Computing the Invariant Structure of Integer Matrices: Fast Algorithms into Practice

Colton Pauderis
cpauderi@uwaterloo.ca

Arne Storjohann
astorjoh@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

We present a new heuristic algorithm for computing the determinant of a nonsingular $n \times n$ integer matrix. Extensive empirical results from a highly optimized implementation show the running time grows approximately as $n^3 \log n$, even for input matrices with a highly nontrivial Smith invariant structure. We extend the algorithm to compute the Hermite form of the input matrix. Both the determinant and Hermite form algorithm certify correctness of the computed results.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

Keywords

Integer matrix; determinant; Hermite normal form

1. INTRODUCTION

Computing the exact integer determinant of a nonsingular integer matrix $A \in \mathbb{Z}^{n\times n}$ is a classical problem. The problem provides a "canonical" example of a key feature of symbolic computation: the growth in bitlength of numbers in the output compared to those in the input. On the one hand, if we let $||A|| = \max_{ij} |A_{ij}|$, then considering a diagonal input matrix we see that we may have $\log |\det A| \geq n \log ||A||$. On the other hand, Hadamard's bound gives that $\log |\det A| \leq (n/2)\log n + n \log ||A||$. Thus, the bitlength of the determinant can be up to n times that of entries in the input matrix. Textbooks on computational mathematics often use the problem of computing the determinant to illustrate the technique of homomorphic imaging and Chinese remaindering: this gives a deterministic algorithm to compute $\det A$ using $O(n^4(\log n + \log ||A||)^2)$ bit operations, even assuming standard, quadratic, integer arithmetic.

A lot of effort has been devoted to obtaining improved upper bounds for the complexity of computing $\det A$. We will not give a complete survey here but refer to [9, 10, 17]. An initial breakthrough was Kaltofen's 1992 [8] division free algorithm to compute the determinant of a matrix over a ring: the algorithm can be adapted to compute $\det A$ in $O\tilde{}(n^{3.5} \log ||A||)$ bit operations. High-order lifting and integrality certification [17] can be used to compute the determinant in about the same time (asymptotically, up to logarithmic factors) as required to multiply together two matrices having the same dimension and size of entries as A, thus in $O\tilde{}(n^3 \log ||A||)$ bit operations assuming standard matrix multiplication and pseudo-linear integer arithmetic. While the algorithm in [17] does achieve the important so called "reduction to matrix multiplication" goal, it does not make an attractive candidate for implementation as it is presented because the constant suppressed by the $O\tilde{}$ notation seems very large. For example, the algorithm begins by embedding the input matrix into a matrix of more than twice the dimension which has many entries chosen randomly. The algorithm we describe in this paper also relies on high-order lifting but does not require an increase in the dimension.

The heuristic determinant algorithm of Abbot, Bronstein & Mulders [1] is based on the well-known phenomenon that the largest invariant factor of the matrix (the smallest positive integer s_n such that $s_n A^{-1}$ is integral) is a factor of the determinant that is often very large. For randomly chosen $v_1, v_2 \in \mathbb{Z}^{n\times 2}$, the minimal $s \in \mathbb{Z}_{>0}$ such that both $sA^{-1}v_1$ and $sA^{-1}v_2$ are integral is likely to be equal to s_n, or at least a large factor, thus decreasing the number of images of the determinant that need to be computed using the classical Chinese remainder based determinant algorithm mentioned above. Consider the following nonsingular input matrix.

$$A = \begin{bmatrix} 33 & 8 & -50 & 45 & -38 \\ -20 & 62 & 39 & 11 & -79 \\ 13 & -82 & -52 & -65 & -37 \\ -35 & -81 & 3 & 114 & 7 \\ -100 & 14 & -114 & -22 & -10 \end{bmatrix}. \qquad (1)$$

If we choose

$$\begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} 10 & 45 \\ -16 & -81 \\ -9 & -38 \\ -50 & -18 \\ -22 & 87 \end{bmatrix}$$

then

$$A^{-1}\left[\begin{array}{cc} v_1 & v_2 \end{array}\right] = \left[\begin{array}{cc} \frac{1428470455}{3313087328} & \frac{43150207}{161614016} \\ \frac{673936589}{2484815496} & \frac{66351701}{121210512} \\ -\frac{1462901509}{9939261984} & -\frac{516047293}{484842048} \\ -\frac{1221838091}{9939261984} & \frac{138504781}{484842048} \\ \frac{89642859}{414135916} & \frac{18215255}{20201752} \end{array}\right].$$

The denominators of $A^{-1}\left[\begin{array}{cc} v_1 & v_2 \end{array}\right]$ have least common multiple $s = 19878523968$, which for this example is actually equal to $-\det A$. But even if the heuristic finds a large factor or even the complete determinant, the running time of the method is still quartic in n, even for a random matrix, because the expected bitlength of the gap between $|\det A|$ and Hadamard's bound (and thus the bitlength of the images needed by the homomorphic imaging scheme to guarantee to compute the correct determinant) is $\Theta(n)$ [1, Section 3].

