Algorithms of Generalized Inverses and their stabilization

Hiroyuki Minakuchi, Hiroshi Kai and Matu–Tarow Noda
Department of Computer Science, Ehime University, Japan

Abstract

An inverse of a matrix which is not necessarily square or is square, but nevertheless singular, is applied to solve ill–conditioned problems such as large sized matrix computations. Such an inverse is referred to as a generalized inverse. Generalized inverses have many applications in engineering problems, such as data analysis, electrical networks, character recognition, and so on. The most frequently used one is a Moore-Penrose type inverse. Several algorithms to compute generalized inverses have been proposed.

We designed a computer software program for obtaining the Moore-Penrose type generalized inverse from the viewpoint of an algorithm stabilization technique proposed by Shirayanagi and Sweedler. The system can compute a generalized inverse of a matrix which permits the use of limited precision computation in a convergent fashion. We already computed the generalized inverse by stabilizing Greville's algorithm on a computer algebra system referred to as Risa/Asir. In this paper, a similar system is designed by a floating-point computation in C. The results of high precision floating computations using the Risa/Asir system and an algorithm stabilization method are compared. The results show an advantage of using our software, especially in computation time. This does not mean that Greville's algorithm may be directly computed with limited precision computation in a convergent fashion, but the algorithm is rather amenable to the stabilization techniques to the algorithm. The results obtained by the system are highly reliable and also show some advantages obtained by similar systems especially in computation time. Through some examples, we show the effectiveness of the stabilization techniques.

1 Introduction

An inverse of a matrix – which is not necessarily square or is square, but nevertheless singular – is applied to solve ill–conditioned problems such as large sized matrix computations. Symbolic computation of the generalized inverse is one of the most interesting application areas of Computer Algebra. Algorithms to obtain generalized inverses have been proposed and compared from the viewpoint of complexity bounds. [5, 4]. Greville's algorithm is one of the best methods for obtaining a symbolic generalized inverse [2].

Greville's algorithm is computed both numerically and symbolically. The algorithm is computed by a numeric computation rapidly but gives instable results. On the other hand, the symbolic computation gives exact results but takes a long time for computation and uses a lot of memory. Therefore, we consider the method of stabilizing an algorithm proposed by Shirayanagi and Sweedler [8]. We already showed the effectiveness of the algorithm stabilization method implemented in a computer algebra system referred to as Risa/Asir [3, 9]. Computations done in [3] were based on this computer algebra system and have the similar defects of the symbolic computation. Thus, in this paper, we compute the generalized inverse by the algorithm stabilization method implemented in an environment of C++ based numeric computation. We can obtain accurate results quickly. For computing long digit floating point, we use a numerical computation package, NTL [10], written in C++.

In the following sections, we describe briefly the generalized inverse of matrices (in **2**) and the algorithm stabilization technique (in **3**) briefly. Finally, Greville's algorithm is stabilized and effectively computed by long digit floating point computations.

2 Algebraic Algorithms of Computing Generalized Inverse

The generalized inverse G of a matrix A is defined as follows:

$$\left\{\begin{array}{lcl} AGA & = & A \\ GAG & = & G \\ (AG)^T & = & AG \\ (GA)^T & = & GA \end{array}\right. \qquad (1)$$

where A and G are $m \times n$ and $n \times m$ matrices with entries in $\mathbf{R}$, respectively. If a matrix G satisfies all four relations, it is called the *Moore-Penrose type* generalized inverse and is denoted as A^+. The generalized inverse A^+ also satisfies the following relations:

1. A^+ is uniquely determined for A

2. if A is regular, $A^{+} = A^{-1}$
3. if $A = 0$, $A^{+} = 0$
4. if A is an $m \times n$ matrix with rank m, then $A^{+} = A^{T}(AA^{T})^{-1}$

We call a Moore–Penrose type generalized inverse simply a 'generalized inverse'. The generalized inverse is usually computed numerically by a SVD (Singular Value Decomposition) algorithm. However, we consider, in this paper, mainly symbolic methods and the algorithm stabilization method to obtain the generalized inverse. Symbolic methods for computing a generalized inverse have been proposed by many authors. Comparisons of symbolic methods were done by Noda and others [5, 4] from the viewpoint of the number of operations needed. They conclude that Greville's algorithm [2] is one of the best algorithms. Thus , in this paper, we compute the generalized inverse by Greville's algorithm. For an input matrix A, Greville's algorithm is described as follows:

1. Decompose input $n \times m$ matrix A into row vectors a_i
$$A = (a_1^T, a_2^T, \cdots, a_n^T)^T$$
2. Let $i \times n$ matrices A_i be
$$A_1 = a_1, \qquad A_i = \begin{pmatrix} A_{i-1} \\ a_i \end{pmatrix}$$
3. For $i = 1, 2, \cdots$, compute $n \times i$ matrices A_i^{+} as
$$A_i^{+} = (A_{i-1}^{+} - b_i^T d_i \mid b_i^T)$$
where
$$\begin{aligned} d_i &= a_i A_{i-1}^{+} \\ c_i &= a_i - d_i A_{i-1} \\ b_i &= \begin{cases} \frac{c_i}{c_i c_i^T} & (c_i \neq 0) \\ \frac{d_i (A_{i-1}^{+})^T}{1 + d_i d_i^T} & (c_i = 0) \end{cases} \\ A_1^{+} &= \begin{cases} \frac{a_1^T}{a_1 a_1^T} & (a_1 \neq 0) \\ a_1^T & (a_1 = 0) \end{cases} \end{aligned}$$
4. After m repeatitions, A_m^{+} gives the Moore–Penrose generalized inverse of A^{+}.

Note that it is necessary to decide precisely whether a value is zero or not in the process of obtaining it b_i and A_1^{+}.

3 How to stabilize algorithms

Shirayanagi and Sweedler proposed a method of stabilizing algorithms [8]. Their motivation was that computations by symbolic algorithms waste a lot of memory by an intermediate swell of coefficients. Thus, if the algorithm is combined with a numeric computation carefully, results may be accurate and stable, and furthermore computations may be done quickly. As a numeric computation, a concept of interval arithmetic is introduced. Coefficients are described by a circular interval number, i.e., a pair of mid–point and small deviations. It is called a Bracket Coefficient. The stabilized algorithm is executed by an increasing precision of inputs, and then the result converges to the true output obtained by symbolic computation. If the bracket coefficient contains zero, then it is rewritten to zero. The process is called a Zero Rewriting.

Here, we show how Greville's Method is stabilized as follows:

1. Variables take values from **Bracket coefficient**s
2. **Zero Rewriting** is applied to the steps to obtain b_i and A_1^+
3. Repeat the algorithm by increasing digits used for computations

For sufficiently large digit computations, the stabilized Greville's method gives a result reasonably approximate to A^+.

Further, we consider the following operations between bracket coefficients. Let $A = [a, \alpha]$ and $B = [b, \beta]$ be bracket coefficients. Arithmetic operations are defined as

$$\begin{array}{ll} A + B \Rightarrow & [a + b, \alpha + \beta] \\ A - B \Rightarrow & [a - b, \alpha + \beta] \\ A \times B \Rightarrow & [a \cdot b, (|a| \cdot \beta) + (\alpha \cdot \beta) + (\alpha \cdot |b|)] \\ A/B \Rightarrow & [a, \alpha] \times [b, \beta]^{-1} \end{array}$$

where

$$[b, \beta]^{-1} = [\frac{b}{(b+\beta)(b-\beta)}, \frac{|\beta|}{(b+\beta)(b-\beta)}].$$

Operations are the same as the circular interval arithmetic. The zero rewriting operation is written as

$$[a, \alpha] = \begin{cases} 0 & \text{for } a - \alpha \leq 0 \leq a + \alpha \\ a & \text{otherwise} \end{cases}$$

4 Compare symbolic and stabilized methods

We show advantages of the stabilized method here. First, computation times of computing a generalized inverse for non–square matrices are compared. Computations are done by a algebraic computation by a Computer Algebra System, Risa/Asir, and by an algorithm stabilization method. We implement two kinds of the algorithm stabilization methods by using

1. high precision floating point package PARI [7] on Risa/Asir,
2. C++ based on the high precision floating point package NTL [6].
3. For comparisons,numeric computations by SVD(Single Value Decomposition) algorithm on Risa/Asir are also implemented.

Symbolic computation is expected to give the exact results of generalized inverses but a lot of memory and computation time is wasted. On the other hand, the algorithm stabilization method is executed by increasing the precision through increasing digit inputs. Thus, computations should be done repeatedly until the result converges to the true output.

