数　値　解　析　32－1
（１９９０．　３．　１７）

ファジィ数の演算シミュレータ

広田豊彦　矢鳴虎夫

九州工業大学情報工学部知能情報工学科

ファジィ数はファジィ集合の要素が実数値となっているもので、それに対して四則演算を定義することができる。しかしファジィ数の四則演算は、一般のファジィ応用で広く使われている推論操作に比べてはるかに複雑な演算を必要とする。本研究では、ファジィ数のディジタル表現とその演算方式について検討を行い、演算シミュレータを作成して実験を行った。2種類の演算方式、全点演算型とサンプリング点演算型について検討した結果、全点演算型は誤った結果が出てくる可能性があること、サンプリング点演算型は誤差を含むものの、どんな場合でもほぼ妥当な結果が得られることが明らかになった。

A Simulator for Fuzzy Number Calculation

Toyohiko Hirota　Torao Yanaru

Department of Artificial Intelligence
Kyushu Institute of Technology

Kawazu 680-4, Iizuka 820, Japan

A fuzzy number is a fuzzy set whose members are real numbers, and arithmetic operations are defined for it. But the arithmetic operation for fuzzy numbers requires far more complicated calculation than an inference operation which is widely used in an usual fuzzy application. In our research, we have studied digital representaions of fuzzy numbers and calculation methods for them, developed a simulator for the calculation, and experimented it. We have examined two types of the methods, all-point-calculation type and sampling-point-calculation type. We have concluded that the all-point-calculation may produce wrong results in several cases, and that, although the results contain some error, the sampling-point-calculation type produces almost valid results in any case.

1 はじめに

L. A. Zadehによって提唱されたファジィ理論は、近年、制御の分野を中心に、研究ならびに実用化が盛んになっている[1]。ファジィ理論では単一の数値のかわりに数値の集合(ファジィ集合)が使われる。しかも集合のそれぞれの要素はグレードとよばれる0から1までの範囲の実数値が対応付けられる。このような膨大な情報量をもつものを1つの要素として扱うことにより、1)あいまいな推論規則をそのまま簡単に記述できる、2)その規則の単純さにもかかわらず、実用的に妥当な制御を行うことができる、という利点が生まれる。しかも推論はファジィ集合の要素ごとに独立に行えるので、並列動作する推論チップを与えることによって、十分高速な推論を行うことが可能である[2]。

ファジィ集合の要素が実数値のとき、ファジィ集合間の四則演算を定義することができる。ところがそのようなファジィ集合(ファジィ数)の四則演算は、推論操作に比べてはるかに複雑な演算を必要とする。そのために実用化はもちろんのこと、研究面でもほとんど進展がないのが実情である。

本研究では、ファジィ数のディジタル表現とその演算方式について、演算シミュレータを用いて実験を行い、ファジィ数の演算に関する問題点を明らかにした。

2 ファジィ数

2.1 ファジィ集合

ある集合Xのファジィ(部分)集合Aは、次のような関数h_Aで定義される。

$$h_A : X \rightarrow [0, 1] \qquad (1)$$

ここで関数h_Aは、集合Xの任意の要素xに対して、それがファジィ集合Aに含まれる度合いを、0から1までの範囲の値で与えている。関数h_Aをファジィ集合Aのメンバシップ関数とよび、関数h_Aが要素xに対して与える値をグレードとよぶ。また、全体集合Xのことをファジィ集合Aの台集合とよぶ。

身長140cmから200cmまでの範囲を台集合として、3つのファジィ集合「高い」、「中くらい」、「低い」のメンバシップ関数を考えてみると、図1のようになる。身長200cmはファジィ集合「高い」のグレードがほぼ1になっている。それに対して「中くらい」や「低い」のグレードは0である。つまり身長200cmはだれが見ても高いということになる。身長140cmは「低い」のグレードが1で、「中くらい」や「高い」のグレードは0であり、だれが見ても低いということになる。ところが身長170cmは「中くらい」のグレードが1ではあるが、「高い」や「低い」のグレードも0ではなくて、0.3くらいの値になっている。つまり、身長170cmは「中くらい」ではあるが、人によって、あるいは場合によっては「低い」または「高い」とみなされることを示している。さらに、身長180cmは「高い」と「中くらい」の両方でともにグレードが0.75になっている。身長180cmは高いとも低いとも、どちらとも言える(言えない)ということになる。

