Deterministic Unimodularity Certification

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

The asymptotically fastest algorithms for many linear algebra problems on integer matrices, including solving a system of linear equations and computing the determinant, use high-order lifting. Currently, high-order lifting requires the use of a randomized shifted number system to detect and avoid error-producing carries. By interleaving quadratic and linear lifting, we devise a new algorithm for high-order lifting that allows us to work in the usual symmetric range modulo p, thus avoiding randomization. As an application, we give a deterministic algorithm to assay if an $n \times n$ integer matrix A is unimodular. The cost of the algorithm is $O((\log n) n^{\omega} \mathsf{M}(\log n + \log ||A||))$ bit operations, where $||A||$ denotes the largest entry in absolute value, and $\mathsf{M}(t)$ is the cost of multiplying two integers bounded in bit length by t.

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

General Terms

Algorithms

Keywords

Integer matrix, unimodular matrix

1. INTRODUCTION

Classical linear algebra problems on an integer matrix $A \in \mathbb{Z}^{n\times n}$ include computing the determinant $\det A$ and a rational solution vector $A^{-1}b$. For computing $A^{-1}b$, the p-adic lifting algorithm of Dixon [4] already achieves an expected running time of $(n^3 \log ||A||)^{1+o(1)}$ bit operations, where $||A|| = \max |A_{ij}|$ denotes the maximum magnitude of entries in A. High-order lifting [16] allows the incorporation of matrix multiplication to reduce the expected running time to $(n^{\omega} \log ||A||)^{1+o(1)}$, where $2 \leq \omega \leq 3$ is the exponent of matrix multiplication [5, §5.10].

But for many problems the improvement in running time is more fundamental than reducing the exponent of n from 3 down to ω. For example, without the use of subcubic matrix multiplication, the previously fastest algorithm for computing $\det A$ uses an expected number of $(n^{3.2} \log ||A||)^{1+o(1)}$ bit operations and $(n^{2.2} \log ||A||)^{1+o(1)}$ bits of intermediate space [11]. The algorithm for $\det A$ based on high-order lifting [16, §13] uses an expected number of $(n^3 \log ||A||)^{1+o(1)}$ bit operations and is space-efficient: intermediate space requirements are bounded by $(n^2 \log ||A||)^{1+o(1)}$ bits. A more striking example is integrality certification: given a second matrix $C \in \mathbb{Z}^{n\times n}$, can all columns of C be generated as a $\mathbb{Z}$-linear combination of columns of A? This question is equivalent to determining if $A^{-1}C$ is a integer matrix. If $||C|| \in \Theta(||A||)$, then directly computing $A^{-1}C$ using a standard method such as quadratic lifting or homomorphic imaging and Chinese remaindering would require on the order of $(n^{\omega+1} \log ||A||)^{1+o(1)}$ bit operations and $\Omega(n^3 \log ||A||)$ bits of intermediate space. (Recall that $f(n) \in \Omega(g(n))$ precisely when $g(n) \in O(f(n))$.) High-order lifting can answer the integrality certification question with a space-efficient algorithm in a Las Vegas fashion using an expected number of $(n^{\omega} \log ||A||)^{1+o(1)}$ bit operations [16, §11].

Our main theoretical contribution is a new deterministic algorithm for high-order lifting that avoids the randomization that was required by the previous algorithm [16]. We will discuss our contribution and its implications in detail below. First let us recall what high-order lifting computes. Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular and let $X \in \mathbb{Z}$ be chosen (usually at random) to be relatively prime to $\det A$, with $\log X \in \Theta(\log n + \log ||A||)$. (Recall that $f(n) \in \Theta(g(n))$ precisely when $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.) In a nutshell, high-order lifting is used to efficiently compute a so-called residue $R \in \mathbb{Z}^{n\times n}$ such that

$$AB = I + RX^k$$

for $k \in O(n)$ a power of two and for $B \in \mathbb{Z}^{n\times n}$ an integer matrix satisfying $||B|| \in O(X^k)$. This bound for $||B||$ gives the bound $||R|| \in O(n||A||)$. Notice that B is necessarily congruent to $A^{-1} \bmod X^k$, and since $k \in \Theta(n)$, writing down B explicitly would require $\Omega(n^3(\log n + \log ||A||))$ bits of space. Instead of computing B explicitly, a sparse inverse expansion [16, §8] of B is computed:

$$A\,\overbrace{((\cdots(*(I+*X^2)+*X^4)(I+*X^4)+\cdots)+*X^k)}^{B} = I+RX^k.$$

Each $*$ is an $n \times n$ integer matrix with entries bounded in magnitude by $O(n||A||)$. Since $k \in \Theta(n)$, the total size of the sparse inverse expansion is $O((\log n)n^2(\log n + \log ||A||))$ bits. For the integrality certification problem mentioned above only the residue R is required. For solving $A^{-1}b$, the sparse inverse expansion is applied in $O(\log n)$ steps to the X-adic expansion of b.