Eberly, Giesbrecht & Villard [5] relax the requirement that the determinant should be certified correct, and use additive preconditioners combined with binary search to get a Monte Carlo algorithm requiring

$$O\tilde{}(n^3(\log||A||)^2\sqrt{\log|\det A|}) \tag{2}$$

bit operations to recover the Smith form (and thus also the determinant) of A. Recall that the Smith form of A is a diagonal matrix $S = \mathrm{Diag}(s_1, s_2, \ldots, s_n)$ such that $S = UAV$ for unimodular matrices U and V. Note that $|\det A| = s_1 s_2 \ldots s_n$. Using Hadamard's bound for $|\det A|$ in (2) gives $O\tilde{}(n^{3.5}(\log||A||)^{2.5})$ in the worst case. This last cost estimate is quite pessimistic in the average case because the algorithm is highly sensitive to the Smith invariant structure of A: for an input matrix with only $O(1)$ nontrivial invariant factors the running time is only $O(n^3(\log n + \log||A||)^2(\log n))$ bit operations. Moreover, a careful analysis in [5] shows that an integer matrix with entries chosen uniformly and randomly from an interval

$$\Lambda = \{a, a+1, \ldots, a+\lambda-1\} \tag{3}$$

for any $a \in \mathbb{Z}$ and $\lambda \in \Omega(n)$ is expected to have $O(1)$ nontrivial invariant factors.

In this paper we describe a new heuristic algorithm for computing $\det A$ that has the following features:

- The algorithm certifies correctness of the computed determinant.

- The algorithm is especially fast for propitious inputs (e.g., matrices with few nontrivial invariant factors) but still seems to work very effectively for matrices with a highly nontrivial invariant structure.

We can illustrate the main idea of our algorithm by using the input example in (1). Consider the first random projection $A^{-1}v_1$. The minimal $d_1 \in \mathbb{Z}_{>0}$ such that $d_1 A^{-1} v_1$ is integral is $d_1 = 9939261984$, giving us a large factor of $\det A$. But instead of only using the denominator of $A^{-1}v_1$, we also use the vector of numerators to produce an upper triangular basis T_1 of a superlattice of the integer lattice generated by the rows of A. In particular, in Section 2 we give an algorithm to produce a nonsingular and upper triangular matrix T_1 with minimal magnitude determinant such that $T_1 A^{-1} v_1$ is integral. For this example we obtain

$$T_1 = \left[\begin{array}{ccccc} 1 & & & 15 & 183835840 \\ & 1 & & 4 & 294625615 \\ & & 1 & 1 & 159758078 \\ & & & 24 & 300295265 \\ & & & & 414135916 \end{array}\right].$$

The matrix T_1 has positive diagonal entries and the off-diagonal entries in each column are nonnegative and strictly less than the diagonal entries in the same column. Recall that such matrices are said to be in Hermite form, a canonical presentation of row lattices. Continuing with the example, we can use the second projection $A^{-1}v_2$ to compute a second triangular factor T_2, the minimal triangular denominator of

$$T_1 A^{-1} v_2 = \left[\begin{array}{c} \frac{165758732}{1} \\ \frac{531308447}{2} \\ \frac{144048601}{1} \\ \frac{541532731}{2} \\ \frac{746825455}{2} \end{array}\right]. \tag{4}$$

We obtain

$$T_2 = \left[\begin{array}{ccccc} 1 & & & & 0 \\ & 1 & & & 1 \\ & & 1 & & 0 \\ & & & 1 & 1 \\ & & & & 2 \end{array}\right].$$

Since the rows of T_1 generate a superlattice of the lattice generated by the rows of A, we can remove T_1 from A by computing

$$B_1 = AT_1^{-1} = \left[\begin{array}{ccccc} 33 & 8 & -50 & -18 & 12 \\ -20 & 62 & 39 & 1 & -51 \\ 13 & -82 & -52 & 5 & 69 \\ -35 & -81 & 3 & 40 & 43 \\ -100 & 14 & -114 & 64 & 32 \end{array}\right].$$

The factor T_2 can also be removed to obtain

$$B_2 = B_1 T_2^{-1} = \left[\begin{array}{ccccc} 33 & 8 & -50 & -18 & 11 \\ -20 & 62 & 39 & 1 & -57 \\ 13 & -82 & -52 & 5 & 73 \\ -35 & -81 & 3 & 40 & 42 \\ -100 & 14 & -114 & 64 & -23 \end{array}\right].$$

Fast unimodularity certification [12] can now be used to certify that B_2 is unimodular (i.e., with determinant ± 1). If B_2 is unimodular then $(\det T_1)(\det T_2)$ must be, up to sign, equal to the determinant of A. (Note that if B_2 is determined to be unimodular, the sign of the determinant can be recovered by computing $\det B_2 \bmod p$ for single odd prime p.) If B_2 had not been unimodular then further projections $v_3, v_4, \ldots$ can be computed and used to find further triangular matrices $T_3, T_4, \ldots$ which can be factored from the work matrix. On the one hand, if k is the number of nontrivial invariant factors in the Smith form of A, then at least k projections will be required to capture the complete determinant. On the other hand, from the lattice conditioning analysis of [3] we know $k + O(1)$ projections will be sufficient with high probability. We remark that in [18, 14] a "bonus idea" method is developed that uses the numerators of the projections to recover the penultimate invariant factor in addition to just the last invariant.

