数理解析研究所講究録
第831巻 1993年 53-72

区間解析と有理数演算による非線形方程式の近似解の精度保証

柏木 雅英 † 大石 進一 ‡
Masahide Kashiwagi Shin'ichi Oishi
† 早稲田大学 理工学部 電子通信学科
‡ 早稲田大学 理工学部 情報学科
E-mail : † kashi@oishi.info.waseda.ac.jp
‡ oishi@oishi.info.waseda.ac.jp

1 はじめに

非線形方程式

$$f(x)=0, \quad f:\boldsymbol{R}^n \to \boldsymbol{R}^n \tag{1}$$

の解 (零点) を求める場合，解析的に求まる場合は稀であり，計算機によって数値的に解を計算することが要求される．非線形方程式を数値的に解くためのアルゴリズムは数多く研究されているが，通常の浮動小数点演算は常に誤差を伴うため，その計算結果は誤差を含むものとなる．近年，区間演算と不動点定理を用いてこのような数値計算の誤差を厳密に把握できることが知られるようになり，このような方法は精度保証付き数値計算と呼ばれる．本稿では，従来の数値計算で求まった近似解を元にして，精度保証を行う方法を提案する．全ての計算を精度保証付き数値計算に置き換えるのでなく，従来法を補完して精度保証を行う方がより実用的であるという考えに基づいている．

非線形方程式の解を計算機上に表現する場合，無限精度の実数値を計算機上で得ることは不可能であるが，計算機上に実現されたプログラムによって，“任意に高精度の値が得られる” 状態を実現することは可能である．例えば，計算機で円周率の値を無限桁求めることは言うまでもなく不可能である．しかし，任意に高精度の値を求められるようなアルゴリズムを我々は既に知っており，それを計算機上に実現できる．そのため我々は円周率の値を “知っている” と認識できると考えられ，方程式の解についても円周率と同様の状態になって初めて “精度保証” されたと考える．

ここでは，“ $a_i \le x \le b_i$, $b_i - a_i \to 0$ を満たす列 $\{a_i, b_i\}$ を計算するプログラムが計算機上に実現” されたとき，実数 x は精度保証されたと考えることにする．すなわち，方程式の解が “精度保証された” ということは，区間 $[a_i, b_i]$ に必ず真の解が含まれていることの保証と，いくらでも精度の良い解を得ることが出来ることの保証という二重の意味を持っている．

注意 1.1 このようなプログラムを，その実数の計算機上の表現と考えることもできる．このようなプログラムが作れる実数は計算可能実数と呼ばれ，構成的実数論と深い関連があるが，ここでは触れない． □

方程式の解についてこのようなプログラムが得られたとき，$[a_0, b_0]$ を計算すればそれが真の解を含む区間解であり，区間幅が誤差評価となる．誤差が大きすぎる場合は，$[a_1, b_1], [a_2, b_2] \cdots$ と順に計算していけば，いくらでも精度の高い解が得られる．すなわち，品質保証のみならずアフターサービスも万全という訳である．1変数の方程式の場合このようなプログラムはいわゆる二分法によって容易に実現できる (但し，後述する方程式の表現誤差の問題を考えるとそう簡単ではない)．2変数以上の場合は，各成分について解を包含するような縮小区間列を計算することになるが，これは簡単ではなく，縮小写像原理などの不動点定理を用いる必要がある．

上の意味での精度保証を行う上で，現状の数値解析法を考えると次の二つの問題点が浮上してくる．

一つは，計算に用いる数値表現体系である．現状の計算機の多くは，実数の計算機上の表現として浮動小数点方式を採用している．この方法は，メモリ上に占める領域が一定で絶対値の大きい数値が扱えるという利点があり，ハードウェア化することにより極めて高速な演算が実現されている．ところが，計算の過程で混入する誤差の見積もりが難しいという欠点があり，また当然表現できる精度には限界があるため，精度保証には適さない．本稿では，精度保証に適した数値表現体系として，分母分子を多倍長の整数として持つ有理数表現を採用する．この方法では，加減乗除において誤差が混入しないため誤差のない線形計算を仮定でき，精度保証のための理論が作りやすい．計算が進むにつれて桁の爆発が起こりやすいという欠点はあるが，アルゴリズムに適切な丸めを入れることによってそれを防ぐ工夫をする．実際，有理数に対して，連分数展開の手法を用いることにより誤差の大きさと方向を指定した丸めを効果的に行うことが出来る．更に，区間演算では計算途中の任意の時点で丸め (両端の数を外側へ丸める) が行えるため，有理数演算を行った場合の速度の低下を比較的抑え易い．

二つ目の問題点は，解析の対象となる非線形方程式の計算機上の表現方法である．方程式の表現が誤差を含んでいれば当然精度保証は出来ない．有理数演算を導入すれば加減乗除だけで構成されている方程式は厳密に表現できるが，$\sqrt{}$，log などの関数が含まれていれば厳密な表現はできない．すなわち，このような場合は誤差評価付きで関数が表現されている必要がある．更に，精度保証された解の反復改良の可能性を保証するためには，任意に高精度に関数値が計算できる必要がある．本稿では，誤差評価付きの関数表現として関数の区間包囲を採用する．また，区間包囲は任意に高精度で行えることを仮定する．すなわち，計算過程において使用される数値の精度，及び級数展開の打ち切り項数を制御できるとする．

また，それらとは別の問題点として，アルゴリズムの実現容易性がある．精度保証付き数値計算が実用になるためには，その利用が比較的容易である必要がある．また，その手法を利用するために事前に多くの検討を行う必要があれば，その段階での間違いの混入が防げなくなり，計算結果の信頼性の低下を招くことになる．すなわち，実用的な精度保証アルゴリズムは，方程式の記述が容易で，またなるべく自動的に解析を行うようなシステムが構築可能でなければならない．