High-order lifting requires a modulus X that is relatively prime to $\det A$. If not known *a priori*, X can be constructed as the power of a prime p that is randomly chosen in the range $2^{k-1} < p < 2^k$, where $k = 6 + \ln\ln(n^{n/2}||A||^n)$. Once a suitable X is in hand, the previously known algorithm for high-order lifting requires further randomization to implement the shifted number system [16, §3-4] to avoid and detect error-producing carries. For example, suppose $X = 10$, and consider perturbing the X-adic expansion of an integer a with a small perturbation γ:

$$\overbrace{5X^4+9X^3+9X^2+8X+9}^{a=59989}+\overbrace{9X+9}^{\gamma=99}=\overbrace{6X^4+8X+8}^{a+\gamma=60088}.$$

If we are interested in only the leading coefficient of the X-adic expansion of a we prefer to work with an approximation of a (i.e., only the first few leading coefficients), but as shown above even a small perturbation can affect the leading coefficient. The shifted number system avoids this problem with high probability by randomly shifting the class of representatives modulo X; instead of the positive range $[0, 9]$ we might choose $[-2, 7]$. In this shifted number system we have

$$\overbrace{6X^4-X-1}^{a=59989}+\overbrace{X^2-1}^{\gamma=99}=\overbrace{6X^4+X^2-X-2}^{a+\gamma},$$

so the leading coefficient of a has not been affected by the perturbation.

The deterministic algorithm for high-order lifting we propose here avoids the following complications of using the shifted number system.

- The modulus X must be chosen large enough to afford sufficiently many choices for the shift to ensure the probability of failure is at most 1/2. That is, the lower bound on the magnitude of X depends not only on the minimum precision required for the lifting process, but also on the number of computations performed.
- After each computation the guard coefficients in the shifted X-adic expansion must be obtained explicitly to detect error-producing carry propagation.

Being able to work with a modulus X that is as small as possible is crucial for an efficient implementation of high-order lifting. In particular, each iteration of the lifting doubles the precision of the output, but doubling the precision of X makes each lifting step at least twice as expensive while decreasing the number of iterations by only one. For our deterministic algorithm, we prove that the modulus X can be any integer relatively prime to $\det A$ that satisfies $X \geq \max(10000, 3.61n^2||A||)$, independent of the number of computations performed. For comparision, the previous randomized algorithm for unimodularity ceritification [16, §7] requires $X \in \Omega(n^4(\log n)||A||)$.

Our implementation of the high-order lifting algorithm (see §5) can choose X to be the product of "word-size" primes, thus keeping intermediate quantities in a residue number system and avoiding conversions that would be required after each step of the computation to check guard coefficients. This allows an efficient reduction to level 3 BLAS.

The rest of this paper is organized as follows. In §2 we offer a detailed study of both linear and quadratic X-adic lifting. Our deterministic high-order lifting algorithm is given in §3. As an application, we give in §4 a deterministic algorithm to certify if an integer matrix has determinant ± 1. In §5 we report on a prototype implementation of the new high-order lifting algorithm. Finally, §6 concludes.

Cost functions.

Cost estimates are given in terms of a multiplication time $\mathsf{M}(t)$ for integers: two integers bounded in magnitude by 2^t can be multiplied using at most $\mathsf{M}(t)$ bit operations. We use ω as the exponent of matrix multiplication: two $n \times n$ matrices over a commutative ring can be multiplied using at most $O(n^\omega)$ ring operations. For more details about integer and matrix multiplication we refer to the textbook by von zur Gathen and Gerhard [5].

In this paper we place no restrictions on ω. In particular, our results remain valid even if the exponent of matrix multiplication is 2.

Preliminaries.

Let $X > 2$ be an integer. For any rational number a that has denominator relatively prime to X, we let $\mathrm{Rem}(a, X)$ denote the unique integer in the usual "symmetric range" modulo X, that is, $\mathrm{Rem}(a, X) \equiv a \bmod X$ and

$$\mathrm{Rem}(a, X) \in \left[-\left\lfloor\frac{X-1}{2}\right\rfloor, \left\lfloor\frac{X}{2}\right\rfloor\right].$$

The next two lemmas follow directly from the above definition.

Lemma 1. $|\mathrm{Rem}(*, X)|/X \leq 1/2$.

Lemma 2. *If* $a \in \mathbb{Z}$ *satisfies* $|a| < X/2$ *then* $\mathrm{Rem}(a, X) = a$.

For a matrix A filled with rational numbers having denominator relatively prime to X, we write $\mathrm{Rem}(A, X)$ for the matrix obtained from A by applying $\mathrm{Rem}(\cdot, X)$ elementwise to each entry.

2. LIFTING

Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular. Given an $X \in \mathbb{Z}_{>2}$ that is relatively prime to $\det A$ (denoted by $X \perp \det A$), X-adic lifting can be used to compute $\mathrm{Rem}(A^{-1}, X^i)$ up to some precision i. We first recall the standard linear and quadratic lifting algorithms in §2.1 and §2.2, respectively, and in §2.3 we give a variation of quadratic lifting that yields a straight line formula

$$B(I+RX)(I+R^2X^2)(I+R^4X^4)\cdots(I+R^{2^{k-1}}X^{2^{k-1}})$$

that is congruent to $A^{-1} \bmod X^{2^k}$.

2.1 Linear lifting

For convenience assume X is odd. This assumption is not required, but simplifies the presentation slightly because for odd X we have $\mathrm{Rem}(A^{-1}, X^k) = C_0 + C_1X + \cdots + C_{k-1}X^{k-1}$ where each C_i satisfies $C_i = \mathrm{Rem}(C_i, X)$.