A main computational task used in the approach just described is to solve nonsingular linear systems to compute projections $A^{-1}v$ for some $v \in \mathbb{Z}^{n\times m}$. The most efficient algorithms for nonsingular linear system solving are based on linear p-adic lifting [3, 4, 11]. The simplest variant of linear p-adic lifting has two phases. The first phase computes a low precision inverse $C := A^{-1} \bmod p$ for a prime p with $\log p \in \Theta(\log n + \log ||A||)$. This first phase thus has cost $O(n^3(\log n + \log ||A||)^2)$ bit operations assuming standard, quadratic integer arithmetic. The second phase computes a truncated p-adic expansion

$$A^{-1}v \equiv c_0 + c_1 p + c_2 p^2 + \cdots + c_{k-1}p^{\ell-1} \bmod p^\ell,$$

each $c_i \in \mathbb{Z}^{n\times m}$ with entries reduced modulo p, from which the rational solution vector can be recovered using rational number reconstruction provided the precision ℓ is high enough. The required precision depends on the size of numerators and denomininators if $A^{-1}v$. Computing one of the terms in the p-adic expansion requires a constant number of premultiplications with C and A of matrices of dimension $n \times m$ filled with entries of bitlength $O(\log n + \log ||A||)$. Thus, the second phase has cost $O(\ell m \times n^2(\log n + \log ||A||)^2)$. Note that if the required precision k and the number of projections m satisfy $\ell m \in O(n)$, the overall cost of producing $A^{-1}v$ is bounded by $O(n^3(\log n + \log ||A||)^2)$ bit operations.

We claimed above that our algorithm is also highly effective for matrices with nontrivial invariant structure, even with $k \in \Omega(n)$ nontrivial Smith invariant factors. However, the method as sketched above would solve $\Omega(n)$ nonsingular rational systems at full precision, equivalent in cost to computing the exact inverse of A. Instead of computing the projection $T_1A^{-1}v_2$ by first computing $A^{-1}v_2$ and then premultiplying by T_1, consider computing $B_1^{-1}v_2$, equal to the vector in (4). Since the first factor T_1 has captured a large factor of the $\det A$, the denominator of $B_1^{-1}v_2$ is very small. It would be desirable to reduce the number of p-adic lifting steps (the precision ℓ) to be proportional to the bitlength of information actually contained in the projection, namely the bitlength of the denominator. We can accomplish this by first precondtioning the projection with a so called high-order residue [12]. A high-order residue R of B_1 can be computed in $O(n^3(\log n + \log ||B_1||)^2(\log n))$ bit operations, and has the property that $B_1^{-1}Rv_2$ will be nearly proper. For this example, one example of such a high-order residue is

$$R = \begin{bmatrix} -15 & -15 & -15 & -15 & 1 \\ 31 & 31 & 31 & 31 & 6 \\ -47 & -47 & -47 & -47 & -4 \\ -30 & -30 & -30 & -30 & 1 \\ -104 & -104 & -104 & -104 & 55 \end{bmatrix}$$

with

$$B_1^{-1}Rv_2 = \begin{bmatrix} -\frac{92}{1} \\ -\frac{97}{2} \\ -\frac{92}{1} \\ -\frac{97}{2} \\ -\frac{97}{2} \end{bmatrix}.$$

The above projection will yield the same triangular factor T_2. Thus, our implementation can effectively exploit the same bitlength versus dimension paradigm we used for our worst case determinant algorithm [17]. For example, compute one projection at full precision, two at about half the precision, four at (at most) a quarter of the precision, etc. In particular, the p-adic lifting precision ℓ and the number of projections m will satisfy $\ell m \in O(n)$ at each phase.

For various reasons, we don't give a rigorous cost analysis of our algorithm in this paper. First, a reduction to matrix multiplication for the problem of computing the determinant has already been given in [17]. Second, a detailed analysis of most of the key subroutines is already available. (For nonsingular solving we refer to [4, 11, 17]. For the high order residue computation see [12].) Third, for the worst case of the problem, when the Hermite form has many nontrivial columns, there remain some technical challenges to overcome to arrive at an algorithm that has expected running time provably cubic in n, see Section 6. We can, though, state the following rigorous cost estimate for an important class of matrices. For an input matrix $A \in \mathbb{Z}^{n\times n}$ that has a Hermite with only $k \in O(1)$ nontrivial columns, the entire Hermite form can be captured with high probability using a single random projection of column dimension $O(1)$. The dominant cost of our algorithm in this case is a single high-order residue computation to certify correctness. The expected running time of our algorithm in this case is $O(n^3(\log n + \log ||A||)^2(\log n))$ bit operations assuming standard integer arithmetic. Moreover, it follows from the proof of [5, Lemma 6.1] that if entries in $A \in \mathbb{Z}^{n\times n}$ are chosen uniformly and randomly from an interval

$$\Lambda = \{a, a+1, \ldots, a+\lambda-1\} \tag{5}$$

for any $a \in \mathbb{Z}$ and $\lambda \in \Omega(n)$, then the Hermite form of A has the shape

$$\left[\begin{array}{c|c} I_{n-k} & * \\ \hline & * \end{array}\right]$$

where the expected value of k is $O(1)$. Our algorithm is thus very likely to be effective for random inputs.

The rest of this paper is organised as follows. Section 2 details our procedure for computing the minimal triangular denominator. Sections 3 and 4 give our algorithms for the determinant and Hermite normal form, respectively. In Section 5, we compare our implementation against implementations of alternative heuristic Hermite form algorithms [13]. Section 6 concludes.

2. AN ALGORITHM FOR MINIMAL TRIANGULAR DENOMINATOR

Let $v \in \mathbb{Q}^{n\times 1}$. In this section we show how to compute a nonsingular upper triangular matrix $T \in \mathbb{Z}^{n\times n}$ with minimal magnitude determinant such that Tv is integral; we call such a T a *minimal triangular denominator* of v. Our algorithm is based on the following lemma.

LEMMA 1. *Let $v \in \mathbb{Q}^{n\times 1}$ and $d \in \mathbb{Z}_{>0}$ be such that $w := dv$ is integral. Write the Hermite form of*

$$B := \left[\begin{array}{c|c} d & \\ \hline w & I_n \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)} \tag{6}$$

as

$$\left[\begin{array}{c|c} * & * \\ \hline & H \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}. \tag{7}$$

Then $H \in \mathbb{Z}^{n\times n}$ is a minimal triangular denominator of v.

