Fraction Free Gaussian Elimination for Sparse Matrices

Hong R. Lee & B. David Saunders
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19711

August 2, 1994

Abstract

A variant of the fraction free form of Gaussian elimination is presented. This algorithm reduces the amount of arithmetic involved when the matrix has many zero entries. The advantage can be great for matrices with symbolic entries (integers, polynomials, expressions in trigonometric functions, etc.). These claims are supported with some analysis and experimental data.

Let A be a $n \times m$ matrix over an integral domain $\mathcal{R}$. We will assume that the leading principal minors of A are nonzero. This assumption is not necessary, but simplifies the discussion by eliminating the issue of pivoting. First we offer a recursive description of Bareiss' standard fraction free Gaussian elimination [Bar68].

$$\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \leq i \leq n, 1 \leq j \leq m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}, \text{ for } 1 \leq k < n, k < i \leq n, k < j \leq m.
\end{aligned}$$

It is well known that

$$A_{i,j}^{(k)} = \begin{vmatrix} A_{1,1} & \cdots & A_{1,k} & A_{1,j} \\ \vdots & & \vdots & \vdots \\ A_{k,1} & \cdots & A_{k,k} & A_{k,j} \\ A_{i,1} & \cdots & A_{i,k} & A_{i,j} \end{vmatrix}.$$

Thus when $m = n$, $\det(A) = A_{n,n}^{(n-1)}$, and when $A = \begin{pmatrix} M & b \\ I & 0 \end{pmatrix}$, for square matrix M and column vector b, the solution to $Mx = b$ is the vector x whose ith entry is $-A_{n+i,n+1}^{(n)}/A_{n,n}^{(n-1)}$. Similarly, when $A = \begin{pmatrix} M & I \\ I & 0 \end{pmatrix}$, for square matrix M, the adjoint of M is the matrix

$-(A^{(n)}_{n+i,n+j})$, etc. Of course, to save space, these computations may be folded into the first n rows in the usual way (Gauss-Jordan elimination).

The point of this paper is to note that for sparse matrices it is often possible to avoid computation of many of the $A^{(k)}_{i,j}$ and/or reduce the number of arithmetic operations used to compute those that are needed. Let us call the algorithm to do this SFF (sparse fraction free algorithm). It's basic form is described in section 1. In an appendix we discuss a variant of the "inner loop" of the computation that can make a time difference of about a factor of two. Section 2 concerns analysis of the computing time for some specific classes of matrices and entries. In section 3 we report on some timing experiments.

1 Sparse Fraction Free Algorithm

The algorithm is similar to Bareiss' described above, except that some of the steps are delayed and ultimately avoided due to a telescoping effect. A second "history" matrix provides the bookkeeping for this process. We describe the algorithm by recursive definitions of the data, B, and the history, h.

$$
\begin{aligned}
B^{(0)}_{i,j} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, & (1)\\
B^{(k)}_{i,j} &= B^{(k-1)}_{i,j}, \text{ (no computation)} & (2)\\
&\quad \text{for } 1 \le k < n, k < i \le n, k < j \le m \text{ such that } B^{(k-1)}_{i,k} B^{(k-1)}_{k,j} = 0, &\\
B^{(k)}_{i,j} &= \frac{A^{(k-1)}_{k,k} A^{(k-1)}_{i,j} - A^{(k-1)}_{i,k} A^{(k-1)}_{k,j}}{A^{(k-2)}_{k-1,k-1}}, \text{ (as in the standard algorithm, but see below)} & (3)\\
&\quad \text{for } 1 \le k < n, k < i \le n, k < j \le m \text{ such that } B^{(k-1)}_{i,k} B^{(k-1)}_{k,j} \ne 0. &
\end{aligned}
$$

In order to perform the computation specified in the last equation, it is necessary to know the $A^{(k)}_{i,j}$'s. It turns out that these may be computed from the the $B^{(k)}_{i,j}$'s with at most a multiplication and (exact) division each as explained below. The fact is simply that the the $B^{(k)}_{i,j}$'s may be not "up to date", i.e., may be $A^{(h)}_{i,j}$'s where $h < k$. The matrix $(h_{i,j})$ is used to express this relation precisely.