Linear X-adic lifting is based on the identity

$$A^{-1} = \overbrace{C_0 + C_1X + \cdots + C_{i-1}X^{i-1}}^{\mathrm{Rem}(A^{-1}, X^i)} + A^{-1}R_iX^i, \qquad (1)$$

where the residue R_i is equal to

$$R_i = (1/X^i)(I - A\,\mathrm{Rem}(A^{-1}, X^i)). \qquad (2)$$

If R_i is known, the next coefficient C_i in the X-adic expansion of A^{-1} can be computed using

$$C_i = \mathrm{Rem}(C_0R_i, X),$$

and the next residue using

$$R_{i+1} = (1/X)(R_i - AC_i). \qquad (3)$$

This gives the following standard algorithm to compute the X-adic expansion of A^{-1}.

```
$C_0 := \mathrm{Rem}(A^{-1}, X)$;
$R_1 := (1/X)(I - AC_0)$;
for $i = 1$ to $k - 1$ do
    $C_i := \mathrm{Rem}(C_0R_i, X)$;
    $R_{i+1} := (1/X)(R_i - AC_i)$
od
```

The essence of linear X-adic lifting is that C_i and R_{i+1} can be computed using only A, R_i and C_0: the other coefficients of the X-adic expansion of A^{-1} are not required. The following theorem captures this essential idea of linear X-adic lifting.

THEOREM 3. *Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular and $X \in \mathbb{Z}_{>2}$ be relatively prime to* $\det A$. *If $B, R \in \mathbb{Z}^{n\times n}$ satisfy*

- $A^{-1} = B + A^{-1}RX^k$

for some $k > 0$, then for any $M \in \mathbb{Z}^{n\times n}$ such that $M \equiv A^{-1}R \bmod X^l$ for some $l > 0$, we have

- $A^{-1} = B + MX^k + A^{-1}R'X^{k+l}$,

where $R' := (1/X^l)(R - AM)$. In particular, we can choose $l = 1$ and $M := \mathrm{Rem}(A^{-1}R, X)$.

2.2 Quadratic lifting

Identically to linear lifting, quadratic lifting, also known as algebraic Newton iteration, is based on the identity

$$A^{-1} = \mathrm{Rem}(A^{-1}, X^{2^i}) + A^{-1}RX^{2^i}, \qquad (4)$$

where the residue R is equal to

$$R = (1/X^{2^i})(I - A\,\mathrm{Rem}(A^{-1}, X^{2^i})). \qquad (5)$$

Note that equations (4) and (5) are exactly equations (1) and (2) but with i replaced by 2^i. In linear lifting we used the local inverse $C_0 = \mathrm{Rem}(A^{-1}, X)$ to increase the precision of the inverse by one at each step. In quadratic lifting, the precision of the inverse is doubled using the observation

$$A^{-1} \equiv \mathrm{Rem}(A^{-1}, X^{2^i})(I + RX^{2^i}) \bmod X^{2^{i+1}}.$$

The standard algorithm [2, Algorithm 3.1] for computing $\mathrm{Rem}(A^{-1}, X^{2^k})$ is shown below.

```
$B := \mathrm{Rem}(A^{-1}, X)$;
for $i = 0$ to $k - 1$ do
    $R := (1/X^{2^i})(I - AB)$;
    $B := \mathrm{Rem}(B(I + RX^{2^i}), X^{2^{i+1}})$
od
```

2.3 Quadratic lifting as a straight line formula

Now suppose we have $\mathrm{Rem}(A^{-1}, X^{2^i})$ only modulo X^{2^i}, say

$$B = \mathrm{Rem}(A^{-1}, X^{2^i}) + CX^{2^i}$$

for some matrix $C \in \mathbb{Z}^{n\times n}$. If we compute the residue $\bar{R}$ from B we obtain

$$\begin{aligned} \bar{R} &= (1/X^{2^i})(I - AB) \\ &= R - AC, \end{aligned}$$

where R is the residue computed from $\mathrm{Rem}(A^{-1}, X^{2^i})$ as shown in (5). However, similar to (4),

$$A^{-1} = B + A^{-1}\bar{R}X^{2^i} \qquad (6)$$

still holds. Although B contains the extra term CX^k, this is cancelled out in (6) by the negation of the same term appearing in $A^{-1}\bar{R}X^k$. The above consideration leads to the following code fragment to compute $B_i \equiv \mathrm{Rem}(A^{-1}, X^{2^i}) \bmod X^{2^i}$ for $i = 0, 1, \ldots, k$.

```
$B_0 := \mathrm{Rem}(A^{-1}, X)$;
for $i = 0$ to $k - 1$ do
    $R_i := (1/X^{2^i})(I - AB_i)$;
    $B_{i+1} := B_i(I + R_iX^{2^i})$
od
```

