(Version: 2016/6/7)

Affine Arithmetic について

柏木 雅英

1 はじめに

浮動小数点数など有限桁の数値を用いる数値計算において、計算を行うと同時にその結果の誤差評価をも同時に計算するような方法を総称して精度保証付き数値計算と呼び、近年急速な進歩を遂げている。精度保証付き数値計算の実現において最も基本的かつ重要な技法に、区間演算が挙げられる。区間演算とは、実数値を [下限, 上限] という 2 つの浮動小数点数で挟まれた区間で表現し、その区間同士の加減乗除等の演算を「演算結果として有り得る集合を包含するように」定義することにより行われるものである。そのとき、区間の両端を計算する際に丸めの向きを「外向き」にしておくことによって丸め誤差の影響分を区間内に収め、丸め誤差の把握を行うことが出来る。

区間演算の問題点の一つとして、確かに計算された区間は真の値を含むものの、区間幅が極端に広くなってしまうことが多い点が挙げられる。この現象は、区間演算が中間変数間の相関性を考慮していないために発生する。Affine Arithmetic は、中間変数間の相関性を考慮することによりこの問題点を解決する。

2 区間演算

2.1 区間演算

$\mathbb{F} \subset \mathbb{R}$ をある計算機で表現可能な実数の集合とする。例えば IEEE 754 Std. の倍精度で表せる数の集合など。区間演算とは、実数値を [下限, 上限] という 2 つ $\mathbb{F}$ の元で挟まれた区間で表現し、その区間同士の加減乗除等の演算を「演算結果として有り得る集合を包含するように」定義することにより行われるものである。つまり、区間演算は、区間 X, Y と、二項演算 $\cdot \in \{+, -, \times, \div\}$、単項演算 g について、

$$\begin{aligned} X \cdot Y &\supset \{x \cdot y \mid x \in X, y \in Y\} \\ g(X) &\supset \{f(x) \mid x \in X\} \end{aligned}$$

を満たすような集合演算として定められる。例えば、$\underline{X}, \overline{X}$ をそれぞれ区間 X の下限、上限として、

$$\begin{aligned} X + Y &= [\underline{X} \mathbin{\underline{+}} \underline{Y}, \overline{X} \mathbin{\overline{+}} \overline{Y}] \\ X - Y &= [\underline{X} \mathbin{\underline{-}} \overline{Y}, \overline{X} \mathbin{\overline{-}} \underline{Y}] \\ X \times Y &= [\min(\underline{X} \mathbin{\underline{\times}} \underline{Y}, \underline{X} \mathbin{\underline{\times}} \overline{Y}, \overline{X} \mathbin{\underline{\times}} \underline{Y}, \overline{X} \mathbin{\underline{\times}} \overline{Y}), \max(\underline{X} \mathbin{\overline{\times}} \underline{Y}, \underline{X} \mathbin{\overline{\times}} \overline{Y}, \overline{X} \mathbin{\overline{\times}} \underline{Y}, \overline{X} \mathbin{\overline{\times}} \overline{Y})] \\ X \div Y &= [\min(\underline{X} \mathbin{\underline{\div}} \underline{Y}, \underline{X} \mathbin{\underline{\div}} \overline{Y}, \overline{X} \mathbin{\underline{\div}} \underline{Y}, \overline{X} \mathbin{\underline{\div}} \overline{Y}), \max(\underline{X} \mathbin{\overline{\div}} \underline{Y}, \underline{X} \mathbin{\overline{\div}} \overline{Y}, \overline{X} \mathbin{\overline{\div}} \underline{Y}, \overline{X} \mathbin{\overline{\div}} \overline{Y})] \quad (Y \not\ni 0) \\ \mathrm{sqrt}(X) &= [\underline{\mathrm{sqrt}}(\underline{X}), \overline{\mathrm{sqrt}}(\overline{X})] \quad (X \geq 0) \end{aligned}$$

とすれば良い。ただし、各 $+, -, \times, \div, \mathrm{sqrt}$ のアンダーライン、オーバーラインは、それぞれ下向き丸め、上向き丸めを表す。区間演算が定義されているようなこの二項演算と単項演算の組合せで書かれた全ての関数 f について、区間演算を行えばその関数の値域の評価を行うことが出来る。す

なわち、f の定義域に含まれる区間 X に対して、その関数を構成する全ての二項演算、単項演算に対して区間演算を行うことにより得られた結果を $f(X)$ とすると、

$$f(X) \supset \{f(x) \mid x \in X\}$$

となることは明らかである。

例えば、10 進数で仮数部の有効数字 3 桁の浮動小数点数で、

$$(1 \div 3) \times 3$$

を計算してみよう。区間演算を使わない普通の計算では、

$$\begin{aligned} 1 \div 3 &\rightarrow 0.333 \\ 0.333 \times 3 &\rightarrow 0.999 \end{aligned}$$

と、誤差が混入する。区間演算だと、

$$\begin{aligned} [1,1] \div [3,3] &\rightarrow [0.333, 0.334] \\ [0.333, 0.334] \times [3,3] &\rightarrow [0.999, 1.01] \end{aligned}$$

