ON THE COMPLEXITY OF INVERTING INTEGER AND POLYNOMIAL MATRICES

Arne Storjohann

October 6, 2014

Abstract. An algorithm is presented that probabilistically computes the exact inverse of a nonsingular $n \times n$ integer matrix A using $(n^3(\log ||A|| + \log \kappa(A)))^{1+o(1)}$ bit operations. Here, $||A|| = \max_{ij} |A_{ij}|$ denotes the largest entry in absolute value, $\kappa(A) := n||A^{-1}||\,||A||$ is the condition number of the input matrix and the "$+o(1)$" in the exponent indicates a missing factor $c_1(\log n)^{c_2}(\text{loglog}\,||A||)^{c_3}$ for positive real constants c_1, c_2, c_3. A variation of the algorithm is presented for polynomial matrices that computes the inverse of a nonsingular $n \times n$ matrix whose entries are polynomials of degree d over a field using $(n^3 d)^{1+o(1)}$ field operations. 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.

Keywords. Integer matrix, polynomial matrix, matrix inverse, Smith normal form, bit complexity, randomized algorithm

Subject classification. 68W30, 15A35

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 $\kappa(A) := n||A||\,||A^{-1}||$ 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 $(n^3(\log ||A|| + \log \kappa(A)))^{1+o(1)}$ 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 $(n^3 \log ||A||)^{1+o(1)}$ 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 two running examples. The first example is based on the matrix

$$A_{\mathsf{well}} = \begin{bmatrix} 133 & 25 & -75 & 192 \\ -165 & -36 & 270 & 72 \\ -246 & -99 & -99 & 198 \\ -60 & 375 & 21 & -150 \end{bmatrix} \tag{1.1}$$

with

$$A_{\mathsf{well}}^{-1} = \begin{bmatrix} \frac{56100}{27010673} & -\frac{7843}{27010673} & -\frac{206069}{81032019} & -\frac{22627}{27010673} \\ \frac{48563}{27010673} & \frac{27298}{81032019} & \frac{17819}{162064038} & \frac{140897}{54021346} \\ \frac{13850}{27010673} & \frac{777005}{243096057} & -\frac{883015}{486192114} & -\frac{33517}{162064038} \\ \frac{201813}{54021346} & \frac{683501}{486192114} & \frac{252293}{243096057} & \frac{25913}{162064038} \end{bmatrix}$$

and $\kappa(A_{\mathsf{well}}) < 7.373$. This example illustrates a property of a well conditioned matrix: for each entry in A_{well}^{-1}, the bitlength of the numerator is less than, or at least not too much larger than, the bitlength of the denominator.

The second example is based on the matrix

$$A_{\mathsf{ill}} = \begin{bmatrix} -195 & -105 & 9 & 242 \\ -387 & -633 & 300 & 508 \\ 63 & -216 & 69 & 19 \\ -399 & -513 & 513 & 260 \end{bmatrix} \tag{1.2}$$

with

$$A_{\mathsf{ill}}^{-1} = \begin{bmatrix} -\frac{3285673}{2394} & \frac{54655}{63} & -\frac{182789}{171} & -\frac{812713}{2394} \\ -\frac{2227201}{2394} & \frac{37048}{63} & -\frac{123905}{171} & -\frac{550897}{2394} \\ -\frac{1503910}{1197} & \frac{50033}{63} & -\frac{167332}{171} & -\frac{371989}{1197} \\ -\frac{8777}{6} & \frac{2774}{3} & -\frac{3418}{3} & -\frac{2171}{6} \end{bmatrix}$$

and $\kappa(A_{\mathsf{ill}}) = 3703894$. This example illustrates a property of an ill conditioned matrix: for some entries in A_{ill}^{-1}, the bitlength of the numerator can be significantly larger than the bitlength of denominator. For ill conditioned input matrices the $n^3 \log \kappa(A)$ term the running time bound for our algorithm can dominate. In the worst case, for example for a unimodular matrix (i.e., with an integral inverse), we can have $\log \kappa(A) = (n \log ||A||)^{1+o(1)}$.

