数理解析研究所講究録
1038巻 1998年 177-182

浮動小数演算に基づく安定化理論計算システムの作成

水口 寛之 (Hiroyuki Minakuchi)、甲斐 博 (Hiroshi Kai)、
野田 松太郎 (Matu-Tarow Noda)
愛媛大学工学部

1. はじめに

数式処理で用いるアルゴリズムの多くは、その厳密性を中心に構成されている。しかし、その半面では代数演算等を除いた多くの課題へのアルゴリズムの適用の困難さを持ち、かつ計算途中で膨大な記憶容量を必要とし、計算時間の増大をもたらしている。これらを克服し、数式処理のアルゴリズムの厳密性をいかしつつ、実用に供するアルゴリズム開発の視点から、数値数式融合算法が提唱され、国内外の多くの研究者によって、その発展がはかられている。しかし、代数的に確立されたアルゴリズムに単に浮動小数計算を適用するのみの計算法においては、本来のアルゴリズムの持つ厳密性を放棄することになり、あまり意味を持たないと言っても過言ではない。アルゴリズムによっては近似精度をいくら上げても真の解に近付かないものが存在する。これらは不安定なアルゴリズムと呼ばれる。このような不安定なアルゴリズムに対して、近似精度を上げていけば確実に真の解に近付くことを保証するのが、アルゴリズムの安定化理論である [2]。我々はすでに安定化理論を計算を容易に行うための計算システムを、数式処理システム Risa/Asir の上に構築し、一般逆行列計算の安定化を行った [1]。しかし、このシステムでは数値計算部分に PARI を用いていることや、インタプリタ方式で計算を行っているため、計算速度の面では数式処理による計算に比較してあまり向上を見ていない。そこで、本研究では数値計算部分を IEEE 方式の浮動小数を用いて C 言語で実行するように変更し、Risa/Asir と結合することにより安定化理論を実現するシステムを作成し、Moore–Penrose 型一般逆行列を例として、数式処理による算法等と主に計算速度の面での比較を行うことを目的とした。

2. 一般逆行列のアルゴリズムと安定化理論

2.1. 一般逆行列の定義

まず、本稿の記述の一貫性を保つために、一般逆行列に対して知られているいくつかの性質や事項をまとめる。

A を $m \times n$ 行列、G を $n \times m$ 行列とする。次の 4 つの方程式、

(i) $AGA = A$

(ii) $GAG = G$

(iii) $(AG)^T = AG$

(iv) $(GA)^T = GA$

において、Gが (i) をみたすとき、GはAの一般逆行列 (A^-) とよばれる。そして、Gが (i)～(iv) をみたすとき、GはAの Moore–Penrose 型一般逆行列 (A^+) とよばれる。本研究では単に一般逆行列といった場合、Moore–Penrose 型一般逆行列を表すものとする。

2.2. 一般逆行列の計算アルゴリズム

文献 [3],[4] 等に示されているように、一般逆行列を求めるためには多くのアルゴリズムが知られている。この大半は数式処理で一般逆行列を求めることを主に論じられているが、一般にはこれら以外に数値計算で一般逆行列を計算する場合が多い。数値計算のアルゴリズムとしては、特異値分解法を用いるのが一般である。しかし、悪条件問題 (一般逆行列を必要とする問題の多くは悪条件であるが) に対しては、微小な特異値の処理等で特異値分解法による数値計算には常に不安がつきまとう。そこで、本研究では数式処理の算法の中、計算速度、メモリ効率が比較的優れ、かつプログラムが容易な Greville のアルゴリズムを用いる。

