Analysis of Algorithms, A Case Study: Determinants of Matrices With Polynomial Entries

W. M. GENTLEMAN
University of Waterloo
and
S. C. JOHNSON
Bell Telephone Laboratories

The problem of computing the determinant of a matrix of polynomials is considered; two algorithms (expansion by minors and expansion by Gaussian elimination) are compared; and each is examined under two models for polynomial computation (dense univariate and totally sparse). The results, while interesting in themselves, also serve to display two points: (1) Asymptotic results are sometimes misleading for noninfinite (e.g. practical) problems. (2) Models of computation are by definition simplifications of reality: algorithmic analysis should be carried out under several distinct computational models and should be supported by empirical data.

Key Words and Phrases: symbolic algebraic computation, analysis of algorithms, determinants of polynomial matrices
CR Categories: 5.25, 5.7

1. INTRODUCTION

Some algorithms which are efficient for integer computations are distinctly suboptimal for the analogous polynomial computations. As an example, Gentleman [3] studied powering of polynomials and showed that, even in the computation of p^4, it can be faster to compute this as $((p^2)p)p$ than as $(p^2)^2$ (as one would for fixed-length integers). Heindel [4] and Fateman [2] have also studied this problem, using more than one model.

We examine the computation of determinants of n-by-n matrices with polynomial entries. At the outset we shall restrict our problem still further by ignoring the important case where the matrix has many zero entries and by assuming that the entries in the matrix are all the same size. Our analysis focuses on two algorithms, Gaussian elimination and expansion by minors, and two computational models, dense univariate and totally sparse; Section 2 describes the algorithms; Section 3 describes the models; and Section 4 describes the application of these

This work was supported in part by the National Research Council of Canada under Grant A7407

Authors' addresses: W.M Gentleman, Mathematics Faculty Computing Facility, University of Waterloo, Waterloo, Ont., Canada, S.C. Johnson, Bell Telephone Laboratories, Murray Hill, NJ 07974.

models to the two algorithms. Finally, Sections 5 through 7 discuss the results of the theoretical analysis and give some empirical data in support of these results.

2. THE DETERMINANT PROBLEM

Suppose we are given an n-by-n matrix of polynomials. We consider two ways of computing the determinant of this matrix: Gaussian elimination and expansion by minors. We assume that all the entries in the matrix are of equal size.

We first describe the Gaussian elimination method, using exact division, which is appropriate for computations over the integers; an Altran program is given in Appendix I. If we ignore pivoting, the algorithm consists of $n-1$ steps, indexed by a variable k running from 1 to $n-1$. The kth step involves the computation of an $n-k+1$ by $n-k+1$ matrix, which we shall call $A^{(k+1)}$; the entries will be denoted $a_{ij}^{(k+1)}$, with $k \leq i, j \leq n$. The original matrix is identified with $A^{(1)}$.

For each k, the entry $a_{ij}^{(k+1)}$ is computed by the formula

$$a_{ij}^{(k+1)} = (a_{kk}^{(k)} a_{ij}^{(k)} - a_{ik}^{(k)} a_{kj}^{(k)}) / a_{k-1,k-1}^{(k-1)}$$

for $k+1 \leq i, j \leq n$, where $a_{00}^{(0)}$ is taken to be 1. The division is always exact, so that each $a_{ij}^{(k)}$ is a polynomial, and $a_{nn}^{(n)}$ is the determinant of $A^{(n)}$.

An analysis of this algorithm (see Bareiss [1], Lipson [5]) shows that each $a_{ij}^{(k)}$ is a determinant (minor) of some k-by-k submatrix of the original matrix. Since we assumed all entries in the original matrix to be of the same size, we may expect that all elements in $A^{(k)}$, for a given k, are the same size.