$$
\begin{aligned}
h^{(-1)}_{0,0} &= -1, &\\
h^{(0)}_{i,j} &= 0, \text{ for } 1 \le i \le n, 1 \le j \le m, & (4)\\
h^{(k)}_{i,j} &= h^{(k-1)}_{i,j}, \text{ for } 1 \le k < n, k < i \le n, k < j \le m \text{ such that } B^{(k-1)}_{i,k} B^{(k-1)}_{k,j} = 0, & (5)\\
h^{(k)}_{i,j} &= k, \text{ for } 1 \le k < n, k < i \le n, k < j \le m \text{ such that } B^{(k-1)}_{i,k} B^{(k-1)}_{k,j} \ne 0. & (6)
\end{aligned}
$$

The references to $A^{(k-1)}_{*,*}$ and $A^{(k-2)}_{*,*}$ can be eliminated in the definition of B because, as is shown in proposition 1, $A^{(m)}_{i,j} = B^{(m)}_{i,j} A^{(m-1)}_{m,m} / A^{(l-1)}_{l,l}$ for $l = h^{(m)}_{i,j}$. It will be convenient to abbreviate the key elements $A^{(m)}_{m+1,m+1}$ as P_m, for $m \ge 0$, with $P_{-1} = 1$. Then the update

(3) to B may be expressed as

$$B_{i,j}^{(k)} = \frac{(B_{k,k}^{(k-1)}P_{k-2}/P_{s-1})(B_{i,j}^{(k-1)}P_{k-2}/P_{v-1}) - (B_{i,k}^{(k-1)}P_{k-2}/P_{u-1})(B_{k,j}^{(k-1)}P_{k-2}/P_{t-1})}{P_{k-2}}$$
$$= \frac{P_{(k-1)}B_{i,j}^{(k-1)}P_{k-2}}{P_{s-1}P_{v-1}} - \frac{B_{i,k}^{(k-1)}B_{k,j}^{(k-1)}P_{k-2}}{P_{u-1}P_{t-1}}, \tag{7}$$

where s, t, u, v are defined by

$$\begin{pmatrix} s & t \\ u & v \end{pmatrix} = \begin{pmatrix} h_{k,k}^{(k-1)} & h_{k,j}^{(k-1)} \\ h_{i,k}^{(k-1)} & h_{i,j}^{(k-1)} \end{pmatrix}.$$

In many cases, computation of expression (7) can be simplified. This will be discussed in the appendix.

Proposition. 1 *Let A be a matrix in $\mathcal{R}^{n,m}$ with nonzero leading principal minors, for an integral domain $\mathcal{R}$. Then for B, h, and P as above and for $i, j > k$, we have*

$$A_{i,j}^{(k)} = B_{i,j}^{(k)}P_{k-1}/P_{l-1}, \ \textit{where } l = h_{i,j}^{(k)},$$

hence the determinant (when $n = m$) is

$$\det(A) = P_{n-1} = B_{n,n}^{(n-1)}P_{n-1}/P_{l-1} \ \ (= A_{n,n}^{(n-1)}), \ \textit{where } l = h_{n,n}^{(n-1)}.$$

The condition on the leading principal minors can be easily removed by the use of pivoting.

Proof. The proof is by induction on k. Since the relation $B_{i,j}^{(k)} = A_{i,j}^{(l)}$, for $l = h_{i,j}^{(k)}$, is evident from the defining formulas, it suffices to show that $A_{i,j}^{(k)} = A_{i,j}^{(l)}P_{k-1}/P_{l-1}$. This is evident when $l = k$. However, $l < k$ occurs only when the sparsity condition holds, $B_{i,k}^{(k-1)}B_{k,j}^{(k-1)} = 0$. By the induction hypothesis, it must be that also $A_{i,k}^{(k-1)}A_{k,j}^{(k-1)} = 0$, so that

$$\begin{aligned} A_{i,j}^{(k)} &= A_{i,j}^{(k-1)}A_{k,k}^{(k-1)}/A_{k-1,k-1}^{(k-2)} \\ &= (\frac{B_{i,j}^{(k-1)}P_{k-2}}{P_{l-1}})(P_{k-1})/(P_{k-2}) \\ &= B_{i,j}^{(k-1)}P_{k-1}/P_{l-1} \\ &= B_{i,j}^{(k)}P_{k-1}/P_{l-1}. \end{aligned}$$

