数式処理 *Bulletin of JSSAC(2017)*
Vol. 23, No. 1, pp. 10 - 19

桂先生名誉会長推戴記念特別研究会論文

行列式演算における 演算速度と速度要因に関する考察

北村竜之介　サレジオ工業高等専門学校
大墨礼子 (兵頭礼子)　サレジオ工業高等専門学校
近藤祐史　香川高等専門学校
村尾裕一　電気通信大学
齋藤友克　(株) アルファオメガ

(Received 2015/12/29 Revised 2016/4/28 Accepted 2016/5/31)

概　要

In general, computation time depends on time complexity, especially in the case of numerical computation, it is an unmistakable fact. However, in the case of symbolic computation, it is not always right. Polynomials have multiple elements, for example, variables, coefficients and constant, therefore computation time depends on many factors in the case of symbolic computation. We have been examining about relationship between computation time and factors of it. In this paper, we gives some considerations about computation time of determinant.

1　はじめに

浮動小数点数を用いて実行する数値計算アルゴリズムの議論において，除算を除いた三則演算のそれぞれ 1 回分のコストを 1 と数えることができ，キャッシュのヒット率が 100%であるならば，演算回数によって演算時間が決定する．また，除算も含めて四則演算それぞれ 1 回分をコスト 1 と数えた場合でも，多くの場合では問題ない．しかし，数式処理ソフトにおいては，32bit の整数，多倍長数，一変数多項式，多変数多項式などもアルゴリズムの四則演算の対象となる．そのため，演算回数によって演算時間が決定づけられるわけではない．浮動小数点数の演算のみを行う数値計算と異なり，数式処理では多項式等の式をそのまま取り扱うため，単純な演算回数のみでは演算速度は決定されず，それ以外の要因についても調査をする必要がある．本研究では，数式処理システム Risa/Asir の行列演算，特に行列式について 2 種類のアルゴリズム，Risa/Asir に既に実装されている分数なしガウスの消去法と，新たに top down 型の再帰的プログラムによって実現する余因子展開法を用いたアルゴリズムを実装し，その挙動についての考察を行う．

2 実装および演算速度の計測

2.1 実装したアルゴリズム

数式処理システム Risa/Asir には，行列式及び逆行列の関数は，分数なしガウスの消去法を用いて実装されている. 行列式の計算アルゴリズムは，分数なしガウスの消去法の他にも余因子展開法，ラプラス展開法，補間法，Berkowitz の方法等，多くのアルゴリズムが存在する．余因子展開法には再帰呼び出しを用いて計算を行う top down 型のアルゴリズムと，全 2×2 行列の行列式の計算結果をメモリ上に保存し，それを利用して全 3×3 行列の行列式計算の結果を計算する bottom up 型の余因子展開法が存在する．本研究では，分数なしガウスの消去法と比較するため，行列式演算に top down 型の余因子展開の計算アルゴリズムを実装する．top down 型の余因子展開法での行列式計算の疑似コードを下記に示す.

```
def matdet(MAT){
    Width, Height = 行列サイズ$
    C=MAT$
    Cmini= 1 行 1 列小さい行列を作成$
    Det=0$
    if(Width > 3){ // 行列が 3 行 3 列より大きい場合
        A=0$
        // C[0][A] の余因子を計算
        while(A<Width){
            Cmini = 余因子展開する行列を作成;
            /*再帰呼出で 1 サイズ小さい行列の行列式を計算*/
            D=matdet(Cmini)$
            if (A が偶数) {
                Det=Det+C[0][A]*D$
            } else {
                Det=Det-C[0][A]*D$
            }
            A=A+1$
        }
        /*3 × 3 の場合はサラスの方法で行列式計算*/
        else if(Width==3){
            return (サラスの方法で行列式を計算)$
        }
    }
    return Det$
}
```

これ以降，四則演算の対象を考慮せず，あらゆる四則演算 1 回分のコストを 1 と数えてよい計

算モデルにおいて，計算量を考察する．具体的には，32bit の整数，多倍長数，一変数多項式，多変数多項式などの区別はせず，これらに対する四則演算 1 回分のコストを 1 として考える．そのもとで，2 つのアルゴリズムの計算量を考えると，ガウスの消去法の計算量は $O(n^3)$ であるが，余因子展開を用いたアルゴリズムでは $O(n!)$ となり，この計算モデルにおいてはガウスの消去法を用いた場合の方が高速に演算が行われると考えられる．