To the best of our knowledge, the best previously known complexity estimate for integer matrix inversion is $(n^{\omega+1} \log ||A||)^{1+o(1)}$ bit operations, supported by any of the classical approaches such as homomorphic imaging and Chinese remaindering (von zur Gathen & Gerhard 2013, Section 5.5), quadratic lifting via Newton iteration, or a recursive version of fraction-free Gaussian elimination (Storjohann 2000, Section 2). Here, ω is the exponent for matrix multiplication over a ring (Bürgisser *et al.* 1996, Chapter 1).

The discovery of an essentially optimal inversion algorithm for generic polynomial matrices by Jeannerod & Villard (2005), and the recent progress made in reducing the complexity of many basic linear algebra problems on integer matrices, motivates us to develop an 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 Kaltofen & Villard (2004) and Storjohann (2005).

It was shown by Dixon (1982) 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 $(n^3 \log ||A||)^{1+o(1)}$ bit operations using linear p-adic lifting. Taking b to be a column of the identity matrix shows that any single column of the inverse can be computed in $(n^3 \log ||A||)^{1+o(1)}$ 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 (von zur Gathen & Gerhard 2013, Section 5.5) require $(n^4 \log ||A||)^{1+o(1)}$ bit operations. A breakthrough is the Krylov approach of Kaltofen (1992) which gives an $(n^{3.5} \log ||A||)^{1+o(1)}$ Las Vegas algorithm. Eberly, Giesbrecht & Villard (2000) describe a Monte Carlo algorithm with running time $(n^{3.5}(\log ||A||)^{1.5})^{1+o(1)}$ to compute not only the determinant but the entire Smith form of A. The Smith form $\mathrm{Diag}(s_1, s_2, \ldots, s_n)$ 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, if k is the number of distinct invariant factors then it can be shown that $k \in O(\sqrt{n(\log n + \log ||A||)})$. The approach of Eberly *et al.* (2000) is based on computing the distinct invariant factors, together with their multiplicities, by computing $k \times (\log n)^{1+o(1)}$ rational linear system solutions, each of which can be accomplished in $(n^3 \log ||A||)^{1+o(1)}$ bit operations using linear p-adic lifting. The overall cost of the algorithm of Eberly *et al.* (2000) 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 exploit the fact that $\sum_{i=1}^{n} \mathrm{bitlength}(s_i) \leq n + \mathrm{bitlength}(\det A)$, 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. The sensitivity of our inversion algorithm to $\log \kappa(A)$ will be explained later in the paper using our running examples A_{well} and A_{ill}.

Next, we motivate our approach for integer matrix inversion by recalling the definition of 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 $A^{\mathsf{adj}} = (\det A)A^{-1}$, but note that the adjoint is well defined also for singular matrices. If the rank of A is strictly less than $n-1$ then the adjoint is the zero matrix. But 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

$$\left[\begin{array}{c|c} I_{n-1} & \\ \hline u & 1 \end{array}\right] \overset{A}{\left[\begin{array}{c|c} C & y \\ \hline x & * \end{array}\right]} \left[\begin{array}{c|c} I_{n-1} & v \\ \hline & 1 \end{array}\right] = \left[\begin{array}{c|c} C & \\ \hline & 0 \end{array}\right]. \tag{1.3}$$

The adjoint of the matrix on the right hand side of (1.3) 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 (1.3) with the adjoint of that side and solving for A^{adj} gives

$$A^{\mathsf{adj}} = \left[\begin{array}{c} v \\ \hline 1 \end{array}\right] (\det C) \left[\begin{array}{c|c} u & 1 \end{array}\right] = \left[\begin{array}{c} (\det C)v \\ \hline \det C \end{array}\right] \left[\begin{array}{c|c} u & 1 \end{array}\right]. \tag{1.4}$$

Note that the expression for A^{adj} in (1.4) 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 $\det C \perp \det A$, where $\perp$ denotes relative primality. Then over the residue class ring $\mathbb{Z}/\langle \det A\rangle$ we have $\det A = 0$ and $\det C$ is a unit, so equation (1.4) holds modulo $\det A$. In other words, the adjoint of A over $\mathbb{Z}$ is element-wise congruent to the rank 1 matrix in (1.4). 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 (1.4) and reducing all entries modulo $\det A$ in the symmetric range $[-\lfloor(|\det A|-1)/2\rfloor, \lfloor|\det A|/2\rfloor]$.