The difference between the above recipe compared to the standard presentation of Newton iteration is that the computation of the B_i for $i \geq 1$ does not include a modulo operation to ensure that $B_i = \mathrm{Rem}(A^{-1}, X^{2^i})$. The benefit of avoiding the modulo operation is that the computation of R_i for $i \geq 1$ can be considerably simplified by using the fact that $R_i = R_{i-1}^2$ for $i = 1, 2, \ldots, k - 1$. Indeed,

$$\begin{aligned} R_i &= (1/X^{2^i})(I - AB_i) \\ &= (1/X^{2^i})(I - AB_{i-1}(I + R_{i-1}X^{2^{i-1}})) \\ &= (1/X^{2^i})(I - (I - R_{i-1}X^{2^{i-1}})(I + R_{i-1}X^{2^{i-1}})) \\ &= R_{i-1}^2. \end{aligned}$$

This gives the following simplified recipe.

```
$B_0 := \mathrm{Rem}(A^{-1}, X)$;
$R_0 := (1/X)(I - AB_0)$;
for $i = 1$ to $k - 1$ do
    $R_i := R_{i-1}^2$
od
```

Once the residues $R_1, R_2, \ldots, R_{k-1}$ have been computed by the simplified recipe, the $B_k \equiv \mathrm{Rem}(A^{-1}, X^{2^k}) \bmod X^{2^k}$ from the original recipe can be expressed as a straight line formula

$$B_k = B_0(I+R_0X)(I+R_1X^2)(I+R_2X^4)\cdots(I+R_{k-1}X^{2^{k-1}}).$$

Because the B_i are computed without a modulo operation we will have

$$B_i = \mathrm{Rem}(A^{-1}, X^{2^i}) + C_iX^{2^i}$$

for some overflow term C_i. For B_0 we have $C_0 = 0$ and hence $||R_0|| \le n||A||$. But because $R_{i+1} = R_i^2$ the bound $||C_i||$ and $||R_i||$ will grow prohibitively large as i increases. For example, if $X = 1000$, then

$$\mathrm{Rem}(777^{-1}, X^8) = -1287001287001287001287,$$

but the straight line encoding for a formula congruent to 777^{-1} modulo X^8 produced by the above loop is

$$\begin{aligned}&\overbrace{-287}^{B_0}(1+\overbrace{223}^{R_0}X)(1+\overbrace{49729}^{R_1}X^2)(1+\overbrace{2472973441}^{R_2}X^4)\\ &\quad= -787078203332224000128700128700128700128 7\\ &\quad\equiv -1287001287001287001287 \bmod X^8\end{aligned}$$

We end this section with the following theorem which captures the essential idea of quadratic X-adic lifting.

THEOREM 4. *Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular and $X \in \mathbb{Z}_{>2}$ be relatively prime to* $\det A$. *For any $B, R \in \mathbb{Z}^{n\times n}$ that satisfy*

- $A^{-1} = B + A^{-1}RX,$

we have

- $A^{-1} = B(I+RX) + A^{-1}R^2X^2.$

PROOF. Multiplying both sides of the equation $A^{-1} = B + A^{-1}RX$ on the right by RX gives $A^{-1}RX = BRX + A^{-1}R^2X^2$, and thus $A^{-1} = B + A^{-1}RX = B + BRX + A^{-1}R^2X^2$. $\square$

3. DOUBLE-PLUS-ONE LIFTING

Define $X_i = X^{2^{i+1}-1}$. Then $X_{i+1} = X_i^2X$ for all $i \ge 0$. We will interleave the processes of quadratic and linear X-adic lifting. At step $i = 0, 1, 2. \dots$ we will "double the precision plus one," that is, compute a matrix B_i that is congruent to $A^{-1} \bmod X_i$ where

$$\begin{aligned}X_0 &= X\\ X_1 &= X_0^2X = X^{2^2-1}\\ X_2 &= X_1^2X = X^{2^3-1}\\ &\vdots\end{aligned}$$

The expansion we will compute has the following form.

$$A^{-1} = (\cdots((B_0(I+R_0X_0)+M_0X_0^2)(I+R_1X_1)+M_1X_1^2)+\cdots)$$

More precisely, we initialize $B_0 := \mathrm{Rem}(A^{-1}, X_0 = X)$ and compute, for $i = 0, 1, 2, \dots$ in succession, the following sparse inverse expansions:

$$\begin{aligned}B_1 &= B_0(I+R_0X_0) + M_0X_0^2 \equiv A^{-1} \bmod X_1(= X_0^2X)\\ B_2 &= B_1(I+R_1X_1) + M_1X_1^2 \equiv A^{-1} \bmod X_2(= X_1^2X)\\ &\vdots\end{aligned} \qquad (7)$$

Each multiplicative factor $(I + R_iX_i)$ encodes a step of quadratic lifting, so that $B_i(I + R_iX_i) \equiv A^{-1} \bmod X_{i-1}^2$, and each additive term $M_iX_i^2$ encodes a single step of linear X-adic lifting so that $B_i(I + R_iX_i) + M_iX_i^2 \equiv A^{-1} \bmod X_{i-1}^2X$.

