Bibliothèque Fuzzy en SageMath, Documentation

Jérémy Marrez

Bibliothèque Fuzzy en SageMath

Documentation

Jérémy Marrez

La bibliothèque Fuzzy permet à la fois de modéliser les nombres flous dans les différentes représentations gauche-droite, et de résoudre des systèmes de polynômes à coefficients flous ou réels.

Nous verrons d'abord comment sont structurées les données floues, puis l'implantation de la méthode algébrique de résolution des systèmes polynomiaux à coefficients flous.

1 Les classes des nombres flous

1.1 La classe NombreFlou

La classe `NombreFlou` permet de modéliser dans la représentation en tuple des nombres flous avec des restrictions gauche et droite linéaires ou quadratiques. Ces nombres flous peuvent être non réduits, i.e. que leur noyau n'est pas forcément réduit à un point.

Le constructeur est `NombreFlou(leftMode, rightMode, leftSpread, rightSpread, leftType, rightType)`.

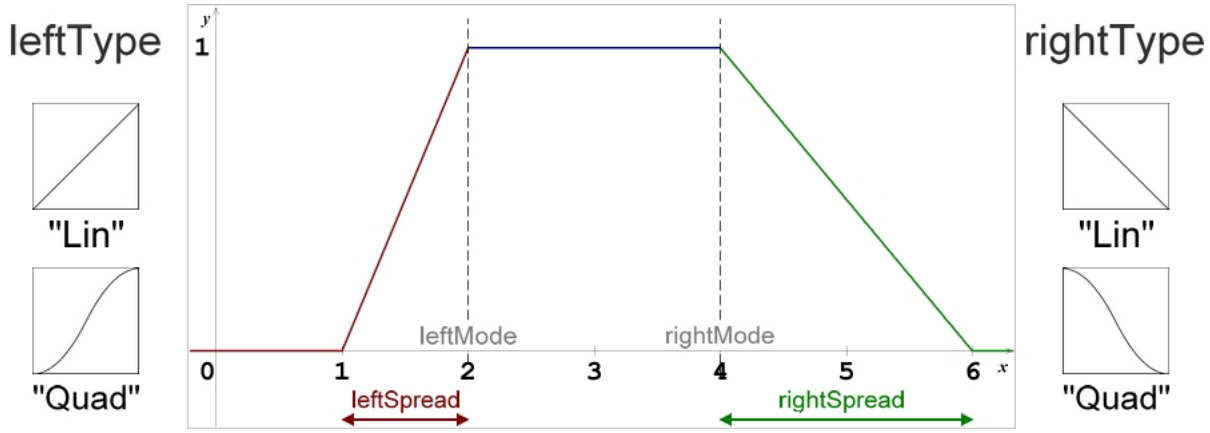
Une instance de cette classe possède les six propriétés suivantes :

- leftMode (un entier) : la borne gauche du noyau du nombre flou
- rightMode (un entier) : la borne droite du noyau du nombre flou
- leftSpread (un entier positif ou nul) : la propagation du nombre flou à gauche du noyau
- rightSpread (un entier positif ou nul) : la propagation du nombre flou à droite du noyau
- leftType (une chaîne de caractères) : le type de la restriction de la fontion d'appartenance à gauche du noyau
- rightType (une chaîne de caractères) : le type de la restriction de la fontion d'appartenance à droite du noyau

Ce ne sont pas toujours, à proprement parler, des nombres flous car leur noyau

n'est pas forcément réduit à un seul élément, mais dans la littérature, les nombres flous non réduits aux restrictions linéaires sont souvent référencés par abus de langage comme des nombres flous trapézoïdaux.

Une instance peut être schématisée comme suit



Pour déclarer un nombre flou, on écrit :

```
n1 = NombreFlou(3,4,2,1,"Lin","Quad")
```

Les différentes méthodes de cette classe permettent à la fois d'afficher et de dessiner les graphes de ses instances, mais aussi d'opérer des calculs sur ces instances et de les ordonner.

Pour commencer, la fonction `print(nombreFlou)` qui permet d'afficher le nombre flou sous la forme du quadruplet (leftMode, rightMode, leftSpread, rightSpread) et précise sa famille en fonction du type de ses restrictions.

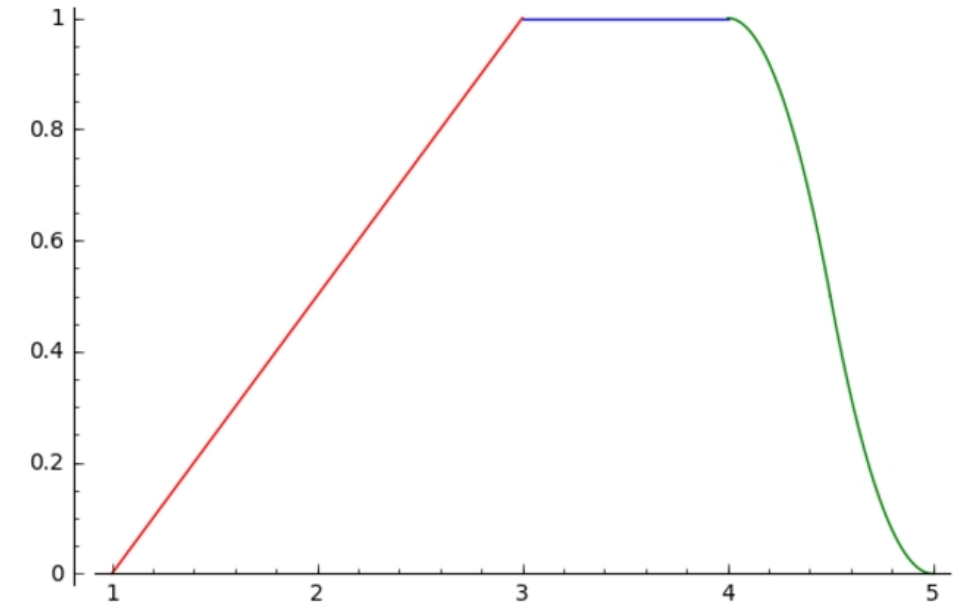
```
print(n1)
```

```
(3,4,2,1), Lin - Quad
```

Les fonctions `get_mode(self)`, `get_leftSpread(self)` et `get_rightSpread(self)` renvoient respectivement le mode, le leftSpread et le rightSpread du nombre flou.

