LE PROBLÈME DU PENDULE INVERSÉ

Sommaire

0. Le problème

Il s'agit de maintenir en équilibre un balai à l'envers, le manche reposant sur le bout de l'index. Phil Wasserman en parle dans *Advanced Method in Neural Computing*, Greg Viot l'étudie dans son article *Fuzzy Logic In C*, et Togai Infralogic (Bart Kosko) en distribue une démonstration sur disquette (pour compatible PC). En fait, le problème du pendule inversé est un bon représentant de l'application la plus commune de la logique floue : le contrôle moteur.

Ce document décrit le programme qui simule un chariot à moteur se déplaçant plus ou moins vite pour contrebalancer le déséquilibre du pendule. Cette simulation est volontairement réductrice : elle ne rend pas compte des phénomènes d'inertie, et les réactions du chariot sont approximatives. Elle est cependant assez rapide pour être utilisable en temps réel, à condition de disposer de capteurs permettant de mesurer les paramètres qui provoquent les réactions du système : l'angle du pendule par rapport à sa position verticale d'équilibre, et la vélocité de variation de cet angle.

1. Modélisation du contrôleur flou

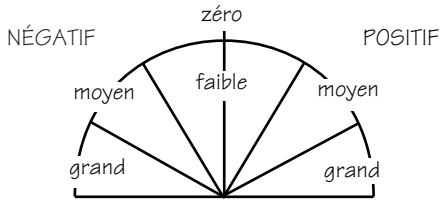
- deux entrées : l'angle et la vélocité ; on suppose que le pendule est à l'état initial d'équilibre : l'angle et la vélocité sont nuls ; une impulsion fait basculer le pendule vers la droite (angle positif) ou vers la gauche (angle négatif), avec une certaine vélocité
- une sortie : la tension du moteur ; la vélocité du basculement va provoquer une réaction plus ou moins énergique du logiciel qui représente le contrôleur : la tension du moteur sera proportionnelle à la mesure de l'angle et de la vélocité.

Ainsi, un basculement vers la gauche (négatif) devra provoquer un déplacement du chariot vers la gauche (négatif). On se donne en gros sept degrés d'appréciation de l'ampleur du basculement, en comptant la position d'équilibre :

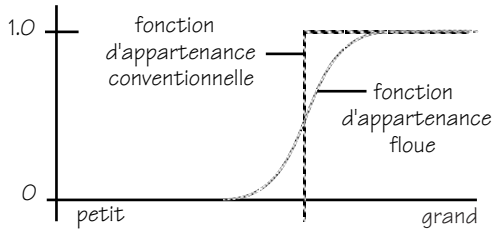
négatif grand	négatif moyen	négatif faible	équilibré zéro	positif faible	positif moyen	positif grand

Pourquoi sept ? Pour la même raison que les musiciens occidentaux codent la dynamique avec sept symboles, de ppp à fff. En plus que sept c'est un joli nombre impair.

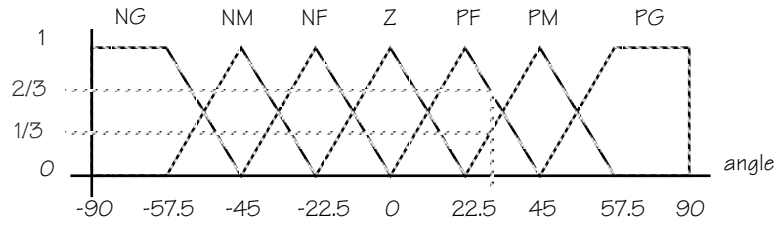
Du point de vue technique, il y aura donc 7 ensembles flous, ayant chacun leur fonction d'appartenance. Ces fonctions permettent d'exprimer des propositions logiques floues du genre : "*presqu'à l'équilibre*", "*un peu penché mais pas trop*", etc… Un angle de 30° correspond à un basculement déjà prononcé, mais plutôt faible que fort.



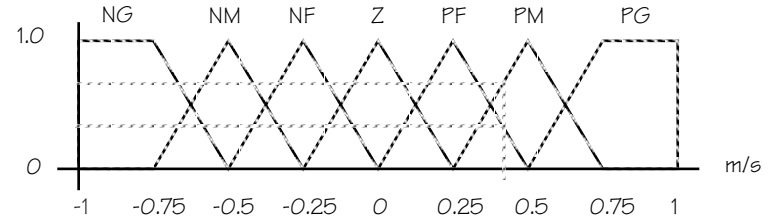
Pour exprimer ceci, il faut que les intersections entre des ensembles voisins soient non-vides, et qu'il y ait une transition douce entre le faible et le fort. Une telle transition, inexprimable en logique conventionnelle parce qu'elle prend ses valeurs de vérité dans le domaine {vrai, faux}, s'exprimera, en logique floue, avec des combinaisons de symboles qui correspondent à une valeur de vérité floue, autrement dit un rationnel, ou un réel entre 0 (complétement faux) et 1 (complétement vrai) :



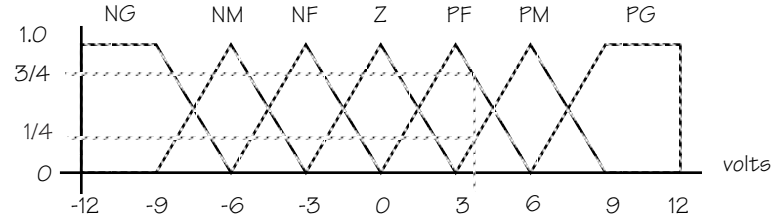
Graphiquement, ces fonctions seront représentées comme des trapèzes ou des triangles (cas particulier de trapèze). Ainsi, un angle de 30°, par exemple, appartiendra à l'ensemble positif faible (PF) pour une valeur de vérité floue de 66%, et à l'ensemble positif moyen (PM) pour 33% :



De la même manière, on représentera la vélocité du basculement entre 0 et 1 m/seconde (basculement vers la droite, positif) ou 0 et -1 (vers la gauche, négatif). En utilisant les mêmes symboles, on pourra dire qu'une chute de 40 cm/s est positive moyenne (PM) à 66%, et positive faible (PF) à 33%. Le graphe des fonctions d'appartenance aura l'allure de :



La tension à appliquer au moteur sera elle-aussi exprimée comme l'appartenance à des ensembles flous, sous la forme d'un réel positif ou négatif, selon que le chariot doit rattraper un basculement vers la droite ou vers la gauche. Cette valeur (dépendante du moteur à contrôler) sera, par exemple, ± 12 volts. On utilisera ici (mais ce n'est pas obligatoire) les mêmes symboles pour caractériser les zones de valeurs pertinentes. Le problème est donc de calculer dans quelles proportions les valeurs de vérité floue des deux paramètres d'entrée vont contraindre la valeur du paramètre de sortie. Ici, par exemple, le signal de sortie 4 volts est le résultat de la combinaison de l'influence à 75% par l'appartenance à la zone positive faible et à 25% de la zone positive moyenne :



Ces influences sont exprimées par des règles symboliques, qui décrivent la relation désirée entre les paramètres d'entrées et celui de sortie : pour un angle de basculement donné, la réaction du moteur doit être proportionnelle à la vélocité ; de même, à vélocité égale, la réaction du moteur doit être proportionnelle à l'angle de basculement. En l'absence de théorie, de telles règles peuvent être établies intuitivement. Par exemple :

- *si l'angle est positif moyen et que la vélocité est positive moyenne*
  alors applique une tension positive moyenne

On pourra ensuite raffiner les règles empiriquement pour les adapter aux conditions physiques du système sous contrôle : on s'apercevra peut-être alors qu'il faut tenir compte du caractère non-linéaire de l'accélération de la chute du pendule ou du déplacement du chariot, pour anticiper correctement les variations d'angle et déclencher la réaction appropriée. Ce qu'il y a de séduisant dans l'approche logique floue, c'est justement sa modularisation : chaque paramètre a sa propre échelle de valeurs, avec des classes de valeurs symbolisées de manière intuitive, et ses propres fonctions d'appartenance. Ceci facilite les réglages indépendants de chaque paramètre et de chaque classe au cours de la mise au point du modèle.