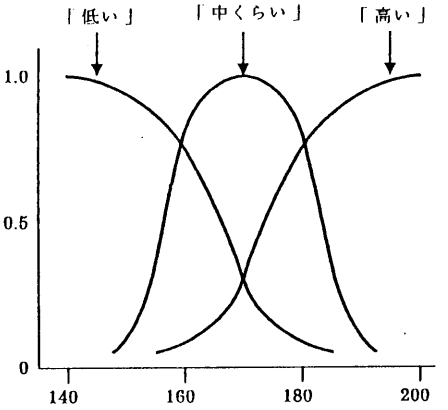

図1 ファジィ集合の例

非ファジィ的な考え方では、たとえば、身長180cm以上は「高い」、180cm以下は「中くらい」という分類をしていた。すると、身長180cmははたして高いのか、それとも中くらいなのか、身長180cmは高いが、身長179cmは中くらいなのか、ということが問題になってくる。このことは、「高い」、「中くらい」、「低い」という3分類から、もっと細かく分類しても同じことである。いずれにしても、ある1つの要素はどれか1つの集合に入るか、入らないかのいずれかである。それに対してファジィ理論では、要素と集合の関係は、0から1までの範囲のグレードで与えられる。もう1つ重要な点は、1つの要素が複数の(論理的には排他的な)集合に対して0以上のグレードを持てるということである。

一般に人の考え方はそれほど厳密ではない。同じ1つの事象に対しても人によってそれに対する判断が異なる。また同じ人でも時と場合によって判断が異なることもめずらしくない。そのような場合に、その判断が異なる根拠が論理的に明白であれば従来型の処理で対応できるが、根拠がはっきりしないときには、全く何の対処もできない。それに対して、ファジィ理論では、「身長が高い」というようなあいまいな判断基準を、ファジィ集合に対応させることによって、あいまいさを残したままの形で表現できる。この点がファジィ理論の特徴である。

ファジィ集合に対して従来の集合を特に区別するときには、クリスプ集合とよぶ。ファジィ集合のメンバシップ関数に対応するものとして、クリスプ集合では定義関数とよ

ばれるものを定めることができる。全体集合X、その部分集合をEとすると、定義関数は次のような形になる。

$$\chi_E : X \rightarrow \{0, 1\} \tag{2}$$

ファジィ集合のメンバシップ関数の値域は、[0, 1]という連続的な区間であるが、クリスプ集合の定義関数は{0, 1}という2つの要素からなる離散的な集合である。定義関数の値は、要素が集合Eに含まれるかどうかによって、次のように定められる。

$$\chi_E(x) = \begin{cases} 1, & x \in E \\ 0, & x \notin E \end{cases} \tag{3}$$

2.2 α-カットと拡張原理

ファジィ集合は図1に示したように、2次元的なグラフで表されるので、そのようなものを直接扱うのは不便なことがある。そこでファジィ集合をふつうの数値で表すための方法として、(弱)α-カットがある。ファジィ集合Aのα-カットA_αは次のように定義される。

$$A_\alpha = \{x \mid h_A(x) \geq \alpha\} \tag{4}$$

台集合が実数で、メンバシップ関数が上に凸で連続であれば、α-カットは図2のようになる。すなわち、α-カットは閉区間$[l, r]$で表されることになる。α-カットはその定義からわかるように、クリスプ集合である。

