
    - Nếu trong *resultPointList* không tồn tại điểm có hoành độ là z thì thêm mới điểm $(z, U(z))$ vào.
    - Ngược lại, ta sẽ cập nhật tung độ của điểm có hoàng độ là z đó nếu $\min\{U(xA), U(xB)\}$ lơn hơn tung độ cũ đã đó của điểm đó.

Kết quả:



Hình 8. Kết quả chưa tối ưu khi cộng số mờ theo nguyên lý mở rộng



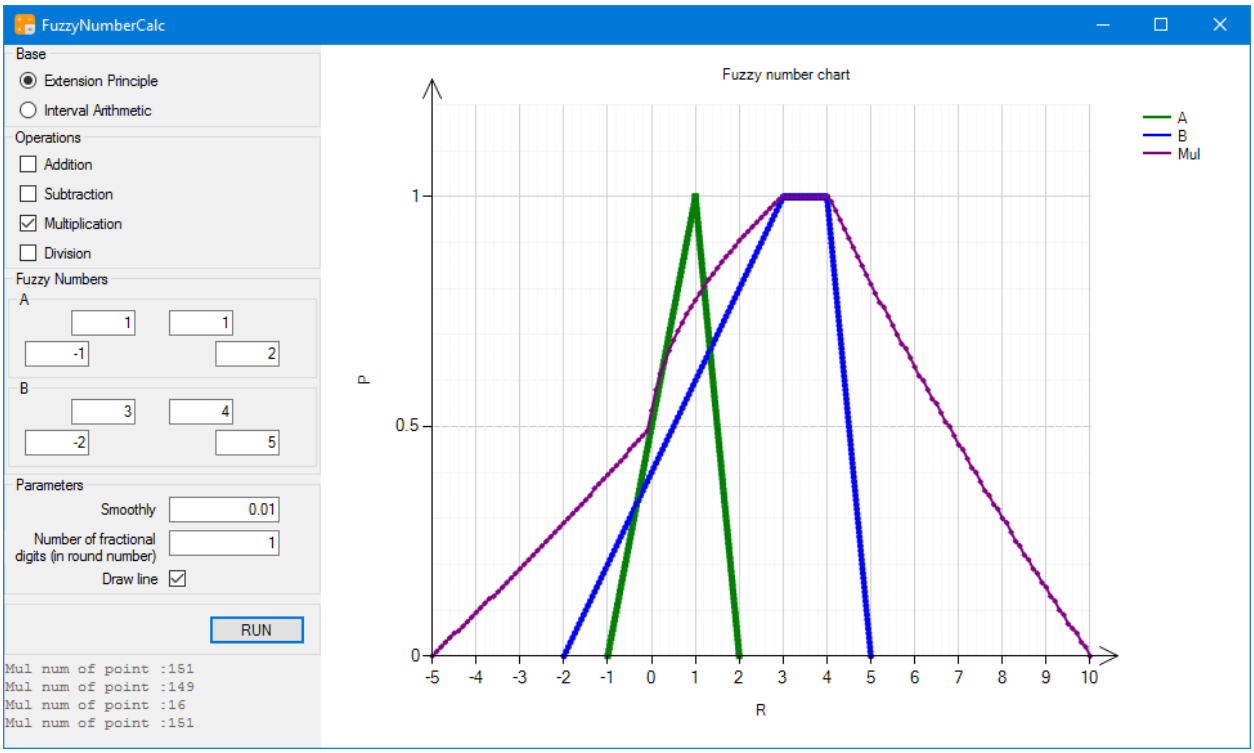
Hình 9. Kết quả chưa tối ưu khi nhân số mờ theo nguyên lý mở rộng

Kết quả nhận được có vẻ chưa được tối ưu, nhất là trong phép nhân. Khi còn rất nhiều điểm không đúng so với tính toán thực tế.