Greville のアルゴリズムとは以下のようなものである。

入力行列を $A = [a_1^T, a_2^T, \cdots, a_n^T]^T$

$a_i \cdots m$ 次元行ベクトル

とする。A_i を、

$$A_1 = a_1 \qquad A_i = \begin{bmatrix} A_{i-1} \\ a_i \end{bmatrix}$$

と定義し、$i = 1, 2, \cdots, m$ において、

$$A_i^+ = [A_{i-1}^+ - b_i^T d_i \quad b_i^T]$$

を順次計算することにより、最終的に A_m^+ が A^+ となる。

なお、d_i, c_i, b_i および初項 A_1^+ は、

$$d_i = a_i A_{i-1}^+$$

$$c_i = a_i - d_i A_{i-1}$$

$$b_i = \begin{cases} \frac{c_i}{c_i c_i^T} & (c_i \neq 0) \\ \frac{d_i (A_{i-1}^+)^T}{1 + d_i d_i^T} & (c_i = 0) \end{cases}$$

$$A_1^+ = \begin{cases} \frac{a_1^T}{a_1 a_1^T} & (a_1 \neq 0) \\ a_1^T & (a_1 = 0) \end{cases}$$

として計算する。アルゴリズム実現上の問題点は、行列の大きさを変化させていく必要のあることと、b_i, A_i^+ の計算に必要な結果の零判定である。

2.3. Greville のアルゴリズムの安定化

次に Greville のアルゴリズムを安定化する方法について述べる。

安定化理論では入力の形に関わらずデータの形が一定なアルゴリズムに関して適用が可能である。Greville のアルゴリズムではゼロ判定が不安定性の原因になっているが、安定化理論ではこの部分を修正し、安定な計算ができるようにする。

Greville のアルゴリズムの安定化は以下のようにして行なった。

まず、入力行列 A の各要素 A_{ij} を a を浮動小数点表現による近似値、α を誤差の上限 1.0×10^{-k}(k は有効桁) として、bracket coefficient、

$$A_{ij} = [a_{ij}, \alpha_{ij}]$$

であらわす。各演算は、

$$\begin{aligned}
&\textbf{加算} : [a,\alpha] + [b,\beta] = [a+b, \alpha+\beta] \\
&\textbf{減算} : [a,\alpha] - [b,\beta] = [a-b, \alpha+\beta] \\
&\textbf{乗算} : [a,\alpha] \times [b,\beta] = [ab, (|a| \times \beta) + (\alpha \times \beta) + (\alpha \times |b|)] \\
&\textbf{除算} : [a,\alpha]^{-1} = \left[\frac{a}{(a+\alpha)(a-\alpha)}, \frac{|\alpha|}{(a+\alpha)(a-\alpha)} \right]
\end{aligned}$$

と計算する。b_i と A_1^+ の計算におけるゼロ判定には Zero Rewriting を用いる。

このようにして計算した A_n^+ の各要素を $[g_{ij}, \gamma_{ij}]$ とし、出力の各要素を G_{ij} とする。各 ij について、

$$G_{ij} = g_{ij}$$

と置き換える。この $m \times n$ 行列 G が出力となる。

3. C による浮動小数計算を用いた安定化理論

上で述べた Greville のアルゴリズムの安定化を実際の計算例に即して述べる。すでに、文献 [1] では、数式処理システム Risa/Asir と組み込まれた多桁整数計算パッケージ PARI を使用した高精度小数計算により、Greville のアルゴリズムの安定化については検討しており、その有用性を確認している。しかし、数式処理システムに組み込まれたり、それと結合して使用されている PARI のような多桁整数パッケージを基本とする「浮動小数計算」では、数式処理システムが本来有しているメモリ使用の多さや計算の低速性といった問題点はなんら解決されない。そこで、ここでは浮動小数計算として一般に広く用いられている IEEE 方式による計算法を用い、プログラムを C 言語で作成し、それを Risa/Asir と結合することにより安定化理論をより高速に実現できる可能性について検討する。Greville のアルゴリズムに関して、検討すべき計算法は以下の通りである。

1. Greville のアルゴリズムを数式処理で実行
2. Greville のアルゴリズムを Risa/Asir 上の PARI を用いた浮動小数による安定化理論で実行
3. Greville のアルゴリズムを完全に数値計算に置き換えて実行

4. Greville のアルゴリズムのゼロ判定部分にしきい値を導入して数値計算を実行
5. Greville のアルゴリズムを IEEE 方式の浮動小数による安定化理論で実行

なお、上でいう数値計算プログラムは C 言語で作成し、IEEE 方式によって計算している。また、計算に用いたコンピュータは Pentium 133MHz のものである。安定化理論では浮動小数を C 言語の float,double float で計算精度を上げながら計算しているが、その他の数値計算は float によっている。以上の計算を実際に一般逆行列計算を必要とする 50×25 行列に対して実行した。以下に述べる通常の浮動小数計算による場合のしきい値との関連をみるため、行列の要素は

