論文

【論 25-8】

高精度内積計算アルゴリズムを用いた連立一次方程式の精度保証付き数値計算法†

大石進一*,**・荻田武史**,*・太田貴久*

ABSTRACT IEEE standard 754 is widely used as a standard of floating-point arithmetic. Most of CPUs in today's computers support IEEE standard 754. Using double precision arithmetic following IEEE standard 754, the authors have proposed fast methods of verifying the accuracy of a numerical solution of a linear system. In this paper, an accurate and fast verification method for a linear system is developed using residual iteration method. The residual iteration requires the availability of high precision computation. Up to now, extended precision, i.e. multiple precision and quadruple precision are used for the accurate computation of the residual. However, such higher precision arithmetic systems are not necessarily available on all computers. Therefore, the residual iteration using such systems does not have the portability. In this paper, an accurate, fast and portable method for a linear system using the fact that an algorithm of accurate dot product can portably be implemented and applied to the residual iteration. Finally, numerical results are presented showing the effectiveness of the proposed verification method.

1. はじめに

本論文では，連立一次方程式

$$Ax = b \tag{1}$$

の数値解の精度保証付き数値計算法について考える．ただし，$n \times n$ 行列 A は実行列で，b は実 n 次元ベクトルとする．ここに，行列 A が実とは，その要素が全て実数であることをいう．以下，本論文では，記号として，$\mathbb{R}$ で実数の集合，$\mathbb{R}^n$ で実 n 次元ベクトル空間を表す．また，ベクトル $x \in \mathbb{R}^n$ の最大値ノルムを

$$\|x\|_\infty = \max_{1 \le i \le n} |x_i| \tag{2}$$

で，行列 $A \in \mathbb{R}^{n \times n}$ の(最大値ノルムによる)作用素ノルムを

$$\|A\|_\infty = \sup_{\|x\|_\infty = 1} \|Ax\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |A_{ij}| \tag{3}$$

で表す．x_i は x の第 i 成分で，A_{ij} は A の第(i, j) 成分とする．

式(1)の形の連立一次方程式は科学技術計算の各分野に現れることが知られている．特に，科学技術計算の分野での応用に際しては，問題の規模である次元 n の大きい問題が現れることが多い．そのように次元数が大きい場合，式(1)の数値解は計算機ハードウェアに実装された浮動小数点演算により，計算されることが普通である[1]．本論文でも，行列 A とベクトル b の各要素は浮動小数点数とし，解の計算には浮動小数点演算を用いるという立場を取る．浮動小数点演算は計算機が発明されて以来，長い間標準化されなかったが，1975年から10年間のデファクトスタンダードの期間を経て，1985年にIEEE754の2進浮動小数点数規格が制定され，今日ではその規格に従ったCPUがほとんどとなっている．IEEE754規格の中では，32ビットの単精度浮動小数点数と64ビットの倍精度浮動小数点数およびそれらの演算が標準化されている．以下，本論文

Numerical Verification Method for Systems of Linear Equations Using Accurate Computation of Dot Product. By *Shin'ichi Oishi* (Faculty of Science and Engineering, Waseda University / CREST, Japan Science and Technology Agency), *Takeshi Ogita* (CREST, Japan Science and Technology Agency / Faculty of Science and Engineering, Waseda University) and *Takahisa Ohta* (Faculty of Science and Engineering, Waseda University).

* 早稲田大学 理工学術院

** 科学技術振興機構 戦略的創造研究推進事業

† 2006年1月25日受付

1 問題の次元数が小さく，かつ，A と b の要素が有理数である場合には，有理数演算により式(1)の正確な解を求めることができる．この場合には精度保証は必要なくなる．しかし，実際的な計算時間とメモリ量の範囲内で扱いうる問題の次元はかなり低く制限される．実際，通常のPCの場合には取り扱える次元の最大が数百次元程度と思われる．有理数演算では，この程度でも，式(1)を解くのに一日かかる．また，その計算時間は浮動小数点数計算に比べて n の指数関数的オーダで増大する．

ではIEEE754規格に従う倍精度浮動小数点数での数値計算のみを考えることにするが，単精度浮動小数点数を用いる場合も同様の議論が成立する．

このIEEE754倍精度浮動小数点数規格は数学的に美しい性質を備えたもので，精度保証付き数値計算に必要な方向丸めの概念も実装されている．これを利用して，*Oishi* と *Rump*[6]は，IEEE754規格での内積計算の精度保証が2回の丸めモードの変更でできることを指摘し，それを基礎に丸めモード制御方式精度保証付き数値計算法を開発した．これにより，IEEE754規格の倍精度浮動小数点演算のもとで，式(1)の数値解を高速に精度保証できることも示されている．

従来，数値解の精度保証のための計算時間は，その数値解を求めるための計算時間に対して数百倍から数千倍程度かかるのが常識的であった．ここでの高速というのは，精度保証にかかる計算時間 t_{ver} が，数値解を求めるための計算時間 t_{sol} に比べて数倍程度であることをいうことにする．実際，文献6)では，A が密行列でガウスの消去法などの直接解法が倍精度演算の範囲で適用できるような問題のクラスに対しては，式(1)において t_{ver} が t_{sol} の5倍程度(計算量は8倍)となる方法が示された．特に，問題の次元数 n が5000程度以下で係数行列が密行列かつ条件数がそれほど大きくない場合，t_{ver} と t_{sol} がほぼ同じくらいになる方法も示されており[6]，さらに文献9)では，密行列の中でより大規模な問題や条件数のある程度大きいクラスの問題に対しても，t_{ver} が t_{sol} の3倍程度(計算量は4倍)で済む方法が示されている．