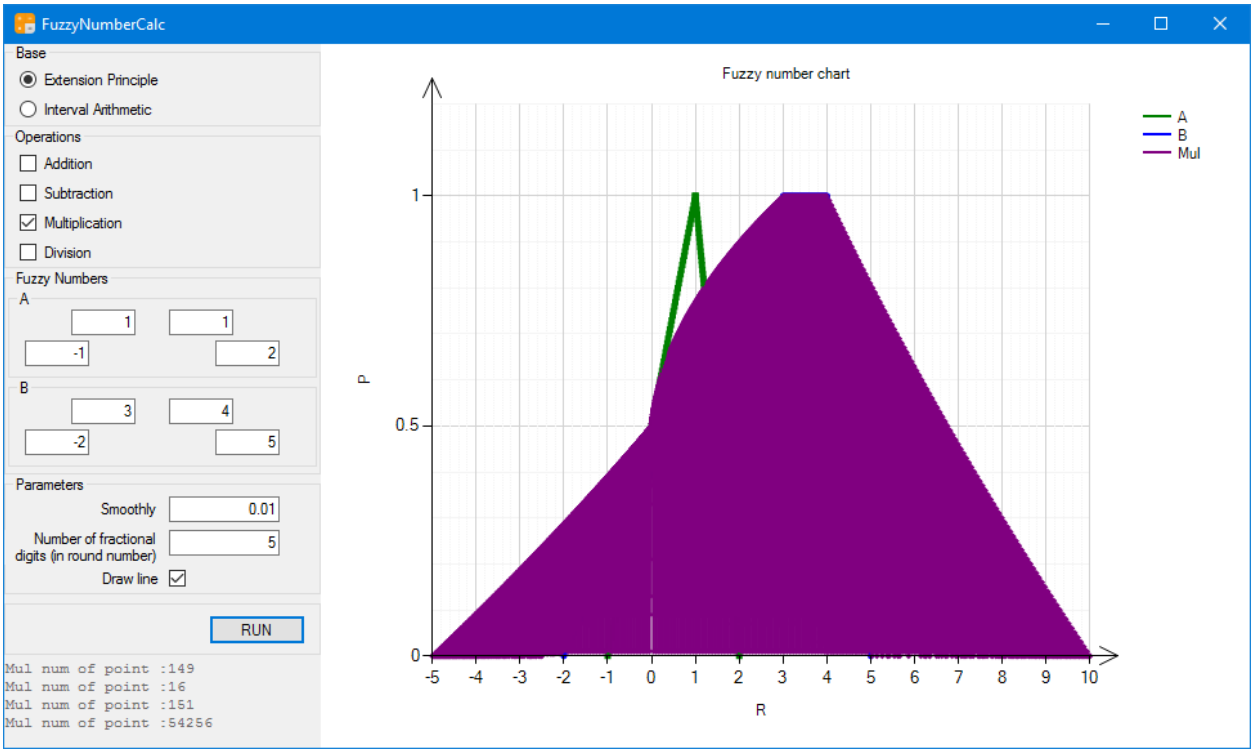
Xem lại trong giải thuật tính toán ta thấy có sơ hở. Đó là cách tính hơi ngược so với mô tả của nguyên lý:

- Trong nguyên lý mở rộng: chọn z trước, rồi tìm x, y của A, B thỏa mãn điều kiện của nguyên lý
- Trong giải thuật hiện thực: chọn x, y rồi mới tính ra z.

Để khắc phục nhược điểm này mà không phải sửa đổi quá nhiều, học viên đã thử làm tròn số của z sau khi tính. Khi đó cần thêm thông số là số chữ số sau dấy phẩy thập phân sau khi làm tròn (*Number of fractional digits*). Kết quả nhận được là tốt hơn hẳn.



Hình 10. Kết quả khi nhân số mờ theo nguyên lý mở rộng với làm tròn 1 chữ số thập phân