のようになる。最初は幅 0 の区間 ($[1,1]$、$[3,3]$) からスタートして徐々に区間の幅が広がってくるが、常に真の値を含みながら計算が行われる。このような計算により、区間演算は丸め誤差の把握を行うことが出来る。

なお、IEEE 754 Std. に従う浮動小数点数を $\mathbb{F}$ として用いる場合、浮動小数点演算の丸めの向きを制御する「丸めモード」を変更できる場合がある。このとき、例えば加算ならば、丸めを $-\infty$ 方向に変更してから $\underline{X} + \underline{Y}$ を計算、丸めを $+\infty$ 方向に変更してから $\overline{X} + \overline{Y}$ を計算、という手順で所望の結果を得ることが出来る。但し、これが可能なのは加算、減算、乗算、除算、平方根の 5 演算のみであり、それ以外の演算 (exp など) は計算機の持っている機能に頼らず自力で実装する必要がある。

2.2 区間演算の過大評価

区間演算は非常に簡単な原理により丸め誤差の把握を行なうことが出来る。ただ、真値は必ず含むものの、想像以上に区間の幅が広がってしまうことも多い。このような過大評価のメカニズムについて説明する。

例として、関数

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

を区間 $[-0.1, 0.1]$ で評価する問題を考える [1]。この関数に $[-0.1, 0.1]$ を代入して区間演算を行い、その途中経過を記すと、

$$\begin{aligned} x+1 &\rightarrow [0.9, 1.1] \\ (x+1)^2 &\rightarrow [0.81, 1.21] \\ 2x &\rightarrow [-0.2, 0.2] \\ (x+1)^2 - 2x &\rightarrow [0.61, 1.41] \end{aligned}$$

[1] この関数は展開して整理しまえば $x^2 + 1$ であるが、それをせずにこの表記のままで計算する問題を考える。

となる。しかし、真の像は、

$$\{f(x) \mid x \in [-0.1, 0.1]\} = [1, 1.01]$$

であり、区間演算の結果は確かに真の像を含むものの区間幅が真値の 80 倍にも膨らんでしまっていることが分かる。この計算は誤差無しで行っているので、この膨らみの原因が数値誤差でないことは明らかである。$(x+1)^2$、$2x$ の像は真の像と一致しているので、最後の減算の部分がこの膨らみの原因である。$(x+1)^2$ と $2x$ はどちらも $[-0.1, 0.1]$ あたりで傾きが 2 に近い、非常に傾きの似通った関数であり、それらの関数同士の差は変動が打ち消し合って定数関数に近くなる。この様子を図 1 に示す。しかし、区間演算はその強い相関を知らないため、幅の大きな区間を生成してし

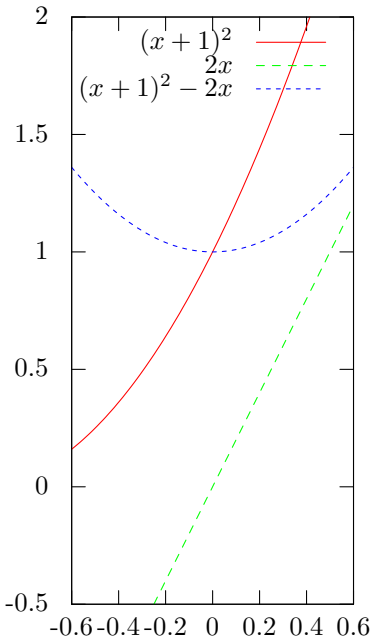

図 1: 傾きの近い関数同士の減算

まう。

この現象を防ぐためには、区間は他の変数との相関の情報を持っている必要がある。Affine Arithmeric は、それを実現する方法の一つである。

3 Affine Arithmetic

3.1 Affine 形式

Affine Arithmetic は、変数間の相関性を考慮することにより区間演算の over-estimation の問題を解決する方法の一つである。この方法は文献 [1] で提案されたものである。また、文献 [2] の方法を簡略化したものと見ることも出来る。

Affine Arithmetic では、変動範囲が $-1 \leq \varepsilon_k \leq 1$ であるようなダミー変数 ε_k を用いて、その線形結合

$$a_0 + a_1\varepsilon_1 + \cdots + a_n\varepsilon_n$$

の形 (Affine 形式) で数 (区間) を表現する。計算機にはその係数 $a_0, \ldots, a_n$ を記憶する。なお、後述するように ε_k の最大数 n は計算の途中で変化する。

例えば、Affine 形式 x, y があって、それが

$$\begin{aligned} x &= 1 + 0.5\varepsilon_1 \\ y &= 1 + 0.5\varepsilon_2 \end{aligned}$$

であったとしよう。これは x と y に相関が無い状態で、このとき (x, y) が取り得る領域は図 2 のようになる。

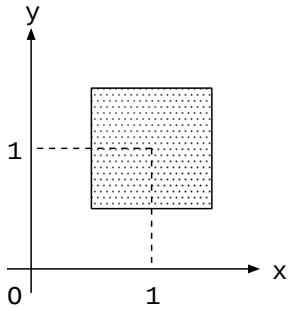

図 2: x と y に相関が無い場合

これに対して、

$$\begin{aligned} x &= 1 + 0.5\varepsilon_1 \\ y &= 1 + 0.4\varepsilon_1 + 0.1\varepsilon_2 \end{aligned}$$