1. ランダムに 0 から 2^{16} までの正整数を発生させる
2. ランダムに 0 ～ 1 区間に浮動小数を発生させる

の 2 通りで作成した。

3.1. 数値的な Greville のアルゴリズムにしきい値が与える効果

Grevill のアルゴリズムは本来は数値的に一般逆行列を求めるために開発されたものだが、ゼロ判定の必要性によって数値計算をそのまま適用することは困難である。その意味で数式処理のアルゴリズムとして用いると、より安全に結果を与えることになる。しかし、ゼロ判定を行うべき要素に対して、しきい値を設定し、計算値がしきい値以下になれば、その要素をゼロとみなすことによって計算を行うことによって比較的高速に数値的に一般逆行列を得ることができる。もちろん、安定化理論を用いるわけではないので、結果に対する保証はない。結果が正しいか否かは数式処理によるものと比較して確かめなければならない。大きさ 50×25 で 0 から 2^{16} までのランダムな整数要素を持つ行列および、その最大要素を 1 として要素の大きさを 0 から 1 までのランダムな小数にした場合対する一般逆行列を Greville のアルゴリズムで計算する。この計算結果を一般逆行列の満たす上述の条件式に代入することによって、結果の正確さを求めたものを表 1 に示す。ここで、○は結果が正確な場合を、×は不正確な場合を示している。

大きな整数要素		0 ～ 1 の小数要素	
しきい値	結果の正しさ	しきい値	結果の正しさ
$\sim 10^2$	×	$\sim 10^{-13}$	×
$10^3 \sim 10^8$	○	$10^{-12} \sim 10^{-7}$	○
$10^9 \sim$	×	$10^{-6} \sim$	×

表 1 数値計算におけるしきい値とその結果

用いたデータ型が float であることを考慮すると、しきい値はゆるされる範囲内で非常に小さく設定するのが当然であるが、ここで用いた例のように大きな桁の整数までを要素にもつ場合をそのまま扱おうとすると、適切なしきい値の設定は非常に困難になる。もちろん、行列の要素を 0 ～ 1 内の値に変換させると、通常考える意味でのしきい値の大きさの範囲になる。しかし、この場合も小さく設定しすぎると再び結果が得にくいという事実を示している。当然ながら、特に悪条件性を持つ行列の逆行列を求めるような場合に、行列要素の値は大きく変化していることが一般であるので、事前に最適なしきい値を設定して

数値計算によって一般逆行列を求めることの困難さが上の例を通してわかる。もちろん、しきい値 = 0(しきい値を設定しない場合) での Greville のアルゴリズムを数値計算した場合の結果は一般に不正確であるといえる。このように一般逆行列を数値的に求めようとする場合には、しきい値の設定には多くの困難がある。

3.2. Cによる浮動小数演算で安定化理論を実行した場合

次に、上で扱ったランダムな整数要素を持つ50×25行列に対して、Greville のアルゴリズムを各種の計算方法で実行した場合の正確さと計算時間について表2にまとめる。Risa/Asir を用いた正確な数式処理計算では行列の各要素は数式として扱われる。一方、Risa/Asir による数値計算では行列の各要素はそれぞれ PARI による浮動小数点数、C による計算では IEEE 形式での浮動小数点数として扱われる。結果の正確さは再び、正確な場合に○、不正確な場合に×として表示した。安定化理論を用いる場合に、PARI による計算では、計算桁数を順次大きくすることによって、正確な一般逆行列が得られるまでの計算時間の合計を示しているが、以下の表の場合は桁数を 9,20,40,60,80 と変化させている。また、しきい値の設定は表1の結果をみても2種の行列に対しては同じものを採用できないので、大きな整数要素をもつ行列に対しては、しきい値 $= 1.0 \times 10^5$、0 ～ 1 の要素を持つものに対しては、しきい値 $= 1.0 \times 10^{-10}$ とした。