La fonction `trace(self)` quant à elle, affiche le graphe de la fonction d'appartenance du nombre flou en fonction de son noyau, de son support et de sa famille (self représente l'objet sur lequel la fonction est appelée).

```
n1.trace()
```



Les instances de cette classe dont les restrictions sont linéaires bénéficient de redéfinitions des opérations binaires +, -, *, /, et celles avec des restrictions quadratiques de redéfinitions des opérations binaires +, -. À noter qu'ici, comme le noyau n'est pas toujours réduit à un seul élément, les calculs qui se font normalement sur le mode s'effectuent sur un intervalle. Pour ce faire, nous utilisons l'arithmétique des intervalles explicitée dans [2] pour la multiplication des nombres flous trapézoïdaux.

Chaque méthode commence par vérifier si les familles des deux opérandes sont bien compatibles pour l'opération concernée en appelant les fonctions `famillesEgales(self,autre)` ou `famillesSymetriques(self,autre)`. Deux familles peuvent être égales et symétriques dans le cas d'une même famille simple, simplement égales, simplement symétriques ou incompatibles.

```
n1 = NombreFlou(3,4,2,1,"Lin","Lin")
n2 = NombreFlou(1,5,1,3,"Lin","Lin")
n3 = NombreFlou(-2,3,5,8,"Lin","Quad")
n4 = NombreFlou(5,8,1,6,"Quad","Lin")
```

```
(NombreFlou.famillesEgales(n1,n2),
 NombreFlou.famillesSymetriques(n1,n2))
```

```
(True, True)
```

```
(NombreFlou.famillesEgales(n2,n3),
 NombreFlou.famillesSymetriques(n2,n3))
```

```
(False, False)
```

```
(NombreFlou.famillesEgales(n3,n4),
 NombreFlou.famillesSymetriques(n3,n4))
```

```
(False, True)
```

Les redéfinitions des opérations + et * vérifient que les familles des opérandes sont bien égales, auquel cas elles opèrent les calculs sur les instances de la classe. Si les familles ne sont pas égales, un message d'erreur est renvoyé.

```
n1 = NombreFlou(-3,5,2,6,"Lin","Quad")
n2 = NombreFlou(2,3,4,5,"Lin","Quad")
n3 = NombreFlou(5,8,2,4,"Quad","Lin")
```

```
print(n1 + n2)
```

```
(-1,8,6,11), Lin - Quad
```

```
print(n2 + n3)
```

```
L'opération ne peut aboutir car les familles des opérandes sont différentes
None
```

Les redéfinitions des opérations unaires - et $\sim$ calculent et renvoient respectivement l'opposé et l'inverse des nombres flous passés en paramètre.

```
n1 = NombreFlou(3,4,2,5,"Quad","Lin")
print(n1)
```

```
(3,4,2,5), Quad - Lin
```

```
n1_op = -n1
print(n1_op)
```

```
(-4,-3,5,2), Lin - Quad
```

```
n1_op_op = -n1_op
print(n1_op_op)
```

```
(3,4,2,5), Quad - Lin
```

```
n1 = NombreFlou(6,8,3,7,"Quad","Lin")
print(n1)
```

```
(6,8,3,7), Quad - Lin
```

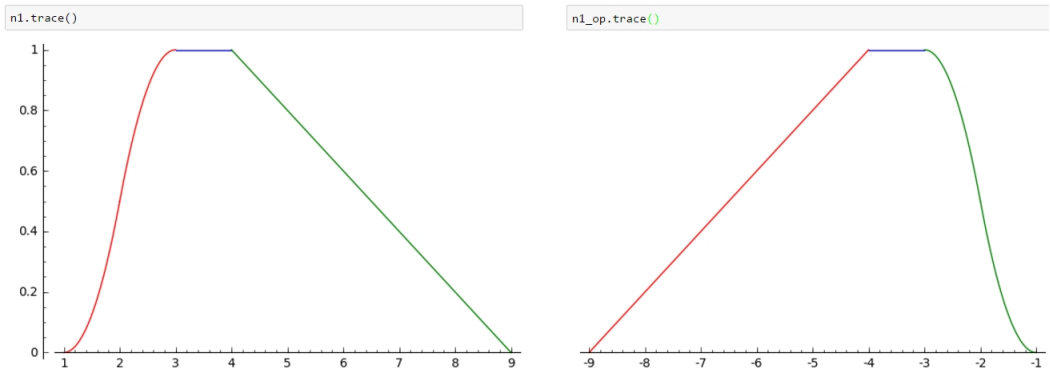
```
n1_inv = ~n1
print(n1_inv)
```

```
(1/8,1/6,7/64,1/12), Lin - Quad
```

```
n1_inv_inv = ~n1_inv
print(n1_inv_inv)
```

```
(6,8,3,7), Quad - Lin
```

Voici les graphes de deux nombres flous opposés :