To compute $a_{ij}^{(k)}$ takes two multiplications, a subtraction, and a division. In general, the cost of multiplying two polynomials will depend on their size; in our situation the size is assumed to depend only on the order of the minor making up the element. Thus we shall use numbers C_{rs} to compute the cost of our algorithm, C_{rs} is the cost of multiplying a minor of order r by a minor of order s. Notice that $C_{1,1}$ is the cost of multiplying two elements from the original matrix. We assume also that an exact division of A by B to yield C has the same cost as a multiplication of B by C, and we ignore the costs of addition and subtraction.

We can now write the cost for Gaussian elimination in terms of the C_{rs}. To compute $a_{ij}^{(k+1)}$ requires two multiplications of cost C_{kk}, a division of cost $C_{k-1,k+1}$, and a subtraction whose cost we do not count. There are, for a given k, $(n-k)^2$ elements $a_{ij}^{(k+1)}$; so the total cost for the Gaussian elimination is

$$G = \sum_{k=1}^{n-1} (n-k)^2 (2C_{kk} + C_{k-1,k+1}).$$

Note that, when $C_{rs} = 1$ for all r and s, representing the familiar floating-point or fixed-length integer case, the cost becomes

$$G = 3 \sum_{k=1}^{n-1} (n-k)^2 = n^3 - \tfrac{3}{2}n^2 + \tfrac{1}{2}n.$$

We now turn our attention to the expansion by minors algorithm; an Altran program for this is given in Appendix II. We again have $n-1$ steps, indexed by k from 2 to n. At each step, we compute all of the k-by-k minors from the first k

columns (there are $\binom{n}{k}$ of them), using the $k-1$ by $k-1$ minors from the first $k-1$ columns, computed in the previous step. We ignore the bookkeeping and concentrate on the cost of the polynomial operations. Computing each k-by-k minor involves k multiplications, and $k-1$ additions and subtractions, which we ignore. Thus, using the C_{rs}, each new minor has a cost of $kC_{k-1,1}$. The total cost is thus

$$M = \sum_{k=2}^{n} \binom{n}{k} kC_{k-1,1} = n \sum_{k=2}^{n} \frac{(n-1)!}{(k-1)!(n-k)!} C_{k-1,1} = n \sum_{k=2}^{n} \binom{n-1}{k-1} C_{k-1,1}$$

$$= n \sum_{k=1}^{n-1} \binom{n-1}{k} C_{k1}.$$

Once again, we examine the cost when each C_{rs} is 1; we have

$$M = n \sum_{k=1}^{n-1} \binom{n-1}{k} = n(2^{n-1} - 1).$$

We now consider the models of polynomial computation.

3. MODELS OF COMPUTATION

The two models of polynomial computation we consider in this section are extremes in the sense that other models tend to lie between them. The models share many similarities, chiefly in the simplifying assumptions. We list the major ones here for future reference.

1. We assume that the algorithm which multiplies a polynomial with n terms by one with m terms has a cost proportional to mn.

2. We assume that the cost of coefficient operations is constant, irrespective of the size of the polynomial. (This is frequently false; we discuss our reasons for making this assumption later.)

3. Finally, we assume that, in multiplying two polynomials, terms explicitly in the product never end up with zero coefficients (cancel out).

Suppose that we multiply two polynomials having m and n terms, respectively. Under Assumption 1, the only difference between models is the *size* of the result; the multiplication *cost* is always mn. In the *sparse* model, we assume no combination of terms at all; the resulting polynomial has mn terms, the maximum possible number. Similarly, an addition of n and m term polynomials yields $m+n$ terms. In the *dense* model we assume that, subject to Assumption 3, the product of m term and n term polynomials has the least possible number of terms: $m+n-1$. This bound is attained when the polynomials are univariate with no nonzero coefficients. Similarly, addition of these polynomials yields $\max(m,n)$ terms.

In the remainder of this section, we examine our assumptions to better place these models into context. Section 4 applies the formulas to the two determinant algorithms.