一方、C 言語で安定化理論を実現するためには、数値計算で用いるデータ型が基本になる。IEEE 方式であるので、float、double は各々10 進数の桁数では 7,15 桁の精度に対応している。より多倍長浮動小数データを用いて安定化理論を実現する必要があるが、現段階ではいまだ実現していない。

計算の種類	大きな整数要素		0 ～ 1 の小数要素	
	正確さ	計算時間 (sec)	正確さ	計算時間 (sec)
Risa/Asir による数式処理	○	2193	○	6.981×10^5
PARI を用いた安定化理論	○	360.7	○	276.6
しきい値を設定した数値計算	○	9.87	○	11.2
Cで計算した安定化理論				
データ型 float	×	–	×	–
データ型 double	○	0.58	○	0.71

表2 Risa/Asir と C での結果の比較

表2より明らかなように、

- 数式処理のみを用いて、一般逆行列を計算しようとすると、結果を得るのに膨大な CPU 時間を要したり、それでもメモリの制約等で結果を得れない場合もあり得策ではない。
- 単純に数値計算 (今の場合は Greville の方法) を適用しようとすると、行列の要素の大きさによって、しきい値の設定に苦慮する必要があり、不適切なしきい値では正しい結果は期待できない。

- 以上の議論を通じて、安定化理論で計算を行うと比較的高速にかつしきい値の選択への配慮の困難もないことが明らかになる。
- 同じ安定化理論を実現する場合でも、数式処理システムに付加えられた多倍整数演算を基礎とする演算体系(Risa/Asir に対してのPARI)を用いて実現した場合と比較すると、C言語で実現する方がはるかに高速である。

ことがわかる。これは、その他の多くの実験例を通じて変わることはない。

4. むすび

以上のように、数式処理を用いて計算すると正確な計算は可能だが、問題によっては計算時間が非常に大きくなることがあり、また、しきい値を用いない数値計算では計算結果が信頼できないことがわかる。しかし、しきい値を用いた数値計算では正確な計算が可能である。一方、安定化理論を用いた計算では区間演算が必要なため通常の数値計算よりは時間がかかるものの、それでも十分高速な計算ができる。今回の実験では簡単な例のため、通常の数値計算でもしきい値を用いることにより正確な計算が可能であるが、実際の問題では最適なしきい値の大きさは問題によって大きく変わり、システマティックに決定するのが難しいという問題がある。一方、安定化理論の場合は精度をあげることにより解は正確な値に収束することを保証している。

安定化理論を実現するためのシステム作成としては、基本の数式処理システムとしてRisa/Asir を採用した場合には、PARIによる多桁計算を行うより、C言語で数値計算を行い、数式処理システムと結合する方がはるかの高速であることが実例を通して確認することができた。もちろん、一般逆行列を求めるような場合に数値計算を単純に行うとゼロ判定等で困難が起こり、しきい値を設定しての処理を行わなければならない。しかし、しきい値の設定方法には問題によって困難がある。適切な大きさのしきい値を設定しなければ安心して結果を得ることができない。それらに対して、数式処理と数値計算との結合によって安定化理論を実現すると、高速にかつ高信頼度の結果を得ることができることがわかった。今後のシステム拡充のために残された課題は、C言語で計算することができる多倍精度の浮動小数システムを容易に扱えるようにして、その上で安定化理論を実現することである。

参 考 文 献

[1] H. Minakuchi,H. Kai,K. Shirayanagi and M.T. Noda, Algorithm stabilization techniques and their application to symbolic computation of generalized inverses, Proc. IMACS–ACA'97, 1997

[2] K. Shirayanagi and M. Sweedler, 1995

[3] 野田松太郎、泉田正則、越智正明: 一般逆行列の数式処理システムによる直接解法とその評価、情報処理学会論文誌 Vol.30,no.11、pp.1376–1384 (1989).

[4] M.T. Noda, T. Saito and K. Makino, ,Algebraic Methods for Computing A Generalized Invers, SIGSAM Bulletin , pp.51–52(1997)