The second equality is by induction. The next shows the main idea, that we get a telescoping product. For the last, observe that $h_{i,j}^{(k)} = h_{i,j}^{(k-1)}$ under precisely the same conditions that $B_{i,j}^{(k)} = B_{i,j}^{(k-1)}$ (conditions in which $l < k$). □

The telescoping feature in the delayed updates reveals the source of most of the advantage of this algorithm. Each computation in the telescoped product would be computed by the standard fraction free method, computations that are here avoided entirely. Additionally, one can often avoid some updates and the division in the innermost loop, and as a result be left with multiplications of smaller cost. This is discussed in the appendix.

Finally, we remark that Bareiss' two step, more generally multi-step, fraction free elimination can also benefit from this method of avoiding computation of telescoping products.

2 Computation Cost Analyses

To study the advantage algorithm SFF can give, let us compare its performance to the standard fraction free method on banded matrices. It is reasonable to restrict our attention to the cost of the multiplications. For this analysis, we will assume the entries conform to the "dense" model of Gentleman and Johnson, [GJ76]. See also [HS75]. In this model, the cost of multiplication and division of entries of measure d and e is $O(de)$. Example entry domains conforming reasonably well to this model are the integers, with the measure of an integer being it's logarithm or bit length, and the domain of dense univariate polynomials over a finite field, where the polynomial degree can be taken as the measure. We will assume the nonzero entries of the input matrix are all of measure d and that no cancellation reduces the expected measure of results, which is $\mu(A_{i,j}^{(k)}) = (k+1) * d$.

Consider a banded matrix with lower bandwidth l and upper bandwidth u, by which we mean that $A_{i,j} \neq 0$ only if $-l \leq j - i \leq u$. In particular, each row and column has at most $l + 1 + u$ nonzero entries. Banded matrices arise in many linear models. Among the most important examples are Hessenberg matrices ($l = 1, u = n$), and pentadiagonal matrices ($l = u = 2$). We don't pursue yet another important class, the tridiagonal matrices, for which the computational results are the same as for pentadiagonal matrices up to a constant factor.

Proposition. 2 *In the dense model described above we have the following growth rates for the costs:*

Algorithm	*Matrix Type* *full*	*banded*	*Hessenberg*	*pentadiagonal*
Standard fraction free (one step form)	$O(n^5 d^2)$	$O((l+u)n^4 d^2)$	$O(n^5 d^2)$	$O(n^4 d^2)$
Algorithm SFF (sparse fraction free)	$O(n^5 d^2)$	$O(lun^3 d^2)$ *(but if* $l = 1$ *or* $u = 1$*, we get* $O((l+u)n^2 d^2)$*)*	$O(n^3 d^2)$	$O(n^3 d^2)$

Proof. In the case of a full matrix, at the kth step, the update is performed on the $(n-k)^2$ entries below and to the right of the k, k entry (the pivot). The entries involved are $k \times k$ minors of the original matrix, hence of measure kd. Thus the cost of the multiplications is $O((kd)^2)$ as is the cost of the division. Summing over k, the cost is $O(\sum_{k=1}^{n}(n-k)^2(kd)^2) = O(n^5 d^2)$.

In the case of a banded matrix (see figure 1) the work of the standard algorithm is restricted to entries in the band so the update at the kth step is to the entries in the band below and to the right of the pivot, fewer than $(n-k)(l+u+1)$. The replacement of a $n-k$ factor by the $l+u+1$ factor is the effect that determines the result when we sum over k. We remark that for entries outside the rectangle consisting of the l rows and u columns immediately below and to the right of the pivot, one fewer multiplication is required in the update. When l and u are held fixed, this reduces by a third the *coefficient* of the leading term asymptotically.