本稿では，上で述べたような数値表現及び関数表現を仮定し，適当な方法によって得られた近似解に対して，精度保証を行うアルゴリズムを提案する．これは，Krawczyk の区間写像を用いた解の包み込みを利用するものである．また，ある意味でのこのアルゴリズムの妥当性を示す．最後に，精度保証システムの

実際の実現と，シミュレーションについて述べる．ここで，方程式の記述が容易で，また自動的に解析を行うようなシステムが構築可能であることを示す．

2 区間演算

本節では，本稿で用いる区間演算の用語について説明する．区間演算は，計算機で実数が厳密に表現できないため，計算機内での数値をある幅を持った区間として扱い，真値の取りうる全ての値を区間内に包含しながら計算を進めようという考えから生まれた．

定義 2.1 (区間，区間ベクトル，区間行列) $x \leq y$ を満たす実数 x, y で定まる $\boldsymbol{R}$ の部分集合 $\{r \mid x \leq r \leq y\}$ を区間といい，$[x, y]$ と表す．また，成分として区間を持つようなベクトル，行列をそれぞれ区間ベクトル，区間行列という．n 次元区間ベクトル $([x_1, y_1], \cdots, [x_n, y_n])^t$ は，ベクトル $x = (x_1, \cdots, x_n)^t$，ベクトル $y = (y_1, \cdots, y_n)^t$ を用いて $[x, y]$ と表す．$n \times m$ 区間行列 $([a_{11}, b_{11}], \cdots, [a_{nm}, b_{nm}])$ についても同様．また，これらの区間ベクトル，区間行列は次のような部分集合を与える．

$$[x, y] = \{z = (z_1, \cdots, z_n)^t \mid z_i \in [x_i, y_i]\} \tag{2}$$

$$[a, b] = \{c = (c_{11}, \cdots, c_{nm}) \mid c_{ij} \in [a_{ij}, b_{ij}]\} \tag{3}$$

□

D を $\boldsymbol{R}$, $\boldsymbol{R}^n$ または $\mathcal{L}(\boldsymbol{R}^m; \boldsymbol{R}^n)$ ($n \times m$ 行列の作る線形空間) の部分集合としたとき，Dに含まれるような区間，区間ベクトル，区間行列の集合を $I(D)$ で表す．

区間 $I = [x, y] \in I(\boldsymbol{R})$ の中心，半径，絶対値を

$$\mathrm{mid}(I) = \frac{x + y}{2} \tag{4}$$

$$\mathrm{rad}(I) = \frac{y - x}{2} \tag{5}$$

$$|I| = \max(|x|, |y|) \tag{6}$$

で定義する．また，区間ベクトル，区間行列 Iの中心，半径，絶対値も，

$\mathrm{mid}(I)$ (各成分の中心によるベクトル，または行列)

$\mathrm{rad}(I)$ (各成分の半径による〃)

$|I|$ (各成分の絶対値による〃)

で定義する．例えば，区間行列 I に対して $\mathrm{rad}(I)$ は，I の各成分の半径を成分として持つような非区間行列である．

区間，区間ベクトル，区間行列の間の演算は，通常の演算 $* \in \{+, -, \times, /\}$ に対して

$$I_1 * I_2 = \{x_1 * x_2 | x_1 \in I_1, x_2 \in I_2\} \tag{7}$$

で定める．これは，実数同士の区間演算を実現すれば，ベクトル，行列に対しては通常の演算順序で各成分に対して実数同士の区間演算を行うことにより実現できる．

また，ベクトル，行列の半順序を次のように定義する．

定義 2.2 (ベクトル，行列の半順序) ベクトル $x, y \in \boldsymbol{R}^n$, $x = (x_1, \cdots, x_n)^t$, $y = (y_1, \cdots, y_n)^t$, 行列 $a, b \in \mathcal{L}(\boldsymbol{R}^m; \boldsymbol{R}^n)$, $a = (a_{11}, \cdots, a_{nm})$, $b = (b_{11}, \cdots, b_{nm})$ の半順序をそれぞれ $\omega \in \{\leq, <, \geq, >\}$ として

$$x \,\omega\, y \Leftrightarrow x_i \,\omega\, y_i \quad \text{for all } i \tag{8}$$

$$a \,\omega\, b \Leftrightarrow a_{ij} \,\omega\, b_{ij} \quad \text{for all } i, j \tag{9}$$

で定める． □

区間ベクトル及び区間行列のノルムは，最大値ノルムを一般化した次のノルムで定める．

定義 2.3 (scaled maximum norm) 与えられた scaling vector $u = (u_1, \cdots, u_n)^t \in \boldsymbol{R}^n$, $u > 0$ に対して，区間ベクトル $I \in I(\boldsymbol{R}^n)$ のノルムを

$$\|I\|_u = \max\{|I_i|/u_i \mid \text{for all } i\} \tag{10}$$

で，区間行列 $A \in I(\mathcal{L}(\boldsymbol{R}^n; \boldsymbol{R}^n))$ のノルムを

$$\|A\|_u = \max\{\|AI\|_u \mid \|I\|_u = 1\} \tag{11}$$

$$= \| \, |A| u \|_u \tag{12}$$

で定める． □

$u = (1, 1, \cdots, 1)^t$のときは，通常の最大値ノルムになる．

通常の $D \subset X = \boldsymbol{R}^n$ から $Y = \boldsymbol{R}^m$ への写像 f をもとに構成された $I(D)$ から $I(Y)$ への写像を一般に区間写像という．