PROOF. The unique unimodular matrix U that transforms B to Hermite form is given by

$$\begin{aligned} U &= \left[\begin{array}{c|c} * & * \\ \hline & H \end{array}\right] \left[\begin{array}{c|c} d & \\ \hline w & I_n \end{array}\right]^{-1} \\ &= \left[\begin{array}{c|c} *d^{-1} & * \\ \hline -Hv & H \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}. \end{aligned} \tag{8}$$

Since U is integral the submatrix $-Hv$ shown in (8) is integral. Now, suppose that T is a minimal triangular denominator of v. Then we could replace H in the definition of U with T and still arrive at an integral matrix in (8). Since U is unimodular, with determinant ± 1, we must have $|\det H| = |\det T|$. $\square$

Let $v \in \mathbb{Q}^{n\times 1}$ and $d \in \mathbb{Z}_{>0}$ be such that $w := dv$ is integral, as in Lemma 1. Let us define the entries of our input vector w and target H as

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \text{and} \quad H = \begin{bmatrix} h_1 & h_{12} & \cdots & h_{1n} \\ & h_2 & \cdots & h_{2n} \\ & & \ddots & \vdots \\ & & & h_n \end{bmatrix}. \tag{9}$$

The obvious approach to compute a minimal triangular denominator (not necessarily in Hermite form) of v is to simply apply unimodular row operations to triangularize the input matrix in (6). To this end, define Gcdex to be the operation that takes as input $a, b \in \mathbb{Z}$ and returns as output s, t, v, h, g such that

$$\begin{bmatrix} s & t \\ v & h \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} g \\ 0 \end{bmatrix}$$

with $sh - tv = \pm 1$ and g a greatest common divisor of a and b. We further specify that $h > 0$ and $0 \le t < h$.

Now, if we compute $s_n, t_n, v_n, h_n, g_n = \text{Gcdex}(d, w_n)$, then the following unimodular transformation of the input matrix zeroes out the entry occupied by w_n.

$$\left[\begin{array}{c|cccc} s_n & & & & t_n \\ \hline & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ v_n & & & & h_n \end{array}\right] \left[\begin{array}{c|cccc} d & & & & \\ \hline w_1 & 1 & & & \\ \vdots & & \ddots & & \\ w_{n-1} & & & 1 & \\ w_n & & & & 1 \end{array}\right] = \left[\begin{array}{c|cccc} g_n & & & & t_n \\ \hline w_1 & 1 & & & \\ \vdots & & \ddots & & \\ w_{n-1} & & & 1 & \\ & & & & h_n \end{array}\right]$$

If we initialize $B \in \mathbb{Z}^{(n+1)\times(n+1)}$ to be the input matrix in (6), the following loop applies similar transformations to zero out the entries occupied by $w_n, w_{n-1}, \ldots, w_1$. After the loop completes the work matrix B will be upper triangular.

```
$g_{n+1} := d$;
for $i = n$ downto $1$ do
    $s_i, t_i, v_i, h_i, g_i := \text{Gcdex}(g_{i+1}, w_i)$;
    $\begin{bmatrix} B[1,*] \\ B[i+1,*] \end{bmatrix} := \begin{bmatrix} s_i & t_i \\ v_i & h_i \end{bmatrix} \begin{bmatrix} B[1,*] \\ B[i+1,*] \end{bmatrix}$
od
```

The problem with this approach is that the top row of the work matrix will fill in and the subsequent application of the unimodular transformations will become too expensive. Indeed, at the start of iteration $i-1$ the work matrix has the shape

$$B = \left[\begin{array}{c|ccc|ccc} g_i & & & & t_i & \cdots & * \\ \hline w_1 & 1 & & & & & \\ \vdots & & \ddots & & & & \\ w_{i-1} & & & 1 & & & \\ \hline & & & & h_i & \cdots & * \\ & & & & & \ddots & \vdots \\ & & & & & & h_n \end{array}\right] \tag{10}$$

Instead, we perform the gcd operations only to determine the diagonal entries h_i, for $i = n, n-1, \ldots, 1$, omitting the application of the unimodular transformation to the work matrix. Once the diagonal entries have been precomputed, the off-diagonal entries $h_{1,i}, \ldots, h_{i-1,i}$ in column i of H, now in increasing order $i = 1, 2, \ldots, n$, are computed by appealing to the definition of H as the minimal denominator of the vector w/d.

We begin by initializing $w^{(0)}$ to be the the first column of the matrix in (6). For some $i \ge 1$, suppose the first $i-1$ columns of H have been computed. Then considering (10), we see that the vector

$$\left[\begin{array}{c} d \\ \hline \bar{w}_1 \\ \vdots \\ \bar{w}_{i-1} \\ \hline w_i \\ \vdots \\ w_n \end{array}\right] := \left[\begin{array}{c|ccc|ccc} 1 & & & & & & \\ \hline & h_1 & \cdots & h_{1,i-1} & & & \\ & & \ddots & \vdots & & & \\ & & & h_i & & & \\ \hline & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{array}\right] \left[\begin{array}{c} d \\ \hline w_1 \\ \vdots \\ w_{i-1} \\ \hline w_i \\ \vdots \\ w_n \end{array}\right]$$

will have all entries divisible by $g_i = d/(h_i h_{i+1} \cdots h_n)$, which is a multiple of $h_1 h_2 \cdots h_{i-1}$. At iteration i the algorithm works with the vector

$$w^{(i-1)} := \left[\begin{array}{c} d \\ \hline \bar{w}_1 \\ \vdots \\ \bar{w}_{i-1} \\ \hline w_i \\ \vdots \\ w_n \end{array}\right] \frac{1}{h_1 h_2 \cdots h_{i-1}}. \tag{11}$$