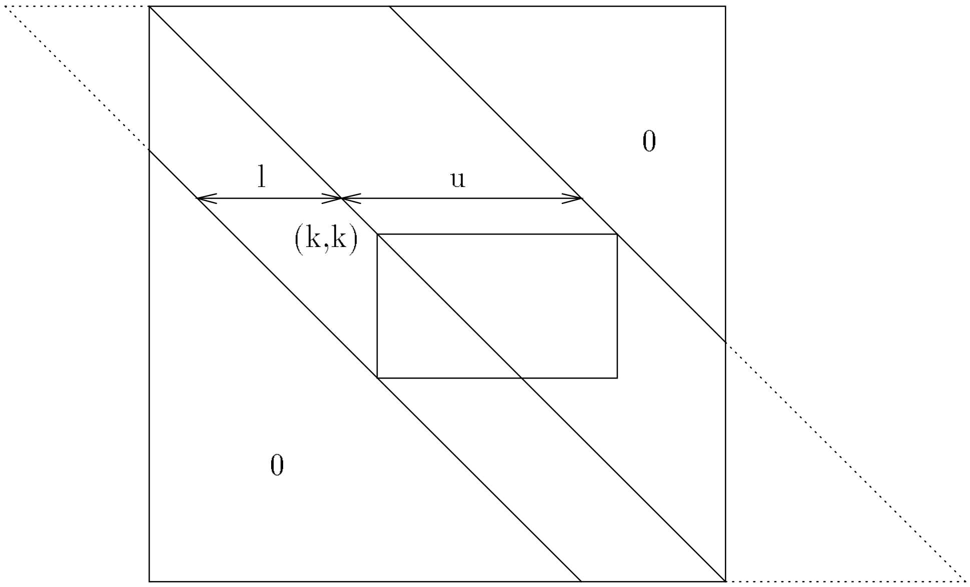

Figure 1: SFF on a banded matrix

However, at the kth step of elimination, algorithm SFF updates entries *only* in the rectangle consisting of the l rows and u columns immediately following the pivot (k, k entry). Thus the number of entries updated is reduced to lu, leading to the indicated asymptotic cost upon summing.

Moreover the entries on the lower and right edges of the rectangle are input matrix entries of measure d, not kd, having never previously been updated. This reduces the cost but not the leading term asymptotically when l and u are greater than one. However, when $l = 1$ or $u = 1$ the rectangle is reduced to a line. The cost of each multiplication is reduced to $kd \times d$ and, of course, the number of entries updated goes from lu to $\max(l, u) = \mathrm{O}(l + u)$. Thus the sum is then $\mathrm{O}(\sum_{k=1}^{n} \max(l, u)k^2 d) = \mathrm{O}((l + u)n^2 d^2)$ reducing the exponent of n to two in the sum.

The values for special cases of the banded matrix formulas then follow: for Hessenberg because $l = 1$ and for pentadiagonal because $l = u =$ constant 2. □

It is worth noting that in many cases the nonzero entries in a symbolic matrix may be sparse expressions in one or more variables. In this case the growth in size of the $A_{i,j}^{(k)}$ and hence the costs of the multiplications is at a faster rate than the dense model indicates. In this case the advantage of working with lower order minors (as SFF does when it can) is even greater.

3 Experiments

To check the speedup in practice, the sparse fraction free Gaussian elimination (SFF) and Bareiss' fraction free Gaussian elimination (FF) were implemented in SAC2, [Col80], and run on a Sequent Symmetry. The optimization detail discussed in the appendix was included in the implementation of SFF used for these measurements. Although the Symmetry had 8 processors to support multiprocessing, these measurements are sequential, using only one processor. We remark that, like other forms of Gaussian elimination, algorithm SFF is readily parallelized.

We first consider the banded structures analyzed in the previous section. Experiments were run on Hessenberg and pentadiagonal matrices with integer entries. The entries in the band were chosen at random in the range $(-2^7, 2^7)$. The times, reported in figure 2 on the next page, conform roughly to the dense model discussed in the previous section. In the plots, the measured times are shown along with curves of the form kn^e, for the sake of comparison. The k's are chosen to put the curve in the vicinity of the data, and the exponents are 5 for FF on Hessenberg matrices, 4 for FF on pentadiagonal, and 3 for SFF in both cases. One sees rough correlation of the data trends with the estimates.

We also ran a full 40×40 integer matrix, getting times of 64.12 seconds for the standard fraction free algorithm, FF, and 64.24 seconds for SFF. We are encouraged by the negligible overhead for the use of SFF in this case (in which it has no advantage).