2. Représentations des connaissances

- variables : les paramètres angle et vélocité sont des grandeurs mesurées à l'extérieur, et transmises au système. Ces valeurs seront représentées par des symboles de variables floues, sous les identificateurs respectifs ANGLE et VÉLOCITÉ.
- ensembles flous associés : à chaque variable est associé un jeu spécifique d'ensembles flous représentant des classes de valeurs pour cette variable, et identifiés chacun par un symbole. On convertit ainsi une valeur quantitative en une valeur qualitative, référencée de manière symbolique :
  - négatif, faible, moyen ou grand : NF, NM, NG
  - zéro : ZE
  - positif, faible, moyen ou grand : PF, PM, PG
- règles : les règles expriment la relation entre l'état du monde et les réactions du contrôleur ; plus précisément, elles formalisent dans quelle mesure une combinaison d'excitations externes (valeurs des paramètres) contraint la réaction interne du contrôleur.

La table suivante présente les règles utilisées par le programme. L'ordre des règles est sans importance. Elles n'ont pas pu être essayées sur une véritable maquette, mais elles constituent quand même une assez bonne approximation des contraintes nécessaires :

règle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
angle	ZE	PG	ZE	NM	PF	PF	ZE	ZE	ZE	NF	NF	ZE	NM	NG	ZE
vélocité	PG	ZE	PM	ZE	ZE	NF	PF	ZE	NF	PF	ZE	NM	ZE	ZE	NG
tension	PG	PG	PM	PM	PF	PF	PF	ZE	NF	NF	NF	NM	NM	NG	NG

Chaque règle est constituée d'un antécédent et d'un conséquent (aussi appelés prémisses et conclusions) reliés explicitement ou implicitement par un connecteur d'implication ; l'antécédent, comme le conséquent, peut contenir plusieurs clauses reliées par des connecteurs (opérateurs) logiques pour exprimer conjonction, *disjonction*, ou encore *négation*. Ici, le connecteur & relie les deux clauses de la règle, et l'implication est implicite. Par exemple, la règle 1 se lit :

- si angle $\in$ ZE & vélocité $\in$ PG
  alors tension $\in$ PG

ou, de façon plus concise : ZE & PG $\rightarrow$ PG, puisqu'ici toutes les règles ont toujours les mêmes deux variables en antécédent, et aussi la même variable en conséquent. Mais ceci n'est, jamais, qu'un cas particulier.

Opérateurs logiques flous

En logique floue, les opérateurs peuvent, bien sûr, prendre des arguments booléens (binaires), mais aussi des réels représentant des valeurs en continu entre 0 et 1 ; le résultat de l'opération est lui aussi un réel entre 0 et 1. Ce sont donc des opérateurs numériques : le ET équivaut au minimum des deux opérandes, le OU au maximum, et le NON au complément.

x	y	x & y	min(x, y)	x | y	max(x, y)	¬ x	compl(x)
0	0	0	0	0	0	1	1
1	0	0t	0	1	1	0	0
0	1	0	0	1	1		
1	1	1	1	1	1		

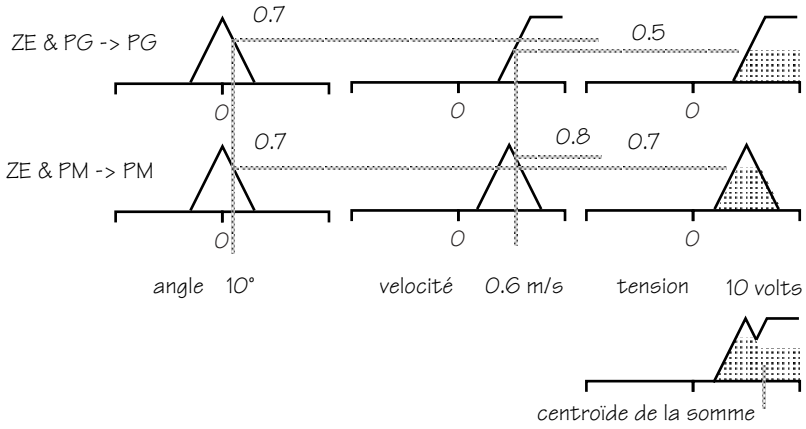
Ainsi, dans la règle (1) ZE & PG $\rightarrow$ PG, l'antécédent est la conjonction de deux classes de valeurs, une pour chaque variable. Au moment de l'évaluation, la valeur de vérité de l'antécédent est celle de la variable la plus faible.

Toutes les règles sont évaluées simultanément (en parallèle), et leur applicabilité est floue également : proportionnelle à la valeur de leur antécédent. Autrement dit, leur conséquent sera appliqué dans la mesure ou leur antécédent est vrai : en logique binaire, l'antécédent ne peut être que 0 ou 1 (vrai ou faux), alors qu'en logique flou, c'est un *réel* entre 0 et 1.

- si l'évaluation de l'antécédent retourne 0, la règle n'est tout simplement pas applicable. Les autres règles vont combiner leurs effets, les plus efficaces étant celles qui sont les plus vraies

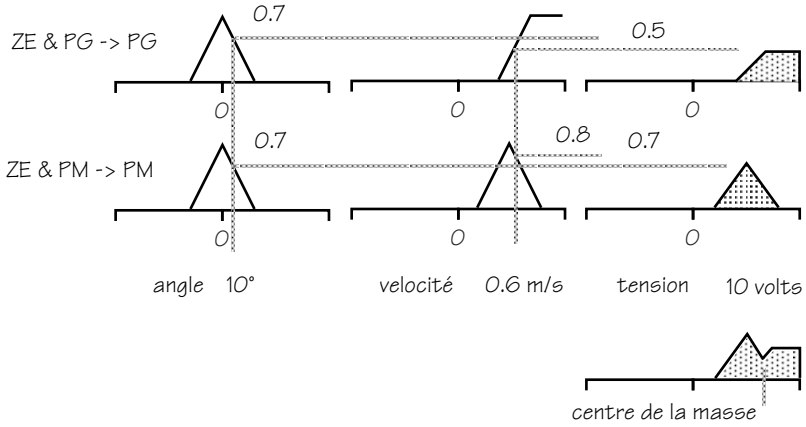
- l'évaluation des conséquents, s'ils ont plusieurs clauses, est similaire à celle des antécédents : la valeur globale en est calculée par la fonction *min*()

Enfin, toutes les règles applicables (c'est-à-dire celles dont la valeur de l'antécédent est non nul) sont effectivement appliquées, selon l'une des deux méthodes présentées ci-après (ici les règles n°1 et n°3) :



La figure ci-dessus représente pour chaque règle et pour chaque variable sa valeur et son degré d'appartenance à l'ensemble flou référencé dans la clause de la règle. Le calcul de la tension en sortie se fait par la méthode du min-max : le min de l'antécédent écrête le trapèze (ou le triangle) représentant l'ensemble flou du conséquent. On fait alors la somme des surfaces, et on en calcule le barycentre. Sa projection sur l'axe des abcisses donne graphiquement la valeur de sortie.

