Logique Floue

Cet exposé a été écrit parce que je n'arrivais pas trouver une documentation sérieuse sur la logique floue. Ce n'est pas un ouvrage de référence. Voici trois adresses de sites contenant des informations plus sérieuses sur la logique floue, communiquées par un aimable lecteur :

http://www.abo.fi/~rfuller/nfs.htmlc
http://perso.club-internet.fr/bmantel
http://perso.wanadoo.fr/lehoux

Introduction

La logique floue a été formulée par A. Zadeh dans le milieu des années 60. Elle sert à pouvoir programmer un ordinateur pour qu'il contrôle une machine un peu comme le ferait un être humain.

A première vue, la logique floue est une généralisation de la logique booléenne classique. Elle ajoute cependant une fonctionnalité déterminante : la possibilité de calculer un paramètre en disant simplement dans quelle mesure il doit se trouver dans telle ou telle zone de valeur.

Afin de permettre au lecteur informaticien de s'essayer à la logique floue, ce livre propose en annexe un "noyau" de logique floue en BASIC, et des explications quant à son usage.

Le lecteur non-informaticien aura peut-être intérêt à ne lire les chapitres 1 et 2 que pour se faire une impression globale des mécanismes en jeu.

1. Un exemple simple : un thermostat

Supposons qu'on veuille utiliser un ordinateur programmé en logique floue pour contrôler un système de chauffage. Le chauffage d'une cuve d'incubation, par exemple.

A quoi pourrait ressembler le programme en logique floue que l'on va faire exécuter par l'ordinateur ? Détaillons les parties de ce programme, avant de l'écrire en entier :

Déclaration des entrée/sorties

L'ordinateur peut connaître une donnée : la température de la cuve.

L'ordinateur peut imposer une chose : l'ampèrage du courant qui passe dans la résistance chauffant la cuve.

Le programme commence donc par les lignes suivantes :

```
temperature = PARAMETRE (0,100,60)
amperage = PARAMETRE (0,5,0)
```

(0,100,60) veut dire que le paramètre temperature est normalement compris entre 0 C° et 100 C° (facultatif), et que si on a pas pu mesurer ladite température, il faut considérer qu'elle vaut 60 C° (inutilisé).

(0,5,0) veut dire que le parametre amperage est normalement compris entre 0 et 5 ampères (facultatif), et que si l'ordinateur n'a pas pu calculer correctement un ampèrage, il doit ordonner de sortir 0 ampères (sécurité).

Déclaration des zones

Il faut définir des zones de température, et des zones d'ampèrage. Par exemple en ajoutant les lignes suivantes au programme :

```
tropfroid = ZONE (temperature, -273, -273, 33, 36)
tiede = ZONE (temperature, 33, 36, 38, 41)
tropchaud = ZONE (temperature, 38, 41, 200, 200)
courantnul = ZONE (amperage, -1, 0, 0, 1)
courantfaible = ZONE (amperage, 0, 1, 1, 2)
courantfort = ZONE (amperage, 4, 5, 5, 6)
```

Dans le cas de "tiede", (temperature, 33, 36, 38, 41) veut dire que "tiede" est une zone de température, qu'elle commence à 33 C°, qu'elle est pleinement réalisée entre 36 C° et 38 C°, et qu'elle cesse à partir de 41 C°.

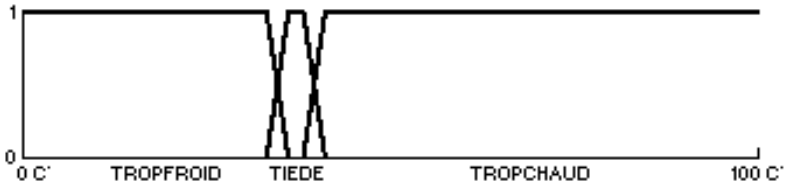

Fig. 1 Le graphe des zones de température déclarées

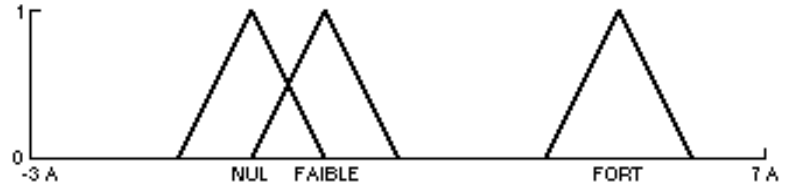

Fig. 2 Le graphe des zones de courant déclarées

Effectuer la mesure de la température

Ca, ce n'est pas un problème de logique floue, mais un problème d'électronique. Si un thermomètre est connecté à l'ordinateur par l'intermédiaire d'une carte I/O, et que les routines nécessaires ont été ajoutées au QBASIC, la ligne de programme servant à faire la mesure pourrait être la suivante :

```
t = MESUREDETEMPERATURE
```

Quand il exécutera cet ordre, l'ordinateur mettra dans la variable nommée "t" le nombre calculé par la fonction MESUREDETEMPERATURE. Par exemple le nombre 40, si le thermomètre a mesuré 40 C°.

Donner les règles d'inférence

C'est à dire expliquer à l'ordinateur les lois ou règles qu'il doit suivre. Par exemple avec les lignes suivantes :

```
SORTIR courantfaible, SI(t, tiede)
SORTIR courantnul, SI(t, tropchaud)
SORTIR courantfort, SI(t, tropfroid)
```

Que fera l'ordinateur lorsqu'il exécutera ces trois lignes ? Cela est illustré graphiquement, pour t = 40 C°, dans les figures 3, 4, 5, 6, 7 et 8. Le "résultat" est montré par la figure 9.

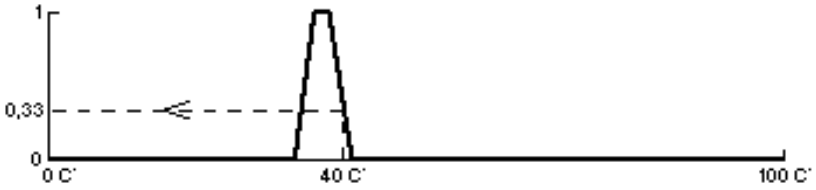

Fig. 3 SI(40, tiede) donne 0.3333. En d'autre termes : "il est vrai à 33%" que 40 C° est dans la zone de température "tiede"

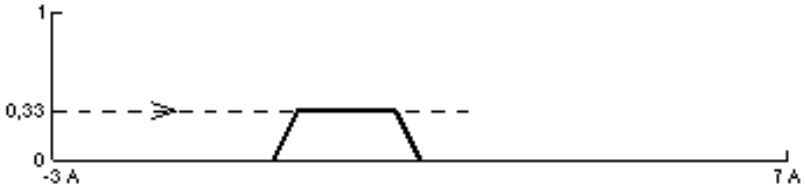

Fig. 4 SORTIR courantfaible, 0.3333 "décapite" la zone courantfaible à une hauteur de 0.3333. En d'autres termes : on impose qu'il faudra tendre "à 33%" à placer le paramètre amperage dans la zone "courantfaible"

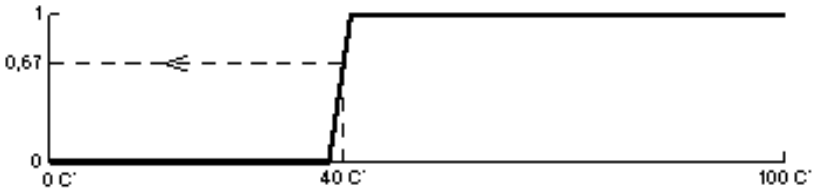

Fig. 5 SI(40, tropchaud) donne 0.6667. En d'autres termes : "il est vrai à 67%" que 40 C° est dans la zone de température "tropchaud"