では、x, y それぞれが取り得る範囲は $[0.5, 1.5]$ で変っていないが、ε_1 の係数を見ると分かるように両者には強い相関があり、(x, y) の取り得る領域は図 3 のようになる。

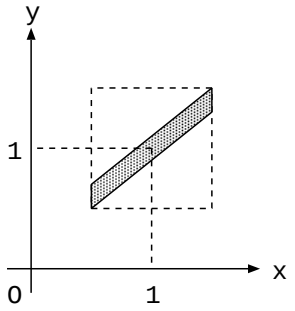

図 3: x と y に相関がある場合

また、この領域は、

$$\begin{aligned} x &= 1 + 0.5\varepsilon_1 \\ y &= 1 + 0.4\varepsilon_1 + 0.1\varepsilon_2 \end{aligned}$$

を

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.4 \end{pmatrix} \varepsilon_1 + \begin{pmatrix} 0 \\ 0.1 \end{pmatrix} \varepsilon_2$$

のような縦ベクトルの和に分解し、

- 点 $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- 係数ベクトル $\begin{pmatrix} 0.5 \\ 0.4 \end{pmatrix}$ を $-1 \sim 1$ 倍して得られる線分
- 係数ベクトル $\begin{pmatrix} 0 \\ 0.1 \end{pmatrix}$ を $-1 \sim 1$ 倍して得られる線分

のミンコフスキー和 (Minkowski sum) と考えることも出来る (図 4) 。

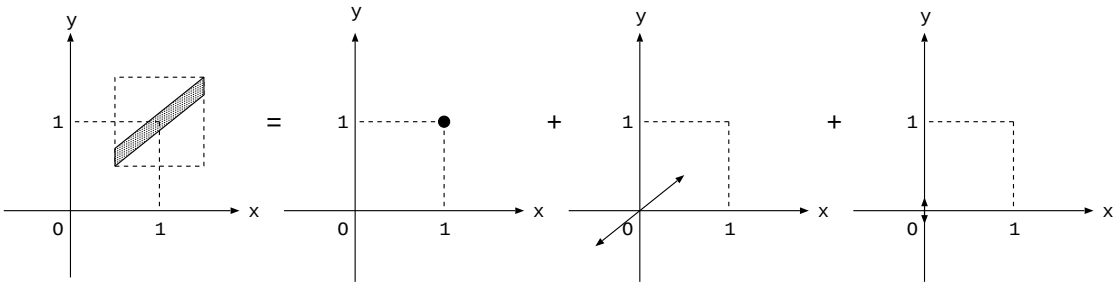

図 4: 集合のミンコフスキー和

n 変数関数 $f(x_1, \ldots, x_n)$ を Affine Arithmetic で評価する場合、最初に与えられる n 個の変数は一般に互いに無相関であるから、n 個の ε を用いて

$$\begin{aligned} x_1 &= \frac{\overline{x_1} + \underline{x_1}}{2} + \frac{\overline{x_1} - \underline{x_1}}{2}\varepsilon_1 \\ x_2 &= \frac{\overline{x_2} + \underline{x_2}}{2} + \frac{\overline{x_2} - \underline{x_2}}{2}\varepsilon_2 \\ &\vdots \\ x_n &= \frac{\overline{x_n} + \underline{x_n}}{2} + \frac{\overline{x_n} - \underline{x_n}}{2}\varepsilon_n \end{aligned}$$

のような affine 形式で初期化しておく。但し、入力変数 x_k の変域を $[\underline{x_k}, \overline{x_k}]$ とする。

なお、Affine 形式

$$x = a_0 + a_1\varepsilon_1 + \cdots + a_n\varepsilon_n$$

は、

$$[a_0 - \delta, a_0 + \delta], \quad \delta = \sum_{i=1}^{n} |a_i|$$

によっていつでも通常の区間に戻すことが出来る。

3.2 Affine 演算における丸め誤差

以下、Affine 形式同士の演算方法について述べる。ただし、まずは簡単のため、演算の過程で発生する丸め誤差を考慮しない形で説明することにする。例えば Affine 形式の係数が有理数表現されているならば、以下の説明の方法はそのまま実現出来る。

3.3 線形演算

Affine 形式の変数における演算は、加減算及び定数倍の場合は自明である。

Affine 多項式 x, y:

$$\begin{aligned} x &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ y &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n \quad , \end{aligned}$$

に対して、加算、減算は次のように定義する。

$$\begin{aligned} x \pm y &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n \\ x \pm \alpha &= (x_0 \pm \alpha) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \quad . \end{aligned}$$

定数の乗算は次のように定義する。

$$\alpha x = (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n \quad .$$

3.4 非線形単項演算

Affine 多項式

$$x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$

に対する非線形な演算 f について、$z = f(x)$ は一般に affine 多項式で表すことは出来ない。そこで、f を線形演算で近似し、近似誤差を新しいダミー変数 ε_{n+1} を導入することによって表すことを考える。

まず、x の変域 I を

$$I = [x_0 - \delta, x_0 + \delta], \quad \delta = \sum_{i=1}^{n} |x_i| \quad ,$$

