平成16年度

卒業論文

大規模スパース行列に対する連立一次方程式の精度保証付き数値計算

平成１７年２月２日

指導教授： 大石　進一　教授

早稲田大学　理工学部　情報学科

1g01p100-2　水島　裕

目 次

第 1 章

序論

1.1 背景

数値計算は数学または応用数学の一分野である. 数学には, 代数学, 幾何学, 解析学, 微分方程式などさまざまな分野があり, 人類の豊かな知識体系の一部をなしている. 数学は科学, 工学に現れる現象, 関係, 設計過程などをモデル化するのにその威力を発揮している. そして, それを解くことで現象を解析し, 工学的な製品を設計することが多い. したがって微分方程式を解く, 積分を計算するといった連続数学を解くことが多くなる. しかし, 連続数学を解くことは難しい場合が多い. そこで, 計算機を利用して連続数学の問題を解こうとすることから数値計算の研究が進み, 理論が作られるようになった.

しかし, 電子計算機を用いた数値計算での時間, 桁, 次元は有限であるため, 実数演算を浮動小数点演算で近似するため, 丸め誤差が生じたり, 微分方程式を有限次元方程式に近似するために生じる打ち切り誤差などさまざまな誤差が含まれる.

特に丸め誤差においては 1 回の演算で生じる誤差はたかだか有限桁の最下位桁の単位程度であるが, 莫大なステップの各ステップにおいて生じるため, 数値計算の誤差を厳密に評価することは非常に困難であると考えられていた.

これに対し, 真の値を含む区間を数とみなす区間解析の概念が導入され, 丸め誤差が存在する浮動小数点演算を用いても, 数学的に正しい結果を数値計算により導く原理が示された. この区間解析の概念は研究され, いろいろな成果が得られたが, 一方では単に浮動小数点演算を区間演算に置き換えると, 多くの場合区間の幅が大幅に増大し, 意味ある結論が得られないことがわかった.

この困難を解消する方法が Kulisch,Krawczyk などにより示されたが, それは, 平均値形式などの関数の評価に微分係数の区間評価を用いる方法や, 近似解が得られた後にその周りで Newton 反復作用素に対して縮小写像原理が成り立つかどうかにより, 区間演算で数値的に検証する方法である. その結果, 真の解を含む区間を縮小させることも可能となり, 数値計算の誤差を厳密に評価することが原理的に可能であることがさまざまな例から明らかにされてきた.

この区間解析や近年能力が大幅に向上してきた計算機を利用することにより, 連続数学の

問題に対しても演算結果が誤差を含んでいても数学的に正しいと保証されるを結論を導くための理論と技術の開発を, 精度保証付き数値計算という.

浮動小数点演算における丸め誤差を含む演算結果の保証が理論的にも実用的にも高い精度で効率よく実現できることが明らかになり, 計算の信頼性という問題は, 数学としての数値解析の観点からようやく取り上げられることとなった. これは単に丸め誤差を厳密に評価するということだけにとどまらず, 科学技術計算の元となった数値解析アルゴリズムそのものに影響を与え, さまざまな数理科学上にあらわれる問題の解を数学的な厳密さで検証する方向にまで展開しつつある.

このように, 精度保証付き数値計算は計算の信頼性の立場から見た今度の科学技術の計算法のあるべき 1 つの方向として考えられている.

1.2 本論文の目的

精度保証付き数値計算が重要であることは上記の内容などから明らかであるが, 現時点で全ての数値計算において効率の良い精度保証を行えるまでに至っていない. そこで数値計算の中から連立一次方程式の精度保証付き数値計算に対し,10000×10000 以上の大規模スパース行列の計算及び, 精度保証を可能にすることを目的とする.

1.3 本論文の構成

本論文の構成は以下の通りである.

第 2 章では, 浮動小数点システムについて述べる.

第 3 章では, 区間演算システムについて述べる.

第 4 章では,MATLAB におけるスパース行列の反復解法について述べる.

第 5 章では, スパース行列に対する連立一次方程式の精度保証付き数値計算について述べる.

第 6 章では, 前章のアルゴリズムにおけるソースを記載する.

第 7 章では, 実際に精度保証付き数値計算を実行し, その実験結果を記載する.

第 8 章では, 本論文の統括をする.

第 2 章

浮動小数点

2.1 はじめに

本章では, 精度保証つき数値計算の基礎となる, 浮動小数点システムについて述べる.

2.2 浮動小数点数

現代の数値計算では, 浮動小数点数が標準として用いられている. 浮動小数点数は現在では標準化が進められ, ほとんどのパソコンやワークステーション, ベクトル計算機, 並列計算機などで同じ形式の浮動小数点体系が用いられている. この体系に基づいて, 高速に浮動小数点演算が実行される回路がコンピュータに実装されているため, 数値計算は浮動小数点数体系上で行われることがほとんどである. そこで, このような浮動小数点数の体系の上での数値計算の精度を保証することが重要である.

そのために, まず浮動小数点数について説明する. 浮動小数点数については IEEE 標準 754 (IEEE standard 754, 以下 IEEE 754 と略記する.) がパソコンやワークステーションなどをはじめとして, 多くのコンピュータで標準的に用いられている. 本論文では, 以下 IEEE 754 に基づく 2 進数浮動小数点数システムを考えることにする.

2.2.1 浮動小数点数

IEEE 754 では 4 つのタイプの数が用意されている. それは規格化 2 進浮動小数点数, 零, 非規格化 2 進浮動小数点数, NaN (非数) である.