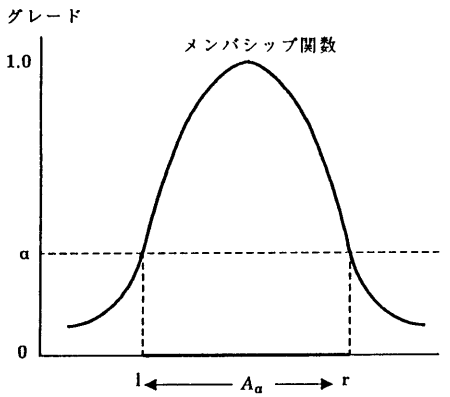

図2 α-カット

このα-カットから逆にメンバシップ関数を再構成することができる。これは分解原理とよばれるもので、次のような関係式で表される。

$$h_A(x) = \bigcup_{0<\alpha\leq 1} \{\alpha \wedge \chi_{A\alpha}(x)\} \tag{5}$$

ここで$\chi_{A\alpha}(x)$はα-カットA_αの定義関数である。記号∧は最小値を、記号∪は最大値をとることを表している。

さて、XからYへの関数

$$f : X \rightarrow Y \tag{6}$$

を考える。関数fはXの要素からYの要素への対応を示しているが、これをクリスプ集合間の対応関係に拡張することができる。すなわち次のように定義する。

$$f(E) = \{y \mid y = f(x);\ x \in E\} \tag{7}$$

このように関数fをクリスプ集合間の対応関係を表すように拡張すると、関数fをファジィ集合Aのα-カットに対して適用することができる。そして適用結果を別のファジィ集合Bのα-カットB_αであると定義する。すなわち、

$$B_\alpha = f(A_\alpha) \tag{8}$$

とする。このようにすると、ファジィ集合A(のα-カット)から仮想的なファジィ集合Bのα-カットを計算することができる。ところが分解原理((5)式)によれば、α-カットから元のメンバシップ関数を再構成できる。すなわち、(8)式(と(5)式)からファジィ集合Bが定義される。このファジィ集合Bはファジィ集合Aに関数fを適用して得られるものであるから、$f(A)$と表すことにすると、(8)式はあらためて、

$$f(A)_\alpha = f(A_\alpha) \tag{9}$$

と書き直すことができる。これが関数の拡張原理とよばれるものである。

上でも述べたようにα-カットから元のメンバシップ関数を再構成することができる。そこで関数の拡張((9)式)をα-カットを使わずに、メンバシップ関数の形で表現すると、次のようになる。

$$h_{f(A)}(y) = \bigcup_{y=f(x)} h_A(x) \tag{10}$$

この式は次のようなことを意味している。ファジィ集合$f(A)$のメンバシップ関数$h_{f(A)}$のyにおける値は、$y=f(x)$となるようなすべてのxに対して、ファジィ集合Aのメンバシップ関数$h_A(x)$の値を計算し、その最大値をとることを意味している。

(9)式と(10)式は等価な表現である。実際に$f(A)$を計算するときにどちらの式が便利かは、場合によって異なるので、それぞれに応じて選択すればよい。

2.3 ファジィ数

ファジィ数とは、実数の集合を台集合とする正規で凸のファジィ集合で、しかもα-カットが閉区間となるものである。ここでファジィ集合が正規であるとは、メンバシップ関数の最大値が1になることである、すなわち、

$$\bigcup_{x \in X} h_A(x) = 1 \qquad (11)$$

である。また、ファジィ集合が凸であるとは、厳密には、任意のa,b(a < b)に対して、

$$h_A(x) \geqq h_A(a) \wedge h_A(b), \quad \forall x \in [a, b] \qquad (12)$$

が成立することである。

ファジィ数のメンバシップ関数の例を図3に示す。ここでは「およそ3」と「およそ7」の2つのファジィ数のメンバシップ関数をあげている。なお、以後「およそ3」、「およそ7」などを、~3、~7のように表すものとする。