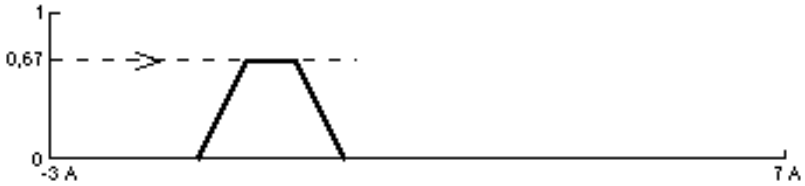

Fig. 6 SORTIR courantnul, 0.6667 "décapite" la zone courantnul à une hauteur de 0.6667. En d'autres termes : on impose qu'il faudra tendre "à 67%" à placer le paramètre amperage dans la zone "courantfaible"

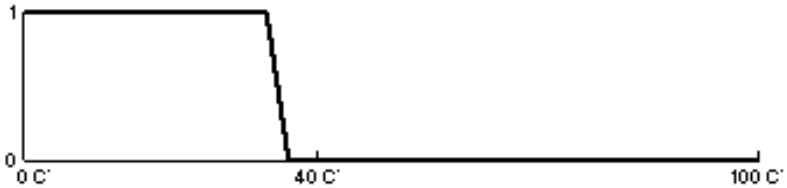

Fig. 7 SI(40, tropfroid) donne 0. En d'autres termes : "il est vrai à 0%" que 40 C° est dans la zone de température "tropfroid"

Fig. 8 SORTIR courantfort, 0 "décapite" complètement la zone courantfort. En d'autres termes : on impose qu'il faudra tendre "à 0%" à placer le paramètre amperage dans la zone "courantfaible"

De façon synthétique, on peut dire que la fonction SI sert à soupeser dans quelle mesure t se trouve dans quelles zones de température, et l'ordre SORTIR sert à imposer dans quelle mesure l'ampèrage devra se trouver dans quelles zones d'intensité de courant. (Ou entre quelles zones.)

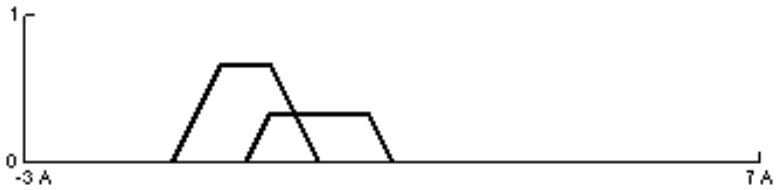

Fig. 9 L'ensemble des zones d'intensité de courant, décapitées. C'est en quelque sorte ce que l'ordinateur a en mémoire après avoir évalué les règles d'inférence

Calculer le résultat

Une fois qu'il a passé en revue toutes les règles d'inférence, on peut demander à l'ordinateur de calculer l'ampèrage. Par exemple avec la ligne suivante :

```
a = CALCUL(amperage)
```

Que fait l'ordinateur lorsqu'il exécute la fonction CALCUL ? Graphiquement, cela revient à estimer où se trouve le centre de masse de l'union des zones d'ampèrage décapitées (fig. 10). Dans notre cas, cela donne 0.38 ampères.

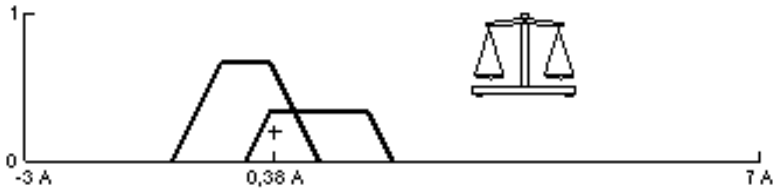

Fig. 10 Estimation de la valeur du paramètre amperage

Ainsi, le nombre 0.38 sera mis dans la variable nommée "a". Insistons sur le fait que 0.38 est le résultat du fait que la variable t contenait le nombre 40. Pour d'autres valeurs de t, les règles d'inférence feront engendrer d'autres valeurs de a (c'est leur rôle).

Utiliser le résultat

De nouveau, ce n'est pas un problème de logique floue, mais d'électronique. La ligne servant à imposer l'ampèrage dans la résistance de chauffe, par l'intermédiaire d'une carte I/O dûment implémentée, pourrait être la suivante :

```
AMPERAGE a
```

Comme, dans notre exemple, la mémoire nommée "a" contient le nombre 0.38, l'exécution de cet ordre aura pour résultat de faire passer un courant de 0.38 A dans la résistance.

Synthèse

Le programme de contrôle de la température de cuve est le suivant :

```
temperature = PARAMETRE (0,100,200)
amperage = PARAMETRE (0,5,0)
tropfroid = ZONE (temperature, -273, -273, 33, 36)
tiede = ZONE (temperature, 33, 36, 38, 41)
tropchaud = ZONE (temperature, 38, 41, 1000, 1000)
courantnul = ZONE (amperage, -1, 0, 0, 1)
courantfaible = ZONE (amperage, 0, 1, 1, 2)
courantfort = ZONE (amperage, 4, 5, 5, 6)
DO
   t = MESUREDETEMPERATURE
   SORTIR courantfaible, SI(t, tiede)
   SORTIR courantnul, SI(t, tropchaud)
   SORTIR courantfort, SI(t, tropfroid)
   a = CALCUL(amperage)
   AMPERAGE a
LOOP
```

Entre le DO et le LOOP se trouvent les lignes qui doivent êtres exécutées sans arrêt pour que le contrôle de la température soit effectué en permanence. Ainsi, l'ordinateur va lire régulièrement la température de la cuve, et décider de l'ampèrage à injecter dans la résistance chauffante. Comme dans notre exemple il n'y a aucune forme d'intégration ou de dérivation par rapport au temps, il correspond exactement une valeur de courant à chaque valeur de température (fig. 11).

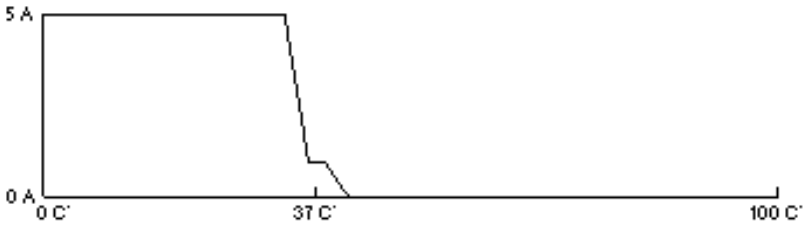

Fig. 11 Le graphe du courant en fonction de la température

2. Toutes les possibilités

L'exemple du thermostat est minimal. En pratique, on peut faire beaucoup de choses :

- Avoir de nombreux paramètres en entrée : pas seulement une simple mesure de température, mais des mesures de vitesse, position, force... Certains paramètres peuvent ne pas êtres directement des mesures de capteurs : l'accélération, par exemple, peut être un paramètre calculé à l'aide des mesures de vitesse. L'erreur sur la position est un paramètre calculé en soustrayant la position mesurée de la position souhaitée.
- Avoir de nombreux paramètres en sortie : pas seulement imposer un ampèrage dans une résistance électrique, mais commander plusieurs moteurs, des lampes, des appareils divers...
- Utiliser des paramètres comme résultats intermédiaires. Ils ne sont ni entrée ni sortie, mais sont calculés pour servir de base aux règles d'inférence suivantes.

- Appliquer des règles d'inférence différentes dans des phases différentes. Ou modifier les formes des zones.
- Utiliser des fonctions logiques floues, extensions des fonctions booléennes NOT, AND, OR... (NON, ET, OU en français.)
- "Astucer", dans la grande tradition de l'informatique (quand la prudence le permet, et la nécessité l'impose). Voir à cet effet, dans l'annexe 2, les ordres et fonctions NETTOYAGEGENERAL, CALCULT, NETTOYAGE.

(Il aurait été souhaitable que ce texte contienne au moins un exemple d'application "imposante" de logique floue : le maintien en équilibre d'un balancier inversé, le contrôle d'un objet en lévitation magnétique active, l'autoguidage d'un missile...)

3. Rappel sur la logique classique

La logique classique, ou logique de Boole (mathématicien britannique), originaire des Anciens Grecs, domine actuellement l'informatique. Avant de parler logique floue, reparlons brièvement de cette logique booléenne classique.

Qu'est-ce qu'un état logique classique ?