To further explore performance of the algorithm, we measured times on integer polynomial banded matrices and on less structured sparse matrices. The table immediately below shows times for the banded polynomial case. The entries in the band are univariate polynomials of degree 2 with coefficients chosen at random from the interval $(-2^7, 2^7)$.

shape	alg	Integer Univariate Polynomial Entries matrix sizes					
		10	12	14	16	18	20
Hessenberg	FF	13.5	46.06	76.17	149.21	321.31	714.15
	SFF	.94	2.15	3.38	5.27	7.99	10.14
pentadiagonal	FF	11.67	36.48	62.19	151.24	270.53	.81
	SFF	.62	.97	1.47	2.08	2.79	3.62

To look at the 50% sparse situation, we used matrices whose nonzero entries are in a checkerboard pattern. SFF updates alternate halves of the nonzero entries in alternate steps. Thus one would expect a time around half that of FF (and we get it). In this case it would be better to preprocess the matrix, discovering that it can be decomposed as a block diagonal matrix, then separately computing the determinant of the two diagonal blocks. More generally it is beneficial to block triangulate any decomposable matrix and apply SFF to the diagonal blocks. Finally we tried sparse, unstructured, and (probably) indecomposable matrices with either 10% nonzero entries in each row or 20% nonzeroes in each row, the column indices of the nonzero entries being chosen at random. The trivial case of a 10% full 10×10 matrix is effectively a diagonal matrix (or has zero determinant). This case wasn't measured. As before, the entries were integer or univariate polynomial. We also tried integral bivariate polynomial entries to show the effect of the often problematic expression swell in sparse expression arithmetic. The bivariate polynomials are of degree 2 (at most) in each variable and sparse. They have an average of 3 nonzero terms: probability of each of the 9 coefficients being nonzero is 1/3 and the nonzero coefficients are again chosen in the interval $(-2^7, 2^7)$.

shape	alg	Integer Entries matrix sizes								
		40	50	60	70	80	90	100	110	120
Hessenberg	FF	15.1	46.4	96.3	168	296	541	787	1340	2380
	SFF	.67	.99	1.59	2.37	3.37	5.92	6.18	9.8	14.5
pentadiagonal	FF	11.0	18.9	34.3	72.1	90.6	135	194	271	458
	SFF	0.25	0.46	0.62	0.83	0.85	1.06	1.56	1.53	1.77