The approach is best illustrated with a concrete example. Let $X = 1000$ and $A = 777$, and consider the computation of 777^{-1}. To begin we initialize $B_0 = \mathrm{Rem}(A^{-1}, X)$ and $R_0 = (1/X)(I - AB_0)$ to obtain

$$\overbrace{777^{-1}}^{A^{-1}} \equiv \overbrace{-287}^{B_0}(1+\overbrace{223}^{R_0}X) \bmod X^2.$$

But

$$\begin{aligned}\overbrace{-287}^{B_0}(1+\overbrace{223}^{R_0}X) &= -65001287\\ &= \overbrace{\quad 1287 \quad}^{\mathrm{Rem}(777^{-1},X^2)} \quad -64\,X^2,\end{aligned}$$

so our straight line formula $-287(1 + 223X)$ for 777^{-1} mod X^2 contains the overflow term $-64\,X^2$. Nonetheless, by Theorem 4 we have

$$\overbrace{777^{-1}}^{A^{-1}} = \overbrace{-287}^{B_0}(1+\overbrace{223}^{R_0}X) + \overbrace{777^{-1}}^{A^{-1}}\overbrace{223^2}^{R_0^2}X^2,$$

so we can proceed with one step of linear X-adic lifting: compute $M_0 = \mathrm{Rem}(A^{-1}R_0^2, X)$ and $R_{i+1} = (1/X)(R_0^2 - AM_0)$ yielding

$$\overbrace{777^{-1}}^{A^{-1}} \equiv \underbrace{\overbrace{-287}^{B_0}(1+\overbrace{223}^{R_0}X) + \overbrace{(-223)}^{M_0}X^2}_{B_1} + \overbrace{777^{-1}}^{A^{-1}}\overbrace{223}^{R_{i+1}}X^2X.$$

Now we have a straight line formula B_1 for 777^{-1} mod X^2X without any overflow:

$$\begin{aligned}B_i &= -287(1+223X) - 223X^2\\ &= -287001287\\ &= \mathrm{Rem}(777^{-1}, X^2X).\end{aligned}$$

In general, the expression B_i may have some overflow, but with a judicious choice of X (based on $||A||$ and n) we can ensure the overflow will be very small. Algorithm `DoublePlusOneLift` is shown in Figure 1.

```
DoublePlusOneLift($A, X, n, k$)
**Input:** $A \in \mathbb{Z}^{n\times n}$, $X \in \mathbb{Z}$ with $X \perp \det A$, $k \in \mathbb{Z}_{>0}$.
**Output:** $B_0, R_0 \dots, R_k, M_0, \dots, M_k \in \mathbb{Z}^{n\times n}$ as in (7).
**Condition:** $X \ge \max(10000, 3.61n^2||A||)$.
$B_0 := \mathrm{Rem}(A^{-1}, X)$;
$R_0 := (1/X)(I - AB_0)$;
**for** $i = 0$ to $k-1$ **do**
    $\bar{R} := R_i^2$;
    $M_i := \mathrm{Rem}(B_0\bar{R}, X)$;
    $R_{i+1} := (1/X)(\bar{R} - AM_i)$
**od**;
**return** $B_0, R_0, R_1, \dots, R_k, M_0, M_1, \dots, M_k$
```

Figure 1: Algorithm `DoublePlusOneLift`

THEOREM 5. *Algorithm* `DoublePlusOneLift` *is correct. Moreover, for all i, $0 \le i \le k$, the output satisfies $||R_i|| < 0.6001n||A||$ and $||B_i|| < 0.6X_i$.*

PROOF. We prove by induction on i that at the start of each loop iteration we have

$$A^{-1} = B_i + A^{-1}R_iX_i, \tag{8}$$

and

$$||B_i|| < 0.6X_i, \tag{9}$$

where $B_i = B_{i-1}(I + R_{i-1}X_{i-1}) + M_{i-1}X_{i-1}^2$. For the base case $i = 0$, Theorem 3 shows (8) and Lemma 1 shows (9). Now assume that (9) and (8) hold for some i, $i \ge 0$. Theorem 4 gives

$$A^{-1} = B_i(I + R_iX_i) + A^{-1}R_i^2X_i^2 \tag{10}$$

and Theorem 3 gives

$$A^{-1} = B_i(I + R_iX_i) + M_iX_i^2 + A^{-1}R_{i+1}X_{i+1},$$

which shows that (8) is satisfied for $i + 1$. From (8) we have $R_i = (1/X_i)(I - AB_i)$ which gives the bound $||R_i|| < (1/X_i)+0.6n||A|| \le 0.6001n||A||$. This bound for $||R_i||$ gives

$$\begin{aligned} ||B_i(I + R_iX_i)|| &\le ||B_i|| + n||B_i||||R_i||X_i \\ &< 0.6X_i + n(0.6X_i)(0.6001n||A||)X_i \\ &= (0.6/X_i + 0.36006n^2||A||)X_i^2 \\ &\le 0.36012n^2||A||X_i^2. \end{aligned} \tag{11}$$

Finally, note that

$$\begin{aligned} ||B_{i+1}||/X_{i+1} &= ||B_i + B_iR_iX_i + M_iX_i^2||/(X_i^2X) \\ &< 0.36012n^2||A||/X + 1/2, \end{aligned}$$

which is < 0.6 using $X \ge 3.61n^2||A||$. This shows that (9) holds for $i + 1$. □

We now turn our attention to bounding the running time. The only subtlety is the computation of $\mathrm{Rem}(A^{-1}, X)$, which requires a recursive reduction to matrix multiplication and the inversion of some integers modulo X. Let $\mathsf{B}(t)$ denote the cost of the extended Euclidean problem with two integers bounded in magnitude by 2^t. We can take $\mathsf{B}(t) \in O(\mathsf{M}(t) \log t)$ (see for example [5, §15]). Then $\mathrm{Rem}(A^{-1}, X)$ can be computed in $O(n^\omega(\log n)\mathsf{M}(X) + n^2(\log n)\mathsf{B}(X))$ bit operations by using a fast algorithm for unimodular triangularization [9] (see for example [16, §2]).