で求める。次に、この領域 I において f をなるべくよく近似するような一次関数 $ax + b$ を求める (図 5 参照)。誤差の最大値

$$\delta = \max_{t \in I} |f(t) - (at + b)|$$

を求め、これをダミー変数 ε_{n+1} の係数とする。すなわち、非線形関数 f は区間 I において

$$f(x) \in ax + b + \delta\varepsilon_{n+1}$$

であり、よって単項演算の結果は

$$a(x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n) + b + \delta\varepsilon_{n+1}$$

とすればよい。これはダミー変数が一つ増加した Affine 形式である。このように定義された単項演算は、追加されたダミー変数の大きさが最小であるという意味で、最適である。

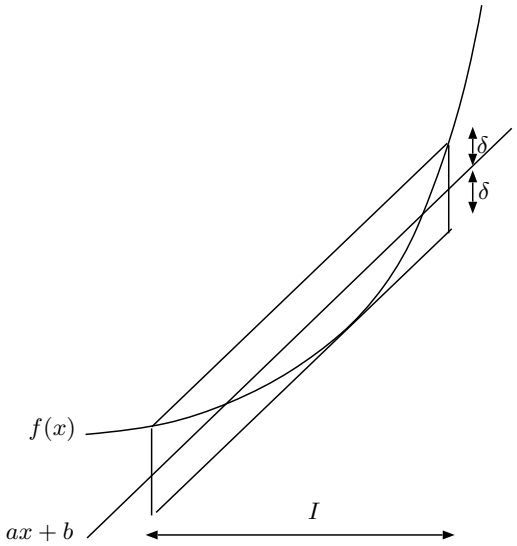

図 5: 非線形関数の線形近似

3.5 非線形二項演算

Affine 多項式 x, y:

$$\begin{aligned} x &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ y &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n \quad , \end{aligned}$$

に対する非線形な二項演算 $z = f(x, y)$ について考える。例えば、$x \times y$, x/y, x^y などである。

単項演算の場合と同様に、x と y の変域 X, Y を求め、$(x, y) \in X \times Y$ において $f(x, y)$ を最良近似する一次式 $ax + by + c$ を求め、$ax + by + c + \delta\varepsilon_{n+1}$ を結果とする方法を使うことは出来る。しかし、単項演算の場合と違ってこの方法は必ずしも最良の近似を与えるとは限らない。これは、x と y に相関性がある場合、点 (x, y) は一般に $X \times Y$ で与えられる長方形領域内の全ての点を取るとは限らないためである。

乗算の場合、変域 X, Y の半径を δ_x, δ_y とすると $x \times y$ は

$$y_0x + x_0y - x_0y_0$$

で最良近似され、誤差は $\delta_x\delta_y$ となるので、

$$\begin{aligned} z &= y_0x + x_0y - x_0y_0 + \delta_x\delta_y\varepsilon_{n+1} \\ &= x_0y_0 + \sum_{i=1}^{n}(y_0x_i + x_0y_i)\varepsilon_i \\ &\quad +(\sum_{i=1}^{n}|x_i|)(\sum_{i=1}^{n}|y_i|)\varepsilon_{n+1} \end{aligned}$$

が用いられることが多い。前述したように、これは最適では無い。

除算は、$x/y = x \times (1/y)$ のように分解し、逆数関数と乗算の組み合わせによって計算することが多い。これも最適では無い。

4 Affine Arithmetic の簡単な例

第 2.2 節で扱った問題を再度取り上げる。関数 f を

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

x の変域を $[-0.1, 0.1]$ としたときの、f の値域を Affine Arithmetic で計算してみる。

- $x = 0 + 0.1\varepsilon_1$ で初期化。
- $x + 1 = 1 + 0.1\varepsilon_1$ 。
- $*^2$ の入力の変域は、$x+1$ を区間に直した $[0.9, 1.1]$。領域 $[0.9, 1.1]$ における関数 $*^2$ の最良近似を求めると、$2 * -0.995$ であり、最大誤差は 0.005。誤差を表すダミー変数 ε_2 を導入して、$(x+1)^2$ の計算結果は、

  $$\begin{aligned} & 2(1 + 0.1\varepsilon_1) - 0.995 + 0.005\varepsilon_2 \\ & = 1.005 + 0.2\varepsilon_1 + 0.005\varepsilon_2 \end{aligned}$$

  とする。
- $2x = 0 + 0.2\varepsilon_1$
- 最終結果は、

  $$\begin{aligned} f &= 1.005 + 0.2\varepsilon_1 + 0.005\varepsilon_2 - (0 + 0.2\varepsilon_1) \\ &= 1.005 + 0.005\varepsilon_2 \end{aligned}$$

  となる。これを区間に直せば、$[1, 1.01]$ が f の値域の評価。

同じ手順で区間演算で計算すると、第 2.2 節で述べた通り $[0.61, 1.41]$ という幅の大きな区間が生成されてしまう。Affine Arithmetic が、$(x+1)^2$ と $2x$ の強い相関を ε_1 の係数が近いという情報によってうまく扱えていること、またそれによって最終結果の区間幅が大きく改善されていることがよく分かる。