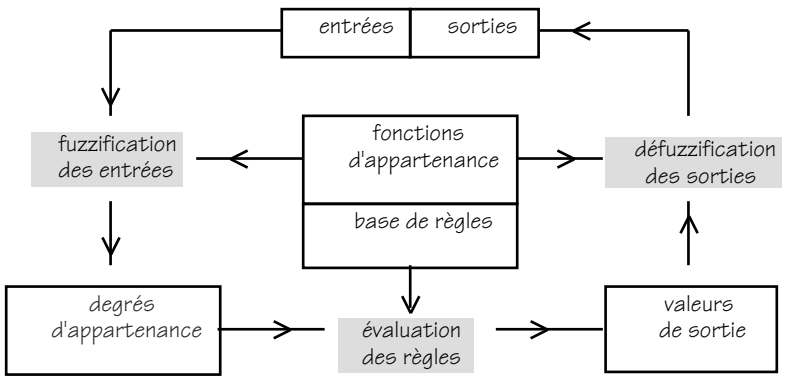
Dans la figure ci-dessous, la méthode dite du prod-max consiste à renormaliser les ensembles. Cette méthode est la plus utilisée parce que théoriquement la moins coûteuse du point de vue calcul.



Cependant, le programme présenté ici utilise la méthode du min-max. On verra plus loin l'opportunité de discuter le coût des différentes méthodes de calcul.

3. Mise en œuvre

Le programme utilise deux structures de données, qui occupent, comme on le voit sur le schéma ci-dessous, une position centrale : les *fonctions d'appartenance* associées à chaque variable, et la *base de règles*. Les valeurs d'appartenance et les valeurs de sortie des règles sont en fait incorporées dans ces structures par le truchement de pointeurs. Les paramètres d'entrée et le signal de sortie sont également des éléments de ces structures.



Les signaux d'entrée sont traités par un processus de fuzzification : le rôle de ce processus est de calculer, pour chaque variable, le degré d'appartenance de sa valeur à chaque ensemble flou associé. Les règles sont alors évaluées, produisant une valeur floue pour chaque conséquent. Ces valeurs sont enfin défuzzifiées, ici par la méthode du min-max, pour produire une seule valeur, la résultante de l'interaction des règles.

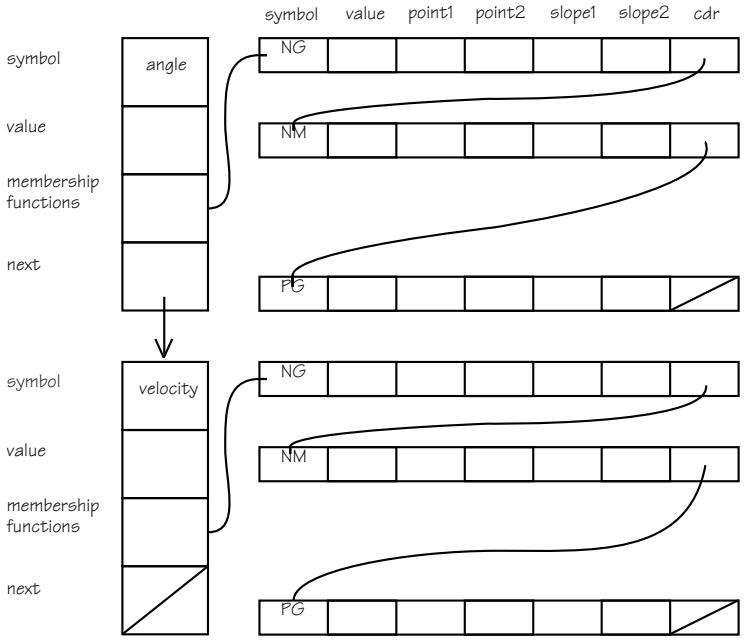
Structures des données

Le programme est conçu pour configurer dynamiquement ses données en les lisant dans un fichier ascii standard ; il construit donc des listes en allouant dynamiquement l'espace mémoire nécessaire au fur et à mesure de sa lecture (cf. *setup_fuzzy_sets*, *setup_rules* et *setup_clause*).

Les fichiers sont lus dans l'ordre suivant : données de la variable 1 d'antécédent, données de la variable 2 d'antécédent, données de la variable 1 de conséquent, données de règles. Le format des fichiers est tel qu'il peut être facilement décrypté par *fscanf*. On remarque que les données sont ici des entiers sur 8 bits non signés, capables d'exprimer des valeurs entre 0 et 255. La valeur 127 est interprétée comme un zéro, et les valeurs inférieures sont interprétées comme négatives.

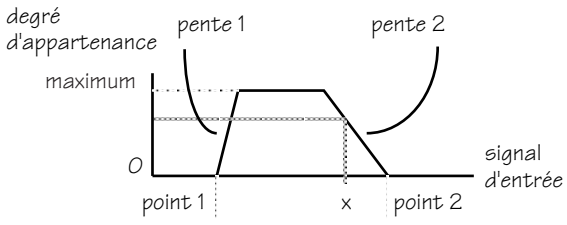
```
Angle                    Velocité                 Tension
NG   0   1  31  63       NG   0   1  31  63       NG   0   1  31  63
NM  31  63  63  95       NM  31  63  63  95       NM  31  63  63  95
NF  63  95  95 127       NF  63  95  95 127       NF  63  95  95 127
ZE  95 127 127 159       ZE  95 127 127 159       ZE  95 127 127 159
PF 127 159 159 191       PF 127 159 159 191       PF 127 159 159 191
PM 159 191 191 223       PM 159 191 191 223       PM 159 191 191 223
PG 191 223 254 255       PG 191 223 254 255       PG 191 223 254 255
```

Ces trois premiers fichiers codent, pour chacune des 3 variables, les 7 ensembles flous sous la forme de 4 nombres (cf. *structure Mfunc*) qui représentent les abcisses des sommets des trapèzes : les ordonnées sont implicites : 0 pour l'ordonnée de la base, et 255 pour l'ordonnée du sommet) ; ces coordonnées sont converties en pentes, pendant la lecture. Le programme mémorise cette information sous la forme de listes de listes.



La première liste (cf. structure *Sysio*) est celle des variables d'antécédent et de leurs ensembles flous (cf. structure *Sysin*) : ici, il y a deux variables, donc la liste n'a que deux éléments ; la deuxième liste (cf. structure *Sysout*) n'a, elle, qu'un élément, puisque les conséquents n'ont qu'une clause.

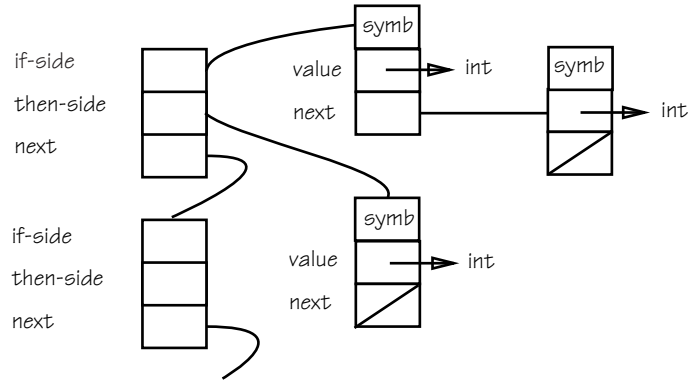
Chaque variable est associée à un symbole (angle, velocity, tension), et une valeur : valeur d'entrée pour l'angle et la vélocité, valeur de sortie pour la tension ; pour chaque variable il y a 7 ensembles flous, chacun associé à un symbole distinct : NG, NM, NF, ZE, PF, PM et PG ; chaque ensemble est associé à un scalaire qui représente le degré d'appartenance de la valeur de la variable pour cet ensemble ; un ensemble est décrit par un trapèzoïde : deux points et deux pentes (ci-dessous) :



Le quatrième fichier est celui des 15 règles de notre exemple :

