



République Algérienne Démocratique et Populaire  
Ministère de l'Enseignement Supérieur et de la Recherche  
Scientifique



**Université des Sciences et de la Technologie Houari Boumediene**

Faculté d'Informatique

Département Intelligence Artificielle et Sciences des Données Spécialité :

Master Systèmes Informatiques Intelligents

Module :

REPRÉSENTATION DES CONNAISSANCES ET LE RAISONNEMENT 2

---

## **Rapport TP :** **Représentation des Connaissances et Raisonnement 2**

---

**Présenté par :**

Ghamraci Rahil

Benzemrane Lydia



1.1	Exemple d'exécution . . . . .	2
1.2	Résultat d'exécution . . . . .	2
1.3	Symbols utilisés . . . . .	4
1.4	Masses de l'expert 1 . . . . .	4
1.5	Masses de l'expert 2 . . . . .	5
1.6	Masses de l'expert 3 . . . . .	5
1.7	Masses de l'expert 4 . . . . .	6
1.8	degrés de croyances et de possibilités . . . . .	7
1.9	résultats . . . . .	7
1.10	combinaison des masses . . . . .	8
1.11	résultat de la combinaison . . . . .	8
1.12	distributions de masses de l'exemple F1 . . . . .	9
1.13	distributions de masses de l'exemple F1 . . . . .	10
1.14	degrés de croyances et de possibilités . . . . .	10
1.15	résultats . . . . .	11
1.16	combinaison des masses . . . . .	11
1.17	résultat de la combinaison . . . . .	11
2.1	Imporation bibliothèque . . . . .	12
2.2	Définition des domaines . . . . .	13
2.3	Fuzzification de h . . . . .	13
2.4	Résultat de Fuzzification de h . . . . .	13
2.5	Fuzzification de dh . . . . .	14
2.6	Résultat de Fuzzification de dh . . . . .	14
2.7	Fuzzification de u . . . . .	14
2.8	Résultat de Fuzzification de u . . . . .	14
2.9	Base de règles . . . . .	15
2.10	Fuzzification . . . . .	15
2.11	l'inférence . . . . .	16
2.12	Résultat d'inférence de l'exemple 1 . . . . .	16
2.13	défuzzification de l'exemple 1 . . . . .	17
2.14	Résultat de défuzzification de l'exemple 1 . . . . .	17
2.15	calcul du centre de gravité de l'exemple 1 . . . . .	17

2.16	Définition des domaines . . . . .	18
2.17	Fuzzification de H . . . . .	19
2.18	Résultat de Fuzzification de H . . . . .	19
2.19	Fuzzification de R . . . . .	19
2.20	Résultat de Fuzzification de R . . . . .	19
2.21	Fuzzification de O . . . . .	20
2.22	Résultat de Fuzzification de O . . . . .	20
2.23	Base de règles 2 . . . . .	20
2.24	Fuzzification 2 . . . . .	21
2.25	l'inférence 2 . . . . .	21
2.26	Résultat d'inférence de l'exemple 2 . . . . .	22
2.27	Résultat de défuzzification de l'exemple 2 . . . . .	22
2.28	calcul du centre de gravité de l'exemple 2 . . . . .	22
3.1	Imporation bibiliothèque . . . . .	23
3.2	Exemple poly-arbre . . . . .	24
3.3	Définition structure . . . . .	24
3.4	Connections entre les noeuds . . . . .	25
3.5	Connections entre les noeuds . . . . .	25
3.6	Affectation des connections au poly arbre . . . . .	25
3.7	Résultat . . . . .	26
3.8	Calcul probabilités . . . . .	26
3.9	Calcul de la probabilité . . . . .	27
3.10	Résultat . . . . .	27
3.11	Calcul de la probabilité . . . . .	27
3.12	Résultat . . . . .	27
3.13	Graphe à connexions multiples . . . . .	28
3.14	distribution conditionnelle . . . . .	29
3.15	Définition structure . . . . .	30
3.16	Connections entre les noeuds . . . . .	31
3.17	Connections entre les noeuds . . . . .	31
3.18	Affectation des connections au graph . . . . .	32
3.19	Résultat . . . . .	32
3.20	Calcul probabilités . . . . .	33
3.21	Résultats du calcul des probabilité . . . . .	34
3.22	Définition structure . . . . .	35
3.23	Définition structure . . . . .	37
3.24	Connections entre les noeuds . . . . .	37
3.25	Connections entre les noeuds . . . . .	37
3.26	Affectation des connections au graph . . . . .	38
3.27	Calcul probabilités . . . . .	38
3.28	Résultats du calcul des probabilité . . . . .	39

# CHAPITRE 1

## TP1 : FONCTIONS DE CROYANCES

### 1.1 Introduction

Ce chapitre aborde le premier travail pratique sur les fonctions de croyances, un concept clé dans la théorie des croyances, souvent utilisée pour gérer l'incertitude et agréger des avis d'experts dans des situations ambiguës. À travers cet exercice, nous explorons les principes de base de la théorie de Dempster-Shafer, en mettant en œuvre la bibliothèque Python pyds pour simuler et calculer des degrés de croyances. Les différentes étapes de ce TP nous conduisent à tester un exemple réel de prise de décision sous incertitude : les avis divergents des experts sur les possibles origines de la transmission du SRAS-CoV-2 à l'homme, ainsi qu'une prédiction dans le cadre d'une compétition de Formule 1. Ce TP nous permettra de comprendre les fondements de la distribution des masses et les méthodes de combinaison de croyances.

## 1.2 Étape 1

### 1.2.1 Tester l'outil pyds

On commence par tester la bibliothèque pyds de python.

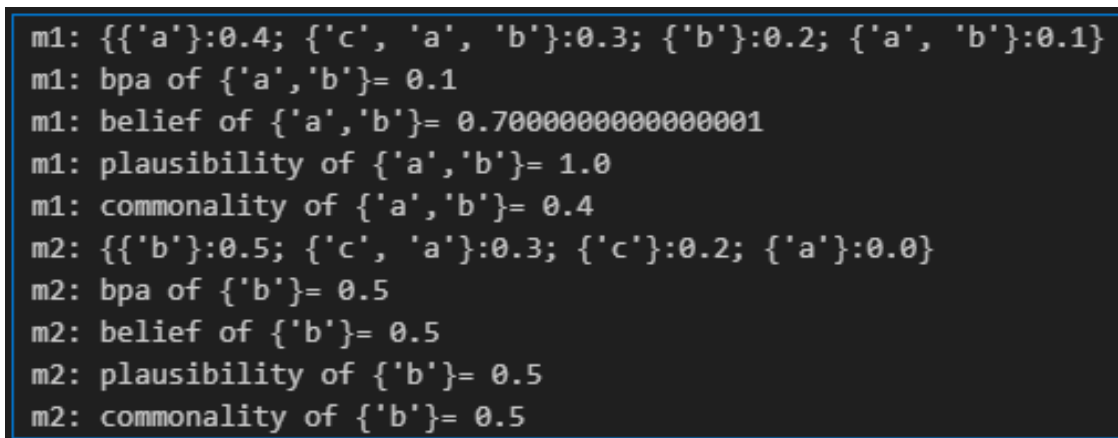


```

1 # testing the lib with an exemple provided in the github https://github.com/reineking/pyds/blob/master/examples.py
2 from pyds import MassFunction
3
4 m1 = MassFunction({'a':0.4, 'b':0.2, 'ab':0.1, 'abc':0.3})
5 m2 = MassFunction({'b':0.5, 'c':0.2, 'ac':0.3, 'a':0.0})
6 print("m1:",m1)
7 print("m1: bpa of {'a','b'}=", m1['ab'])
8 print("m1: belief of {'a','b'}=", m1.bel('ab'))
9 print("m1: plausibility of {'a','b'}=", m1.pl('ab'))
10 print("m1: commonality of {'a','b'}=", m1.q('ab'))
11 print("m2:",m2)
12 print("m2: bpa of {'b'}=", m2['b'])
13 print("m2: belief of {'b'}=", m2.bel('b'))
14 print("m2: plausibility of {'b'}=", m2.pl('b'))
15 print("m2: commonality of {'b'}=", m2.q('b'))

```

FIGURE 1.1 – Exemple d'exécution



```

m1: {'a':0.4; {'c', 'a', 'b'}:0.3; {'b'}:0.2; {'a', 'b'}:0.1}
m1: bpa of {'a','b'}= 0.1
m1: belief of {'a','b'}= 0.7000000000000001
m1: plausibility of {'a','b'}= 1.0
m1: commonality of {'a','b'}= 0.4
m2: {'b':0.5; {'c', 'a'}:0.3; {'c'}:0.2; {'a'}:0.0}
m2: bpa of {'b'}= 0.5
m2: belief of {'b'}= 0.5
m2: plausibility of {'b'}= 0.5
m2: commonality of {'b'}= 0.5

```

FIGURE 1.2 – Résultat d'exécution

## 1.3 Étape 2

### 1.3.1 Implementation d'un exemple du TD

Dans cette étape, nous allons tester l'exemple du TD portant sur les sources potentielles de la transmission du SRAS-CoV-2 à l'homme.

Nous rappelons que les avis des différents experts étaient répartis comme suit :

- **L'Organisation mondiale de la santé (OMS)** stipule que l'origine est due :
  - à une transmission de l'animal à l'homme (soit directe, soit via un hôte animal intermédiaire) à 55%,
  - à une transmission par certains aliments surgelés dans la chaîne du froid ou à une éventuelle évasion d'un laboratoire à 30%,
  - à une transmission directe de l'animal à l'homme à 13%.
- **Les experts chinois** affirment que le virus est causé par une transmission de l'animal à l'homme via un hôte animal intermédiaire à 68%. Par ailleurs, ils excluent l'hypothèse que le virus ait échappé d'un laboratoire.
- **L'administration américaine** atteste qu'une éventuelle évasion d'un laboratoire en est responsable à 75%.
- **L'opinion publique** est dans l'ignorance totale.

Nous commençons par la distribution des masses.

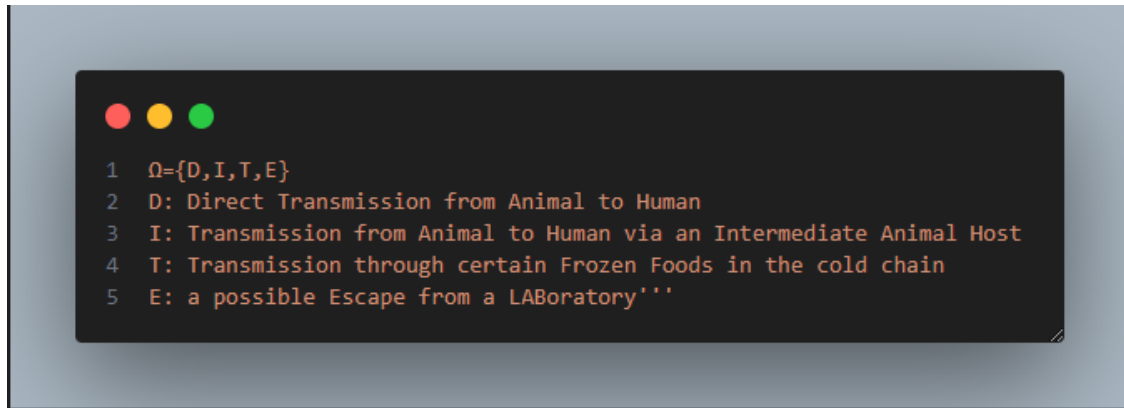


FIGURE 1.3 – Symbols utilisés

La distribution des masses du premier expert.

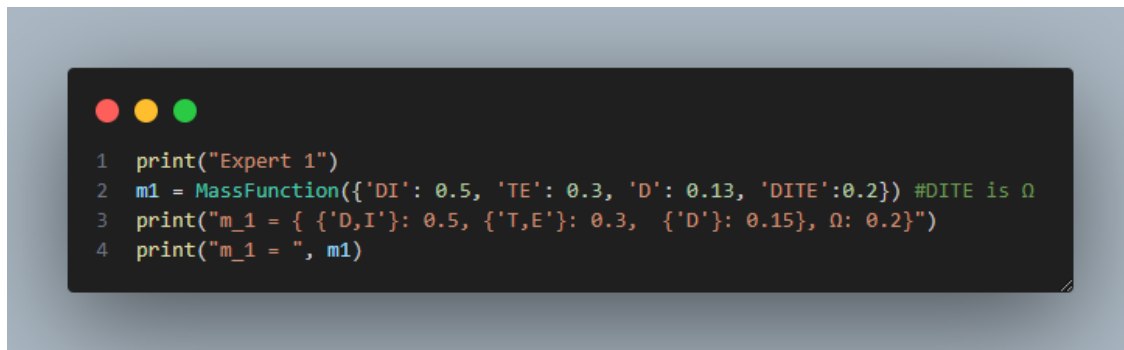
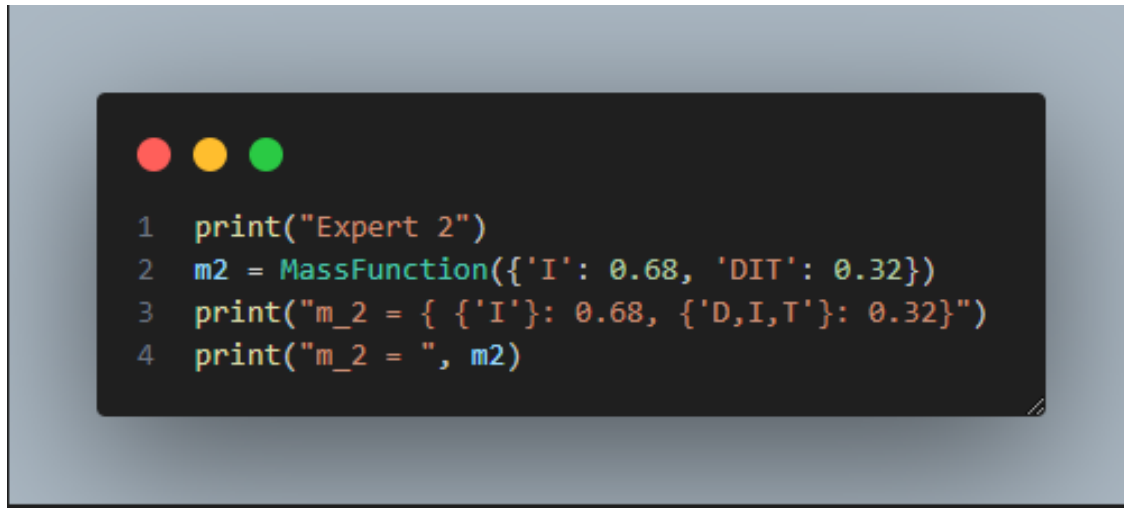