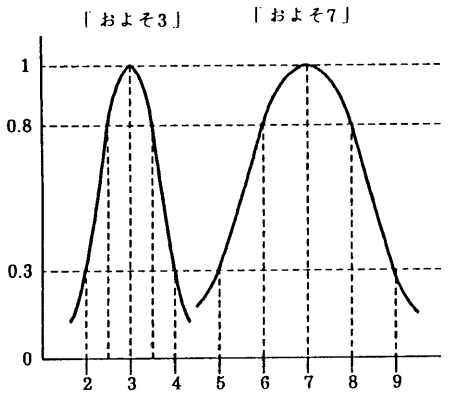

図3　ファジィ数の例

ファジィ数の演算は2.2で述べた拡張原理にしたがう。たとえば、次のような関数f(x)

$$f(x) = 2x + 1$$

に~3を代入して計算してみる。まず~3のα-カットを計算する。ここでは次の3つのα-カットだけを考えることにする。

$$\sim 3_\alpha = \left\{ \begin{matrix} [3.0, 3.0], & \alpha = 1.0 \\ [2.5, 3.5], & \alpha = 0.8 \\ [2.0, 4.0], & \alpha = 0.3 \end{matrix} \right\}$$

それぞれのα-カットに対して関数f(x)を計算すると、f(~3)のα-カットが得られる。

$$f(\sim 3)_\alpha = \left\{ \begin{matrix} [7.0, 7.0], & \alpha = 1.0 \\ [6.0, 8.0], & \alpha = 0.8 \\ [5.0, 9.0], & \alpha = 0.3 \end{matrix} \right\}$$

これは図3に示した「およそ7」(~7)になっている。

メンバシップ関数が左右対称であるようなファジィ数Aは、メンバシップ関数の型を表すL、中心を表すc、広がりを表すwを使って次のように表現される。

$$A = (c, w)_L \qquad (13)$$

このとき、ファジィ数Aのメンバシップ関数は次のようになる。

$$h_A(x) = L((x-c)/w) \qquad (14)$$

関数L(x)は型関数とよばれるもので、以下のような性質を満たす。

1) $L(x) = L(-x)$
2) $L(0) = 1$
3) $L(x)$は$[0, \infty)$において厳密に減少する

関数L(x)の例としては次のようなものがある。

$$L_1(x) = max(0, 1-|x|^p)$$
$$L_2(x) = e^{-|x|^p}$$
$$L_3(x) = 1/(1+|x|^p) \qquad (p > 0)$$

$L_2(x)$で$p=2$とすると、$(c, w)_L$のメンバシップ関数は図4のようになる。

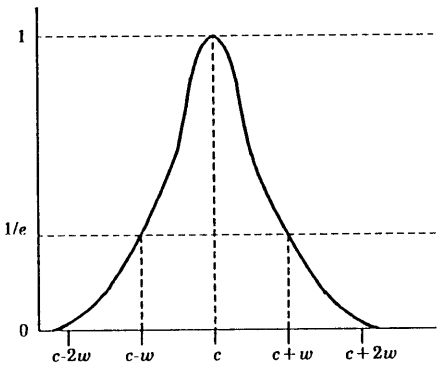

図4　$(c, w)_L$のメンバシップ関数

3　ファジィ数のディジタル表現

ファジィ数の演算をするためには、メンバシップ関数を演算が可能な形で表現しなければならない。そのための方法として次のようなものが考えられる。

1) メンバシップ関数の形を決めて、パラメタによってそれぞれのファジィ数を表すたとえば、型関数

$$L(x) = max(0, 1-|x|)$$

を定めるとファジィ数

$$A = (c, w)_L$$

はcとwの組合せで表される。

2) メンバシップ関数を次のように離散的に表す。

$$h_A = L(-1)/(c-w) + L(0.9)/(c-0.9w) + \cdots + 1/c + \cdots + L(1)/(c+w) \quad (15)$$

(注)a/bという表記は、台集合中の点bにおけるメンバシップ関数の値がaであることを表している。

このようにすると、21個の型関数の値とcとwの組合せでメンバシップ関数が決まる。この場合、型関数はきちんとした関数形で与えられなくてもよい。(型関数$L(x)$は$x>1$に対して$L(x)\simeq 0$であることを前提としている。また、型関数の対称性を使えば、10個の関数値がわかればよいことになる。)