En logique classique, il y a deux états logiques : faux et vrai. A quoi servent ils ? En gros, à faire le même genre de choses qu'on fait avec les nombres. Exemple :

Quelle est le nombre donnant "la hauteur en mètres de la tour Eiffel" ? Réponse : 300

Quel est l'état logique de l'affirmation "la hauteur de la tour Eiffel est plus grande que la hauteur de l'Everest" ? Réponse : faux

Les états logiques nous servent à énoncer les propriétés de ce qui nous entoure. On n'utilise pas nécessairement toujours les mots "faux" et "vrai". Suivant les circonstances, et l'usage qu'on en fait, on utilise les expressions "0" et "1", "off" et "on", "ouvert" et "fermé", "non" et "oui"...

Comme les ordinateurs ne travaillent souvent qu'avec des nombres, en informatique on a adopté la convention d'utiliser 0 et 1 pour dire faux et vrai.

En d'autres termes : Imaginons que je vous dise que j'ai stocké, dans la mémoire nommée "memo" d'un ordinateur, un état logique disant si vous avez gagné au loto. Si quand vous donnez à l'ordinateur l'ordre suivant :

```
PRINT memo
```

le chiffre 1 apparaît à l'écran, c'est que vous avez gagné au loto...

(Sur certains systèmes informatiques, 0 veut dire faux, et -1 veut dire vrai (QBASIC), ou -1 veut dire faux, et 1 veut dire vrai, ou encore, 1 veut dire faux, et -1 veut dire vrai. Parfois, dans un même système, on inverse par-ci par-là la valeur de faux et de vrai. Peu importe. Dans notre cas, posons que 0 veut dire faux, et 1 veut dire vrai.)

(Note sans rapport direct avec le sujet : on peut aussi utiliser des états logiques pour représenter des nombres. Par exemple :

fauxfauxfauxfaux	vaut	0
fauxfauxfauxvrai	vaut	1
fauxfauxvraifaux	vaut	2
fauxfauxvraivrai	vaut	3
fauxvraifauxfaux	vaut	4
fauxvraifauxvrai	vaut	5
fauxvraivraifaux	vaut	6
fauxvraivraivrai	vaut	7
vraifauxfauxfaux	vaut	-8
vraivraivraivrai	vaut	-1
vraivraivraifaux	vaut	-2...

(Remplacez faux et vrai par 0 et 1, et vous aurez simplement 0, 1, 2, 3, 4, 5, 6, 7, -8, -1 et -2 en binaire signé. C'est ainsi que les ordinateurs travaillent, tout au fond de leurs entrailles.))

Qu'est-ce qui engendre des états logiques classiques ?

Diverses choses :

- Des capteurs : porte ouverte ou fermée, lampe allumée ou éteinte, détecteur d'incendie ayant détecté ou non de la fumée... Par exemple, si "le capteur numéro 3" est un détecteur de fumée, l'exécution par l'ordinateur de l'ordre :

```
e = CAPTEUR(3)
```

  mettra 1 dans la variable e si de la fumée est détectée par le capteur. Sinon, 0.

- Le contenu de tables de vérité, ou des constantes. Par exemple, si le tableau nommé "sol" indique la présence d'objets sur un sol virtuel, alors l'exécution de l'ordre :

```
e = sol(35, 12)
```

  mettra 1 dans la variable e s'il y a un objet àl'endroit dont les coordonnées sont 35, 12. Sinon, 0.

- Le résultat de comparaisons : x = y, x < y, x >= y...

  Exemples :

`4 > 2`	vaut 1 parce-qu'il est vrai que 4 est plus grand que 2
`3 = 6`	vaut 0 parce-qu'il est faux que 3 est égal à 6
`1 <= 9`	vaut 1 parce-qu'il est vrai que 1 est plus petit ou égal à 9
`b > 8`	vaut 1 si la variable b contient un nombre supérieur à 8, et vaut 0 si b est inférieur ou égal à 8
`PRINT 6 / 3`	fera afficher le nombre 2 à l'écran
`PRINT 3 > 9`	fera afficher l'état logique 0 à l'écran
`som = 3 + 9`	mettra le nombre 12 dans la variable nommée "som"
`na = 3 < 7`	mettra l'état logique 1 dans la variable na

>, <, >=, <=, =, <> (pas égal) sont des opérateurs au même titre que les quatre opérations +, -, * (fois), /(divisé).

2 + 3 donne 5, et 9 < 3 donne 0. La différence, c'est que les quatre opérations fournissent un résultat qui est un nombre réel, tandis que les opérateurs de comparaison fournissent un résultat qui est un état logique (0 ou 1).

Qu'est-ce qu'une fonction logique classique ?

Les fonctions logiques généralement disponibles sur les ordinateurs sont ET, OU et NON (AND, OR et NOT en anglais). Elles fonctionnent comme ceci :

soit a et b des variables dans l'état 0 ou 1.

a ET b	donne 1 si a=1 et b=1, sinon 0
a OU b	donne 1 si a=1 ou b=1, et 0 si a=0 et b=0
NON a	donne 1 si a=0, et 0 si a=1

Exemples :

1 ET 1	donne 1
1 ET 0	donne 0
1 OU 0	donne 1
NON 1	donne 0

5<1 OU 6=6 donne 1

Les fonctions logiques ont des niveaux de priorité : dans des expressions comme NON a ET b OU c, l'ordinateur évalue d'abord le NON, puis le ET, puis le OU. Pour imposer l'ordre des évaluations, on peut utiliser des parenthèses, comme pour les opérations arithmétiques : NON(a ET (b OU c)).

Les fonctions ET et OU sont des opérateurs au même titre que les quatre opérations :

3 - 1 donne 2 1 ET 0 donne 0

La différence est que les fonctions logiques s'appliquent à des états logiques, et fournissent des états logiques.

L'opérateur NON est un opérateur au même titre que l'opérateur d'inversion - :

- 7 donne -7 NON 0 donne 1

(Comme les états logiques sont souvent représentés par des nombres, généralement 0 et 1, des calculs hybrides entre fonctions arithmétiques et logiques sont possibles. Du genre de :)

```
r = (b < 7 AND (1 + k)) * h
```

(Les fonctions logiques peuvent aussi souvent s'appliquer à des nombres réels, mais le résultats de ces calculs est très variable d'un système à l'autre. 8 ET 3 donne 8 sur un Spectrum, 0 sur un BBC ou en QBASIC, et 1 avec les compilateurs C ANSI.)

(Ces astuces sont bien entendu fortement déconseillées pour des programmes "sérieux". On doit appliquer les opérateurs logiques uniquement à des paramètres logiques, et les opérateurs arithmétiques uniquement à des nombres entiers ou réels. Les états logiques ne peuvent êtres utilisés que par les ordres prévus pour (IF, WHILE, UNTIL...). Le passage de nombres entiers ou réels vers des états logiques se fait uniquement à l'aide des fonctions prévues à cet effet (<, >, <=, =...).)

A quoi peut servir un état logique classique ?

- Un état logique peut être mémorisé. Vous pouvez par exemple écrire la ligne suivante en BASIC :

```
e = 9>x
```

  Après l'exécution de cette ligne, la variable e contiendra 1 si 9>x, et 0 si 9<=x. On peut aussi écrire des choses comme :

```
e = 9>x AND x>2
e = 1
```

- Un état logique peut être imposé à un objet. L'ordre suivant pourrait par exemple servir à mettre en route un moteur :

```
ETATDUMOTEUR 1
```

Celui-ci va l'éteindre :

```
ETATDUMOTEUR 0
```

L'ordre suivant va éteindre ou mettre en route un moteur, suivant que e contient 0 ou 1 :

```
ETATDUMOTEUR e
```

On peut donner àl'ordinateur les ordres suivants, bien que ce soit un peu original :

```
ETATDUMOTEUR 9>x
ETATDUMOTEUR CAPTEUR(3)
```

- Un état logique peut servir à décider s'il faut faire quelque chose ou non. Nottament avec la structure IF ... THEN ... (si ... alors ...). Par exemple :

```
IF x<7 THEN y = x+5
```

(Si la variable x contient un nombre plus petit que 7, alors calcule combien fait x+5, et met le résultat dans la variable y.)

```
IF x>=9 THEN PRINT "OK"
```