ZE PG PG	ZE PM PM	PF ZE PF	ZE ZE ZE	ZE NF NF	ZE NM NM	NG ZE NG
PG ZE PG	NM ZE PM	PF NF PF		NF PF NF	NM ZE NM	ZE NG NG
		ZE PF PF		NF ZE NF		

il sert à construire une troisième liste, qui comprend ici 15 éléments (cf. structure *Rule*) ; chaque élément renvoie à deux sous-listes, correspondant respectivement aux clauses de son antécédent et à celles de son conséquent (ici un seule clause conséquente) ; chaque clause (cf. structure *Elt*) est associée au symbole de l'ensemble flou auquel elle fait référence, ainsi qu'à un scalaire correspondant à l'appartenance de la valeur à cet ensemble. En fait chaque clause pointe directement sur le scalaire membre de la structure qui représente l'ensemble flou (cf. fonction *evaluate*) :



Structure de contrôle

Le programme prend en entrée deux arguments : l'angle et la vélocité initiale ;

- la fonction *setup* est chargée de l'initialisation des structures de données : c'est pour ça qu'elle reçoit en paramètre un vecteur de pointeurs sur les chaînes de caractères qui correspondent aux noms des fichiers de données ;
- vient ensuite une boucle qui ne se termine que lorsque le pendule est revenu à l'état d'équilibre ; en guise de garde-fou, la variable *tally* force l'arrêt au bout de quarante tours, mais c'est peut-être inutile ;
  - la fonction *reset_outputs* n'est pas nécessaire au bon fonctionnement du programme : voir la discussion ci-après ;
  - la fonction *control_in* simule l'entrée des valeurs qui devraient normalement provenir d'instruments de mesure externes ;
  - ces valeurs doivent d'abord subir une *fuzzification*,
  - avant que les règles puissent être *évaluées* ;
  - puis les valeurs des conséquents sont *défuzzifiées*, pour produire l'unique résultante de sortie ;
  - la fonction *control_out* simule la sortie du signal et calcule les nouveaux signaux d'entrée ;
  - la fonction *dump* ne sert qu'à révéler l'état courant des données, tour après tour ; ces résultats sont présentés et discutés dans la section suivante.

4. Essais commentés

- **exemple 1, ANGLE = -27 et VELOCITÉ = 33** : le programme ramène le pendule à l'équilibre en 7 tours ; ces données s'affichent sur l'écran à la fin de chaque tour, il est donc normal que les valeurs aient déjà été décrémentées ; il est intéressant d'examiner les données de la fonction *dump* pour voir quelles règles sont déclenchées. Valeurs entre - 127 et + 128 (cf. fonction *control_out*) :

tour	1	2	3	4	5	6	7
tension	14	1	1	1	1	1	0
angle	-13	-6	-3	-1	0	0	0
vélocité	16	8	4	2	1	0	0

RÉSULTAT DU DUMP

représentation interne : valeurs entre 0 et 255
(1) symbole (2) valeur (3) point 1 (4 point 2

```
Angle : 100                Velocity : 160             Tension : 113
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 189 [63 127]          NF : 0 [63 127]            NF : 189 [63 127]
ZE : 35 [95 159]           ZE : 0 [95 159]            ZE : 0 [95 159]
PF : 0 [127 191]           PF : 217 [127 191]         PF : 35 [127 191]
PM : 0 [159 223]           PM : 7 [159 223]           PM : 7 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE =  35 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE =   0 -> PG =   0
Rule  3, ZE =  35 , PM =   7 -> PM =   7
Rule  4, NM =   0 , ZE =   0 -> PM =   7
Rule  5, PF =   0 , ZE =   0 -> PF =  35
Rule  6, PF =   0 , NF =   0 -> PF =  35
Rule  7, ZE =  35 , PF = 217 -> PF =  35
Rule  8, ZE =  35 , ZE =   0 -> ZE =   0
Rule  9, ZE =  35 , NF =   0 -> NF = 189
Rule 10, NF = 189 , PF = 217 -> NF = 189
Rule 11, NF = 189 , ZE =   0 -> NF = 189
Rule 12, ZE =  35 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE =   0 -> NM =   0
Rule 14, NG =   0 , ZE =   0 -> NG =   0
Rule 15, ZE =  35 , NG =   0 -> NG =   0

Angle = 114 — velocity = 143                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 114                Velocity : 143             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 91 [63 127]           NF : 0 [63 127]            NF : 91 [63 127]
ZE : 133 [95 159]          ZE : 112 [95 159]          ZE : 112 [95 159]
PF : 0 [127 191]           PF : 112 [127 191]         PF : 112 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 133 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 112 -> PG =   0
Rule  3, ZE = 133 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 112 -> PM =   0
Rule  5, PF =   0 , ZE = 112 -> PF = 112
Rule  6, PF =   0 , NF =   0 -> PF = 112
Rule  7, ZE = 133 , PF = 112 -> PF = 112
Rule  8, ZE = 133 , ZE = 112 -> ZE = 112
Rule  9, ZE = 133 , NF =   0 -> NF =  91
Rule 10, NF =  91 , PF = 112 -> NF =  91
Rule 11, NF =  91 , ZE = 112 -> NF =  91
Rule 12, ZE = 133 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 112 -> NM =   0
Rule 14, NG =   0 , ZE = 112 -> NG =   0
Rule 15, ZE = 133 , NG =   0 -> NG =   0

Angle = 121 — velocity = 135                          • • • End of Data • • *
```

```
• • • Membership Functions • • •
Angle : 121                Velocity : 135            Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]             NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]            NM : 0 [31 95]
NF : 42 [63 127]           NF : 0 [63 127]           NF : 42 [63 127]
ZE : 182 [95 159]          ZE : 168 [95 159]         ZE : 168 [95 159]
PF : 0 [127 191]           PF : 56 [127 191]         PF : 56 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]          PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]          PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 182 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 168 -> PG =   0