For example, the matrix

$$A = \begin{bmatrix} -1 & 7 & 8 & 1 \\ 3 & -2 & 7 & -4 \\ 7 & -8 & -1 & -1 \\ -6 & 9 & -6 & -4 \end{bmatrix}$$

has $\det A = -2677$, which is relatively prime to $\det C = 226$. Formula (1.4) gives

$$\begin{aligned} A^{\mathsf{adj}} &\equiv \begin{bmatrix} -221 \\ -241 \\ 155 \\ 226 \end{bmatrix} \begin{bmatrix} -1209 & -934 & -154 & 1 \end{bmatrix} \pmod{-2677} \\ (1.5) &\equiv \begin{bmatrix} -511 & 285 & -767 & -221 \\ -424 & 226 & -364 & -241 \\ -5 & -212 & 223 & 155 \\ -180 & 399 & -3 & 226 \end{bmatrix} \pmod{-2677}. \end{aligned}$$

For this example, which is well conditioned, $\det A$ is large enough to capture all entries in A^{adj}; the matrix in (1.5), obtained by multiplying out the outer product and reducing entries in the symmetric range modulo -2677, 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 minors of A of dimension $n-1$ have a common factor with $\det A$ (i.e., the Smith form of A is nontrivial). (Our running example matrices A_{well} and A_{ill} both have nontrivial Smith forms, which necessitated our choice of a different matrix with trivial Smith form for the example in (1.5).) Our solution is to extend (1.4) to an A with nontrivial invariant structure by giving an expression for $A^{\mathsf{adj}} \bmod \det A$ as the sum of scaled outer products.

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

Each v_i is a column vector and each u_i a row vector. We call this construction an outer product adjoint for A. Actually, since all entries in A^{adj} are divisible by $(\det A)/s_n$, our definition of the outer product adjoint in Section 3 is as shown above but with $\det A$ replaced by s_n, and the left hand side replaced with $s_n A^{-1}$, that is,

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

If we let $\mathrm{Rem}(\cdot, s_n)$ denote reduction modulo s_n, then expanding the outer product adjoint yields $\mathrm{Rem}(s_n A^{-1}, s_n)$.

When applying the outer product adjoint to compute $s_n A^{-1}$, a problem occurs if s_n is too small to capture the entries of $s_n A^{-1}$ in the symmetric range modulo $\det A$. In this case we divide the computation of $s_n A^{-1}$ into two parts by using the decomposition $s_n A^{-1} = \mathrm{Rem}(s_n A^{-1}, s_n) + s_n R$.

1. Compute an outer product adjoint for A and then expand to recover $\mathrm{Rem}(s_n A^{-1}, s_n)$ explicitly.

2. Compute R using a classical technique such as p-adic lifting.

The bitlength of entries in R is $O(\log \kappa(A))$, leading directly to the cost $(n^3 \log \kappa(A))^{1+o(1)}$ for part 2. We remark, however, that the cost of our algorithm for part 1 is also sensitive to $\log \kappa(A)$ and has running time $(n^3(\log ||A|| + \log \kappa(A)))^{1+o(1)}$; the sensitivity on $\log \kappa(A)$ of the running time of computing an outer product adjoint is explained in Example 4.8.