5 丸め誤差を考慮した Affine Arithmetic

計算機で実数をそのまま扱うことは出来ず浮動小数点数しか扱えないのが普通なので、前節のアルゴリズムをそのまま無誤差で実行することは出来ない。区間演算の場合は、単に「外側に」丸めることによって、真の変域を数学的に厳密に包含する性質を失うこと無く計算機上に実装することが出来る。しかし、Affine Arithmetic においては、ε の係数を上向きに丸めても下向きに丸めても厳密性が失われてしまう。

以下、浮動小数点数を用いて Affine Arithmetic を数学的に厳密に実装する方法を示す。3 通りの方法を示すが、これらには計算速度と性能 (狭い包含を得られるか) のトレードオフがある。

5.1 方法 1(最も高性能な方法)

丸め誤差の影響で、加減算及び定数乗算すら正確に実行することは不可能である。そこで、これらの線形演算をも非線形演算と扱い、ダミー変数を追加することが考えられる。

例えば、線形計算の真の結果を

$$x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$

計算値を

$$f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n$$

とする。ここで、各係数の誤差 $\delta_i = x_i - f_i$ が何らかの方法で見積もれたとすると、

$$\begin{aligned} x &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + \delta_0 + \delta_1\varepsilon_1 + \cdots \delta_n\varepsilon_n \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots |\delta_n|)\varepsilon_{n+1} \end{aligned}$$

となり、丸め誤差を表現するための新しいダミー変数 ε_{n+1} を導入してその係数を $\sum_{i=0}^{n} |\delta_i|$ の上界とすれば良い。

非線形計算は次のようにする。単項演算 f に対して、その線形近似 $ax + b$ を a, b がともに浮動小数点という条件で作成し、x の変域 I におけるその誤差の上限を δ とする。単項演算の結果は

$$\begin{aligned} x &= a(x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n) + b + \delta\varepsilon_{n+1} \\ &= ax_0 + b + ax_1\varepsilon_1 + \cdots + ax_n\varepsilon_n + \delta\varepsilon_{n+1} \end{aligned}$$

であるが、その計算値を

$$f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + \delta\varepsilon_{n+1}$$

とする。誤差を $\delta_0 = ax_0 + b - f_0$, $\delta_i = ax_i - f_i$ とすると、線形の場合と同様にして、

$$\begin{aligned} x &= ax_0 + b + ax_1\varepsilon_1 + \cdots + ax_n\varepsilon_n + \delta\varepsilon_{n+1} \\ &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n + \delta\varepsilon_{n+1} \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + \delta_0 + \delta_1\varepsilon_1 + \cdots \delta_n\varepsilon_n + \delta\varepsilon_{n+1} \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots |\delta_n| + \delta)\varepsilon_{n+1} \end{aligned}$$

となる。このように、線形計算部分で発生した誤差は非線形計算用に追加した ε_{n+1} に加算することによって新たに ε を追加する必要はない。この方法は、丸め誤差もきちんと ε を使って取り扱うことによって、計算途中に入った丸め誤差がその後の計算で増大することを防げる。高性能であるが、単なる線形計算でも ε が一つ増加するため、計算ステップが長くなると計算時間がかかってしまうことが予想される。

丸め誤差 δ_i の推定は、いったん各係数を区間演算で計算し、その中心を 計算値 f_i 、半径を誤差 δ_i とする方法が考えられる。

5.2 方法 2(丸め誤差専用ダミー変数を導入する方法)

方法 1 は最も高性能であるが、ε が増加し過ぎるという問題点がある。そこで、丸め誤差を格納する専用のダミー変数 ε_r を全ての Affine 多項式に追加することを考える。

$$x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n + \delta_r\varepsilon_r$$

このダミー変数 ε_r は Affine 形式で表された全ての変数に対して、別の変数とみなす。すなわち、全ての Affine 変数は、それぞれ固有の ε_r を持つ。また、ε_r の係数は常に正とする。例えば、Affine 変数

$$\begin{aligned} x &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n + x_r\varepsilon_r \\ y &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n + y_r\varepsilon_r \end{aligned}$$

の和は、

$$\begin{aligned} x + y &= (x_0 + y_0) + (x_1 + y_1)\varepsilon_1 + \cdots + (x_n + y_n)\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + +\delta_0 + \delta_1\varepsilon_1 + \cdots + \delta_n\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots + |\delta_n| + x_r + y_r)\varepsilon_r \end{aligned}$$

のように計算する。差は、

$$\begin{aligned} x - y &= (x_0 - y_0) + (x_1 - y_1)\varepsilon_1 + \cdots + (x_n - y_n)\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + +\delta_0 + \delta_1\varepsilon_1 + \cdots + \delta_n\varepsilon_n + (x_r + y_r)\varepsilon_r \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots + |\delta_n| + x_r + y_r)\varepsilon_r \end{aligned}$$

のようにする。このように、x と y に含まれている ε_r は「別の変数」と見なし、減算においてもキャンセルしないことに注意。定数倍は、

$$\begin{aligned} ax &= ax_0 + ax_1\varepsilon_1 + \cdots + ax_n\varepsilon_n + |a|x_r\varepsilon_r \\ &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n + |a|x_r\varepsilon_r \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + +\delta_0 + \delta_1\varepsilon_1 + \cdots + \delta_n\varepsilon_n + |a|x_r\varepsilon_r \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots + |\delta_n| + |a|x_r)\varepsilon_r \end{aligned}$$