(Si la variable x contient un nombre plus grand ou égal à9, alors affiche OK àl'écran.)

```
IF e THEN PRINT "test"
```

(Si la variable e contient 1, alors affiche "test" àl'écran.)

```
IF NOT e OR x<=y THEN ALLUMERMOTEUR
```

(Si e contient 0, ou si x est plus petit ou égal ày, alors effectue l'ordre ALLUMERMOTEUR.)

```
IF x>9 THEN
   ETATDUMOTEUR 1
ELSE
   ETATDUMOTEUR 0
END IF
```

(Plus propre qu'un exemple précédent. ELSE veut dire "sinon".)

```
x=0
DO
   PRINT x
   x=x+1
LOOP UNTIL x>9
```

(Affichera les nombres de 0 à9 àl'écran. DO veut dire "faire", LOOP veut dire "boucle", et UNTIL veut dire "jusqu'àce que".) (Note : pour arrêter un programme en QBASIC (quand il ne s'arrête pas tout seul), il faut garder la touche Ctrl enfoncée, taper la touche Break/Pause, relacher la touche Ctrl, et parfois encore taper un coup sur Enter.)

```
CLS
FOR ligne = 1 TO 22
   FOR colonne = 1 TO 80
      x = ((colonne - 1) / 79) * 3.1416 *
4
      y = ((22 - ligne) / 21) * 3.1416 * 2
      IF COS(x) > SIN(y) THEN
         LOCATE ligne, colonne
         PRINT "X"
      END IF
   NEXT colonne
NEXT ligne
```

(Dessinera grossièrement le domaine de l'équation cos(x) >sin(y).) Si vous voyez un jour des lignes comme celles-ci dans un programme, vous pouvez les effacer sans craintes :

```
IF 2=3 THEN ALLUMERMOTEUR
IF 0 THEN x = 67
```

4. La logique floue

Remarquant qu'un ordinateur programmé "normalement" avait généralement des réactions moins habiles que celles d'un être humain, A. Zadeh a contribué à résoudre ce problème en imaginant une généralisation de la logique classique. Ce nouveau langage, la logique floue, est conçu pour réaliser des programmes informatiques, ou des circuits électroniques, ayant un comportement plus proche des réflexes humains.

Qu'est-ce qu'un état logique en logique floue ?

Un état logique flou est une "amplitude" entre 0 et 1. En d'autres termes : un nombre réel dans l'interval [0,1]. Par exemple : 0, 0.1, 0.25, 0.4563, 0.99, 1...

On pourrait dire les choses ainsi :

0	veut dire "complètement faux"
0.2	veut dire "un peu vrai"
0.5	veut dire "à moitié vrai"
0.9	veut dire "presque tout à fait vrai"
1	veut dire "tout à fait vrai"

Qu'est-ce qui engendre des états logiques flous ?

- Le contenu de tables de vérité, ou des constantes.
- Des capteurs : porte plus ou moins ouverte, lampe plus ou moins allumée... Par exemple, si "le capteur 3" est un détecteur de fumée :

```
e = CAPTEURFLOU(3)
```

  mettra un nombre entre 0 et 1 dans la variable e, suivant la quantitéde fumée détectée. Par exemple 0 quand il n'y a pas de fumée, 1 pour toute fumée dense, et entre 0 et 1 pour une fumée plus ou moins légère. (Ou encore : une arctangente de la quantitéde fumée...)

- Le résultat d'un test si une valeur se trouve dans une zone donnée. (Voir fig. 1 et 3.) Par exemple, en posant qu'au début du programme il y ait eu la définition de zone suivante :

```
tiede = ZONE(31, 35, 37, 41)
```

  alors l'évaluation de la fonction SI(mesure, tiede) donnera :

0	si mesure vaut 22
0	si mesure vaut 31
0.25	si mesure vaut 32
0.5	si mesure vaut 33
0.75	si mesure vaut 34
1	si mesure vaut 35
1	si mesure vaut 36
1	si mesure vaut 37

0.5	si mesure vaut 39
0.3333	si mesure vaut 40
0	si mesure vaut 41
0	si mesure vaut 42

De même, SIPP(mesure, tiede) donnera une valeur entre 0 et 1, chiffrant dans quelle mesure le paramètre mesure est Plus Petit que les valeurs de la zone tiede :

1	si mesure vaut 22
1	si mesure vaut 25
1	si mesure vaut 31
0.75	si mesure vaut 32
0.5	si mesure vaut 33
0.25	si mesure vaut 34
0	si mesure vaut 35
0	si mesure vaut 36
0	si mesure vaut 37
0	si mesure vaut 39
0	si mesure vaut 41
0	si mesure vaut 42

Il existe également SIPG (Plus Grand), SIPPOE (Plus Petit Ou Egal), SIPGOE (Plus Grand Ou Egal) et SIPE (Pas Egal).

Les fonctions logiques floues

Les fonctions logiques floues sont les mêmes que les fonctions classiques : ET, OU et NON. Elles donnent les résultats suivants :

Soit a et b des états logiques flous (des nombres réels dans l'interval [0,1]).

a ET b	donne a si a<b, et b si b<a ("le plus petit des deux")
a OU b	donne a si a>b, et b si b>a ("le plus grand des deux")
NON a	donne 1-a ("une sorte d'inverse de a")

Exemples :

1 ET 0.3	donne 0.3
0.67 ET 0	donne 0
0.12 OU 0.98	donne 0.98
NON 0.54	donne 0.46

Notez que si l'on contraint a et b à être 0 ou 1, on retombe dans la logique booléenne classique.

Dans le cadre du noyau en QBASIC, la syntaxe des fonctions logiques a été modifiée :

a ET b	devient ET (a, b)
a OU b	devient OU (a, b)
NON a	devient NON (a)

(Les lecteurs n'aimant pas le Lisp devront faire preuve d'indulgence. C'était encore la façon la plus simple, rapide et universelle de procéder. Surtout dans le cas d'une implémentation en C, un préprocesseur pourrait arranger les choses. L'idéal sont les possibilités du C++.)

Exemples :

```
SORTIR courantfort, ET(SI(t1, froid), SI(t2, pasdanger))
SORTIR courantfaible, OU(SI(t, froid), SI(t, gele))
SORTIR courantfort, ET(SI(t1, froid), NON(SI(t2, danger)))
IF VRAI(ET(SI(t1, froid), SI(t2, froid))) THEN PRINT "Froid !"
```

(Un ami m'a signalé une autre façon de réaliser les opérations en logique floue :

a ET b	est le résultat du calcul a x b
a OU b	est le résultat du calcul a + b - a x b

Cette autre approche a l'inconvénient de nécessiter des capacités de calcul plus importantes et les immenses avantages d'être plus fluide et de maintenir l'information au fil des calculs (il est possible de retrouver exactement a si on connaît b et c et qu'on sait que a OU b = c)).

A quoi peut servir une amplitude logique floue ?

- Un état logique flou peut être mémorisé. Vous pouvez par exemple écrire la ligne suivante en BASIC :

```
e = SI(t1, froid)
```

Après l'exécution de cette ligne, la variable e contiendra un chiffre entre 0 et 1, suivant que t1 se trouve plus ou moins dans la zone "froid".

- Un état logique peut être imposé à un objet. L'ordre suivant pourrait par exemple servir à donner à un moteur une vitesse entre l'arrêt complet et le maximum, suivant le nombre entre 0 et 1 que contient e :

```
ETATDUMOTEUR e
```

Ou, bien sûr :

```
ETATDUMOTEUR SI(t, froid)
```

L'ordre suivant affichera un nombre àl'écran, entre 0 et 1, chiffrant dans quelle mesure le contenu de la variable t1 est plus ou moins dans la zone "froid" :

```
PRINT SI(t1, froid)
```

- Un état logique peut servir à décider s'il faut faire quelque chose ou non. Par exemple :

```
IF VRAI(SI(t, froid)) THEN PRINT "Sortez vos manteaux"
IF VRAI(OU(SI(t1, froid), SI(t2, froid))) THEN PRINT "Froid"
IF SI(t1, froid) > SI(t2, froid) THEN PRINT "Cuve 1 plus froide"
```

- Un état logique peut servir à dire dans quelle mesure un paramètre doit se trouver dans telle ou telle zone de valeur. Cela se fait en deux temps :

  Primo, avec l'ordre SORTIR, il sert à dire dans quelle mesure un paramètre devra se trouver dans l'une ou l'autre zone de valeurs. (Voir le chapitre 1, et fig. 4, 6, 8 et 9.)