FIGURE 1.4 – Masses de l'expert 1



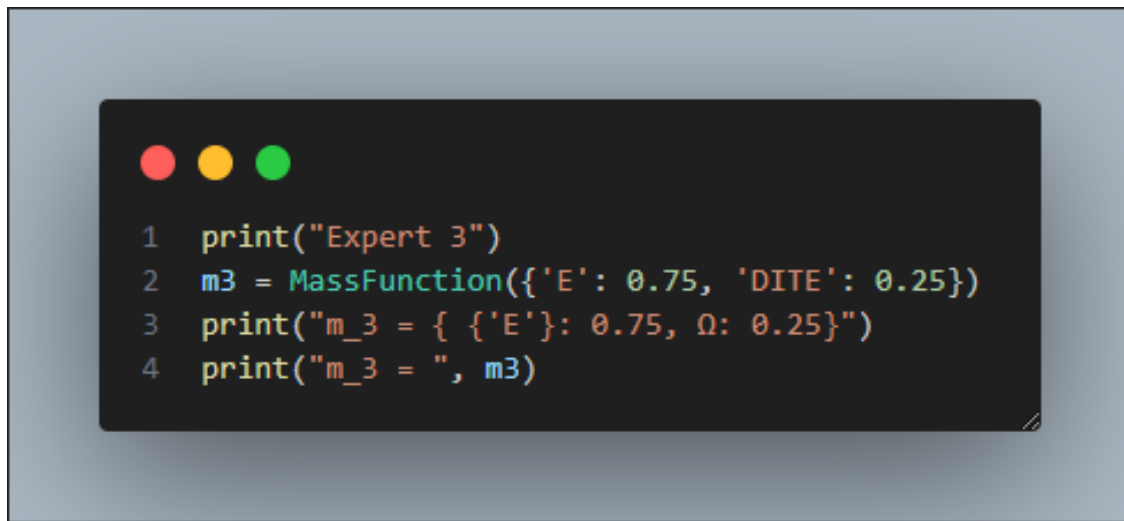
La distribution des masses du deuxième expert.

A terminal window with a dark background and light blue text. It contains four lines of Python code. The first line prints "Expert 2". The second line creates a MassFunction object m2 with two entries: 'I' with a mass of 0.68 and 'DIT' with a mass of 0.32. The third line prints a dictionary representation of m2. The fourth line prints the variable m2.

```
1 print("Expert 2")
2 m2 = MassFunction({'I': 0.68, 'DIT': 0.32})
3 print("m_2 = { {'I': 0.68, {'D,I,T': 0.32}}")
4 print("m_2 = ", m2)
```

FIGURE 1.5 – Masses de l'expert 2

La distribution des masses du troisième expert.

A terminal window with a dark background and light blue text. It contains four lines of Python code. The first line prints "Expert 3". The second line creates a MassFunction object m3 with two entries: 'E' with a mass of 0.75 and 'DITE' with a mass of 0.25. The third line prints a dictionary representation of m3. The fourth line prints the variable m3.

```
1 print("Expert 3")
2 m3 = MassFunction({'E': 0.75, 'DITE': 0.25})
3 print("m_3 = { {'E': 0.75, Ω: 0.25}}")
4 print("m_3 = ", m3)
```

FIGURE 1.6 – Masses de l'expert 3

La distribution des masses du quatrième expert.

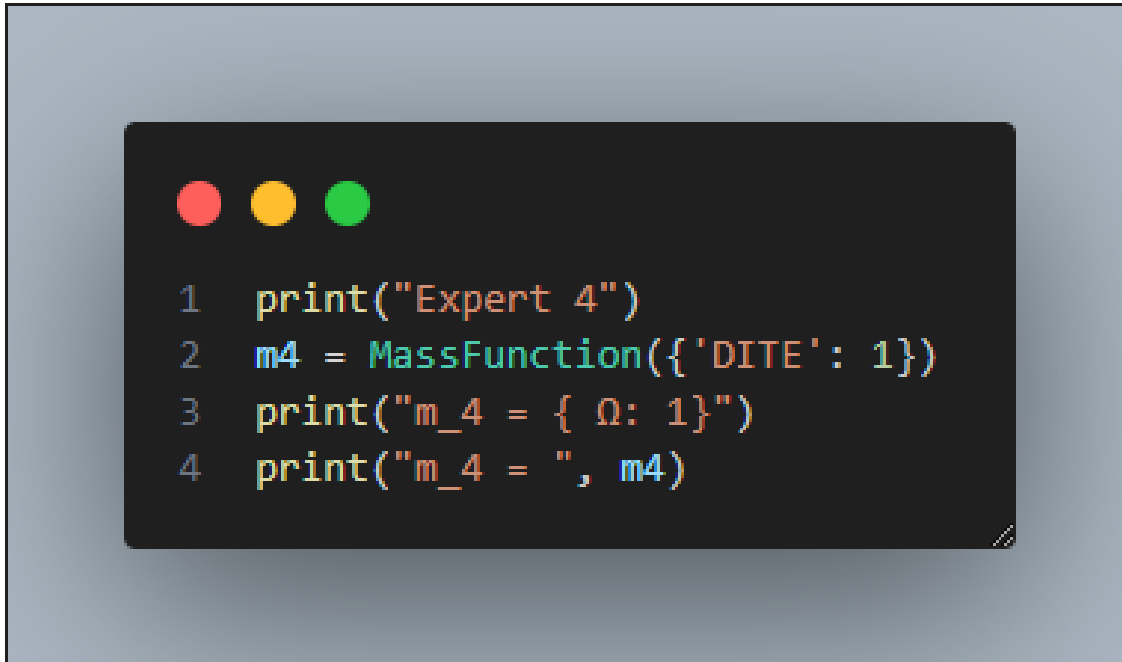


FIGURE 1.7 – Masses de l'expert 4

Le calcul des degrés de croyance et de possibilité.

```

1 print("belief and plausibilit of the first expert")
2 print("bel_1 = ", m1.bel())
3 print("pl_1 = ", m1.pl())
4
5 print("belief and plausibilit of the second expert")
6 print("bel_2 = ", m2.bel())
7 print("pl_2 = ", m2.pl())
8
9 print("belief and plausibilit of the third expert")
10 print("bel_3 = ", m3.bel())
11 print("pl_3 = ", m3.pl())

```

FIGURE 1.8 – degrés de croyances et de possibilités

Le résultat de calcul des degrés de croyance et de possibilité.

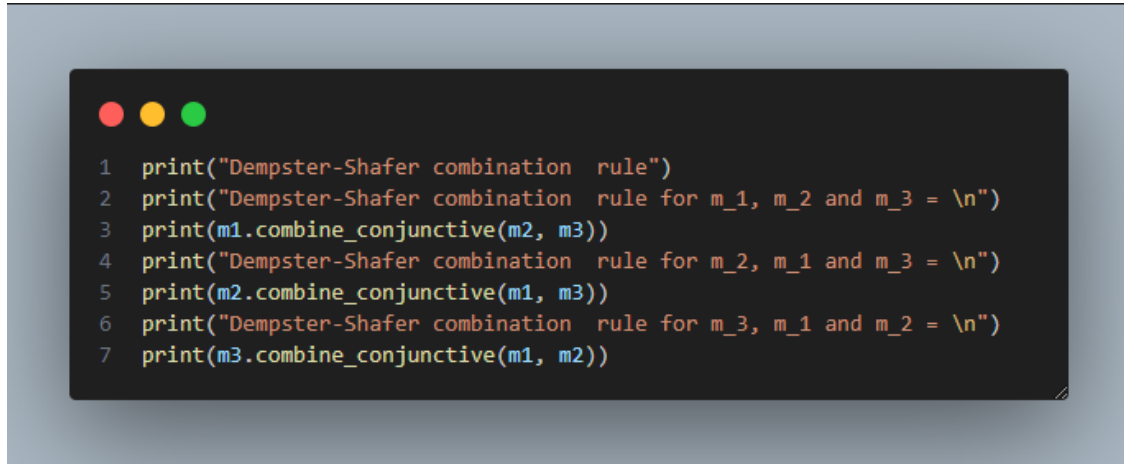
```

belief and plausibilit of the first expert
bel_1 = {frozenset(): 0.0, frozenset({'D'}): 0.13, frozenset({'I'}): 0.0, frozenset({'E'}): 0.0, frozenset({'T'}): 0.0, frozenset({'D', 'I'}): 0.0, frozenset({'D', 'E'}): 0.0, frozenset({'D', 'T'}): 0.0, frozenset({'I', 'E'}): 0.0, frozenset({'I', 'T'}): 0.0, frozenset({'E', 'T'}): 0.0, frozenset({'D', 'I', 'E'}): 0.0, frozenset({'D', 'I', 'T'}): 0.0, frozenset({'D', 'E', 'T'}): 0.0, frozenset({'I', 'E', 'T'}): 0.0, frozenset({'D', 'I', 'E', 'T'}): 0.0}
pl_1 = {frozenset(): 0.0, frozenset({'D'}): 0.8300000000000001, frozenset({'I'}): 0.7, frozenset({'E'}): 0.5, frozenset({'T'}): 0.0, frozenset({'D', 'I'}): 0.83, frozenset({'D', 'E'}): 0.5, frozenset({'D', 'T'}): 0.0, frozenset({'I', 'E'}): 0.5, frozenset({'I', 'T'}): 0.0, frozenset({'E', 'T'}): 0.0, frozenset({'D', 'I', 'E'}): 0.83, frozenset({'D', 'I', 'T'}): 0.0, frozenset({'D', 'E', 'T'}): 0.5, frozenset({'I', 'E', 'T'}): 0.0, frozenset({'D', 'I', 'E', 'T'}): 0.0}
belief and plausibilit of the second expert
bel_2 = {frozenset(): 0.0, frozenset({'D'}): 0.0, frozenset({'I'}): 0.0, frozenset({'E'}): 0.68, frozenset({'T'}): 0.0, frozenset({'D', 'I'}): 0.0, frozenset({'D', 'E'}): 0.0, frozenset({'D', 'T'}): 0.0, frozenset({'I', 'E'}): 0.68, frozenset({'I', 'T'}): 0.0, frozenset({'E', 'T'}): 0.0, frozenset({'D', 'I', 'E'}): 0.0, frozenset({'D', 'I', 'T'}): 0.0, frozenset({'D', 'E', 'T'}): 0.0, frozenset({'I', 'E', 'T'}): 0.0, frozenset({'D', 'I', 'E', 'T'}): 0.0}
pl_2 = {frozenset(): 0.0, frozenset({'D'}): 0.32, frozenset({'I'}): 0.32, frozenset({'E'}): 1.0, frozenset({'T'}): 0.32, frozenset({'D', 'I'}): 0.32, frozenset({'D', 'E'}): 0.32, frozenset({'D', 'T'}): 0.32, frozenset({'I', 'E'}): 1.0, frozenset({'I', 'T'}): 0.32, frozenset({'E', 'T'}): 0.32, frozenset({'D', 'I', 'E'}): 0.32, frozenset({'D', 'I', 'T'}): 0.32, frozenset({'D', 'E', 'T'}): 0.32, frozenset({'I', 'E', 'T'}): 0.32, frozenset({'D', 'I', 'E', 'T'}): 0.32}
belief and plausibilit of the third expert
bel_3 = {frozenset(): 0.0, frozenset({'D'}): 0.0, frozenset({'I'}): 0.0, frozenset({'E'}): 0.75, frozenset({'T'}): 0.0, frozenset({'D', 'I'}): 0.0, frozenset({'D', 'E'}): 0.0, frozenset({'D', 'T'}): 0.0, frozenset({'I', 'E'}): 0.75, frozenset({'I', 'T'}): 0.0, frozenset({'E', 'T'}): 0.0, frozenset({'D', 'I', 'E'}): 0.0, frozenset({'D', 'I', 'T'}): 0.0, frozenset({'D', 'E', 'T'}): 0.0, frozenset({'I', 'E', 'T'}): 0.0, frozenset({'D', 'I', 'E', 'T'}): 0.0}
pl_3 = {frozenset(): 0.0, frozenset({'D'}): 0.25, frozenset({'I'}): 0.25, frozenset({'E'}): 1.0, frozenset({'T'}): 0.25, frozenset({'D', 'I'}): 0.25, frozenset({'D', 'E'}): 0.25, frozenset({'D', 'T'}): 0.25, frozenset({'I', 'E'}): 1.0, frozenset({'I', 'T'}): 0.25, frozenset({'E', 'T'}): 0.25, frozenset({'D', 'I', 'E'}): 0.25, frozenset({'D', 'I', 'T'}): 0.25, frozenset({'D', 'E', 'T'}): 0.25, frozenset({'I', 'E', 'T'}): 0.25, frozenset({'D', 'I', 'E', 'T'}): 0.25}

```

FIGURE 1.9 – résultats

La Combinaison de Dempster-Shafer.