We adapt our approach for integer matrix inversion to obtain an algorithm for polynomial matrix inversion. Let K be a field, and let $A \in \mathsf{K}[z]^{n\times n}$ 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 $(n^{\omega+1}d)^{1+o(1)}$ field operations from K. We refer to Jeannerod & Villard (2005) 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

& Villard (2005) propose a new approach that, for a generic A with n a power of two, will compute A^{-1} in $(n^3d)^{1+o(1)}$ field operations. A natural analogue of the condition number for polynomial matrix is $\deg A + \deg A^{-1}$, the sum of the maximal entry degree in A and the maximum over all entries in A^{-1} of the difference of the numerator and denominator degree. A direct application of our approach would lead to an algorithm for polynomial matrix inversion with running time of

$$(n^3d + n^3(\deg A + \deg A^{-1}))^{1+o(1)}$$

field operations, which could be $(n^4d)^{1+o(1)}$ in the worst case. However, by first performing a random shift $z \to z + \alpha$ of the indeterminant and reverting the polynomials in A, we can instead work with $B := z^d((A\,|_{z=z+\alpha})\,|_{z=1/z})$ that has $\deg B + \deg B^{-1} = 0$. We thus obtain a Las Vegas probabilistic algorithm with running time $(n^3d)^{1+o(1)}$ for any nonsingular input.

The rest of this paper is organized as follows. In a paragraph at the end of this section we define our complexity model in terms of some standard cost functions for basic arithmetic operations over $\mathbb{Z}$ and $\mathsf{K}[z]$. Section 2 fixes some notation and recalls some definitions, including that of the Smith canonical form.

Section 3 defines the outer product adjoint and develops a recipe for its computation. Section 4 gives a recipe for a key computational step that the recipe in the previous section leaves unspecified. Sections 3 and 4 develop results generically so they apply over both $\mathsf{R} = \mathbb{Z}$ and $\mathsf{R} = \mathsf{K}[z]$.

Section 5 shows how to use the recipes of the previous sections to obtain a Las Vegas randomized algorithm for integer matrix inversion algorithm. Section 6 gives the adaptation of the inversion algorithm to polynomial matrices. Section 7 concludes.

Cost functions. Let $\mathsf{M} : \mathbb{Z}_{>0} \longrightarrow \mathbb{R}_{>0}$ be such that integers bounded in magnitude by 2^t can be multiplied using at most $\mathsf{M}(t)$ bit operations. The algorithm of Schönhage & Strassen (1971) allows $\mathsf{M}(t) = O(t(\log t)(\operatorname{loglog} t))$, and recently some asymptotically faster algorithm have been developed (De *et al.* 2013; Fürer 2009). We assume that $\mathsf{M}(a) + \mathsf{M}(b) \leq \mathsf{M}(a + b)$ and $\mathsf{M}(ab) \leq \mathsf{M}(a)\mathsf{M}(b)$

for $a, b \in \mathbb{N}_{\geq 2}$. We refer to von zur Gathen & Gerhard (2013, 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 $\mathsf{B}(t) = \mathsf{M}(t) \log t$. Then the extended gcd problem with two integers bounded in magnitude by 2^t, and the rational number reconstruction problem (von zur Gathen & Gerhard 2013, Section 5.10) with modulus bounded by 2^t, can be solved with $O(\mathsf{B}(t))$ bit operations (Schönhage 1971).

We will overload notation slightly and use $\mathsf{M} : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{R}_{>0}$ as a cost function for polynomial multiplication: two polynomials in $\mathsf{K}[z]$ of degree bounded by d can be multiplied using at most $\mathsf{M}(d)$ 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 $\mathsf{B}(d) = \mathsf{M}(d) \log d$. We refer to von zur Gathen & Gerhard (2013, 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 $\mathsf{R} = \mathbb{Z}$ and $\mathsf{R} = \mathsf{K}[z]$. Following Newman (1972), we prescribe a complete set of non-associates $\mathcal{A}(\mathsf{R})$ and, for every nonzero $s \in \mathsf{R}$, a complete set of residues $\mathcal{R}(\mathsf{R}, s)$. For $\mathbb{Z}$ we choose

$$\mathcal{A}(\mathbb{Z}) = \{0, 1, 2, \ldots\} \quad \text{and} \quad \mathcal{R}(\mathbb{Z}, s) = \left[-\left\lfloor \frac{|s| - 1}{2} \right\rfloor, \left\lfloor \frac{|s|}{2} \right\rfloor \right].$$