定義 2.4 (区間包囲，区間拡張) $X = \boldsymbol{R}^n$, $Y = \boldsymbol{R}^m$, $D \subset X$ とする．区間写像 $F : I(D) \to I(Y)$ が $f : D \to Y$ の区間包囲であるとは，

$$F(I) \supset f(I) \quad \text{for all } I \in I(D) \tag{13}$$

が成立することをいう．また，F が f の区間拡張であるとは，F が f の区間包囲で，

$$F([x, x]) = f(x) \quad \text{for all } x \in D \tag{14}$$

が成立することをいう． □

加減乗除などの二項演算，sin，logなどの単項演算の区間包囲は計算機上で容易に実現できる．従って，これらの基本演算の組み合わせで記述された関数の区間包囲も容易に実現できる．しかし，有限桁の浮動小数点演算を用いる限り，厳密な意味での区間拡張の実現は不可能である．

3 区間写像による方程式の表現

非線形方程式の近似解を精度保証する際，方程式自体をどう記述するかということは極めて重要な問題である．従来，精度保証の重要性が主張されるわりには，この方程式の記述の問題は余り真剣に議論されていないように思える．方程式を $f(x)=0$ と書くとき，非線形写像 f は，一般的には計算機上で正確に表現できない．精度保証を行うためには，

1. x に対して $f(x)$ を誤差評価付きで計算できること．

2. x に対して $f(x)$ を任意に高精度に計算できること．

が必要である．

ここでは，f の計算機上の表現 F として，f の区間包囲を用いる．この場合出力の区間幅がそのまま誤差評価を表す．非線形写像全体を考えると，区間包囲の構成は一般的には難しいと考えられる．しかし，通常我々が解析の対象とする非線形写像は，性質の良く分かっている基本関数の組み合わせであるものがほとんどなので，基本関数の区間包囲を作っておけばほとんどの非線形写像に対して自然に区間包囲が実現できることになる．更に，任意高精度計算を可能とするため，以下を仮定する．

仮定 3.1 f の区間包囲の族 $\{F_\varepsilon\}$ ($\varepsilon>0$, $F_\varepsilon : I(\boldsymbol{R}^n) \to I(\boldsymbol{R}^m)$) が存在して，以下を満たす:

> 任意の $c \in \boldsymbol{R}^n$ に対して，$|I_k - c| \to 0$, $\varepsilon_k \to 0$ $(k \to \infty)$ なる任意の列 $\{I_k, \varepsilon_k\}$, $I_k \in I(\boldsymbol{R}^n)$, $\varepsilon_k > 0$ を考えると，
>
> $$\mathrm{rad}(F_{\varepsilon_k}(I_k)) \to 0 \quad (k \to \infty) \tag{15}$$
>
> が成立する．

□

ε は計算の精度を表し，これが 0 に近づくほど無限精度の計算に近くなり，F_ε は区間拡張に近くなる．もちろん，$\varepsilon=0$(厳密な区間拡張) は一般には不可能であり，これは要求しない．

有理数演算の導入などにより計算機内で任意精度実数の扱いを実現すれば，この仮定を満たすような区間包囲の族 $\{F_\varepsilon\}$ は計算機上に構成できる．ε の大きさに応じて計算精度 (演算に用いる数の精度，級数の反復回数) を制御し，$f(I)$ を含むような出力 $F(I)$ を計算すればよい．ε が小さいほど，高精度の数値を用い，級数の反復回数も多くする．

また，この仮定を満たすように基本関数の区間包囲を構成できれば，それの組み合わせで構成される写像も仮定3.1を満たすように作れる．この場合，f を $f^1 \circ f^2 \circ \cdots \circ f^m$ のように合成関数で表し，$f^1, \cdots, f^m$ の仮定 3.1 を満たすような区間包囲族 $\{F_\varepsilon^1\}, \cdots, \{F_\varepsilon^m\}$ を用いて $\{F_\varepsilon^1 \circ F_\varepsilon^2 \circ \cdots \circ F_\varepsilon^m\}$ を f の区間包囲族とする．このとき，各 $\{F_\varepsilon^k\}$ の ε と精度との関係を "揃えて" おかないと，ε に対する精度の上昇が遅いような部分が Bottle-Neck となってしまい，全体としての精度の上がり方が遅くなり，結果として必要な精度を得るために無駄な計算をすることになる．

以上を考えると，実際には次のようにするのが簡単であろう．ある ε が与えられたとき，その ε に応じた精度の数値体系 (例えば $\varepsilon = 10^{-15}$ ならば 10 進 15 桁の浮動小数点) を用い，基本演算はその数値体系での最良の区間包囲を行うようにする．これで，大抵の場合は仮定 3.1を満たすはずである．

4 区間演算による近似解の包み込み

非線形方程式 $f(x) = 0$ の近似解に対して精度保証を行うことを考える．写像 f が，数学における写像の定義の通り点を与えると点を返すような単なる対応として計算機上に表現されていたならば，精度保証はほとんど不可能である．f がそのような black box ならば，ある近似解における残差がどんなに小さかったとしてもその近くに真の解が存在する保証にはならない．ある点における情報はそのすぐ近くの点についての情報を何一つ与えない，まさに "一寸先は闇" といった状態である．よって，ただの一点でなくある領域についての情報が必要となるが，そこで写像 f による入力区間の像を容易に包み込める区間包囲を用いる．ここでの区間演算の役割は，もはや数値誤差の包み込みという消極的なものでなく，区間の像を容易に包み込めるという性質を積極的に利用したものである．この場合，扱われる区間は微小区間でなく，ある程度の幅を持った区間となる．