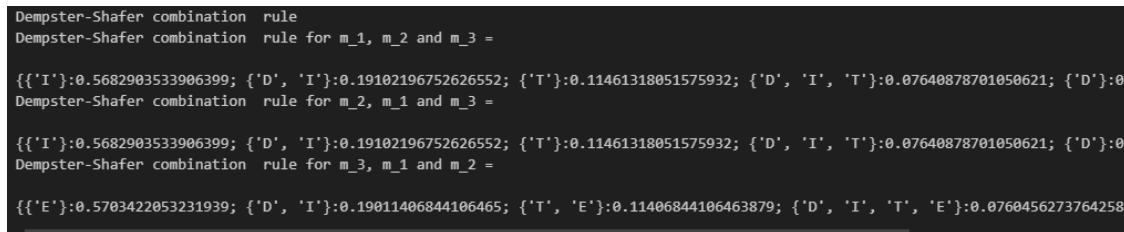
```

1 print("Dempster-Shafer combination rule")
2 print("Dempster-Shafer combination rule for m_1, m_2 and m_3 = \n")
3 print(m1.combine_conjunctive(m2, m3))
4 print("Dempster-Shafer combination rule for m_2, m_1 and m_3 = \n")
5 print(m2.combine_conjunctive(m1, m3))
6 print("Dempster-Shafer combination rule for m_3, m_1 and m_2 = \n")
7 print(m3.combine_conjunctive(m1, m2))

```

FIGURE 1.10 – combinaison des masses

Le résultat de la combinaison.



```

Dempster-Shafer combination rule
Dempster-Shafer combination rule for m_1, m_2 and m_3 =
{{'I':0.5682903533906399; {'D', 'I':0.19102196752626552; {'T':0.11461318051575932; {'D', 'I', 'T':0.07640878701050621; {'D':0
Dempster-Shafer combination rule for m_2, m_1 and m_3 =
{{'I':0.5682903533906399; {'D', 'I':0.19102196752626552; {'T':0.11461318051575932; {'D', 'I', 'T':0.07640878701050621; {'D':0
Dempster-Shafer combination rule for m_3, m_1 and m_2 =
{{'E':0.5703422053231939; {'D', 'I':0.19011406844106465; {'T', 'E':0.11406844106463879; {'D', 'I', 'T', 'E':0.0760456273764258

```

FIGURE 1.11 – résultat de la combinaison

En conclusion, l'hypothèse de la transmission d'animal à l'homme via un hôte intermédiaire est la plus soutenue.

## 1.4 Étape 3

### 1.4.1 Modélisation d'un exemple


Pour créer un exemple de Dempster-Shafer pour prédire le vainqueur de la F1 sur des bases réalistes et cohérentes, nous pouvons considérer les performances récentes, notamment lors du dernier Grand Prix de Mexico, où Carlos Sainz de Ferrari a remporté la victoire, suivi de Lando Norris en deuxième position et de Charles Leclerc en troisième. Max Verstappen, généralement l'un des meilleurs performeurs, a été pénalisé deux fois et a terminé en sixième position, ce qui pourrait influencer la probabilité attribuée par certains experts concernant ses chances de remporter la prochaine course.

L'expert 1 pense que Max Verstappen gagnera avec une probabilité de 70%. Bien que Verstappen ait été un favori constant, ses récentes pénalités et ses problèmes de performance suggèrent qu'il pourrait ne pas être aussi susceptible de décrocher la première place qu'auparavant. Par conséquent, la probabilité est légèrement inférieure à celle d'avant.

L'expert 2 attribue une confiance de 50% à la victoire de Max Verstappen, mais considère également Lando Norris et Charles Leclerc comme des potentiels vainqueurs, leur assignant une croyance combinée de 40

L'expert 3 attribue 55% à Carlos Sainz, en raison de sa récente victoire au Mexique, démontrant sa capacité à décrocher une autre victoire. De plus, cet expert attribue 35% à Lewis Hamilton, qui a fait preuve de résilience et de compétences stratégiques en course malgré des difficultés récentes.

La distribution des masses des différents experts.



```

1  print("Expert 1")
2  m1 = MassFunction({'M': 0.7, 'MLCSH': 0.3}) #MLCSH is Ω
3  print("m_1 = { {'M'}: 0.7,  Ω: 0.3}")
4  print("m_1 = ", m1)
5
6  print("Expert 2")
7  m2 = MassFunction({'M': 0.5, 'LC': 0.4, 'MLCSH':0.1})
8  print("m_2 = { {'M'}: 0.5, {'L,C'}: 0.4} ,  Ω: 0.1}")
9  print("m_2 = ", m2)
10
11 print("Expert 3")
12 m3 = MassFunction({'S': 0.55, 'H': 0.35 , 'MLCSH':0.1})
13 print("m_3 = { {'S'}: 0.55, {'S'}: 0.35, Ω: 0.1}")
14 print("m_3 = ", m3)

```

FIGURE 1.12 – distributions de masses de l'exemple F1

Résultat d'exécution.

```

Expert 1
m_1 = { {'M'}: 0.7,  Ω: 0.3}
m_1 = { {'M'}:0.7; {'S', 'H', 'C', 'L', 'M'}:0.3}
Expert 2
m_2 = { {'M'}: 0.5, {'L,C'}: 0.4} ,  Ω: 0.1}
m_2 = { {'M'}:0.5; {'C', 'L'}:0.4; {'S', 'H', 'C', 'L', 'M'}:0.1}
Expert 3
m_3 = { {'S'}: 0.55, {'S'}: 0.35, Ω: 0.1}
m_3 = { {'S'}:0.55; {'H'}:0.35; {'S', 'H', 'C', 'L', 'M'}:0.1}

```

FIGURE 1.13 – distributions de masses de l'exemple F1

Le calcul des degrés de croyance et de possibilité.

```

1  print("belief and  plausibilit of the first expert")
2  print("bel_1 = ", m1.bel())
3  print("pl_1 = ", m1.pl())
4
5  print("belief and  plausibilit of the second expert")
6  print("bel_2 = ", m2.bel())
7  print("pl_2 = ", m2.pl())
8
9  print("belief and  plausibilit of the third expert")
10 print("bel_3 = ", m3.bel())
11 print("pl_3 = ", m3.pl())

```

FIGURE 1.14 – degrés de croyances et de possibilités

Le résultat de calcul des degrés de croyance et de possibilité.

```

belief and plausibilit of the first expert
bel_1 = {frozenset(): 0.0, frozenset({'S'}): 0.0, frozenset({'H'}): 0.0, frozenset({'C'}): 0.0, frozenset({'L'}): 0.0, frozenset(
pl_1 = {frozenset(): 0.0, frozenset({'S'}): 0.3, frozenset({'H'}): 0.3, frozenset({'C'}): 0.3, frozenset({'L'}): 0.3, frozenset(
belief and plausibilit of the second expert
bel_2 = {frozenset(): 0.0, frozenset({'S'}): 0.0, frozenset({'H'}): 0.0, frozenset({'C'}): 0.0, frozenset({'L'}): 0.0, frozenset(
pl_2 = {frozenset(): 0.0, frozenset({'S'}): 0.1, frozenset({'H'}): 0.1, frozenset({'C'}): 0.5, frozenset({'L'}): 0.5, frozenset(
belief and plausibilit of the third expert
bel_3 = {frozenset(): 0.0, frozenset({'S'}): 0.55, frozenset({'H'}): 0.35, frozenset({'C'}): 0.0, frozenset({'L'}): 0.0, frozene
pl_3 = {frozenset(): 0.0, frozenset({'S'}): 0.65, frozenset({'H'}): 0.44999999999999996, frozenset({'C'}): 0.1, frozenset({'L'}):

```

FIGURE 1.15 – résultats

La Combinaison de Dempster-Shafer.

```

1 print("Dempster-Shafer combination rule for m_1, m_2 and m_3 = \n")
2 print(m1.combine_conjunctive(m2, m3))
3 print("Dempster-Shafer combination rule for m_2, m_1 and m_3 = \n")
4 print(m2.combine_conjunctive(m1, m3))
5 print("Dempster-Shafer combination rule for m_3, m_1 and m_2 = \n")
6 print(m3.combine_conjunctive(m1, m2))

```

FIGURE 1.16 – combinaison des masses

Le résultat de la combinaison.

```

Dempster-Shafer combination rule for m_1, m_2 and m_3 =
{{'M'}:0.7916666666666666; {'C', 'L'}:0.16666666666666666; {'S', 'H', 'C', 'L', 'M'}:0.04166666666666664}
Dempster-Shafer combination rule for m_2, m_1 and m_3 =
{{'M'}:0.7916666666666666; {'C', 'L'}:0.16666666666666666; {'S', 'H', 'C', 'L', 'M'}:0.04166666666666664}
Dempster-Shafer combination rule for m_3, m_1 and m_2 =
{{'S'}:0.445945945945946; {'H'}:0.28378378378378377; {'M'}:0.18918918918918917; {'S', 'H', 'C', 'L', 'M'}:0.08108108108108109}

```

FIGURE 1.17 – résultat de la combinaison

En conclusion , l'hypothèse Max Verstappen gagnera est la plus soutenue.

## 1.5 Conclusion

En conclusion, ce premier travail pratique sur les fonctions de croyances a permis d'expérimenter concrètement la théorie de Dempster-Shafer pour modéliser et agréger des opinions contradictoires, en s'appuyant sur des exemples variés. Nous avons observé comment la combinaison des masses peut renforcer certaines hypothèses, en tenant compte de la fiabilité des sources et des niveaux d'incertitude. Ces exercices montrent l'utilité de la théorie des croyances pour une prise de décision éclairée dans des contextes complexes, où des sources d'informations multiples, parfois contradictoires, sont présentes.

## 2.1 Introduction

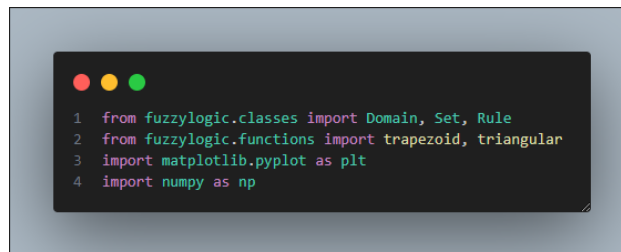
Ce travail pratique explore l'application de la logique floue à travers la bibliothèque fuzzylogic de Python pour modéliser un système complexe dans le contexte de la Formule 1. Nous avons défini deux variables d'entrée : la performance du moteur de la voiture et les réflexes du conducteur, et une variable de sortie : le nombre d'obstacles évités. Ce TP met en lumière l'utilité de la logique floue pour intégrer des concepts linguistiques tels que "élevé" ou "excellent" dans des systèmes de prise de décision, tout en illustrant les étapes de fuzzification, d'inférence et de défuzzification.

## 2.2 Étape 1

### 2.2.1 Implementation d'un exemple du TD

Dans cette étape, nous allons tester l'exemple du TD portant sur l'ajustement d'une vanne dans une usine de fonderie. Il s'agit de régler un paramètre  $u$  servant au débit d'une vanne entre un réceptacle contenant du métal en fusion, et un deuxième bassin dont le niveau est mesuré par la hauteur  $h$ . Ce dernier se déversant dans un moule.

Nous commençons par l'importation de la bibliothèque fuzzylogic.

A screenshot of a code editor window with a dark background and light-colored text. The code is as follows:

```
1 from fuzzylogic.classes import Domain, Set, Rule
2 from fuzzylogic.functions import trapezoid, triangular
3 import matplotlib.pyplot as plt
4 import numpy as np
```

FIGURE 2.1 – Importation bibliothèque



Nous définissons ensuite les domaines de nos variables.

```
1
2 H = Domain('H', 75, 85)
3
4
5 DH = Domain('DH', -9, 9)
6
7
8 U = Domain('U', -10, 10)
```

FIGURE 2.2 – Définition des domaines

Nous passons à la fuzzification pour les variables d'entrée h et dh et la variable de sortie u.

```
1 H.P = trapezoid(75,77,78,79)
2 H.M = trapezoid(78,79,81,83)
3 H.H = custom_membership
4 H.P.plot()
5 H.M.plot()
6 H.H.plot()
```

FIGURE 2.3 – Fuzzification de h

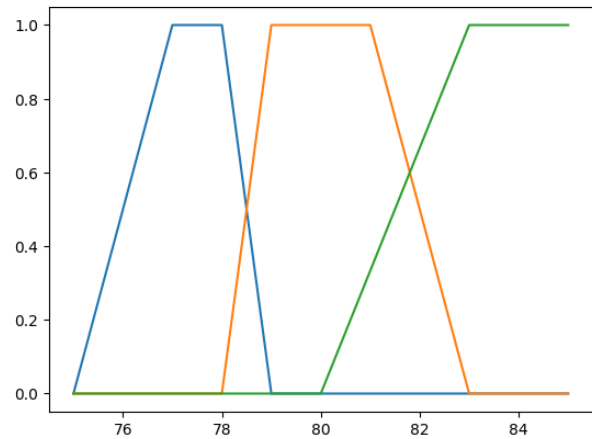
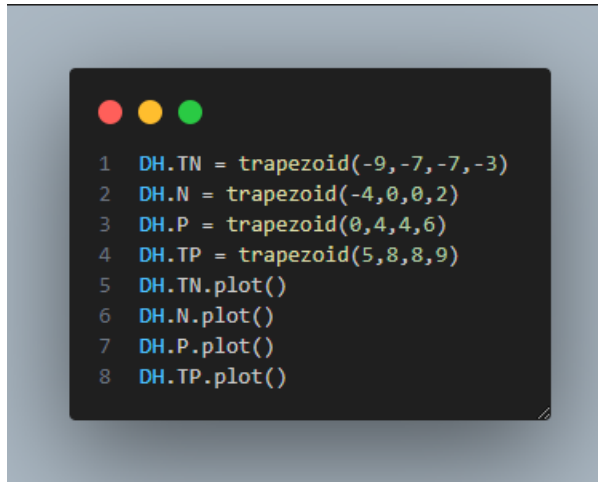