Enfin, les redéfinition des opérations - et / vérifient que les familles des opérandes sont symétriques, auquel cas elles opèrent les calculs sur les instances de la classe. L'opérateur - calcule l'opposé de l'opérande de droite et l'additionne avec l'opérande de gauche (même principe avec / et l'inverse). Si les familles ne sont pas symétriques, un message d'erreur est renvoyé.

```
n1 = NombreFlou(-3,5,2,6,"Lin","Quad")
n2 = NombreFlou(2,3,4,5,"Lin","Quad")
n3 = NombreFlou(5,8,2,4,"Quad","Lin")
```

```
print(n1 - n2)
```

```
L'opération ne peut aboutir car les familles des opérandes ne sont pas symétriques
None
```

```
print(n2 - n3)
```

```
(-6,-2,8,7), Lin - Quad
```

Les méthodes de classe `maxi(nombreFlou1,nombreFlou2)` et `mini(nombreFlou1,nombreFlou2)` renvoient respectivement le max et le min des nombres flous passés en paramètre, qui peuvent être un nombre flou différent. Ces fonctions induisent un ordre partiel sur les nombres flous, implantés dans les fonctions `pluspetit(self,autre)` et `plusgrand(self,autre)` qui utilisent respectivement maxi et mini pour ordonner partiellement les nombres flous (tous ne sont pas comparables). La fonction `egale(self, autre)` vérifie que toutes les propriétés des deux opérandes sont identiques afin de savoir si elles sont bien égales.

```
n1 = NombreFlou(-4,7,3,4,"Lin","Lin")
n2 = NombreFlou(2,7,1,0,"Lin","Lin")
```

```
maximum = NombreFlou.maxi(n1,n2)
print(maximum)
```

```
(2,7,1,4), Lin - Lin
```

```
NombreFlou.plusgrand(n1,n2)
```

```
Ces deux arguments ne sont pas comparables pour cette relation d'ordre partiel
```

```
minimum = NombreFlou.mini(n1,n2)
print(minimum)
```

```
(-4,7,3,4), Lin - Lin
```

```
NombreFlou.pluspetit(n1,n2)
```

```
True
```

```
NombreFlou.equivalent(n1,n2)
```

```
False
```

La fonction `forme_parametrique(self,A=None)` prend en entrée un nombre flou sous forme tuple et un anneau de polynômes univariés en une variable fixée r (par défaut l'anneau $\mathbb{Q}[r]$). Elle passe le nombre flou dans sa forme paramétrique en calculant la fonction inverse en r de chaque restriction de la fonction d'appartenance (deux fonctions pour les restrictions de type quadratique). La fonction renvoie une instance de la classe `NombreFlouPM`, décrite plus loin, avec ces fonctions inverses en r, la famille du nombre flou en chaîne de caractère sous la forme "leftType - rightType", et l'anneau A si précisé.

```
n1 = NombreFlou(1,6,4,2,"Lin","Lin")
```

```
n1PM = n1.forme_parametrique()
print(n1PM)
```

```
[4*r - 3,-2*r + 8] , Lin - Lin
```

```
n2 = NombreFlou(2,4,1,3,"Lin","Quad")
```

```
n2PM = n2.forme_parametrique()
print(n2PM)
```

```
[r + 1,[3*sqrt(-1/2*r + 1/2) + 4, -3*sqrt(1/2)*sqrt(r) + 7]] , Lin - Quad
```

1.2 La classe NombreFlouRed

La classe `NombreFlouRed` hérite de la classe `NombreFlou`. Elle permet de modéliser dans la représentation en tuple des nombres flous avec des restrictions gauche et droite linéaires ou quadratiques. Ces nombres flous sont réduits, i.e. que leur noyau est réduit à un point qui est leur mode.

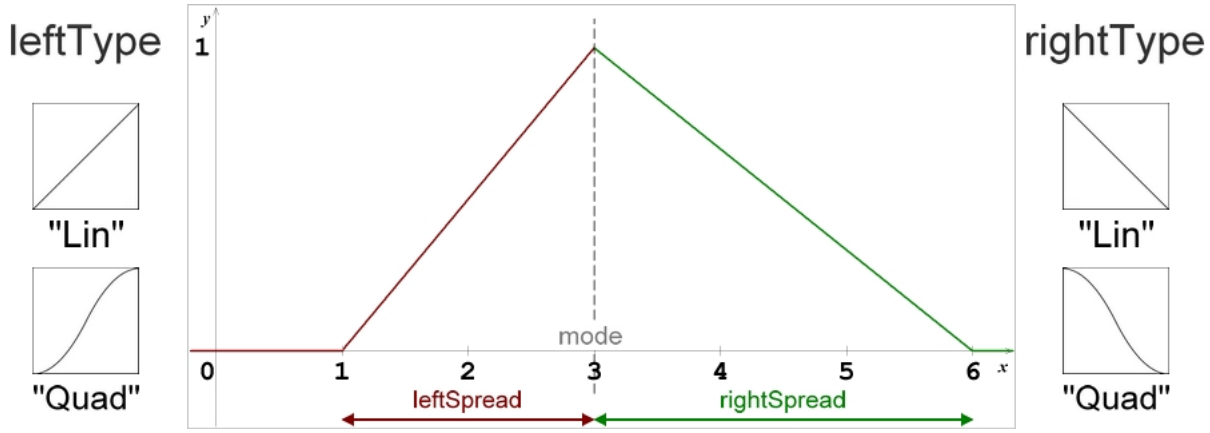
Le constructeur est `NombreFlouRed(mode, leftSpread, rightSpread, leftType, rightType)`.