The aim of every model is to describe and focus on some aspect of reality; models can frequently be described best by describing what they ignore, not what they contain. Both of these models are extreme; we suggest that most practical problems will show aspects of both models. In the most radical departure from reality, both models ignore coefficient growth. Speaking roughly, models that treat coefficient growth behave as though another (dense) polynomial variable were added to the model; for example, univariate polynomials whose coefficients grow behave very similarly to two-variable polynomials. Thus the addition of variable-length coefficients to the dense model makes the model "less dense," since intermediate results grow in size more rapidly than the univariate model we consider. Similarly, putting a dense dimension into the sparse model makes the model "more dense." We choose to analyze the sparse and dense models in their purest form and then to back up our conclusions with empirical data.

The other assumption which deserves mention is that multiplications of m terms by n terms takes time mn. In the dense case, this is immediate by using the classical multiplication; we ignore the fast Fourier techniques. In the sparse case, we usually wish to sort the terms in the product into some kind of canonical ordering. The most naive algorithm would take a time proportional to $mn \log(mn)$; the algorithm used in Altran takes a time proportional to $mn \log(\min(m,n))$. This appears to be the fastest known algorithm; we know of none which actually runs in time mn. Nevertheless, we shall neglect the effect of the log term; in practice, the bookkeeping and coefficient operations run as mn and dominate the sorting term over a wide range of practical problems.

4. APPLICATION TO THE ALGORITHMS

In this section we apply the models described in Section 3 to the algorithms described in Section 2. Recall the cost formulas for Gaussian and minor expansion:

$$G = \sum_{k=1}^{n-1} (n-k)^2(2C_{kk} + C_{k+1,k-1}); \quad M = n \sum_{k=1}^{n-1} \binom{n-1}{k} C_{k1}.$$

By Assumption (1) in Section 3, each C_{rs}, the cost of multiplying an rth-order minor by an sth-order minor, is given by

$$C_{rs} = S_r S_s,$$

where S_r and S_s are the number of terms in an rth-order and an sth-order minor, respectively. We may thus write

$$G = \sum_{k=1}^{n-1} (n-k)^2(2S_k^2 + S_{k+1}S_{k-1}); \quad M = nS_1 \sum_{k=1}^{n-1} \binom{n-1}{k} S_k.$$

The problem is now reduced to computing a value for S_k for each model and then doing some summation. In the following discussion the superscripts d and s refer to the dense and sparse models, respectively. We assume that, in each model, the initial entries of the matrix have t terms.

Computing $S_k^{(d)}$ is very simple: a dense polynomial with t terms has degree $t-1$; so a k-by-k determinant of such polynomials has degree $k(t-1)$, and thus

$k(t-1)+1$ terms. We compute

$$M^{(d)} = nt \sum_{k=1}^{n-1} \binom{n-1}{k} (k(t-1)+1)$$

$$= nt \left(\sum_{k=1}^{n-1} \binom{n-1}{k} + (t-1) \sum_{k=1}^{n-1} \binom{n-1}{k} k \right)$$

$$= nt \left(2^{n-1} - 1 + (t-1) \sum_{k=1}^{n-1} \binom{n-1}{k} k \right).$$

Moreover,

$$\sum_{k=1}^{n-1} \binom{n-1}{k} k = (n-1) \sum_{k=1}^{n-1} \frac{(n-2)!}{(n-k-1)!(k-1)!} = (n-1) \sum_{k=1}^{n-1} \binom{n-2}{k-1}$$

$$= (n-1) \sum_{k=0}^{n-2} \binom{n-2}{k} = (n-1)2^{n-2}.$$

Thus

$$M^{(d)} = nt\,(2^{n-1} - 1 + (n-1)(t-1)2^{n-2}).$$

So $M^{(d)}$ depends exponentially on n, and quadratically on t.