Rule  3, ZE = 182 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 168 -> PM =   0
Rule  5, PF =   0 , ZE = 168 -> PF =  56
Rule  6, PF =   0 , NF =   0 -> PF =  56
Rule  7, ZE = 182 , PF =  56 -> PF =  56
Rule  8, ZE = 182 , ZE = 168 -> ZE = 168
Rule  9, ZE = 182 , NF =   0 -> NF =  42
Rule 10, NF =  42 , PF =  56 -> NF =  42
Rule 11, NF =  42 , ZE = 168 -> NF =  42
Rule 12, ZE = 182 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 168 -> NM =   0
Rule 14, NG =   0 , ZE = 168 -> NG =   0
Rule 15, ZE = 182 , NG =   0 -> NG =   0

Angle = 124 — velocity = 131                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 124                Velocity : 131            Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]             NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]            NM : 0 [31 95]
NF : 21 [63 127]           NF : 0 [63 127]           NF : 21 [63 127]
ZE : 203 [95 159]          ZE : 196 [95 159]         ZE : 196 [95 159]
PF : 0 [127 191]           PF : 28 [127 191]         PF : 28 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]          PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]          PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 203 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 196 -> PG =   0
Rule  3, ZE = 203 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 196 -> PM =   0
Rule  5, PF =   0 , ZE = 196 -> PF =  28
Rule  6, PF =   0 , NF =   0 -> PF =  28
Rule  7, ZE = 203 , PF =  28 -> PF =  28
Rule  8, ZE = 203 , ZE = 196 -> ZE = 196
Rule  9, ZE = 203 , NF =   0 -> NF =  21
Rule 10, NF =  21 , PF =  28 -> NF =  21
Rule 11, NF =  21 , ZE = 196 -> NF =  21
Rule 12, ZE = 203 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 196 -> NM =   0
Rule 14, NG =   0 , ZE = 196 -> NG =   0
Rule 15, ZE = 203 , NG =   0 -> NG =   0

Angle = 126 — velocity = 129                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 126                Velocity : 129            Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]             NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]            NM : 0 [31 95]
NF : 7 [63 127]            NF : 0 [63 127]           NF : 7 [63 127]
ZE : 217 [95 159]          ZE : 210 [95 159]         ZE : 210 [95 159]
PF : 0 [127 191]           PF : 14 [127 191]         PF : 14 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]          PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]          PG : 0 [191 255]
```

```
• • • Rule Base • • •
Rule  1, ZE = 217 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 210 -> PG =   0
Rule  3, ZE = 217 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 210 -> PM =   0
Rule  5, PF =   0 , ZE = 210 -> PF =  14
Rule  6, PF =   0 , NF =   0 -> PF =  14
Rule  7, ZE = 217 , PF =  14 -> PF =  14
Rule  8, ZE = 217 , ZE = 210 -> ZE = 210
Rule  9, ZE = 217 , NF =   0 -> NF =   7
Rule 10, NF =   7 , PF =  14 -> NF =   7
Rule 11, NF =   7 , ZE = 210 -> NF =   7
Rule 12, ZE = 217 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 210 -> NM =   0
Rule 14, NG =   0 , ZE = 210 -> NG =   0
Rule 15, ZE = 217 , NG =   0 -> NG =   0

Angle = 127 — velocity = 128                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 127                Velocity : 128             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 0 [63 127]            NF : 0 [63 127]
ZE : 224 [95 159]          ZE : 217 [95 159]          ZE : 217 [95 159]
PF : 0 [127 191]           PF : 7 [127 191]           PF : 7 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 224 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 217 -> PG =   0
Rule  3, ZE = 224 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 217 -> PM =   0
Rule  5, PF =   0 , ZE = 217 -> PF =   7
Rule  6, PF =   0 , NF =   0 -> PF =   7
Rule  7, ZE = 224 , PF =   7 -> PF =   7
Rule  8, ZE = 224 , ZE = 217 -> ZE = 217
Rule  9, ZE = 224 , NF =   0 -> NF =   0
Rule 10, NF =   0 , PF =   7 -> NF =   0
Rule 11, NF =   0 , ZE = 217 -> NF =   0
Rule 12, ZE = 224 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 217 -> NM =   0
Rule 14, NG =   0 , ZE = 217 -> NG =   0
Rule 15, ZE = 224 , NG =   0 -> NG =   0

Angle = 127 — velocity = 127                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 127                Velocity : 127             Tension : 127
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 0 [63 127]            NF : 0 [63 127]
ZE : 224 [95 159]          ZE : 224 [95 159]          ZE : 224 [95 159]
PF : 0 [127 191]           PF : 0 [127 191]           PF : 0 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 224 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 224 -> PG =   0
Rule  3, ZE = 224 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 224 -> PM =   0
Rule  5, PF =   0 , ZE = 224 -> PF =   0
Rule  6, PF =   0 , NF =   0 -> PF =   0
Rule  7, ZE = 224 , PF =   0 -> PF =   0
Rule  8, ZE = 224 , ZE = 224 -> ZE = 224
Rule  9, ZE = 224 , NF =   0 -> NF =   0
Rule 10, NF =   0 , PF =   0 -> NF =   0
Rule 11, NF =   0 , ZE = 224 -> NF =   0
Rule 12, ZE = 224 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 224 -> NM =   0
Rule 14, NG =   0 , ZE = 224 -> NG =   0
Rule 15, ZE = 224 , NG =   0 -> NG =   0

Angle = 127 — velocity = 127                          • • • End of Data • • •
```