Krawczyk の区間写像を用いる方法は，このような区間包囲による精度保証の方法の代表的なものである．これは，導関数 f' の区間包囲を利用し，簡易ニュートン反復に対して縮小写像原理の成立を確かめる方法である．導関数 f' の区間包囲を構成するためには f' のプログラムによる表現が必要になるが，f がプログラムで書かれている場合には，自動微分の技法を用いることにより容易に f' の区間包囲を構成できる．

Krawczyk の区間写像を用いた解の包み込みを以下に示す．

定理 4.1 (Krawczyk の区間写像による解の包み込み) [1, 2] $U \subset \boldsymbol{R}^n$，$f : U \to \boldsymbol{R}^n$ を C^1級とする．ある区間 $T \in I(U)$ について，区間行列 M を

$$M = E - L^{-1}F'(T) \tag{16}$$

で，区間写像 K を

$$K(I) = c - L^{-1}f(c) + M(I - c) \tag{17}$$

$$c = \mathrm{mid}(I) \tag{18}$$

で定義する．ただし，E は単位行列，L は (T における微分の近似である) 正則な非区間行列，区間写像 F' を導関数 f' の区間包囲とする．

ここで，

$$K(T) \subset T \tag{19}$$

$$\|M\| < 1 \tag{20}$$

が満たされるならば，以下が成立する．

(1) T に方程式 $f(x)=0$ の解 x^* が唯一存在する．

(2) 区間反復を

$$I_0 = T \tag{21}$$

$$I_{k+1} = I_k \cap K(I_k) \tag{22}$$

で定義すると，

$$x^* \in I_k \quad \text{for all } k\,, \tag{23}$$

$$\|\operatorname{rad}(I_{k+1})\| \leq \|M\| \cdot \|\operatorname{rad}(I_k)\| \tag{24}$$

が成立する．

(3) $\forall x_0 \in T$ から出発する簡易ニュートン反復

$$x_{k+1} = x_k - L^{-1} f(x_k) \tag{25}$$

に対して，

$$x_k \in T \quad \text{for all } k\,, \tag{26}$$

$$x_k \to x^* \quad \text{as } k \to \infty\,, \tag{27}$$

$$\|x_k - x^*\| \leq \frac{\|L^{-1} f(x_k)\|}{1 - \|M\|} \tag{28}$$

$$\|x_k - x^*\| \leq \frac{\|M\|^k \|L^{-1} f(x_0)\|}{1 - \|M\|} \tag{29}$$

が成立する．

□

この定理は，簡易ニュートン反復を定義する写像を

$$g(x) = x - L^{-1} f(x) \tag{30}$$

とすると区間写像 K，区間行列 M が，

$$K(I) \supset \{g(x) \mid x \in I\} \quad \text{if } I \subset T \tag{31}$$

$$M \supset \{g'(x) \mid x \in T\} \tag{32}$$

を満たすように定義されていることから，縮小写像原理により証明される．

この方法は，近似解が与えられたとき，その点を元にして真解の包み込みを行うには適していない．なぜなら，近似解に対して，真解を含むような適当な大きさの区間Tを決定する必要があり，その方法が不明であるためである．区間演算を近似解の精度保証に利用するための方法としては，Rumpの提案する手法 [3] がある．近似解を初期点とするある種の区間反復を行うものであるが，理論的にやや疑問があり，また十分良い近似解に対して必ず停止するという保証もない．

ここでは，Krawczykの方法に対して，近似解が与えられたときそれに対する具体的な初期区間 T を与える方法を与え，その方法で十分良い近似解に対して真解の包み込みが行えることを示す．

初期区間 T は次のように与える．まず，近似解 x で $\|f'(x)^{-1}f(x)\|$ (ニュートン法の一回目の反復における修正量) を計算する．次に，x を中心とし，計算した量の $\rho > 1$ 倍を半径とする球 (区間) を作り，この領域に対してKrawczykの方法を適用する．まとめると次のようになる．

アルゴリズム 4.1 (近似解を元にした真解の包み込み) $U \subset \boldsymbol{R}^n$，$f : U \to \boldsymbol{R}^n$ を C^1 級とし，$\boldsymbol{R}^n$ のノルムは $u > 0$ を scaling vector とする scaled maximum norm とする．$\{F_\varepsilon\}, \{F'_\varepsilon\}$ はそれぞれ f, f' の区間包囲の族で，仮定3.1を満たすとする．近似解 $c \in U$，精度 ε が与えられているとし，$\rho > 1$ (定数) とする．

(1) 近似解 c に対して，$F_\varepsilon(c), F'_\varepsilon(c)$ を計算する．

(2) $L \in F'_\varepsilon(c)$ を任意に決定．

(3) L^{-1} が存在しなければ失敗．存在すれば，$L^{-1}F_\varepsilon(c)$ を計算し，

$$\delta = \rho\|L^{-1}F_\varepsilon(c)\|_u, \tag{33}$$

$$T = [-\delta u, \delta u] + c \tag{34}$$

とする．

(4) $M = E - L^{-1}F'_\varepsilon(T)$ とし，条件

$$c - L^{-1}F_\varepsilon(c) + M(T - c) \subset T \tag{35}$$

$$\|M\|_u < 1 \tag{36}$$

が成立すれば，区間 T に真解が唯一存在する．

□

次に，アルゴリズム4.1によって十分良い近似解に対して真解の包み込みが行えることを，次の定理4.2に示す．

定理 4.2 (アルゴリズム 4.1による真解の包み込み) アルゴリズム4.1の仮定の下で，$x^* \in U$ が，$f(x^*) = 0, \exists f'(x^*)^{-1}$ を満たすとする．このとき，近似解の列 $\{c_k\}$ と精度の列 $\{\varepsilon_k\}$ が $c_k \to x^*, \varepsilon_k \to 0\ (k \to \infty)$ を満たすならば，アルゴリズム4.1は十分大きい k に対して解 x^* を包みこむ．□