Test inputs matrices

$$A = (a_{i,j}) \qquad \text{for} \;\; i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n$$

discussed here are

1. Elements $a_{i,j}$ are integers in 0 to $2^{16} - 1 = 65535$ and are selected randomly. Matrix size is $m = 50, n = 25$.
2. Elements are the same as above. Matrix size is $m = 100, n = 25$.
3. Elements $a_{i,j}$ are floating numbers in 0 to 1/65535=0.000015258789 and are selected randomly. Matrix size is $m = 50, n = 25$.
4. Elements are the same as above. Matrix size is $m = 100, n = 25$.
5. A special matrix discussed [1]

$$A = \begin{pmatrix} a^{-1} - 1 & a^{-1} & a^{-1} & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & a & a & a - 1 \end{pmatrix}$$

for $a = 0.999999999$.

Computations of the algorithm stabilization method are repeated by increasing the digits. In each repetition, an approximate generalized inverse A^+ of an input A is computed. To show the convergence property, it is substituted into G in (1). Error is defined that average of difference of elements of right side and left side in (1) that we showed in section 2. The computation stops if an average error of four equations of (1) becomes very small. The results of the CPU time (shown in sec.) and error are shown in

Table 1 for random integer elements and 50×25 matrix,

Table 2 for random integer elements and 100×25 matrix,

Table 3 for random floating point elements and 50×25 matrix,

Table 4 random floating point elements and 100×25 matrix,

Table 6 an example discussed by Corless and Jeffrey [1].

Computations are done on a MMX–Pentium 233MHz computer with 64MB of RAM.

Table 1. $a_{i,j}$ are random integers $0 \sim 65535$, 50×25

Symbolic(Risa/Asir)			PARI on Risa/Asir		Numeric(NTL)	
time(sec)	error	digits	time(sec)	error	time(sec)	error
1.654×10^3	0	20	17.70	303.00	10.29	2163.3
		30	18.71	171.83	11.81	992.04
		50	20.05	0.07701	13.49	0.12134
		80	21.43	1.5×10^{-13}	17.71	5.6×10^{-12}

Table 2. $a_{i,j}$ are random integers $0 \sim 65535$, 100×25

Symbolic(Risa/Asir)			PARI on Risa/Asir		Numeric(NTL)	
time(sec)	error	digits	time(sec)	error	time(sec)	error
8.532×10^3	0	20	57.65	3359.5	39.76	2517.7
		30	67.29	1421.7	45.12	1438.1
		50	74.79	0.14861	55.04	0.12208
		80	96.61	0.012510	73.40	0.012843
		100	103.99	0.0047396	87.45	0.0066235
		120	107.43	1.9×10^{-21}	113.47	4.7×10^{-31}

Results of both examples in Table.1 and Table.2 show that the algorithm stabilization works well. Results converge to accurate values after several repetitions. Matrices take too much computation time using the symbolic computation. However, the algorithm stabilization method gives accurate results in a reasonable amount of time.

Table 3. $a_{i,j}$ are random float, $0 \sim 1/65535$, 50×25

Symbolic(Risa/Asir)			PARI on Risa/Asir		Numeric(NTL)	
time(sec)	error	digits	time(sec)	error	time(sec)	error
5.265×10^5	0	20	18.34	5.9237	10.46	0.18945
		30	17.16	81.5729	11.92	0.44138
		50	19.42	0.011616	13.85	1.1×10^{-8}
		60	19.02	2.4×10^{-22}	17.25	0.54×10^{-34}

Table 4. $a_{i,j}$ are random float, $0 \sim 1/65535$, 100×25

Symbolic(Risa/Asir)			PARI on Risa/Asir		Numeric(NTL)	
time(sec)	error	digits	time(sec)	error	time(sec)	error
4.212×10^6	0	20	58.64	5.5830	40.47	0.11014
		30	59.31	143.49	46.91	0.20391
		50	86.19	0.046192	58.43	0.017624
		60	83.36	0.004459	65.12	5.6×10^{-14}
		80	91.36	0 6.1439×10^{-25}	84.07	0.94×10^{-51}

Rough features for Tables.3 and Table.4 are the same as Table.1 and Table.2 . The algorithm stabilization method works well also for ill–conditioned matrices whose elements consists of very small floating point numbers. There is little defference between the computation time of the types of implementations of figh precision floating–point computation packages, PARI and NTL. Further, results of symbolic computations in Table.1 and Table.3 show the computation time depends on the types of matrix elements and digits used in the computation. But the algorithm stabilization method doesn't depend on types and the magnitude of matrix elements.

