

On the complexity of inverting integer and polynomial matrices

Arne Storjohann

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

December 31, 2008

Abstract

An algorithm is presented that probabilistically computes the exact inverse of a nonsingular $n \times n$ integer matrix A using $O^\sim(n^3 \log \|A\| + n^3 \log \kappa(A))$ bit operations. Here, $\|A\| = \max_{ij} |A_{ij}|$ denotes the largest entry in absolute value, $\kappa(A) := \|A^{-1}\| \|A\|$ is the condition number of the input matrix and the soft-O notation O^\sim indicates some missing $\log n$ and $\log \log \|A\|$ factors. Thus, when the input matrix is well conditioned, with $\kappa(A)$ bounded by a polynomial function of $n \log \|A\|$, the inverse is computed using an expected number of $O^\sim(n^3 \log \|A\|)$ bit operations. For comparison, the exact inverse can require more than $n^3 \log_2 \|A\|$ bits to represent. A variation of the algorithm is presented for polynomial matrices. The inverse of any nonsingular $n \times n$ matrix whose entries are polynomials of degree d over a field can be computed using an expected number of $O^\sim(n^3 d)$ field operations. Similar to the integer case, this cost estimate matches up to logarithmic factors the number of field elements required to represent the inverse in the general case. Both algorithms are randomized of the Las Vegas type: failure may be reported with probability at most $1/2$, and if failure is not reported then the output is certified to be correct in the same running time bound.

1 Introduction

Let $A \in \mathbb{Z}^{n \times n}$ be nonsingular. We denote by $\|A\| := \max |A_{ij}|$ the maximum magnitude of entries in A , and by $\kappa(A) := \|A\| \|A^{-1}\|$ the condition number of the matrix with respect to the max norm. We describe an algorithm to compute the exact inverse of A using an expected number of $O^\sim(n^3 \log \|A\| + n^3 \log \kappa(A))$ bit operations. Thus, for a well conditioned A , with $\kappa(A)$ bounded by a polynomial function of $n \log \|A\|$, the running time is bounded by $O^\sim(n^3 \log \|A\|)$ bit operations. For comparison, the sum of the bitlengths of all entries in the inverse of a nonsingular $A \in \mathbb{Z}^{n \times n}$ may be more than $n^3 \log_2 \|A\|$ bits.

To illustrate the new algorithm and to clarify concepts as they arise we will use the following example: the matrix

$$A = \begin{bmatrix} -2 & -20 & -56 & 14 & -8 \\ -6 & 50 & -9 & -32 & 1 \\ 42 & 7 & -33 & -28 & -27 \\ 48 & 60 & -11 & 52 & 80 \\ 19 & 46 & 24 & 79 & -79 \end{bmatrix} \in \mathbb{Z}^{5 \times 5} \quad (1)$$

has inverse

$$A^{-1} = \begin{bmatrix} -\frac{26043493}{2917152684} & -\frac{5490137}{486192114} & \frac{4021909}{243096057} & \frac{6778619}{1458576342} & -\frac{272863}{1458576342} \\ \frac{9875}{729288171} & \frac{3475145}{243096057} & -\frac{548447}{243096057} & \frac{1762390}{729288171} & \frac{2477998}{729288171} \\ -\frac{11588543}{729288171} & -\frac{1412126}{243096057} & -\frac{201472}{243096057} & -\frac{719995}{729288171} & \frac{597362}{729288171} \\ \frac{17906471}{2917152684} & -\frac{2530643}{486192114} & -\frac{1585607}{243096057} & \frac{4895885}{1458576342} & \frac{7206599}{1458576342} \\ -\frac{402739}{486192114} & -\frac{110342}{81032019} & -\frac{332956}{81032019} & \frac{1356854}{243096057} & -\frac{1345549}{243096057} \end{bmatrix} \in \mathbb{Q}^{5 \times 5}$$

and $\kappa(A) \approx 1.324$. This example illustrates a key property enjoyed by a well conditioned matrix that our algorithm exploits: for each entry in A^{-1} , the bitlength of the numerator is less than, or at least not too much larger than, the bitlength of the denominator.

To the best of our knowledge, the best previously known complexity estimate for integer matrix inversion is $O(n^{\omega+1} \log \|A\|)$ bit operations, supported by any of the classical approaches such as homomorphic imaging and Chinese remaindering [6, Section 5.5], quadratic lifting via Newton iteration, or a recursive version of fraction-free Gaussian elimination [24, Section 2]. Here, ω is the exponent for matrix multiplication over a ring [2, Chapter 1] and the soft- O notation $O\tilde{}$ indicates some missing $\log n$ and $\log \log \|A\|$ factors.

Now consider the case of polynomial matrices. Let \mathbb{K} be a field, and let $A \in \mathbb{K}[z]^{n \times n}$ be nonsingular with entries bounded in degree by $d > 0$. The inverse of A may require on the order of $n^3 d$ field elements to represent. Similar to the integer case, a variety of classical approaches for polynomial matrix inversion exist, all with a cost estimate of $O\tilde{}(n^{\omega+1} d)$ field operations from \mathbb{K} . We refer to [10, Section 1] for a brief survey of previous methods, and for a discussion of some of the progress made in obtaining faster algorithms for problems on polynomial matrices. Jeannerod and Villard [10] propose a new approach that, for a generic A with n a power of two, will compute A^{-1} in $O\tilde{}(n^3 d)$ field operations. In this paper we adapt our approach for integer matrix inversion to the polynomial case. By incorporating some algebraic preconditioning techniques that are particular to polynomials, we obtain a Las Vegas probabilistic algorithm that has expected running time $O\tilde{}(n^3 d)$ for any nonsingular input.

The discovery of the essentially optimal inversion algorithm for generic polynomial matrices in [10], and the recent progress made in reducing the complexity of many basic linear algebra problems on integer matrices, motivates us to develop an alternative algorithm for

inversion that is applicable to integer matrices. Assuming $\omega = 3$, we recall in the next two paragraphs some results for two computational problems that are particularly relevant for this paper: nonsingular rational linear system solving and determinant/Smith-form computation. For a survey of work that has been done on computing these and other integer matrix invariants, as well as incorporating fast matrix multiplication techniques, we refer to [12, 25].

It was shown already more than a quarter of a century ago that, given as input a nonsingular matrix $A \in \mathbb{Z}^{n \times n}$ and a column vector $b \in \mathbb{Z}^{n \times 1}$, the rational system solution $A^{-1}b$ can be computed in $O^\sim(n^3 \log \|A\|)$ bit operations using linear p -adic lifting [15, 3]. Taking b to be a column of the identity matrix shows that any single column of the inverse can be computed in $O^\sim(n^3 \log \|A\|)$ bit operations. Linear p -adic lifting is a key subroutine of the algorithm in this paper. We show that lifting can be used to compute all n columns of the inverse of a well conditioned matrix in essentially the same time as a single column.

Now consider the computation of the determinant, assuming $\omega = 3$. Classical methods such as homomorphic imaging [6, Section 5.5] require $O^\sim(n^4 \log \|A\|)$ bit operations. A breakthrough was the Krylov approach of Kaltofen [11] which gives an $O^\sim(n^{3.5} \log \|A\|)$ Las Vegas algorithm. More recently, a Monte Carlo algorithm with running time $O^\sim(n^{3.5} (\log \|A\|)^{1.5})$ to compute not only the determinant but the entire Smith form of A is given by Eberly et al. [5]. The Smith form $\text{Diag}(s_1, s_2, \dots, s_n)$ of an $A \in \mathbb{Z}^{n \times n}$ is a canonical diagonalization under unimodular pre- and post-multiplication, see Section 2. The invariant factors s_i satisfy $|\det A| = s_1 s_2 \cdots s_n$, so the determinant of A is easily recovered once the form is known. By nontrivial invariant factors of A we mean those that are > 1 . Although an input matrix may have up to n nontrivial invariant factors, the number of distinct invariant factors can be shown to be bounded by $O(\sqrt{n(\log n + \log \|A\|)})$. The approach in [5] is based on computing the at most $k \in O^\sim(\sqrt{n \log \|A\|})$ distinct invariant factors, together with their multiplicities, by computing $O^\sim(k)$ rational linear system solutions, each of which can be accomplished in $O^\sim(n^3 \log \|A\|)$ bit operations using linear p -adic lifting. The overall cost of the algorithm in [5] is thus sensitive to k , the number of distinct invariant factors. On the one hand, the algorithm we present here avoids this sensitivity to k by being able to exploit the fact that $\sum_{i=1}^n \text{bitlength}(s_i) \leq n + \text{bitlength}(\det A)$, independent of the invariant structure of A . On the other hand, our algorithm is sensitive to the condition number $\kappa(A)$. We compute all n invariant factors in succession in time proportional to about $n^2 \sum_{i=1}^n (\log s_i + \log \kappa(A))$ bit operations.

Next, we motivate our approach for integer matrix inversion by recalling a fact about the adjoint of an $n \times n$ matrix A over a field, denoted by A^{adj} . Recall that A^{adj} is the $n \times n$ matrix with entry in row i column j equal to $(-1)^{i+j}$ times the determinant of the $(n-1) \times (n-1)$ submatrix of A obtained by deleting row j and column i . If A is nonsingular then $A^{\text{adj}} = (\det A)A^{-1}$, but note that the adjoint is well defined also for singular matrices. In particular, suppose that A has rank $n-1$. Then A has at least one nonzero minor of dimension $n-1$. Assume without loss of generality (up to a row and column permutation) that the leading $(n-1) \times (n-1)$ submatrix C of A is nonsingular, and partition A as

$$A = \left[\begin{array}{c|c} C & y \\ \hline x & a \end{array} \right].$$

If we set $u := -xC^{-1}$, a row vector of dimension $n - 1$, and $v := -C^{-1}y$, a column vector of dimension $n - 1$, then

$$\left[\begin{array}{c|c} I_{n-1} & \\ \hline u & 1 \end{array} \right] \overset{A}{\left[\begin{array}{c|c} C & y \\ \hline x & * \end{array} \right]} \left[\begin{array}{c|c} I_{n-1} & v \\ \hline & 1 \end{array} \right] = \left[\begin{array}{c|c} C & \\ \hline & 0 \end{array} \right]. \quad (2)$$

The adjoint of the matrix on the right hand side of (2) will have all entries zero except for the entry in the last row and last column, which will be equal to $\det C$. Replacing both sides of (2) with the adjoint of that side and solving for A^{adj} gives

$$A^{\text{adj}} = \left[\begin{array}{c} v \\ 1 \end{array} \right] (\det C) [u \mid 1] = \left[\begin{array}{c} (\det C)v \\ \det C \end{array} \right] [u \mid 1]. \quad (3)$$

Note that the formula for A^{adj} in (3) is valid also when the entries of A are coming from a principal ideal ring, even a ring with zero divisors such as a residue class ring of the integers, provided that $\det A = 0$ and $\det C$ is a unit from the ring (i.e., C is invertible over the ring). Consider in particular a nonsingular matrix $A \in \mathbb{Z}^{n \times n}$ for which the leading $(n - 1) \times (n - 1)$ submatrix C satisfies $\det C \perp \det A$. Then over the residue class ring $\mathbb{Z}/\langle \det A \rangle$, we have $\det A \equiv 0 \pmod{\det A}$ and $\det C$ a unit, so equation (3) holds modulo $\det A$. In other words, the adjoint of A over \mathbb{Z} is element-wise congruent to the rank 1 matrix in (3). Moreover, if $\det A$ is large enough, the exact adjoint of A over \mathbb{Z} can be obtained by multiplying out the outer product in (3) and reducing all entries modulo $\det A$ in the symmetric range $[-(|\det A| - 1)/2, |\det A|/2]$.