2.2 使用する行列データ

行列式の演算時間を測定する際，データとして使用する行列を以下に示す．

1. 1 桁の整数
   1 から 9 までの整数をランダムに生成し，要素に格納する．以下に行列の例を示す．
$$\begin{pmatrix} 5 & 6 & 3 \\ 3 & 8 & 1 \\ 3 & 3 & 4 \end{pmatrix}$$
2. 一変数多項式
   最大次数 5 次の一変数多項式を要素に格納する．一変数多項式生成の擬似コードを以下に示す．
```
for(K=0;K<5;K++){
  a_{ij} += (random( )%10)*x^K;
}
```
3. ファンデルモンド行列
$$\begin{pmatrix} 1 & x & x^2 & x^3 \\ 1 & y & y^2 & y^3 \\ 1 & z & z^2 & z^3 \\ 1 & u & u^2 & u^3 \end{pmatrix}$$
4. 多変数要素
   多変数の単項式要素を持つ行列を作成する．一変数多項式の場合と同様，各要素が多変数多項式となる実験を行ったが実験環境の都合上，多変数多項式ではごく小さな行列の場合しか計算不可能であった．そのため，各要素は単項式となっているが計算過程及び結果が多変数多項式となる行列を用いる．行列は，各列で同一変数を，次数は列方向，行方向ともに同一となるものがないように乱数を用いて配置する．以下に行列の例を示す．
$$\begin{pmatrix} x^3 & y^4 & z & q^2 \\ x^4 & y & z^2 & q^3 \\ x & y^2 & z^3 & q^4 \\ x^2 & y^3 & z^4 & q \end{pmatrix}$$

これらの行列データでのを用いて計算速度を計測する．実行環境は，CPU　AMD Phenom II X4 945(3008.42MHz)，Memory 4GB，OS FreeBSD 10.0-Release 64bit 搭載のマシンとする．

3 演算速度の比較

3.1 行列式および逆行列の演算時間の比較

2.2 に示した行列データを用いて，行列式の演算の速度を計測した．表 1, 2, 3, 4 に行列式の計測時間を示す.

表 1: 行列式計算時間 (一桁の整数)

サイズ	消去法 (sec)	余因子展開 (sec)
4	3.00e -05	2.50e -05
5	9.00e -06	4.60e -05
6	1.60e -05	0.000216
7	2.30e -05	0.001308
8	3.60e -05	0.009375
9	5.10e -05	0.0848
10	6.90e -05	0.8673
11	0.0001938	9.097
12	0.0002549	111.7
13	0.000266	1425
14	0.000196	
15	0.000273	
16	0.000441	

行列の要素が 1 桁の整数や一変数多項式の場合，ガウスの消去法が高速に行列式の計算を行うことができ，ファンデルモンド行列や多変数要素では余因子展開法が高速に行列式の計算を行うことができる．四則演算の対象を考慮せず，あらゆる四則演算 1 回分のコストを 1 と考えてよい計算モデルでは，行列の要素が 1 桁の整数や一変数多項式の場合，ガウスの消去法が高速に行列式の計算を行うことができ，ファンデルモンド行列や多変数要素では余因子展開法が高速に行列の計算式の計算が可能なことが説明不可能である．そのため，ここからはアルゴリズム中の四則演算の対象となっている多倍長数，一変数多項式，多変数要素を詳細に調査していく．

3.2 演算中の項数の増減と係数の計測

数式処理においては，式の演算は人間が行うように，式のまま実行される．そのため，演算途中で多項式同士の演算が起こった場合には式が持つ項数が大きく増減する場合がある．ここでは，ガウスの消去法と余因子アルゴリズムを用いた場合に，行列がどのように処理されていくかを考える．ガウスの消去法で行列式を求める場合は，行列を上三角行列になるよう変形を行い，対角成分を乗算することで行列式を求めることができる．消去を行う過程において，多項式同士の演算が多数発生し，また計算によって求められた結果も項数が多い．一方，top down 型の余因子展開法を考えると，再帰呼び出しの結果を用いて行列式を計算していくため，ガウスの消去法と比較して計算途中で項数が膨らむことなく演算が行われると考えられる．このような現象か

表 2: 行列式計算時間 (1 変数多項式)

サイズ	消去法 (sec)	余因子展開 (sec)
4	0.000304	0.000356
5	0.002513	0.001716
6	0.001931	0.009952
7	0.004596	0.07221
8	0.01974	0.5805
9	0.0336	5.39
10	0.0336	47.4
11	0.05615	565.7
12	0.08379	
13	0.1345	
14	0.1922	
15	0.284	
16	0.4161	

表 3: 行列式計算時間 (ファンデルモンド行列)

サイズ	消去法 (sec)	余因子展開 (sec)
4	8.798e-05	2.2e-05
5	2e-06	7e-05
6	0.02578	0.000482
7	1.667	0.004227
8		0.04197
9		0.4607
10		5.387