3) メンバシップ関数を次のようにα-カットで表す。

$$\begin{aligned} h_A &= [a_1, b_1] && \alpha=0.1 \\ &= [a_2, b_2] && \alpha=0.2 \\ &\cdots && \cdots \\ &= [a_9, b_9] && \alpha=0.9 \\ &= c && \alpha=1.0 \end{aligned} \quad (16)$$

方法1)がもっとも簡単であるが、メンバシップ関数の形を固定することになるので、どのような形がよいのか、固定してもよいのかどうかなどについての議論が必要であろう。型関数を使う表示ではないが、似たようなアプローチとして、Teodorescu[3]はファジィ数を三角形や台形のメンバシップ関数で表現することを提案している。

方法2)は、もっとも一般的にメンバシップ関数を表現できる。しかしこの方法では、拡張原理に基づいてファジィ数の演算を行うとき、種々の問題が発生する。本研究では、この方法に基づく演算シミュレータを作成し、シミュレーションを行った。これについては4章で議論する。

方法3)は拡張原理による計算が簡単であるという点で、方法2)よりもすぐれている。この方法の問題点は、メンバシップ関数が多峰性になったときに、扱いが困難になることである。もちろんファジィ数はその定義からして、メンバシップ関数は必ず単峰性である。しかしファジィ数をファジィ推論に使おうとすると、そのときには単峰性が維持されるとはかぎらないので、この問題についても考慮しておく必要があるだろう。

4 ファジィ数の演算シミュレータ

4.1 全点演算型

ファジィ数のディジタル表現とその演算方式の研究を行うことを目的として、簡単なファジィ数の演算シミュレータを作成し、実験を行った。このシミュレータを使って実際にファジィ数の演算を行うことが目的ではないので、今回作成したシミュレータは、2つのファジィ数の加算だけを行う特殊なものである。

ファジィ数の演算は、2.2で述べた拡張原理に基づいて行う。2.2の(10)式は1変数関数の拡張を表しているが、2変数関数でも同様な式が成り立つ。具体的には、2つのファジィ数A、Bのメンバシップ関数をそれぞれ、$h_A(x)$、$h_B(y)$とすると、$C=A+B$のメンバシップ関数$h_C(z)$は、

$$h_C(z) = \bigcup_{z=x+y} (h_A(x) \wedge h_B(y)) \quad (17)$$

となる。

$z=x+y$となるxとyの組合せは、x、yが実数であることから、無限に存在する。しかし今回作成したシミュレータでは、21点でサンプリングされたメンバシップ関数を対象とするので、すべてのxとyの組合せでも441通りしかない。一般性を保つためには、結果のメンバシップ関数$h_C(z)$も21点でサンプリングされる必要があるが、最初に作成した全点演算型のシミュレータでは、441点をそのまま表示している。ただし、xとyのサンプリング値によっては、異なるxとyの組合せで同じzの値になることがあり、その場合には(13)式にしたがって、より大きい値をとってきている。

2つのファジィ数について毎回メンバシップ関数の値(21点のサンプリング値)を入力するのはたいへんであり、また今回の研究目的からもそこまで細かく指定する必要はない。そこで次に示す3つの型関数を用意し、どの型関数を使うのかを指定することとした。

$$\begin{aligned} L_1(x) &= e^{-4x^2} \\ L_2(x) &= 1 + e^{-4} - e^{-4(1-x)^2} \\ L_3(x) &= max(0, 1-|x|) \end{aligned}$$

シミュレータは、2つの演算数についてそれぞれ、中心c、幅w、型関数の番号を指定すると、2つの演算数と演算結果のメンバシップ関数を図示する。図5と図6はシミュレータによる演算結果を示している。図5、図6とも、上の2つのグラフは演算数のメンバシップ関数であり、下の1つが結果のメンバシップ関数である。それぞれのグラフの左上の数字は、左が中心c、右が幅wを表している。x軸の目盛りはサンプリング点を示しているが、演算結果のグラフにおいては、その点が計算されているとはかぎらない。