となる。

非線形演算においては、

$$\begin{aligned} x &= ax_0 + b + ax_1\varepsilon_1 + \cdots + ax_n\varepsilon_n + |a|x_r\varepsilon_r + \delta\varepsilon_{n+1} \\ &= (f_0 + \delta_0) + (f_1 + \delta_1)\varepsilon_1 + \cdots + (f_n + \delta_n)\varepsilon_n + |a|x_r\varepsilon_r + \delta\varepsilon_{n+1} \\ &= f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + \delta_0 + \delta_1\varepsilon_1 + \cdots\delta_n\varepsilon_n + |a|x_r\varepsilon_r + \delta\varepsilon_{n+1} \\ &\subset f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots|\delta_n| + |a|x_r + \delta)\varepsilon_{n+1} + 0\varepsilon_r \end{aligned}$$

のように ε_r 相当分は ε_{n+1} に組み入れ、ε_r の係数は 0 にする。

この方法は方法 1 には劣るが、ε の増加を非線形演算のみに抑えており、計算速度に配慮した方法と言える。

5.3 方法 3(最も高速な方法)

方法 2 で導入した丸め誤差専用の ε_r の使用範囲を更に拡大し、非線形演算において追加される ε も ε_r で面倒をみてしまおうという方法である。すなわち、方法 2 の非線形演算を、

$$f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + (|\delta_0| + |\delta_1| + \cdots |\delta_n| + |a|x_r + \delta)\varepsilon_r$$

のようにしてしまう。

この方法では、もはや ε の追加は発生しないので高速であるが、Affine Arithmetic らしい性能はもはや発揮しないものと推測される。

5.4 3 手法のまとめ

3 手法は、まとめると、

	方法 1	方法 2	方法 3
線形計算における ε の追加	あり	なし	なし
非線形計算における ε の追加	あり	あり	なし
特殊ダミー変数 ε_r の使用	なし	あり	あり
包含性能	◎	○	△
計算速度	△	○	◎

という違いがある。

6 Affine Arithmetic の性能評価

以下、実例により Affine Atrithmetic の性能評価を行う。

6.1 人工的な数値例

Affine Arithmeric の性能が顕著に見えるような人工的な例題を示す。

$$\begin{aligned} f(x) &= x^2 - 2x \\ g(x) &= x(x+1)(\frac{1}{x} - \frac{1}{x+1}) \\ &\text{evaluate } f(g(x)) \\ x &\in 10000 + [-\varepsilon, \varepsilon] \end{aligned}$$

$g(x) = 1$ なので $f(g(x)) = -1$(定数関数) であるが、この数式の手順の通りで計算するものとする。

これを、

- 区間演算
- 平均値形式
- Affine Arithmetic (方法 1, 2, 3)

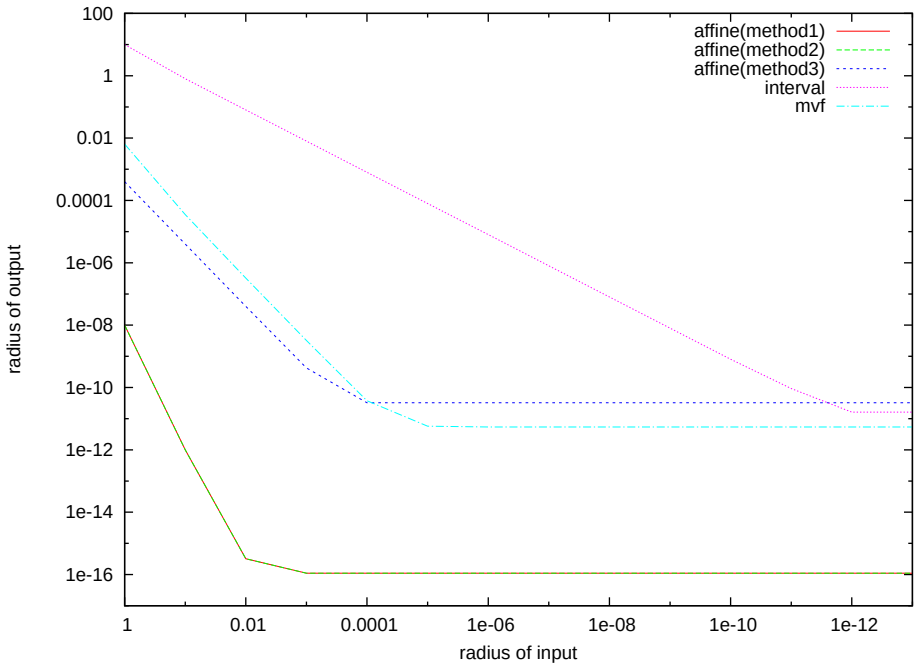

図 6: 区間幅の比較

で入力区間幅を変化させながら計算した。出力区間幅との関係をグラフにしたものを図 6 に示す。

これを見ると、区間演算が一番悪く、平均値形式は入力区間幅を小さくするにつれて出力区間幅は改善しているが、10^{-11} 程度で改善が止まってしまうことが分かる。Affine Arithmetic の方法 1,2 は、10^{-16} まで改善している。Affine Arithmetic の方法 3 は、ほぼ平均値形式と同様の傾向を示している。