For example, the matrix

$$A = \begin{bmatrix} -93 & -32 & 8 & 44 \\ -76 & -74 & 69 & 92 \\ -72 & -4 & 99 & -31 \\ -2 & 27 & 29 & 67 \end{bmatrix}$$

has $\det A = 64334045$, which is relatively prime to $\det C = 533666$. Formula (3) gives

$$\begin{aligned} A^{\text{adj}} &\equiv \begin{bmatrix} -28408 \\ 861667 \\ 181262 \\ 533666 \end{bmatrix} \begin{bmatrix} -27259506 & 11404741 & -8995165 & 1 \end{bmatrix} \pmod{64334045} \\ &\equiv \begin{bmatrix} -853217 & 368292 & -179420 & -28408 \\ 409178 & -744548 & 233455 & 861667 \\ -584392 & 295157 & 438250 & 181262 \\ 62584 & 183281 & -289125 & 533666 \end{bmatrix} \pmod{64334045}. \end{aligned} \quad (4)$$

For this example, which is well conditioned, $\det A$ is large enough to capture all entries in A^{adj} ; the matrix in (4), obtained by multiplying out the outer product and reducing entries in the symmetric range modulo 64334045, is the adjoint of A over \mathbb{Z} .

To adapt the approach just described to compute the inverse of an arbitrary input matrix requires handling the case when all minor of dimension $n - 1$ in A have a common factor with $\det A$ (i.e., the Smith form of A is nontrivial). (Our example matrix (1) has nontrivial Smith form $\text{snf}(A) = \text{Diag}(1, 1, 1, 6, 486192114)$ which is why we chose a different matrix for the example in (4).) Our solution is to extend formula (3) to an A with nontrivial invariant structure by giving an expression for $A^{\text{adj}} \bmod \det A$ as the sum of scaled outer products.

$$A^{\text{adj}} \equiv \frac{\det A}{s_n} v_n u_n + \frac{\det A}{s_{n-1}} v_{n-1} u_{n-1} + \cdots + \frac{\det A}{s_1} v_1 u_1 \pmod{\det A}.$$

Each v_i is a column vector and each u_i a row vector. We call this construction an outer product adjoint formula for A . Actually, since all entries in A^{adj} are divisible by $(\det A)/s_n$, our definition of the formula in Section 4 is as shown above but with $\det A$ replaced by s_n , and the left hand side replaced with $s_n A^{-1}$.

We remark that the existence of an outer product adjoint formula for any nonsingular $A \in \mathbb{R}^{n \times n}$ over any principal ideal domain \mathbb{R} follows directly from the existence of the Smith form over \mathbb{R} . Our contribution in this paper is twofold. First, we identify some useful properties of the formula: (1) when $\mathbb{R} = \mathbb{Z}$ or $\mathbb{R} = \mathbb{K}[z]$, an outer product adjoint formula can be represented using about the same space to represent A ; (2) an outer product adjoint formula gives an essentially optimal method to compute $\text{Rem}(A^{\text{adj}}v, \det A)$ for a given vector v that has entries reduced modulo $\det A$. Second, and the main purpose of this paper, we exploit these properties to obtain fast probabilistic algorithms to compute an outer product adjoint formula.

When applying the outer product adjoint formula to compute A^{adj} , a problem occurs if $|\det A|$ is too small to capture the entries of A^{adj} in the symmetric range modulo $\det A$. In this case we divide the computation of A^{adj} into two parts. Consider the decomposition $s_n A^{-1} = \text{Rem}(s_n A^{-1}, s_n) + s_n R$ where Rem denotes the remainder modulo s_n . First we compute $\text{Rem}(s_n A^{-1}, s_n)$ via an outer product adjoint formula. Then we compute the remaining part $R := (1/s_n)(s_n A^{-1} - \text{Rem}(s_n A^{-1}, s_n))$ using linear p -adic lifting. Unfortunately, the cost of computing both of these parts is sensitive to $\log \kappa(A)$.

The rest of this paper is organized as follows. In Section 2 we fix some notation and recall some definitions, including that of the Smith canonical form. Section 3 considers the problem of computing, for a nonsingular $A \in \mathbb{R}^{n \times n}$, a Smith decomposition: $A = * \text{snf}(A) *$ where $*$ are $n \times n$ matrices over \mathbb{R} . Section 4 defines, and gives an algorithm for computing, the outer product adjoint formula, which is essentially a Smith decomposition $s_n A^{-1} = * \text{snf}(s_n A^{-1}) *$. Sections 3 and 4 develop results generically so they apply both over $\mathbb{R} = \mathbb{Z}$ and $\mathbb{R} = \mathbb{K}[z]$. Section 5 gives our integer matrix inversion algorithm, and Section 6 the variation for polynomial matrices. Fortunately, we can apply some simple algebraic preconditioning techniques to transform an original input matrix $A \in \mathbb{K}[z]^{n \times n}$ of degree d into a new matrix B that will have determinant of degree nd . Once the inverse of B has

been determined via an outer product adjoint formula, the preconditioning can be reversed in the allotted time to recover the inverse of A .

Cost functions

Let $M: \mathbb{Z}_{>0} \rightarrow \mathbb{R}_{>0}$ be such that integers bounded in magnitude by 2^t can be multiplied using at most $M(t)$ bit operations. The Schönhage–Strassen algorithm [21] allows $M(t) = O(t(\log t)(\log \log t))$. We assume that $M(a) + M(b) \leq M(a + b)$ and $M(ab) \leq M(a)M(b)$ for $a, b \in \mathbb{N}_{\geq 2}$. We refer to [6, Section 8.3] for further references and discussion about integer multiplication.

It will be useful to define an additional function B for bounding the cost of integer gcd-related computations. We can take $B(t) = M(t) \log t$. Then the extended gcd problem with two integers bounded in magnitude by 2^t , and the rational number reconstruction problem [6, Section 5.10] with modulus bounded by 2^t , can be solved with $O(B(t))$ bit operations [20] (compare with [19]).

We will overload notation slightly and use $M: \mathbb{Z}_{\geq 0} \rightarrow \mathbb{R}_{>0}$ as a cost function for polynomial multiplication: two polynomials in $\mathbb{K}[z]$ of degree bounded by d can be multiplied using at most $M(d)$ field operations. Similarly, B will be used as a cost function for gcd-related problems like rational function reconstruction and extended gcd. Similar to the integer case, we can take $B(d) = M(d) \log d$. We refer to [6, Section 11.1] for more details and references.

2 Definitions, notation and preliminaries

Let R be a principal ideal ring, a commutative ring with identity in which every ideal is principal. In this paper our focus is on the integral domains $R = \mathbb{Z}$ and $R = \mathbb{K}[z]$. Following [17], we prescribe a complete set of non-associates $\mathcal{A}(R)$ and, for every nonzero $s \in R$, a complete set of residues $\mathcal{R}(R, s)$.

$$\begin{array}{c|c} R = \mathbb{Z} & R = \mathbb{K}[z] \\ \hline \mathcal{A}(\mathbb{Z}) = \{0, 1, 2, \dots\} & \mathcal{A}(\mathbb{K}[z]) = \{0\} \cup \{f \in \mathbb{K}[z] \mid f \text{ is monic}\} \\ \mathcal{R}(\mathbb{Z}, s) = \left[-\left\lfloor \frac{|s|-1}{2} \right\rfloor, \left\lfloor \frac{|s|}{2} \right\rfloor\right] & \mathcal{R}(\mathbb{K}[z], s) = \{f \in \mathbb{K}[z] \mid \deg f < \deg s\} \end{array} \quad (5)$$

Note that our choice for $\mathcal{R}(\mathbb{Z}, s)$ corresponds to the usual “symmetric range” modulo s .

For nonzero $N \in R$, the function $\text{Rem}(a, N)$ returns the element of $\mathcal{R}(R, N)$ that is congruent to a modulo N . The next lemma follows as a consequence of our choices for $\mathcal{R}(R, N)$.

Lemma 1. *Let $a, N \in R$ with N nonzero. Then $a = \text{Rem}(a, N)$ if*

$$\begin{array}{l} R = \mathbb{Z} : \quad |N| \geq 2|a| + 2 \\ R = \mathbb{K}[z] : \quad \deg N \geq \deg a + 1 \end{array}$$

For $a, s \in R$, we denote by $\gcd(a, s)$ the unique principal generator in $\mathcal{A}(R)$ of the ideal generated by a and s . We allow \gcd to take an arbitrary number of arguments, including matrices and vectors as well as individual elements of R . For example, if B is a matrix over R and s is an element of R , then $\gcd(B, s)$ denotes the gcd of s and all entries in B .

We can use the definition of \gcd over R to induce definitions of \mathcal{A} and \mathcal{R} for a residue class ring $R/\langle s \rangle$ from the definitions of \mathcal{A} and \mathcal{R} over R . This will be useful below where we recall how the Smith form over R can be computed by working over $R/\langle s \rangle$ for a well chosen s . For nonzero $s \in R$, we identify the residue class ring $R/\langle s \rangle$ with the set of elements $\mathcal{R}(R, s)$, and define

$$\mathcal{A}(R/\langle s \rangle) = \{\gcd(a, s) \mid a \in \mathcal{R}(R, s)\} \quad \text{and} \quad \mathcal{R}(R/\langle s \rangle, b) = \mathcal{R}(R, \gcd(b, s)).$$

These choices for \mathcal{A} and \mathcal{R} allow us to easily obtain algorithms for basic operations over $R/\langle s \rangle$ in terms of algorithms for basic operations over R , see [24, Section 1.1].

The Smith canonical form

Corresponding to every $A \in R^{n \times n}$ there exist unimodular (invertible over R) matrices $P, Q \in R^{n \times n}$ such that $\text{snf}(A) = S = PAQ = \text{Diag}(s_1, s_2, \dots, s_n)$ with S in Smith canonical form [17, Chapter II], that is, with $s_i \mid s_{i+1}$ for $1 \leq i \leq n-1$ and $s_i \in \mathcal{A}(R)$ for $1 \leq i \leq n$. When R is a principal ideal domain, s_n is an associate of $(\det A)/\gcd(A^{\text{adj}})$. Thus, the largest invariant factor s_n is the smallest nonzero element of R (minimal degree over $K[z]$ and minimal magnitude over \mathbb{Z}) such that $s_n A^{-1}$ is over R .