図5はファジィ数~3とファジィ数~5(いずれも幅wは2、型関数は$L_1(x)$)の加算の結果を示している。結果は直観的にもあきらかなように、~8であり、幅は4になっている。結果のメンバシップ関数が階段状になっているが、x軸の目盛り上の点だけをとり、中間の値を捨てれば、なめらかなメンバシップ関数を得ることができる。

図6も図5と同じく、2つのファジィ数~3と~5の加算であるが、~3の幅が11、~5の幅が13となっている。これは(17)式に基づいてメンバシップ関数を計算するときに、すべてのサンプリングされたxとyの組合せに対して$z=x+y$が異なるようにしたものである。したがって、441点がそのままプロットされている。正確な値はプロットされた点群の上限を結んだものになるが、それを大きく下回る点が多数プロットされている。本来ならばすべてのxとyの組合せについてメンバシップ関数の値を計算し、その最大値をとるべきところを、サンプリングされた箇所しか計算していないためにこのようになるのである。つまりファジィ数の演算においてうかつにサンプリングされた点を使うと、全く誤った計算値が出てくることがわかる。

なお、ここには示していないが、型関数の違いによる特別な影響は認められなかった。

4.2 サンプリング点演算型

全点演算型の問題点は、(17)式において、$z=x+y$となるすべてのxとyの組合せについて、メンバシップ関数の値を

計算しなければならないのに、そのうちの1つのxとyの組合せについてしか計算していないということである。そこで、zを固定し、xの21個のサンプリング点に対して、それぞれy=z-xを計算することを考えてみる。

ファジィ数の加算の結果の中心cと幅wは、単純に演算数の中心と幅をそれぞれ加えたものになる。したがって、演算結果のメンバシップ関数のサンプリング点zはただちに計算できる。上で述べたように、xについてはサンプリング点をとることにすれば、その点のメンバシップ関数の値は与えられている。問題は、そのときのy=z-xの値が、yの方のサンプリング点と一致するとはかぎらないということである。そこで、yに等しいかそれ以下の最大のサンプリング点のメンバシップ関数の値をとってくることにする。これはメンバシップ関数を階段状の関数で近似することを意味している(図7)。

上の考えに基づいて作成したのがサンプリング点演算型のシミュレータである。内部の演算方式を除くと、全点演算型のシミュレータと同じである。図8、図9がサンプリング点演算型シミュレータの演算結果を示している。それぞれ図5、図6と同じ演算数を対象としている。この方式によれば、いずれの場合でも妥当なメンバシップ関数が得られる。

サンプリング点演算型の問題点は、演算数のうちの一方はサンプリング点でない箇所を計算に使うので、それによ

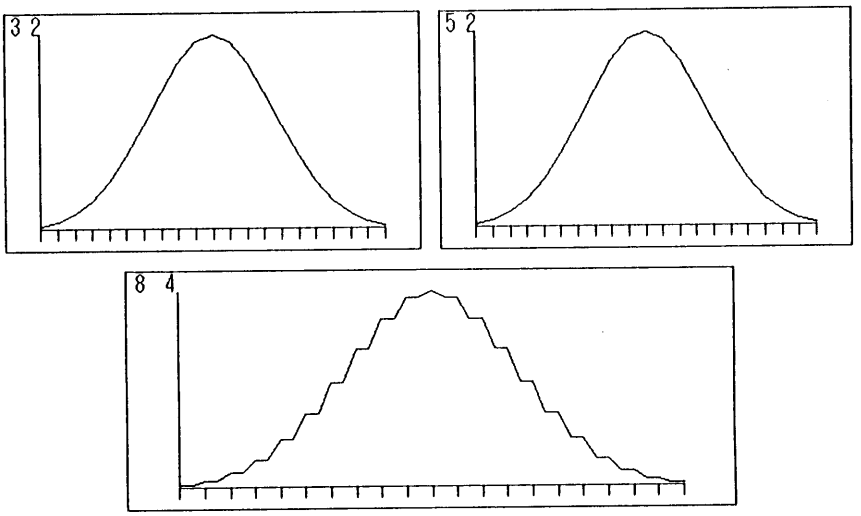

図5　全点演算型~3+~5(幅が同じ)