本論文では，IEEE754規格を満たす倍精度浮動小数点演算のみを用いて高速かつ高精度に内積計算を行うアルゴリズムを示し，A が密行列でガウスの消去法などの直接解法が倍精度演算の範囲で適用できるような問題のクラスに対しては，その高精度内積計算法を利用して，式(1)の数値解の残差反復計算が高速に実行できることを述べるとともに，その数値解の高速で高精度な精度保証法を提案する．本論文の議論のポイントは，このような高精度な精度保証法が，丸めモード制御方式精度保証付き数値計算法の高速性を損なうことなく実装できることを示すことである．

2. 浮動小数点演算と高精度内積計算

本論文で必要となるIEEE754規格を満たす倍精度浮動小数点数に成り立つ重要な性質を概観する．

2.1 IEEE754の倍精度浮動小数点数規格

まず，IEEE754の倍精度浮動小数点数規格について概観しよう．IEEE754規格での倍精度浮動小数点数は64ビットで表される浮動小数点数であり，つぎのような形をしている．

$$f=(-1)^s\times 2^e\times k \tag{4}$$

ここに，s は符号ビット(正ならば0，負ならば1)，e は下駄履き表現された指数部(11ビット)，k は仮数部(53ビット)である．e および k はそれぞれ，2進数整数，2進数小数である．仮数部は，正規化できる範囲では $k=1.k_1k_2\cdots k_{52}$ のように先頭の1ビットを暗黙に仮定しているので，実際には52ビットを保持するだけで53ビットを表現している．f の値が大きすぎて正規化できない場合をオーバフロー，小さすぎて正規化できない場合をアンダフローと呼ぶが，これらの詳細については文献1)を参照されたい．

$\mathbb{F}$ でIEEE754規格の倍精度浮動小数点数の集合を表すことにする．IEEE754規格では倍精度浮動小数点演算は丸めモードを指定することにより定義されている．丸めモードはIEEE754では4種類あるが，ここでは，本論文に関係する3つの丸めモードの定義を紹介する．

(1) $-\infty$ **方向への丸め(下への丸め)** $c\in\mathbb{R}$ を $f\le c$ を満たす最も大きな浮動小数点数 $f\in\mathbb{F}$ へ丸める．下への丸めを $\nabla:\mathbb{R}\to\mathbb{F}$ で表す．

(2) $+\infty$ **方向への丸め(上への丸め)** $c\in\mathbb{R}$ を $f\ge c$ を満たす最も小さな浮動小数点数 $f\in\mathbb{F}$ へ丸める．これを $\triangle:\mathbb{R}\to\mathbb{F}$ と表す．

(3) **最近点への丸め** $c\in\mathbb{R}$ を $|f-c|$ が最小となる浮動小数点数 $f\in\mathbb{F}$ へ丸める．これを $\square:\mathbb{R}\to\mathbb{F}$ と表す．

IEEE754規格での倍精度浮動小数点演算は，四則演算子 $\cdot\in\{+,-,\times,/\}$ と丸め演算子 $\bigcirc\in\{\triangle,\nabla,\square\}$ に対して

$$x\odot y=\bigcirc(x\cdot y),\ (\forall x,y\in\mathbb{F}) \tag{5}$$

が満たされるように定義される．ここで，式(5)の左辺の $\odot$ は浮動小数点演算を表し，$x\cdot y$ は通常の実数演算を表すものとする．

この定義から任意の $x,y\in\mathbb{F}$ と $\cdot\in\{+,-,\times,/\}$ に対して

$$\nabla(x\cdot y)\le x\cdot y\le\nabla(x\cdot y)$$

が成り立つ．したがって，特に，$x_i,y_i\in\mathbb{F}$, $(i=1,2,\cdots,n)$ のとき

$$\mathrm{fl}_\nabla\left(\sum_{i=1}^n x_i\times y_i\right)\le\sum_{i=1}^n x_i\times y_i\le\mathrm{fl}_\triangle\left(\sum_{i=1}^n x_i\times y_i\right)$$

が成り立つことがわかる．ここで，$\mathrm{fl}_\triangle$ は $+\infty$ 方向の丸めモードで，$\mathrm{fl}_\triangledown$ は $-\infty$ 方向の丸めモードでそれぞれ浮動小数点演算を行うことを表す．これは内積の上限と下限が厳密に計算できることを示している．また，式(4)から，倍精度浮動小数点数の絶対値が正しく計算できることや($s=0$ とすればよい)，2つの倍精度浮動小数点数の大小が正しく判定できることがわかる．したがって，$v\in\mathbb{F}^n$ が

$$\underline{v}_i \leqq v_i \leqq \overline{v}_i$$

を満たしているとき

$$\|v\|_\infty \leqq \max_{1\leq i\leq n}\left(\max\left\{|\underline{v}_i|,|\overline{v}_i|\right\}\right)$$