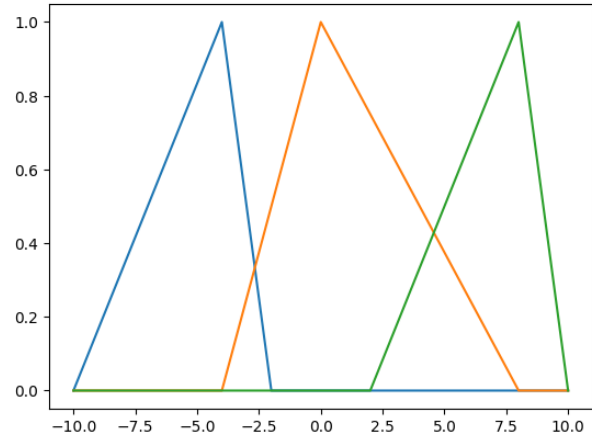
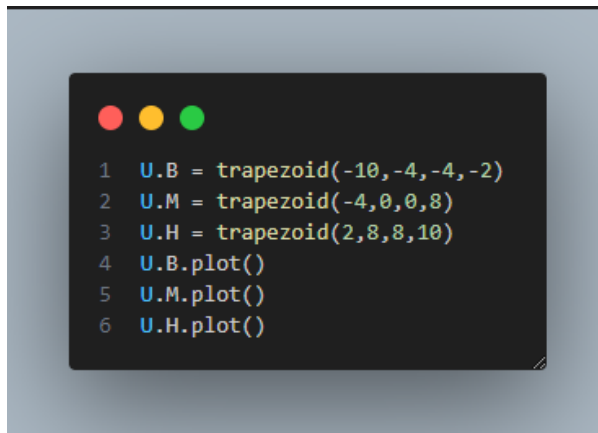
FIGURE 2.4 – Résultat de Fuzzification de h



```

1 DH.TN = trapezoid(-9,-7,-7,-3)
2 DH.N = trapezoid(-4,0,0,2)
3 DH.P = trapezoid(0,4,4,6)
4 DH.TP = trapezoid(5,8,8,9)
5 DH.TN.plot()
6 DH.N.plot()
7 DH.P.plot()
8 DH.TP.plot()

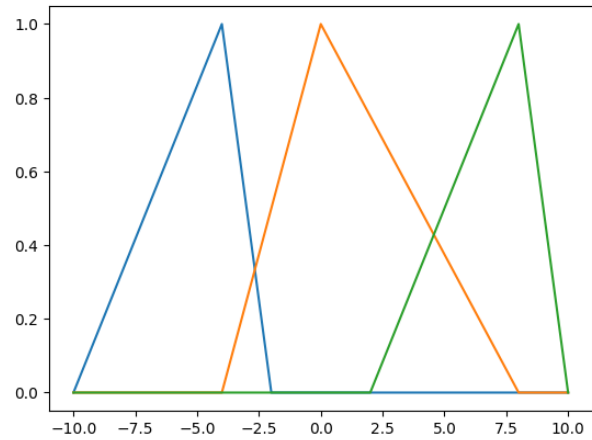
```

FIGURE 2.5 – Fuzzification de  $dh$ FIGURE 2.6 – Résultat de Fuzzification de  $dh$ 


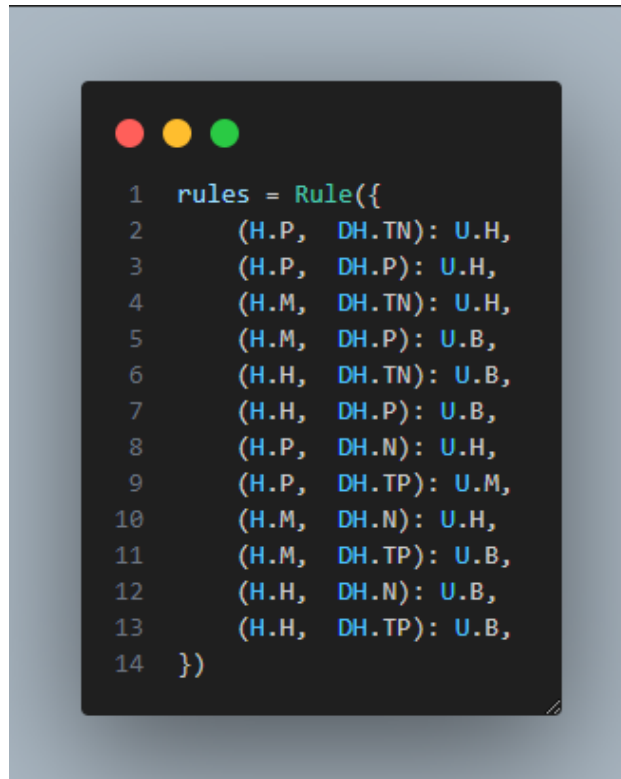
```

1 U.B = trapezoid(-10,-4,-4,-2)
2 U.M = trapezoid(-4,0,0,8)
3 U.H = trapezoid(2,8,8,10)
4 U.B.plot()
5 U.M.plot()
6 U.H.plot()

```

FIGURE 2.7 – Fuzzification de  $u$ FIGURE 2.8 – Résultat de Fuzzification de  $u$

Continuons avec la construction de la base de règles.

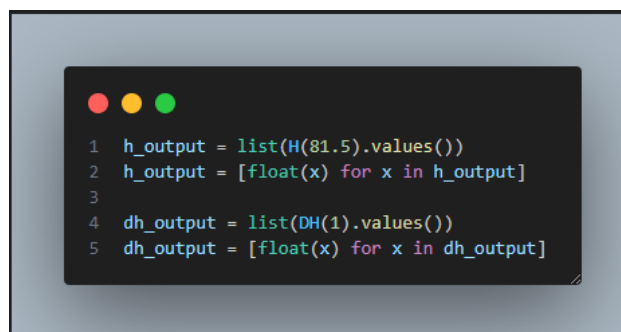


```
1  rules = Rule({
2      (H.P,  DH.TN): U.H,
3      (H.P,  DH.P): U.H,
4      (H.M,  DH.TN): U.H,
5      (H.M,  DH.P): U.B,
6      (H.H,  DH.TN): U.B,
7      (H.H,  DH.P): U.B,
8      (H.P,  DH.N): U.H,
9      (H.P,  DH.TP): U.M,
10     (H.M,  DH.N): U.H,
11     (H.M,  DH.TP): U.B,
12     (H.H,  DH.N): U.B,
13     (H.H,  DH.TP): U.B,
14 })
```

FIGURE 2.9 – Base de règles

Application de la méthode d'inférence (Mandani) :

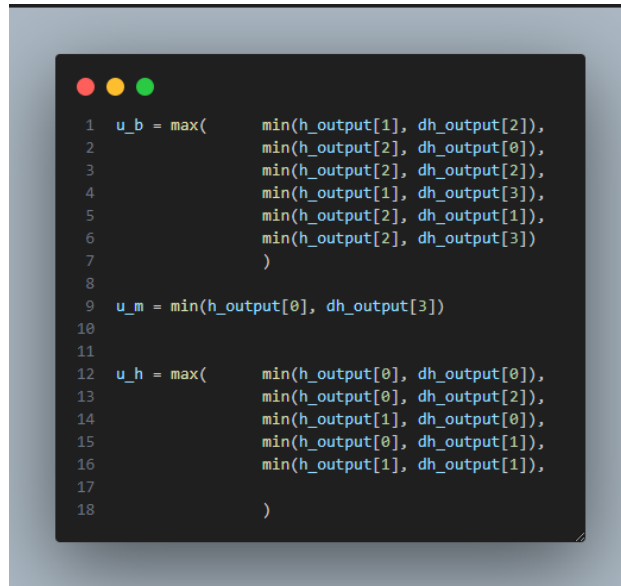
Fuzzification des valeurs de d et h avant l'inférence



```
1  h_output = list(H(81.5).values())
2  h_output = [float(x) for x in h_output]
3
4  dh_output = list(DH(1).values())
5  dh_output = [float(x) for x in dh_output]
```

FIGURE 2.10 – Fuzzification

Inférer ne utilisant le min entre les varibales de la meme règle, ensuite l'application du max aux résultats qui portent sur la meme valeur de la variable de sortie.



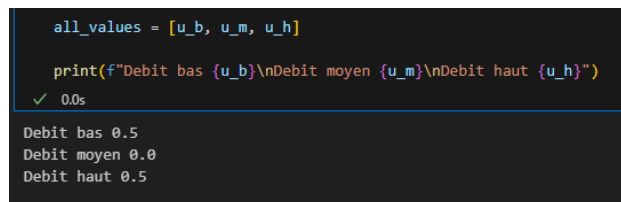
```

1  u_b = max(      min(h_output[1], dh_output[2]),
2                  min(h_output[2], dh_output[0]),
3                  min(h_output[2], dh_output[2]),
4                  min(h_output[1], dh_output[3]),
5                  min(h_output[2], dh_output[1]),
6                  min(h_output[2], dh_output[3])
7                  )
8
9  u_m = min(h_output[0], dh_output[3])
10
11
12  u_h = max(      min(h_output[0], dh_output[0]),
13                  min(h_output[0], dh_output[2]),
14                  min(h_output[1], dh_output[0]),
15                  min(h_output[0], dh_output[1]),
16                  min(h_output[1], dh_output[1]),
17
18                  )

```

FIGURE 2.11 – l'inférence

Résultat de l'inférence :



```

all_values = [u_b, u_m, u_h]

print(f"Debit bas {u_b}\nDebit moyen {u_m}\nDebit haut {u_h}")
✓ 0.0s
Debit bas 0.5
Debit moyen 0.0
Debit haut 0.5

```

FIGURE 2.12 – Résultat d'inférence de l'exemple 1

Et comme dernière étape nous devons faire la défuzzification en calculant le centre de gravité pour avoir au final une valeur non floue de la variable de sortie  
code :

```

1 fig, axis = plt.subplots(figsize=(7, 5))
2
3 x_U = U.range
4
5 U_0 = np.zeros_like(x_U)
6 parametres_U = [U.B, U.M, U.H]
7 j=0
8 colors = ['b', 'g', 'm']
9 for each in parametres_U:
10     axis.plot(x_U, each.array(), colors[j], linewidth=1)
11     axis.fill_between(x_U, U_0, [min(all_values[j], x) for x in each.array()],
12                      facecolor=colors[j], alpha=0.7)
13     j+=1

```

FIGURE 2.13 – défuzzification de l'exemple 1

Résultat :

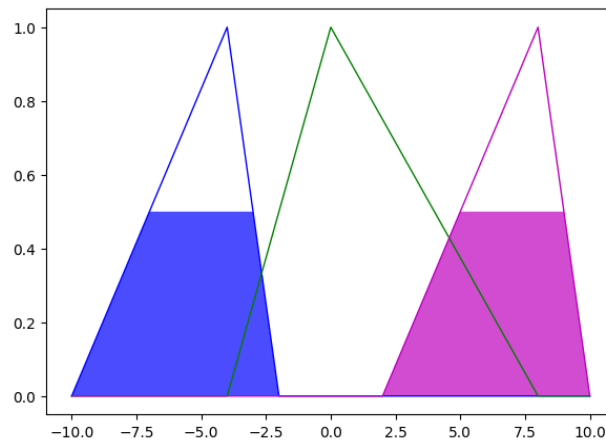


FIGURE 2.14 – Résultat de défuzzification de l'exemple 1

Calcule du centre de gravité

```

values = {H: 81.5, DH: 1}
gravity_center = rules(values)
print(f"gravity center for values h= 81.5 and dh = 1 : {gravity_center}")

```

✓ 0.1s Python

gravity center for values h= 81.5 and dh = 1 : -1.7460317460317487

FIGURE 2.15 – calcul du centre de gravité de l'exemple 1

## 2.3 Étape 2

### 2.3.1 Modélisation d'un exemple

Dans cette section, nous présentons une modélisation basée sur la logique floue pour simuler la performance d'une voiture de Formule 1 dans le cadre d'une course. L'objectif est d'analyser comment la performance du moteur de la voiture et les réflexes du pilote influencent le nombre d'obstacles évités avec succès. Cette modélisation utilise deux variables d'entrée et une variable de sortie, définies comme suit :

**Variable d'entrée 1 : Performance du moteur de la voiture (H)** Cette variable représente la puissance et l'efficacité du moteur de la voiture, avec les termes linguistiques suivants :

- **H** : Performance élevée
- **VH** : Performance très élevée
- **EH** : Performance extrêmement élevée

**Variable d'entrée 2 : Réflexes du pilote (R)** Cette variable représente le temps de réaction et les réflexes du pilote, avec les termes linguistiques suivants :

- **G** : Bons réflexes
- **E** : Réflexes excellents

**Variable de sortie : Obstacles évités (O)** Cette variable exprime le nombre d'obstacles que le pilote parvient à éviter avec succès au cours d'une course. Les plages pour les fonctions d'appartenance sont les suivantes :

- **Bas (L)** : (0, 2, 4)
- **Modéré (M)** : (3, 6, 8)
- **Haut (H)** : (7, 9, 12)

Définissons des domaines de variables.

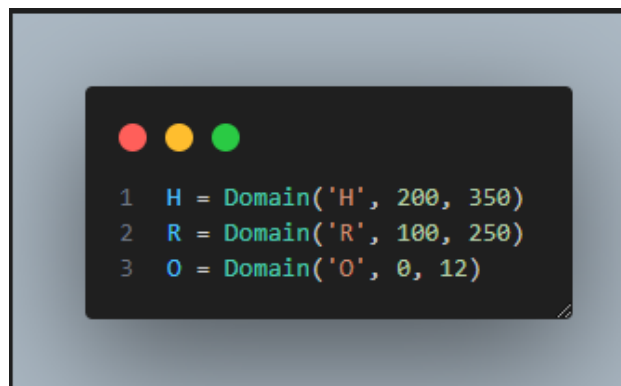


FIGURE 2.16 – Définition des domaines

Fuzzification des variables d'entrée H et R et la variable de sortie O.