この結果は、次の表で説明できる。

	区間演算	平均値形式、方法 3	方法 1,2
入力変数の区間幅に関する相関	×	○	○
中間変数に入った誤差の相関	×	×	○

中間変数に入る誤差に由来する出力の誤差が 10^{-11} 程度存在することが観察出来る。Affine Arithmetic の方法 1,2 は中間変数に入った誤差を取り扱うためのダミー変数の追加を行うため、中間変数に入った誤差を軽減することが可能となっていることが分かる。

すなわち、入力区間幅が 0 であるような関数評価を改善するには、中間変数に混入する誤差を取り扱う必要があり、このような問題に対しては平均値形式や Affine Arithmetic の方法 3(ダミー変数の追加が無い) は無力である。

6.2 Henon Map

Henon Map [3] と呼ばれる、chaotic な挙動で知られる 2 次元力学系

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 - ax_i^2 + y_i \\ bx_i \end{pmatrix}$$

を例題とする。$b = 0.3$ のとき、$a \geq 1.06$ で chaos が発生することが知られている。ここでは、chaos 発生直前の $a = 1.05$ とし、区間幅の広がり方を調べる。

まず、初期値を

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} [-10^{-5}, 10^{-5}] \\ [-10^{-5}, 10^{-5}] \end{pmatrix}$$

とし、軌道を

- 区間演算
- Affine Arithmetic (方法 1)
- Affine Arithmetic (方法 2)
- Affine Arithmetic (方法 3)

で計算した例を図 7 に示す。横軸は反復回数、縦軸は区間幅 (x, y のうち大きい方) である。

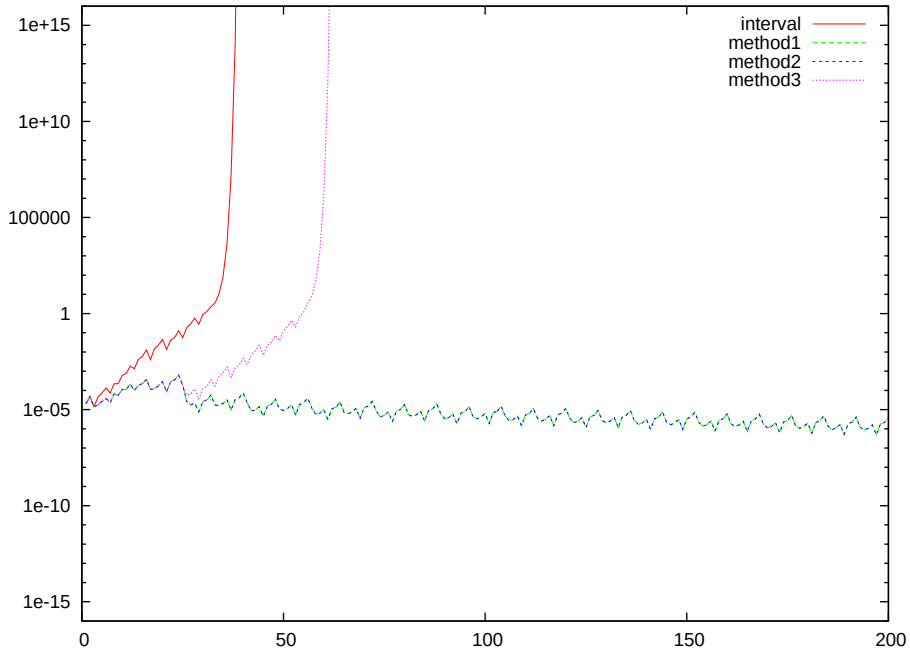

図 7: Henon Map の軌道の区間幅 (初期区間幅 $= 2 \times 10^{-5}$)

これを見ると、区間演算はすぐに発散、方法 1 と方法 2 は発散せずにきちんと幅を保っていることが分かる。方法 3 は反復の初期段階こそうまく行っているように見えるが、途中で発散しはじめる。途中で入る誤差をきちんと扱っていないので、初期区間幅に対して途中で混入する誤差が有意な大きさを持つ段階になると、区間演算と同様に発散し始めてしまっていると考えられる。計算時間を、表 1 に示す。CPU は core i7 2640M (2.8G)、OS は ubuntu 10.04 LTS (64bit) である。

次に、初期区間幅を 0 にして同様の計算を行った結果を図 8 に示す。

これだと、方法 3 は区間演算と同様、発散してしまい、Affine Arithmetic の意味が無くなっている。初期区間にしか ε が割り当てられないので、これは予想通りの結果と言える。方法 1,2 は狭い区間を保ったまま計算出来ている。

区間演算	0.017
Affine Arithmetic (方法 1)	1.969
Affine Arithmetic (方法 2)	0.613
Affine Arithmetic (方法 3)	0.112

表 1: 計算時間 (単位: msec)