We now compute off-diagonal entries $h_{1,i}, h_{2,i}, \ldots, h_{i-1,i}$ to be the unique integers in the range $[0, h_i)$ such that the vector

$$\left[\begin{array}{c|ccc|ccc} 1 & & & & & & \\ \hline & 1 & & & h_{1,i} & & \\ & & \ddots & & \vdots & & \\ & & & 1 & h_{i-1,i} & & \\ \hline & & & & h_i & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{array}\right] w^{(i-1)} \tag{12}$$

has all entries divisible by h_i.

Algorithm `hcol`, shown in Figure 1, implements the above method to compute H. Given a nonzero integer b, operation $\text{Rem}(a, b)$ computes the unique integer in the range $0 \le$

```
hcol$(w, d)$
Input: A vector $w \in \mathbb{Z}^{n\times 1}$ and $d \in \mathbb{Z}$.
Output: $H \in \mathbb{Z}^{n\times n}$, a minimal triangular denominator of $w/d$.

  1. [1. Diagonal entries]
     $g := d$;
     for $i = n$ downto 1 do
         $*, t_i, *, h_i, g :=$ Gcdex$(g, w_i)$
     od;

  2. [2. Off-diagonal entries]
     for $i = 1$ to $n$ do
         assert: $\left[\begin{array}{c} d \\ w \end{array}\right] = w^{(i-1)}$
         if $h_i = 1$ then next fi;
         for $k = 1$ to $i - 1$ do
             $h_{k,i} :=$ Rem$(-t_i w_k, h_i)$;
             $w_k :=$ Rem$(w_k + h_{k,i} w_i, d)$
         od;
         $w_i := h_i w_i$;
         $d, w := d/h_i, w/h_i$
     od;
     return $H$ as in (9)
```

Figure 1: Algorithm hcol

Rem$(a, b) < |b|$ that is congruent to a modulo b. Phase 1 computes the diagonal entries of H. Phase 2 computes the off-diagonal entries in column i of H for $i = 1, \ldots, n$. The vector w and modulus d are updated in place. At the start of loop iteration i the algorithm works with the $n + 1$ integers in $w^{(i-1)}$ as shown in (12). The inner loop and first line after the inner loop computes the off-diagonal entries in column i of H and performs the update shown in (12). The last line removes the common factor h_i to arrive at $w^{(i)}$.

To analyse the cost of algorithm hcol we need to introduce a cost model. The number of bits in the binary representation of an integer a is given by

$$\lg a = \begin{cases} 1, & \text{if } a = 0 \\ 1 + \lfloor \log_2 |a| \rfloor, & \text{if } a > 0. \end{cases}$$

Using standard integer arithmetic [2, Chapter 3], a and b can be multiplied together in $O((\lg a)(\lg b))$ bit operations. Operation Gcdex also costs $O((\lg a)(\lg b))$ bit operations. For an integer a and nonzero integer b, we can express $a = qb + r$ with $0 \le r < |b|$ using $O((\lg q)(\lg b))$ bit operations.

Theorem 2. *If entries in $w \in \mathbb{Z}^{n\times 1}$ are reduced modulo d, then algorithm* hcol *computes the minimal triangular denominator (in Hermite form) of w/d using $O(n(\log d)^2)$ bit operations.*

Proof. Correctness of the algorithm follows from the previous discussion.

For the cost analysis, let D denote the initial value of d as passed into the algorithm. Assume without loss of generality that $D > 1$. Phase 1 performs n extended gcd computations with numbers bounded in magnitude by D. This has cost bounded by $O(n(\log D)^2)$ bit operations.

Now consider phase 2. At the start of each loop iterations all entries in w are reduced modulo d, a divisor of D. Using the fact that $|t_i| < h_i$ and $|h_{k,i}| < h_i$, each individual operation from $\{+, -, \times, /, \mathrm{Rem}\}$ performed during iteration i of the loop uses $O((\lg D)(\lg h_i))$ bit operations. Since the number of operations from $\{+, -, \times, /, \mathrm{Rem}\}$ performed during a single iteration of the outer loop is bounded by $O(n)$, there exists an absolute constant c such that loop iteration i has cost bounded by $cn(\lg D)(\lg h_i)$ bit operations. Let $L = \{i \mid h_i > 1\}$. Then the total cost of phase 2 is bounded by

$$\begin{aligned} \sum_{i\in L} cn(\lg D)(\lg h_i) &\le cn(1 + \log_2 D)\sum_{i\in L}(1 + \log_2 h_i) \\ &\le cn(2\log_2 D)(2\sum_{i\in L}\log_2 h_i) \\ &= 4cn(\log_2 D)(\log_2 h_1 h_2 \cdots h_n) \\ &\le 4cn(\log_2 D)(\log_2 D). \end{aligned}$$

□

We remark that our determinant algorithm will call hcol with a sequence of vectors that have denominator $d_1, \ldots, d_k$ with $d_1 \cdots d_k = |\det A|$. The total cost of all calls to hcol will thus be bounded by $O(n(\log|\det A|)^2)$ bit operations, which becomes $O(n^3(\log n + \log||A||)^2)$ in the worst case using Hadamard's bound for $|\det A|$.