  Secundo, la fonction CALCUL fait la synthèse de toute l'information donnée par les ordres SORTIR. Il fait en quelque sorte "la moyenne pondérée". Il "soupèse" dans quelle mesure les ordres SORTIR ont préconisés que le paramètre soit dans telle ou telle zone de valeur, et décide quelle valeur finale, précise, aura le paramètre. (Voir le chapitre 1, et fig. 10.)

5. Commentaires

La logique floue a les avantages de ses inconvénients.

D'un coté elle n'offre pas la rigueur mathématique des systèmes de contrôle comme les PID (Proportionnel, Intégral,

Dérivé : méthode générale pour la régulation de moteurs, systèmes de positionnement, haut-parleurs...). Imaginez vous un système flou intervenant directement sur un signal HI-FI ?

D'un autre coté, quand de toute façon cette rigueur est inutile, voire illusoire, la logique floue offre une immense souplesse, des possibilités de raffinement extrêmes, et une programmation beaucoup plus instinctive. Imaginons un système sans arrêt ballotté par des rafales de vent; a quoi bon faire des calculs mathématiques subtils pour essayer de le contrôler ?

(Il existe des théories "classiques" qui gèrent les phénomènes aléatoires, ainsi que les imprécisions de mesure, beaucoup plus sérieusement qu'en logique floue. Elles proposent des calculs très rigoureux. Suivant les circonstances, cette rigueur sera plus puissante que la logique floue, ou sera une complication inutile.)

La logique floue a beau avoir une apparence "humaine", son utilisation demandera quand même un certain flair, et de l'expérience.

La logique floue n'est pas cantonnée à l'automatisation. Elle est un formalisme utile pour les systèmes experts, l'informatique en général, la modélisation de systèmes biologiques et humains... Les OS et les programmes de bureautique seraient plus souples si leur programmation faisait appel à la logique floue.

Un des apports principaux de la logique floue sera peut-être de permettre aux particuliers de programmer eux-même le comportement général des objets qui les entourent. Un enfant éveillé est capable de changer la programmation en logique floue d'une voiture téléguidée.

Un détail pratique : Dans le cadre du noyau de logique floue en BASIC fourni dans ce livre, on peut mélanger des opérations en logique classique et des opérations en logique floue: grâce aux fonctions de traduction FUZZY et VRAI (Voir l'Annexe 2). (Avec un système conçu d'origine pour la logique floue, cette traduction d'un système à l'autre se ferait automatiquement.)

6. Perspectives

On peut broder à l'infini autour du concept de logique floue. En matière d'automatisation, quelques voies sont à distinguer :

- On peut observer un humain en train de commander une machine, pour en déduire les règles d'inférence. Ensuite, on implémente ces règles dans un ordinateur, en logique floue.
- Grâce à des capteurs, on peut aussi laisser un ordinateur "observer" un humain en train de commander une machine, pour qu'il en dégage lui-même les règles d'inférence. Des réseaux de neurones flous seraient fort utiles à cet effet.
- Enfin, On peut mettre un ordinateur aux commandes d'une machine, et le laisser trouver tout seul les règles d'inférence. L'ordinateur sait seulement qu'il doit chercher à maximiser des paramètres comme "rendement", et minimiser des paramètres comme "usure". Ici, les réseaux de neurones sont indispensables. Le principal avantage de cette façon de procéder est que si la machine change ultérieurement de façon de travailler (petite panne, usure, changement d'environnement...) l'ordinateur pourra s'y adapter tout seul. (Systèmes de Brooks, robots prévus pour les missions sur Mars...)

Annexe 1
Le noyau de logique floue en BASIC

Ce noyau est une "version minimale". Il ne procède pas à des vérifications soignées, ne permet pas "d'effacer" des paramètres et zones après les avoir déclarés, ne permet pas d'imposer des amplitudes de zones dans un contexte récursif, n'alloue pas un nombre illimité de zones et paramètres, est limité dans la possibilité de consulter ce qui a été déclaré, impose que toutes les zones aient un nom différent, ne fournit pas d'opérations mathématiques sur et entre les zones... Accessoirement, la convention un peu douteuse a été prise comme-quoi les ordres et fonctions propres à la logique floue sont en français. Le lecteur passant du noyau donné ici à un système officiel, devra s'attendre à des changements de forme.

Le noyau a été écrit en QBASIC (® Microsoft). Le QBASIC est livré d'office avec le MS-DOS (® Microsoft), et se trouve donc dans le répertoire DOS de presque tous les PC. (Il eut été plus propre de l'écrire en C++, mais à l'époque l'auteur ne pratiquait pas encore ce langage.)

Pour lancer le QBASIC quand on est sous MS-DOS (le symbole C:\>_ apparaît, ou quelque chose qui y ressemble), il faut taper QBASIC au clavier, et enfoncer la touche Enter. Parfois, sous Windows (® Microsoft), un bouton poussoir QBASIC est disponible dans une des fenêtres du "Gestionnaire de programmes". Sinon, cherchez l'icône du QBASIC avec le "Gestionnaire de fichiers", et double-cliquez dessus, ou larguez-la sur une fenêtre du gestionnaire de programmes.

Les lecteurs n'ayant aucune connaissance en BASIC ou QBASIC peuvent soit apprendre sur le tas, aux commandes d'un PC (le fichier d'aide est compréhensible pour un informaticien), soit utiliser les manuels fournis par l'éditeur de MS-DOS, soit acheter dans le commerce (magasins d'informatique, FNAC, librairies scientifiques...) un des nombreux livres traitant de QBASIC ou de QuickBASIC (version professionnelle, avec compilateur).

Le QBASIC ayant une façon un peu ennuyeuse d'assimiler les fonctions et procédures, certains lecteurs préfèrerons peut-être d'abord taper le noyau comme simple texte dans l'éditeur EDIT, puis le charger sous QBASIC.

Afin de diminuer le travail de recopie, signalons que tout le programme peut être tapé en minuscules, et qu'un grand nombre d'espaces ne doivent pas obligatoirement êtres tapés: par exemple a = b + 3 peut sans risque être entré a=b +3.

Les lignes commençant par DECLARE vont être ajoutées automatiquement par le QBASIC. Ne pas s'en inquiéter.

Si vous êtes en train de lire ce texte sur un écran d'ordinateur, ne recopiez pas le programme à la main : faites un copier-coller vers un éditeur texte.

Le listing du noyau contient très peu de REMarques. En cas d'incompréhension de son mode de fonctionnement, l'annexe 2 devrait être d'un certain secours. Ce noyau de logique floue n'a pour ainsi dire aucune répercussion sur le système. Il crée deux tableaux et ajoute quelques fonctions et procédures, mais il ne fait rien du genre d'accéder directement à la mémoire, ou utiliser des routines d'interruption.

(Comment s'utilise un noyau : Si vous voulez faire exécuter par l'ordinateur un programme utilisant la logique floue, il faut le taper ou l'insérer à l'endroit où se trouve la ligne "REM insérer ici votre programme".)