The classical approach to compute the Smith form over $R = \mathbb{Z}$ or $R = K[z]$ is to apply a sequence of elementary row and column operations. A well known problem is that entries in the work matrix can grow excessively large. To avoid this phenomenon, many authors (e.g., [4, 9, 8]) have used the idea of the following lemma in conjunction with that of Lemma 1. The lemma follows from existence and uniqueness of the Smith form over any principal ideal ring [13], in particular over $R/\langle s \rangle$. For a proof see [22, Theorem 12].

Lemma 2. *Let $A \in R^{n \times n}$ be nonsingular, and let $s \in R$ be a nonzero multiple of s_n , the largest invariant factor of A . If S is the Smith form of A over R , and \bar{S} is the Smith form of $\bar{A} = \text{Rem}(A, s)$ over $R/\langle s \rangle$, then $\bar{S} = \text{Rem}(S, s)$.*

To help clarify the algorithms developed in the subsequent sections, we sketch here an approach of [7] to transform a nonsingular $A \in \mathbb{Z}^{n \times n}$ to Smith form. To slightly simplify the presentation, we will work with the input matrix augmented with an initial row and column of zeroes.

$$\left[\begin{array}{c|c} 0 & \\ \hline & A \end{array} \right] \in \mathbb{Z}^{(n+1) \times (n+1)}$$

The nonzero invariant structure remains unchanged. Compute $y \in \mathbb{Z}^{n \times 1}$ such that $\gcd(Ay) = \gcd(A)$. Now compute $x \in \mathbb{Z}^{1 \times n}$ such that

$$\left[\begin{array}{c|c} 1 & x \\ \hline & I_n \end{array} \right] \left[\begin{array}{c|c} 0 & \\ \hline & A \end{array} \right] \left[\begin{array}{c|c} 1 & \\ \hline y & I_n \end{array} \right] = \left[\begin{array}{c|c} s_1 & u \\ \hline v & * \end{array} \right],$$

with s_1 equal to the gcd of all entries in A . Next, apply some more elementary row and column operations to zero out entries below and to the right of s_1 .

$$\left[\begin{array}{c|c} 1 & -u/s_1 \\ \hline & I_n \end{array} \right] \left[\begin{array}{c|c} s_1 & u \\ \hline v & * \end{array} \right] \left[\begin{array}{c|c} 1 & \\ \hline -v/s_1 & I_n \end{array} \right] = \left[\begin{array}{c|c} s_1 & \\ \hline & B \end{array} \right].$$

The remaining invariant factors are computed in a similar fashion by recursing on B , which by construction has $\text{snf}(B) = \text{Diag}(s_2, s_3, \dots, s_n, 0)$. To avoid the problem of expression swell we can apply Lemma 2. In particular, if we have precomputed a nonzero multiple s of s_n , then it suffices to compute x and y such that $s_1 = \text{Rem}(xAy, s)$. Furthermore, the entries of B may be reduced modulo s . Actually, since s_1 divides all entries in $\text{Diag}(s_1, B)$, we can continue with $(1/s_1)B$ and modulus s/s_1 instead.

3 The Smith decomposition

Let $B \in \mathbb{R}^{n \times n}$ be nonsingular with $\text{snf}(B) = \text{Diag}(g_1, g_2, \dots, g_n)$. In this section we adapt the approach for Smith form computation described in the previous section to compute not only the Smith form of B , but to also construct column vectors $v_1, v_2, \dots, v_n \in \mathbb{R}^{n \times 1}$ and row vectors $u_1, u_2, \dots, u_n \in \mathbb{R}^{1 \times n}$ such that B can be expressed as the sum of scaled outer products:

$$B = T_{n+1} = g_1 v_n u_n + g_2 v_{n-1} u_{n-1} + \dots + g_n v_1 u_1. \quad (6)$$

Note that initially we will work directly over \mathbb{R} . After, we will apply the idea of Lemma 2 to avoid the problem of expression swell.

For convenience define $g_0 = 1$. Define $e_j = g_j/g_{j-1}$ so that $g_j = e_1 e_2 \dots e_j$, $1 \leq j \leq n$. We will construct matrices $B_1, B_2, \dots, B_{n+1} \in \mathbb{R}^{n \times n}$ such that

$$B_j = \frac{1}{g_{j-1}} \left(B - \overbrace{(g_1 v_n u_n + g_2 v_{n-1} u_{n-1} + \dots + g_{j-1} v_{n-j+2} u_{n-j+2})}^{T_j} \right) \quad (7)$$

with

$$\text{snf}(B_j) = \text{Diag} \left(\underbrace{e_j}_{g_j/g_{j-1}}, \underbrace{e_j e_{j+1}}_{g_{j+1}/g_{j-1}}, \dots, \underbrace{e_j e_{j+1} \dots e_n}_{g_n/g_{j-1}}, 0, 0, \dots, 0 \right). \quad (8)$$

Considering (8), $\text{snf}(B_{n+1})$ will be the zero matrix, showing that (6) will follow from (7) for $j = n + 1$.

We now explain how to construct the B_j by induction on j , $1 \leq j \leq n + 1$, with base case $j = 1$. Initialize $B_1 = B$. Then B_1 satisfies (7) and (8). Suppose B_j satisfying (7) and (8) have been computed for $1 \leq j \leq i$, for some i with $1 \leq j \leq n$. We show how to compute B_{i+1} satisfying (7) and (8). Because \mathbb{R} is a principal ideal ring, there exist vectors $x \in \mathbb{R}^{1 \times n}$ and $y \in \mathbb{R}^{n \times 1}$ such that $e_i := x B_i y$, the gcd of all entries of B_i . (We defer until Sections 5 and 6 to describe how to compute x and y .) The nonzero invariant structure of a matrix does not change if we embed into a larger zero matrix. Let $v := B_i y$ and $u := x B_i$,

and consider the unimodular transformation of the matrix obtained from B_i by augmenting with an initial row and column of zeroes.

$$\left[\begin{array}{c|c} 1 & x \\ \hline & I_n \end{array} \right] \left[\begin{array}{c|c} 0 & \\ \hline & B_i \end{array} \right] \left[\begin{array}{c|c} 1 & \\ \hline y & I_n \end{array} \right] = \left[\begin{array}{c|c} e_i & u \\ \hline v & B_i \end{array} \right] \quad (9)$$

Let $v_{n-i+1} := v/e_i$ and $u_{n-i+1} := u/e_i$, and apply another unimodular transformation to the matrix on the right hand side of (9) to zero out entries below and to the right of e_i .

$$\left[\begin{array}{c|c} 1 & -u_{n-i+1} \\ \hline & I_n \end{array} \right] \left[\begin{array}{c|c} e_i & u \\ \hline v & B_i \end{array} \right] \left[\begin{array}{c|c} 1 & \\ \hline -v_{n-i+1} & I_n \end{array} \right] = \left[\begin{array}{c|c} e_i & \\ \hline & B_i - e_i v_{n-i+1} u_{n-i+1} \end{array} \right] \quad (10)$$

All entries of the matrix on the right hand side of (10) are divisible by e_i . Let $B_{i+1} = (1/e_i)(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{R}^{n \times n}$, matching (7) for $j = i + 1$. Since the matrix on the right of (10) has the same nonzero invariant factors as B_i , we conclude that (8) also holds for $j = i + 1$. The procedure just described is summarized by the following code fragment, correctness of which follows by induction.

Input: • nonsingular $B \in \mathbb{R}^{n \times n}$
 Let $B_1 = B$.
for i **from** 1 **to** n **do**
 Let $x \in \mathbb{R}^{1 \times n}$ and $y \in \mathbb{R}^{n \times 1}$ be such that $x B_i y = \gcd(B_i)$.
 $v := B_i y$; $u := x B_i$; $e_i := xv$;
 $g_i := e_1 e_2 \dots e_i$; $v_{n-i+1} := v/e_i$; $u_{n-i+1} := u/e_i$;
 Let $B_{i+1} = \frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{R}^{n \times n}$.
 assert: Equations (7) and (8) hold over \mathbb{R} for $j = i + 1$.
od
assert: $B = g_1 v_n u_n + g_2 v_{n-1} u_{n-1} + \dots + g_n v_1 u_1$.

Figure 1: Computing a Smith decomposition

Figure 2 adjust the algorithm in Figure 1 to compute a decomposition in (6) that only holds modulo s , where s is a nonzero multiple of g_n . All operations are performed explicitly over \mathbb{R} , but Lemma 2 is applied at iteration i to compute modulo s/g_{i-1} . There are two subtleties to be aware of. First, at iteration i the matrix B_i is only correct modulo s/g_{i-1} . By Lemma 2, we need to compute e_i as the gcd of all entries of B_i over $\mathbb{R}/\langle s/g_{i-1} \rangle$: over \mathbb{R} we compute the gcd of s/g_{i-1} with all entries in B_i , and then reduce modulo s/g_{i-1} . Second, if s is an associate of g_n , then there exists a minimal $k \leq n$ such that $g_k = g_{k+1} = \dots = g_n$, in which case B_k is congruent the zero matrix modulo s/g_{k-1} : this is handled by exiting the loop early since the remaining part of the sum $g_k v_{n-k+1} + g_{k+1} v_{n-k} u_{n-k} + \dots + g_n v_1 u_1$ in (6) is known to congruent to zero modulo s in this case.

Input: • nonsingular $B \in \mathbb{R}^{n \times n}$

• $s \in \mathbb{R}$, a nonzero multiple of the largest invariant factor of B

Let $B_1 = B$.

for i **from** 1 **to** n **do**

Let $x \in \mathbb{R}^{1 \times n}$ and $y \in \mathbb{R}^{n \times 1}$ be such that $\text{Rem}(xB_i y, s/g_{i-1}) = \text{Rem}(\text{gcd}(B_i, s/g_{i-1}), s/g_{i-1})$.

$v := \text{Rem}(B_i y, s/g_{i-1}); \quad u := \text{Rem}(x B_i, s/g_{i-1}); \quad e_i := \text{Rem}(x v, s/g_{i-1});$

if $e_i = 0$ **then break fi;**

$g_i := e_1 e_2 \dots e_i; \quad v_{n-i+1} := v/e_i; \quad u_{n-i+1} := u/e_i;$

Let $B_{i+1} = \frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{R}^{n \times n}$.

assert: Equations (7) and (8) hold modulo s/g_i for $j = i + 1$.

od

assert: $B \equiv g_1 v_n u_n + g_2 v_{n-1} u_{n-1} + \dots + g_i v_{n-i+1} u_{n-i+1} \pmod{s}$.