COROLLARY 6. *If $k \in O(\log n)$ and $\log X \in O(\log n + \log ||A||)$ then the running time of algorithm* `DoublePlusOneLift` *is bounded by $O((\log n)n^\omega\, \mathsf{M}(\log n + \log ||A||) + n^2(\log n)\mathsf{B}(\log n + \log ||A||))$ bit operations.*

The following corollary will be useful in the next section. We first sketch the main idea of the proof. If A is unimodular then we must have $B_{k-1}(I + R_{k-1}X_{k-1}) = A^{-1} + EX_{k-1}^2$ for some integer matrix E. Now, if k is large enough, so that X_{k-1}^2 is large enough compared to $||A^{-1}||$, then the magnitude bounds established in the proof of Theorem 5 imply that $||E|| < X/2$. In the last iteration of the loop we have $\bar{R} = AE$, and because $||E|| < X/2$ is small enough we have $M_{k-1} = \mathrm{Rem}(E, X) = E$, and hence R_k will be zero.

COROLLARY 7. *If k is chosen large enough to satisfy*

$$X_{k-1}^2 \ge (n^{(n-1)/2}||A||^{n-1})/(n^2||A||),$$

then A is unimodular if and only if R_k is the zero matrix.

PROOF. Assume A is unimodular. Then A^{-1} is integral and by Hadamard's bound we have

$$\begin{aligned} ||A^{-1}|| &\le (n-1)^{(n-1)/2}||A||^{n-1} \\ &\le (n^2||A||)X_{k-1}^2. \end{aligned}$$

Since $B_{k-1}(I + R_{k-1}X_{k-1})$ is congruent to A^{-1} mod X_{k-1}^2, we have

$$A^{-1} = B_{k-1}(I + R_{k-1}X_{k-1}) + EX_{k-1}^2 \tag{12}$$

for some E. Solving (12) for E, and then using (11) and (12), gives

$$\begin{aligned} ||E|| &= ||A^{-1} - B_{k-1}(I + R_{k-1}X_{k-1})||/X_{k-1}^2 \\ &\le ||A^{-1}||/X_{k-1}^2 + ||B_{k-1}(I + R_{k-1})|| \\ &\le n^2||A|| + 0.36012n^2||A|| \\ &< X/2, \end{aligned}$$

where the last inequality follows from the assumption $X \ge 3.61n^2||A||$. Let $\bar{R} = R_{k-1}^2$. Then considering equation (10) with $i = k - 1$ and (12) gives

$$A^{-1}\bar{R}X_{k-1}^2 = EX_{k-1}^2.$$

Now consider the step of linear X-adic lifting during the last loop iteration in the algorithm. The algorithm computes $M_{k-1} = \mathrm{Rem}(A^{-1}\bar{R}, X) = \mathrm{Rem}(E, X) = E$, where the last equality follows from $||E|| < X/2$. Finally, since $M_{k-1} = E$ we have $\bar{R} - AM_{k-1} = 0$. □

4. DETERMINISTIC UNIMODULARITY CERTIFICATION

Algorithm `UniCert`, shown in Figure 2, modifies `DoublePlusOneLift` from the previous section to deterministically assay if a given matrix A is unimodular: instead of being given as input, the modulus X is chosen to be a power of 2; to save space, the $O(\log n)$ matrices comprising the sparse inverse expansion formula are not saved during the computation; and the loop includes a check for early termination in case the *a priori* bound for $||A^{-1}||$ is pessimistic.

THEOREM 8. *Algorithm* `UniCert` *is correct. The cost of the algorithm is $O((\log n)n^\omega\, \mathsf{M}(\log n + \log ||A||))$ bit operations and the intermediate space requirement is bounded by $O(n^2(\log n + \log ||A||))$ bits.*

PROOF. The running time bound follows from Corollary 6 by noting that B_0 can be computed in $O(n^\omega(\log n)+n^\omega \mathsf{M}(e))$ bit operations by first computing $\mathrm{Rem}(A^{-1}, 2)$ and then doubling the precision up to $2^{\lceil \log_2 e \rceil}$ using algebraic Newton iteration.

We now prove correctness. Note that the algorithm computes the same quantities as algorithm `DoublePlusOneLift`, so let M_i and R_{i+1} be the matrices M and R computed in iteration i of the loop. At the end of loop iteration i the algorithm has computed a sparse inverse formula B_{i+1} such that $A^{-1} = B_{i+1} + A^{-1}R_{i+1}$. If R_{i+1} is zero then A^{-1} is evidently integral; this shows that the algorithm returns "Yes" only if A is unimodular.