$G^{(d)}$ is conceptually easier, but computationally harder, to compute. We have

$$G^{(d)} = \sum (n-k)^2 (2(k(t-1)+1)^2 + ((k+1)(t-1)+1)((k-1)(t-1)+1))$$

$$= \sum (n-k)^2 (3(k(t-1)+1)^2 - (t-1)^2)$$

$$= \tfrac{1}{30} n(n-1)((3n^3 + 3n^2 - 7n + 8)(t-1)^2$$

$$+ 15n(n+1)(t-1) + 15(2n-1)).$$

Since this depends quintically on n and quadratically on t, $M^{(d)}$ is asymptotically greater than $G^{(d)}$ as n goes to infinity. Table I gives the values of $G^{(d)}$ and $M^{(d)}$ for small values of n and t. Notice that $M^{(d)}$ is smaller than $G^{(d)}$ for $n \le 5$, for $n=6$ and $t > 1$, and for $n=7$ and $t > 3$. In all other cases, $G^{(d)}$ is smaller. Thus, in spite of the evidently poor asymptotic growth of $M^{(d)}$, it appears that (at least counting the number of multiplications) expansion by minors may be faster in many practical dense situations.

The analysis in the sparse case is considerably clearer. We must first compute the sizes $S_k^{(s)}$; a k-by-k determinant is the sum of $k!$ products, each of which is the product of k t-term polynomials. By the assumption of sparseness, each product has t^k terms, and there is no cancellation in the additon; thus $S_k^{(s)} = k!t^k$.

We may now compute

$$M^{(s)} = nt \sum_{k=1}^{n-1} \binom{n-1}{k} k!t^k = nt \sum_{k=1}^{n-1} [(n-1)!/(n-k-1)!]\, t^k$$

$$= n!t^n \sum_{k=1}^{n-1} [t^{-(n-k-1)}/(n-k-1)!] \le n!t^n e^{1/t}.$$

Table I. Comparison of M^(d) with G^(d)

		t							
n	1	2	3	4	5	6	7	8	9
m 2	2	8	18	32	50	72	98	128	162
g 2	3	11	23	39	59	83	111	143	179
m 3	9	42	99	180	285	414	567	744	945
g 3	15	70	163	294	463	670	915	1198	1519
m 4	28	152	372	688	1100	1608	2212	2912	3708
g 4	42	250	634	1194	1930	2842	3930	5194	6634
m 5	75	470	1185	2220	3575	5250	7245	9560	12195
g 5	90	672	1818	3528	5802	8640	12042	16008	20538
m 6	186	1332	3438	6504	10530	15516	21462	28368	36234
g 6	165	1517	4313	8553	14237	21365	29937	39953	51413
m 7	441	3570	9387	17892	29085	42966	59535	78792	100737
g 7	273	3038	8981	18102	30401	45878	64533	86366	111377
m 8	1016	9200	24552	47072	76760	113616	157640	208832	267192
g 8	420	5572	16996	34692	58660	88900	125412	168196	217252
m 9	2295	23022	62181	119772	195795	290250	403137	534456	684207
g 9	612	9552	29892	61632	104772	159312	225252	302592	391332
m 10	5110	56300	153570	296920	486350	721860	1003450	1331120	1704870
g 10	855	15519	49611	103131	176079	268455	380259	511491	662151

The cost approaches this bound very quickly as n increases. Notice that $M^{(s)}/S^{(s)}_{(n)} \leq e^{1/t}$; so that, in general, expansion by minors does *less than three multiplications per term in the answer*!

The Gaussian result is harder to write in closed form. We have

$$G^{(s)} = \sum_{k=1}^{n-1} (n-k)^2(2S_k^2 + S_{k+1}S_{k-1})$$

$$= \sum_{k=1}^{n-1} (n-k)^2(2(k!)^2t^{2k} + (k-1)!(k+1)!t^{2k})$$

$$= \sum_{k=1}^{n-1} (n-k)^2(k!t^k)^2(2 + (k+1)/k) = \sum_{k=1}^{n-1} (n-k)^2(k!t^k)^2(3 + 1/k).$$

Clearly, the terms grow very quickly with increasing k. In fact, $G^{(s)}$ is certainly larger than its last term $(k=n-1)$:

$$G^{(s)} \geq 3((n-1)!t^{n-1})^2;$$

so

$$G^{(s)}/S_n^{(s)} \geq 3((n-1)!)^2 t^{2n-2}/n!\,t^n = 3(n-1)!\,t^{n-2}/n.$$

Thus the cost per term grows exponentially in n. Further analysis shows that $G^{(s)}$ is *always* greater than $M^{(s)}$.