2 進規格化浮動小数点数

2 進規格化浮動小数点数とは

$$a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_N}{2^N}\right) \times 2^e, \quad (d_i = 0 \text{ または } 1). \tag{2.1}$$

と書ける数をいう. $e_{\min}$ を負の整数, $e_{\max}$ を正の整数として, e は $e_{\min} \leq e \leq e_{\max}$ となる整数である.

$$m = \frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_N}{2^N}. \tag{2.2}$$

を符合付き仮数といい, e を指数という. 指数 e も 2 進数で表される. 通常, 単精度, 倍精度 (8 byte = 64 bit) , 拡張倍精度 (10 byte = 80 bit) の浮動小数点システムがあるが, それぞれつぎのような浮動小数点システムである.

$$N = 24, \quad (-126 \leq e \leq 127),$$
$$N = 53, \quad (-1022 \leq e \leq 1023),$$
$$N = 64, \quad (-16382 \leq e \leq 16383).$$

規格化 2 進浮動小数点システムにおいて表される数の絶対値の最大値は

$$x_{\max} = \left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^N}\right) 2^{e_{\max}}. \tag{2.3}$$

であり, その最小値は

$$x_{\min} = \frac{1}{2} 2^{e_{\min}}. \tag{2.4}$$

である. 倍精度数では (2.3),(2.4) はそれぞれ

$$(2 - 2^{52}) \times 2^{1023} = 1.7976931348623157 \cdots \times 10^{308}, \tag{2.5}$$
$$2^{-1023} = 2.22507358507201 \cdots \times 10^{-308}. \tag{2.6}$$

である. $|x| > x_{\max}$ のときにオーバーフローが生じたという.

倍精度浮動小数点数においては, 仮数部が 53bit である.

$$2^{-53} = 1.1102230246 \cdots \times 10^{-16}. \tag{2.7}$$

より, 倍精度浮動小数点数は 10 進数で約 16 桁の精度がある.

零

零は規格化されて

$$+\left(\frac{0}{2} + \frac{0}{2^2} + \frac{0}{2^3} + \cdots + \frac{0}{2^N}\right) 2^{e_{\min}}. \tag{2.8}$$

と表される.

非規格化 2 進浮動小数点数

IEEE 754 では浮動小数点数は指数部が $e_{\min}$ となったとき, 仮数部の最初の桁が 1 より小さい数を表すために, デフォルトで最初の桁を 1 とすることをやめ, ここが 0 となる数を置くことを許す規格となっている. これを非規格化数という. 非規格化数の範囲に数が入ることを, 漸近アンダーフローという.

このような非規格化浮動小数点数の正で最も小さな数は

$$2^{-1074} = 4.94065645841246544\cdots \times 10^{-324}. \tag{2.9}$$

である. これ以下の数になると, アンダーフローが生じたという.

NaN

このほかに, IEEE 754 ではつぎのような特別な数が用意されている.

(a) NaN(Not a Number) は $\sqrt{-5}, \dfrac{\infty}{\infty}, +\infty + (-\infty)$ など不当な演算の結果として得られる.

(b) $\pm\infty$ はオーバーフローの結果や零で割った結果として得られる.

(c) ± 0 はアンダーフローか $\pm\infty$ での割り算の結果として得られる.

2.2.2 浮動小数点数と丸め

IEEE 754 では, つぎの 4 つの丸めのモードが指定できる. c を実数 ($c \in \boldsymbol{R}$) とする.

上向きの丸め

c 以上の浮動小数点数の中で最も小さい数に丸める. これを $\triangle : \boldsymbol{R} \to \boldsymbol{F}$ と表す. アルゴリズムでの表記は UP とする.

下向きの丸め

c 以下の浮動小数点数の中で最も大きい数に丸める. これを $\triangledown : \boldsymbol{R} \to \boldsymbol{F}$ と表す. アルゴリズムでの表記は DOWN とする.

最近点への丸め

c に最も近い浮動小数点数に丸める. これを $\square : \boldsymbol{R} \to \boldsymbol{F}$ と表す. アルゴリズムでの表記は NEAR とする. もし, このような点が 2 点ある場合には, 仮数部の最後のビットが偶数である浮動小数点数に丸める. これを偶数丸め方式という.

切り捨て

絶対値が c 以下の浮動小数点数の中で, c に最も近いものに丸める. アルゴリズムでの表記は ZERO とする.

2.2.3 浮動小数点数の性質

丸めの演算を写像として $\bigcirc : \boldsymbol{R} \to \boldsymbol{F}$ と書く. すなわち, $\bigcirc$ は $\triangle, \triangledown, \square$ のいずれかと考える. IEEE 754 では, 丸めの演算はつぎの条件を満たす.

$$\begin{aligned} &\bigcirc x = x, \quad (\text{任意の } x \in \boldsymbol{F} \text{について}), \\ &x \leqq y \to \bigcirc x \leqq \bigcirc y, \quad (\text{任意の } x, y \in \boldsymbol{R} \text{について}). \end{aligned} \tag{2.10}$$

また, $x \in \boldsymbol{F}$ のとき, 符号を変えることにより, $-x$ や $|x|$ が得られるので, これらは正確に計算される.

IEEE 754 では, つぎの性質が成立する.

$$\begin{aligned} &\square\,(-x) = -\square\, x, \quad (\text{任意の } x \in \boldsymbol{R} \text{について}), \\ &\triangle(-x) = -\triangledown\, x, \quad (\text{任意の } x \in \boldsymbol{R} \text{について}), \\ &\triangledown(-x) = -\triangle\, x, \quad (\text{任意の } x \in \boldsymbol{R} \text{について}). \end{aligned} \tag{2.11}$$

IEEE 754 では, 浮動小数点数演算 ($\boldsymbol{F}$ 上での四則演算) は丸めとの関係により, つぎのように定義されている. $\cdot \in \{+, -, \times, /\}$, $\bigcirc \in \{\triangle, \triangledown, \square\}$ のとき

$$x \bigodot y = \bigcirc(x \cdot y), \quad (\text{任意の } x, y \in \boldsymbol{R} \text{について}). \tag{2.12}$$

この式は, 左辺の浮動小数点数の四則演算の結果 $x \bigodot y$ は, 右辺の数学的に正しい (実数としての) 四則演算の結果 $x \cdot y$ を指定された丸めを行って得られた数 $\bigcirc(x \cdot y)$ に一致するように計算することを表している.

また, 平方根も

$$(\sqrt{x})_{fp} = \bigcirc(\sqrt{x}), \quad (\text{任意の } x \in \boldsymbol{F} \text{について}). \tag{2.13}$$

と, 浮動小数点数演算によって計算された平方根 $(\sqrt{x})_{fp}$ は, 正確な実数演算で計算された平方根 $\sqrt{x}$ を指定された丸めの方向へ丸めた数となる.

第 3 章

区間演算

3.1 はじめに

本章では, 精度保証付き数値計算の基礎となる区間演算システムについて述べる.