3. THE PROJECTION METHOD FOR DETERMINANT

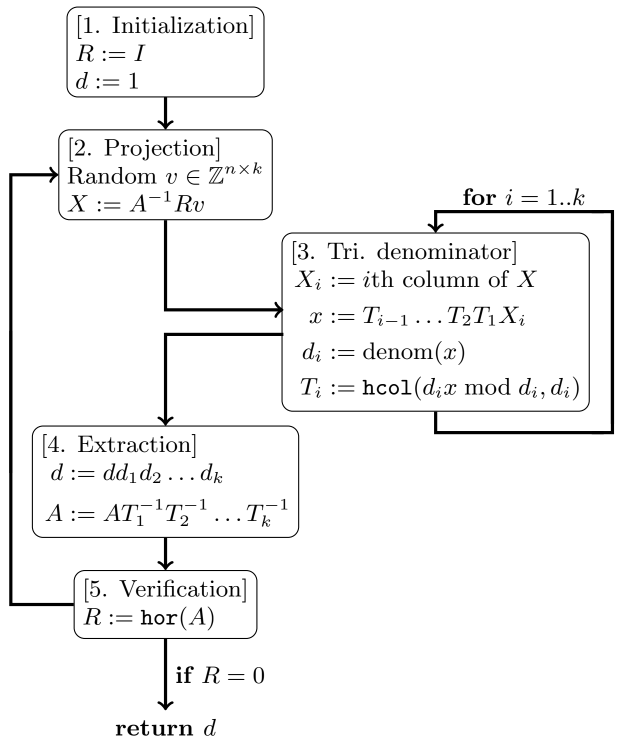
[1. Initialization]
$R := I$
$d := 1$
[2. Projection]
Random $v \in \mathbb{Z}^{n\times k}$
$X := A^{-1}Rv$
for $i = 1..k$
[3. Tri. denominator]
$X_i := i$th column of X
$x := T_{i-1}\ldots T_2 T_1 X_i$
$d_i := \mathrm{denom}(x)$
$T_i :=$ hcol$(d_i x \bmod d_i, d_i)$
[4. Extraction]
$d := d d_1 d_2 \ldots d_k$
$A := A T_1^{-1} T_2^{-1} \ldots T_k^{-1}$
[5. Verification]
$R :=$ hor(A)
if $R = 0$
return d

Figure 2: Overview of determinant algorithm.

Figure 2 gives a high-level overview of our determinant algorithm. The first phase uses a highly-optimized implementation of p-adic linear system solving [3] to compute

$X = A^{-1}v$, a projection of the inverse A^{-1} for a random block of vectors $v \in \mathbb{Z}^{n\times k}$. The procedure described in the previous section computes minimal triangular denominators T_i such that $T_i T_{i-1} \cdots T_1$ is a minimal triangular denominator of the first i columns of X, $i = 1, 2, \ldots, k$. The factor $T_k \cdots T_2 T_1$ is next extracted from the input by updating A as $A := AT_1^{-1}T_2^{-1} \cdots T_k^{-1}$. Finally, to check the completeness of the determinant extracted thus far, a high-order residue R is computed by the method of double-plus-one lifting [12] (denoted `hor` in Figure 2). If this check fails, a new random block of vectors is chosen and the process repeats.

We note that the above algorithm is randomized in the Las Vegas sense: the output is always correct, but the number of iterations required to produce that output may vary. In the case of generic matrices, the computation of the high order residue (the "Verification" phase in Figure 2) serves only to verify the correctness of the result. Omitting this verification step, then, yields a faster Monte Carlo randomized algorithm: one that computes the determinant from a single projection with high probability.

4. EXTENSION TO HERMITE FORM

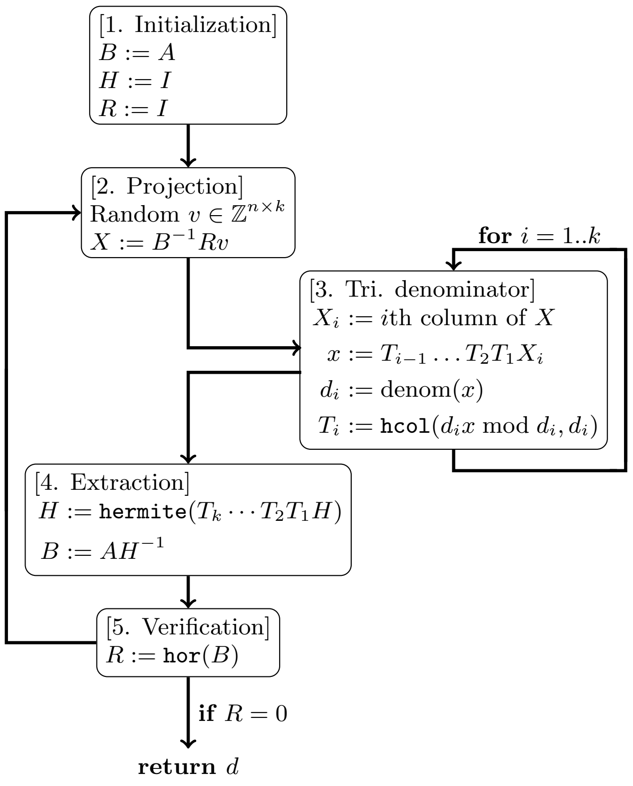

Figure 3: Overview of Hermite form algorithm.

A conceptually straightforward extension of the projection method allows the recovery of the entire Hermite form. Figure 3 gives an overview. The Hermite form algorithm differs from the preceding determinant algorithm only in the "extraction" phase.