We compare the CPU time and error of stabilization method with the SVD method. Numeric SVD method is known as one of the fastest computation method of the generalized inverse. The computation time and error of SVD method depend on threshold values and are shown in Table 5,for examples 1 and 3. The computation time needed for the algorithm stabilization seems as fast as it for SVD method. For SVD method,appropriate threshold values should be chosen.

Table 5. Computation time and error of SVD method

Example.1 (0-65535,50 × 25)				Example.3 (1-1/65535,50 × 25)			
digit	threshold	time(sec)	error	digit	threshold	time(sec)	error
20	1.0×10^{-10}	14.57	5.52×10^{-6}	20	1.0×10^{-10}	14.64	9.68×10^{-7}
	1.0×10^{-20}	22.66	1.99×10^{-13}		1.0×10^{-20}	23.80	6.85×10^{-17}
	1.0×10^{-30}	29.82	2.53×10^{-13}		1.0×10^{-30}	33.93	3.09×10^{-16}
30	1.0×10^{-10}	23.16	1.19×10^{-5}	30	1.0×10^{-10}	34.02	1.03×10^{-7}
	1.0×10^{-20}	29.66	9.43×10^{-16}		1.0×10^{-20}	33.67	19.67×10^{-18}

Table 6. Example shown in [1] for $a = 0.999999999$

Symbolic(Risa/Asir)			PARI on Risa/Asir		Numeric(NTL)	
time(sec)	error	digits	time(sec)	error	time(sec)	error
0.01	0	20	0.02	1×10^8	0.01	0.3186×10^{15}
		30	0.03	2×10^{-6}	0.01	25848
		50	0.04	1.42×10^{-22}	0.02	5×10^{-11}

An example of the generalized inverse of ill–conditioned matrix shown in [1] is solved easily by the algorithm stabilization method. Since the matrix size is small, the symbolic computation gives the result in a reasonable amount of time.

5 Conclusion

In this paper, we discussed the computation of the Moore–Penrose type generalized inverse by using Greville's algorithm. Results are shown as follows:

- How the algorithm stabilization method gives satisfactory results for obtaining the generalized inverse of a given matrix whose elements are both integers and floating point numbers.
- Since the computation time of the symbolic computation depends on the size of matrices, it is huge especially for large sized matrices and also depends on types and digits of numbers of matrix elements.
- The computation time by the algorithm stabilization method is faster than the usual symbolic computation time.
- Especially, the time of the algorithm stabilization method does not depend on properties of matrix elements.

We must discuss the following steps to be taken in the near future,

- Apply the algorithm stabilization method to other algorithms, such as Glassey's algorithm for the generalized inverse and others.
- Establish a generally used computation system of the algorithm stabilization method.

Finally, authors would like to thank Fujitsu Lab. Co. for its financial support.

References

[1] R.M. Corless and D.J. Jeffrey : The Turing factorization of a rectangular matrix, SIGSAM Bulletin, Volume 31 Number 3, pp.20–28, 1997.

[2] T. Greville : Some applications of pseudoinverse of a matrix, SIAM Review, 2, pp.15–22, 1960.

[3] 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, pp.1–16 (Electronic Proceedings), 1997.

[4] M.T. Noda, I. Makino and T. Saito : Algebraic Methods for Computing a Generalized Inverse, SIGSAM Bulletin, Volume 31 Number 3, pp.51–52, 1997.

[5] M.T. Noda, M. Izumida and M. Ochi : Evaluation of exact computation methods of the generalized inverse, Jap. Jour. Inf. Proc., 30, pp.1376–1384 ,1989. (in Japanese)

[6] V. Shoup, : A New Polynomial Factorization Algorithm and its Implementation, J.Symb.Comp.,20,pp.363–397, 1995.

[7] C. Batut, D. Bernardi, H. Cohen, M. Olivier : "User's Guide to PARI–GP", 1993.

[8] K. Shirayanagi and M. Sweedler: A theory of stabilizing algebraic algorithms, MSI Tech. Rep. 95–28, Cornell Univ., 1995.

[9] M.Noro and T.Shimoyama:"Asir User's Manual",1995.

[10] V. Shoup:NTL package. Available from `http://www.cs.wisc.edu/~shoup`