3.2 区間解析

以前に述べたような精度保証付き数値計算の原理は, 実数値で与えられる真の解の上限と下限を浮動小数点演算により計算しようとするものであった. このような考え方は区間解析と呼ばれる数学的理論と密接な関係がある. この区間解析についてまず簡単に述べる.

区間解析では区間を数の拡張と考える. そして, その四則演算が定義される. これを区間演算という.

3.3 区間演算

機械区間演算を定義するのが目標であるが, まず, 実数上での演算を仮定する区間演算の概念を説明するここでの議論において四則演算は実数の四則演算とする.

区間解析においては, 区間とは

$$[\underline{x}, \overline{x}] = \{x \in \boldsymbol{R} \mid \underline{x} \leq x \leq \overline{x}\} \tag{3.1}$$

と表される閉区間であるとする. ただし,$\underline{x} \leq \overline{x} \ \in \boldsymbol{R}$ で, それぞれの区間の下端, 上端とする. 以下, 記号の節約のため,

$$[x] = [\underline{x}, \overline{x}] \tag{3.2}$$

と区間を表すことにし, 区間 $[x]$ と単に書けば, それは $[x] = [\underline{x}, \overline{x}]$ を表すものとする. $\underline{x} = \overline{x}$ となる区間 $[x]$ は点区間という. 点区間は実数であるので, 点区間 $[x]$ の表す実数を x と書くことにする.

区間 $[x]$ について以下を定義する.

$$d([x]) = \overline{x} - \underline{x}, \ r([x]) = \frac{\overline{x} - \underline{x}}{2}, \ m([x]) = \frac{\overline{x} + \underline{x}}{2} \tag{3.3}$$

$d([x]), r([x]), m([x])$ をそれぞれ区間 $[x]$ の直径, 半径, 中心という.

区間 $[x]$ の最小絶対値と最大絶対値をそれぞれ次で定義する.

$$\begin{aligned} \langle [x] \rangle &= \min\{|x| \mid x \in [x]\} \\ |[x]| &= \max\{|x| \mid x \in [x]\} = \max\{|\underline{x}|, |\overline{x}|\} \end{aligned} \tag{3.4}$$

二つの区間 $[x], [y]$ が与えられたときその二つの区間の四則演算を次で定義する.

$$[x] \circ [y] = \{x \circ y \mid x \in [x], y \in [y]\} \tag{3.5}$$

ただし,$\circ \in \{+, -, \times, /\}$. これを区間演算という. この定義では区間演算は無限回の実数演算をしないと実行できないような印象を与えるが, 実は次が成立する.

$$\begin{aligned} [x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\ [x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\ [x] \times [y] &= [\min\{\underline{x}\underline{y}, \overline{x}\overline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\overline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}\}] \\ [x]/[y] &= [x] \times \left[\frac{1}{\overline{y}}, \frac{1}{\underline{y}}\right], \ (0 \notin [y]) \end{aligned} \tag{3.6}$$

これから区間演算が有限回の実数演算で実行できることがわかる. かけ算 (表 3.1) と割算 (表 3.2) においては, 場合分けにより, より少ない手間で計算する次のような計算規則も簡単に導ける.

	$y \geq 0$	$y \ni 0$	$y \leq 0$
$x \geq 0$	$[\underline{x}\underline{y}, \overline{x}\overline{y}]$	$[\overline{x}\underline{y}, \overline{x}\overline{y}]$	$[\overline{x}\underline{y}, \underline{x}\overline{y}]$
$x \ni 0$	$[\underline{x}\overline{y}, \overline{x}\overline{y}]$	$[\min\{\underline{x}\overline{y}, \overline{x}\underline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\overline{y}\}]$	$[\overline{x}\underline{y}, \underline{x}\underline{y}]$
$x \leq 0$	$[\underline{x}\overline{y}, \overline{x}\underline{y}]$	$[\underline{x}\overline{y}, \underline{x}\underline{y}]$	$[\overline{x}\overline{y}, \underline{x}\underline{y}]$

Table 3.1: 区間 $[x], [y]$ の乗算 $[x][y]$

区間演算については, 包含関係における単調性

$$[x] \subseteq [x'], \ [y] \subseteq [y'] \Rightarrow [x] \circ [y] \subseteq [x'] \circ [y'], \ \circ \in \{+, -, \times, /\} \tag{3.7}$$

	$y \geq 0$	$y \leq 0$
$x \geq 0$	$[\underline{x}/\overline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\underline{y}]$
$x \ni 0$	$[\underline{x}/\underline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\overline{y}]$
$x \leq 0$	$[\underline{x}/\underline{y}, \overline{x}/\overline{y}]$	$[\overline{x}/\underline{y}, \underline{x}/\overline{y}]$

Table 3.2: 区間 $[x], [y]$ の除算 $[x]/[y]$

が成立する.

また, 加法と乗法に関し交換則と結合則が成立する.

$$
\begin{aligned}
&[x] \circ [y] = [y] \circ [x], \ \circ \in \{+, \times\} \\
&[x] \circ ([y] \circ [z]) = ([x] \circ [y]) \circ [z], \ \circ \in \{+, \times\}
\end{aligned} \tag{3.8}
$$

しかし, 加法と乗法の逆元は存在しない. すなわち,$-[x] = [-\overline{x}, -\underline{x}]$ であるが

$$
\begin{aligned}
&0 = [0] \subseteq [x] - [x] = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] \\
&1 = [1] \subseteq [x]/[x]
\end{aligned} \tag{3.9}
$$

となることがわかる. 上式で等号は $[x]$ が点区間のときのみ成立する.

また, 分配則も区間演算に対しては成立しない. そのかわり次の劣分配則が成立する.

$$
[x] \times ([y] + [z]) \subseteq [x] \times [y] + [x] \times [z] \tag{3.10}
$$

この式で等号は例えば区間 $[y]$ と $[z]$ が同じ符合をもつときに成立する.