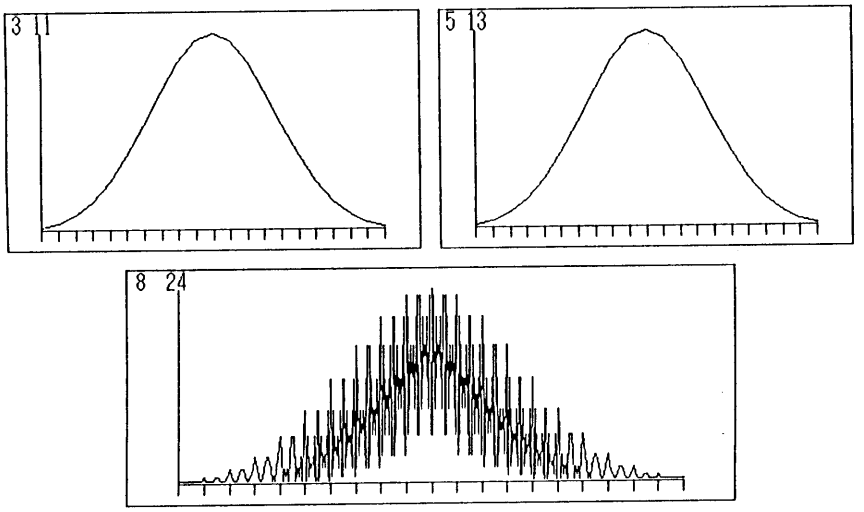

図6　全点演算型~3+~5(幅が異なる)

る誤差が入ってくることである。この誤差は演算を繰り返すと累積する可能性がある。この点については今後さらに考察をすすめる必要がある。

5　おわりに

本研究では、ファジィ数の演算について、全点演算型とサンプリング点演算型の2種類のシミュレータを作成し、実験を行った。

全点演算型は、場合によっては全く誤った計算値が出てくることが明らかになった。これは単に全点演算型が適切でないということだけではなく、ファジィ数のディジタル処理に関する根本的な問題点を示唆している。

サンプリング点演算型の方は、メンバシップ関数を階段状とみなすことにより、全点演算型のような不合理は生じない。しかしこの場合には、結果のメンバシップ関数はサンプリング点自体が始めから誤差を含んでいることになる。したがって、ファジィ数の演算を繰り返すと誤差が累積することになる。このような誤差の累積が最終結果にどのような影響を及ぼすのか、今後十分に検討を行う必要がある。