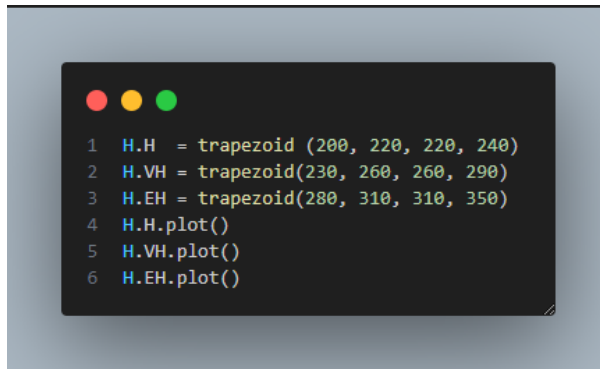


FIGURE 2.17 – Fuzzification de H

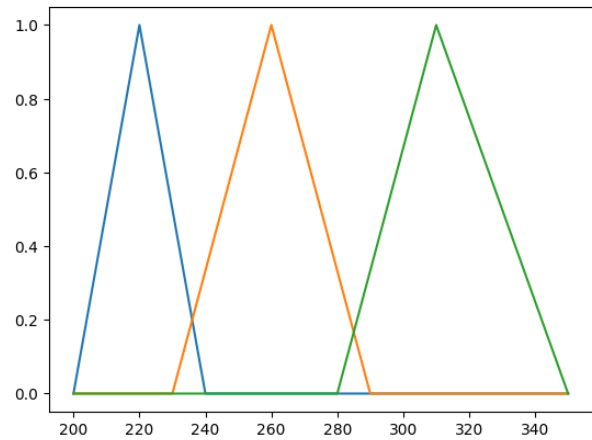


FIGURE 2.18 – Résultat de Fuzzification de H

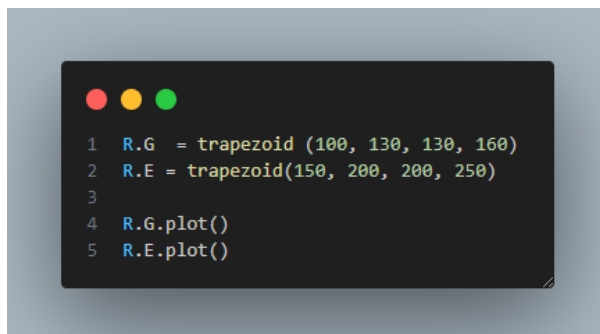


FIGURE 2.19 – Fuzzification de R

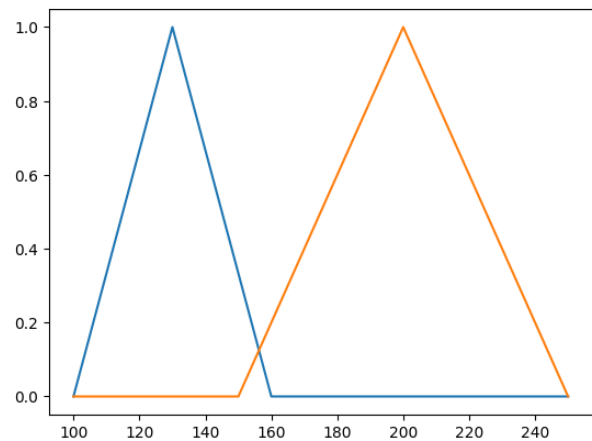
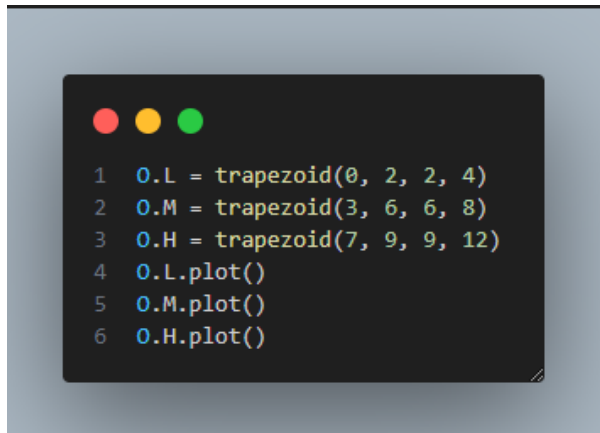
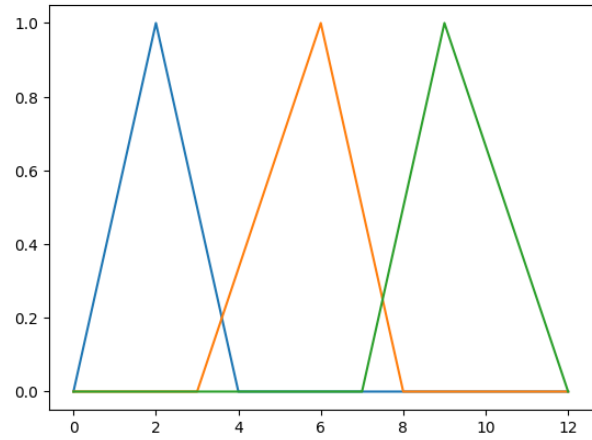


FIGURE 2.20 – Résultat de Fuzzification de R

FIGURE 2.21 – Fuzzification de  $O$ FIGURE 2.22 – Résultat de Fuzzification de  $O$ 

Construction de la base de règles.

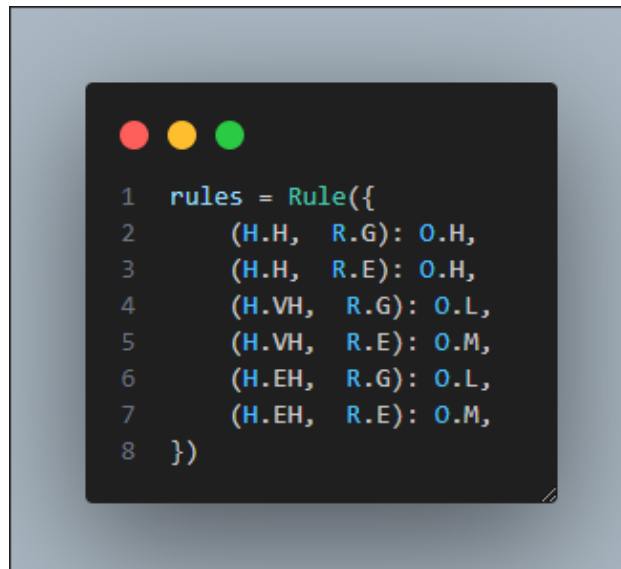
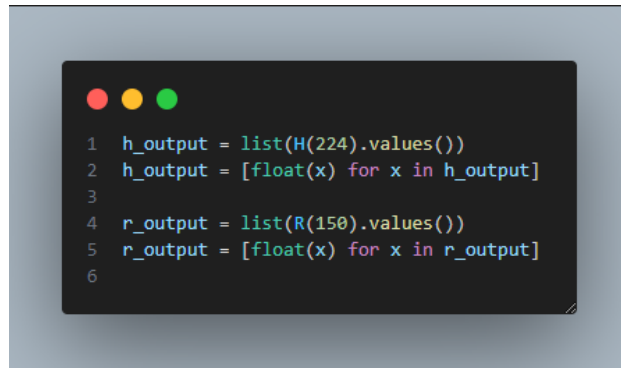


FIGURE 2.23 – Base de règles 2



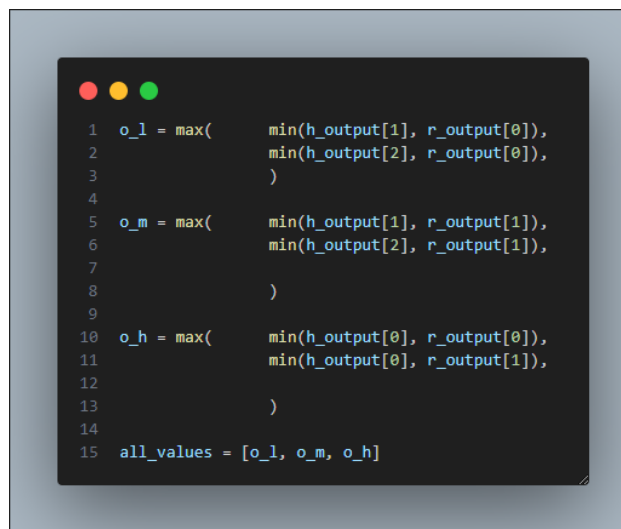
Application de la méthode d'inférence (Mandani) :  
Fuzzification des valeurs de H et R avant l'inférence

A screenshot of a code editor with a dark background and light-colored text. The code is in Python and consists of six lines. Line 1: `h_output = list(H(224).values())`. Line 2: `h_output = [float(x) for x in h_output]`. Line 3: (empty line). Line 4: `r_output = list(R(150).values())`. Line 5: `r_output = [float(x) for x in r_output]`. Line 6: (empty line). The code is enclosed in a light gray border with three colored circles (red, yellow, green) in the top left corner.

```
1 h_output = list(H(224).values())
2 h_output = [float(x) for x in h_output]
3
4 r_output = list(R(150).values())
5 r_output = [float(x) for x in r_output]
6
```

FIGURE 2.24 – Fuzzification 2

Inférer en utilisant le min entre les variables de la même règle, ensuite l'application du max aux résultats qui portent sur la même valeur de la variable de sortie.

A screenshot of a code editor with a dark background and light-colored text. The code is in Python and consists of 15 lines. Line 1: `o_l = max( min(h_output[1], r_output[0]),`. Line 2: `min(h_output[2], r_output[0]),`. Line 3: `)`. Line 4: (empty line). Line 5: `o_m = max( min(h_output[1], r_output[1]),`. Line 6: `min(h_output[2], r_output[1]),`. Line 7: `)`. Line 8: (empty line). Line 9: (empty line). Line 10: `o_h = max( min(h_output[0], r_output[0]),`. Line 11: `min(h_output[0], r_output[1]),`. Line 12: `)`. Line 13: (empty line). Line 14: (empty line). Line 15: `all_values = [o_l, o_m, o_h]`. The code is enclosed in a light gray border with three colored circles (red, yellow, green) in the top left corner.

```
1 o_l = max( min(h_output[1], r_output[0]),
2 min(h_output[2], r_output[0]),
3 )
4
5 o_m = max( min(h_output[1], r_output[1]),
6 min(h_output[2], r_output[1]),
7 )
8
9
10 o_h = max( min(h_output[0], r_output[0]),
11 min(h_output[0], r_output[1]),
12 )
13
14
15 all_values = [o_l, o_m, o_h]
```

FIGURE 2.25 – l'inférence 2

Résultat de l'inférence :

```
Low skepped obstacles 0.5
Moderate skipped obstacles 0.0
Height skipped obstacles 0.5
```

FIGURE 2.26 – Résultat d'inférence de l'exemple 2

Défuzzification et calcul du centre de gravité pour avoir au final une valeur non floue de la variable de sortie O

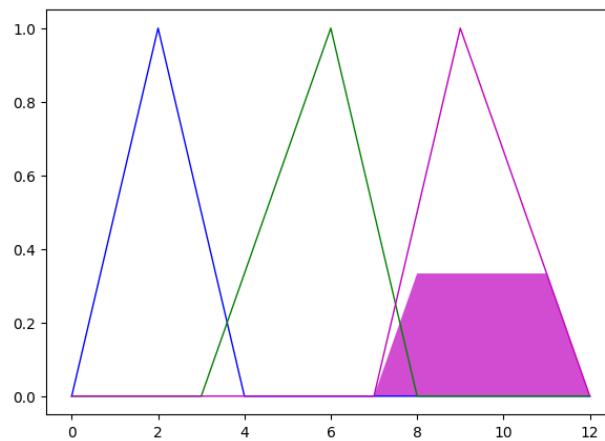


FIGURE 2.27 – Résultat de défuzzification de l'exemple 2

Calcul du centre de gravité

```
values = {H: 224, R: 150}
gravity_center = rules(values)
print(f"gravity center for values h= 224 and dh = 150 : {gravity_center}")
[ ] ✓ 0.0s
gravity center for values h= 224 and dh = 150 : 8.615384615384615
```

FIGURE 2.28 – calcul du centre de gravité de l'exemple 2

## 2.4 Conclusion

Ce TP a permis de concevoir un système flou capable de modéliser des scénarios réalistes et incertains, en utilisant des règles d'inférence pour relier des entrées linguistiques à des résultats mesurables. Grâce à la logique floue, nous avons simulé comment les performances d'un moteur et les réflexes d'un pilote influencent le nombre d'obstacles évités. Ce travail illustre l'efficacité des systèmes flous pour résoudre des problèmes complexes dans des environnements variés et ouvre des perspectives d'application dans d'autres domaines technologiques.

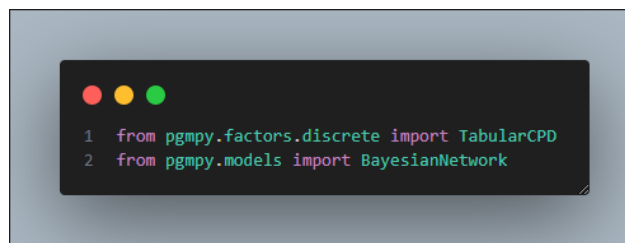
## 3.1 Introduction

Les réseaux bayésiens sont des outils puissants pour modéliser des relations probabilistes entre variables dans des systèmes incertains. Ce TP vise à explorer leur construction et leur analyse à travers des exemples pratiques, en utilisant la bibliothèque pgmpy. L'objectif est de comprendre comment modéliser des situations complexes, effectuer des inférences et appliquer ces concepts à des problèmes réels.