Figure 2: Computing a Smith decomposition modulo s

Our example matrix (1) has $\text{snf}(A) = \text{Diag}(1, 1, 1, 6, 486192114)$ and thus a Smith decomposition looks like

$$\begin{bmatrix} & & A & & \\ -6 & 39 & 37 & -1 & 34 \\ 34 & 31 & -23 & -13 & -60 \\ -33 & 44 & -61 & -14 & 36 \\ 3 & 24 & 33 & 90 & -36 \\ -44 & 8 & 25 & 17 & -76 \end{bmatrix} \equiv v_5 u_5 + v_4 u_4 + v_3 u_3 + 6v_2 u_2 \pmod{486192114}.$$

Possible candidates for the v_* and u_* computed by the algorithm in Figure 2 are

$$\left[v_5 \mid v_4 \mid v_3 \mid v_2 \right] = \left[\begin{array}{c|c|c|c} -181370347 & -136725485 & -73722068 & 16586019 \\ 80747651 & -55699749 & 128441852 & -2597696 \\ 206748189 & 86729736 & 128981629 & 31649411 \\ 200642793 & -213293490 & 189870633 & -27171926 \\ 105280182 & -80860249 & -88302930 & 8973973 \end{array} \right]$$

and

$$\begin{bmatrix} u_5 \\ u_4 \\ u_3 \\ u_2 \end{bmatrix} = \begin{bmatrix} 210868706 & 162161847 & -191502252 & 229783944 & 81503766 \\ -116555956 & -71504576 & 108324989 & 97759333 & -178866998 \\ -201927321 & 185436071 & 122733185 & 241507744 & -224966490 \\ 31802687 & -30546106 & 14610312 & -39897254 & -31315274 \end{bmatrix}.$$

4 The outer product adjoint formula

Let \mathbb{R} be a principal ideal domain and let $A \in \mathbb{R}^{n \times n}$ be nonsingular with Smith form $S = UAV = \text{Diag}(s_1, s_2, \dots, s_n)$. Let v_i and u_i be column i and row i of the unimodular matrices V and U respectively, $1 \leq i \leq n$. Inverting both sides of the equation $S = UAV$, multiplying by s_n , and solving for $s_n A^{-1}$ gives

$$s_n A^{-1} = V(s_n S^{-1})U = \frac{s_n}{s_n} v_n u_n + \frac{s_n}{s_{n-1}} v_{n-1} u_{n-1} + \dots + \frac{s_n}{s_1} v_1 u_1. \quad (11)$$

Note that $\text{snf}(s_n A^{-1}) = \text{Diag}(s_n/s_n, s_n/s_{n-1}, \dots, s_n/s_1)$. Now consider taking equation (11) modulo s_n . Since each outer product $v_i u_i$ is scaled by s_n/s_i , the equation will still hold modulo s_n if entries in v_i and u_i are reduced modulo s_i , $1 \leq i \leq n$. This is made precise in the following definition.

Definition 3. An outer product adjoint formula of a nonsingular $A \in \mathbb{R}^{n \times n}$ is set of tuples $(s_{n-i+1}, v_{n-i+1}, u_{n-i+1})_{1 \leq i \leq k}$ such that:

- the Smith form of A is $\text{Diag}(1, 1, \dots, 1, s_{n-k+1}, s_{n-k+2}, \dots, s_n)$ with $s_{n-k+1} \neq 1$,
- $v_{n-i+1} \in \mathcal{R}(\mathbb{R}, s_{n-i+1})^{n \times 1}$ and $u_{n-i+1} \in \mathcal{R}(\mathbb{R}, s_{n-i+1})^{1 \times n}$ for $1 \leq i \leq k$,
- $s_n A^{-1} \equiv \frac{s_n}{s_n} v_n u_n + \frac{s_n}{s_{n-1}} v_{n-1} u_{n-1} + \dots + \frac{s_n}{s_{n-k+1}} v_{n-k+1} u_{n-k+1} \pmod{s_n}$.

Our example matrix (1) has $\text{snf}(A) = (1, 1, 1, 6, 486192114)$ and thus an outer product adjoint formula for A looks

$$\begin{array}{c} (48619211)A^{-1} \\ \left[\begin{array}{ccccc} 293976 & 4400604 & -3031875 & 1175085 & -5335416 \\ 5720512 & 3878750 & 2657920 & 982372 & 290686 \\ 5595092 & -803624 & -4142521 & -1010731 & 1654028 \\ -2940960 & -2029452 & 1296141 & 5609775 & -1756794 \\ 1614607 & -2857735 & 962332 & 345436 & -3126620 \end{array} \right] \equiv v_5 u_5 + (81032019)v_4 u_4 \end{array}$$

where the equation holds modulo 48619211. Possible candidates for the v_* and u_* are

$$\left[v_4 \mid v_5 \right] = \begin{bmatrix} 0 & -77803731 \\ -2 & -123712574 \\ 0 & 114019307 \\ -2 & 230738811 \\ 1 & -217914158 \end{bmatrix}$$

and

$$\begin{bmatrix} u_4 \\ u_5 \end{bmatrix} = \left[\begin{array}{ccccc} 3 & 3 & 0 & -2 & 0 \\ -68275484 & 1328828 & -146932559 & 183249787 & 221429404 \end{array} \right].$$

```

OuterProductAdjoint( $A, s_n$ )
Input: • nonsingular  $A \in \mathbb{R}^{n \times n}$ 
          •  $s_n \in \mathbb{R}$ , the largest invariant factor of  $A$ 
Output: An outer product adjoint formula of  $A$ .
Let  $B_1 = s_n A^{-1}$  and  $s_{n+1} = s_n$ .
for  $i$  to  $n$  do
  Let  $x \in \mathbb{R}^{1 \times n}$  and  $y \in \mathbb{R}^{n \times 1}$  such that  $\text{Rem}(xB_i y, s_{n-i+2}) = \text{Rem}(\text{gcd}(B_i, s_{n-i+2}), s_{n-i+2})$ .
   $u := \text{Rem}(xB_i, s_{n-i+2}); \quad v := \text{Rem}(B_i y, s_{n-i+2}); \quad e_i := \text{Rem}(xv, s_{n-i+2});$ 
  if  $e_i = 0$  then break fi;
   $s_{n-i+1} := s_{n-i+2}/e_i; \quad u_{n-i+1} := u/e_i; \quad v_{n-i+1} := v/e_i;$ 
  Let  $B_{i+1} = \frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{R}^{n \times n}$ .
od;
return  $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}$ 

```

Figure 3: Algorithm OuterProductAdjoint

Algorithm OuterProductAdjoint shown in Figures 4 will compute an outer product adjoint formula. The algorithm is identical to that shown in Figure 2 except with $s = s_n$ and $g_i = s_n/s_{n-i+1}$.

When we specialize to the rings $\mathbb{R} = \mathbb{Z}$ and $\mathbb{R} = \mathbb{K}[z]$ most of the algorithm will be implemented exactly as written. Note that the vector y that needs to be chosen at the start of each iteration is required only to construct v . We will appeal to known results to construct v as an \mathbb{R} -linear combination of $B_i Y$ for a Y that is randomly chosen so that $\text{gcd}(B_i Y, s_{n-i+2}) = \text{gcd}(B_i, s_{n-i+1})$ with high probability. Once v is known, x is computed as the solution to an extended euclidean problem.

The main technical issue we face is to efficiently compute $\text{Rem}(B_i Y, s_{n-i+2})$ for a $Y \in \mathcal{R}(\mathbb{R}, s_{n-i+1})^{n \times m}$. (The problem of computing a vector-matrix product $\text{Rem}(xB_i, s_{n-i+2})$ is analogous so we focus here on the matrix-vector case.) Exactly as in Section 3, but here with $s = s_n$ and $g_i = s_n/s_{n-i+1}$, if we define

$$T_i = \frac{s_n}{s_n} v_n u_n + \frac{s_n}{s_{n-1}} v_{n-1} u_{n-1} + \cdots + \frac{s_n}{s_{n-i+2}} v_{n-i+1} u_{n-i-1} \in \mathbb{R}^{n \times n}, \quad (12)$$

then at loop iteration $i = 1, 2, \dots, k$ we work with the matrix

$$B_i = \frac{s_{n-i+2}}{s_n} (s_n A^{-1} - T_i) \in \mathbb{R}^{n \times n} \quad (13)$$

and modulus s_{n-i+2} , where s_{n+1} is defined to be s_n . $\text{Rem}(T_i Y, s_n)$ can be computed efficiently

for a given $Y \in \mathcal{R}(\mathbb{R}, s_n)^{n \times m}$ by using nested multiplication.

$$\begin{aligned}
T_i Y &\equiv \left(\sum_{j=1}^{i-1} \frac{s_n}{s_{n-j+1}} v_{n-j+1} u_{n-j+1} \right) Y \pmod{s_n} \\
&\equiv \sum_{j=1}^{i-1} \frac{s_n}{s_{n-j+1}} \underbrace{\text{Rem}(v_{n-j+1} (u_{n-j+1} \overbrace{\text{Rem}(Y, s_{n-j+1})}^{Y_{n-j+1}})), s_{n-j+1}}_{X_{n-j+1}} \\
&\equiv \frac{s_n}{s_n} \left(X_n + \dots + \frac{s_{n-i+5}}{s_{n-i+4}} \left(X_{n-i+4} + \frac{s_{n-i+4}}{s_{n-i+3}} \left(X_{n-i+3} + \frac{s_{n-i+3}}{s_{n-i+2}} X_{n-i+2} \right) \right) \dots \right)
\end{aligned}$$

Algorithm OPM shown in Figures 4 implements the above scheme. In the first loop, iteration j does $O(nm)$ arithmetic operations modulo s_{n-j+1} with operands from $\mathcal{R}(\mathbb{R}, s_{n-j+1})$ and $\mathcal{R}(\mathbb{R}, s_{n-j+2})$. Iteration j of the second loop does $O(nm)$ operations modulo s_{n-j+1} with operands from $\mathcal{R}(\mathbb{R}, s_{n-j+1})$. Lemma 4 follows from the superlinearity of M .