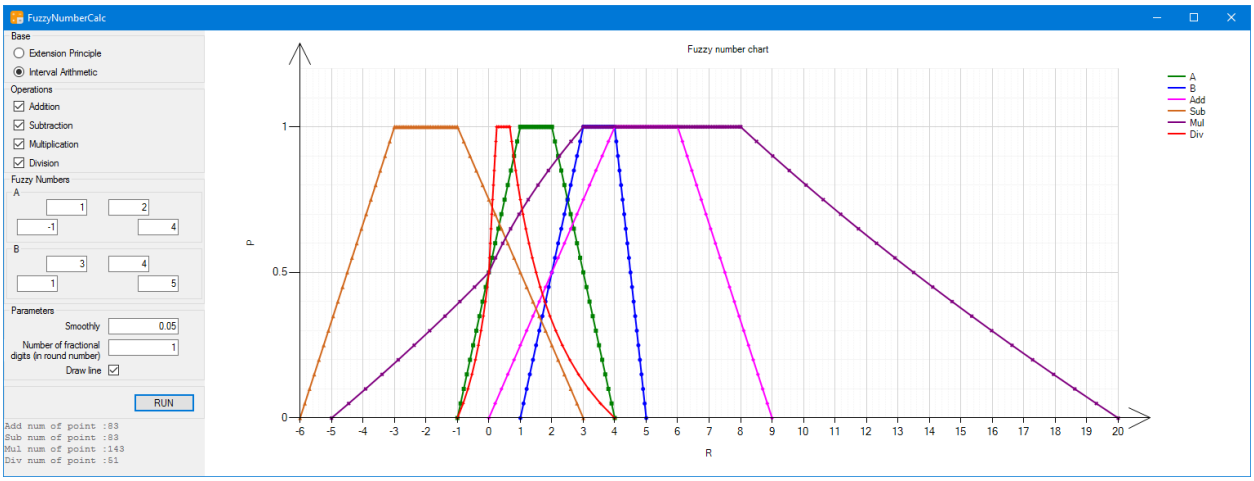


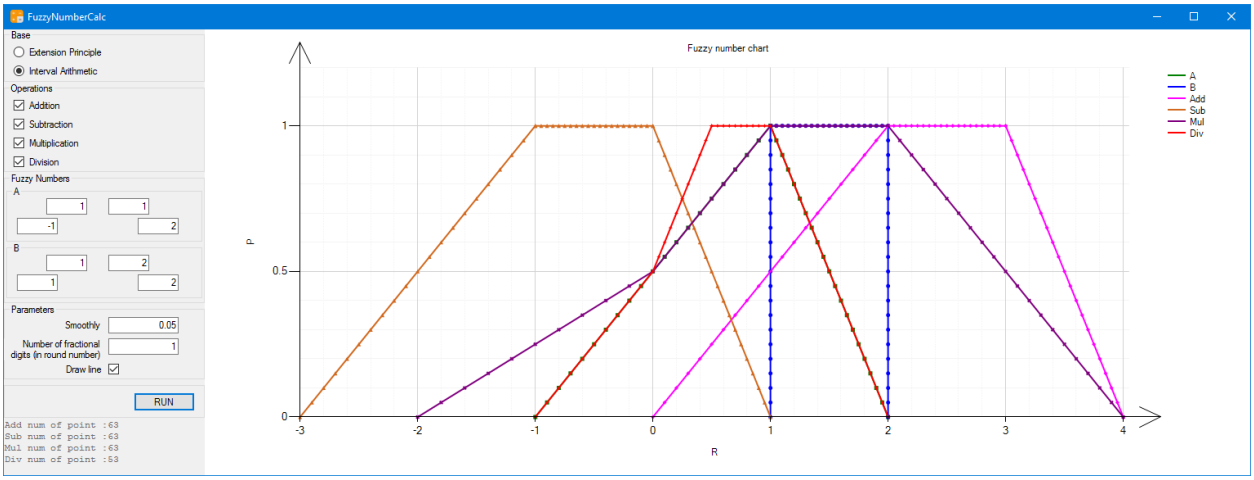
Hình 11. Kết quả khi nhân số mờ theo nguyên lý mở rộng với làm tròn 5 chữ số thập phân