Le noyau:

```
DECLARE FUNCTION aumoinsunpeuvrai! (a!)
DECLARE FUNCTION calcul! (param!)
DECLARE SUB NETTOYAGE (param!)
DECLARE FUNCTION calcult! (param!)
DECLARE FUNCTION centre! (x1!, x2!, x3!, x4!, h!)
DECLARE FUNCTION contrainte! (valeur!, param!)
DECLARE FUNCTION et! (a!, b!)
DECLARE FUNCTION fuzzy! (a!)
DECLARE FUNCTION non! (a!)
DECLARE FUNCTION ou! (a!, b!)
DECLARE FUNCTION ouex! (a!, b!)
DECLARE FUNCTION parametre! (min!, max!, sec!)
DECLARE FUNCTION securite! (param!)
DECLARE FUNCTION si! (valeur!, z!)
DECLARE FUNCTION sipe! (valeur!, z!)
DECLARE FUNCTION sipg! (valeur!, z!)
DECLARE FUNCTION sipgoe! (valeur!, z!)
DECLARE FUNCTION sipp! (valeur!, z!)
DECLARE FUNCTION sippoe! (valeur!, z!)
DECLARE FUNCTION surface! (x1!, x2!, x3!, x4!, h!)
DECLARE FUNCTION tresvrai! (a!)
DECLARE FUNCTION vrai! (a!)
DECLARE FUNCTION zone! (param!, x1!, x2!, x3!, x4!)

REM Noyau de logique floue
REM E. Brasseur
REM octobre 1993

REM liste des parametres:
DIM SHARED pm(0 TO 31, 0 TO 2) AS SINGLE
DIM SHARED pmlibre AS INTEGER

REM liste des zones:
DIM SHARED zo(0 TO 127, 0 TO 5) AS SINGLE
DIM SHARED zolibre AS INTEGER

REM inserer ici votre programme




FUNCTION aumoinsunpeuvrai (a)
   aumoinsunpeuvrai = (a &gt; .25)
END FUNCTION

FUNCTION calcul (param)
   c = calcult(param)
   NETTOYAGE param
   calcul = c
END FUNCTION

FUNCTION calcult (param)
   re = 0
```

```
      st = 0
      FOR z = 0 TO zolibre - 1
         IF zo(z, 0) = param THEN
            s = surface(zo(z, 1), zo(z, 2), zo(z, 3), zo(z, 4), zo(z, 5))
            c = centre(zo(z, 1), zo(z, 2), zo(z, 3), zo(z, 4), zo(z, 5))
            re = re + c * s
            st = st + s
         END IF
      NEXT z
      IF st &lt;&gt; 0 THEN
         calc = re / st
      ELSE
         calc = pm(param, 2)
      END IF
      calcult = calc
   END FUNCTION

   FUNCTION centre (x1, x2, x3, x4, h)
      xg = h * (x2 - x1) + x1
      xd = -h * (x4 - x3) + x4
      centre = ((x4 + x1) / 2 + (xg + xd) / 2) / 2
   END FUNCTION

   FUNCTION contrainte (valeur, param)
      r = valeur
      IF valeur &lt; pm(param, 0) THEN r = pm(param, 0)
      IF valeur &gt; pm(param, 1) THEN r = pm(param, 1)
      contrainte = r
   END FUNCTION

   FUNCTION et (a, b)
      reponse = a
      IF b &lt; a THEN reponse = b
      et = reponse
   END FUNCTION

   FUNCTION fuzzy (a)
      IF a = (2 = 3) THEN
         rep = 0
      ELSE
         rep = 1
      END IF
      fuzzy = rep
   END FUNCTION

   SUB modifieparametre (p, min, max, secu)
      pm(p, 0) = min
      pm(p, 1) = max
      pm(p, 2) = secu
   END SUB

   SUB modifiezone (z, p, x1, x2, x3, x4)
      zo(z, 0) = p
```

```
      zo(z, 1) = x1
      zo(z, 2) = x2
      zo(z, 3) = x3
      zo(z, 4) = x4
   END SUB

   SUB NETTOYAGE (param)
      FOR z = 0 TO zolibre - 1
         IF zo(z, 0) = param THEN zo(z, 5) = 0
      NEXT z
   END SUB

   SUB NETTOYAGEGENERAL
      FOR z = 0 TO zolibre - 1
         zo(z, 5) = 0
      NEXT z
   END SUB

   FUNCTION non (a)
      non = 1 - a
   END FUNCTION

   FUNCTION ou (a, b)
      reponse = a
      IF b &gt; a THEN reponse = b
      ou = reponse
   END FUNCTION

   FUNCTION ouex (a, b)
      ouex = et(ou(a, b), non(et(a, b)))
   END FUNCTION

   FUNCTION parametre (min, max, sec)
      pm(pmlibre, 0) = min
      pm(pmlibre, 1) = max
      pm(pmlibre, 2) = sec
      pmlibre = pmlibre + 1
      parametre = pmlibre - 1
   END FUNCTION

   FUNCTION securite (param)
      securite = pm(param, 2)
   END FUNCTION

   FUNCTION si (valeur, z)
      x1 = zo(z, 1)
      x2 = zo(z, 2)
      x3 = zo(z, 3)
      x4 = zo(z, 4)
      zocalc = 0
      IF valeur &gt;= x2 AND valeur &lt;= x3 THEN
         zocalc = 1
      ELSE
```

```
      IF valeur &gt;= x1 AND valeur &lt; x2 THEN
         zocalc = (valeur - x1) / (x2 - x1)
      ELSE
         IF valeur &gt; x3 AND valeur &lt;= x4 THEN
            zocalc = 1 - (valeur - x3) / (x4 - x3)
         END IF
      END IF
   END IF
   si = zocalc
END FUNCTION

FUNCTION sipe(valeur, z)
   sipe = non(si(valeur, z))
END FUNCTION

FUNCTION sipg (valeur, z)
   x1 = zo(z, 1)
   x2 = zo(z, 2)
   x3 = zo(z, 3)
   x4 = zo(z, 4)
   zocalc = 0
   IF valeur &gt;= x4 THEN
      zocalc = 1
   ELSE
      IF valeur &gt;= x3 AND valeur &lt; x4 THEN
         zocalc = (valeur - x3) / (x4 - x3)
      END IF
   END IF
   sipg = zocalc
END FUNCTION

FUNCTION sipgoe (valeur, z)
   sipgoe = ou(si(valeur, z), sipg(valeur, z))
END FUNCTION

FUNCTION sipp (valeur, z)
   x1 = zo(z, 1)
   x2 = zo(z, 2)
   x3 = zo(z, 3)
   x4 = zo(z, 4)
   zocalc = 0
   IF valeur &lt;= x1 THEN
      zocalc = 1
   ELSE
      IF valeur &gt; x1 AND valeur &lt;= x2 THEN
         zocalc = 1 - (valeur - x1) / (x2 - x1)
      END IF
   END IF
   sipp = zocalc
END FUNCTION

FUNCTION sippoe (valeur, z)
   sippoe = ou(si(valeur, z), sipp(valeur, z))
```

```
END FUNCTION

SUB sortir (z, amplitude)
   zo(z, 5) = amplitude
END SUB

FUNCTION surface (x1, x2, x3, x4, h)
   xg = h * (x2 - x1) + x1
   xd = -h * (x4 - x3) + x4
   surf = (xg - x1) * h / 2 + (xd - xg) * h + (x4 - xd) * h / 2
   surface = surf
END FUNCTION

FUNCTION tresvrai (a)
   tresvrai = (a &gt; .75)
END FUNCTION

FUNCTION vrai (a)
   vrai = (a &gt;.5)
END FUNCTION

FUNCTION zone (param, x1, x2, x3, x4)
   zo(zolibre, 0) = param
   zo(zolibre, 1) = x1
   zo(zolibre, 2) = x2
   zo(zolibre, 3) = x3
   zo(zolibre, 4) = x4
   zo(zolibre, 5) = 0
   zolibre = zolibre + 1
   zone = zolibre - 1
END FUNCTION
```

Annexe 2
Les fonctions et procédures du noyau

Les abréviations suivantes sont utilisées:

elc	état logique classique ("vrai" ou "faux")	(booléen)
elf	état logique flou ([0,1])	(entre 0 et 1)
nr	nombre réel	(nombre normal)
np	numéro de paramètre	(bête entier: 1, 2, 3...)

nz	numéro de zone	(bête entier: 1, 2, 3...)

AUMOINSUNPEUVRAI(elf)

Le résultat de l'évaluation de cette fonction est un état logique classique : "vrai" si elf > 0.5, "faux" sinon.