## 3.2 Étape 1

### 3.2.1 Installation d'une toolbox des Réseaux Bayésiens

Après avoir installer la bibliothèque pgmpy en executant la commande `pip install pgmpy` nous avons imporeter cette dernière comme suit :

A terminal window with a dark background and light-colored text. It shows two lines of Python code being executed. The first line imports TabularCPD from pgmpy.factors.discrete, and the second line imports BayesianNetwork from pgmpy.models. The terminal has three colored dots (red, yellow, green) in the top left corner, indicating it is a standard macOS-style window.

```
1 from pgmpy.factors.discrete import TabularCPD
2 from pgmpy.models import BayesianNetwork
```

FIGURE 3.1 – Imporation bibiliothèque

### 3.3 Étape 2

Dans cette section , nous allons essayer de générer le poly-arbre suivant :

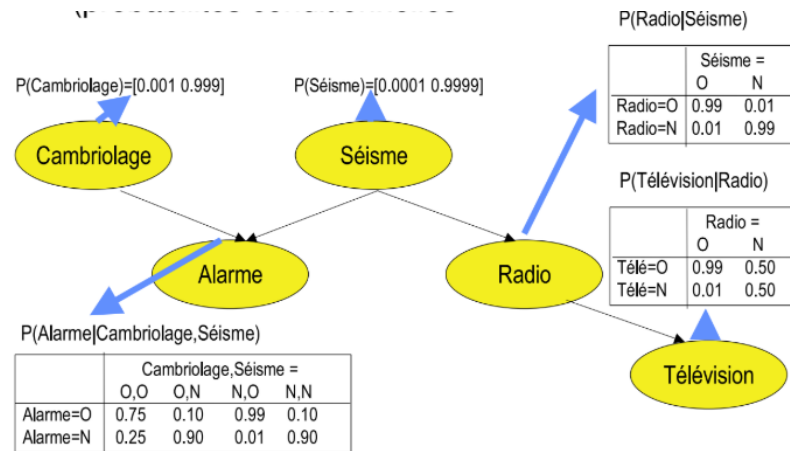


FIGURE 3.2 – Exemple poly-arbre

Commençons tout d'abord par la définition de la structure du poly-arbre

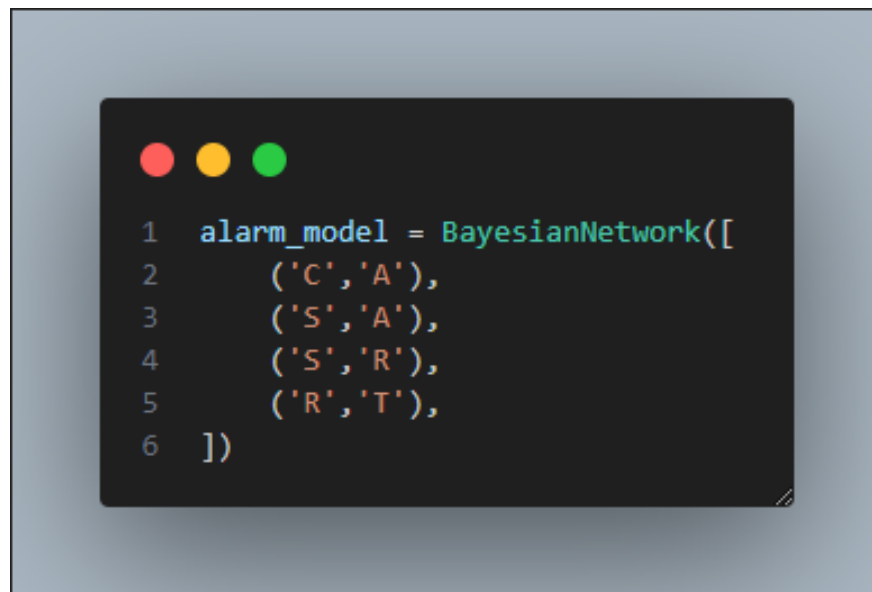
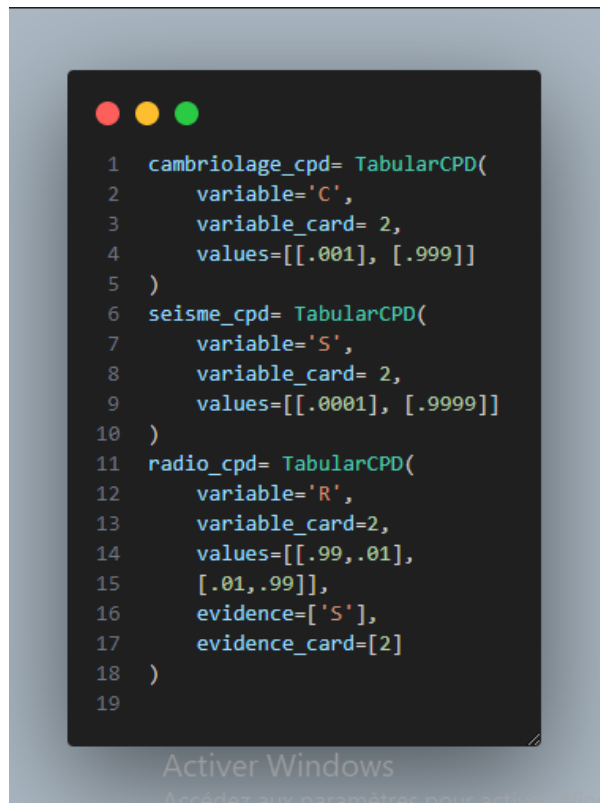


FIGURE 3.3 – Définition structure

Passons à la définition des relations :

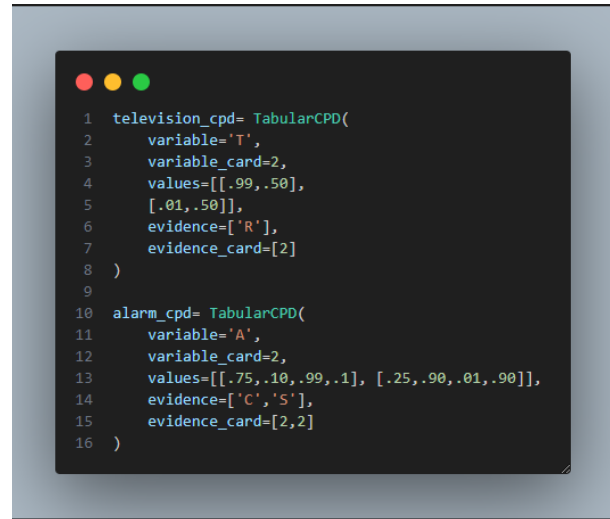


```

1  cambriolage_cpd= TabularCPD(
2      variable='C',
3      variable_card= 2,
4      values=[[.001], [.999]]
5  )
6  seisme_cpd= TabularCPD(
7      variable='S',
8      variable_card= 2,
9      values=[[.0001], [.9999]]
10 )
11 radio_cpd= TabularCPD(
12     variable='R',
13     variable_card=2,
14     values=[[.99,.01],
15            [.01,.99]],
16     evidence=['S'],
17     evidence_card=[2]
18 )
19

```

FIGURE 3.4 – Connections entre les noeuds

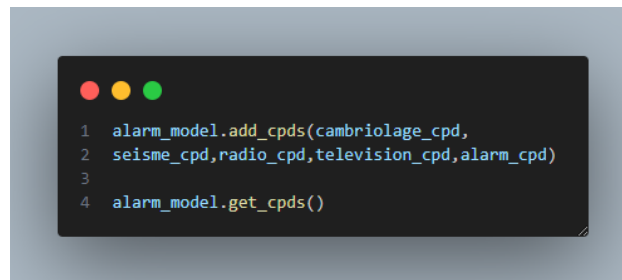


```

1  television_cpd= TabularCPD(
2      variable='T',
3      variable_card=2,
4      values=[[.99,.50],
5             [.01,.50]],
6      evidence=['R'],
7      evidence_card=[2]
8  )
9
10 alarm_cpd= TabularCPD(
11     variable='A',
12     variable_card=2,
13     values=[[.75,.10,.99,.1], [.25,.90,.01,.90]],
14     evidence=['C','S'],
15     evidence_card=[2,2]
16 )

```

FIGURE 3.5 – Connections entre les noeuds



```

1  alarm_model.add_cpds(cambriolage_cpd,
2      seisme_cpd,radio_cpd,television_cpd,alarm_cpd)
3
4  alarm_model.get_cpds()

```

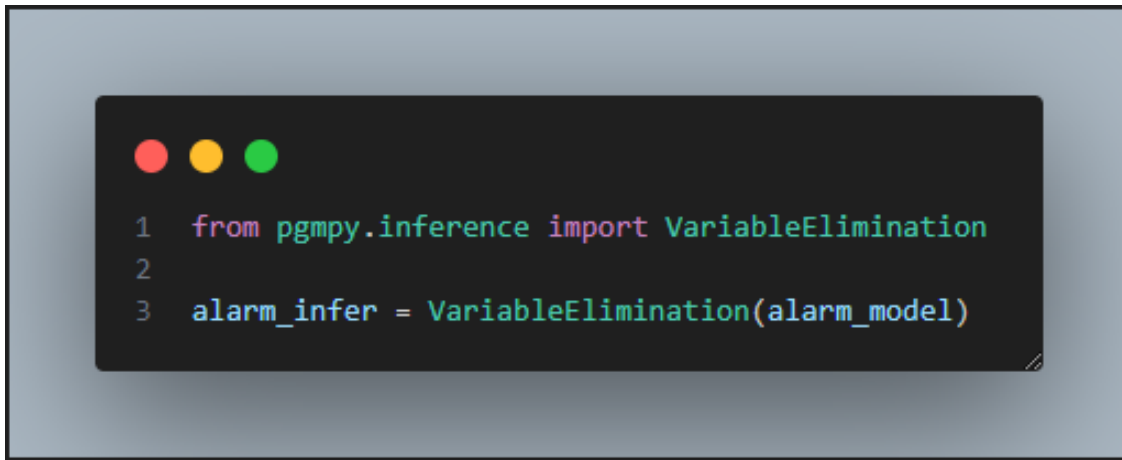
FIGURE 3.6 – Affectation des connections au poly arbre

Résultat de la définition de la structure et affectation des connection :

```
[<TabularCPD representing P(C:2) at 0x1ffb440fca0>,  
<TabularCPD representing P(S:2) at 0x1ffb440e440>,  
<TabularCPD representing P(R:2 | S:2) at 0x1ffb440e6b0>,  
<TabularCPD representing P(T:2 | R:2) at 0x1ffb440ea70>,  
<TabularCPD representing P(A:2 | C:2, S:2) at 0x1ffb440f160>]
```

FIGURE 3.7 – Résultat

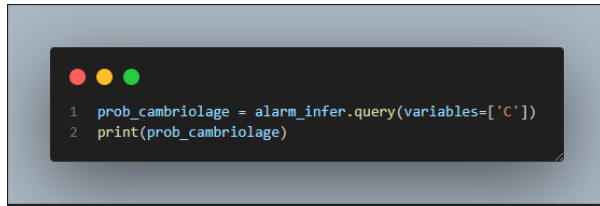
Calculons maintenant les probabilités de ce réseau :

A screenshot of a code editor window with a dark background and light-colored text. The editor has three colored window control buttons (red, yellow, green) in the top-left corner. The code is as follows:

```
1 from pgmpy.inference import VariableElimination  
2  
3 alarm_infer = VariableElimination(alarm_model)
```

FIGURE 3.8 – Calcul probabilités

Afichage de la probabilité de cambriolage :



```

1 prob_cambriolage = alarm_infer.query(variables=['C'])
2 print(prob_cambriolage)

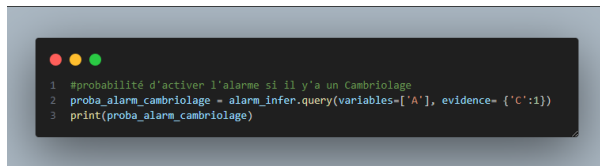
```

FIGURE 3.9 – Calcul de la probabilité

C	phi(C)
C(0)	0.0010
C(1)	0.9990

FIGURE 3.10 – Résultat

Probabilité d'activer l'alarme si il y'a un Cambriolage



```

1 #probabilité d'activer l'alarme si il y'a un Cambriolage
2 proba_alarm_cambriolage = alarm_infer.query(variables=['A'], evidence= {'C':1})
3 print(proba_alarm_cambriolage)

```

FIGURE 3.11 – Calcul de la probabilité

A	phi(A)
A(0)	0.1001
A(1)	0.8999

FIGURE 3.12 – Résultat

### 3.4 Étape 3

Dans cette section , nous allons générer le graphe à connexions multiples suivant :

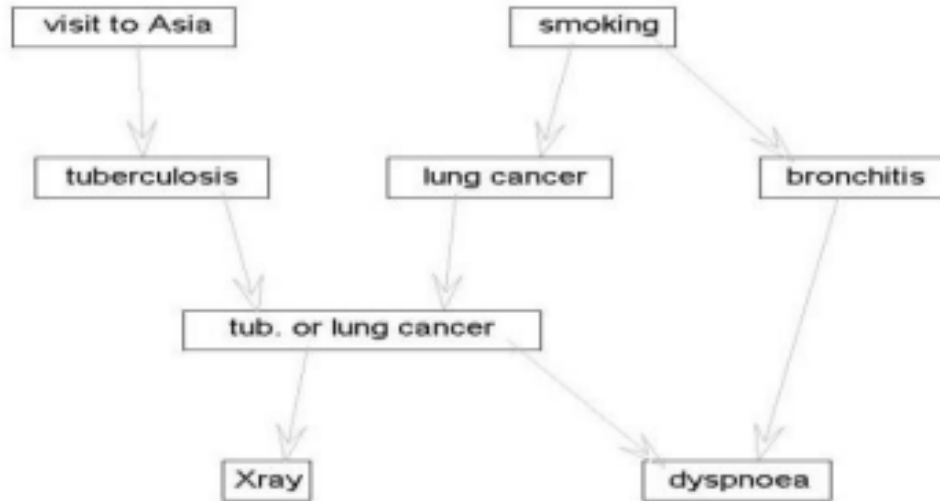


FIGURE 3.13 – Graphe à connexions multiples



Le graphe a pour distribution conditionnelle les probabilités suivantes :