By definition, $X_{k-1} = X^{2^k-1}$ and hence $X_{k-1}^2 = X^{2^{k+1}-2}$. By the choice of k in the algorithm, the condition on X_{k-1}^2 stipulated by Corollary 7 holds, and if A is unimodular the algorithm will return "Yes" before completing loop iteration $i = k - 1$. □

```
UniCert($A, n$)
Input: $A \in \mathbb{Z}^{n\times n}$.
Output: "Yes" if $A$ is unimodular, "No" otherwise.
Let $X = 2^e$ with $e$ minimal such that
        $X \geq \max(10000, 3.61n^2||A||)$.
Let $k \in \mathbb{Z}$ be minimal such that
        $X^{2^{k+1}-2} \geq (n^{(n-1)/2}||A||^{n-1})/(n^2||A||)$.
if $\det \mathrm{Rem}(A, 2) = 0$ then return "No" fi;
$B_0 := \mathrm{Rem}(A^{-1}, X)$;
$R := (1/X)(I - AB_0)$;
for $i = 0$ to $k - 1$ do
    $\bar{R} := R^2$;
    $M := \mathrm{Rem}(B_0\bar{R}, X)$;
    $R := (1/X)(\bar{R} - AM)$;
    if $R$ is the zero matrix then return "Yes" fi
od;
return "No"
```

Figure 2: Algorithm UniCert

5. EMPIRICAL RESULTS

The following describes a prototype implementation of `DoublePlusOneLift` in terms of the level 3 BLAS. As presented, `DoublePlusOneLift` requires X and k (respectively, the base of the X-adic lifting and the number of lifts) as input. For purposes of this implementation, X and k were selected to be sufficiently large as required by Theorem 5 and Corollary 7.

The Basic Linear Algebra Subprograms (BLAS) are a ubiquitous interface providing, as the name suggests, low-level linear algebra routines: scalar/vector, matrix/vector, and matrix/matrix multiplication. Our implementation relies primarily on the level 3 BLAS routines provided by the Automatically Tuned Linear Algebra Software (ATLAS) library [18]: a widely used, portable, highly optimized implementation of the BLAS. In addition, we make limited use of the Integer Matrix Library (IML) [3] and the GNU Multi-Precision Arithmetic (GMP) library [7] for matrix inversion over a finite field and arbitrary precision integer arithmetic respectively.

ATLAS operates on matrices of double-precision floating-point numbers. To allow our algorithm to work with matrices of arbitrarily large integers despite this limitation, we employ a standard modular scheme: operations on matrices over the integers are instead performed on their residues modulo multiple suitably-sized primes. Using the bound from Theorem 5, we choose moduli $q_1, q_2, \ldots q_m$ with $Y = q_1q_2\ldots q_m$ and $Y \geq 1.2002n||A||$. Each prime q_i is chosen small enough so that all entries in a matrix product can be represented exactly by the 53 bits of a double's mantissa. That is, for matrices of dimension n, q_i must satisfy $n(q_i - 1)^2 \leq 2^{53} - 1$.

The X-adic lifting steps of the algorithm also require some quantities be computed modulo X. Again, this quantity may also be too large to allow the BLAS routines to be applied directly. We use an additional set of prime moduli $p_1, p_2, \ldots p_l$ with $X = p_1p_2\ldots p_l$ and $X \geq 3.61n^2||A||$.

Working over two coprime bases requires converting between residues in the two bases. The following code snippet illustrates this.

```
for $i = 0$ to $k - 1$ do
    $M_i := \mathrm{Rem}(B_0 \,\mathrm{Rem}(R_i, X)^2, X)$
    $R_{i+1} := \mathrm{Rem}(X^{-1}(R_i^2 - A\,\mathrm{Rem}(M_i, Y)), Y)$
od;
```

Note that M_i is computed with respect to the X-basis, but, in the next line, is used in a computation in the Y-basis.

The basis extension technique of Shenoy and Kumaresan [15] can be used to compute such conversions. Briefly, this technique extends standard Chinese Remainder Theorem-based reconstruction by requiring a redundant modulus and corresponding residue be maintained. Residues with respect to basis Y can then be reconstructed exactly, rather than modulo Y as in standard methods. Basis extension, then, can be accomplished by performing this reconstruction modulo each prime in the target basis X.

The number of moduli in bases X and Y are denoted l and m, respectively. Computing a single residue in basis X from a complete set of residues in basis Y requires $\Theta(l)$ operations. So, converting between bases requires $\Theta(lm)$ operations. Moreover, both l and m are bounded by $O(\log||A|| + \log n)$. The cost of basis conversion at each iteration then is $O(n^2(\log||A|| + \log n)$ and is dominated by the cost of the matrix multiplications. Empirically, profiling the implementation confirms that the cost of basis extension is, in fact, negligible.

Timings.

Tables 1 and 2 summarize our experimental results. These timings were made on an Intel 1.3 GHz Itanium2 with 192 GB RAM running GNU/Linux 2.4.21. The software was compiled with gcc 4.1.2 and linked against IML 1.0.3, ATLAS 3.6.0, and GMP 4.1.3. Tests were performed on randomly generated input matrices of two types: those with single decimal digit entries and those with 100 decimal digit entries.