OPM($s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, Y$)

Input: • nonzero $s \in \mathcal{A}(\mathbb{R})$

- $v_{n-j+1} \in \mathcal{R}(\mathbb{R}, s_{n-j+1})^{n \times 1}$ and $u_{n-j+1} \in \mathcal{R}(\mathbb{R}, s_{n-j+1})^{1 \times n}$, $1 \leq j \leq i-1$
- $s_{n-i+2} | s_{n-i+3} | \dots | s_n$ with $s_n = s$ and s_{n-i+2} a nonunit if $i > 1$
- $Y \in \mathcal{R}(\mathbb{R}, s)^{n \times m}$

Output: $\text{Rem}((\sum_{j=1}^{i-1} (s/s_{n-j+1}) v_{n-j+1} u_{n-j+1}) Y, s)$.

if $i \leq 1$ **then return** the $n \times m$ zero matrix \mathbf{f} ;

$Y_{n+1} := Y$; $s_{n+1} := s_n$;

for j **from** 1 **to** $i-1$ **do**

$Y_{n-j+1} := \text{Rem}(Y_{n-j+2}, s_{n-j+1})$; $X_{n-j+1} := \text{Rem}(v_{n-j+1} (u_{n-j+1} Y_{n-j+1}), s_{n-j+1})$

od;

$V := X_{n-i+2}$;

for j **from** $i-2$ **downto** 1 **do** $V := \text{Rem}(X_{n-j+1} + (s_{n-j+1}/s_{n-j})V, s_{n-j+1})$ **od**;

return V

Figure 4: Algorithm OPM

Lemma 4. *Algorithm 4 (OPM) is correct. The cost of the algorithm is*

$$\mathbb{R} = \mathbb{Z} : \quad O(nm M(\log \prod_{i=1}^k s_{n-i+1})) \text{ bit operations.}$$

$$\mathbb{R} = \mathbb{K}[z] : \quad O(nm M(\sum_{i=1}^k \deg s_{n-i+1})) \text{ field operations from } \mathbb{K}.$$

Now consider the computation of $\text{Rem}(B_i Y, s_{n-i+2})$. Notice that that the matrices B_i are not explicitly computed. Instead, from the definition of B_i in (13) we have that

$$\text{Rem}(B_i Y, s_{n-i+2}) = \text{Rem} \left(\frac{s_{n-i+2}}{s_n} (s_n A^{-1} Y - \text{Rem}(T_i Y, s_n)), s_{n-i+2} \right)$$

Let $N \in \mathbb{R}$ be relatively prime to s_n . If $\text{Rem}(B_i Y, N) = B_i Y$ then also

$$\text{Rem}(\text{Rem}(B_i Y, N), s_{n-i+1}) = \text{Rem}(B_i Y, s_{n-i+1})$$

and we can apply the following recipe.

$$\text{Rem}(B_i Y, s_{n-i+2}) = \left[\begin{array}{l} W_1 := \text{Rem}(A^{-1}Y, N); \\ W_2 := \text{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, Y); \\ \text{return } \text{Rem}(\text{Rem}(s_{n-i+2}W_1 - (s_{n-i+2}/s_n)W_2, N), s_{n-i+2}) \end{array} \right] \quad (14)$$

Theorem 5. *If $Y \in \mathcal{R}(\mathbb{R}, s_{n-i+2})^{n \times m}$ then recipe (14) returns $\text{Rem}(B_i Y, s_{n-i+2})$ if*

$$\mathbb{R} = \mathbb{Z} : \quad N \geq s_{n-i+2}(n s_{n-i+2} \|A^{-1}\| + 1) + 2.$$

$$\mathbb{R} = \mathbb{K}[z] : \quad \deg N \geq \max(\deg s_{n-i+2}, \deg s_n A^{-1} + 2 \deg s_{n-i+2} - \deg s_n).$$

Proof. The matrices $s_n A^{-1}Y$ and $\text{Rem}(T_i Y, s_n)$ are over \mathbb{R} and their difference is divisible by s_n/s_{n-i+2} . The result will follow if we show that N satisfies the bounds of Lemma 1, with $B_i Y$ playing the role of a .

Over $\mathbb{K}[z]$ we have $\deg s_n A^{-1}Y \leq \deg s_n A^{-1} + \deg Y \leq \deg s_n A^{-1} + \deg s_{n-i+2} - 1$ and $\deg \text{Rem}(T_i Y, s_n) \leq \deg s_n - 1$. The bound for $\deg N$ given in theorem is obtained by adding one to the maximum of these two bounds and subtracting $\deg s_n/s_{n-i+2}$.

Over \mathbb{Z} we have $\|s_n A^{-1}Y\| \leq n s_n \|A^{-1}\| \|Y\| \leq n s_n \|A^{-1}\| s_{n-i+2}/2$ and $\|\text{Rem}(T_i Y, s_n)\| \leq s_n/2$, hence the difference of these matrices multiplied by s_{n-i+2}/s_n has entries bounded in magnitude by $n s_{n-i+2} \|A^{-1}\| \|Y\| + s_{n-i+2}/2$; the bound for N in the theorem is obtained by multiplying this bound two and adding two. \square

Corollary 6. *Consider the case $\mathbb{R} = \mathbb{K}[z]$. If $Y \in \mathbb{K}^{n \times m}$ and $\deg s_n A^{-1} - \deg s_n \leq 0$, then recipe (14) returns $\text{Rem}(B_i Y, s_{n-i+2})$ if $\deg N \geq \deg s_{n-i+2}$.*

5 Computing the inverse of an integer matrix

Algorithm `IntInverse` is shown in Figure 5. Phase 1 uses randomization to probabilistically compute the largest invariant factor of the input matrix $A \in \mathbb{Z}^{n \times n}$, together with a bound for $\|A^{-1}\|$. Phase 2 adapts algorithm `OuterProductAdjoint` shown in Figure 3 by incorporating randomization to obtain a probabilistic algorithm for computing the outer product adjoint of A . Phase 3 certifies correctness of the computed outer product adjoint formula from phase two and computes the explicit inverse of A . In the following subsections we fill in the implementation details and prove probability results and complexity estimates for each phase. First we recall some known results.

By the denominator of a rational matrix or vector we mean the smallest positive integer multiple that will clear the denominators, assuming the fractions are reduced.

Lemma 7 ([5, Theorem 2.1]). *Let $A \in \mathbb{Z}^{n \times n}$ be nonsingular. If $X \in L^{n \times 2}$ is chosen uniformly and randomly, where $L := \{0, 1, \dots, M-1\}$ with $M := 6 + \lceil 2n(\log_2 n + \log_2 \|A\|) \rceil$, then the denominator of $A^{-1}X$ is the largest invariant factor of A with probability at least $1/3$.*

IntInverse(A)

Input: Nonsingular $A \in \mathbb{Z}^{n \times n}$.

Output: A^{-1}

Note: : Fail may be returned with probability $< 1/2$.

1. **[Initialization]**

Choose \bar{p} uniformly and randomly from among the first $12 \lceil \log_2(n^{n/2} \|A\|^n) \rceil$ primes;
 $p :=$ the smallest positive integer power of \bar{p} such that $p \geq \log \sqrt{n} + \log \|A\|$;
if $\bar{p} \nmid \det A$ **return** Fail **else** $C := \text{Rem}(A^{-1}, p)$ **fi**;
 $\alpha := \|A^{-1}X\|$, where $X \in \{-1, 1\}^{n \times 4}$ is chosen uniformly and randomly;
 $M := 6 + \lceil 2n(\log_2 n + \log_2 \|A\|) \rceil$; $L := \{0, \dots, M-1\}$;
 $s :=$ the denominator of $A^{-1}X$, where $X \in L^{n \times 12}$ is chosen uniformly and randomly;
 $m := 2 \lceil (2 + \log_2 n) / \log_2 3 \rceil$;

2. **[Compute Outer Product Adjoint Formula]**

Let B_1 denote sA^{-1} and $s_{n+1} = s$.

for i **to** n **do**

Choose $Y \in L^{n \times m}$ uniformly and randomly;

$Y := \text{Rem}(Y, s_{n-i+2})$;

Set $N := p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal s.t. $p^k \geq s_{n-i+2}(n\alpha s_{n-i+2} + 1) + 2$;

Compute $V := \text{Rem}(B_i Y, s_{n-i+2})$ using recipe (14);

Use Lemma 10 to compute x and y such that $\gcd(xVy, s_{n-i+2}) = \gcd(V, s_{n-i+2})$;

$v := \text{Rem}(Vy, s_{n-i+2})$;

Compute $u := \text{Rem}(B_i^T x^T, s_{n-i+2})^T$ using recipe (14);

$e_i := \text{Rem}(xv, s_{n-i+2})$; **if** $e_i = 0$ **then break else** $s_{n-i+1} := s_{n-i+2}/e_i$ **fi**;

if ($i = 1$ **and** $s_n \neq s$) **or** $\prod_{j=1}^i s_{n-j+1} > n^{n/2} \|A\|^n$ **or** $e_i \nmid u$ **then return** Fail **fi**;

$u_{n-i+1} := u/e_i$; $v_{n-i+1} := v/e_i$;

Let B_{i+1} denote $\frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{Z}^{n \times n}$.

od

3. **[Compute Inverse and Assay Correctness]**

if $\text{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, A) \neq 0_{n \times n}$ **then return** Fail **fi**;

$C_0 := \text{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, I_n)$;

$N_1 := p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal s.t. $p^k s \geq 2\alpha s + 2$;

$C_1 := \text{Rem}(s \text{Rem}(A^{-1}, N_1), N_1)$, computed using p -adic lifting;

$C := \text{Rem}(N_1 \text{Rem}(1/N_1, s)C_0 + s \text{Rem}(1/s, N_1)C_1, N_1 s)$;

$N_2 := p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal s.t. $p^k \geq (2n/s) \|A\| \|C\| + 2$;

if $\text{Rem}(A \text{Rem}((1/s)C, N_2), N_2) \neq I_n$ **then return** Fail **fi**;

return $(1/s)C$, with fractions reduced

Figure 5: Algorithm IntInverse

An inspection of the proof of [5, Theorem 2.1] reveals that the definition of M in Lemma 7 is driven by the size of s_n , the largest invariant factor of A , and is otherwise independent of $\|A\|$ and n . Since s_n is a factor of $\det A$, we have $s_n \leq \det A \leq n^{n/2}\|A\|^n$, the second inequality being implied by Hadamard's bound. The next result follows as a corollary of the proof of [5, Theorem 2.1].

Corollary 8. *Let $s \in \mathbb{Z}_{>0}$ satisfy the bound $s \leq n^{n/2}\|A\|^n$. For any $B \in \mathbb{Z}^{n \times n}$, if $X \in L^{n \times 2}$ is chosen uniformly and randomly, then $\gcd(BX, s) = \gcd(B, s)$ with probability at least $1/3$.*

The main computational tool in our algorithm is linear p -adic lifting [3, 15], used to compute $\text{Rem}(A^{-1}X, p^k)$ for a given $X \in \mathbb{Z}^{n \times m}$ and k . The presentation and analysis in [3, 15] assumes standard integer arithmetic, but if the bitlength of p is well chosen then fast integer arithmetic can be incorporated without much difficulty by appealing to fast algorithms for arithmetic operations such as radix conversion. We refer to [16, Section 5] for a derivation of parts (a) and (b) of the following.