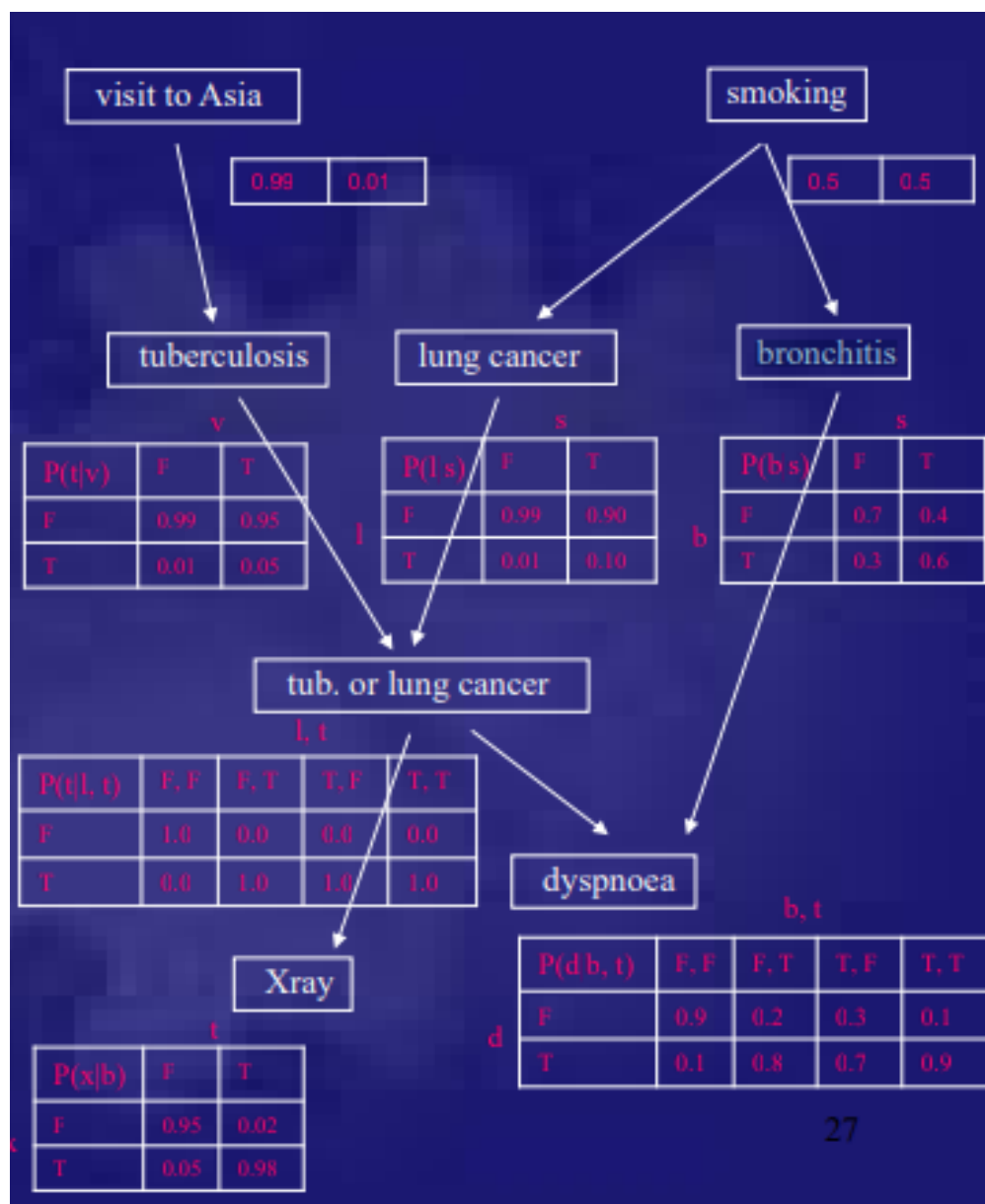



FIGURE 3.14 – distribution conditionnelle

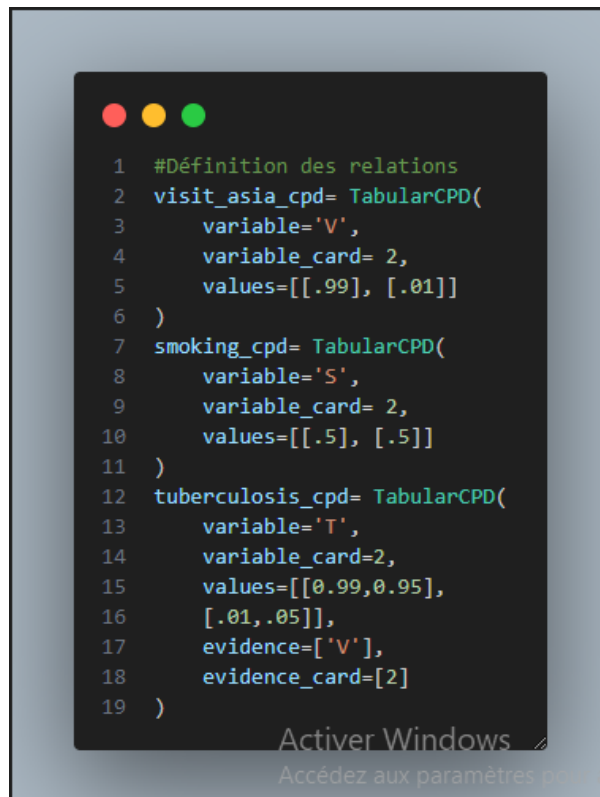
Définition de la structure du graph :



```
1  medical_model = BayesianNetwork([
2      ('V', 'T'),
3      ('S', 'L'),
4      ('S', 'B'),
5      ('T', 'TL'),
6      ('L', 'TL'),
7      ('TL', 'X'),
8      ('B', 'D'),
9      ('TL', 'D'),
10 ])
```

FIGURE 3.15 – Définition structure

Définition des relations :



```

1  #Définition des relations
2  visit_asia_cpd= TabularCPD(
3      variable='V',
4      variable_card= 2,
5      values=[[.99], [.01]]
6  )
7  smoking_cpd= TabularCPD(
8      variable='S',
9      variable_card= 2,
10     values=[[.5], [.5]]
11 )
12 tuberculosis_cpd= TabularCPD(
13     variable='T',
14     variable_card=2,
15     values=[[0.99,0.95],
16             [.01,.05]],
17     evidence=['V'],
18     evidence_card=[2]
19 )

```

FIGURE 3.16 – Connections entre les noeuds

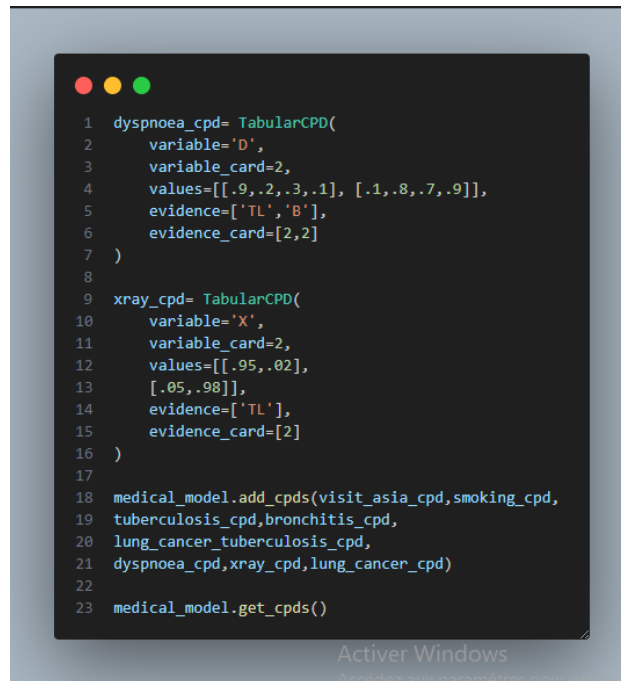


```

1  lung_cancer_cpd= TabularCPD(
2      variable='L',
3      variable_card=2,
4      values=[[.99,.90],
5              [.01,.10]],
6      evidence=['S'],
7      evidence_card=[2]
8  )
9  bronchitis_cpd= TabularCPD(
10     variable='B',
11     variable_card=2,
12     values=[[.7,.4],
13             [.3,.6]],
14     evidence=['S'],
15     evidence_card=[2]
16 )
17 lung_cancer_tuberculosis_cpd= TabularCPD(
18     variable='TL',
19     variable_card=2,
20     values=[[1.0,.0,.0,.0],
21             [0,1.0,1.0,1.0]],
22     evidence=['L','T'],
23     evidence_card=[2,2]
24 )

```

FIGURE 3.17 – Connections entre les noeuds



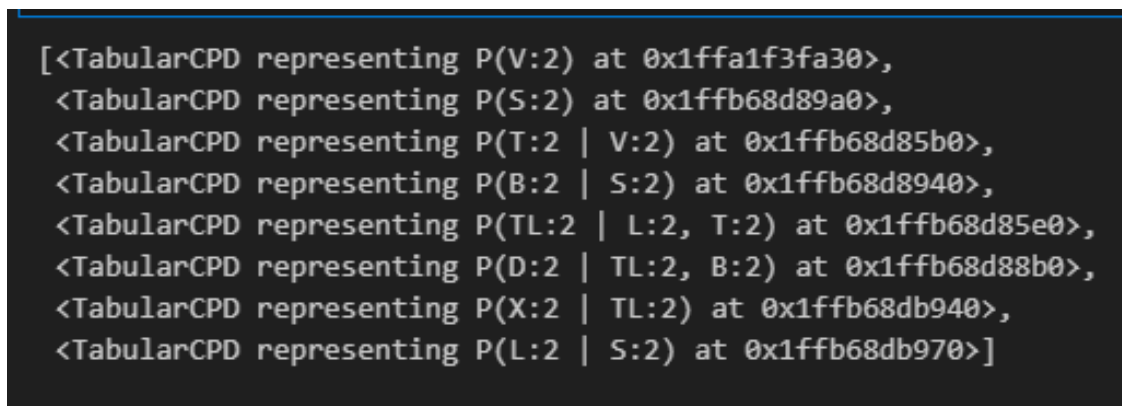
```

1  dyspnoea_cpd= TabularCPD(
2      variable='D',
3      variable_card=2,
4      values=[[.9,.2,.3,.1], [.1,.8,.7,.9]],
5      evidence=['TL','B'],
6      evidence_card=[2,2]
7  )
8
9  xray_cpd= TabularCPD(
10     variable='X',
11     variable_card=2,
12     values=[[.95,.02],
13             [.05,.98]],
14     evidence=['TL'],
15     evidence_card=[2]
16 )
17
18 medical_model.add_cpds(visit_asia_cpd,smoking_cpd,
19 tuberculosis_cpd,bronchitis_cpd,
20 lung_cancer_tuberculosis_cpd,
21 dyspnoea_cpd,xray_cpd,lung_cancer_cpd)
22
23 medical_model.get_cpds()

```

FIGURE 3.18 – Affectation des connections au graph

Résultat de la définition de la structure et affectation des connections :



```

[<TabularCPD representing P(V:2) at 0x1ffa1f3fa30>,
 <TabularCPD representing P(S:2) at 0x1ffb68d89a0>,
 <TabularCPD representing P(T:2 | V:2) at 0x1ffb68d85b0>,
 <TabularCPD representing P(B:2 | S:2) at 0x1ffb68d8940>,
 <TabularCPD representing P(TL:2 | L:2, T:2) at 0x1ffb68d85e0>,
 <TabularCPD representing P(D:2 | TL:2, B:2) at 0x1ffb68d88b0>,
 <TabularCPD representing P(X:2 | TL:2) at 0x1ffb68db940>,
 <TabularCPD representing P(L:2 | S:2) at 0x1ffb68db970>]

```

FIGURE 3.19 – Résultat

Calculons maintenant les probabilités de ce réseau :

```
1 medical_infer = VariableElimination(medical_model)
2 proba_visit_asia= medical_infer.query(variables=['V'])
3 print(proba_visit_asia)
4 print('\n')
5 proba_tub_visit = medical_infer.query(variables=['T'], evidence= {'V':1})
6 print(proba_tub_visit)
7 print('\n')
8 proba_dyspnoea_tub_lung_bronchitis = medical_infer.query(
9     variables=['D'], evidence= {'B':1, 'TL':0})
10 print(proba_dyspnoea_tub_lung_bronchitis)
```

FIGURE 3.20 – Calcul probabilités

Afichage des probabilités :

+-----+-----+	
V	phi(V)
+=====+	
V(0)	0.9900
+-----+-----+	
V(1)	0.0100
+-----+-----+	
+-----+-----+	
T	phi(T)
+=====+	
T(0)	0.9500
+-----+-----+	
T(1)	0.0500
+-----+-----+	
+-----+-----+	
D	phi(D)
+=====+	
D(0)	0.2000
+-----+-----+	
D(1)	0.8000
+-----+-----+	