Elle permet d'écrire des lignes de programme comme :

```
IF AUMOINSUNPEUVRAI(SI(t, tropchaud)) THEN PRINT "Les problèmes commencent"
```

Voir les fonctions VRAI et TRESVRAI.

CALCUL(np) Voir le chapitre 1.

Le résultat de l'évaluation de cette fonction est un nombre réel :

Une fois que toutes les règles d'inférence permettant de donner une amplitude aux zones d'un paramètre ont été données (avec l'ordre SORTIR), la fonction CALCUL donne le résultat final.

La fonction CALCUL remet à zéro l'amplitude de toutes les zones du paramètre calculé. En d'autres termes: l'effet des ordres SORTIR est annulé. (Insistons : uniquement l'effet des ordres SORTIR concernant les zones définies pour le paramètre venant d'être calculé par CALCUL.)

Si les amplitudes de toutes les zones du paramètre sont à zéro, et que donc l'ordre CALCUL ne peut rien déterminer, il prendra la valeur de sécurité donnée lors de la déclaration du paramètre (fonction PARAMETRE).

CALCUL ne tient pas compte des limites données au paramètre lors de sa déclaration. Voir CONTRAINTE.

Dans les cas où il faut calculer plusieurs paramètres, les fonctions CALCUL peuvent tout aussi bien êtres placées chacune juste après ses règles d'inférence propres (SORTIR), ou regroupées ensemble après toutes les règles d'inférence données en vrac.

La façon dont la fonction CALCUL doit déterminer la valeur d'un paramètre de sortie ne semble pas définie de façon rigoureuse. L'auteur a donc choisi la solution qui lui semblait la plus rapide (voir le listing du noyau).

CALCULT(np)

Le résultat de l'évaluation de cette fonction est un nombre réel :

Idem que CALCUL, mais ne remet pas à zéro les amplitudes données par SORTIR aux zones.

CONTRAINTE(nr, np)

Le résultat de l'évaluation de cette fonction est un nombre réel :

CONTRAINTE sert à remettre nr dans les limites initialement données au paramètre numéro np. Exemple :

Soit la définition de paramètre

```
temperature = parametre(0, 100, 60)
```

Après l'exécution de la ligne suivante :

```
v2 = CONTRAINTE(v1, temperature)
```

v2 vaudra 0 si v1<0, v1 si 0<v1<100, et 100 si v1>100

ET(elf1, elf2)

Voir le chapitre 4.

Le résultat de l'évaluation de cette fonction est un état logique flou : elf1 si elf1<elf2, elf2 sinon. En d'autres termes : le plus petit d'entre elf1 et elf2. Exemple :

```
SORTIR courantfort, ET(SI(t, froid), SI(volume, grand))
```

ET peut être imbriquée avec elle-même, ou avec OU et NON :

```
SORTIR courantfaible, ET(SI(t1, tiede), OU(SI(t2, cryogenique), SI(t2, gele)))
```

ET peut évidemment être utilisée en-dehors de l'ordre SORTIR :

```
temporaire = ET(SI(t, tiede), SI(t, gele))
temporaire2 = OU(temporaire, SI(t, glace))
SORTIR courantfaible, temporaire2
```

L'ordre suivant ne mettra jamais dans la variable e un état logique flou supérieur à 0.8 :

```
e = ET(SI(t, froid), 0.8)
```

FUZZY(elc)

Le résultat de l'évaluation de cette fonction est un état logique flou : 1 si elc vaut "vrai", et 0 si elc vaut "faux". Exemples :

```
SORTIR courantfort, OU(SI(t, froid), FUZZY(prixenergie = 0))
SORTIR courantfaible, FUZZY(x < 3 AND y = 4)
```

La fonction symétrique de la fonction FUZZY est la fonction VRAI.

MODIFIEPARAMETRE np, nr1, nr2, nr3

L'exécution de cette procédure permet de changer les données d'un paramètre. Elle peut être utilisée n'importe quand.

Les nouvelles données sont : nr1, nr2, nr3. Elles remplacent les données initialement transmises avec la fonction PARAMETRE.

Par exemple, si on avait fait la déclaration de parametre suivante :

```
temperature = PARAMETRE(0, 100, 37)
```

On peut modifier ces nombres 0, 100, et 37 n'importe quand :

```
MODIFIEPARAMETRE temperature, -10, 80, 0
```

MODIFIEZONE nz, np, nr1, nr2, nr3, nr4

L'exécution de cette procédure permet de changer les données d'une zone. Elle peut être utilisée n'importe quand.

Elle fonctionne de la même façon que l'ordre MODIFIEPARAMETRE, et permet de changer les données np, nr1, nr2, nr3, et nr4 d'une zone déclarée avec la fonction ZONE.

Par exemple, si on avait fait la déclaration de zone suivante :

```
vite = zone(vitesse, 60, 80, 100, 120)
```

On peut modifier n'importe quand les données de la zone dont le numéro est contenu dans la variable nommée "vite" :

```
MODIFIEZONE vite, vitesse, 70, 90, 120, 140
```

Cette procédure ne modifie pas la valeur de l'amplitude éventuellement déjà donnée par un ordre SORTIR.

NETTOYAGE np

L'exécution de cette procédure remet à zéro les amplitudes de toutes les zones du paramètre np. (Amplitudes données avec SORTIR.) Exemple :

```
NETTOYAGE temperature
```

(NETTOYAGE est ce que fait CALCUL, mais pas CALCULT.)

NETTOYAGEGENERAL

L'exécution de cette procédure remet à zéro les amplitudes de toutes les zones de tous les paramètres. (Amplitudes données avec SORTIR.)

Ordre normalement inutile, mais permettant d'être sûr. Par exemple dans des programmes hétéroclites où on ne donne pas toujours une amplitude à toutes les zones d'un paramètre de sortie.

NON(elf)

Voir le chapitre 4.

Le résultat de l'évaluation de cette fonction est un état logique flou "inverse" de elf : 1-elf.

Voir les fonctions ET et OU.

OU(elf1, elf2)

Voir le chapitre 4.

Le résultat de l'évaluation de cette fonction est un état logique flou : elf1 si elf1>elf2, sinon : elf2. En bref : le plus grand des deux.

Cette fonction logique floue donne la plus "favorable" de deux amplitudes elf1 et elf2.

Symétrique de la fonction ET. S'utilise d'exactement la même façon.

OUEX(elf1, elf2)

Voir le chapitre 4.

Le résultat de l'évaluation de cette fonction donne un état logique flou. Elle est la transposition de la fonction booléenne "ou exclusif", soit : (elf1 OU elf2) ET NON (elf1 ET elf2)

OUEX détermine dans quelle mesure elf1 ou elf2 sont vrai, mais pas les deux à la fois.

Voir les fonctions NON, OU et ET.

PARAMETRE(nr1, nr2, nr3)

Voir le chapitre 1.

Le résultat de l'évaluation de cette fonction est un numéro de paramètre. Ce numéro est destiné à être stocké dans une variable, "pour référence ultérieure".

Cette fonction déclare un paramètre. Que ce soit une entrée, sortie, ou paramètre intermédiaire.

L'usage des trois nombres réels donnés est le suivant : nr1 : limite inférieure des valeurs que prendra probablement le paramètre.

nr2 : limite supérieure des valeurs que prendra probablement le paramètre.

nr3 : sécurité

nr1 et nr2 ne sont jamais utilisés, sauf par la fonction CONTRAINTE.

nr3 est la valeur que donnera l'ordre CALCUL s'il n'arrive pas à calculer une valeur.

Exemple :

```
vitesse = PARAMETRE(0, 200, 50)
```

PARAMETRE ne doit être utilisé qu'une seule fois par paramètre, au début du programme. Il y a une limite au nombre de paramètres que l'on peut déclarer. Comme il est, le noyau autorise la déclaration de 32 paramètres. Pour augmenter ce nombre, il suffit de modifier l'ordre DIM pm(0 TO 31, 0 TO 2).