Une instance de cette classe possède les cinq propriétés suivantes :

- mode (un entier) : le mode du nombre flou
- leftSpread (un entier positif ou nul) : la propagation du nombre flou à gauche du noyau
- rightSpread (un entier positif ou nul) : la propagation du nombre flou à droite du noyau
- leftType (une chaîne de caractères) : le type de la restriction de la fontion d'appartenance à gauche du noyau
- rightType (une chaîne de caractères) : le type de la restriction de la fontion

d'appartenance à droite du noyau

On a donc ici leftMode = rightMode = mode.
Une instance peut être schématisée comme suit



Certaines fonctions sont redéfinies dans cette classe. Pour commencer, la fonction d'affichage `print(nombreFlou)` qui affiche le nombre flou sous la forme du triplet (mode, leftSpread, rightSpread) et précise sa famille.

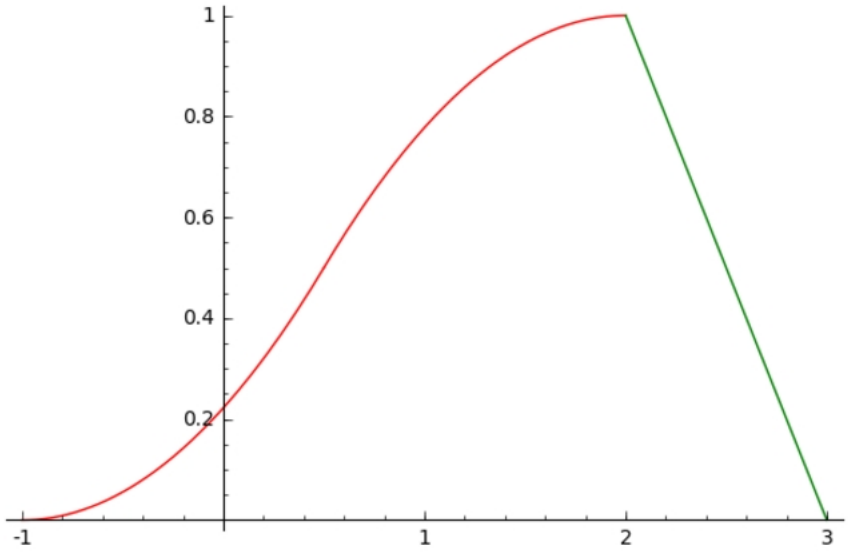
Les redéfinitions des opérations unaires - et $\sim$ sont aussi redéfinies pour prendre en compte le fait que le noyau est unique, et renvoient bien une instance de la classe `NombreFlouRed`.

Il en va de même avec les opérations binaires + et * dont les calculs sont exactement ceux vu en **??** avec le noyau réduit à un élément.

```
n1 = NombreFlouRed(2,3,1,"Quad","Lin")
print(n1)
```

```
(2,3,1), Quad - Lin
```

```
n1.trace()
```




```
n1_op = -n1
print(n1_op)
```

```
(-2,1,3), Lin - Quad
```

```
n2 = NombreFlouRed(4,3,3,"Quad","Lin")
```

```
res = n1 + n2
print(res)
```

```
(6,6,4), Quad - Lin
```

1.3 La classe NombreFlouPM

La classe `NombreFlouPM` permet de modéliser un nombre flou dans la représentation paramétrique.

Le constructeur est `NombreFlouPM(bas, haut, famille, A=None)`.

Une instance de cette classe possède les quatre propriétés suivantes :

- bas : la fonction inverse de la restriction gauche, en la variable para, qui donne pour chaque r dans [0,1] la borne gauche de la coupe-r correspondante
- haut : la fonction inverse de la restriction droite, en la variable para , qui

donne pour chaque r dans [0,1] la borne droite de la coupe-r correspondante
- famille (une chaîne de caractères) : la famille du nombre flou
- para : le générateur de l'anneau donné en paramètre (ou par défaut le générateur de l'anneau de polynômes à une variable sur le corps de rationnels).

Les attributs haut et bas sont chacun soit une fonction en *para* pour le cas triangulaire, soit un tableau de deux fonctions pour le cas quadratique.

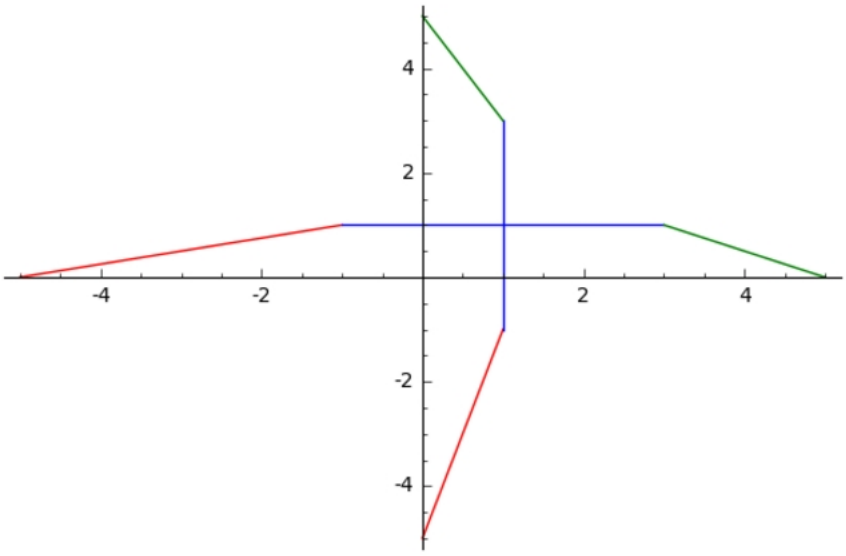
Dans cette classe, les différentes méthodes permettent également à la fois d'afficher et de dessiner les graphes de ses instances, et d'opérer des calculs sur ces instances et de les ordonner.