関数 $f : D \subset \boldsymbol{R} \to \boldsymbol{R}$ を領域 D の任意の閉区間の上で連続な初等関数とする. 関数 f を次のようにして $I\boldsymbol{R}$ 上の関数に拡張することができる.

$$
f([x]) = \{f(x) \mid x \in [x]\} \tag{3.11}
$$

$I\boldsymbol{R}$ を実数上の有界閉区間の集合とする. 関数 $f : D \subset \boldsymbol{R} \to \boldsymbol{R}$ が初等関数の合成や四則演算に基づく演算規則で定義されているとき, f を $I\boldsymbol{R}$ 上の関数に拡張したい. しかし,$f([x])$

を厳密に計算することは非常な計算時間を要することがある. そこで,f の演算規則を単純に区間演算で置き換えた演算規則によって定義される関数を f の区間拡張といい $f_{\square}$ で表す.

$$f([x]) \subseteq f_{\square}([x]) \tag{3.12}$$

が成立する. 区間拡張は一意的に定まるわけではなく, f を定義する数式を同等な他の表現形式に書き直したとき, それらの表現に基づく区間拡張が一般に異なる評価を与えることに注意する.

上式で等号が成立するとき, 厳密な包み込みができたという. 厳密な包み込みは一般に難しいが,f の数学的な表現式に区間値パラメータが現れず, また, $[x]$ が一度しかその表現に現れなければ, その表現に基づく区間拡張が厳密な包み込みを与えることが知られている. 一般に式の表現に $[x]$ が少なく現れれば現れるほど, 区間包囲は厳密な包み込みに近くなることが多いことが経験されている. 区間 $[x]$ の幅 $d([x])$ が小さいとき,

$$q(f([x]), f_{\square}([x])) \leq \alpha d([x]) \tag{3.13}$$

となることが知られている. ただし, α は正の $[x]$ によらない定数である.

なお, 一般に関数の値域 $f([a,b])$ を $f([a,b]) \subset [c,d]$ と区間 $[c,d]$ で評価するとき, 区間 $[c,d]$ のことを $f([a,b])$ の区間包囲という. 区間拡張は区間包囲の一種である.

3.3.1 中心と半径による区間

区間演算におけるかけ算は, 場合分けも多く複雑である. これをシンプルに計算するために区間の中心と半径による表示をこの節で説明する. 点をかける区間の計算が簡単にできることを示す.α を実数とする. 区間を $[\beta-\delta, \beta+\delta]$ と表す.β は区間の中心で $\delta \geqq 0$ は半径である. このとき

$$\alpha[\beta-\delta, \beta+\delta] = [\alpha\beta - |\alpha|\delta, \alpha\beta + |\alpha|\delta] \tag{3.14}$$

が成立する. ただし, 四則演算は厳密にできるとする. 実際

$$\min\{\alpha\beta - \alpha\delta, \alpha\beta + \alpha\delta\} = \alpha\beta - |\alpha\beta| = \alpha\beta - |\alpha|\delta \tag{3.15}$$

$$\max\{\alpha\beta - \alpha\delta, \alpha\beta + \alpha\delta\} = \alpha\beta + |\alpha\beta| = \alpha\beta + |\alpha|\delta \tag{3.16}$$

$$\tag{3.17}$$

より式 (3.14) が示された.

第 4 章

MATLABにおける
スパース行列の反復解法

4.1 はじめに

本章では, 本論分で使用する MATLAB と大規模スパース行列に向いた解法である反復解法について説明する.

4.2 MATLAB

Matlab は, 行列計算およびグラフィックスに優れたソフトウェアで, 世界的に理工学系から経済分野まで広く使われている.Matlab はインタプリタ言語であって, ループ演算などは Fortran や C のコンパイラに比べると実行速度は遅い. ただし, 行列演算などについては高速な計算ができ, さらに容易にプログラムが組むことができるので, 安定性, 高速性に優れている.

4.3 スパース行列

行列の要素の中で非零なものが非常に少ない行列をスパース行列という. スパース行列は科学, 工学を問わずいろいろな分野で現れる. そこで, スパース性を生かした高速解法が発展してきた. したがって, スパース行列を扱う手法について学ぶことは応用上重要な意義がある. この行列の特性によって MATLAB ではつぎのようなことができる.

1. 行列の非ゼロ要素のみをインデックスと共に格納
2. ゼロ要素への演算を省略して計算時間を短縮

4.3.1 スパース行列ストレージ

フル行列の場合 MATLAB ではすべての行列要素を内部に格納する. ゼロ要素部分も, その他の行列要素部分と同じストレージ容量を必要とする. ただし, スパース行列の場合 MATLAB

は非ゼロ要素とそのインデックスのみを格納する. この方法の場合, ゼロ要素が含まれる割合の高い大きな行列に対してはデータストレージに必要なメモリ総量をかなり削減できる.

4.4 反復解法

反復解法とは, ある決められた一連の手続きを何度も繰り返すことによって解の精度を上げていく方法である. 反復解法の長所の一つは, 係数行列のスパース性を保持できることである. 連立一次方程式の解法というとガウスの消去法が有名であるが, これは数値計算の世界では直接解法と呼ばれ, 一般的に直接解法は大規模スパース系には向いていないと考えられている. 直接解法は数値計算的に非常に安定した解法であるが, 計算過程で係数行列のスパース性を著しく失わせる傾向があるため, まず, 計算容量（計算に必要なコンピュータメモリ量）の点で不利である.

更に, 直接解法の場合, 計算が完了するまで解の予想が困難であるため, 解に関する何らかの情報を得られるまでに必要な計算量は常に一定（問題サイズの 3 乗オーダ）であり, 得られる計算解の精度は高い傾向にあるが, ユーザが常にそこまでの精度を求めているかどうかは疑問で, せいぜい数桁正しい解が得られればよいという場合でも, やはり同じ計算量が必要である.

4.4.1 反復解法の弱点

数値計算における最大の弱点は, 数学的に解の収束が保証されていたとしても, 係数行列の性質が悪いと丸め誤差によって必ずしも計算解が真の解に収束するとは限らないことである. 対称正定値行列などの一部の特殊な行列を除いては, 決定的なアルゴリズムをつくることは困難とされている. したがって問題に応じてアルゴリズムを変える必要がある, というのが現状である.

4.5 MATLABで使用できる反復解法

連立一次方程式の反復解法は, ガウス・ザイデル法やSOR法（逐次過剰緩和法）を代表とする定常反復法系統と,CG法（共役勾配法）を代表とする非定常反復法系統の大きく二つに分類できる.MATLABには現時点（virsion7.0）では, 連立方程式のために9つの反復解法が用意されているが, それらはすべて非定常反復法である.
以下, 反復解法である,PCG法,Bi-CGSTAB法,GMRES法を説明する.