To summarize the results of this section: in the dense case, for small matrices expansion by minors takes fewer multiplications, while for large problems Gaussian elimination takes fewer; in the sparse case, expansion by minors always takes fewer multiplications. Section 5 discusses these findings in more detail.

5. DISCUSSION

Recall that the sparse and dense polynomial models represented two extremes, where the results of the computation grew as fast or as slowly as possible without cancellation. Inspection of the formulas for M and G shows clearly what is happening: when S_r grows slowly (e.g. polynomially) in k, the $\binom{n-1}{k}$ term in the formula for M causes an exponential dependence of M on n, while G has only a polynomial dependence on n. Thus for large enough n, G is smaller. On the other hand, when S_r grows rapidly (e.g. exponentially), the $S_k{}^2$ terms in the formula for G come to dominate the $\binom{n-1}{k} S_r$ term in the formula for M, and M is smaller.

The question of which algorithm to use in practice thus seems to depend on how rapidly we expect S_k to grow with respect to k. If we believe that our problems include a considerable combination of terms and small answers, we tend to favor Gaussian elimination; if we believe in large answers, as caused by variable-length coefficients and multivariate problems, we tend to favor expansion by minors.

Another striking feature of the above results is the extent to which an "exponential" formula for M is better than a "polynomial" formula for G over a considerable range of practical problems, even in the dense case. This serves to show the pitfalls of asymptotic analysis when applied to noninfinite problems.

6. SOME EMPIRICAL RESULTS

Since the analyses in the previous sections make many simplifying assumptions (although the typical asymptotic analysis makes more!), the concluded superiority of minor expansions may still not appear in practice. To investigate how well our assumptions are borne out, we made a number of tests with the Altran programs, given in Appendixes I and II, which tend to support the above analysis. As an example, we attempted to compute the determinants of n-by-n symmetric Toeplitz matrices for orders through 8, using Gaussian elimination and expansion by minors. The nth-order matrix was defined in terms of $n+1$ variables, $X_0, X_1, \ldots, X_n$, by $a_{ij} = X_{|i-j|}$.

The results are given in Table II. These represent CPU time, in seconds, for the Altran programs running with a 20,000-word workspace in an IBM 360/75 at Waterloo University.

Thus, even with a highly structured matrix in seven or eight variables, the minor method is a clear winner.

Table II

Order	Minors	Gauss
4	14.5	10.4
5	36.5	47.2
6	97.3	247.6
7	252.43	>1475.2 (out of time)
8	>588 (out of space)	—

7. CONCLUSION

We hope to have made two larger points in addition to the direct results of this paper. We have seen that an asymptotically inferior method cannot always be dismissed when dealing with practical problems. In effect, the algebraic manipulations which are currently practical frequently lie much closer to zero than to infinity. This is not to say that asymptotic analysis is not useful, but just that it must be kept in its place!

Our second goal has been to indicate that rival models, or extreme models, can and should be used to gain insight into the algorithms being studied. In this sense, a continuum of crude approximations seems to us less revealing than a couple of well-chosen extreme points.