La fonction `print(nombreFlou)` permet d'afficher le nombre flou sous la forme d'une liste [bas, haut] et précise sa famille. La fonction `trace(self)` affiche la fonction d'appartenance du nombre flou en fonction des fonctions bas et haut. Le graphe obtenu est une rotation plane de 90ř par rapport à l'origine et une symétrie verticale par rapport à l'axe du mode du graphe de la fonction d'appartenance.

On le voit plus clairement en superposant les graphes des deux représentations d'un même nombre flou :

```
n1 = NombreFlou(-1,3,4,2,"Lin","Lin")
n1PM = n1.forme_parametrique()
```

```
n1.trace() + n1PM.trace()
```



La fonction `get_types(self)` renvoie, à partir de la famille du nombre flou paramétrique sous forme de chaîne de caractère "leftType - rightType", la liste de

ses types [leftType, rightType]. Cette fonction permet d'utiliser les fonctions `famillesEgales(self,autre)` ou `famillesSymetriques(self,autre)` exactement de la même façon que dans la classe `NombreFlou` pour vérifier la compatibilité des familles pour chaque opération.

Les redéfinitions des opérations binaires + et * (resp. - et /) peuvent alors vérifier que leurs opérandes ont bien des familles égales (resp. symétriques) avant d'effectuer les calculs de l'arithmétique floue vue en **??** pour la forme paramétrique.

On obtient ainsi un polymorphisme des opérations binaires + et - dans les deux représentations.

```
n1 = NombreFlou(1,2,2,1,"Lin","Quad")
n2 = NombreFlou(2,3,1,2,"Quad","Lin")
```

```
n3 = n1 - n2
print(res)
```

```
(-7,4,5,7), Lin - Quad
```

```
p1 = n1.forme_parametrique()
p2 = n2.forme_parametrique()
print(p1)
print(p2)
```

```
Symbolic Ring
[[2*sqrt(1/2)*sqrt(r) - 1, -2*sqrt(-1/2*r + 1/2) + 1],-r + 3] , Quad - Lin
[r + 1,[2*sqrt(-1/2*r + 1/2) + 3, -2*sqrt(1/2)*sqrt(r) + 5]] , Lin - Quad
```

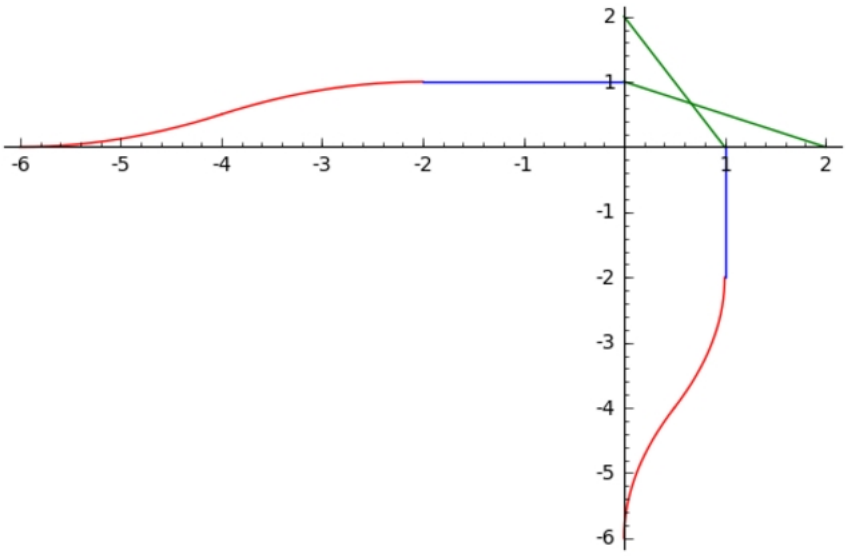
```
p3 = p1 - p2
print(p3)
```

```
[[4*sqrt(1/2)*sqrt(r) - 6, -4*sqrt(-1/2*r + 1/2) - 2],-2*r + 2] , Quad - Lin
```

```
n3PM = n3.forme_parametrique()
print(n3PM)
```

```
Symbolic Ring
[[4*sqrt(1/2)*sqrt(r) - 6, -4*sqrt(-1/2*r + 1/2) - 2],-2*r + 2] , Quad - Lin
```

```
n3.trace() + p3.trace()
```



Les méthodes de classe `maxi(nombreFlou1,nombreFlou2)` et `mini(nombreFlou1,nombreFlou2)` procèdent exactement comme les fonctions maxi et mini de la classe `NombreFlou`, en s'adaptant à la structure des nombres flous paramétriques, et induisent un ordre partiel sur les nombres flous. La fonction `egale(self, autre)` vérifie que les propriétés des instances sont identiques, et les fonctions `plusgrand(self,autre)` et `pluspetit(self,autre)` utilisent respectivement maxi et mini pour ordonner partiellement les nombres flous (tous ne sont pas comparables).

Un ordre total sur les nombres flous dans la forme paramétrique est donné via des calculs d'intégrales dans [1]. Cette méthode a été implantée dans la fonction `R(self, autre)` et est décrite ci-dessous :

Soit un nombre flou u de forme paramétrique $[\underline{u}(r), \overline{u}(r)]$, avec $0 \leq r \leq 1$, on définit :

$$Mag(u) = \frac{1}{2}\sum_{0}^{1}(\underline{u}(r) + \overline{u}(r) + \underline{u}(1) + \overline{u}(1))f(r)dr,$$

$$Momag(u) = \frac{1}{2}\sum_{0}^{1}(\underline{u}(r) - \overline{u}(r) + \underline{u}(1) - \overline{u}(1))dr$$

où la fonction poids $f(\alpha)$ est positive et croissante sur $[0, 1]$ avec $f(0) = 0$ et $\sum_0^1 f(\alpha)d\alpha = \frac{1}{2}$. Ici, on utilise $f(r) = r$.