4.5.1 PCG法

PCG法は係数行列Aが対称正定値の場合に最も効果的な解法である. ただし, 与えられた係数行列が正定値かどうかを判定するのは困難であるため, あらかじめ係数行列が正定値になることが分かっている問題に有効である.

MATLABでは関数pcgが用意されており, 反復法の収束においては同値で条件の良い方程式に変換してから解くと速く収束することが多い. この操作を前処理という. 前処理行列としてAを近似する行列Mあるいは$M = M(1)M(2)$のような下三角行列$M(1)$と上三角行列$M(2)$を用意し,$Ax = b$の代わりに$M^{-1}Ax = M^{-1}b$を解く. このように条件の良い方程式に変換してから解く前処理付き共役勾配法をPCG法という.

4.5.2 Bi-CGSTAB法

Bi-CGSTAB法はランチョス原理に基づく反復解法で, 係数行列Aが非対称行列の場合によく使われる解法である.Bi-CG法系統の解法は, 後述のGMRES法系統の解法と違い, リスタートなどのパラメータ設定が基本的に不要であるため使い勝手が良く, ユーザに好まれる傾向がある.MATLABでは関数bicgstabが用意されており, 基本的な使い方は関数pcgと同様である.Bi-CGSTABの前処理法はいくつかあるが, 本論文では, 不完全LU分解を用いている.

4.5.3 GMRES 法

GMRES 法はアーノルディ原理に基づく反復解法で Bi-CGSTAB 法と同様に, 係数行列 A が非対称行列の場合によく使われる解法である. オリジナルの GMRES 法は計算量及び計算容量の点で実用的でなく, 適当な正整数 k に対して,k 回ごとにリスタートすることによって必要な記憶容量を減らした GMRES(k) 法が実際に用いられる.MATLAB では, 関数 gmres が用意されており, 基本的な使い方は関数 bicgstab と同じである. ただし, リスタートのためのパラメータ設定が必要である.

リスタートを行う GMRES(k) 法において,k の値は本来大きいほどオリジナルの GMRES 法に近づき収束も速くなる傾向にあるが, その分必要なメモリ量も増加する. これは解きたい問題に応じて対応すれば良い.

第 5 章

スパース行列に対する連立一次方程式の精度保証付き数値計算

5.1 はじめに

本章では,$Ax = b$ の連立一次方程式とスパース行列に対する連立一次方程式の解の精度保証を行う手法について述べる.

5.2 連立１次方程式の精度保証付き数値計算

A を $n \times n$ 行列,x,b を n ベクトルとする方程式

$$Ax = b \tag{5.1}$$

連立一次方程式の解の精度保証を行う手法を以下に上げる.

5.2.1 逆行列を用いる方法

$Ax = b$ の近似解があたえられたときに, その近くに真の解が存在するか否かの判定および, 真の解との誤差を求める方法を以下に述べていく. 以下, 次の定理が成り立つ.

方程式 $Ax = b$ の近似解 $\tilde{x}$ と A の逆行列の近似行列 R が求められたとき, 行列 $G = RA - I$ が不等式

$$||G|| < 1 \tag{5.2}$$

を満たすときは, 逆行列 A^{-1} が存在し

$$||A^{-1}|| \leqq \frac{||R||}{1 - ||RA - I||} \tag{5.3}$$

および $\tilde{x}$ を真の解として

$$||x^* - \tilde{x}|| \leqq \frac{||R(A\tilde{x} - b)||}{1 - ||RA - I||} \tag{5.4}$$

が成り立つ.

この定理によって, 近似解が求められたときその近くに真の解 $\tilde{x}$ が存在するか否かや, 真の

解と近似解との間の誤差を評価する.R と x は通常の浮動小数点演算により近似的に求めたとする.$r = Ax - b$ とおく.$G = RA - I$ と res の区間包囲を求めるためのプログラムは以下のようになる.

$setround(down)$ %下丸め計算

$\underline{G} = RA - I$

$\underline{res} = A\tilde{x} - b$

$setround(up)$ %上丸め計算

$\overline{G} = RA - I$

$\overline{res} = A\tilde{x} - b$

$res_{mid} = ((\underline{res} + \overline{res})/2)$ %中心と半径のかたちに変換

$res_{rad} = (res_{mid} - \underline{res})$

よって,

$$G = RA - I \in \{\underline{G}, \overline{G}\}$$

$$res_{mid} - res_{rad}\mathfrak{1} \leqq A\tilde{x} - b \leqq res_{mid} + res_{rad}$$

ここで上記の区間包囲が求められる. また, ここでノルムはすべて最大値ノルムとする.

$G = \max\{|\underline{G}|, |\overline{G}|\}$

$setround(down)$ %下丸め計算

$\underline{\triangle} = R * res_{mid}$

$setround(up)$ %上丸め計算

$\overline{\triangle} = R * res_{mid}$

$$\triangle = \max(|\underline{\triangle}|, |\overline{\triangle}|) + |R| * res_{rad} \qquad \%\text{Algorithm5.1}$$

$$G_{norm} = \max_{i=1,2,\cdots,n} \sum_{j=1}^{n} G_{ij}$$

$$R_{norm} = \max_{i=1,2,\cdots,n} \sum_{j=1}^{n} |R|_{ij}$$

$$\triangle_{norm} = \max_{i=1,2,\cdots,n} \{\triangle_i\}$$

$$D = -(G_{norm} - 1)$$

$$if \quad (D > 0)$$

$$err_{norm} = \frac{\triangle_{norm}}{D}$$

$$else$$

$$printf(\text{"}vrificationfailed\text{"})$$

Algorithm5.1 の補足

$$R * res_{mid} - |R| * res_{rad} \leqq R(A\tilde{x} - b) \leqq R * res_{mid} + |R| * res_{rad}$$

$$|R(A\tilde{x} - b)| \leqq \max(|\underline{\triangle}|, |\overline{\triangle}|) + |R| * res_{rad} \leqq \triangle.$$

ただし, 行列やベクトルの絶対値はその各要素の絶対値を要素とする行列やベクトルとする. 上記のプログラムにおいて精度保証が成功したとする, このとき上記の定理より $||A^{-1}|| \leqq A_{invnorm}, ||x^* - \tilde{x}|| \leqq ||err||$ が成立する.