Dimension	Time	
1000	57 s	
2000	454 s	(≈ 7.6 minutes)
4000	3756 s	(≈ 62.6 minutes)
8000	41120 s	(≈ 11.4 hours)

Table 1: Small entries: 1 decimal digit entries

Dimension	Time	
200	5 s	
400	33 s	
1000	472 s	(≈ 8 minutes)
2000	4336 s	(≈ 1.2 hours)

Table 2: Large entries: 100 decimal digit entries

The published timings[1] for linear system solving with IML were performed on a very similar machine and provide a point of comparison. Using IML, solving a linear system

[1]`http://www.cs.uwaterloo.ca/~astorjoh/iml.html`

of dimension 2000 with 100-digit entries required about 1.3 hours. Here, computing the sparse inverse expansion with input of the same size required approximately the same amount of time. IML timings in P. Giorgi's dissertation [6] are also comparable: for input of dimension 2000 with 30 digit entries, solving a linear system and computing a sparse inverse expansion both require about thirty minutes. This result suggests (at least for input of about this size) that high-order lifting has some potential as a practical approach to the problems of, for instance, determinant calculation or integrality certification.

Dimension	Input size (MB)	Peak usage (MB)
200	2.56	51.52
400	10.24	212
1000	64	1360
2000	256	5600

Table 3: Memory usage with 100 **decimal digit input entries**

Dimension	Input size (MB)	Peak usage (MB)
1000	24	208
2000	96	832
4000	384	3332
8000	1536	13312

Table 4: Memory usage with 1 **decimal digit input entries**

Tables 3 and 4 show the memory usage of selected computations. The majority of the extra space is used to store intermediate computations in the residue number systems. That is, the extra space required is proportional to the number of elements in the two coprime bases and, consequently, exceeds the size of the input by only a logarithmic factor.

6. CONCLUSIONS AND FUTURE WORK

Given a nonsingular $A \in \mathbb{Z}^{n\times n}$ together with an $X \in \mathbb{Z}_{>2}$ such that $X \perp \det A$, we describe an algorithm to compute a high-order residue $R \in \mathbb{Z}^{n\times n}$ such that $BA = I + RX^k$, together with a sparse inverse expansion for the matrix $B \equiv A^{-1} \bmod X^k$. The algorithm is deterministic compared to the previously known approach which relied on a randomized shifted number system [16, §3-4].

The algorithm reduces the computation to matrix multiplications, the number of which can be exactly quantified *a priori*. In addition to X, the implementation makes use of a second modulus $Y \in \mathbb{Z}_{>2}$ such that $Y \perp X$. The two coprime moduli X and Y should satisfy $X \geq 3.61n^2||A|| \geq Y \geq 1.2002n||A||$. Initialization requires one matrix inversion modulo X and one matrix multiplication modulo Y. Next, fewer than $\log_2 n$ iterations are performed, each iteration requiring at most two matrix multiplications modulo X plus two matrix multiplications modulo Y. The timings of a prototype implementation, which reduces the computation to level 3 BLAS by choosing X and Y to be the product of word-size primes, demonstrate the effectiveness of the approach.

Our work is motivated by the applications of high-order lifting to solving a wide variety of linear algebra problems. These include unimodularity certification (testing if $\det A = \pm 1$), integrality certification (testing if $A^{-1}C$ is integral for a given matrix C), and more difficult problems such as certifying the rank [17] or computing the determinant [16, §13].

In this paper we have specified in detail how to apply the high-order lifting algorithm `DoublePlusOneLift` to obtain a deterministic algorithm for unimodularity certification. Much work remains to investigate the application of `DoublePlusOneLift` — with a view towards minimizing the number of required "word-size" matrix multiplications — to obtain algorithms for some of the seemingly more difficult problems mentioned above. Practically fast implementations of algorithms for problems like computing normal forms of matrices will use a multi-faceted approach: numeric-symbolic solvers [13], hybrid methods [14], output sensitive techniques [10], relaxed lifting [1] and heuristic methods designed to exploit the structure of commonly occurring or generic cases [12]. We expect that the unimodularity and integrality certification using high-order lifting will be a key ingredient used to certify correctness in a Las Vegas fashion the output of algorithms that would otherwise be Monte Carlo.

One quibble with algorithm `DoublePlusOneLift` is that it produces a sparse inverse formula that requires $\Omega(\log n)$ as much space as required to write down the input matrix. For an input matrix with order 10000, this could be a factor of 10 blowup. However, for applications like unimodularity and integrality certification, only the final residue R such that $AB = I + RX^k$ is required. Moreover, for linear system solving, the components $(R_0, M_0), (R_1, M_1), \ldots$ of the sparse inverse expansion of B can be applied as they are computed, avoiding the need to keep them [16].

An interesting question is to devise a method which avoids the need to randomly find an X that is relatively prime to $\det A$. For polynomial matrices this can be done using triangular x-basis decompositions [8].

7. REFERENCES

[1] J. Berthomieu and R. Lebreton. Relaxed p-adic Hensel lifting for algebraic systems. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'12*. ACM Press, New York, 2012.

[2] D. Bini and V. Y. Pan. *Polynomial and Matrix Computations, Vol 1: Fundamental Algorithms.* Birkhauser, Boston, 1994.

[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] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2nd edition, 2003.

[6] P. Giorgi. *Arithmetic and algorithmic in exact linear algebra for the LinBox library.* PhD thesis, École normale supérieure de Lyon, LIP, Lyon, France, December 2004.

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