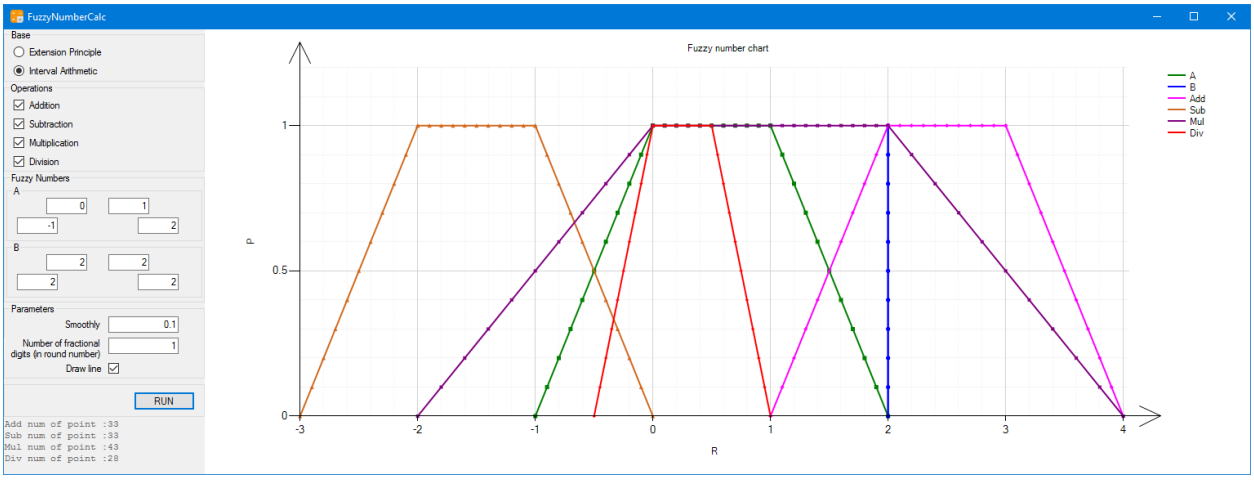
4. Kết quả

Một số kết quả khi chạy chương trình

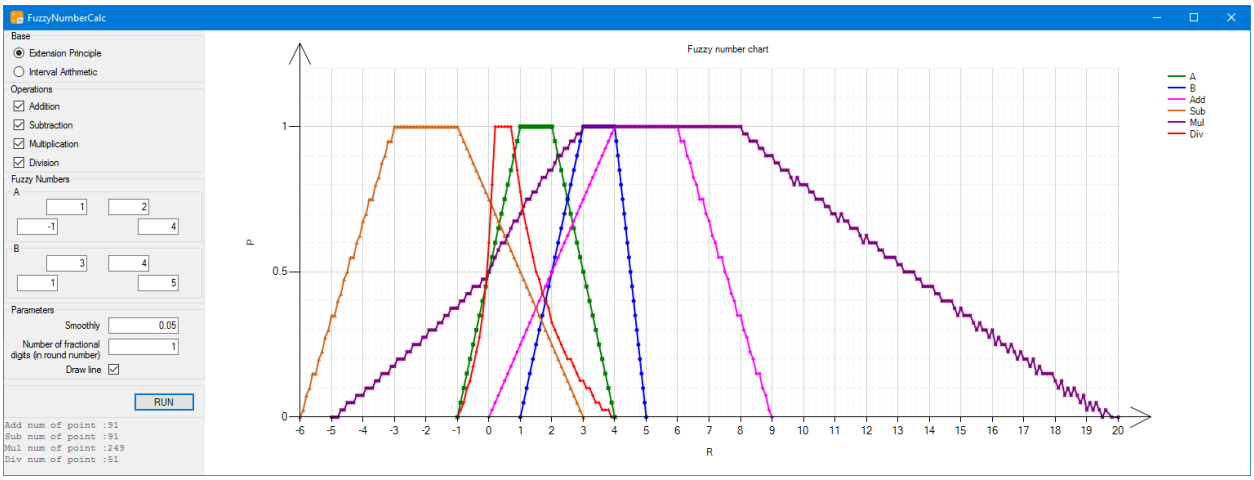
4.1. Theo nguyên lý lát cắt α

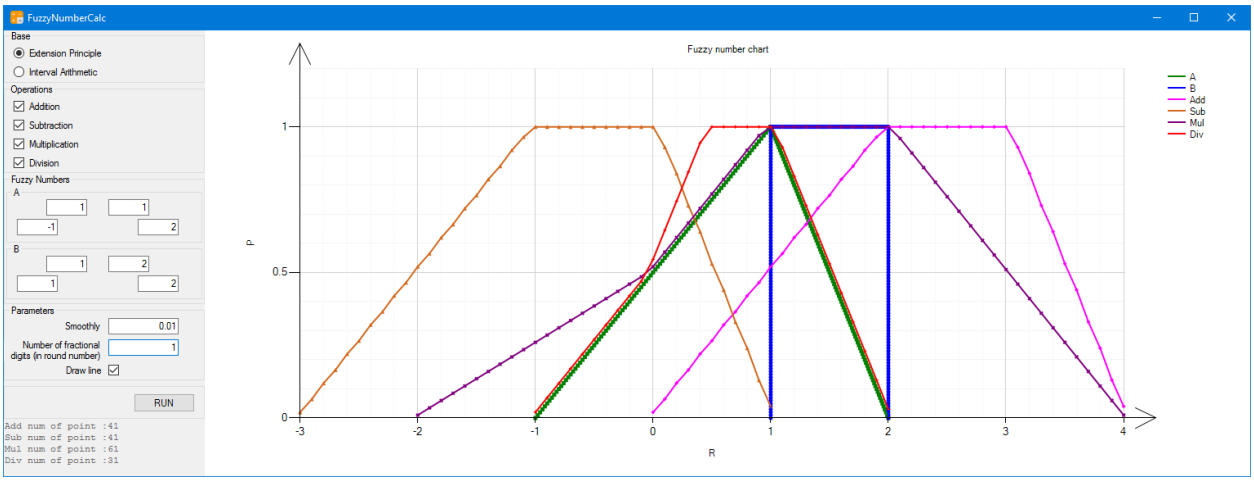


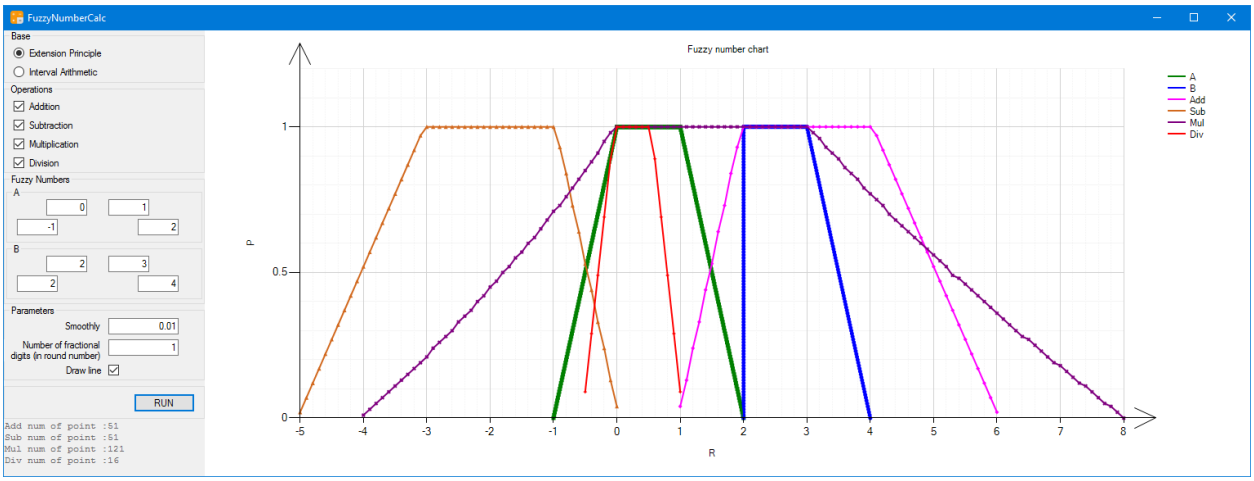




4.2. Theo nguyên lý mở rộng







5. Kết luận

Sau khi làm bài tập này học viên đã hiểu hơn về tập mờ, số mờ, cũng như cách thực hiện các phép toán trên số mờ.

Phương pháp tính theo nguyên lý mở rộng hiện thực phức tạp hơn nguyên lý lát cắt α, học viên mất nhiều thời gian để tìm hiểu và hiện thực nó hơn.

Phương pháp tính theo nguyên lý mở rộng tính toán tốn nhiều thời gian hơn và cho kết qua không đẹp mắt bằng nguyên lý lát cắt α.

Tuy ý tưởng tiếp cận của hai nguyên lý là khác nhau, nhưng thực tế cho thấy, kết quả của các phép toán cơ bản (cộng, trừ, nhân, chia) là giống nhau. Đây là một điều rất thú vị đối với học viên khi thực hiện bài tập này.

Tham khảo

[1] L. A. Zadeh, *Fuzzy Sets*, 1965

[2] Cao Hoàng Trụ, *Trí Tuệ Nhân Tạo = Thông Minh + Giải Thuật*, 2010