- **EXEMPLE 2 : ANGLE = 33 ET VÉLOCITÉ = -27** : le programme ramène le pendule à l'équilibre en 7 tours ; ces données s'affichent sur l'écran à la fin de chaque tour, il est donc normal que les valeurs aient déjà été décrémentées ; il est intéressant d'examiner les données de la fonction *dump* pour voir quelles règles sont déclenchées.

tour	1	2	3	4	5	6	7
tension	32	1	1	1	1	1	0
angle	16	8	4	2	1	0	0
vélocité	-13	-6	-3	-1	0	0	0

RÉSULTATS DU DUMP

```
Angle : 160                Velocity : 100             Tension : 159
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 189 [63 127]          NF : 0 [63 127]
ZE : 0 [95 159]            ZE : 35 [95 159]           ZE : 0 [95 159]
PF : 217 [127 191]         PF : 0 [127 191]           PF : 189 [127 191]
PM : 7 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE =   0 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE =  35 -> PG =   0
Rule  3, ZE =   0 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE =  35 -> PM =   0
Rule  5, PF = 217 , ZE =  35 -> PF = 189
Rule  6, PF = 217 , NF = 189 -> PF = 189
Rule  7, ZE =   0 , PF =   0 -> PF = 189
Rule  8, ZE =   0 , ZE =  35 -> ZE =   0
Rule  9, ZE =   0 , NF = 189 -> NF =   0
Rule 10, NF =   0 , PF =   0 -> NF =   0
Rule 11, NF =   0 , ZE =  35 -> NF =   0
Rule 12, ZE =   0 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE =  35 -> NM =   0
Rule 14, NG =   0 , ZE =  35 -> NG =   0
Rule 15, ZE =   0 , NG =   0 -> NG =   0

Angle = 143 — velocity = 114                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 143                Velocity : 114             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 91 [63 127]           NF : 91 [63 127]
ZE : 112 [95 159]          ZE : 133 [95 159]          ZE : 112 [95 159]
PF : 112 [127 191]         PF : 0 [127 191]           PF : 112 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 112 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 133 -> PG =   0
Rule  3, ZE = 112 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 133 -> PM =   0
Rule  5, PF = 112 , ZE = 133 -> PF = 112
Rule  6, PF = 112 , NF =  91 -> PF = 112
Rule  7, ZE = 112 , PF =   0 -> PF = 112
Rule  8, ZE = 112 , ZE = 133 -> ZE = 112
Rule  9, ZE = 112 , NF =  91 -> NF =  91
Rule 10, NF =   0 , PF =   0 -> NF =  91
Rule 11, NF =   0 , ZE = 133 -> NF =  91
Rule 12, ZE = 112 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 133 -> NM =   0
Rule 14, NG =   0 , ZE = 133 -> NG =   0
Rule 15, ZE = 112 , NG =   0 -> NG =   0

Angle = 135 — velocity = 121                          • • • End of Data • • •
```

```
• • • Membership Functions • • •
Angle : 135                Velocity : 121             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 42 [63 127]           NF : 42 [63 127]
ZE : 168 [95 159]          ZE : 182 [95 159]          ZE : 168 [95 159]
PF : 56 [127 191]          PF : 0 [127 191]           PF : 56 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 168 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 182 -> PG =   0
Rule  3, ZE = 168 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 182 -> PM =   0
Rule  5, PF =  56 , ZE = 182 -> PF =  56
Rule  6, PF =  56 , NF =  42 -> PF =  56
Rule  7, ZE = 168 , PF =   0 -> PF =  56
Rule  8, ZE = 168 , ZE = 182 -> ZE = 168
Rule  9, ZE = 168 , NF =  42 -> NF =  42
Rule 10, NF =   0 , PF =   0 -> NF =  42
Rule 11, NF =   0 , ZE = 182 -> NF =  42
Rule 12, ZE = 168 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 182 -> NM =   0
Rule 14, NG =   0 , ZE = 182 -> NG =   0
Rule 15, ZE = 168 , NG =   0 -> NG =   0

Angle = 131 — velocity = 124                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 131                Velocity : 124             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 21 [63 127]           NF : 21 [63 127]
ZE : 196 [95 159]          ZE : 203 [95 159]          ZE : 196 [95 159]
PF : 28 [127 191]          PF : 0 [127 191]           PF : 28 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 196 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 203 -> PG =   0
Rule  3, ZE = 196 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 203 -> PM =   0
Rule  5, PF =  28 , ZE = 203 -> PF =  28
Rule  6, PF =  28 , NF =  21 -> PF =  28
Rule  7, ZE = 196 , PF =   0 -> PF =  28
Rule  8, ZE = 196 , ZE = 203 -> ZE = 196
Rule  9, ZE = 196 , NF =  21 -> NF =  21
Rule 10, NF =   0 , PF =   0 -> NF =  21
Rule 11, NF =   0 , ZE = 203 -> NF =  21
Rule 12, ZE = 196 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 203 -> NM =   0
Rule 14, NG =   0 , ZE = 203 -> NG =   0
Rule 15, ZE = 196 , NG =   0 -> NG =   0

Angle = 129 — velocity = 126                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 129                Velocity : 126             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 7 [63 127]            NF : 7 [63 127]
ZE : 210 [95 159]          ZE : 217 [95 159]          ZE : 210 [95 159]
PF : 14 [127 191]          PF : 0 [127 191]           PF : 14 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]
```

```
• • • Rule Base • • •
Rule  1, ZE = 210 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 217 -> PG =   0
Rule  3, ZE = 210 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 217 -> PM =   0
Rule  5, PF =  14 , ZE = 217 -> PF =  14
Rule  6, PF =  14 , NF =   7 -> PF =  14
Rule  7, ZE = 210 , PF =   0 -> PF =  14
Rule  8, ZE = 210 , ZE = 217 -> ZE = 210
Rule  9, ZE = 210 , NF =   7 -> NF =   7
Rule 10, NF =   0 , PF =   0 -> NF =   7
Rule 11, NF =   0 , ZE = 217 -> NF =   7
Rule 12, ZE = 210 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 217 -> NM =   0
Rule 14, NG =   0 , ZE = 217 -> NG =   0
Rule 15, ZE = 210 , NG =   0 -> NG =   0

Angle = 128 — velocity = 127                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 128                Velocity : 127             Tension : 128
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 0 [63 127]            NF : 0 [63 127]
ZE : 217 [95 159]          ZE : 224 [95 159]          ZE : 217 [95 159]
PF : 7 [127 191]           PF : 0 [127 191]           PF : 7 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 217 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 224 -> PG =   0
Rule  3, ZE = 217 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 224 -> PM =   0
Rule  5, PF =   7 , ZE = 224 -> PF =   7
Rule  6, PF =   7 , NF =   0 -> PF =   7
Rule  7, ZE = 217 , PF =   0 -> PF =   7
Rule  8, ZE = 217 , ZE = 224 -> ZE = 217
Rule  9, ZE = 217 , NF =   0 -> NF =   0
Rule 10, NF =   0 , PF =   0 -> NF =   0
Rule 11, NF =   0 , ZE = 224 -> NF =   0
Rule 12, ZE = 217 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 224 -> NM =   0
Rule 14, NG =   0 , ZE = 224 -> NG =   0
Rule 15, ZE = 217 , NG =   0 -> NG =   0

Angle = 127 — velocity = 127                          • • • End of Data • • •

• • • Membership Functions • • •
Angle : 127                Velocity : 127             Tension : 127
NG : 0 [0 63]              NG : 0 [0 63]              NG : 0 [0 63]
NM : 0 [31 95]             NM : 0 [31 95]             NM : 0 [31 95]
NF : 0 [63 127]            NF : 0 [63 127]            NF : 0 [63 127]
ZE : 224 [95 159]          ZE : 224 [95 159]          ZE : 224 [95 159]
PF : 0 [127 191]           PF : 0 [127 191]           PF : 0 [127 191]
PM : 0 [159 223]           PM : 0 [159 223]           PM : 0 [159 223]
PG : 0 [191 255]           PG : 0 [191 255]           PG : 0 [191 255]

• • • Rule Base • • •
Rule  1, ZE = 224 , PG =   0 -> PG =   0
Rule  2, PG =   0 , ZE = 224 -> PG =   0
Rule  3, ZE = 224 , PM =   0 -> PM =   0
Rule  4, NM =   0 , ZE = 224 -> PM =   0
Rule  5, PF =   0 , ZE = 224 -> PF =   0
Rule  6, PF =   0 , NF =   0 -> PF =   0
Rule  7, ZE = 224 , PF =   0 -> PF =   0
Rule  8, ZE = 224 , ZE = 224 -> ZE = 224
Rule  9, ZE = 224 , NF =   0 -> NF =   0
Rule 10, NF =   0 , PF =   0 -> NF =   0
Rule 11, NF =   0 , ZE = 224 -> NF =   0
Rule 12, ZE = 224 , NM =   0 -> NM =   0
Rule 13, NM =   0 , ZE = 224 -> NM =   0
Rule 14, NG =   0 , ZE = 224 -> NG =   0
Rule 15, ZE = 224 , NG =   0 -> NG =   0

Angle = 127 — velocity = 127                          • • • End of Data • • •
```

5. Quelques remarques

L'implémentation en entiers positifs : …

Les types *distance*, *slope* et *sysval*

Le simulacre du retour à l'équilibre : il est tout à fait discutable.

Une astuce inutile à l'initialisation : les valeurs d'appartenance des variables sont initialisées par un entier incrémenté à chaque nouvelle allocation de structure. Ceci permet de vérifier que les règles pointent bien sur les bonnes adresses.

La fonction *reset_outputs* : elle n'est pas nécessaire au bon fonctionnement du programme, mais elle est bien commode pour debugger : en effet, les règles ne couvrent pas toutes les possibilités de combinaisons de valeurs pour les paramètres d'entrée (ainsi, il n'y a pas de règle pour angle = PG et vélocité = PG), et dans certains cas, aucune règle n'étant applicable, les valeurs ne sont pas mises à jour. Ceci ne facilite pas l'examen des données puisqu'on ne sait pas de quand datent les valeurs mémorisées. Une remise à zéro générale règle la question. Par ailleurs, si aucune règle n'est applicable, la somme des surfaces est nulle ; dans ce cas, le programme avorte puisqu'il y aurait de toutes façons une erreur de division par zéro.

6. Bibliographie

- Togai Infralogic, Inc. 1990. Fuzzy-C pendulum demonstration, version 3,1. Irvine CA
- Viot (Greg), 1993. "*Fuzzy Logic In C*", Dr Dobb's Journal #197 (February, 1993)
- Wasserman (Philip D.), Advanced Method in Neural Computing, Add. Wesley