(証明) 以下，$\|\cdot\| = \|\cdot\|_u$ とする．

$c = c_k, \varepsilon = \varepsilon_k$ としてアルゴリズム 4.1を実行したときの L, δ, T, M をそれぞれ L_k, δ_k, T_k, M_k と書く．

仮定より x^* は孤立解であるから，十分大きい k に対して式 (35), (36) が成立した場合，T_k が包みこむ解は x^* である．よって十分大きい k に対して式 (35), (36) が成立することを言えばよい．

式 (35) について，

$$1 + \|M_k\|\rho \le \rho \tag{37}$$

$$\Downarrow$$

$$\|L_k^{-1} F_{\varepsilon_k}(c_k)\| + \|M_k\|\delta_k \le \delta_k \tag{38}$$

$$\Downarrow$$

$$|L_k^{-1} F_{\varepsilon_k}(c_k)| + |M_k|\delta_k u \le \delta_k u \tag{39}$$

$$\Downarrow$$

$$\text{式 (35)} \tag{40}$$

となるので，十分大きい k に対して式 (37) をいえばよい．ここで，式 (36) と $\rho > 1$ を考えると，

$$\|M_k\| \to 0 \quad (k \to \infty) \tag{41}$$

を示せば題意が示されることが分かる．

まず，$\{F'_\varepsilon\}$ が仮定 3.1を満たすので，$\varepsilon_k \to 0$, $c_k \to x^*$ より

$$\mathrm{rad}(F'_{\varepsilon_k}(c_k)) \to 0 \quad (k \to \infty) \tag{42}$$

が分かる．

次に，f が C^1 級なので，式 (42) を用いると，

$$\|L_k - f'(x^*)\| \le \|L_k - f'(c_k)\| + \|f'(c_k) - f'(x^*)\| \tag{43}$$

$$\le 2\|\mathrm{rad}(F'_{\varepsilon_k}(c_k))\| + \|f'(c_k) - f'(x^*)\| \tag{44}$$

$$\to 0 \tag{45}$$

となる．よって，$\beta \stackrel{\mathrm{def}}{=} \|f'(x^*)^{-1}\|$ とすると，$\beta\|L_k - f'(x^*)\| < 1$ となるような十分大きい k に対して，ある定数 $\gamma > \beta$ が存在し，

$$\|L_k^{-1}\| \le \frac{\beta}{1 - \|L_k - f'(x^*)\|\beta} \le \gamma \tag{46}$$

が成立する．

また，$\varepsilon_k \to 0$, $c_k \to x^*$ と f の連続性により

$$\|F_{\varepsilon_k}(c_k)\| \le \|F_{\varepsilon_k}(c_k) - f(c_k)\| + \|f(c_k)\| \tag{47}$$

$$\le 2\|\mathrm{rad}(F_{\varepsilon_k}(c_k))\| + \|f(c_k)\| \tag{48}$$

$$\to 0 \tag{49}$$

が分かる．よって，式 (46) より

$$\delta_k = \rho\|L_k^{-1}F_{\varepsilon_k}(c_k)\| \tag{50}$$
$$\leq \rho\gamma\|F_{\varepsilon_k}(c_k)\| \tag{51}$$
$$\to 0 \tag{52}$$

となる．よって，

$$|T_k - x^*| \leq |T_k - c_k| + |c_k - x^*| \tag{53}$$
$$= \delta_k u + |c_k - x^*| \tag{54}$$
$$\to 0 \tag{55}$$

が分かる．従って，$\varepsilon_k \to 0$ を考えると，

$$\mathrm{rad}(F'_{\varepsilon_k}(T_k)) \to 0 \quad (k \to \infty) \tag{56}$$

となる．

以上により，式 (41) を示すことが出来る．式 (42), (56) を用いて

$$\|M_k\| = \|E - L_k^{-1}F'_{\varepsilon_k}(T_k)\| \tag{57}$$
$$\leq \gamma\|L_k - F'_{\varepsilon_k}(T_k)\| \tag{58}$$
$$\leq \gamma(\|L_k - f'(c_k)\| + \|f'(c_k) - F'_{\varepsilon_k}(T_k)\|) \tag{59}$$
$$\leq 2\gamma(\|\mathrm{rad}(F'_{\varepsilon_k}(c_k))\| + \|\mathrm{rad}(F'_{\varepsilon_k}(T_k))\|) \tag{60}$$
$$\to 0 \tag{61}$$

となり，題意は示された． □

ρ が 1 よりも大きい実数であればこの定理は成立するが，幾つかの理由により，$\rho = 2$ が最適であると考えている．

5 有理数演算による反復改良

前節により，近似解を用いた真解の包み込みが可能になった．本節では反復改良の方法について述べる．

本稿では方程式を記述する写像 f は正確に表現できないという立場を取っており，区間包囲 F の返す区間は幅を持つので，当然定理 4.1の (2), (3) を満たすような反復改良は行えない．また，仮に区間包囲 f が幅 0 の区間を返したとしても，理論通り全く丸めのない区間反復を有理数を用いて行えば，分母分子が“桁の爆発”を起こし，ほとんど反復を進めることはできない．このような問題点を解決した区間反復法を，