によって，v の最大値ノルムの上限が計算できることがわかる．同様に，$A=(a_{ij})\in\mathbb{F}^{n\times n}$ で

$$\underline{a}_{i,j} \leqq a_{i,j} \leqq \overline{a}_{i,j}$$

となるとき

$$\|A\|_\infty = \max_{1\leq i\leq n}\left[\mathrm{fl}_\triangle\left(\sum_{j=1}^{n}\max\left\{|\underline{a}_{i,j}|,|\overline{a}_{i,j}|\right\}\right)\right]$$

によって A の最大値ノルムの上界が計算できることがわかる．

以上の事柄を基礎に，文献6)では式(1)の高速精度保証の理論が展開されており，それは本論文での議論の基礎ともなっている．本論文での新しい提案の基礎はつぎの小節で紹介される．

2.2 高精度内積計算アルゴリズム

1950年代のRunge-Kutta法に対する *Gill* の丸め誤差対策の技法に関する議論から出発して，浮動小数点数の加算の誤差に対する深い議論がされるようになった．その大きな成果の一つが次の1969年の *Knuth* の定理である．ここで，定理中のアルゴリズムは可読性の点からMATLABのプログラムに似た形式で表現しており，$\oplus$ 及び $\ominus$ は，それぞれ倍精度浮動小数点演算による加算と減算を意味する．また，特に断りのない限り最近点への丸めモードで演算は実行される．

定理1 (*Knuth*[4]) $a,b\in\mathbb{F}$ とする．$x\in\mathbb{F}$ と $y\in\mathbb{F}$ をつぎのアルゴリズム **TwoSum** によって計算すると $a+b=x+y$ が成立する．

function $[x,y]=$ **TwoSum**(a,b)
$x=a\oplus b$;
$b_V=x\ominus a$; $a_V=x\ominus b_V$; $b_R=b\ominus b_V$; $a_R=a\ominus a_V$;
$y=a_R\oplus b_R$;

(定理終)

定理1は和 $a+b$ を $a\oplus b$ で近似すると，その誤差 $a+b-(a\oplus b)$ が再び $\mathbb{F}$ の要素となり，$y\in\mathbb{F}$ として厳密に計算されるという驚くべき事実を示している．また，アルゴリズム **TwoSum** は単なる加減算の組み合わせで書かれており，条件分岐を含まないので，浮動小数点演算のステップ数は6ステップであっても，実際の数値計算においては，コンパイラの最適化のため，$x=a\oplus b$ を計算する時間に対して6倍もかからず，高速に計算される点に注意する．

1971年，*Dekker* はこれと同様な定理が浮動小数点数の乗算についても成立することを示した．まず，準備として，次の定理を示す．

定理2 (*Dekker*[2]) $a\in\mathbb{F}$ とする．つぎのアルゴリズム **Split** によって，仮数部の先頭ビットからの有効桁数(暗黙の1ビットを含む)がそれぞれ26ビット以内(27ビットめ以降はすべてゼロ)である a_H と a_L を計算すると，$a=a_H+a_L$ で $|a_H|\geq|a_L|$ となり，non-overlapping[2] な和となる．

function $[a_H,a_L]=$ **Split**(a)
$c=(2^{27}+1)\oplus a$; $d=c\ominus a$;
$a_H=c\ominus d$;
$a_L=a\ominus a_H$;

(定理終)

この定理を基に，*Dekker* は次の定理が成り立つことを示した．定理中のアルゴリズムは，*Veltkamp* によるものである．

定理3 (*Dekker*[2]) $a,b\in\mathbb{F}$ とする．$x\in\mathbb{F}$ と $y\in\mathbb{F}$ をつぎのアルゴリズム **TwoProduct** で計算すると，計算の途中でアンダフローが起きなければ，$a\times b=x+y$ が成り立つ．

function $[x,y]=$ **TwoProduct**(a,b)
$x=a\oplus b$;
$[a_H,a_L]=$ **Split**(a); $[b_H,b_L]=$ **Split**(b);
$c_1=x\ominus a_H\oplus b_H$; $c_2=c_1\ominus a_L\oplus b_H$;
$c_3=c_2\ominus a_H\oplus b_L$;
$y=a_L\oplus b_L\ominus c_3$;

(定理終)

アンダフローが起きた場合，著者らは次の結果を得ている．

定理4 (*Ogita-Rump-Oishi*[5]) 定理3においてアンダフローが計算途中で起きたとすると次の評価が成立する：

2 $x,y\in\mathbb{F}$ が“non-overlapping”であるとは，$x=r\times 2^s$ かつ $|y|<2^s$ あるいは $y=r\times 2^s$ かつ $|x|<2^s$ の様な整数 r と s が存在する事をいう($r=0$ のときは，適当な s に対してnon-overlappingとなる)．

$$|x+y-a\times b| \leqq 4\eta \qquad (6)$$

ただし，ηはアンダフローユニット(IEEE754倍精度のとき，$\eta = 2^{-1074}$)である．（定理終）

しかし，実際にはアンダフローが起きることはまれであることと，簡単のため，本論文ではアンダフローが起きた場合の処理については議論しないことにする．

以上の定理を基に，ベクトル$p = (p_1, p_2, \cdots, p_n)^T \in \mathbb{F}^n$に対して，ベクトル$q = (q_1, q_2, \cdots, q_n)^T \in \mathbb{F}^n$を計算するアルゴリズム**SumEFT**を導入する．ただし，p^Tはpの転置を意味する．このとき，$q_n = \mathrm{fl}_\square(\Sigma_{i=1}^n p_i)$となり，$q_1, q_2, \cdots, q_{n-1}$は$q_n$の計算誤差を表す．ただし，$\mathrm{fl}_\square$は最近点への丸めモードで浮動小数点演算を行うことを表す．