Le code

```
/*  General-purpose fuzzy inference engine supporting any number of system inputs and outputs, membership
    functions, and rules.
•   Membership functions can be any shape defineable by 2 points and 2 slopes — trapezoids, triangles, rectangles,
    etc. Rules can have any number of antecedents and outputs, and this can vary from rule to rule.
    -   "Min" method is used to compute rule strength,
    -   "Max" for applying rule strengths,
    •   "Center-of-Gravity" for defuzzification. */

#include <stdio.h>
#include <console.h>                                                    // MacOs specific

typedef int distance ;
typedef int slope ;
typedef int sysval ;

const int MaxName = 11 ;                                    // max number of characters in names

/*  Sysio structure builds a list of system inputs and a list of system outputs. After initialization, these lists are fixed,
    except for value field which is updated on every inference pass. */

struct Sysio
{   char symb[MaxName] ;                                          // name of system input/output
    sysval value ;                                                // value of system input/output
    struct Mfunc * membership ;                  // list of membership functions for this system input/output
    struct Sysio * next ; } ;                                     // pointer to next input/output

// Membership functions are associated with each system input and output.

struct Mfunc
{   char symb[MaxName] ;                                  // symbol of membership function (fuzzy set)
    sysval value ;                                  // degree of membership or output strength
    distance point1 ;                                 // leftmost x-axis point of memb. function
    distance point2 ;                                // rightmost x-axis point of memb. function
    slope slope1 ;                                      // slope of left side of memb. function
    slope slope2 ;                                     // slope of right side of memb. function
    struct Mfunc * next ; } ;                                  // pointer to next memb. function

/*  Each rule has an if-side and a then-side. Elements making up if-side are pointers to antecedent values inside Mfunc
    structure. Elements making up then-side of rule are pointers to output strength values, also inside Mfunc structure.
    Each rule structure contains a pointer to next rule in rule base. */

struct Rule
{   struct Elt * if_side ;                                    // list of antecedents in rule
    struct Elt * th_side ;                                        // list of outputs in rule
    struct Rule * next ; } ;                                      // next rule in rule base

struct Elt
{   char symb[MaxName] ;                                          // symbol of fuzzy value
    sysval * value ;                          // pointer to antecedent (or output strength) value
    struct Elt * next ; } ;                     // next antecedent (or output) element in rule

Rule * rule_base ;                                                // head of rule-list
Sysio * sysin ;                                                   // head of antecedent-list
Sysio * sysout ;                                                  // head of consequent-list

#include <stdlib.h>
#include <string.h>

sysval max (sysval x, sysval y) { return (x > y ? x : y) ; }
sysval min (sysval x, sysval y) { return (x < y ? x : y) ; }

/*  This implementation of inverted pendulum control problem has:
    •   System Inputs : 2 (pendulum angle and velocity) ;
    •   System Outputs : 1 (force supplied to base of pendulum) ;
    •   Membership Functions : 7 per system input/output ;
    •   Rules : 15 (each with 2 antecedents and 1 output).
•   If more precision required, integers are to be changed to real numbers. */

const sysval MaxVal = 255 ;                          // max number assigned as degree of membership
const sysval ZeroV = 255 / 2 ;                       // virtual zero value (signed short 8-bit integer)
const int NbInputs = 2 ;                                                    // number of inputs
const int NbOutputs = 1 ;                                                   // number of outputs
const int NbMf = 7 ;                                          // number of membership functions
const int NbRules = 15 ;                                                    // number of rules

int tally = 0 ;                                                             //• • • debug trick
```

```
Elt * setup_clause (Mfunc *, char *) ;                                  // proto
Rule * setup_rules (char *) ;                                           // proto
Sysio * setup_fuzzy_sets (char *) ;                                     // proto
sysval trapezoid (Mfunc *) ;                                            // proto
void defuzzify (Sysio *) ;                                              // proto
void dump (sysval, sysval) ;                                            // proto
void dump_mfuncs (Sysio *) ;                                            // proto
void dump_rules (Rule *) ;                                              // proto
void error_opening_file (char *) ;                                      // proto
void error_reading_file (char *, char *) ;                              // proto
void evaluate (Rule *) ;                                                // proto
void eval_membership (Mfunc *, sysval) ;                                // proto
void fuzzify (Sysio *) ;                                                // proto
void control_in (sysval, sysval) ;                                      // proto
void malloc_error (char *) ;                                            // proto
void no_match (sysval, sysval) ;                                        // proto
void control_out (sysval, sysval &, sysval &) ;                         // proto
void reset_outputs (Sysio *) ;                                          // proto
void setup (char **) ;                                                  // proto
void set_console (int, int) ;                                  // Mac Os specific

//FILE * display = {stdout} ;
FILE * display = fopen ("FL Engine Data", "w") ;                        // print to file

inline void set_console (int r, int c)                         // Mac Os specific
{   console_options.title = "\pInverted Pendulum" ;
    console_options.top = 45 ;                                          // 50
    console_options.left = 9 ;                                          // 10
    console_options.nrows = r ;                                         // 38
    console_options.ncols = c ; }                                       // 80

inline void usage (char * prog)
{   printf ("%s <angle> <velocity>\n", prog) ;
    printf ("Must supply 2 integers in [0, 255]\n") ;
    exit (1) ; }

void main (int argc, char **argv)
{   char * fname[] = { "Angle.dat", "Velocity.dat", "Force.dat", "Rules.dat" };  // not parameters yet
    set_console (15, 40) ;                                          // Mac Os specific
    argc = ccommand (&argv) ;                                       // Mac Os specific
    if (argc != 3) usage (argv[0]) ;
    sysval angle = atoi (argv[1]) ;                                 // assume integer type
    sysval velocity = atoi (argv[2]) ;                              // assume integer type
    setup (fname) ;
    sysval * force = & sysout->value ;
    while (angle != ZeroV || velocity != ZeroV || * force != ZeroV)         // until stable
    {   reset_outputs (sysout) ;
        if (++tally > 40) break ;                                   // • • • debug trick
        control_in (angle, velocity) ;
        fuzzify (sysin) ;
        evaluate (rule_base) ;
        defuzzify (sysout) ;
        control_out (* force, angle, velocity) ;
        dump (angle, velocity) ; } }

void control_in (sysval angle, sysval velocity)
{   sysin->value = angle ;
    (sysin->next)->value = velocity ; }

inline sysval new_input (sysval x) { return (ZeroV + ((x - ZeroV) / 2)) ; }     // simulacre

void control_out (sysval f, sysval & a, sysval & v)             // simulate effects of applying rules
{   a = new_input (a) ;
    v = new_input (v) ;
    printf ("%3d -> %3d %3d\n", f - ZeroV, a - ZeroV, v - ZeroV) ; }    // assume integer type

void reset_outputs (Sysio * start)                                  // debugging purpose only
{   Sysio * var ;                                                   // pointer to system output
    Mfunc * M ;                                 // pointer to output membership function
    for (var = start ; var ; var = var->next)
    {   for (M = var->membership ; M ; M = M->next)
            M->value = 0 ;                                          // reset all outputs
        var->value = 0 ; } }                            // reset value for each system output

/*  Fuzzify
•   Degree of membership value is calculated for each membership function of each system input. Values correspond to
antecedents in rules. */

void fuzzify (Sysio * start)
{   Sysio * var ;                                                   // pointer to system input
    Mfunc * M ;                                             // pointer to membership function
    for (var = start ; var ; var = var->next)
        for (M = var->membership ; M ; M = M->next)
            eval_membership (M, var->value) ; }
```

```
/*  Compute Degree of Membership — degree to which input is a member of mf is calculated as follows:
    1. Compute delta terms to determine if input is inside or outside membership function.