文献 [4] で提案している．これは，有理数演算を利用し，丸めの制御によって区間幅の線形縮小を保証するものである．ここでは，アルゴリズムの概略を示す．

アルゴリズム 4.1で区間 T に真解 x^* が包み込まれたとする．

まず，区間ベクトル $I \subset T$，ベクトル $p \in \boldsymbol{Q}^n$ に対して，$K(I;p)$ を次で定義する．

$$K(I;p) = c - L^{-1}F_{\varepsilon(c,p)}(c) + M(I-c) \quad (c = \mathrm{mid}(I)) \tag{62}$$

ただし，$\varepsilon(c,p)$ は，$\mathrm{rad}(L^{-1}F_{\varepsilon(c,p)}(c)) \leq p$ を満たすように c,p に応じて選ばれるものとする．

反復改良は，丸めの過程が入るのでやや複雑である．

まず最初に，縮小定数の余裕分 $\kappa \in \boldsymbol{Q}^{n\times n} > 0$ (行列) を決定する．これは，

$$\|\kappa + |M|\|_u < 1 \tag{63}$$

を満たす行列とする．これは，丸めによって縮小率を $\|M\|_u$ から $\|\kappa + |M|\|_u$ に悪化させることを意味する．この κ を用いて丸めを制御する．以下に，区間 I_k から区間 I_{k+1} を生成するアルゴリズムをまとめる．

(1) 現在の区間幅 $\mathrm{rad}(I_k)$ (ベクトル) と κ から，丸めの許容誤差 (関数値の計算誤差，区間の丸め) を決定する．具体的には，

$$r_k + p_k \leq \kappa\,\mathrm{rad}(I_k) \tag{64}$$

を満たすベクトル $r_k > 0$，$p_k > 0$ を決定する．

(2)

$$I^*_{k+1} = K(I_k;p_k) \tag{65}$$

によって I^*_{k+1} を誤差無しで計算する．

(3) I^*_{k+1} を誤差 r_k 以内で外側に丸め，I^{**}_{k+1} とする．

(4)

$$I_{k+1} = I_k \cap I^{**}_{k+1} \tag{66}$$

によって I_{k+1} を決定する．

$\mathrm{rad}(I_k)$ は次第に小さくなっていくので，丸めの許容誤差も次第に小さくなっていく．すなわち，反復の初期段階は粗い精度で計算し，反復が進行するにつれ高精度になっていくようなアルゴリズムになっている．

この反復によって生成される区間列 $\{I_k\}$ は，

$$x^* \in I_k \quad \text{for all } k\,, \tag{67}$$

$$\|\,\mathrm{rad}(I_{k+1})\|_u \leq \|\kappa + |M|\|_u \cdot \|\,\mathrm{rad}(I_k)\|_u \tag{68}$$

を満たす．

6 精度保証システムの実現

ここでは，本稿で提案された精度保証アルゴリズムが実際に実現できることを，システムの試作によって示す．

本稿のアルゴリズムを実現するためには，少なくとも任意に高精度な演算体系が必要である．また，精度保証付き数値計算が実用になるためには，その利用が比較的容易である必要がある．その手法を利用するために事前に多くの検討を行う必要があれば，その段階での間違いの混入が防げなくなり，計算結果の信頼性の低下を招くことになる．すなわち，方程式の記述が容易で，自動的に解析を行うようなシステムを構築しなければならない．このためには，区間演算や自動微分が容易に実現できることが必要であり，演算子の多重定義の可能な object 指向的な言語が望ましい．

現在試作中のシステムでは，David I. Bell 氏による

CALC - C-style arbitrary precision calculator : version 1.25.0

を改良したものを用いている．このソフトウェアは全て C 言語で記述されており，任意精度計算の可能な対話的な電卓プログラムである．数値演算は全て有理数で行われ，任意精度計算が可能な様々な数学関数を備えている．文法は C 言語に極めて類似しており，新しい関数を定義することによってプログラミングが行える．インタプリタであるため変数の型に関して柔軟であり，ベクトルや行列は要素として任意の型を持つことが出来る．更に，ユーザーが新しい型を定義することも出来，それに対する演算子の働きを多重定義出来るため，極めて容易に言語の拡張が行える．

この CALC に新しい型を追加し，また幾つかの関数を定義することによって，本稿で述べたアルゴリズムをほぼ完全に実現することが出来た．以下，その概要を説明する．

区間演算の実現 区間演算を実現するために，上限と下限からなる区間型を定義し，演算子の多重定義によってその加減乗除を定義した．これは，非区間数との混在も可能とした．また，$\mathrm{mid}, \mathrm{rad}, |\cdot|, \|\cdot\|_u$ などの関数を定義した．これらの関数は，区間だけでなくベクトルや行列に対しても適用可能とした．

自動微分の実現 自動微分を，Bottom Up 型のアルゴリズムにより実現した．関数値と入力変数に関する偏微分値を持つような自動微分型を定義し，その加減乗除を演算子多重定義によって定義した．これにより，関数への入力ベクトルをこの型で初期化しておけば，その後の演算で得られる全ての数値について自動的に入力変数に関する偏微分値が得られる．また，先に定義した区間型を同時に用いると，自然に関数のヤコビアンの区間包囲が実現できる．

数学関数の実現 上で定義した 2 つの型を考慮して，数学関数を定義した．すなわち，区間の入力に対して区間包囲として働くように，自動微分型の入力に対して自動微分を行うようにした．また，この場合の区間包囲の精度は，ある大域変数 (仮定 3.1の ε に相当) に従うようにした．これにより，仮定 3.1 を満たすような関数は極めて容易に記述可能となった．