Lemma 9. *Let $A \in \mathbb{Z}^{n \times n}$ be nonsingular. Suppose we have $p \in \mathbb{Z}_{>0}$ with $p \perp \det A$ and $\log p \in \Theta(\log n + \log \|A\|)$, together with $C := \text{Rem}(A^{-1}, p)$.*

- (a) *If $N := p^k$ and $Y \in \mathbb{Z}^{n \times m}$ satisfies $\log \|Y\| \in O(\log N)$, then $\text{Rem}(A^{-1}Y, N)$ can be computed with $O(n^2km \mathbf{B}(\log n + \log \|A\|) + nm \mathbf{B}(k(\log n + \log \|A\|)))$ bit operations.*
- (b) *If $Y \in \mathbb{Z}^{n \times m}$ satisfies $\log \|Y\| \in O(n(\log n + \log \|A\|))$ then $A^{-1}Y$ can be computed with $O(n^3m \mathbf{B}(\log n + \log \|A\|))$ bit operations.*
- (c) *Consider recipe (14). If $\prod_{j=1}^{i-1} s_{n-i+1} \leq n^{n/2}\|A\|^n$ and $\log N \in O(\log s_{n-i+2} + \log n + \log \|A^{-1}\|)$, then $\text{Rem}(B_i Y, N)$ can be computed in*

$$O(n^2 k_{n-i+2} m \mathbf{B}(\log n + \log \|A\|) + nm \mathbf{B}(n(\log n + \log \|A\|)))$$

bit operations, where

$$k_{n-i+2} := 1 + \frac{\log s_{n-i+2}}{\log n + \log \|A\|} + \frac{\log \kappa(A)}{\log n + \log \|A\|}$$

with $\kappa(A) := \|A^{-1}\| \|A\|$.

Proof. For a detailed derivation of parts (a) and (b) we refer to [16, Section 5]. To understand part (c) it will be helpful to recall the cost analysis of part (a). Each step of p -adic lifting involves two matrix \times matrix products: $A \times *$ and $C \times *$ where $*$ is an $n \times m$ matrix with entries bounded in bitlength by $O(\log p)$. This gives rise to the $O(n^2km \mathbf{B}(\log n + \log \|A\|))$ term. The second term bounds the cost of all the additional work, especially, at the end of the lifting, the radix conversion from p -adic to standard representation of nm numbers bounded in bitlength by $\log N$.

Now consider part (c). Algorithm OPM computes W_2 in $O(nm \mathbf{M}(n(\log n + \log \|A\|)))$ bit operations (Lemma 4). This also bounds the cost of the arithmetic operations performed

in the return statement of recipe (14). The cost of computing $W_1 := \text{Rem}(A^{-1}Y, N)$ is as stated in (a) with $k = \log_p N$. The definition of k_{n-i+2} is such that $k_{n-i+2} \geq 1$ and $k_{n-i+2} \in \Theta(\log_p N)$. The cost estimate stated in part (c) simplifies the second term of the cost estimate from part (a) by using $k_{n-i+2} \in O(n)$. \square

The usual extended euclidean problem takes as input a column vector $v \in \mathbb{Z}^{n \times 1}$ and asks for a row vector $x \in \mathbb{Z}^{1 \times n}$ such that $xv = \gcd(v)$. The next lemma recalls a method for solving deterministically a variation of the problem when the input is a matrix.

Lemma 10. *Given $V \in \mathcal{R}(\mathbb{Z}, s)^{n \times m}$, an $x \in \mathcal{R}(\mathbb{Z}, s)^{1 \times n}$ and $y \in \mathcal{R}(\mathbb{Z}, s)^{m \times 1}$ such that $\gcd(xVy, s) = \gcd(V, s)$ can be computed in $O(nm \mathbf{B}(\log s))$ bit operations.*

Proof. In [23, Section 4.1] a method is derived to compute a y such that $\gcd(Vy, s) = \gcd(V, s)$. The cost of the method is $O(nm \mathbf{B}(\log s))$ bit operations, plus m calls to a subroutine that takes as input integers a and b , and computes a c such that $\gcd(a + bc, s) = \gcd(a, b, c)$: each such c can be computed in $O(\mathbf{B}(\log s))$ bit operations using operation `Stab` from [26]. Once y has been computed, compute x to be a solution to the standard extended euclidean problem with input Vy . \square

Phase 1

Phase one uses randomization to compute three numbers probabilistically: a small prime power p that is relatively prime to $\det A$; an α that satisfies $\|A^{-1}\| \leq \alpha \leq n\|A^{-1}\|$; the largest invariant factor $s = s_n$ of the input matrix. Part (a) of the following lemma gives some properties that are guaranteed to hold by construction, while part (b) gives properties that only hold probabilistically. This distinction between properties that are guaranteed to hold versus properties that only hold probabilistically is important for our cost analysis of phases 2 and 3 of the algorithm.

Lemma 11. *Phase 1 of algorithm `IntInverse` uses $O(n(\log n + \log \log \|A\|))$ random bits and completes in $O(n^3 \mathbf{B}(\log n + \log \|A\|))$ bit operations. `Fail` will be returned with probability at most $1/12$. If `Fail` is not returned, then (a) will hold, and (b) will hold with overall probability at least $1 - 1/6$.*

(a) $p \perp \det A$, $\log p \geq \log \sqrt{n} + \log \|A\|$ and $\log p \in \Theta(\log n + \log \|A\|)$, $\alpha \leq n\|A^{-1}\|$, and s is a divisor of the largest invariant factor of A .

(b) $\alpha \geq \|A^{-1}\|$, and s is the largest invariant factor of A .

Proof. By Hadamard's bound $|\det A| \leq n^{n/2} \|A\|^n$, so $\det A$ is divisible by at most the log of this many distinct prime divisors. This shows that the randomly chosen prime \bar{p} will be a divisor of $\det A$ with probability at most $1/12$. Assume henceforth that `Fail` has not been returned.

First consider part (a). From the prime number theorem we know that $\log \bar{p} \in \Theta(\log n + \log \log \|A\|)$; the upper bound and lower bounds for $\log p$ follow by construction. The upper

bound for α uses $\|X\| = 1$: $\|A^{-1}X\| \leq n\|A^{-1}\|\|X\| = n\|A^{-1}\|$. In [1] it is shown that s is a factor of s_n .

Now consider part (b). We will separately bound the probability that that $\alpha < \|A^{-1}\|$ by $1/16$, and that s is a proper divisor of the largest invariant factor of A by $64/729$. Since the sum of these two probabilities is less than $1/6$, the claim in the lemma that part (b) will hold with probability at least $1 - 1/6$ will follow.

To see that $\alpha < \|A^{-1}\|$ with probability at most $1/16$, let $x = [x_1 \mid \bar{x}]^T \in \{-1, 1\}^{n \times 1}$ be chosen uniformly and randomly, and let $a = [a_1 \mid \bar{a}] \in \mathbb{Q}^{1 \times n}$ be a row of A^{-1} which has a maximal magnitude entry, which without loss of generality we will assume is the principal entry a_1 . Consider the dot product $ax = a_1x_1 + \bar{a}\bar{x}$. A sufficient condition for $|ax| \geq \|A^{-1}\|$ to hold is that $\text{sign}(a_1x_1) = \text{sign}(\bar{a}\bar{x})$; since x_1 is chosen uniformly and randomly from $\{-1, 1\}$, the probability that $|ax| < \|A^{-1}\|$ is less than $1/2$. Choosing $X \in \{-1, 1\}^{n \times 4}$ to have four columns as in the algorithm gives four independent trials of this idea, so $\alpha < \|A^{-1}\|$ with probability less than $1/2^4$.

Now consider s . Since X is chosen from $L^{n \times (2 \times 6)}$, the probability that s is a proper divisor of the largest invariant factor of A is bounded by the probability that 6 independent trials of the approach of Lemma 7 all fail: this is bounded by $(2/3)^6 = 64/729$.

Now consider the running time. The first $12 \lfloor \log_2(n^{n/2} \|A\|^n) \rfloor$ primes can be constructed in the allotted time using the sieve of Eratosthenes (see [14, Section 4.5.4]). The modular inverse C can be computed in the allotted time using Gaussian elimination. By Lemma 11, part (b), the rational system solutions $A^{-1}X$ can be computed in the allotted time using p -adic lifting. \square

Phase 2

Phase 2 of algorithm `IntInverse` adapts algorithm `OuterProductAdjoint` shown in Figure 3 to the integer case. The main change is that at iteration i we recover the gcd e_i of all entries in B_i probabilistically by computing $\text{gcd}(B_i Y, s_{n-i+2})$ for a randomly chosen Y . Not only might some of the e_i be computed incorrectly, phase 2 also depends on the numbers s and α that are computed probabilistically in phase 1 (cf. part (b) of Lemma 11). For this reason, we need to bound the running time of phase 2 independently of the correctness of s , α , and the e_i computed at each loop iteration.

Lemma 12. *Phase 2 of algorithm `IntInverse` uses $O(n^2(\log n)(\log n + \log \log \|A\|))$ random bits and completes in $O(n^2(\log n)\mathbf{B}(n(\log n + \log \|A\|)) + n^3(\log n + \log \kappa(A))\mathbf{B}(\log n + \log \|A\|))$ bit operations.*

Proof. Each loop iteration checks that $\prod_{j=1}^i s_{n-j+1} \leq n^{n/2} \|A\|^n$, so the sum of the bitlengths of the moduli s_{n-i+2} over all loop iterations is bounded by $O(n(\log n + \log \|A\|))$. Excluding the calls to recipe (14), a total cost bound of $O(nm \mathbf{B}(n(\log n + \log \|A\|)))$ bit operations for the entire phase follows from the superlinearity of \mathbf{B} : $\sum_{j=1}^{i-1} \mathbf{B}(\log s_{n-j+1}) \in O(\mathbf{B}(\sum_{j=1}^{i-1} \log s_i))$.

The cost of the two calls to recipe (14) for single loop iteration is given by part (c) of Lemma 11. Let k_{n-i+2} be defined as in Lemma 11. Then

$$\sum_{j=1}^{i-1} k_{n-i+2} \in O\left(n + \frac{n \log \kappa(A)}{\log n + \log \|A\|}\right).$$

The cost bound in the lemma incorporates the simplification $\log n + \log \|A\| \geq \log n$. \square

Lemma 13. *If s is the largest invariant factor of A and $\alpha \geq \|A^{-1}\|$, then phase 2 of algorithm `IntInverse` will not return `Fail` and computes a correct outer product adjoint formula for A with probability at least $1 - 1/4$.*

Proof. Phase 2 is identical to algorithm `OuterProductAdjoint` shown in Figure 3 except that e_i is computed probabilistically. BY Theorem 5, recipe (14) will correctly compute V . Corollary 8 gives that $e_i \neq \gcd(B_i, s)$ with probability at most $(2/3)^{m/2}$. Since m is chosen so that $(2/3)^{m/2} \leq 1/(4n)$, the probability that $e_i \neq \gcd(B_i, s_{n-i+2})$ for one or more i is at most $1/4$. \square

Phase 3

Lemma 14. *If s is the largest invariant factor of A , $\alpha \geq \|A^{-1}\|$, and the quantities $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}$ computed in phase 2 comprise an outer product adjoint formula for A , then phase 3 of algorithm `IntInverse` will not return `Fail`.*