    2. If outside, degree of membership is 0. Otherwise, smaller of delta_1 * slope1 and delta_2 * slope2 applies.
    3. Enforce upper limit. */

void eval_membership (Mfunc * M, sysval input)
{   distance delta_1, delta_2 ;
    delta_1 = input - M->point1 ;
    delta_2 = M->point2 - input ;
    if ((delta_1 <= 0) || (delta_2 <= 0)) M->value = 0 ;                    // input outside memb. function
    else M->value = min (min (M->slope1 * delta_1, M->slope2 * delta_2), MaxVal) ; }          // enforce upper limit

/*  Rule Evaluation — Each rule consists of
    -  list of pointers to antecedents (if-side)
    -  list of pointers to outputs (then-side)
    -  pointer to next rule in rule base
•   When a rule is evaluated, its antecedents are ANDed together, using a minimum function, to form strength of rule.
•   Then strength is applied to each of listed rule outputs.
•   If an output has already been assigned a rule strength, during current inference pass, a maximum function is used to
    determine which strength should apply. */

void evaluate (Rule * start)
{   Rule * rule ;
    Elt * E ;                                                                  // pointer to rule element
    sysval strength ;                                                  // strength of rule currently being evaluated
    for (rule = start ; rule ; rule = rule->next)                                                          //
    {   strength = MaxVal ;                                                  // max rule strength allowed
        for (E = rule->if_side ; E ; E = E->next)
            strength = min (strength, * (E->value)) ;                                  // determine strength
        for (E = rule->th_side ; E ; E = E->next)
            * (E->value) = max (strength, * (E->value)) ;     } }                      // apply strength

// Defuzzification

void defuzzify (Sysio * start)
{   Sysio * var ;                                                          // pointer to system output
    Mfunc * M ;                                                  // pointer to output membership function
    sysval area, areas, products ;
    for (var = start ; var ; var = var->next)                  // compute defuzzified value for each system output
    {   products = areas = 0 ;
        for (M = var->membership ; M ; M = M->next)
        {   area = trapezoid (M) ;
            products += area * ((M->point2 - M->point1) / 2) ;          // sum of products of area by centroid
            areas += area ; }                                            // sum of clipped trapezoid areas
        if (! areas) no_match (sysin->value, (sysin->next)->value) ;          // avoid zero divide error
        else var->value = products / areas ; } }                                  // weighted average

/*  trapezoid — compute area of trapezoid
•   each inference pass produces a new set of output strengths which affect the areas of trapezoidal membership
    functions used in center-of-gravity defuzzification
•   area values must be recalculated with each pass
•   area of trapezoid is h * (a + b) / 2
    where h = height = output-strength = M->value
          b = base = M->point2 - M->point1
          a = top, must be derived from h, b, and slopes1 and 2 */

sysval trapezoid (Mfunc * M)
{   return (M->value * (2 * (M->point2 - M->point1) - ((M->value / M->slope1) + (M->value / M->slope2))) / 2) ; }

void setup (char ** datafile)
{   tally = 10 ;                                                                          // • • • debug trick
    sysin = setup_fuzzy_sets (datafile[0]) ;                                          // system inputs & ouput
    tally = 20 ;                                                                          // • • • debug trick
    sysin->next = setup_fuzzy_sets (datafile[1]) ;
    tally = 30 ;                                                                          // • • • debug trick
    sysout = setup_fuzzy_sets (datafile[2]) ;
    tally = 0 ;                                                          // • • • debug trick : reset counter
    rule_base = setup_rules (datafile[3]) ; }                                                  // rule data
```

```
Sysio * setup_fuzzy_sets (char * fname)
{   FILE * data ;
    Sysio * var ;
    Mfunc * M, * previous = NULL ;                  // ptr to current & previous membership fun. data structures
    char symbol [MaxName] ;
    distance a, b, c, d, e ;
    int r ;
    if ( (data = fopen (fname, "r")) == NULL) error_opening_file (fname) ;
    fscanf (data, "%s", symbol) ;
    var = (Sysio *) malloc (sizeof (Sysio)) ;
    strcpy (var->symb, symbol) ;
    var->value = tally ;                                                    // • • • debug trick
    var->membership = NULL ;
    var->next = NULL ;
    while ((r = fscanf (data, "%s %d %d %d %d", symbol, &a, &b, &c, &d)) != EOF)   // assume integer type
    {   if (r < 5) error_reading_file (fname, symbol) ;
        M = (Mfunc *) malloc (sizeof (Mfunc)) ;
        strcpy (M->symb, symbol) ;
        if (! var->membership) var->membership = M;                         // 1st time around
        if (previous) previous->next = M ;                                  // next time around, link it
        previous = M, previous->next = NULL ;                               // 1st time around, clean link
        M->value = tally++ ;                                                // • • • debug trick
        M->point1 = a ; M->point2 = d ;
        if ((e = b - a) > 0) M->slope1 = MaxVal / e ; else error_reading_file (fname, symbol) ;   // slope 1
        if ((e = d - c) > 0) M->slope2 = MaxVal / e ; else error_reading_file (fname, symbol) ; }  // slope 2
    fclose (data) ;
    return (var) ; }

Rule * setup_rules (char * fname)
{   FILE * data ;
    Rule * rule, * top, * previous = NULL ;
    Mfunc * fun ;
    Elt * side ;
    char sym1 [MaxName], sym2 [MaxName], sym3 [MaxName] ;
    int r ;
    if ((data = fopen (fname, "r")) == NULL) error_opening_file (fname) ;
    while ((r = fscanf (data, "%s %s %s", sym1, sym2, sym3)) != EOF)
    {   if (r < 3) error_reading_file (fname, sym1) ;
        rule = (Rule *) malloc (sizeof (Rule)) ;
        if (! previous) top = rule ;                                        // first time around
        else previous->next = rule ;                                        // link previous rule to new one
        rule->next = NULL ;                                                 // clean link
        previous = rule ;                                                   // remember
        fun = sysin->membership ;                                           // sys input 1
        rule->if_side = setup_clause (fun, sym1) ;                          // install antecedent 1
        fun = (sysin->next)->membership ;                                   // sys input 2
         (rule->if_side)->next = setup_clause (fun, sym2) ;                 // install antecedent 2
        fun = sysout->membership ;                                          // sys output 1
        rule->th_side = setup_clause (fun, sym3) ; }                        // install consequent 1
    fclose (data) ;
    return (top) ; }

// setup_clause — allocate new clause and install pointer to value already in Mfunc structure

Elt * setup_clause (Mfunc * start, char * fuzzy_val)
{   Mfunc * M ;
    Elt * E = NULL ;
    for (M = start ; M ; M = M->next)
    {   if (strcmp (M->symb, fuzzy_val) == 0)                               // this is the one
        {   E = (Elt *) malloc (sizeof (Elt)) ;
            if (! E) malloc_error (fuzzy_val) ;
            strcpy (E->symb, fuzzy_val) ;                                   // remember symbol
            E->value = & M->value ;                                         // pointer to sysval
            E->next = NULL ;                                                // clean link
            break ; } }                                                     // only one will match
    return (E) ; }

void error_opening_file (char * s) { printf ("\nError opening %s\n", s) ; exit (1) ; }
void malloc_error (char * s) { printf ("\nError allocating %s\n", s) ; exit (2) ; }
void error_reading_file (char * s1, char * s2) { printf ("\nError reading %s, value %s\n", s1, s2) ; exit (3) ; }

void no_match (sysval v1, sysval v2)
{   printf ("\ndefuzzify: no rule for %d & %d\n", v1, v2) ; exit (4) ; }     // assume integer type

void dump (sysval angle, sysval velocity)
{   fputs ("• • • Membership Functions • • •\n\n", display) ;
    dump_mfuncs (sysin) ;
    dump_mfuncs (sysin->next) ;
    dump_mfuncs (sysout) ;
    fputs ("• • • Rule Base • • •\n", display) ;
    dump_rules (rule_base) ;
    fprintf (display, "\n\nAngle = %d — velocity = %d", angle, velocity) ;   // assume integer type
    fputs ("\n• • • End of Data • • •\n", display) ; }
```

```
void dump_mfuncs (Sysio * var)
{   Mfunc * M ;
    fprintf (display, "%s : %d\n", var->symb, var->value) ;                          // assume integer type
    for (M = var->membership ; M ; M = M->next)
        fprintf (display, "%s : %d [%d %d]\n", M->symb, M->value, M->point1, M->point2) ;   // assume integer type
    fputc ('\n', display) ; }

void dump_rules (Rule * start)
{   Rule * rule ;
    Elt * E ;
    int Rno = 1 ;
    for (rule = start ; rule ; rule = rule->next)
    {   fprintf (display, "\nRule %2d", Rno++) ;
        for (E = rule->if_side ; E ; E = E->next)
            fprintf (display, ", %s = %3d ", E->symb, * (E->value)) ;                // assume integer type
        for (E = rule->th_side ; E ; E = E->next)
            fprintf (display, "-> %s = %3d ", E->symb, * (E->value)) ; } }           // assume integer type
```

Index ???

Table des matières ???