以上のプログラムを実行するにあたり,4 回目の丸めの指定と $4n^3 + O(n^2)$ 回の浮動小数点演算で十分である. 逆行列を求めるのに $2n^3$ 回の浮動小数点演算が通常必要である. また, 単純に近似解を求めるだけならば, ガウスの消去法や LU 分解を用いると $\frac{2n^3}{3}$ の手間で求められるので, 逆行列を用いて近似解を導出し, その近似解を精度保証するのに $6n^3$ の計算量がかかることになる.

5.3 スパース行列に対する連立１次方程式の精度保証付き数値計算

A を $n \times n$ 行列,x,b を n ベクトルとする方程式

$$Ax = b \tag{5.5}$$

ここで,A が大規模スパース行列であるような連立１次方程式を解くことを考える.

スパース行列に対する精度保証付き数値計算の手法は基本的に (5.2) 節の密行列に対する手法と同じであるが, 大規模なスパース行列を扱う場合, 逆行列を直接求めることは望ましくない. なぜならば, 係数行列 A がスパースであっても, その逆行列がスパースになるとは限らず逆行列が密行列となってしまう結果, 莫大なメモリを必要とするためコストパフォーマンスが悪くなる. これではスパース性を生かすことができないのである.

そこで, 近似逆行列 R をベクトル単位で考える.$||R||_\infty$ は

$$||R||_\infty = \max_{||x||_\infty=1} ||Ax||_\infty = \max_{i=1,2,\cdots,n} \sum_{j=1}^{n} |R_{ij}| \tag{5.6}$$

であるから,R の行ベクトル単位を数値計算することで, メモリの消費を抑えることができる.R の i 行の行ベクトルを y_i とし, 単位行列 I の i 行目の行ベクトルを e_i とすると $RA = I$ より

$$y_i A = e_i \tag{5.7}$$

$$A^t y_i^t = e_i^t \tag{5.8}$$

以上より,R の i 行の行ベクトル y_i を求めることで近似逆行列を導き, メモリ消費を抑え, 大規模なスパース行列の精度を保証することができるのである.

y_i^t の求め方は前節で説明した反復解法の bicgstab 法を使用する. これは,

1. パラメータ設定が基本的に不要であるため使い勝手が良く, ポピュラーであること.

2. 係数行列 A が非対称行列でも適用可能であること.

を考慮したものである.
なお, 反復法の収束において同値で条件の良い方程式に変換してから解くため, 不完全 LU 分解を前処理として行う.

第 6 章

ソース

本章では，大規模スパース行列対する精度保証付き数値計算のソースを記載する．

6.1 function vlin_test.m

```
function err = vlin_test(A,b,func,tol1,tol2,loop)
system_dependent('setround',0);

if isequal(func,'gmres')
    funcin = '(P*A,P*b,10,tol1,loop,L,U)';
else
    funcin = '(P*A,P*b,tol1,loop,L,U)';
    restart = 0;
end

nnz_A = nnz(A)
tic;
% 不完全 LU 分解の条件を自動で変更
for ii=2:5
    switch ii
    case 1
        tolilu = '0'
    case 2
        tolilu = 1e-3
    case 3
        tolilu = 1e-6
    case 4
        tolilu = 1e-9
    case 5
        tolilu = 1e-12
    end
    % 不完全 LU 分解
    [L,U,P] = luinc(A,tolilu);
    % Ax=b を解く
    [x,flag,relres,iter_sol] = eval([func funcin]);
    if flag == 0
        nnz_L = nnz(L)
        nnz_U = nnz(U)
        break
    else
        ii, flag
    end
    clear L U
end
t_sol = toc
iter_sol

% 精度保証
disp('*** verification ***')
tic;
[err,iter_ver] = vlin_sparse(A,b,x,func,tol2,loop);
t_ver = toc
iter_ver
```

6.2 function vlin_sparse.m

```
function [err,iter_ver] = vlin_sparse(A,b,x,func,tol2,loop)
%  正方行列でなければ終了
[m,n] = size(A);
if m ~= n
    disp('rectangular matrix')
    return
end

if isequal(func,'gmres')
    funcin = '(B,P*e,10,tol2,loop,L,U)';
else
    funcin = '(B,P*e,tol2,loop,L,U)';
end

% 不完全LU分解
[L,U,P] = luinc(A',1e-3);
B = P*A';

system_dependent('setround',-Inf);
rl = A*x - b;
system_dependent('setround',Inf);
ru = A*x - b;
rmid = 0.5*(rl + ru);
rrad = rmid - rl;


G_norm = 0;
Rr_norm = 0;

e = zeros(n,1);
iter_ver = 0;
ii = 2;
for i=1:n

    system_dependent('setround',0);

    e(i) = 1;
    % A'y=eを解く
    [v,flag,relres,iter] = eval([func funcin]);
    iter_ver = iter_ver + iter;
    % 不完全LU分解の条件を自動で変更
    if flag ~= 0
        flag
        ii = ii + 1;
        switch ii
        case 2
            tolilu = 1e-3
        case 3
            tolilu = 1e-6
        case 4
            tolilu = 1e-9
        case 5
            tolilu = 1e-12
```

```
        otherwise
            err = Inf;
            error('impossible!')
        end
        clear L U P B
        [L,U,P] = luinc(A',tolilu);
        nnz_L = nnz(L)
        nnz_U = nnz(U)
        B = P*A';
        i = i - 1;
    else
        system_dependent('setround',-Inf);
        tg_l = v'*A - e';
        yl = v'*rmid;

        system_dependent('setround',Inf);
        tg_u = v'*A - e';
        tg = max(abs(tg_l), abs(tg_u));
        norm_tg = sum(tg);
        yu = v'*rmid;
        yu = max(abs(yl),abs(yu));
        norm_rr = yu + abs(v')*rrad;
        if norm_tg >=1
            i, norm_tg
            err = Inf;
            error('norm_tg is larger than 1.')
        end
        G_norm = max(G_norm, norm_tg);
        Rr_norm = max(Rr_norm, norm_rr);

        e(i) = 0;
        i
    end
end

if G_norm < 1
    d = -(G_norm - 1);
    err = Rr_norm/d;
else
        err = Inf;
end

system_dependent('setround',0);
```

第 7 章

実験結果

7.1 はじめに

本章では,6 章のソースを使用した大規模スパース行列における実験結果について述べる.