Note that our choice for $\mathcal{R}(\mathbb{Z}, s)$ corresponds to the usual "symmetric range" modulo s. For $\mathsf{K}[z]$ we choose

$$\mathcal{A}(\mathsf{K}[z]) = \{0\} \cup \{f \in \mathsf{K}[z] \mid f \text{ is monic}\}$$

and

$$\mathcal{R}(\mathsf{K}[z], s) = \{f \in \mathsf{K}[z] \mid \deg f < \deg s\}.$$

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

Lemma 2.1. *Let $a, N \in \mathsf{R}$, N nonzero. Then $a = \mathrm{Rem}(a, N)$ if*

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

For $a, s \in \mathsf{R}$, we denote by $\gcd(a, s)$ the unique principal generator in $\mathcal{A}(\mathsf{R})$ 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 $\mathsf{R}/\langle s\rangle$ 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 $\mathsf{R}/\langle s\rangle$ for a well chosen s. For nonzero $s \in \mathsf{R}$, we identify the residue class ring $\mathsf{R}/\langle s\rangle$ with the set of elements $\mathcal{R}(\mathsf{R}, s)$, and define

$$\mathcal{A}(\mathsf{R}/\langle s\rangle) = \{\gcd(a, s) \mid a \in \mathcal{R}(\mathsf{R}, s)\}$$

and

$$\mathcal{R}(\mathsf{R}/\langle s\rangle, b) = \mathcal{R}(\mathsf{R}, \gcd(b, s)).$$

These choices for $\mathcal{A}$ and $\mathcal{R}$ allow us to easily obtain algorithms for basic operations over $\mathsf{R}/\langle s\rangle$ in terms of algorithms for basic operations over R (Storjohann 2000, Section 1.1).

The Smith canonical form. Corresponding to every $A \in \mathsf{R}^{n\times m}$ there exist unimodular (invertible over R) matrices $U \in \mathsf{R}^{n\times n}$ and $V \in \mathsf{R}^{m\times m}$ such that

$$\mathrm{snf}(A) = S = UAV = \mathrm{Diag}(s_1, s_2, \ldots, s_r, 0, 0, \ldots, 0)$$

with S in Smith form (Newman 1972, Chapter II), that is, with $s_i | s_{i+1}$ for $1 \leq i \leq r-1$ and $s_i \in \mathcal{A}(\mathsf{R})$ for $1 \leq i \leq r$. The Smith form is a canonical form for matrices under left and right multiplication of unimodular matrices.

In this paper we are particularly interested in the case when R is a principal ideal domain (e.g., $\mathsf{R} = \mathbb{Z}$ and $\mathsf{R} = \mathsf{K}[x]$) and

$A \in \mathsf{R}^{n\times n}$ is square and nonsingular. In this case, $r = n$ and s_n is an associate of $(\det A)/\gcd(A^{\mathsf{adj}})$. Thus, the largest invariant factor s_n is the "smallest" nonzero element of R (minimal degree over $\mathsf{K}[z]$ and minimal magnitude over $\mathbb{Z}$) such that $s_n A^{-1}$ is over R.

The classical approach to compute the Smith form of a matrix over the Euclidean domains $\mathsf{R} = \mathbb{Z}$ or $\mathsf{R} = \mathsf{K}[z]$ 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., Domich *et al.* 1987; Hafner & McCurley 1991; Iliopoulos 1989) have used the idea of the following lemma in conjunction with that of Lemma 2.1. The lemma follows from existence and uniqueness of the Smith form over any principal ideal ring (Kaplansky 1949), in particular over the residue class ring $\mathsf{R}/\langle s\rangle$. For a proof of Lemma 2.2 we refer to Storjohann (1996, Theorem 12).