アルゴリズム1 (*Ogita-Rump-Oishi*[5]) ベクトルの総和の無誤差変換法：

function q = **SumEFT**(p)
$q_1 = p_1$;
for $i = 2 : n$
　$[q_i, q_{i-1}]$ = **TwoSum**(p_i, q_{i-1});
（アルゴリズム終）

定理1より，$p \in \mathbb{F}^n$を入力として，**SumEFT**で$q \in \mathbb{F}^n$を計算したとすると，明らかに

$$\sum_{i=1}^n p_i = \sum_{i=1}^n q_i \qquad (7)$$

が成り立つ[3]．

さらに，つぎの内積の高精度計算アルゴリズム**Dot2**を導入する．

アルゴリズム2 (*Ogita-Rump-Oishi*[5]) 内積計算の高精度計算法：

function s = **Dot2**(x, y)
$[p_1, s_1]$ = **TwoProduct**(x_1, y_1);
for $i = 2 : n$
　$[h_i, r_i]$ = **TwoProduct**(x_i, y_i);
　$[p_i, q_i]$ = **TwoSum**(p_{i-1}, h_i);
　$s_i = s_{i-1} \oplus (q_i \oplus r_i)$;
$s = p_n \oplus s_n$;
（アルゴリズム終）

この**Dot2**を用いると，$x, y \in \mathbb{F}^n$に対して内積$x^T y = \Sigma_{i=1}^n x_i \times y_i$を倍精度演算のさらに2倍の精度，すなわち4倍精度で計算したかのような結果が得られることが示されている[5]．

また，つぎの内積の無誤差変換アルゴリズム**DotEFT**を導入する．

アルゴリズム3 内積計算の無誤差変換法：

function v = **DotEFT**(x, y)
$[p_1, v_1]$ = **TwoProduct**(x_1, y_1);
for $i = 2 : n$
　$[h_i, v_i]$ = **TwoProduct**(x_i, y_i);
　$[p_i, v_{n+i-1}]$ = **TwoSum**(p_{i-1}, h_i);
$v_{2n} = p_n$;
（アルゴリズム終）

この**DotEFT**を用いると，$x, y \in \mathbb{F}^n$に対して

$$x^T y = \sum_{i=1}^{2n} v_i \qquad (8)$$

という内積からベクトルの総和への誤差の無い変換が実行される．また，このようにして得られたベクトルvに対して**SumEFT**を繰り返し適用することによって，任意に精度の良い内積計算が可能となる．

3. 高精度精度保証法

2002年，*Oishi*と*Rump*[6]は式(1)の数値解に対する高速精度保証法を提案した．ここでは，この精度保証法を内積の高精度計算アルゴリズム**Dot2**及び**DotEFT**を用いて高速かつ高精度に実装する方法を示す．

3.1 残差反復法

本節では，連立一次方程式(1)の数値解に対する残差反復法について考える．ただし，$A \in \mathbb{F}^{n\times n}$で$x, b \in \mathbb{F}^n$とする．$\tilde{x} \in \mathbb{F}^n$を式(1)の数値解とする．$A_e \in \mathbb{F}^{n\times(n+1)}$と$\tilde{x}_e \in \mathbb{F}^{n+1}$を

$$A_e = (A \mid b), \quad \tilde{x}_e = \begin{pmatrix} \tilde{x} \\ -1 \end{pmatrix} \qquad (9)$$

のように定義する．ただし，$-1 \in \mathbb{F}$である．このとき

$$A_e \tilde{x}_e = A\tilde{x} - b$$

が成立する．ここで

$$\tilde{r} = \left(\mathbf{Dot2}\left(A_e^{(1)}, \tilde{x}_e\right), \mathbf{Dot2}\left(A_e^{(2)}, \tilde{x}_e\right), \cdots, \mathbf{Dot2}\left(A_e^{(n)}, \tilde{x}_e\right)\right)^T \qquad (10)$$

とする．ただし，$A_e^{(i)}$は行列A_eの第i行目を転置した列ベクトルとする．これにより，式(10)の$\tilde{r}$は残差$r = A\tilde{x} - b$を内積の高精度計算アルゴリズム**Dot2**によって計算したものとなる．次に，倍精度浮動小数点演算を用いて計算した$Az = \tilde{r}$の数値解を$\tilde{z}$とすると

$$x_{\mathrm{new}} = \tilde{x} \ominus \tilde{z} \qquad (11)$$

によって，式(1)の新しい数値解x_{new}が得られる．これが本論文での残差反復法の実装法の提案である．

3 このような性質を持つアルゴリズムは“distillation algorithm”と呼ばれる．

この方法では，多倍長浮動小数点数パッケージ等を利用せず，内積の高精度計算アルゴリズム**Dot2**のみを用いているため，IEEE754の倍精度浮動小数点数規格に従う計算システムであればポータブルに実装できることがわかる(付録にMATLABでの実装例を示す)．