また、今回はあまりふれていないが、メンバシップ関数をα-カットで表示する方法がある。その場合には、α-カッ

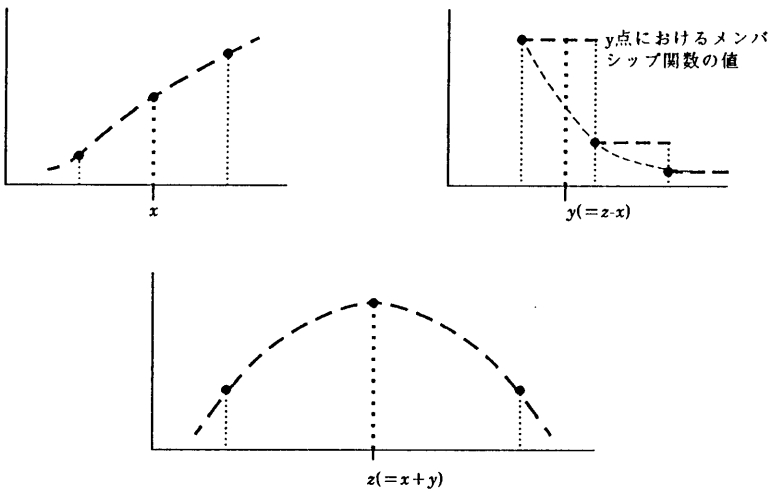

図7　サンプリング点演算型の考え方

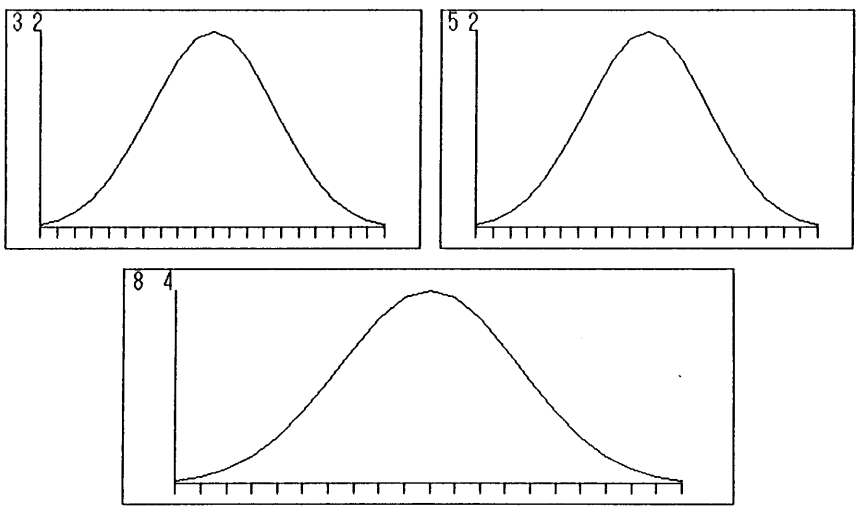

図8　サンプリング点演算型~3+~5(幅が同じ)

トは区間で表されるので、ファジィ数の演算は区間演算に帰着することになり、今回検討したような問題は生じない。しかしこんどは、多峰性を扱うのが困難になるので、そのことを念頭に置いて検討しなければならない。

謝辞

日頃ご議論いただく本学制御システム工学科 山川烈教授、内野英治博士、福岡県北九州工業試験場 松家繁氏、ならびに産業医科大学 堀岡正夫氏に感謝します。

参考文献

[1] 寺野寿郎他編「ファジィシステム入門」(1987、オーム社)

[2] 山川烈「ファジィハードウェアシステム —ディジタル方式とアナログ方式のファジィ推論チップ—」(1989、情報処理、VOl.30, No.8(931-941))

[3] Teodorescu, et. al., "Binary-Like Codes for Trapezoidal Fuzzy Numbers (1989, Proceedings of the 1989 IFSA Congress, (548-551))

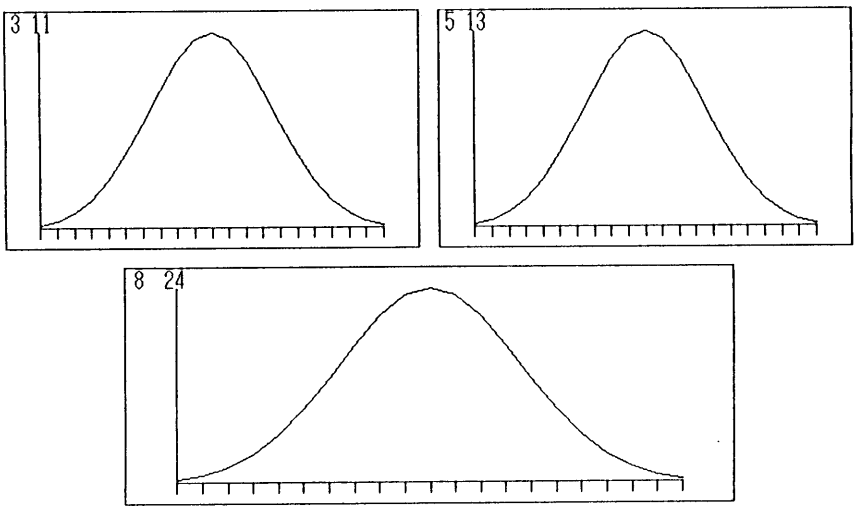

図9 サンプリング点演算型~3+~5(幅が異なる)