In the determinant algorithm, the triangular denominators are repeatedly extracted from the same matrix in place (i.e., $A := AT_1^{-1} \ldots T_k^{-1}$) and then discarded. Here, each set of T_i is combined into a single work matrix H (i.e., $H := T_k \ldots T_1 H$) which eventually contains the Hermite form of A. This process may cause growth in the off-diagonal entries of H. A special Hermite normal form algorithm [16] (denoted `hermite` in Figure 3) for triangular matrices provides an efficient scheme for appropriately reducing the off-diagonal entries.

Additionally, H must be extracted from the original input matrix — not a work matrix — at each stage. That is, A is not updated in-place; rather, the subsequent projection and verification phases operate on $B := AH^{-1}$.

The additional cost required to combine the minimal triangular denominators and extract them from the input matrix complicates attempts to obtain a strong result for the asymptotic complexity of the algorithm. Even if there are very few invariant factors, the Hermite form may have many non-trivial columns in the worst case.

5. IMPLEMENTATION

Although the Hermite normal form algorithm of the previous section is not asymptotically optimal, empirical tests with a careful C implementation bear out its effectiveness in practice.

Our implementation relies on several existing, highly efficient libraries. Nonsingular system solving is provided by the Integer Matrix Library (IML) [3]. Additional routines not available elsewhere (algorithm `hcol`, for one) are implemented in terms of the integer arithmetic routines of the GNU Multi-Precision Arithmetic (GMP) library [6]. The algorithm and implementation presented in [12] gives a routine for computing the high-order residue.

The projection-based method for extracting the invariant structure is sensitive to the number of non-trivial invariant factors. A matrix with many non-trivial invariant factors requires the computation of many projections. However, as generic matrices are expected to have very few non-trivial invariant factors - and often only one - the practical performance of the algorithm in the common case is very good. Typically, only a single projection is required to extract the entirety of the invariant structure.

Yet, while the projection method performs most dramatically on generic matrices, the algorithm can also be effectively applied to matrices specifically constructed to have many non-trivial invariant factors and, in turn, highly non-trivial Hermite and Smith forms. This implementation is robust in its ability to handle all input matrices without making undue concessions to either the common or exceptional cases. No special manual tuning is used to better handle any particular case.

We discuss some specific implementation concerns below.

Projection size.

The column dimension k of the random $v \in \mathbb{Z}^{n\times k}$ used to compute the projection $x := A^{-1}v$ (cf. the "Projection" phase in figure 3) may be varied from one iteration to the next. Choosing a value for k, the size of the projection, is an inexact process driven mostly by empirical observation. A reality of the lifting-based linear system solving algorithms, like those in IML, is that the cost of initialization can meet or surpass the cost of the lifting steps themselves. Indeed, to cover a wide range of inputs, IML has been tuned to balance the cost of initialization with the cost of lifting. The relevant consequence here, then, is that the cost of computing a projection of multiple columns is negligibly more than the cost of working with a single column.

Preliminary observations suggest that an initial projection of eight columns works well. Eight columns are sufficient to capture all invariant factors in the case of a random matrix without being prohibitively more costly than working with a single column.

Subsequent iterations can use much larger projections as the largest invariant factors will have already been extracted and, consequently, the system can be solved at a much lower precision. The scheme used in this implementation is somewhat coarse, but effective. The second iteration uses a projection of $n/10$ columns; the third iteration uses a projection of n columns. Additionally, rather than a randomly chosen matrix, the third iteration uses the identity matrix. This guarantees that only three iterations are ever required and obviates a final high-order residue computation to certify the result.

It is perhaps possible to choose the size of projections adaptively, based on the size of the denominator of the previous projection (or, even better, based on the expected size of the next one). If the previous denominator is larger than expected, the next denominator may be relatively small and, thus, the next projection could be made larger without incurring much additional cost. An adaptive scheme of this sort would require quantifying, perhaps only in a heuristic sense, the expected size of the invariant factors extracted at each iteration.

Combining slices.

As each projection yields a portion of the Hermite form corresponding to only a few invariant factors, combining these "slices" involves operations on highly structured matrices. Generally, the non-zero elements of these matrices are confined to only a few columns. Thus, storing only the non-trivial columns immediately improves both the time and space requirements. Two operations — multiplication and reduction to Hermite form — are implemented for matrices in this packed representation. Both operations use existing algorithms modified to concern themselves with only the non-trivial columns: the former is based on classical matrix multiplication while the latter uses an algorithm for the Hermite form for triangular matrices [16].

Experimental results.

Two classes of matrices are used to illustrate the performance of the implementation at both extremes of the spectrum of input matrices.

Firstly, to test the implementation on the generic case, we use matrices with random 8-bit entries; these results are shown in Table 1. Each of the random matrices used in Table 1 had three or fewer non-trivial elements on the diagonal of the Hermite form. As expected, only a single projection was required in each case.

Following the example of Jäger and Wagner [7], we use the following class of matrices to test performance on inputs with many non-trivial invariant factors:

$$A_n = [a_{i,j}] \text{ with } a_{i,j} = (i-1)^{j-1} \bmod n \text{ for } 1 \leq i,j \leq n$$

If n is prime, A_n is nonsingular, typically has more than $n/2$ non-trivial invariant factors, and s_n is very large relative to n. For instance, A_{113} has 72 non-trivial invariant factors, the largest of which is 253 bits in length. In addition to having the desired structural properties, A_n can be quickly and straightforwardly constructed, allowing for consistent comparisons between implementations. Results for the A_n matrices are shown in Table 2.