3.2 高速精度保証法

連立一次方程式(1)の高速精度保証では，つぎの定理が基礎となる[4]．

定理5 (*Banach*) A を $n\times n$ 実行列，b を n 次元実ベクトルとして連立一次方程式

$$Ax = b \tag{12}$$

を考える．適当な $n\times n$ 実行列 R が存在して

$$\|RA - I\|_\infty < 1 \tag{13}$$

を満たすとき，A は可逆で，任意の n 次元実ベクトル $\tilde{x}$ に対して

$$\|x^* - \tilde{x}\|_\infty \leq \frac{\|R(A\tilde{x} - b)\|_\infty}{1 - \|RA - I\|_\infty} \tag{14}$$

となる．ただし，I は n 次元単位行列で，$x^* = A^{-1}b$ とする．　(定理終)

定理5において，実際の応用では R としては A の近似逆行列を取り，$Ax = b$ の近似解を $\tilde{x}$ とする．この定理を基に，連立一次方程式の精度保証法を実装する．

A を $n\times n$ 行列として連立一次方程式 $Ax = b$ を解くとき，$\mathrm{cond}(A) = \|A\|\|A^{-1}\|$ をその条件数という．ここで，b は n 次元ベクトルである．b が $b + \Delta b$ に変化したとき，解は $\mathrm{cond}(A)\Delta b$ のオーダーで変化する．したがって，条件数の大きな問題では，右辺ベクトルの少しの変化が解の大きな変化を引き起こす．IEEE754の倍精度浮動小数点演算で計算する場合，仮数の精度は高々 $2^{-53} = 1.110\cdots\times 10^{-16}$ であるから，条件数が $\mathcal{O}(10^{16})$ に近くなると，丸め誤差の混入する数値計算での数値解の精度は仮数部の精度で1桁あるかないかというオーダーになり，計算の限界となる．ここでは，$\mathcal{O}(10^{14})$ 程度より大きな条件数を持つ問題を悪条件な問題と呼んでいる．ただし，本論文で扱う問題の範囲は A の条件数が $\mathcal{O}(10^{16})$ より小さい問題である．それよりも条件数が大きい問題に対する精度保証法については，文献8)を参照されたい．

この定理では $\tilde{x}$ は任意であるが，我々の実装法においては，3.1節で提案した高精度内積計算アルゴリズム**Dot2**を用いた残差反復法により必要な回数だけ反復した式(1)の近似解を取るものとする．また，式(10)では，高精度な残差 $r = A\tilde{x} - b$ の近似値 $\tilde{r}$ を計算しているが，精度保証の過程では，さらに $|r - \tilde{r}| \leqq q$ を満たすようなベクトル q を求める必要がある．ただし，2つのベクトル $x, y \in \mathbb{R}^n$ に対して $x \leqq y$ は，すべての i に対して $x_i \leqq y_i$ が成り立つことを意味する．そこで，**DotEFT**及び**SumEFT**を用いて高精度に残差の近似値とその誤差限界を求めるMATLABのアルゴリズムも実装例として付録に示しておく．

したがって，IEEE754の倍精度浮動小数点数規格に従う計算システムであれば，本論文におけるアルゴリズムはすべてポータブルに実装できることがわかる．

4. 数値例

ここでは，提案したアルゴリズムの有効性を確認するために行った数値実験の中から幾つかを紹介する．

4.1 悪条件の問題

行列の条件数が $\mathcal{O}(10^{14})$ 以上となる問題の例として，ヒルベルト行列を係数行列とする問題を考える．$n\times n$ ヒルベルト行列 H_n はその第 (i,j) 成分を h_{ij} とするとき

$$h_{ij} = \frac{1}{i + j - 1} \tag{15}$$

で定義される行列である．ここでは，$n = 11$ のヒルベルト行列 H_{11} に，要素 h_{ij} の分母の公倍数 s をかけた行列 $K = s\cdot H_{11}$ を係数行列とする連立一次方程式を考える．また，$x = (1, 1, \cdots, 1)^T$ が厳密解であるように右辺ベクトルは $b = \mathrm{fl}(K\cdot(1, 1, \cdots, 1)^T)$ とする．このとき，丸め誤差は発生しない．すなわち，11次元ベクトル x についての連立一次方程式

$$Kx = b \tag{16}$$

を考える．K の条件数は，$\mathrm{cond}_\infty(K) = \|K\|_\infty\|K^{-1}\|_\infty = 1.23\cdots\times 10^{15}$ である．これを本論文で提案した高精度かつ高速な残差反復法と精度保証プログラムを用いてMATLAB上で解いた例を以下に示す．

```
>> K = myhilb(11);     % K: the 11 x 11 Hilbert matrix
>> b = K*ones(11,1);   % b: a right-hand side vector
>> R = inv(K);         % R: an approximate inverse of K
>> xs = R*b;           % xs: an approximate solution
of Kx = b
>> xs(11)
ans =
   0.99945068359375
>> [e,alpha] = vlin(K,b,R,xs,0)      % e: verified er-
ror bound of xs
e =
```