表 4: 行列式計算時間 (多変数要素)

サイズ	消去法 (sec)	余因子展開 (sec)
4	0.000202	4.1e-05
5	0.001855	0.000191
6	0.04741	0.001208
7	2.081	0.01517
8		0.07579
9		0.8009
10		9.297

ら，演算中に要素同士の演算が起こった際の項数の増減が演算速度に影響を与えるとも考えられるため，ガウスの消去法および余因子展開法の両アルゴリズムにおいて，アルゴリズム中の個々の演算における係数の膨張を計測する．

まず，行列式演算における，計算途中の項数の合計をプログラムを用いて計測する．ガウスの消去法においては，１行前進消去の処理を行う毎に消去した行の各要素の項数を Risa/Asir の nmono() 関数を用いて計測し，その総和を求める．

余因子展開法においては，行列が 2 行 2 列または 3 行 3 列になるまで再帰呼び出しを行い，サラスの方法を用いて行列式を計算後，上位呼び出しに復帰する．再帰呼び出しから復帰する際に求められた行列式の項数を計測し，同様に Risa/Asir の nmono() 関数を用いて計測し，その総和を求める．式を含む要素を持つ場合に項数の増減が発生するため，一変数多項式，ファンデルモンド行列，多変数要素の場合の測定を行う．また比較のため，一桁の整数の要素を持つ行列の場合も測定する．

演算中の項数の膨張を見ると，一変数多項式では余因子展開の場合に大きくなり，ファンデルモンド行列および多変数要素ではガウスの消去法の場合に顕著に項数が膨張している．また，一変数多項式の場合に余因子展開では項数は膨張するものの，7×7 行列においてガウスの消去法の場合の 4 倍弱であるのに対し，ファンデルモンド行列の場合は，5×5 行列で消去法は余因子展開の 6 倍，多変数要素の場合は 3×3 行列で消去法は余因子展開の 26 倍と大きな倍率で膨張する．項の膨張および演算時間を考えると，演算中の項数の膨張が演算時間に大きな影響を与えていると考えられる．しかし，数式処理システムでの要因をより詳細に考えると，各項にかかっている係数も無関係であるとは言い切れない．

次に，演算過程で，各項の係数を調査し，係数がどの程度の膨らみを持つのかを推測する．それぞれ係数の出現回数の計測タイミングは，項数の計測と同様にガウスの消去法においては，１行前進消去の処理を行う毎に消去した行の各要素の演算結果の係数を Risa/Asir の coeff() 関数を用いて計測し，出現した係数の桁数及び出現回数を計測，余因子展開法においては，こちらも項数の計測の場合と同様に，再帰呼び出しから復帰する際に求められた行列式に出現する係数を Risa/Asir の coeff() 関数を用いて計測し，出現した係数の桁数及び出現回数を計測する．

一変数多項式，ファンデルモンド行列，多変数要素ともに，計算前の各要素の項の係数は 10 以下の自然数とごく小さいものを使用している．演算過程での係数の出現回数を，係数の桁数ごとに調査する．下記に，計測した係数の出現回数を示す．

Asir 言語を用いて係数の出現回数を求めているため，特にガウスの消去法の場合ではごく小さなサイズでのみの測定となるが，ファンデルモンド行列および多変数要素の場合では高速に演算できる余因子展開を用いたアルゴリズムの方が桁数の少ない係数で，さらに出現回数も少なく，この結果だけを見ると係数の桁数や出現回数も演算速度との関連があると考えられる．また，一変数多項式および１桁の整数を要素に持つ行列の場合では，項数および係数の出現回数が小さく抑えられている．本研究で行った限られた入力データ及び実験結果からのみの考察ではあるが，演算速度の面だけでなく，項数および係数の出現回数が小さく抑えられる場合は，より大きなサイズの行列まで演算を行うことができていると考えられる．数式処理システムでは，四則演算の

回数と演算対象の複雑度に依存した個々の演算の演算コストの両方を考慮する必要がある．今回の実験ではごく限られた入力要素ではあるが，項数および係数の出現回数を演算の複雑度と考えると，入力の種類とアルゴリズムの選択の両方に依存して，演算の複雑度を抑えられた場合は高速に演算ができていると考えられる．