精度保証システムの実現 以上で定義した機能を用いて，与えられた近似解に対して精度保証を行うことが出来るようになった．具体的には，近似解，関数名，精度要求を与えて，それに対して精度保証を試みる関数を作成した．近似解が十分良い近似ならば，真の解を包含する区間を求め，それを精度要求まで反復改良して精度要求以下の幅を持つ区間を返す．近似解が悪ければその旨表示する．

自動微分により導関数の記述を必要としないので，本システムに対する入力は，方程式のプログラムによる記述と近似解だけである．プログラムに誤りが無いことを証明することは出来ないが，仮に誤りが無いとすると，このシステムによって精度保証された解は数学的に厳密なものである．

以下に，実行例を挙げる．

方程式

$$2x_0^2 - x_1 = 0 \tag{69}$$

$$1/x_0 - x_1 = 0 \tag{70}$$

は，

```
define func(x)
{
    local y;
    mat y[2];

    y[0] = 2 * x[0]^2 - x[1];
    y[1] = 1 / x[0] - x[1];
    return y;
}
```

と書けばよい．これは，2次元のベクトル型xを受け取って，2次元のベクトル型yを返す‘func’という名前の関数の定義である．但し，x及びyの成分の型に関する定義はなく，この関数の動作は実行時にx[0]及びx[1]に入っていた型によって決まる．従って，区間演算や自動微分を行う型の入力を与えてやるだけで，区間演算や自動微分を行える．

このプログラムと，近似解 $(0.8, 1.25)$ を与えたときの実行の様子を以下に示す．

```
C-style arbitrary precision calculator.
calc version: 1.25.0 - k1.0 (go32 on msdos)
Copyright (c) 1992 David I. Bell
Modified 1993 by Masahide Kashiwagi
[Type "exit" to exit, or "help" for help.]

01> read kashi
```

```
02> read vector
03> read func2
04> x = v(0.8, 1.25)
05> r = kashi(x, "func", 1e-15)
calculating F(c) and F'(c)
select L; calculating inverse of L
calculating interval T

mat [2] (2 elements, 2 nonzero):
  [0] = [.78,.82]
  [1] = [1.23,1.27]

calculating F'(T)
calculating M = E - L^{-1}F'(T)
testing if solution exists in T
solution is found in

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79302401894043323480,~.79437755586271637149]
  [1] = [~1.25822699184298215188,~1.26145804752709658827]

making kappa
norm of M
~.08077639210286090964
contraction ratio
~.15202907185633891409
I1

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79245283018867924528,~.79487179487179487179]
  [1] = [~1.25806451612903225806,~1.26190476190476190476]

I2

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79365079365079365079,~.79381443298969072164]
  [1] = [~1.25980392156862745098,~1.26004728132387706855]

..... 中略 .....

I11

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79370052598409813311,~.79370052598410248792]
  [1] = [~1.25992104989487017174,~1.25992104989487571136]
```

```
I12

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79370052598409955359,~.79370052598409987478]
  [1] = [~1.25992104989487293214,~1.25992104989487336619]

06> printf("%f\n", r)

mat [2] (2 elements, 2 nonzero):
  [0] = [~.79370052598409955359,~.79370052598409987478]
  [1] = [~1.25992104989487293214,~1.25992104989487336619]

07> printf("%r\n", r)

mat [2] (2 elements, 2 nonzero):
  [0] = [99360729/125186674,256618531/323319089]
  [1] = [338560044/268715285,140427629/111457483]

08> exit
```

'数字>' で始まる行は，user の入力を表す．'05>' の行は，x を近似解として，'func' という名前の関数によって与えられる方程式の解の精度保証を試み，もし精度保証が出来たならば半径が 1e-15 以下になるまで反復改良させるという意味である．'06>', '07>' の行で，小数と有理数で結果を表示させている．途中経過は全て小数で表示させているが，これは分かりやすさのためであり内部では全て有理数で計算されている．小数表示の先頭の '~' は，小数の数字の続きがまだあることを表している．

次に，ロジスティック写像

$$x_{i+1} = rx_i(1 - x_i) \tag{71}$$

$$r = 3.816 \tag{72}$$

$$x_0 = 0.3 \tag{73}$$

の軌道を解として持つ方程式

```
define func(x)
{
    local y, i;
    mat y[10];

    y[0] = x[0] - .3;
    for (i=1; i<10; i++) {
        y[i] = x[i] - 3.816 * x[i-1] * (1 - x[i-1]);
    }

    return y;
}
```

に対して，近似解

```
[0] = .3
[1] = .80136
[2] = .6074390858
[3] = .9099513122
[4] = .3126827409
[5] = .8201051248
[6] = .5629848178
[7] = .938861595
[8] = .2190403097
[9] = .6527712658
```

を与えたときの精度保証された軌道を示す (半径 1e-20 以下).

```
[0] = [.3,.3]
[1] = [.80136,.80136]
[2] = [~.60743908592639999999,~.60743908592640000000]
[3] = [~.90995131218318341652,~.90995131218318341652]
[4] = [~.31268274097551572739,~.31268274097551572740]
[5] = [~.82010512490345107651,~.82010512490345107651]
[6] = [~.56298481758424348040,~.56298481758424348040]
[7] = [~.93886159506880445157,~.93886159506880445158]
[8] = [~.21904030942590454201,~.21904030942590454201]
[9] = [~.65277126507189257283,~.65277126507189257284]

[0] = [3/10,3/10]
[1] = [10017/12500,10017/12500]
[2] = [17487622756/28789096983,18104511185/29804653017]
[3] = [15331818335/16849053493,30647235319/33680082559]
```

```
[4] = [4149996280/13272226881,6546975001/20938076021]
[5] = [7152763991/8721764776,48768990085/59466754449]
[6] = [14637954131/26000619686,5590544273/9930186567]
[7] = [20711532584/22060261803,7829040939/8338865899]
[8] = [35950151828/164125735223,10253925518/46812961253]
[9] = [40004570553/61284208870,40643508873/62263017764]
```