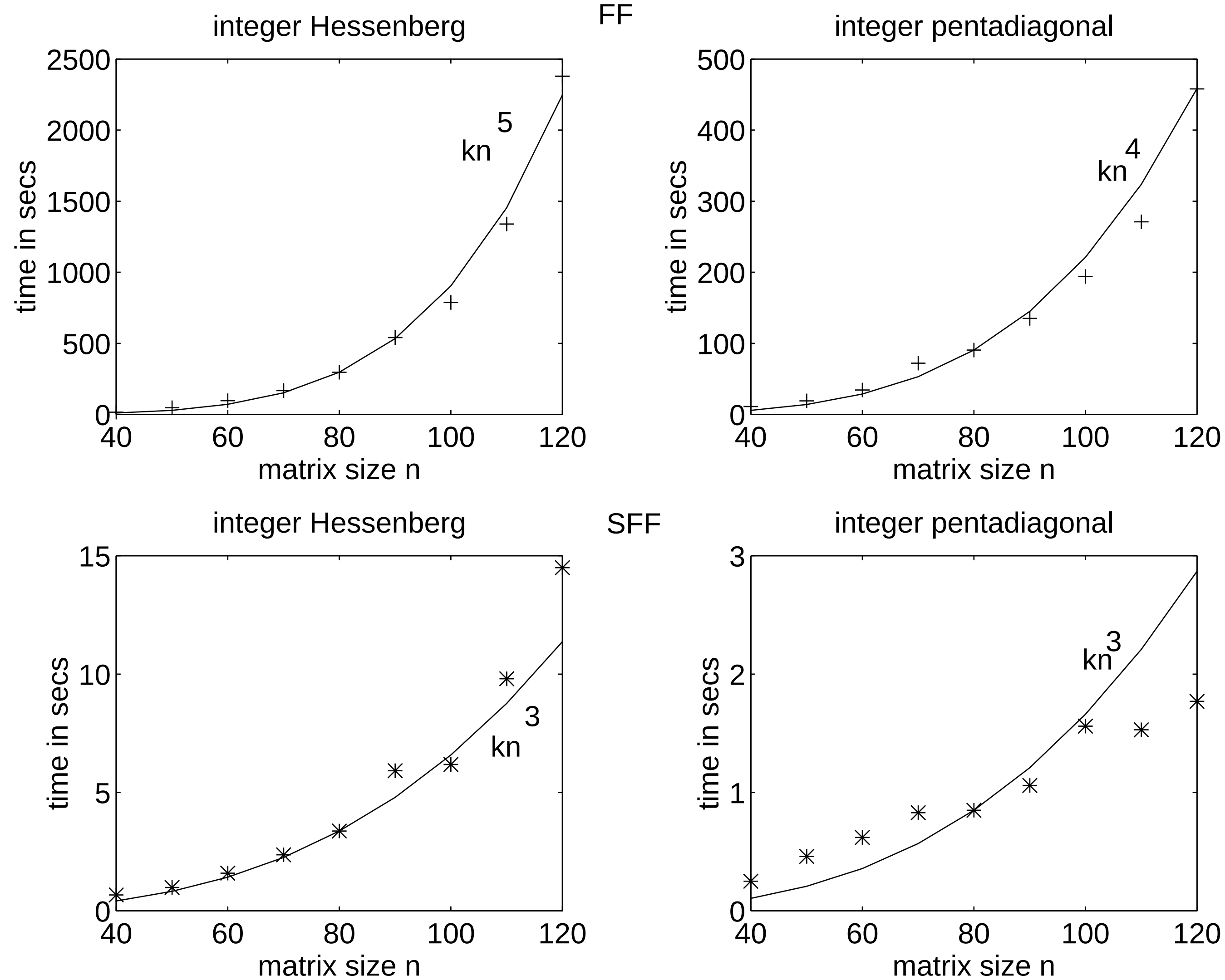

Figure 2:

entry types	size	alg	pattern of nonzero entries		
			checkerboard	10% nonzero	20% nonzero
integers	40×40	FF	24.62	15.26	36.33
		SFF	13.88	5.86	27.64
univariate polynomials	20×20	FF	606.98	161.05	528.05
		SFF	283.69	4.13	310.03
bivariate polynomials	10×10	FF	1023.18	-	183.17
		SFF	673.99	-	46.42

The speedups in the random matrices are not as exciting as in the banded cases. This is primarily due to the fill-in that occurs. For example, in the 20% full 40×40 matrix, we found essentially complete fill-in had occurred by the middle (20th) step. Thus beyond that point SFF has no advantage. Better pivoting strategy has the potential to give significant advantage here.

The implementations of SFF and FF that were used in these measurements employed a partial pivoting strategy in which the smallest element of the kth column is chosen as the pivot at the kth step. Complete pivoting (choosing the pivot anywhere in the remaining $n - k$ rows and $n - k$ columns) is likely to pay off for symbolic matrices because of the high benefit of keeping expressions small. However, it is not so obvious on what principle to choose a pivot. Among the features to consider when choosing are small size, step of latest update (small $h_{i,j}$), and number of zeroes in the row and column of the element. Of course, small size reduces the cost of multiplying the pivot during the step. Small $h_{i,j}$ has the advantage that size is *likely* to be small because the position has been subjected to less expression swell and, furthermore, all other entries in the same row or column are equally "out of date" suggesting likely small size for them also, hence further update cost savings. Finally, choosing an element in a very sparse row and/or column, decreases the likelyhood of fill-in. Beyond this, global dependency analysis may be used to reduce fill-in and is likely to work well with SFF. We have experimented some with pivot strategies, but no clear choice is evident. The issue of pivoting strategy in symbolic matrices is beyond the scope of this paper. It is a topic well worth further study.

4 Conclusion

We have presented a variant of Bareiss' fraction free algorithm. The variant, SFF, avoids certain ultimately telescoping products. This leads to improved performance on sparse matrices, vastly improved performance for some banded matrices in particular.

Polyalgorithms are used in practice for determinant and related problems, because of tradeoffs between the algorithms with respect to matrix size, entry sparsity, matrix sparsity, and available parallelism. Polyalgorithms select among minor expansion, modular methods (evaluation/interpolation), and Gaussian elimination (including Bareiss' version). The SFF form should make Gaussian elimination advantageous for a wider class of matrices.

We thank the editor and the referees for many useful suggestions which have improved the paper substantially.