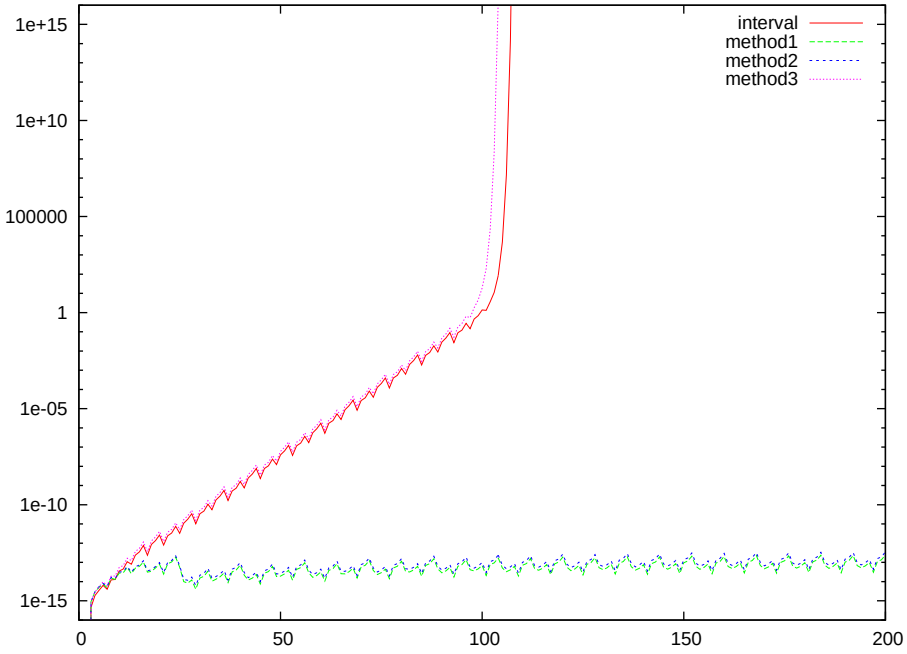

図 8: Henon Map の軌道の区間幅 (初期区間幅 = 0)

幅あり、幅 0 のどちらでも、方法 1 と方法 2 の間に大きな差は見られなかった。

6.3 三項間漸化式

Affine Arithmetic における丸め誤差の扱いについて、ここまで挙げた例では方法 3 が劣るのはよく分かるものの、方法 1 と方法 2 に顕著な差は見られなかった。方法 2 では線形演算において入る誤差をラフに扱っているので方法 1 より劣るはずである。しかし、その誤差も非線形演算が出現すればきちんと非線形演算で追加された ε に反映されて相関の考慮の対象になるため、劣化はほとんど見られないと考えられる。逆に言えば、線形演算ばかりで構成されていれば、方法 2 の方法 1 に対する劣化がはっきり見えるはずである。

そこで、そのような例題を一つ示す。数列 $\{x_n\}$ を、

$$x_{n+2} = 3x_{n+1} - 2x_n$$

で定める。初期値は $x_0 = x_1 = 0.9$ とする。真の解は全ての n について $x_n = 0.9$ と定数である。この数列を、$n = 30$ まで計算してみる。

まず `double` で計算すると、

$$x_{30} = 0.89999997019767797\ldots$$

となる。区間演算を行なうと、

$$x_{30} = [-0.074710728957010364, 1.8747106991546883]$$

と、実際に入った誤差と比べてもかなりの過大評価となる。これを、Affine Arithmetic の方法 1, 2, 3 で計算してみると、

$$\begin{aligned} x_{30} &= [-0.55613991960628062, 2.3561393831644795] \quad (\text{方法 3}) \\ x_{30} &= [-0.55613991960628062, 2.3561393831644795] \quad (\text{方法 2}) \\ x_{30} &= [0.89999907612800844, 0.90000038743019018] \quad (\text{方法 1}) \end{aligned}$$

のようになった。方法 3 のみならず方法 2 でも過大評価が起こってしまっているが、方法 1 はタイトな評価が出来ていることが分かる。

しかし、一般に Affine Arithmetic で評価したい関数は多くの非線形演算を含んでいることが多いであろうことから、計算コストを考えると、この例のような特殊な場合を除けば方法 2 が最も勧められる。

7 Affine Arithmetic の改良

前述したように、Affine Arithmetic の二項演算は、追加されるダミー変数の係数が必ずしも最小でないという意味で、最適では無い。最適な乗算を与える方法として [4] が、最適では無いがより良い除算を与える方法として [5] がある。

参考文献

[1] Marcus Vinícius A. Andrade, João L. D. Comba and Jorge Stolfi: “Affine Arithmetic”, INTERVAL’94, St. petersburg (Russia), March 5-10, 1994.

[2] Masahide Kashiwagi : “Interval Arithmetic with Linear Programming — Extension of Yamamura’s Idea —”, Proc. 1996 International Symposium on Nonlinear Theory and its Applications (NOLTA’96 Symposium), pp.61–64 (Kochi, Japan, October 7–9, 1996).

[3] M. Henon : “A two-dimensional mapping with a strange attractor”, Communications in Mathematical Physics 50 (1), pp.69–77 (1976).

[4] 宮島 信也, 宮田 孝富, 柏木 雅英 : “アフィン演算における最良乗算について”, 電子情報通信学会論文誌 (A), Vol.J86-A, No.2, pp.150–159 (2003.2).

[5] 宮島 信也, 宮田 孝富, 白井 健一, 柏木 雅英 : “アフィン演算における乗除算について”, 電子情報通信学会論文誌 (A), Vol.J86-A, No.3, pp.232–240 (2003.3).