Proof. Assume that $s = s_n$, $\alpha \geq \|A^{-1}\|$ and $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}$ is an outer product adjoint formula, as specified in the lemma. Then the first call to `OPM` computes $\text{Rem}(sA^{-1}A, s)$ which will be the zero matrix and the second call computes $C_0 = \text{Rem}(sA^{-1}, s)$. The matrix C_1 is equal to $\text{Rem}(sA^{-1}, N_1)$. The matrix C is obtained by Chinese remaindering together C_0 and C_1 to get $C = \text{Rem}(sA^{-1}, N_1s)$. Since $\|sA^{-1}\| \leq s\alpha$, and $N_1s \geq 2\alpha s + 2$, by Lemma 1 we have $sA^{-1} = \text{Rem}(sA^{-1}, N_1s)$ and we conclude that $C = sA^{-1}$. It follows that the second last line will not return `Fail` since $(1/s)AC$ is indeed the identity matrix. \square

Lemma 15. *If phase 3 of algorithm `IntInverse` does not return `Fail`, then the correct inverse of A is returned.*

Proof. If the first call to `OuterProductAdjoint` does not return `Fail`, then C_0 as computed in the next line has the property that $\text{Rem}(AC_0, s)$ is the zero matrix. By construction of C , $\text{Rem}(C, s) = C_0$, so $\text{Rem}(AC, s)$ is also the zero matrix: we conclude that AC is divisible by s . An *a priori* bound for $\|(1/s)AC\|$ is $(n/s)\|A\|C\|$. By Lemma 1, the choice of N_2 guarantees that $(1/s)AC = \text{Rem}((1/s)AC, N_2)$. If $(1/s)AC = I_n$ then $(1/s)C$ is indeed the inverse of A . \square

Lemma 16. *Phase 3 of algorithm `IntInverse` completes in $O(n^2 \mathbf{B}(n(\log n + \log \|A\|)) + n^3(\log \kappa(A))/(\log n + \log \|A\|) \mathbf{B}(\log n + \log \|A\|))$ bit operations.*

Proof. The cost of the two calls to algorithm `OuterProductAdjoint` is $O(n^2 \mathbf{B}(n(\log n + \log \|A\|)))$ bit operations (Lemma 3). Since $\alpha \leq n\|A^{-1}\|$ and $s \leq n^{n/2}\|A\|^n$, all of N_1 , N_1s and N_2 will have bitlength bounded by $O(n(\log n + \log \|A\|))$. Thus, the cost of computing C , $\text{Rem}((1/s)C, N_2)$, and reducing the fractions in $(1/s)C$ is bounded by $O(n^2 \mathbf{B}(n(\log n + \log \|A\|)))$ bit operations as well.

The costs of computing $\text{Rem}(A^{-1}, N_1)$ via lifting and computing the matrix product $A \text{Rem}((1/s)C, N_2)$ are sensitive to α . In particular, both $\log N_1$ and $\log N_2$ are $O(\log n + \log \kappa(A))$. By Lemma 11, C_1 can be computed in

$$O(n^3(\log \kappa(A))/(\log n + \log \|A\|)\mathbf{B}(\log n + \log \|A\|)) \quad (15)$$

bit operations. Now consider the matrix product $A \text{Rem}((1/s)C, N_2)$. First we convert $\text{Rem}((1/s)C, N_2)$ to a p -adic expansion with $O((\log \kappa)/(\log n + \log \|A\|))$ terms. Multiplying each term by A has the same cost as in (15). Convert back to standard representation. The cost of the radix conversion is bounded by $O(n^2 \mathbf{B}(n(\log n + \log \|A\|)))$ bit operations. \square

Theorem 17. *Let $A \in \mathbb{Z}^{n \times n}$ be nonsingular. Algorithm `IntInverse` uses $O(n^2(\log n)(\log n + \log \log \|A\|))$ random bits and completes in $O(n^2(\log n)\mathbf{B}(n(\log n + \log \|A\|)) + n^3(\log n + \log \kappa(A))\mathbf{B}(\log n + \log \|A\|))$ bit operations. The algorithm either returns `Fail` or A^{-1} . `Fail` is returned with probability less than $1/2$.*

Proof. Lemma 15 guarantees that an incorrect result will not be returned. By Lemma 14, the algorithm will not return `Fail` provided that phase 1 does not return fail and computes s and α correctly, and that phase 2 computes a correct outer product adjoint formula. Summing the probabilities from Lemmas 11 and Lemma 13 gives $1/12 + 1/6 + 1/4 \leq 1/2$. The dominant cost is phase 2 (Lemma 24). \square

Corollary 18. *There exists a Las Vegas algorithm that computes the exact inverse of a nonsingular $A \in \mathbb{Z}^{n \times n}$ using an expected number of $O(n^3(\log \|A\| + \log \kappa(A)))$ bit operations.*

6 Computing the inverse of a polynomial matrix

Let $A \in \mathbb{K}[z]^{n \times n}$ be nonsingular with degree $d > 0$, \mathbb{K} a field. Recall that the degree of the determinant of A is bounded by nd , while entries in the adjoint matrix $A^{\text{adj}} := (\det A)A^{-1}$ are minors of dimension $n - 1$ and thus are bounded in degree by $(n - 1)d$. Similar to the integer case, the cost of computing an outer product adjoint formula for A will be sensitive to the difference between $\deg A^{\text{adj}}$ and $\deg \det A$. Generically, $\deg A^{\text{adj}} - \deg \det A = (n - 1)d - nd < 0$, but we may have $\deg A^{\text{adj}} - \deg \det A \leq (n - 1)d$, the upper bound being achieved for certain unimodular matrices.

Fortunately, we can apply some simple algebraic preconditioning techniques to transform the original A into another matrix of degree d that has determinant of degree nd . Consider

PolyInverse(A, Λ)

Input: • nonsingular $A \in \mathbb{K}[z]^{n \times n}$ of degree $d > 0$, \mathbb{K} a field
 • a set Λ of elements of \mathbb{K}

Output: A^{-1} or Fail. Fail will be returned with probability $\leq 4nd/\#\Lambda$.

1. **[Initialization]**

if $\det \text{Rem}(A, z) \neq 0$ **then**

$\alpha := 0$; $A_1 := A$;

else

 Choose α uniformly and randomly from Λ ;

$A_1 := A|_{z=z+\alpha}$; **if** $\det \text{Rem}(A_1, z) = 0$ **then** Fail **fi**;

fi;

$A_2 := z^d (A_1|_{z=1/z})$;

 Choose β uniformly and randomly from Λ ;

if $\det \text{Rem}(A_2, z - \beta) = 0$ **then** Fail **fi**;

$p := (z - \beta)^d$; $C := \text{Rem}(A_2^{-1}, p)$;

 Choose $y \in \Lambda^{n \times 1}$ uniformly and randomly;

$s :=$ the denominator of $A_2^{-1}y$;

2. **[Compute Outer Product Adjoint Formula]**

Let B_1 denote sA_2^{-1} and $s_{n+1} = s$.

for i **to** n **do**

if $i > 1$ **then** choose $y \in \Lambda^{n \times 1}$ uniformly and randomly **fi**;

 Set $N := p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal s.t. $\deg p^k \geq \deg s_{n-i+2}$;

 Compute $v := \text{Rem}(B_i y, s_{n-i+2})$ using recipe (14);

 Compute $x \in \mathcal{R}(\mathbb{K}[z], s_{n-i+2})^{1 \times n}$ such that $\gcd(xv, s_{n-i+2}) = \gcd(v, s_{n-i+2})$;

 Compute $u := \text{Rem}(B_i^T x^T, s_{n-i+2})^T$ using recipe (14);

$e_i := \text{Rem}(xv, s_{n-i+2})$; **if** $e_i = 0$ **then** break **else** $s_{n-i+1} := s_{n-i+2}/e_i$ **fi**;

if ($i = 1$ **and** $s_n \neq s$) **or** $\sum_{j=1}^i s_{n-j+1} > nd$ **or** $e_i \nmid u$ **then** return Fail **fi**;

$u_{n-i+1} := u/e_i$; $v_{n-i+1} := v/e_i$;

 Let B_{i+1} denote $\frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathbb{K}[z]^{n \times n}$.

od;

if $\sum_{j=1}^{i-1} \neq nd$ **then** Fail **fi**;

3. **[Compute Inverse and Assay Correctness]**

if $\text{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, v_2) \neq 0_{n \times n}$ **then** return Fail **fi**;

$C := \text{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}, I_n)$;

if $\text{Rem}(A_2 \text{Rem}((1/s)C, p), p) \neq I_n$ **then** return Fail **fi**;

return $((z^d/s)C)|_{z=1/(z-\alpha)}$, with fractions reduced

Figure 6: Algorithm PolyInverse

the following input matrix over $\mathbb{K}[z]$ where $\mathbb{K} = \mathbb{Z}/\langle 97 \rangle$.

$$A_1 = \begin{bmatrix} 72z^2 + 37z + 74 & 87z^2 + 44z + 29 & 7z + 56 \\ 89z^2 + 68z + 95 & 11z^2 + 48z + 50 & 64z + 75 \\ 87z^2 + 31z + 46 & 77z^2 + 95z + 1 & 63z + 8 \end{bmatrix}.$$

The determinant of A_1 is 17. But consider the matrix obtained from A_1 by reverting the polynomials via a change of variables.

$$A_2 = z^2(A_1|_{z=1/z}) = \begin{bmatrix} 74z^2 + 37z + 72 & 29z^2 + 44z + 87 & 56z^2 + 7z \\ 95z^2 + 68z + 89 & 50z^2 + 48z + 11 & 75z^2 + 64z \\ 46z^2 + 31z + 87 & z^2 + 95z + 77 & 8z^2 + 63z \end{bmatrix}.$$

The leading coefficient matrix of A_2 (the coefficient of z^2) is equal to the constant coefficient matrix of A_1 : since this is nonsingular the degree of the determinant of A_2 will be equal to nd . Indeed, $\det A_2 = 17z^6$. If we start with an A that does not have a nonsingular constant coefficient matrix we apply a shift to produce $A_1 := A|_{z=z+\alpha}$ for a randomly chosen $\alpha \in \mathbb{K}$. These ideas are summarized in the following lemma, the proof of which is elementary.

Lemma 19. *Let $A \in \mathbb{K}[z]^{n \times n}$ with degree bounded by d and let $\alpha \in \mathbb{K}$. If $A_1 := A|_{z=z+\alpha}$ and $A_2 := z^d(A_1|_{z=1/z})$ then $A_1^{-1} = (z^d(A_2^{-1}))|_{z=1/z}$ and $A^{-1} = (A_1^{-1})|_{z=z-\alpha}$. We also have $\det A_2 = z^{nd}((\det A_1)|_{z=1/z})$, so if $\det \text{Rem}(A_1, z) \neq 0$ then $\deg \det A_2 = nd$.*

In addition to a nonsingular input $A \in \mathbb{K}[z]^{n \times n}$, our algorithm for computing the inverse will take as input a subset Λ of elements of \mathbb{K} . Probability estimates will depend on $\#\Lambda$. The following result is the polynomial analogue of Corollary 8.