References

[Bar68] E. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Mathematics of Computation*, 22:565–578, 1968.

[Col80] G. E. Collins. ALDES and SAC2 now available. *SIGSAM Bulletin*, 14:2–19, 1980.

[GJ76] W. M. Gentleman and S. C. Johnson. Analysis of algorithms, a case study: Determinants of matrices with polynomial entries. *ACM Transactions on Mathematical Software*, 2:232–241, 1976.

[HS75] E. Horowitz and S. Sahni. On computing the exact determinant of matrices with polynomial entries. *Journal of the ACM*, 22:38–50, 1975.

5 Appendix

We consider two approaches to the computation of formula (7) showing that one method, called "division first" below, can be more than twice as fast as the other "exact division" method.

To simplify notation, we write formula (7) as

$$B_{i,j}^{(k)} = \frac{SVP_{k-2}}{P_{s-1}P_{v-1}} - \frac{TUP_{k-2}}{P_{t-1}P_{u-1}}. \tag{8}$$

where

$$\begin{pmatrix} S & T \\ U & V \end{pmatrix} = \begin{pmatrix} B_{k,k}^{(k-1)} & B_{k,j}^{(k-1)} \\ B_{i,k}^{(k-1)} & B_{i,j}^{(k-1)} \end{pmatrix}, \begin{pmatrix} s & t \\ u & v \end{pmatrix} = \begin{pmatrix} h_{k,k}^{(k-1)} & h_{k,j}^{(k-1)} \\ h_{i,k}^{(k-1)} & h_{i,j}^{(k-1)} \end{pmatrix}.$$

Each term may simplify further. For example, if $k-1=s$ or $k-1=v$ then the first term simplifies to SV/P, where $P=P_{v-1}$ or P_{s-1}, respectively. In our implementation we proceed as follows. Up-to-date entries in row k, $A_{k,j}^{(k-1)}$, are computed as in proposition 1, and the $B_{k,j}^{(k-1)}$ are kept as well. Then if $v<k-1$, $\bar{S}V/P_{v-1}$ is computed using the up-to-date $\bar{S}=A_{k,j}^{(k-1)}$, otherwise SV/P_{s-1} is computed using $S=B_{k,j}^{(k-1)}$. Since this S and P_{s-1} are likely smaller expressions, this reduces arithmetic cost. A similar process is used to compute the second term. Thus the expression always reduces to the form $SV/P - TU/\hat{P}$, where $\hat{P}=P_{t-1}$ or P_{u-1}. The divisions are not necessarily exact, but the expression may be computed (for entries in an arbitrary integral domain) in the form $(SV\hat{P}-TUP)/P\hat{P}$, this division being exact. It is usually better to divide first then subtract, providing of course that the entry domain admits a suitable form of division with remainder such as the Euclidean algorithm or pseudo-division.

To indicate the advantage of division first, consider the case in which the $\deg S = \deg T = d$, $\deg U = \deg V = \lambda d$, and then $\deg P = \deg \hat{P} = \lambda d - 1$. We simplify to this bivariate representation of the degrees for the rough analysis that follows. It is representative of cases likely to occur during elimination, But note that we mean to represent the case that the degrees of $U, V, P, \hat{P}$ are *approximately* related to those of S and T by a factor λ. In fact, when $P=\hat{P}$, there is a simplification of the computation to $(SV-TU)/P$, and the issue at hand is moot. Let us further assume that multiplication of degree n by degree m polynomials costs nm and division costs $(n-m)m$. We ignore addition and subtraction. Then we have

$SV/P - TU/\hat{P}$(division first)

operation	degree	cost
$X_1 := SV$	$(1+\lambda)d$	λd^2
$X_2 := X_1/P$	$d+1$	$\lambda d^2 - (1-\lambda)d - 1 = (d+1)(\lambda d - 1)$
$X_3 := TU$	$(1+\lambda)d$	λd^2
$X_4 := X_3/\hat{P}$	$d+1$	$\lambda d^2 - (1-\lambda)d - 1$
$X_5 = X_2 - X_4$	$d+1$	$-$
X_1 thru X_5 total		$4\lambda d^2 - (2-2\lambda)d - 2$