最後に，f が計算機上で厳密に表現できない場合を示す．方程式

$$\exp(x_0) - x_1 = 0 \tag{74}$$

$$1/x_0 - x_1 = 0 \tag{75}$$

に対して，プログラムは，

```
define func(x)
{
    local y;
    mat y[2];

    y[0] = Exp(x[0]) - x[1];
    y[1] = 1 / x[0] - x[1];
    return y;
}
```

となる．これに対して，近似解 $(0.57, 1.75)$ を与えたときの実行結果を示す．

```
C-style arbitrary precision calculator.
calc version: 1.25.0 - k1.0 (go32 on msdos)
Copyright (c) 1992 David I. Bell
Modified 1993 by Masahide Kashiwagi
[Type "exit" to exit, or "help" for help.]

01> read kashi
02> read vector
03> read func4
04> x = v(0.57, 1.75)
05> r = kashi(x, "func", 1e-20, 1e-5)
calculating F(c) and F'(c)
select L; calculating inverse of L
calculating interval T

mat [2] (2 elements, 2 nonzero):
  [0] = [~.54297297297297297,~.59702702702702702]
```

```
  [1] = [~1.72297297297297297,~1.77702702702702702]

calculating F'(T)
calculating M = E - L^{-1}F'(T)
testing if solution exists in T
solution is found in

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56511192296287881775,~.56915930631600411033]
  [1] = [~1.75928947672290683843,~1.76711486529345841829]

making kappa
norm of M
~.14453469455923763238
contraction ratio
~.25924365719150842510
I1

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56410256410256410256,~.57142857142857142857]
  [1] = [~1.75862068965517241379,~1.76785714285714285714]

I2

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56666666666666666666,~.56756756756756756756]
  [1] = [1.7625,~1.76388888888888888888]

I3

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56707317073170731707,~.56721311475409836065]
  [1] = [~1.76313594662218515429,~1.76331360946745562130]

epsilon=2.5e-6
I4

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56713426853707414829,~.56715210355987055016]
  [1] = [~1.76320939334637964774,~1.76323529411764705882]

epsilon=6.25e-7
I5

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56714178544636159039,~.56714471968709256844]
```

```
  [1] = [~1.76322115384615384615,~1.76322489391796322489]

..... 中略 .....

epsilon=~2.2204460492503130808 4e-21
I22

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56714329040978387299,~.56714329040978387300]
  [1] = [~1.76322283435189671021,~1.76322283435189671023]

epsilon=~5.551115123125782702 11e-22
I23

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56714329040978387299,~.56714329040978387300]
  [1] = [~1.76322283435189671022,~1.76322283435189671022]

06> printf("%f\n", r)

mat [2] (2 elements, 2 nonzero):
  [0] = [~.56714329040978387299,~.56714329040978387300]
  [1] = [~1.76322283435189671022,~1.76322283435189671022]

07> printf("%r\n", r)

mat [2] (2 elements, 2 nonzero):
  [0] = [27114125437/47808245104,10030022741/17685165126]
  [1] = [68117035367/38632119571,211541770679/119974495882]

08> exit
```

途中に現れる 'epsilon=' の行は，反復改良中に区間包囲 F_ε の計算精度が足りなくなり (出力区間幅が広すぎる)，計算精度 ε が小さくされたことを表している．現在のシステムでは，ε は四則演算には影響せず (四則演算は丸めのない有理数演算)，exp の区間包囲の計算精度が影響を受けるようになっている．

7 むすび

本稿では，区間解析と有理数演算による非線形方程式の近似解の精度保証について報告した．

本報告の手法を任意精度計算が出来ないような数値体系の下で使用した場合を考える．本報告で保証した事柄は，

(1) 解が包み込まれたときそこに数学的に厳密な意味で唯一解が存在すること の保証

(2) 解が包み込まれたとき任意にその解の精度を高められることの保証

(3) 十分良い近似解に対して真解の包み込みが可能であることの保証

の3つである．任意精度計算が出来ない場合は，(2), (3) の保証は失われるが，(1) は保たれる．すなわち，その様な場合でも，必ず出来るとは言えないがその数値体系下で可能な範囲で解の存在保証は出来，またその数値体系下で可能な範囲で反復改良も出来る．

現在最も大きな問題点は，区間包囲の計算精度 ε の大きさの決定法である．大きすぎれば良い近似解に対して精度保証に失敗する危険があり，小さすぎれば精度の要求が高くなりすぎ，計算量が増大する．

また，現在試作中のシステムは基本的にインタプリタであるため，速度に限界がある．より高速な実用システムを実現する場合に発生する問題点についても考えていきたい．

謝辞

日頃御指導頂く早稲田大学堀内和夫教授，熱心に御議論頂いた早稲田大学大石研究室黒武者健一氏に深謝する．

参考文献

[1] R. E. Moore : "A Test for Existence of Solutions to Nonlinear Systems", SIAM J. Numer. Anal., Vol. 14, pp.611–615 (1977).

[2] A. Neumaier : "Interval methods for systems of equations", Cambridge University Press (1990).

[3] S. M. Rump : "Solving Nonlinear Systems with Least Significant Bit Accuracy", Computing, 29, pp.183–200 (1982).

[4] 柏木雅英, 木村孝, 井上晃, 大石進一 : "有理数演算を用いた精度保証付き区間反復法", 信学技報, NLP91-96, pp.23–29 (1992).