Ces deux calculs sont effectués par les méthodes `momag(self)` et `momag(self)`.

À partir de 2 nombres flous sous forme paramétrique a et b, la méthode `R(self, autre)` calcule les quantités

$$R(a,\lambda) = Mag(a) + \lambda \cdot Momag(a) \text{ et } R(b,\lambda) = Mag(b) + \lambda \cdot Momag(b),$$

où

$$\lambda = \begin{cases} 0 & \text{si } Mag(a) \neq Mag(b), \\ 1 & \text{si } Mag(a) = Mag(b) \text{ et } z >= 0, \\ -1 & \text{si } Mag(a) = Mag(b) \text{ et } z < 0. \end{cases}$$

avec $z = \frac{a.bas(1)+a.haut(1)}{2}$, et on renvoie ces deux valeurs dans un tuple.

Alors, pour deux nombres flous a et b, on définit le rang de a et b comme suit :

- $R(a,\lambda) > R(b,\lambda) \Leftrightarrow a > b$,
- $R(a,\lambda) < R(b,\lambda) \Leftrightarrow a < b$,
- $R(a,\lambda) = R(b,\lambda) \Leftrightarrow a \sim b$.

2 Méthodes de résolution algébrique

Les méthodes listées ci-après sont celles implantées pour la résolution algébrique de systèmes de polynômes réels et entrant dans la mise en oeuvre de l'algorithme de Wu.

`classe(p)` : Retourne la classe d'un polynôme (le maximum parmi les indices des variables apparaissant dans p)

`degre(p)` : Retourne le degré d'un polynôme (le degré maximum en la variable dont l'indice est la classe de p, i.e. en la variable principale de p)

`init(p)` : Retourne le coefficient dominant en la variable principale de p

`tail(p)` : Renvoie la queue d'un polynôme p, i.e. p - son terme dominant

`estConstant(p,i)` : Renvoie True si p est constant en la variable d'indice i, et False sinon

`ordreRitt(a,b)` : Renvoie 1 si $a > b$, 0 si $a = b$, -1 si $a < b$ pour l'ordre de Ritt

`estReduit(a,b)` : Renvoie True si a est réduit par rapport à b, et False sinon

`basicSet(F)` : Calcule un ensemble basique d'un ensemble de polynômes F

`pseudoDiv(p,f)` : Effectue la pseudo-division de p par f et renvoie la liste $[quotient, reste, init(f), init(f)^j]$

`premEnsTri(p,T)` : Effectue la pseudo-division de p par tous les polynômes de l'ensemble T et renvoie le reste

`PREM(F,B)` : Effectue la pseudo-division de chaque polynôme de F par tous les polynômes de l'ensemble T et renvoie la liste des restes

`CharacSet(F)` : Calcule l'ensemble caractéristique de l'ensemble de polynômes F

`Wu(F)` : Algorithme de Wu : Calcule l'ensemble des ensembles caractéristiques à partir du système F

`prodInit(B)` : Renvoie le produit des initiaux des polynômes de l'ensemble B

`ensInit(Z)` : Renvoie la liste des produits des initiaux pour chaque ensemble de Z

3 Algorithme principal

L'utilisateur entre son système polynomial à coefficients flous sous la forme d'une liste de couples :

$$\begin{aligned} System = [&(\text{membre de gauche equation } 1, \text{membre de droite equation } 1),\\ &(\text{membre de gauche equation } 2, \text{membre de droite equation } 2),\\ &\vdots\\ &(\text{membre de gauche equation } s, \text{membre de droite equation } s)] \end{aligned}$$

où chaque membre d'une équation est également donné par une liste de couples :

membre d'une equation $=$ [(Coefficient flou monôme 1, terme monôme 1), ..., (Coefficient flou monôme m, terme monôme m)]

Par exemple, pour le système

$$F : \begin{cases} x + (-1, 1, 1) = (-2, 1, 1)y^2, \\ x + (3, 1, 1) = (2, 1, 1)y^2 \end{cases}$$

on écrit

```
systeme = [([(NombreFlouRed(1,0,0,"Quad","Quad"),x), (NombreFlouRed(-1,1,1,"Quad","Quad"),1)],
           [(NombreFlouRed(-2,1,1,"Quad","Quad"),y**2)]),
           ([(NombreFlouRed(1,0,0,"Quad","Quad"),x), (NombreFlouRed(3,1,1,"Quad","Quad"),1)],
           [(NombreFlouRed(2,1,1,"Quad","Quad"),y**2)])]
```

`resolution_reelle_systemes_flous(Systeme, n)` : Renvoie, à partir d'un système flou `Systeme` à s équations, donné comme ci-dessus, et d'un nombre de variables n, l'ensemble des solutions réelles exactes de `Systeme`. Pour cela, elle utilise principalement les fonctions suivantes.

`RealTransform(dep,I)` : à partir d'un système flou `dep` à s équations donné comme ci-dessus et d'une liste de signes `I` $\in \{-1, 1\}^n$, construit le système `SI` à coefficients réels à $3s$ équations.