4 まとめ

今回，行列式の演算において，要素に多項式を持つ行列を 2 つのアルゴリズムにおいて実装し，その挙動及び演算速度，演算途中での項数および係数の桁数毎の出現回数のデータを計測した．本研究での実験では，計測結果より，項数および係数が小さく抑えられた場合，演算時間を大きく縮めることができている．しかし，数式処理システムでは，四則演算の回数と演算対象の複雑度に依存した個々の演算の演算コストの両方を考慮する必要があり，その両方を考慮しなければ適切なアルゴリズムの選択は難しい．演算対象の複雑度は，一般的には演算を繰り返す毎に増加するが，演算過程において約分や減算が発生し，逆に複雑度が減少するという場合もある．さらに適用するアルゴリズムに依存するのみではなく，入力データの性質にも大きく依存する．この点が数値計算との大きな違いであり，数式処理におけるアルゴリズムの選択が難しくなる大きな要因である. 本研究ではごく限られた行列要素での実験において，演算時間と演算の複雑度を演算結果の項数及び係数の出現回数で調査し，その傾向からアルゴリズム選択の一考察を述べた．一般的な場合においての，数式処理システム上でのアルゴリズム選択法の確立は非常に難しいと考えられるが，今後もアルゴリズムの実装を進め，アルゴリズムの選択肢を増やすことによって入力データとの組み合わせも増加し，高速な演算が可能な組み合わせを選択可能となるようにシステムの充実を図っていく．

参 考 文 献

[1] Noriko HYODO, Hirokazu MURAO, Tomokatsu SAITO. Matrix Multiplication Made Fast – Practival View of Fast Matrix Operation for Computer Algebra System 数式処理, Vol.11 No.3,4 pages 3–19, 日本数式処理学会, 2005.

[2] 北村竜之介, 兵頭礼子, 近藤祐史, 村尾裕一, 齋藤友克. Risa/Asir での行列演算高速化の試み数式処理, Vol.21 No.1 pages 17–23, 日本数式処理学会, 2015.

表 5: 一変数多項式の場合の演算中の項数合計

サイズ	ガウスの消去法	余因子展開
2	8	4
3	55	7
4	313	66
5	999	328
6	3739	2497
7	5852	20187

表 6: ファンデルモンド行列の場合の演算中の項数合計

サイズ	ガウスの消去法	余因子展開
4	159	28
5	999	145
6		876
7		6139
8		49120
9		442089
10		4420900

表 7: 多変数要素の場合の演算中の項数合計

サイズ	ガウスの消去法	余因子展開
2	42	32
3	12688	484
4		1776
5		9805
6		52821

表 8: １桁の整数の場合の演算中の項数合計

サイズ	ガウスの消去法	余因子展開
2	2	1
3	10	1
4	28	6
5	58	43
6	106	276
7	178	1939
8	276	15520
9	394	139689
10	558	1376740
11	756	15285271
12	999	183665184

表 9: 消去法での 1 桁の整数の場合の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁	5 桁	6 桁	7 桁	8 桁
2	2	2						
3	5	6	2					
4	20	9						
5	29	11	15	2				
6	39	28	10	8				
7	86	47	16	7	11			
8	138	69	22	17	11	0	1	
9	170	89	38	28	19	10		
10	169	140	43	129	43	10	2	2
11	344	153	89	79	20	25	7	1
12	327	171	131	138	49	16	7	13

表 10: 消去法での一変数多項式の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁	5 桁	6 桁
2	7					
3	53	2				
4	36	1				
5	128	32				
6	333	222	68	26	10	
7	751	341	150	62	24	6

表 11: 消去法でのファンデルモンド行列の係数の出現回数

サイズ	1 桁	2 桁
4	86	
5	322	2

表 12: 消去法での多変数要素の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁
2	17	56	2	
3	108	329	372	10

表 13: 余因子展開での 1 桁の整数の場合の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁	5 桁	6 桁	7 桁	8 桁	9 桁	10 桁
2	0	1								
3	0	0	1							
4	0	4								
5	4	34	3							
6	2	150	113	11						
7	198	1394	296	46	5					
8	240	5136	9354	698	87	5				
9	19080	78006	38828	3066	432	35	2			
10	111600	693768	470526	71964	7346	498	33	5		
11	418320	8538720	5877768	321390	42516	5418	467	29	3	
12	7338240	80297280	83585664	1221240	139674	87146	9308	809	59	4

表 14: 余因子展開での一変数多項式の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁	5 桁	6 桁
2	3					
3	6	1				
4	14	4				
5	22	31				
6	1404	1083	447	56	2	
7	7331	3945	1957	511	39	2

表 15: 余因子展開でのファンデルモンド行列の係数の出現回数

サイズ	1 桁
4	48
5	360
6	2880
7	25200
8	241920
9	2540160
10	29030400

表 16: 余因子展開での多変数要素の係数の出現回数

サイズ	1 桁	2 桁	3 桁	4 桁	5 桁	6 桁
2	3	28	1			
3	12	85	185	5		
4	54	802	2906	24		
5	765	8859	17453	7836		
6	3090	68119	199646	196634	64640	24