The following tables summarize the experimental results. These timings were made on an "M1 Medium" Amazon EC2 instance with 3.75 GiB RAM and a 64-bit Intel Xeon E5-2650 at 2GHz. The software was compiled with GCC 4.6.3 and linked against IML 1.0.3, ATLAS 3.10.1, and GMP 5.1.1.

Sage 5.5 [15], compiled from source and run on the same machine, provides a point of comparison for our implementation. For matrices of the size considered here, Sage uses a modular algorithm [13] most effective in the random case. Our implementation compares favourably with Sage.

n	time (s)		n	time (s)	
	`iherm`	Sage		`iherm`	Sage
100	0.09	0.635	125	0.14	0.844
200	0.42	2.25	250	0.75	3.83
400	3.08	12.2	500	5.66	24.7
800	22.5	81.1	1000	42.0	152
1600	171	681	2000	348	1365
3200	1625		4000	3214	

Table 1: Time to compute Hermite form of random $n \times n$ matrix with 8-bit entries.

For random matrices (Table 1), the computation time grows roughly as $n^3 \log n$. That is, doubling the input dimension increases the cost by a factor of slightly less than nine. The fit is not perfect, however: smaller inputs slightly outperform expectations while larger inputs are slightly underperforming. For instance, the ratio between results for $n = 4000$ and $n = 2000$ is greater than nine, but is near seven between $n = 1000$ and $n = 500$. Smaller input matrices may be taking advantage of some beneficial machine-specific cache effects.

n	k	time (s)	
		`iherm`	Sage
101	56	0.52	1.13
211	118	2.91	19.4
401	266	18.6	531
809	503	118	
1601	1060	831	
n	k	time (s)	
		`iherm`	Sage
127	77	0.83	2.24
251	132	5.90	44.6
503	252	32.0	1520
1009	663	221	
2003	1041	1410	

Table 2: Time to compute Hermite form of A_n with k non-trivial invariant factors.

For inputs with many non-trivial invariant factors (Table 2), the empirical timings again grow roughly as $n^3 \log n$. Although the algorithm runs much faster overall on generic inputs, the rate of growth exhibited by the timings is the same for both types of inputs.

6. CONCLUSIONS AND FUTURE WORK

Although experiments demonstrate excellent performance in practice, we do not provide a worst-case cost analysis for our Hermite normal form algorithm. There are two essential roadblocks. Firstly, repeatedly combining the results of `hcol` may, in the worst-case, be costly; there is the potential for expressions swell, and while each minimal triangular denominator has few non-trivial columns in practice, this need not always be the case. Secondly, preliminary analysis suggests that in the worst case the entries in AH^{-1} can have n more bits compared to the entries in A. However, this growth has not been observed in practice, even for matrices with highly non-trivial Hermite forms.

The algorithm for Hermite normal form given here can be extended to one for Smith normal form. An efficient algorithm for finding the Smith normal form of a triangular input matrix is given in [16]; this algorithm can be directly applied to the result of our Hermite form computation.

7. REFERENCES

[1] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'99*, pages 197–204. ACM Press, New York, 1999.

[2] E. Bach and J. Shallit. *Algorithmic Number Theory*, volume 1 : Efficient Algorithms. MIT Press, 1996.

[3] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'05*, pages 92–99. ACM Press, New York, 2005.

[4] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.

[5] W. Eberly, M. Giesbrecht, and G. Villard. Computing the determinant and Smith form of an integer matrix. In *Proc. 31st Ann. IEEE Symp. Foundations of Computer Science*, pages 675–685, 2000.

[6] T. Granlund and the GMP development team. Gnu mp: The GNU multiple precision arithmetic library, 2011. Edition 5.0.2. `http://gmplib.org`.

[7] G. Jäger and C. Wagner. Efficient parallelizations of hermite and smith normal form algorithms. *Parallel Comput.*, 35(6):345–357, 2009.

[8] E. Kaltofen. On computing determinants of matrices without divisions. In P. S. Wang, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'92*, pages 342–349. ACM Press, New York, 1992.

[9] E. Kaltofen and G. Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *J. Computational Applied Math.*, 162(1):133–146, Jan. 2004. Special issue: Proceedings of the International Conference on Linear Algebra and Arithmetic 2001, held in Rabat, Morocco, 28–31 May 2001, S. El Hajji, N. Revol, P. Van Dooren (guest eds.).

[10] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3–4):91–130, 2004.

[11] T. Mulders and A. Storjohann. Diophantine linear system solving. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'99*, pages 281–288. ACM Press, New York, 1999.

[12] C. Pauderis and A. Storjohann. Deterministic unimodularity certification. In J. van der Hoeven and M. van Hoeij, editors, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'12*, pages 281–288. ACM Press, New York, 2012.

[13] C. Pernet and W. Stein. Fast computation of Hermite normal forms of integer matrices. *Journal of Number Theory*, 130(7), 2010.

[14] D. Saunders and Z. Wan. Smith normal form of dense integer matrices, fast algorithms into practice. In J. Gutierrez, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'04*, pages 274–281. ACM Press, New York, 2004.

[15] W. Stein et al. *Sage Mathematics Software (Version 5.5.0)*. The Sage Development Team, 2012. `http://www.sagemath.org`.

[16] A. Storjohann. Computing Hermite and Smith normal forms of triangular integer matrices. *Linear Algebra and its Applications*, 282:25–45, 1998.

[17] A. Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21(4):609–650, 2005. Festschrift for the 70th Birthday of Arnold Schönhage.

[18] Z. Wan. *Computing the Smith Forms of Integer Matrices and Solving Related Problems*. PhD thesis, University of Deleware, 2005.