4 この定理は古くから知られており，実質的に*Banach*の縮小写像の原理が起源であると思われるので，*Banach*に負うものとする．

```
   0.014274999840825
alpha =
   0.041381835937500
>> xs = iter_ref(K,b,R,xs); xs(11)   % 1st iterative re-
finement of xs
ans =
   1.000000026822090
>> e = vlin(K,b,R,xs,1,alpha)
e =
   3.004930254062938e-006
>> xs = iter_ref(K,b,R,xs); xs(11)   % 2nd iterative re-
finement of xs
ans =
   1.000000000032014
>> e = vlin(K,b,R,xs,1,alpha)
e =
   1.220669425188106e-008
>> xs = iter_ref(K,b,R,xs); xs(11)   % 3rd iterative re-
finement of xs
ans =
   1.000000000000023
>> e = vlin(K,b,R,xs,1,alpha)
e =
   2.285908904331256e-011
>> xs = iter_ref(K,b,R,xs); xs(11)   % 4th iterative re-
finement of xs
ans =
   1.000000000000000
>> e = vlin(K,b,R,xs,1,alpha)
e =
   5.576489225820260e-014
>> xs = iter_ref(K,b,R,xs); xs(11)   % 5th iterative re-
finement of xs
ans =
    1
>> e = vlin(K,b,R,xs,1,alpha)
e =
   1.166207784303186e-016
>> xs = iter_ref(K,b,R,xs); xs(11)   % 6th iterative re-
finement of xs
ans =
    1
>> e = vlin(K,b,R,xs,1,alpha)
e =
    0
```

この例においては，R = inv(K) という命令によって，行列 K の逆行列を倍精度浮動小数点演算で求めている．丸め誤差のため，計算された R は K の近似逆行列になっている．式(16)の初期近似解としては，方程式を倍精度浮動小数点演算で計算した解(xs = R*b という命令で求めた解)を採用している．尚，xs(11) という命令は，ベクトル x の第11番目の成分を表示せよとの命令になっている．また，vlin で1回目のみ出力している alpha は $\|RA-I\|_\infty$ の上限であり，alpha < 1 であれば提案方式による精度保証が可能となる．これは，解がどのように収束するかを見るために，例としてxs の一つの成分を表示させたものである．今の場合は，厳密解は $x=(1,1,\cdots,1)^T$ であるから，小数点以下4桁目以降に誤差があることが見て取れる．

次の命令では，計算した近似解 $\tilde{x}$ = xs のもつ精度を本論文で提案した方法に基づいて実装した高精度精度保証プログラム vlin によって計算している．その結果，誤差 $e=\|A^{-1}b-\tilde{x}\|_\infty$ は $1.427\cdots\times10^{-2}$ 以下であるとの見積もりが得られている．

その次の命令では，3.1節で提案した高精度内積計算法に基づいた残差反復法により，一回反復した xs を計算している．そして，その第11成分 xs(11) を表示している．この結果から，近似解の小数点以下7桁目に誤差があることが見て取れる．その次の命令ではその誤差を $\|A^{-1}b-x\|_\infty \leqq 3.004\cdots\times10^{-6}$ 以下であるとの見積もりが得られている．

以下これが繰り返されている．一回の残差反復では 10^{-3} 程度のオーダで残差反復した解の正しい桁が増えていき，6回目の反復で最後の桁(least significant digit)まで正しく計算されたことがわかる[5]．また，誤差評価もそのことを正しく評価できていることがわかる．

4.2 良条件の問題

条件数は $\mathcal{O}(10^{14})$ に比較して小さいが，問題の次元が $n=1000$ と前節の問題に比して大きな問題を例として取り上げる．すなわち，A は 1000×1000 実行列でその要素は乱数であるとする．

```
>> n = 1000;
>> A = randn(n);      % A: an n x n random matrix
>> b = A*ones(n,1);
>> cond(A)            % condition number estimator
ans =
   2.450763206290818e+004
```

より，A の条件数は $\mathcal{O}(10^4)$ 程度である．b = A*ones(n,1); からわかるように，$b=A\cdot(1,1,\cdots,1)^T$ として式(1)を考えている．ただし，丸め誤差の存在により，この場合の式(1)の厳密解は $x=(1,1,\cdots,1)^T$ と若干異なることに注意する．

このようにして作成した A と b を用いて，$Ax=b$ の精度保証を実行する．

```
>> R = inv(A);
>> xs = R*b; xs(1000)
ans =
   1.000000000000102
```

5 誤差限界 $e=0$ ということは，近似解 $\tilde{x}$ は厳密解そのものであることを意味する．

```
>> [e,alpha] = vlin(A,b,R,xs,0)
e =
   2.219658324896038e-011
alpha =
   8.249229812288443e-010
>> xs = iter_ref(A,b,R,xs); xs(1000)
ans =
   1.00000000000002
>> e = vlin(A,b,R,xs,1,alpha)
e =
   1.107097103095627e-016
```

この例では，一回の残差反復でほぼ最終桁まで正しい結果が得られていることがわかる．また，本論文で提案している高精度精度保証プログラムがシャープな誤差評価を与えていることがわかる．この点に関し，残差を高精度に計算しない従来の精度保証プログラム vlin_pre で精度を計算すると

```
>> e = vlin_pre(A,b,R,xs)
e =
   4.566030510093495e-011
```

となって，本来，近似解が持っている精度をシャープに評価できていないことがわかる．ここで，式(1)の近似解を求める時間(A のLU分解にかかる時間)と残差を高精度に計算しない従来の精度保証プログラムで精度保証する時間(本論文ではこれを標準精度保証法の計算時間という)と残差を高精度に計算する本論文で提案した方法での時間(新法の計算時間と呼ぶ)を**表1**に示す．表中に示した数字は各次元とも，異なった100のランダム行列 A を生成し，$b = A \cdot (1, 1, \cdots, 1)^T$ として，式(1)を解いた時間を平均したものである．尚，精度保証の時間には A の近似逆行列 R を計算する時間を含めていない．計算に用いたコンピュータはMac Power Book G4(1.25GHz CPU)で計算に用いた言語はSlab[3]である．SlabはMATLABに似た数値計算ツールだが，精度保証ができるように改良されたものであり，MATLABと同様，LAPACKをもとにして，これを使いやすくするインタプリタとなっている．