REFERENCES

1. Bareiss, E.H. Silvester's identity and multistep integer-preserving Gaussian elimination. *Math. Computation 22* (1968), 565–578.
2. Fateman, R.J. On the computational powers of polynomials. Dep. Math. Rep., M.I.T., Cambridge, Mass.
3. Gentleman, W.M. Optimal multiplication chains for computing a power of a symbolic polynomial. SIGSAM Bull. 18, April 1971. To appear in *Math. Computation.*
4. Heindel, L.E. Computation of powers of multivariate polynomials over the integers. *J. Computer Syst. Sci.*, 1 (Feb. 1972).
5. Lipson, J.D. Symbolic methods for the computer solution of linear equations with applications to flowgraphs. In P.G. Tobey, Ed., Proc. 1968 Summer Institute on Symbolic Mathematical Computation, I.B.M. Programming Lab. Rep. FSC69-0312, June 1969.

Received June 1973, revised October 1974

(*Please see appendixes on next page*)

APPENDIX I

```
        procedure det( a, n )
        algebraic array a;  integer n;  value a, n

#       this procedure calculates the determinant of the
#       n by n matrix a by integer preserving gaussian
#       elimination

        integer array (1:n) reorder
        integer i, ii, j, k, kk, sign=1
        algebraic divisor = 1

        do k = 1, n
            reorder(k) = k
            doend
        do k = 1, n-1
            do i = k, n
                if( a( reorder(i), k ) .ne. 0 ) goto pivot
                doend
            return(0)

   pivot:
            if( i .eq. k ) kk = reorder(k)
            else do
                kk = reorder(i);  reorder(i) = reorder(k)
                reorder(k) = kk;  sign = -sign
                doend
            do i = k+1, n
                ii = reorder(i)
                do j = k+1, n
                    a(ii,j) = ( a(ii,j)*a(kk,k) - a(ii,k)*a(kk,j) ) /
                        divisor
                    doend
                a(ii,k) = .null.
                doend
            divisor = a(kk,k)
            do j = k,n
                a(kk,j) = .null.
                doend
            doend

        return( sign * a( reorder(n), n ) )
        end
```

APPENDIX II

```
        procedure det(a,n)

        algebraic array a;  integer n;  value a, n

#       this procedure calculates the determinant of the n order
#       matrix a by expanding by minors
        integer array (0:n,0:n) binom
        altran integer pascal
        algebraic array (0:1,0:pascal(n,binom)) minor=0

#       pascal initializes the binomial coefficient table, and
#       returns as its value n choose n/2 less 1
#       we keep track of our minors using the method of indexing
#       described in CACM 3 (1960), p. 235 (R. M. Brown)
```

```
        integer loc, addr, sign, m, j, k, old=0, new=1
        integer array (0:n+1) i = -1

        i(0) = n
        do loc = 1, n; minor(old, loc-1) = a(1, loc); doend
        do m = 1, n-1

#           compute m+1 order minors from m order

            loc = 0
            do j = 1, m; i(j) = m-j; doend
   nextminor:
            k = n-1; j = 0; addr = loc+binom(k, m+1); sign = 1
   nextuse:
            if( k .eq. i(j+1) ) do
                j = j+1; sign = -sign; doend
            else _
                minor(new, addr) = minor(new, addr) +
                    sign * a(m+1, k+1) * minor(old, loc)
            if( k .gt. 0 ) do
                k = k-1; addr = addr-binom(k, m-j); goto nextuse
                doend

#               dispose of unnecessary minor and increment indices

            minor(old, loc) = 0
            loc = loc+1
            do j = m, 1, -1
                i(j) = i(j)+1
                if( i(j) .lt. i(j-1) ) goto nextminor
                else i(j) = m-j
                doend


#           all m+1 order minors are now calculated

            old = 1-old; new = 1-new
            doend;

        return( minor( old, 0 ) )
        end
```

```
        procedure pascal(n, binom)
        integer n, binom;  array binom

#       initializes the table of binomial coefficients,  and returns the
#       maximum number of minors,  less 1

        integer i, j

        binom(0, 0) = 1
        do i = 1, n
            binom(i, 0) = 1;  binom(i-1, i) = 0
            do j = 1, i
                binom(i, j) = binom(i-1, j-1)+binom(i-1, j)
                doend
            doend

        return( binom(n, iquo(n, 2))-1)

        end
```