Lemma 20. *Let $s \in \mathbb{K}[z]$ be nonzero. For any $B \in \mathbb{K}[z]^{n \times n}$, if $y \in \Lambda^{n \times 1}$ is chosen uniformly and randomly, then $\gcd(By, s) = \gcd(B, s)$ with probability at least $1 - (\deg s)/\#\Lambda$.*

Proof. Let $g = \gcd(B, s)$. Then $\gcd(By, s) = \gcd(B, s)$ if and only if $\gcd((B/g)y, s) = 1$. Let p be an irreducible divisor of s . Then the residue class ring $\mathbb{K}[z]/\langle p \rangle$ is a field and it is easy to see that $\gcd((B/g)y, p) = 1$ with probability at least $1 - 1/\#\Lambda$. The result follows by noting that s , and hence also s/g , has at most $\deg s$ distinct irreducible divisors. \square

The next result follows from Lemma 20 since the largest invariant factor s_n of A has degree bounded by nd . In particular, $A^{-1}y$ has denominator s_n if and only if $\gcd((s_n A^{-1})y, s_n) = 1$.

Corollary 21. *Let $A \in \mathbb{K}[z]^{n \times n}$ be nonsingular of degree d . If $y \in \Lambda^{n \times 1}$ is chosen uniformly and randomly, then the denominator of $A^{-1}y$ is equal to the largest invariant factor of A with probability at least $1 - nd/\#\Lambda$.*

We refer to [16, Section 5] for a derivation of part (a) of the next lemma. The derivation of part (b) is similar to that of part (c) of Lemma 11. The $n\mathbf{B}(nd)$ term comes from the application of algorithm OPM. The $n^2 k_{n-i+2} \mathbf{B}(d)$ term captures the cost of performing $O(k_{n-i+2})$ lifting steps to obtain $\text{Rem}(A^{-1}y, N)$.

Lemma 22. Let $A \in \mathbb{K}[z]^{n \times n}$ be nonsingular of degree d . Suppose we have $p \in \mathbb{K}[z]_{>0}$ with $p \perp \det A$ and $\deg p = d$, together with $C := \text{Rem}(A^{-1}, p)$.

- (a) If $y \in \mathbb{K}[z]^{n \times 1}$ satisfies $\deg y \in O(nd)$, then $A^{-1}y$ can be computed with $O(n^3 \mathbf{B}(d))$ field operations from \mathbb{K} .
- (b) Consider recipe (14). If $\sum_{j=1}^{i-1} \deg s_{n-i+1} \leq nd$ and $\deg N \in O(\deg s_{n-i+2})$, then $\text{Rem}(B_i y, N)$ can be computed in $O(n^2 k_{n-i+2} \mathbf{B}(d) + n \mathbf{B}(nd))$ field operations from \mathbb{K} , where $k_{n-i+2} := 1 + (\deg s_{n-i+2})/d$.

Algorithm `PolyInverse` is shown in Figure 6.

Phase 1

Lemma 23. Phase 1 of algorithm `PolyInverse` completes in $O(n^3 \mathbf{B}(nd))$ field operations. `Fail` is returned with probability at most $2nd/\#\Lambda$. If `Fail` is not returned, then $\deg \det A_2 = nd$ and s is the largest invariant factor of A with probability at least $1 - nd/\#\Lambda$.

Proof. Since $\deg \det A \leq nd$, the randomly chosen shift α is a root of $\det A$ with probability at most $nd/\#\Lambda$. Similarly, $\det \text{Rem}(A_2, t - \beta) = 0$ with probability at most $nd/\#\Lambda$.

Now assume that `Fail` is not returned. That $\deg \det A_2 = nd$ follows from Lemma 19. Corollary 21 gives the probability estimate for the correctness of s . By Lemma 22, the computation of $A^{-1}y$, which dominates the cost of the phase, can be accomplished in the allotted time. \square

Note that the second probability estimate in Lemma 23 is conditional. In other words, the probability that phase does not return `fail` and s is computed correctly is at least $1 - 2nd/\#\Lambda - nd/\#\Lambda$.

Phase 2

Lemma 24. Phase 2 of algorithm `PolyInverse` completes in $O(n^2 \mathbf{B}(nd))$ field operations.

Proof. Each iteration check that $\sum_{j=1}^i \deg s_{n-j+1} \leq nd$, so the sum of the degrees of the moduli s_{n-i+2} over all the loop iterations is bounded by $O(nd)$. Excluding the calls to recipe (14), a total cost bound of $O(n \mathbf{B}(nd))$ field operations for the entire phase follows from the superlinearity of \mathbf{B} .

The cost of the two calls to recipe (14) for a single loop iterations is given by part (b) of Lemma 22. Let k_{n-i+2} be as defined in Lemma 22. Then $\sum_{j=1}^{i-1} k_{n-i+2} \in O(n)$. The overall cost of all calls to recipe (14) is thus $O(n^2 \mathbf{B}(nd))$ field operations. \square

Lemma 25. If s is the largest invariant factor of A_2 , then phase 2 of algorithm `PolyInverse` computes an outer product adjoint formula for A with probability at least $1 - nd/\#\Lambda$.

Proof. Phase 2 is identical to algorithm `OuterProductAdjoint` shown in Figure 3 except that e_i is computed probabilistically for $i > 1$. By assumption $s = s_n$, so during the first iteration the modulus s_{n+1} and $B_1 = sA_2^{-1}$ are correct. The first iteration will correctly compute $e_1 = 1$ since the vector y is reused from phase 1. Suppose $s_{n+1}, s_n, \dots, s_{n-i+2}$ and hence B_1, B_2, \dots, B_i are computed correctly up to some i . By Corollary 6, recipe (14) will correctly compute v . By Lemma 20, the probability that e_i is computed incorrectly is bounded by $(\deg s_{n-i+2})/\#\Lambda$. Since $\sum_{j=2}^i \deg s_{n-i+2} \leq \sum_{j=2}^n \deg s_i \leq nd$, the sum of the failure probability over all loop iterations is bounded by $nd/\#\Lambda$. \square

Phase 3

Lemma 26. *If s is the largest invariant factor of A , and phase 2 computes a correct outer product adjoint formula for A , then phase 3 of algorithm `PolyInverse` will not return `Fail`.*

Proof. Assume that $s = s_n$ and $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1 \leq j \leq i-1}$ is an outer product adjoint formula for A_2 , as specified in the lemma. The first call to `OPM` computes $\text{Rem}(sA^{-1}, s)$ which will be the zero matrix. The second call computes $C = \text{Rem}(sA_2^{-1}, s)$. Since $\deg \det A_2 = nd$, we have $\deg sA_2^{-1} < \deg s$ and by Lemma 1 we conclude that $C = sA_2^{-1}$. It follows that the second last line will not return `Fail` since $A_2(1/s)C = I_n$. \square

Lemma 27. *If phase 3 of algorithm `PolyInverse` does not return `Fail`, then the correct inverse of A is returned.*

Proof. If the first call to `OuterProductAdjoint` does not return `fail`, then C as computed in the next line has the property that $\text{Rem}(A_2C, s)$ is the zero matrix. Thus A_2C is divisible by s . An *a priori* upper bound for the degree of $(1/s)A_2C$ is $(\deg A_2) - 1$, so if the second last line does not return `Fail`, then $(1/s)C$ is indeed the inverse of A . \square

Theorem 28. *Let $A \in \mathbb{K}[z]^{n \times n}$ be nonsingular with degree d . Algorithm `PolyInverse` completes in $O(n^2 \mathbf{B}(nd))$ field operations. The algorithm either returns `Fail` or A^{-1} . `Fail` is returned with probability $4nd/\#\Lambda$.*

To ensure of positive probability of success we require that $\#\mathbb{K}$ is large enough. If \mathbb{K} is too small we can work over an algebraic extension of degree $O(\log nd)$ over \mathbb{K} . The cost bound of Theorem 28 increases by some factors of $\log nd$.

Corollary 29. *Let \mathbb{K} be a field and z an indeterminate. There exists a Las Vegas algorithm that computes the exact inverse of a nonsingular $A \in \mathbb{K}[z]^{n \times n}$ using an expected number of $O(n^3 \deg A)$ field operations from \mathbb{K} .*

7 Conclusions

The main outstanding problem is to remove the dependence of the complexity of the integer matrix inversion algorithm on $\kappa(A)$. A promising approach is to use additive preconditioning as described in [18].

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] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1996.
- [3] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.
- [4] P. D. Domich, R. Kannan, and L. E. Trotter, Jr. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1):50–59, 1987.
- [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] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2 edition, 2003.
- [7] M. Giesbrecht. Fast computation of the Smith normal form of an integer matrix. In A. Levelt, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '95*, pages 110–118. ACM Press, New York, 1995.
- [8] J. L. Hafner and K. S. McCurley. Asymptotically fast triangularization of matrices over rings. *SIAM Journal of Computing*, 20(6):1068–1083, Dec. 1991.
- [9] C. S. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM Journal of Computing*, 18(4):658–669, 1989.
- [10] C. P. Jeannerod and G. Villard. Essentially optimal computation of the inverse of generic polynomial matrices. *Journal of Complexity*, 21:72–86, 2005.
- [11] 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.
- [12] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3–4):91–130, 2004.
- [13] I. Kaplansky. Elementary divisors and modules. *Trans. of the Amer. Math. Soc.*, 66:464–491, 1949.

- [14] D. E. Knuth. *The Art of Computer Programming, Vol.2, Seminumerical Algorithms*. Addison–Wesley (Reading MA), 2 edition, 1981.
- [15] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proc. EUROSAM '79, volume 72 of Lecture Notes in Compute Science*, pages 65–72, Berlin-Heidelberg-New York, 1979. Springer-Verlag.
- [16] T. Mulders and A. Storjohann. Certified dense linear system solving. *Journal of Symbolic Computation*, 37(4):485–510, 2004.
- [17] M. Newman. *Integral Matrices*. Academic Press, 1972.
- [18] V. Pan, R. R. D. Ivolgin, B. Murphy, Y. Tang, and X. Yan. Additive preconditioning for matrix computations. In *Third International Computer Science Symposium in Russia, CSR 2008 Moscow, Russia, June 7-12, 2008 Proceedings*, LNCS 5010, pages 327–383. Springer Verlag, 2008.
- [19] V. Y. Pan and X. Wang. On rational number reconstruction and approximation. *SIAM Journal of Computing*, 33(2):502–503, 2004.
- [20] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.
- [21] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
- [22] A. Storjohann. Near optimal algorithms for computing Smith normal forms of integer matrices. In Y. N. Lakshman, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '96*, pages 267–274. ACM Press, New York, 1996.
- [23] A. Storjohann. A solution to the extended gcd problem with applications. In W. W. Küchlin, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '97*, pages 109–116. ACM Press, New York, 1997.
- [24] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Swiss Federal Institute of Technology, ETH–Zurich, 2000.
- [25] 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.
- [26] A. Storjohann and T. Mulders. Fast algorithms for linear algebra modulo N . In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms — ESA '98*, LNCS 1461, pages 139–150. Springer Verlag, 1998.