表1 精度保証のための計算時間 [sec]（第3, 4列における括弧内の数字はLU分解の時間に対して何倍計算時間を要しているかを示している）

n	LU分解	標準精度保証法	新精度保証法
200	0.02	0.04 (2.00)	0.04 (2.00)
300	0.04	0.14 (3.50)	0.17 (4.25)
400	0.07	0.31 (4.42)	0.35 (5.00)
500	0.13	0.50 (3.85)	0.63 (4.84)
600	0.21	0.87 (4.14)	1.01 (4.81)
700	0.30	1.31 (4.37)	1.53 (5.10)
800	0.45	2.23 (4.95)	2.37 (5.23)
900	0.59	2.89 (4.90)	3.07 (5.20)
1000	0.81	3.84 (4.74)	4.31 (5.32)

近似解(LU分解)を計算する計算のため浮動小数点演算の回数(flopsという単位が標準で用いられる．floating point operationsの略である．)は四則演算をすべて一回と数えると $2/3n^3$ flopsである．一方，標準精度保証法も新精度保証法も近似逆行列 R を計算する時間は含めていないので，行列の積を2回計算する手間の $4n^3$ flopsである．よって，浮動小数点演算の回数の比でいえば，精度保証にかかる計算時間は標準法も新しい方法もともにLU分解法の計算時間の6倍となる．表1に示した実際の数値計算例においてはこの比は2つの精度保証法とも6倍より少なくなっている．これは行列の積の計算がATLASによって生成された最適化BLASを基に行われていることに起因する．すなわち，最適化BLASはキャッシュヒット率を最大にするように行列の積を計算するように生成されている．ガウスの消去法の中にもこの最適化は生かされているが，行列の積の方が単純なアルゴリズムで計算できるので，この最適化の効果がより生かされる．したがって，精度保証に必要な主計算が行列の積であることから，実際の計算時間が浮動小数点演算の回数の比よりは短縮されているのである．

また，標準精度保証法の計算時間と新しい精度保証法の計算時間は，後者が高精度な内積計算を残差の計算に用いているので，よりかかっているが，ほぼ同じオーダーであることが確認された．すなわち，残差の計算において高精度な内積計算を用いてもその演算回数は $\mathcal{O}(n^2)$ であり，しかも高精度な内積計算法に要する時間が浮動小数点演算での内積計算に要する時間の一定の倍率でしか時間がかからないことからこれは理論的にも理解される．すなわち，新しい精度保証法は標準精度保証法の高速性を損なうことなく，高精度性が達成されていることがわかる．

謝 辞

本研究は文部科学省科学研究費補助金(特別推進研究「精度保証付き数値計算学の確立」No.17002012)の補助を受けた．

A 精度保証プログラム

ここでは，実際の精度保証に用いたプログラムを示す[6]．これらはMATLABの関数である．

6 %から行末まではコメント

以下は，それぞれ**TwoSum**，**Split**及び**TwoProduct**である．

```
function [x,y] = TwoSum(a,b)
% TwoSum: error-free transformation of sum a + b to x + y.
  x = a + b;
  t = x - a;
  y = (a - (x - t)) + (b - t);

function [ah,al] = Split(a)
% Split: error-free transformation of a to ah + al.
  c = (2^27 + 1)*a;
  ah = c - (c - a);
  al = a - ah;

function [x,y] = TwoProduct(a,b)
% TwoProduct: error-free transformation of
% product a x b to x + y.
  x = a*b;
  [ah,al] = Split(a); [bh,bl] = Split(b);
  c1 = x - ah*bh; c2 = c1 - al*bh; c3 = c2 - ah*bl;
  y = al*bl - c3;
```

以下は，それぞれ**Dot2**，**SumEFT**及び**DotEFT**である．

```
function res = Dot2(x,y)
%  Dot2: calculation of dot product as if computed in
%  quadruple precision.
  n = length(x);
  [p,s] = TwoProduct(x(1),y(1));
  for i=2:n
      [h,r] = TwoProduct(x(i),y(i));
      [p,q] = TwoSum(p,h);
      s = s + (q + r);
  end
  res = p + s;

function q = SumEFT(p)
%  SumEFT: error-free transformation of summation.
%    input
%      p: a real n-vector
%    output
%      q: a real n-vector s.t. sum(q) = sum(p)

  q = p;
  for i=2:length(q)
      [q(i),q(i-1)] = TwoSum(q(i),q(i-1));
  end

function v = DotEFT(x,y)
%  DotEFT: error-free transformation of
%  dot product to summation.
%    input
%      x,y: real n-vectors
%    output
%      v: a real 2n-vector s.t. sum(v) = dot(x,y)
  n = length(x);
  v = zeros(2*n,1);
  [t,v(1)] = TwoProduct(x(1),y(1));
  for i=2:n
      [h,v(i)] = TwoProduct(x(i),y(i));
      [t,v(n+i-1)] = TwoSum(t,h);
  end
  v(2*n) = t;
```