7.2 実行環境

1. CPU : Intel(R) Xeon(TM) 3.06GHz
2. Memory : 1 GB RAM
3. OS : Windows XP
4. MATLAB version : 7.0

7.3 使用するスパース行列

本論文では, 実験行列として Matrix Market にあるスパース行列を使用する. 以下の実験行列は全て,Matrix Market の SPARSKIT Collection に格納されているデータである.

表 7.1: 実験に使用した DRIVCAV セットに用意されている行列

行列	ファイル名	行列サイズ	非零要素数
e20r0000	e20r0000.mtx.gz	4241	131412
e20r1000	e20r1000.mtx.gz	4241	131430
e30r0000	e30r0000.mtx.gz	9661	305794
e30r1000	e30r1000.mtx.gz	9661	306002
e40r0000	e40r0000.mtx.gz	17281	553216
e40r1000	e40r1000.mtx.gz	17281	553562

表 7.2: 実験に使用した DRIVCAV OLD のセットに用意されている行列

行列	ファイル名	行列サイズ	非零要素数
CAVITY16	CAVITY16.mtx.gz	4562	137887
CAVITY17	CAVITY17.mtx.gz	4562	131735
CAVITY18	CAVITY18.mtx.gz	4562	138040
CAVITY19	CAVITY19.mtx.gz	4562	131735
CAVITY20	CAVITY20.mtx.gz	4562	131735
CAVITY21	CAVITY21.mtx.gz	4562	131735
CAVITY22	CAVITY22.mtx.gz	4562	138040
CAVITY23	CAVITY23.mtx.gz	4562	131735
CAVITY24	CAVITY24.mtx.gz	4562	138040
CAVITY25	CAVITY25.mtx.gz	4562	131735
CAVITY26	CAVITY26.mtx.gz	4562	138040

表 7.3: 実験に使用した FIDAP のセットに用意されている行列

行列	ファイル名	行列サイズ	非零要素数
FIDAP015	FIDAP015.mtx.gz	6867	96421
FIDAP018	FIDAP018.mtx.gz	5773	69231
FIDAP019	FIDAP019.mtx.gz	12005	259561
FIDAP031	FIDAP031.mtx.gz	3909	91165
FIDAP035	FIDAP035.mtx.gz	19716	217972
FIDAP037	FIDAP037.mtx.gz	3565	67591
FIDAPM08	FIDAPM08.mtx.gz	3876	86793
FIDAPM09	FIDAPM09.mtx.gz	4683	93733
FIDAPM11	FIDAPM11.mtx.gz	22294	617874
FIDAPM29	FIDAPM29.mtx.gz	13668	183394
FIDAPM37	FIDAPM37.mtx.gz	9152	765944

表 7.4: 実験に使用した TOKMAK のセットに用意されている行列

行列	ファイル名	行列サイズ	非零要素数
UTM3060	UTM3060.mtx.gz	3060	42211
UTM5940	UTM5940.mtx.gz	5940	83842

7.3.1 実行手順

上に述べた行列を使用するときの手順を示す. 連立一次方程式 $Ax = b$ における右辺ベクトル b はすべての要素が 1 のベクトルと A の積とする. また, 前処理として不完全 LU 分解を用いて, その fill-in の基準となるしきい値を 10^{-3} から 10^{-12} とする. 反復方法は bi-cgstab 法を利用し, 相対残差を vlin_test.m では 10^{-12},vlin_sparse.m では 10−3, 最大反復回数を 1000 回とした.

```
>>load matrixdata.mtx
>>A = spconvert(matrixdata);
>>n = length(A),b = A * ones(n,1);
```

これにより得られた数値解に精度保証をかける.

```
error = vlin_test(A,b,'bicgstab',1e-12,1e-3,1000)
```

として行う.vlin_test.m,vlin_sparse.m の中身については第 5 章に記載してあるので, そちらを参照していただきたい.

7.3.2 結果

大規模スパース行列に対する精度保証付き数値計算を行うプログラム (vlin_test.m) の実行結果を以下に示す.

表において err は

$$\frac{||R(A\tilde{x} - b)||_\infty}{1 - ||RA - I||_\infty}$$

をあらわす. また, 次元は行列名に示した行列の次元数を,tolilu は不完全 LU 分解の fill-in の基準となるしきい値を,nnz_L,nnz_U は不完全 LU 分解のそれぞれの非零要素数を,times(s) は実行時間を示す.

表 7.5: DRIVCAV にセットされている行列の実験結果

行列名	次元	tolilu	nnz_L	nnz_U	error	time(s)
e20r0000	4241	1.00e-03	646885	902660	1.74E-08	2441.7
e20r1000	4241	1.00E-06	853176	986803	8.97E-09	685.27
e30r0000	9661	1.00E-06	3068851	3209257	5.98E-09	5041.8
e30r1000	9661	1.00E-06	3068851	3416928	5.20E-09	5085.2
e40r0000	17281	1.00E-06	7310358	7750614	8.21E-10	22152
e40r1000	17281	1.00E-06	6909118	8143949	2.94E-08	22675

表 7.6: DRIVCAV OLD にセットされている行列の実験結果

行列名	次元	tolilu	nnz_L	nnz_U	error	time(s)
CAVITY16	4562	1.00E-06	954590	1046092	1.53E-08	747.06
CAVITY17	4562	1.00E-03	687110	926318	7.98E-09	3208.0
CAVITY18	4562	1.00E-06	948637	1046760	8.96E-08	747.00
CAVITY19	4562	1.00E-03	654852	942218	6.91E-09	4426.7
CAVITY20	4562	1.00E-06	941387	1061353	5.91E-08	751.58
CAVITY21	4562	1.00E-06	646484	921030	9.98E-08	7428.9
CAVITY22	4562	1.00E-06	930857	1075108	1.71E-07	749.78
CAVITY23	4562	1.00E-06	877799	1004653	1.25E-07	778.48
CAVITY24	4562	1.00E-06	926996	1083525	2.57E-07	751.56
CAVITY25	4562	1.00E-06	875442	991848	2.22E-07	846.58
CAVITY26	4562	1.00E-06	918345	1068615	4.04E-10	2579.5