`SolPos(SI)` : Renvoie à partir d'un système `SI` à coefficients réels à $3s$ équations, l'ensemble des solutions positives de `SI`. Elle fait appel pour cela à la fonction `Wu(systeme)` qui implante l'algorithme de Wu, i.e. renvoie l'ensemble Z des ensembles caractéristiques B du système réel passé en paramètre, puis à la fonction `get_zeros(`Z`)` qui à partir de l'ensemble Z des ensembles caractéristiques renvoie les zéros de la variété $V(\texttt{SI}) = \bigcup_{B \in Z} V(B/I_B)$ où $I_B = \prod_{b \in B} init(b)$, i.e. les solutions positives de `SI`.

4 Tests

Des fonctions `print` ont été appelées dans les différentes méthodes afin de pouvoir suivre correctement le déroulement des algorithmes.

4.1 Exemple 1 :

$$F : \begin{cases} x + (-1, 1, 1) = (-2, 1, 1)y^2, \\ x + (3, 1, 1) = (2, 1, 1)y^2 \end{cases}$$

```
systeme =  [([(NombreFlouRed(1,0,0,"Quad","Quad"),x), (NombreFlouRed(-1,1,1,"Quad","Quad"),1)], [(NombreFlouRed(-2,1,1,"Quad","Quad"),y**2)]),\
     ([(NombreFlouRed(1,0,0,"Quad","Quad"),x), (NombreFlouRed(3,1,1,"Quad","Quad"),1)], [(NombreFlouRed(2,1,1,"Quad","Quad"),y**2)])]
```

```
resolution_reelle_systemes_flous(systeme2,2)
```

```
Pour j =  0  :  I =  [-1, -1] ,  x négatif  et  y négatif
ColonnesDistinctes =  []
ListeSystemes =  []
C =  [-1, 1, 1, -1, 1, 1]
La colonne C est une nouvelle colonne, cpt est incrémenté, cpt =  0  et on ajoute C à ColonnesDistinctes à l'indice  0
S(I) = [-y^2 + 1, 2*y^2 - x - 1, -2*y^2 - x + 3]
Le système S(I) est un nouveau système, on l'ajoute à ListeSystemes à l'indice  0
On résout S(I)
Solutions positives de S(I) : set([(1, 1)])
On insère ces solutions dans lb à  l'incide 0
On ajoute dans sol le produit de Kronecker de I =  [-1, -1]  avec lb[ 0 ]
sol =  set([(-1, -1)])


Pour j =  1  :  I =  [-1, 1] ,  x négatif  et  y positif
ColonnesDistinctes =  [[-1, 1, 1, -1, 1, 1]]
ListeSystemes =  [[y^2 - 1, 2*y^2 - x - 1, 2*y^2 + x - 3]]
C =  [-1, 1, 1, -1, 1, 1]
La colonne C est déjà présente dans ColonnesDistinctes à l'indice  0
On ajoute dans sol le produit de Kronecker de I =  [-1, 1]  avec lb[ 0 ]
sol =  set([(-1, 1), (-1, -1)])


Pour j =  2  :  I =  [1, -1] ,  x positif  et  y négatif
ColonnesDistinctes =  [[-1, 1, 1, -1, 1, 1]]
ListeSystemes =  [[y^2 - 1, 2*y^2 - x - 1, 2*y^2 + x - 3]]
C =  [1, 1, 1, 1, 1, 1]
La colonne C est une nouvelle colonne, cpt est incrémenté, cpt =  1  et on ajoute C à ColonnesDistinctes à l'indice  1
S(I) = [2*y^2 + x - 1, -2*y^2 + x + 3, -y^2 + 1]
Le système S(I) est un nouveau système, on l'ajoute à ListeSystemes à l'indice  1
On résout S(I)
Solutions positives de S(I) : set([])
On insère ces solutions dans lb à  l'incide 1
On ajoute dans sol le produit de Kronecker de I =  [1, -1]  avec lb[ 1 ]
sol =  set([(-1, 1), (-1, -1)])


Pour j =  3  :  I =  [1, 1] ,  x positif  et  y positif
ColonnesDistinctes =  [[-1, 1, 1, -1, 1, 1], [1, 1, 1, 1, 1, 1]]
ListeSystemes =  [[y^2 - 1, 2*y^2 - x - 1, 2*y^2 + x - 3], [2*y^2 + x - 1, 2*y^2 - x - 3, y^2 - 1]]
C =  [1, 1, 1, 1, 1, 1]
La colonne C est déjà présente dans ColonnesDistinctes à l'indice  1
On ajoute dans sol le produit de Kronecker de I =  [1, 1]  avec lb[ 1 ]
sol =  set([(-1, 1), (-1, -1)])



{(-1, -1), (-1, 1)}
```

On obtient pour solution la variété $V = \{(x = -1, y \pm 1)\}$.

4.2 Exemple 2 :

$$F : \begin{cases} (2,1,1)xy + (3,1,1)x^2y^2 + (2,1,1)x^3y^3 = (7,3,3), \\ (5,1,1)xy + (2,3,1)x^2y^2 + (2,2,1)x^3y^3 = (9,6,3) \end{cases}$$

```
systeme2 =[([(NombreFlouRed(2,1,1,"Quad","Quad"),x*y), (NombreFlouRed(3,1,1,"Quad","Quad"),x**2*y**2), (NombreFlouRed(2,1,1,"Quad","Quad"),x**3*y**3)],\
      [(NombreFlouRed(7,3,3,"Quad","Quad"),1)]), ([(NombreFlouRed(5,1,1,"Quad","Quad"),x*y), (NombreFlouRed(2,3,1,"Quad","Quad"),x**2*y**2)\
      , (NombreFlouRed(2,2,1,"Quad","Quad"),x**3*y**3)], [(NombreFlouRed(9,6,3,"Quad","Quad"),1)])]
```