残差反復法のMATLABでの実装法の例を以下に示す．ただし，アルゴリズム中のR及びx_sは，それぞれAの近似逆行列，$Ax = b$の近似解を表す．

```
function xs = iter_ref(A,b,R,xs)
%  iter_ref: iterative refinement for an approximate
%  solution of Ax = b using accurate dot product.
%  input
%    A: a real n x n matrix, b: real n-vector
%    R: an approximate inverse of A, xs: an approximate
%    solution of Ax = b
%  output
%    xs: an updated approximate solution of Ax = b

  [m,n] = size(A);  % size of A
  r = zeros(n,1);
  for i=1:n         % residual A*xs - b by Dot2
      r(i) = Dot2([A(i,:),b(i)]',[xs; -1]);
  end
  z = R*r;          % approximate solution of Az = r
  xs = xs - z;      % update xs
```

以下に，高精度な残差計算とその誤差限界を計算するプログラムを示す．ただし，

```
system_dependent('setround',mode)
```

は丸めモードの変更をする命令で，mode = 'nearest'は最近点への丸めモードに，mode = -infは，$-\infty$方向の丸めモードに，mode = +infは$+\infty$方向の丸めモードにそれぞれ変更することを表す．

```
function [rmid,rrad] = accresidual(A,b,xs)
%  accresidual: accurate computation of A*xs - b and
%  its error bound.
%  input
%    A: a real n x n matrix, b: a real n-vector
%    xs: an approximate solution of Ax = b
%  output
%    rmid: an approximation of A*xs - b
%    rrad: an error bound of rmid

  system_dependent('setround','nearest');
  n = length(b);
  rmid = zeros(n,1);
  rrad = rmid;
```

```
for i=1:n
  v = DotEFT([A(i,:),b(i)]',[xs; -1]);
  v = SumEFT(v);
  rmid(i) = v(end);
  system_dependent('setround',+inf);
  rrad(i) = sum(abs(v(1:end-1)));
  system_dependent('setround','nearest');
end
```

最後に，連立一次方程式 $Ax = b$ の精度保証アルゴリズムを以下に示す．

```
function [e,alpha] = vlin(A,b,R,xs,mode,alpha)
%  vlin: fast verification of an approximate solution
%  xs of Ax = b.
%  input
%    A: a real n x n matrix, b: real n-vector
%    R: an approximate inverse of A, xs: an approximate
%    solution of Ax = b
%    mode: If mode = 0, then compute alpha s.t.
%    alpha >= norm(RA - I,inf).
%          Otherwise, skip calculation of alpha.
%  output
%    e: an error bound of xs
%    alpha: alpha >= norm(RA - I,inf)

  [m,n] = size(A);
  if mode == 0
   I = speye(n);          % I: n x n identity matrix
   system_dependent('setround',-inf);
                          % rounding to -infinity
   Gl = R*A - I;          % lower bound of R*A - I
   system_dependent('setround',+inf);
                          % rounding to +infinity
   Gu = R*A - I;          % upper bound of R*A - I
   Gu = max(abs(Gl),abs(Gu));
                          % upper bound of |R*A - I|
   alpha = norm(Gu,inf);
                          % alpha >= norm(RA - I,inf)
  end

  if alpha < 1
   [rmid,rrad] = accresidual(A,b,xs);
                          % accurate residual and its error bound
   system_dependent('setround',+inf);
   t = R*rrad;
   vu = R*rmid + t;
   system_dependent('setround',-inf);
   vl = R*rmid - t;
   vu = max(abs(vl),abs(vu));   % vu >= |R*(A*xs - b)|
   d = 1 - alpha;
   system_dependent('setround',+inf);
   e = norm(vu,inf)/d;          % error bound of xs
   system_dependent('setround','nearest');
  else
   error('verification failed');
  end
```

参 考 文 献

1) ANSI/IEEE: IEEE Standard for Binary Floating Point Arithmetic, Std 754-1985 ed., New York, IEEE (1985)
2) T. J. Dekker: A floating-point technique for extending the available precision, Numer. Math., **18**, 224/242 (1971)
3) R. B. Kearfott, M. Neher, S. Oishi, F. Rico: Libraries, tools, and interactive systems for verified computations: four case studies, in Numerical Software with Result Verification (R. Alt, A. Frommer, R. B. Kearfott, and W. Luther eds.), Lecture Notes in Computer Science, 2991, Springer-Verlag, Heidelberg (2004)
4) D. E. Knuth: The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Reading, Massachusetts: Addison-Wesley (1969)
5) T. Ogita, S. M. Rump, S. Oishi: Accurate Sum and Dot Product, SIAM J. Sci. Comput., **26**-6, 1955/1988 (2005)
6) S. Oishi, S. M. Rump: Fast verification of solutions of matrix equations, Numer. Math., **90**-4, 755-773 (2002)
7) T. Yamamoto: Error bounds for approximate solutions of systems of equations, Japan J. Appl. Math., **1**-1, 157-171 (1984)
8) 太田貴久，荻田武史，S. M. Rump，大石進一：悪条件連立一次方程式の精度保証付き数値計算法，日本応用数理学会論文誌，**15**-3，269/287 (2005)
9) 荻田武史，大石進一：大規模連立一次方程式のための高速精度保証法，情報処理学会論文誌: 数理モデル化と応用，**46**-SIG10 (TOM12)，10/18 (2005)