表 7.7: FIDAP にセットされている行列の実験結果

行列名	次元	tolilu	nnz_L	nnz_U	error	time(s)
FIDAP015	6867	1.00E-09	193593	367392	3.37E-04	747.88
FIDAP018	5773	1.00E-09	209548	292409	1.30E-03	946.52
FIDAP019	12005	1.00E-12	523293	486137	7.36E-05	1610.1
FIDAP031	3909	1.00E-06	210115	248711	1.02E-09	221.50
FIDAP035	19716	1.00E-09	607184	662599	6.83E-04	5200.5
FIDAP037	3565	1.00E-03	37396	38695	3.25E-13	97.531
FIDAPM08	3876	1.00E-06	397410	525357	2.02E-10	365.28
FIDAPM09	4683	1.00E-06	232706	431041	1.69E-11	362.38
FIDAPM11	22294	1.00E-06	10517500	11635415	1.50E-08	11341
FIDAPM29	13668	1.00E-06	697698	868566	1.76E-11	2567.0
FIDAPM37	9152	1.00E-09	1860738	1907389	4.95E-08	3954.8

表 7.8: TOKMAK にセットされている行列の実験結果

行列名	次元	tolilu	nnz_L	nnz_U	error	time(s)
UTM3060	3060	1.00E-06	373282	416872	4.79E-08	255.34
UTM5940	5940	1.00E-09	1512912	1574201	1.46E-06	1412.0

第 8 章

統括

8.1 はじめに

本章では，本論文の統括について述べる．

8.2 実行結果について

第 7 章に記載した大規模スパース行列に対する精度保証付き数値計算の実験結果から, 実行時間, 実験対象であるスパース行列の次元数, 実験で使用した反復解法について以下に記す.

8.2.1 実行時間

本論文で使用したプログラム (vlin_test.m,vlin_sparse.m) では, 反復解法を何度も使用するため実行時間がかかってしまう. 実験の対象となるスパース行列のサイズや非零要素数, その密度によって実行時間は左右されるので, それに応じて反復解法や前処理の条件を変える必要性がある.

実行時間を短くするには, 不完全 LU 分解の fill-in の基準となるしきい値を小さくし, 完全な LU 分解に近づければよいということがわかった. だが, その代償として L と U の非零要素が爆発的に増えるため, メモリを大量に消費してしまう. メモリが最大パフォーマンスを超えてしまえば, 実験は強制終了せざるを得なくなってしまうため, 本論文ではメモリを最重要に考えた.

8.2.2 次元数

前章の実験結果のとおり,22000×22000 の行列まで精度保証することができた. プログラムを改良する度に, 精度保証が可能となる次元数が増え, 精度保証においても, より高く精度

を保証できるようになったことから, 生成したプログラムの有用性を活かせることができたと思う.

8.2.3 反復解法

数ある反復解法の中でも,bicg 法,gmres 法,bicgstab 法の 3 種類の解法を取り上げたが, 最終的には bicgstab 法を用いて実験結果を得ることにした. それは, 大規模スパース行列を解くプロセスにおいて, 繰り返し行った実験の結果から bicgstab 法が特に実行時間が短く, 結果が安定して算出されるということがわかったからである.

8.3 まとめ

本論文は, スパース行列の特性を活かした大規模行列における連立一次方程式の解の精度保証について研究を行ってきた. 中でも, これまで不可能であった 10000×10000 以上のサイズの行列を数値計算と共に精度保証することが本論文の目的であった. 結果として, 本論文の目的を上回る 20000×20000 以上のサイズの行列の精度を保証するに至ったが, さらにサイズの大きい行列に対しても, 精度保証をかけることが可能であると思う.

さらに大規模なスパース行列を精度保証付き数値計算するためには, プログラムの工夫だけでなく, 実行環境の工夫が必要である. 特に, 最も時間がかかっている $A^t y_i^t = e_i^t$ の反復解法における処理の部分は独立しているため, 複数の CPU で並列計算をするという工夫が効果的であると考える.

以上をもって, 本論文の統括とする.

謝辞

本研究を進めるに当たり, たくさんの方々から多くの助言や指導を頂きました. まず, 本研究を進めるに当たり, 終始丁寧な御指導を賜り, 最後まで御面倒を見て下さいました, 大石 進一教授に深く感謝いたします.

大変お忙しい中でも私の質問に対して熱心に答えて下さいました, 理工学研究科客員講師, 荻田 武史氏, 丸山 晃佐氏に大変感謝いたします.

研究室内や合宿など, 日常的な場面でお世話になりました, 大石研究室博士課程 1, 尾崎 克久氏, 大石研究室修士課程 2 年, 大山 博毅氏, 森山 敦史氏, 並びに大石研究室修士課程 1 年, 坂内 太郎氏, 島本 誠氏, 関本 竜平氏, 徳永 克久氏, 福地 健氏, 細田 幸裕氏, 横山 大輔氏に深く感謝いたします.

研究内容だけでなくいろいろな場面で意見の交換や協力などをして下さいました, 大石研究室 4 年, 有滝 貴広氏, 岩田 揚平氏, 大成 高顕氏, 川崎 文裕氏, 栗原 崇氏, 佐藤 友規氏, 平野 晃氏, ト 詣各氏, 吉岡 毅氏, 吉田 昂央氏には様々な助言をもらい大変感謝しています.

参考文献

1. MATLAB HOME PAGE

   http://www.cybernet.co.jp/matlab/

2. Matrix Market HOME PAGE

   http://math.nist.gov/MatrixMarket/

3. 大石進一:“応用数学セミナー 数値計算”, 裳華房,(1999).

4. 大石進一:“Linux 数値計算ツール”, コロナ社,(2000).

5. 大石進一:“精度保証付き数値計算”, コロナ社,(2000).

6. 大石進一:“MATLAB による数値計算”, 培風館,(2000).

7. Shin'Ichi Oishi,Siegfried M.Rump:“Fast verification of solutions of matrix equations”, Numerische Mathematik,vol 90(2002),pp.755–773.

8. 山下栄吉:“マイクロ波シュミレータの基礎”, 電子情報通信学会,(2004).