less than a minute ago
0.018 seconds

```
resolution_reelle_systemes_flous(systeme,2)
```

less than a minute ago
0.348 seconds

```
Pour j =  0  :  I =  [-1, -1] ,  x négatif  et  y négatif
ColonnesDistinctes =  []
ListeSystemes =  []
C =  [1, 1, 1, 1, 1, 1, 1, 1]
La colonne C est une nouvelle colonne, cpt est incrémenté, cpt =  0  et on ajoute C à ColonnesDistinctes à l'indice  0
S(I) = [2*x^3*y^3 + 2*x^2*y^2 + 5*x*y - 9, 2*x^3*y^3 + 3*x^2*y^2 + 2*x*y - 7, 2*x^3*y^3 + 3*x^2*y^2 + x*y - 6, x^3*y^3 + x^2*y^2 + x*y - 3]
Le système S(I) est un nouveau système, on l'ajoute à ListeSystemes à l'indice  0
On résout S(I)
Solutions positives de S(I) : set([(1/y, 'R+')])
On insère ces solutions dans lb à  l'incide 0
On ajoute dans sol le produit de Kronecker de I =  [-1, -1]  avec lb[ 0 ]
sol =  set([(1/y, 'R-')])


Pour j =  1  :  I =  [-1, 1] ,  x négatif  et  y positif
ColonnesDistinctes =  [[1, 1, 1, 1, 1, 1, 1, 1]]
ListeSystemes =  [[2*x^3*y^3 + 2*x^2*y^2 + 5*x*y - 9, 2*x^3*y^3 + 3*x^2*y^2 + 2*x*y - 7, 2*x^3*y^3 + 3*x^2*y^2 + x*y - 6, x^3*y^3 + x^2*y^2 + x*y - 3]]
C =  [-1, 1, -1, 1, -1, 1, -1, 1]
La colonne C est une nouvelle colonne, cpt est incrémenté, cpt =  1  et on ajoute C à ColonnesDistinctes à l'indice  1
S(I) = [-2*x^3*y^3 + 3*x^2*y^2 - 2*x*y - 7, -2*x^3*y^3 + 3*x^2*y^2 - x*y - 6, -x^3*y^3 + x^2*y^2 - x*y - 3, -2*x^3*y^3 + 2*x^2*y^2 - 5*x*y - 9]
Le système S(I) est un nouveau système, on l'ajoute à ListeSystemes à l'indice  1
On résout S(I)
Solutions positives de S(I) : set([])
On insère ces solutions dans lb à  l'incide 1
On ajoute dans sol le produit de Kronecker de I =  [-1, 1]  avec lb[ 1 ]
sol =  set([(1/y, 'R-')])


Pour j =  2  :  I =  [1, -1] ,  x positif  et  y négatif
ColonnesDistinctes =  [[1, 1, 1, 1, 1, 1, 1, 1], [-1, 1, -1, 1, -1, 1, -1, 1]]
ListeSystemes =  [[2*x^3*y^3 + 2*x^2*y^2 + 5*x*y - 9, 2*x^3*y^3 + 3*x^2*y^2 + 2*x*y - 7, 2*x^3*y^3 + 3*x^2*y^2 + x*y - 6, x^3*y^3 + x^2*y^2 + x*y - 3], [2*x^3*y^3
- 3*x^2*y^2 + 2*x*y + 7, 2*x^3*y^3 - 3*x^2*y^2 + x*y + 6, x^3*y^3 - x^2*y^2 + x*y + 3, 2*x^3*y^3 - 2*x^2*y^2 + 5*x*y + 9]]
C =  [-1, 1, -1, 1, -1, 1, -1, 1]
La colonne C est déjà présente dans ColonnesDistinctes à l'indice  1
On ajoute dans sol le produit de Kronecker de I =  [1, -1]  avec lb[ 1 ]
sol =  set([(1/y, 'R-')])


Pour j =  3  :  I =  [1, 1] ,  x positif  et  y positif
ColonnesDistinctes =  [[1, 1, 1, 1, 1, 1, 1, 1], [-1, 1, -1, 1, -1, 1, -1, 1]]
ListeSystemes =  [[2*x^3*y^3 + 2*x^2*y^2 + 5*x*y - 9, 2*x^3*y^3 + 3*x^2*y^2 + 2*x*y - 7, 2*x^3*y^3 + 3*x^2*y^2 + x*y - 6, x^3*y^3 + x^2*y^2 + x*y - 3], [2*x^3*y^3
- 3*x^2*y^2 + 2*x*y + 7, 2*x^3*y^3 - 3*x^2*y^2 + x*y + 6, x^3*y^3 - x^2*y^2 + x*y + 3, 2*x^3*y^3 - 2*x^2*y^2 + 5*x*y + 9]]
C =  [1, 1, 1, 1, 1, 1, 1, 1]
La colonne C est déjà présente dans ColonnesDistinctes à l'indice  0
On ajoute dans sol le produit de Kronecker de I =  [1, 1]  avec lb[ 0 ]
sol =  set([(1/y, 'R+'), (1/y, 'R-')])


{(1/y, 'R+'), (1/y, 'R-')}
```

On obtient pour solution la variété $V = \{(x = \frac{1}{y}, y) \mid y \in \mathbb{R}\backslash\{0\})\}$.

Références

[1] R Ezzati, S Khezerloo, and S Ziari. Application of parametric form for ranking of fuzzy numbers. *Iranian Journal of Fuzzy Systems*, 12(1) :59–74, 2015.

[2] A Taleshian and S Rezvani. Multiplication operation on trapezoidal fuzzy numbers. 2011.