FIGURE 3.21 – Résultats du calcul des probabilité

### 3.5 Étape 4

#### Modélisation d'un Réseau Bayésien Simplifié pour une Course F1

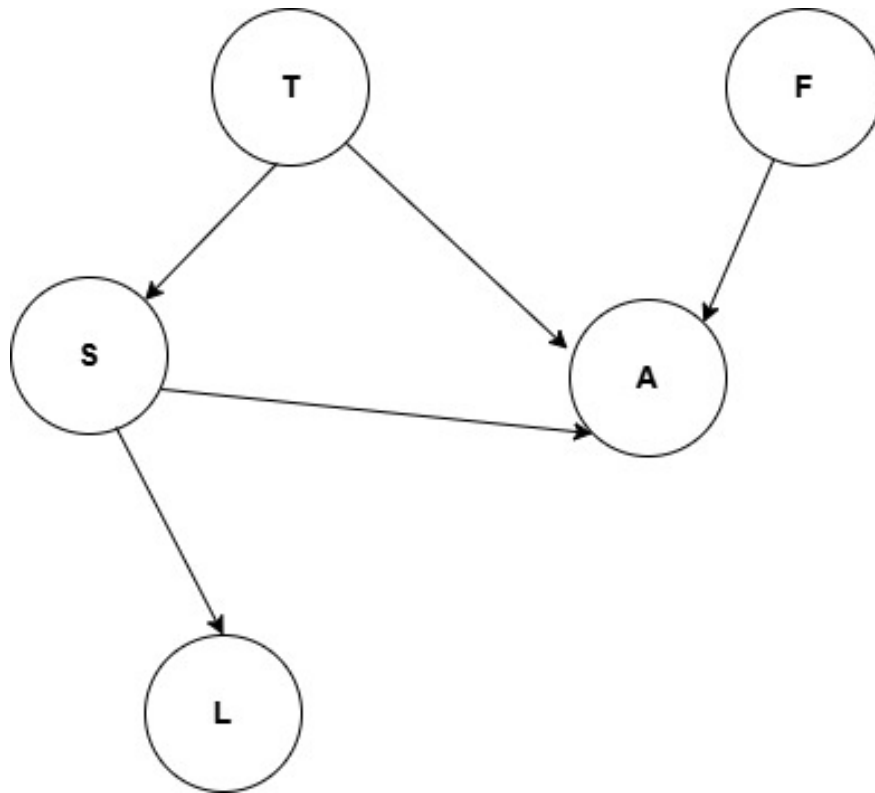


FIGURE 3.22 – Définition structure

Ce modèle de réseau bayésien pour une course F1 représente les relations entre plusieurs facteurs clés suivants : l'état de la piste (**Track Condition**), la fatigue du pilote (**Fatigue**), la vitesse (**Speed**), les accidents (**Accident**) et le temps par tour (**Lap Time**).

#### Relations Modélisées

- **L'état de la piste (T) :**
  - Influence directement la **vitesse (S)** du pilote : une piste en mauvais état réduit la vitesse.
  - Influence également le risque d'**accident (A)** : une piste glissante ou endommagée augmente le risque d'accident.
- **La fatigue du pilote (F) :**
  - A un impact direct sur le risque d'**accident (A)** : un pilote fatigué est plus susceptible de faire une erreur, augmentant ainsi le risque d'accident.
- **La vitesse (S) :**
  - Impacte directement le risque d'**accident (A)** : une vitesse excessive ou inadéquate peut conduire à des accidents.

- Affecte également le **temps par tour (L)** : une vitesse plus élevée peut réduire le temps par tour, tandis qu'une vitesse plus faible peut l'augmenter.

## Résumé du Modèle

Le modèle capture les dépendances entre ces variables et peut être utilisé pour évaluer des probabilités conditionnelles dans différents scénarios, comme l'impact d'une piste glissante ou d'un pilote fatigué sur les performances ou la sécurité.

## Représentation des Relations

Le réseau est défini comme suit :

Track Condition (T)  $\longrightarrow$  Speed (S)  $\longrightarrow$  Lap Time (L)

Track Condition (T), Fatigue (F), Speed (S)  $\longrightarrow$  Accident (A)

## Distributions Conditionnelles

### 1. État de la piste (T)

T	Probabilité
<i>Bonnepiste</i> ( $T = 0$ )	0.9
<i>Mauvaispiste</i> ( $T = 1$ )	0.1

### 2. Fatigue (F)

F	Probabilité
<i>Pasdefatigue</i> ( $F = 0$ )	0.85
<i>Fatigue</i> ( $F = 1$ )	0.15

### 3. Vitesse (S | T)

T	S = 0 (Bonne vitesse)	S = 1 (Mauvaise vitesse)
$T = 0$ ( <i>Bonnepiste</i> )	0.8	0.2
$T = 1$ ( <i>Mauvaispiste</i> )	0.4	0.6

### 4. Accident (A | T, F, S)

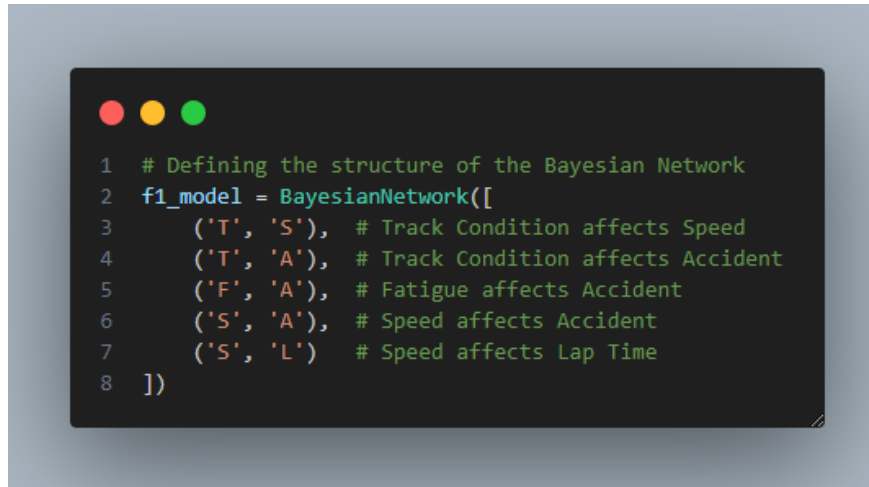
T	F	S	A = 0 (Pas d'accident)	A = 1 (Accident)
0	0	0	0.99	0.01
0	0	1	0.90	0.10
0	1	0	0.80	0.20
0	1	1	0.50	0.50
1	0	0	0.60	0.40
1	0	1	0.40	0.60
1	1	0	0.30	0.70
1	1	1	0.10	0.90



## 5. Temps par tour (L | S)

S	L = 0 (Bon temps)	L = 1 (Mauvais temps)
$S = 0$ (Bonnevitesse)	0.7	0.3
$S = 1$ (Mauvaisevitesse)	0.4	0.6

Définition de la structure du graph :



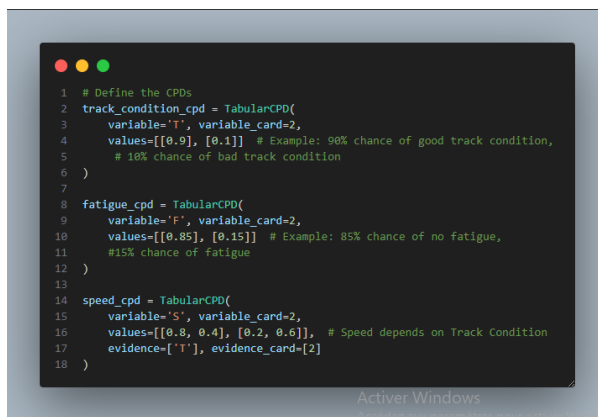
```

1 # Defining the structure of the Bayesian Network
2 f1_model = BayesianNetwork([
3     ('T', 'S'), # Track Condition affects Speed
4     ('T', 'A'), # Track Condition affects Accident
5     ('F', 'A'), # Fatigue affects Accident
6     ('S', 'A'), # Speed affects Accident
7     ('S', 'L') # Speed affects Lap Time
8 ])

```

FIGURE 3.23 – Définition structure

Définition des relations :

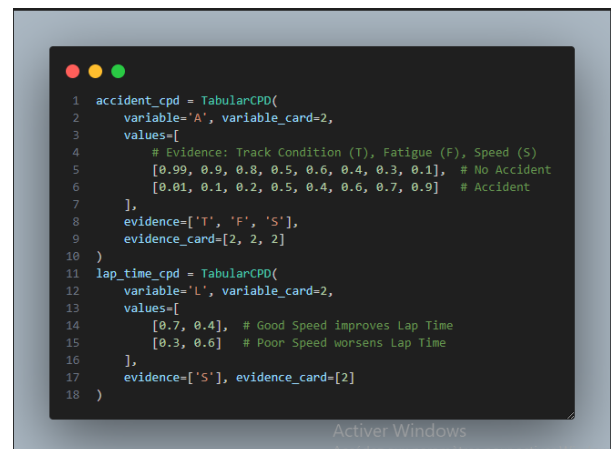


```

1 # Define the CPDs
2 track_condition_cpd = TabularCPD(
3     variable='T', variable_card=2,
4     values=[[0.9], [0.1]] # Example: 90% chance of good track condition,
5     # 10% chance of bad track condition
6 )
7
8 fatigue_cpd = TabularCPD(
9     variable='F', variable_card=2,
10    values=[[0.85], [0.15]] # Example: 85% chance of no fatigue,
11    # 15% chance of fatigue
12 )
13
14 speed_cpd = TabularCPD(
15     variable='S', variable_card=2,
16     values=[[0.8, 0.4], [0.2, 0.6]], # Speed depends on Track Condition
17     evidence=['T'], evidence_card=[2]
18 )

```

FIGURE 3.24 – Connections entre les noeuds

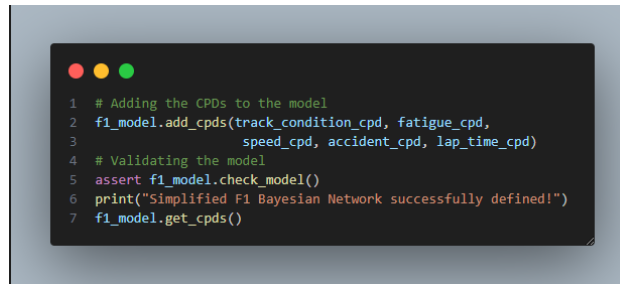


```

1 accident_cpd = TabularCPD(
2     variable='A', variable_card=2,
3     values=[
4         # Evidence: Track Condition (T), Fatigue (F), Speed (S)
5         [0.99, 0.9, 0.8, 0.5, 0.6, 0.4, 0.3, 0.1], # No Accident
6         [0.01, 0.1, 0.2, 0.5, 0.4, 0.6, 0.7, 0.9] # Accident
7     ],
8     evidence=['T', 'F', 'S'],
9     evidence_card=[2, 2, 2]
10 )
11
12 lap_time_cpd = TabularCPD(
13     variable='L', variable_card=2,
14     values=[
15         [0.7, 0.4], # Good Speed improves Lap Time
16         [0.3, 0.6] # Poor Speed worsens Lap Time
17     ],
18     evidence=['S'], evidence_card=[2]
19 )

```

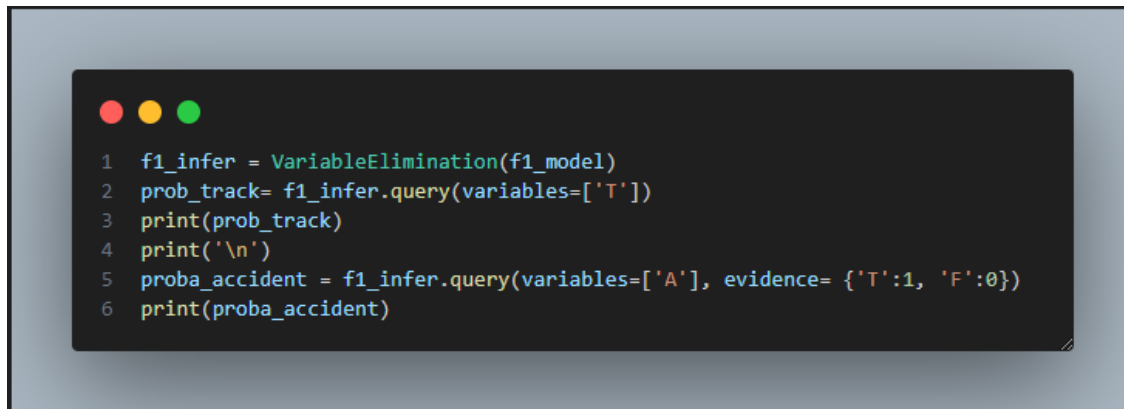
FIGURE 3.25 – Connections entre les noeuds



```
1 # Adding the CPDs to the model
2 f1_model.add_cpds(track_condition_cpd, fatigue_cpd,
3                   speed_cpd, accident_cpd, lap_time_cpd)
4 # Validating the model
5 assert f1_model.check_model()
6 print("Simplified F1 Bayesian Network successfully defined!")
7 f1_model.get_cpds()
```

FIGURE 3.26 – Affectation des connections au graph

Calcul des probabilités de track et d'accident ce réseau :



```
1 f1_infer = VariableElimination(f1_model)
2 proba_track= f1_infer.query(variables=['T'])
3 print(proba_track)
4 print('\n')
5 proba_accident = f1_infer.query(variables=['A'], evidence= {'T':1, 'F':0})
6 print(proba_accident)
```

FIGURE 3.27 – Calcul probabilités

Afichage des probabilités :

+-----+-----+
T   phi(T)
+=====+
T(0)   0.9000
+-----+-----+
T(1)   0.1000
+-----+-----+
+-----+-----+
A   phi(A)
+=====+
A(0)   0.4800
+-----+-----+
A(1)   0.5200
+-----+-----+

FIGURE 3.28 – Résultats du calcul des probabilité

## 3.6 Conclusion

Ce TP a permis d'explorer la création et l'analyse de réseaux bayésiens, depuis des structures simples jusqu'à des graphes à connexions multiples. En modélisant un problème réel, nous avons mis en évidence leur utilité pour représenter et résoudre des systèmes incertains. Cette approche offre une méthode rigoureuse et intuitive pour analyser des relations complexes et prendre des décisions basées sur des probabilités.

## CHAPITRE 4

# TP4 : INFÉRENCE LOGIQUE ET PROPAGATION GRAPHIQUE EN THÉORIE DES POSSIBILITÉS

### 4.1 Introduction