$(SV\hat{P}-TUP)/(P\hat{P})$(exact division)		
operation	degree	cost
$Y_1 := SV$	$(1+\lambda)d$	λd^2
$Y_2 := Y_1\hat{P}$	$d+2\lambda d-1$	$(\lambda^2+\lambda)d^2-(1+\lambda)d=(1+\lambda)d(\lambda d-1)$
$Y_3 := TU$	$(1+\lambda)d$	λd^2
$Y_4 := Y_3P$	$d+2\lambda d-1$	$(\lambda^2+\lambda)d^2-(1+\lambda)d$
$Y_5 := Y_2-Y_4$	$d+2\lambda d-1$	$-$
$Y_6 := P\hat{P}$	$2\lambda d-2$	$\lambda^2d^2-2\lambda d+1=(\lambda d-1)^2$
$Y_7 := Y_5/Y_6$	$d+1$	$2\lambda d^2-(2-2\lambda)d-2=(d+1)(2\lambda d-2)$
Y_1 thru Y_7 total		$(3\lambda^2+6\lambda)d^2-(4+2\lambda)d-1$

Thus for large d, the division-first approach has the advantage by a factor of roughly $(3\lambda^2+6\lambda)/4\lambda$, i.e., 1.5 to 2.25 for λ between 0 and 1,

The following two lemmas provide the necessary machinery for the division-first approach in the specific cases where the entry domain is the integers or the ring of (univariate) polynomials over a unique factorization domain. This latter category includes multivariate polynomials of course.

Proposition. 3 *If $a,b,c,d\in \mathbf{Z}$ and $\frac{a}{b}-\frac{c}{d}\in \mathbf{Z}$, then for q,r,q',r' defined by $a=qb+r, 0\le |r|<|b|, rb>0$ and $c=q'd+r', 0\le |r'|<|d|, r'd>0$, it follows that $\frac{a}{b}-\frac{c}{d}=q-q'$.*

To paraphrase (for $a=SV, b=P, c=TU, d=\hat{P}$): If Euclidean division $(SV)/P$ is done so that the remainder has the sign of P and similarly for $(TU)/\hat{P}$, then the difference of these fractions is just the difference of the quotients.

Proof. Exercise for the reader. □

Proposition. 4 *If $A,B,C,D\in\mathcal{U}[x]$ and $A/B-C/D\in\mathcal{U}[x]$, where $\mathcal{U}$ is a unique factorization domain, then for α,Q,R such that $\alpha A=QB+R, 0\le\deg(R)<\deg(B)$, there exist unique Q',R' such that $\alpha C=Q'D+R', 0\le\deg(R')<\deg(D)$ and $A/B-C/D=\frac{Q-Q'}{\alpha}$.*

To paraphrase: Any α sufficient for one of the pseudo-divisions will suffice for the other and will divide all coefficients of $Q-Q'$. We remark that use of an $\alpha\ne 1$ can be required. For example consider the following polynomials in $\mathbf{Z}[x]$: $A=2x^3-x-1, B=2x^2-x-1, C=x^2+2x+1, D=2x^2+3x+1$. Then the pseudo-divisions of A by B and C by D both require $\alpha=2$ and yet $BD|(AD-BC)$ in $\mathbf{Z}[x]$.

Proof. Consider α' such that $\alpha' C=Q'D+R', 0\le\deg(R')<\deg(D)$, and such that α' is relatively prime to the content of Q'. Such a pseudodivision multiplier always exists, for otherwise, α' would have a common factor with R' as well, and this factor can be removed. Then

$$\alpha\alpha'(\frac{A}{B}-\frac{C}{D})=(\alpha' Q-\alpha Q')+(\frac{\alpha' R}{B}-\frac{\alpha R'}{D}).$$

Since the left hand side is a polynomial by hypothesis, so is the right hand side. Thus the rightmost difference, the R's term, having degree of numerator less than degree of denominator, must be zero. But then α' divides $\alpha Q'$. Since α' has no common factor with Q', we have $\alpha' | \alpha$ and the conclusion follows. $\square$

These variants of the arithmetic in computing $B_{i,j}^{(k)}$ can reduce the cost. Not only will the number of operations be fewer, but also the sizes of the operands may be much smaller. In the experiments, SFF names the algorithm based on the $B_{i,j}^{(k)}$ recurrence, including these optimizations when applicable.