Les données d'un paramètre peuvent êtres modifiées n'importe quand, grâce à l'ordre MODIFIEPARAMETRE. On peut aussi jongler avec les numéros de paramètre. Soit temperature1 et temperature2 deux variables contenant des numéros de paramètre, on peut les intervertir de la façon suivante :

```
temporaire = temperature1
temperature1 = temperature2
temperature2 = temporaire
```

SECURITE(np)

Le résultat de l'évaluation de cette fonction est la valeur de sécurité donnée au paramètre lors de sa déclaration par la fonction PARAMETRE.

Si par exemple on a fait la déclaration de paramètre suivante :

```
acceleration = PARAMETRE(-9, 9, 1)
```

L'ordre suivant fera afficher le nombre 1 à l'écran :

```
PRINT SECURITE(acceleration)
```

SI(nr, nz)

Voir les chapitres 1 et 4.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr se trouve dans la zone nz.

nz est le numéro d'une zone (déclarée par la fonction ZONE).

Le résultat sera 0 si nr est en-dehors de la zone, 1 s'il est tout à fait dedans, et entre 0 et 1 s'il est "sur les bords de la zone".

Exemples :

```
SORTIR courantfort, SI(t, froid)
SORTIR courantfort, SI(20, bonnetemperature)
```

Il est également possible, mais de mauvais goût, de vérifier si une valeur se trouve dans une zone pour laquelle elle n'a pas été définie. Du genre de :

```
SORTIR courantfort, SI(temperature, courantfaible)
```

SIPE(nr, nz)

Voir la fonction SI. PE veut dire Pas Egal.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr se trouve hors de la zone nz.

SIPG(nr, nz)

Voir la fonction SI. PG veut dire Plus Grand.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr est plus grand que la zone nz. ("A droite de la zone.")

SIPGOE(nr, nz)

Voir la fonction SI. PGOE veut dire Plus Grand Ou Egal.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr est plus grand ou égal à la zone nz. ("Dedans, ou à droite".)

SIPP(nr, nz)

Voir la fonction SI. PP veut dire Plus Petit.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr est plus petit que la zone nz. ("à gauche de la zone".)

SIPPOE(nr, nz)

Voir la fonction SI. PPOE veut dire Plus Petit Ou Egal.

Le résultat de l'évaluation de cette fonction sera un état logique flou chiffrant dans quelle mesure le nombre nr est plus petit ou égal à la zone nz. ("Dedans, ou à gauche".)

SORTIR nz, elf

Voir les chapitres 1 et 4.

L'exécution de cette procédure impose une amplitude elf à la zone nz.

L'effet de SORTIR est de stocker en mémoire l'information comme quoi la zone est "tronquée" à une hauteur donnée

par elf. Cette information sera utilisée lors de l'usage de la fonction CALCUL.

En d'autres termes : SORTIR dit dans quelle mesure le paramètre auquel se rattache la zone, devra tendre à se trouver dans cette zone.

Il est permis d'utiliser SORTIR plusieurs fois de suite pour une même zone, mais seul le dernier usage en date compte.

SORTIR zone, 0 remet à zéro l'amplitude de la zone.

NETTOYAGE remet à zéro l'amplitude de toutes les zones d'un paramètre.

L'ordre NETTOYAGEGENERAL remet à zéro toutes les amplitudes de toutes les zones, et donc annule l'effet de tous les ordres SORTIR.

TRESVRAI(elf)

Le résultat de l'évaluation de cette fonction est un état logique classique : "vrai" pour toute amplitude logique floue elf plus grande que 0.75, "faux", sinon.

Voir les fonctions VRAI et AUMOINSUNPEUVRAI.

VRAI(elf)

Le résultat de l'évaluation de cette fonction est un état logique classique : "vrai" pour toute amplitude logique floue elf plus grande que 0.5, "faux", sinon.

Cette fonction transforme une amplitude logique floue elf en état booléen classique. C'est nécessaire dès qu'on veut appliquer des états flous à des fonctions et procédures du QBASIC (IF, WHILE, UNTIL, AND, OR, NOT...) Permet d'écrire des lignes de programme comme :

```
IF VRAI(SI(t, tropchaud)) THEN ALARM 1
e = VRAI(SI(t, tropchaud))
IF e THEN ALARM 1
```

Voir également les fonctions AUMOINSUNPEUVRAI et TRESVRAI.

Pour obtenir l'équivalent de "FAUX(elf)", il faut utiliser la fonction booléenne NOT du QBASIC :

```
IF NOT VRAI(SI(t, tropchaud)) THEN PRINT "OK"
```

(Ce qui donne le même résultat que :)

```
IF VRAI(SIPE(t, tropchaud)) THEN PRINT "OK"
```

La fonction symétrique de la fonction VRAI, est la fonction FUZZY.

ZONE(np, nr1, nr2, nr3, nr4)

Voir les chapitres 1 et 4.

Le résultat de l'évaluation de cette fonction est un numéro de zone.

Cette fonction sert à déclarer une zone, rattachée à un paramètre. np est le numéro de ce paramètre, nr1 est la valeur pour laquelle cette zone commence, nr2 est la valeur pour laquelle cette zone est pleinement réalisée, nr3 est la valeur pour laquelle la zone commence à cesser, et nr4 est la valeur à partir de laquelle la zone cesse définitivement.

Surtout s'il s'agit d'une zone de sortie, il ne faut pas que sa surface soit nulle. Par contre, on peut avoir nr2 = nr3, ou nr1 = nr2 et/ou nr3 = nr4. On peut dire que les zones ont deux usages :

- Permettre, grâce à l'ordre SI, de déterminer dans quelle(s) zone(s) se trouve un paramètre mesuré.

- Permettre d'imposer dans quelle(s) zone(s) devra être un paramètre qu'on veut imposer à une machine, grâce à l'ordre SORTIR, puis à la fonction CALCUL.

ZONE n'est utilisé qu'une seule fois pour chaque zone, en début de programme. Tel qu'il est, le noyau autorise la déclaration de 128 zones. Pour augmenter ce nombre, il suffit de modifier l'ordre DIM zo(0 TO 127, 0 TO 5) (ou utiliser l'ordre REDIM du QBASIC).

A moins d'utiliser des tableaux, deux zones ne peuvent pas avoir le même nom, même si elles se rattachent à des paramètres différents.

ZONE est une fonction normale, dont le résultat est un simple nombre entier (le numéro de la zone), destiné à être mémorisé dans une variable. Elle fonctionne d'une façon analogue à PARAMETRE : elle fait retenir des données à l'ordinateur, et fournit un numéro (le numéro de zone), qui est stocké dans une variable "pour référence ultérieure".

Si par exemple on a les déclarations suivantes au début du programme :

```
niveau = PARAMETRE(0, 100, 100)
plein = zone(niveau, 80, 100, 1000, 1000)
```

Alors l'ordre suivant affichera 0.5 à l'écran :

```
PRINT SI(90, plein)
```

L'ordre suivant affichera un nombre entier à l'écran (inutile et imprédictible) :

```
PRINT plein
```

Ravoir les chiffres donnés lors des déclarations

Tous les chiffres donnés lors des déclarations avec PARAMETRE et ZONE se trouvent dans les deux tableaux pm et zo.

Si par exemple on a donné la déclaration suivante :

```
tropchaud = zone (temperature, 38, 41, 1000, 1000)
```

Alors l'ordre

```
n = pm(tropchaud, 2)
```

mettra 41 dans la variable n.

Remerciements

La mise en page, les dessins et les tests des programmes ont été effectués avec une station de travail Acorn Archimedes prêtée par M. Frédéric Cloth.

Cet exposé à bénéficié des critiques de MM. Didier Bizzarri et Frédéric Cloth.

Matériel d'impression du livre imprimé en 1994 : Yves-Dominique Frank et Robert Franck.

Renseignements sur l'édition, les codes à barres: M. Chinh Dang Vu, Dr. Oosterbosch, le service Dépot Légal de la Bibliothèque Royale Albert I, l'AFNIL, La Technothèque.

Encouragements et conseils: Centre Electronique Liégeois, France Plumer, Roeland Zeh, Dimitri Gathy, Alexandra Polmans

Eric Brasseur - 1994 [Accueil | eric.brasseur@gmail.com]