LEMMA 2.2. *Let $A \in \mathsf{R}^{n\times n}$, and let $s \in \mathsf{R}$ be a nonzero multiple of the largest nonzero invariant factor of A. If S is the Smith form of A over R, and $\bar{S}$ is the Smith form of $\bar{A} = \mathrm{Rem}(A, s)$ over $\mathsf{R}/\langle s\rangle$, then $\bar{S} = \mathrm{Rem}(S, s)$.*

3. The outer product adjoint

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

$$\begin{aligned} s_nA^{-1} &= V(s_nS^{-1})U \\ &= \frac{s_n}{s_n}v_nu_n + \frac{s_n}{s_{n-1}}v_{n-1}u_{n-1} + \cdots + \frac{s_n}{s_1}v_1u_1 && (3.1)\\ &= (e_1)v_nu_n + (e_1e_2)v_{n-1}u_{n-1} + \cdots + (e_1e_2\cdots e_n)v_1u_1, && (3.2) \end{aligned}$$

where $e_i = s_{n-i+2}/s_{n-i+1}$, $1 \le i \le n$, $s_{n+1} := s_n$. Note that that the Smith form of s_nA^{-1} is $\mathrm{Diag}(e_1, e_1e_2, \ldots, e_1e_2\cdots e_n)$.

Now consider taking equation (3.1) modulo s_n. Since the 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, $1 \leq i \leq n$.

DEFINITION 3.3. *An* outer product adjoint *of a nonsingular $A \in \mathsf{R}^{n\times n}$ 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* $\mathrm{Diag}(1, 1, \ldots, 1, s_{n-k+1}, s_{n-k+2}, \ldots, s_n)$ *with $s_{n-k+1} \neq 1$,*
- *$v_{n-i+1} \in \mathcal{R}(\mathsf{R}, s_{n-i+1}))^{n\times 1}$ and $u_{n-i+1} \in \mathcal{R}(\mathsf{R}, s_{n-i+1})^{1\times n}$ for $1 \leq i \leq k$, and*
- *$s_n A^{-1} \equiv \frac{s_n}{s_n} v_n u_n + \frac{s_n}{s_{n-1}} v_{n-1} u_{n-1} + \cdots + \frac{s_n}{s_{n-k+1}} v_{n-k+1} u_{n-k+1}$ $(\mathrm{mod}\ s_n)$.*

EXAMPLE 3.4. The example input matrix A_{well} in (1.1) has

$$\mathrm{snf}(A_{\mathsf{well}}) = \mathrm{Diag}(s_1, s_2, s_3, s_4) = \mathrm{Diag}(1, 3, 6, 486192114)$$

and

$$s_4 A_{\mathsf{well}}^{-1} = \begin{bmatrix} 1009800 & -141174 & -1236414 & -407286 \\ 874134 & 163788 & 53457 & 1268073 \\ 249300 & 1554010 & -883015 & -100551 \\ 1816317 & 683501 & 504586 & 77739 \end{bmatrix}.$$

An outer product adjoint for A_{well} is given by

$$v_4 u_4 + (81032019) v_3 u_3 + (162064038) v_2 u_2$$
$$\equiv \begin{bmatrix} 1009800 & -141174 & -1236414 & -407286 \\ 874134 & 163788 & 53457 & 1268073 \\ 249300 & 1554010 & -883015 & -100551 \\ 1816317 & 683501 & 504586 & 77739 \end{bmatrix} \quad (\mathrm{mod}\ s_4).$$

Note that for this example, which is well conditioned, expanding the outer product adjoint and reducing modulo $s_4 = 486192114$

yields $s_4 A_{\mathsf{well}}^{-1}$. Valid choices for the v_* and u_* are

$$\begin{bmatrix} v_2 & v_3 & v_4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 59489394 \\ 2 & 1 & -59489391 \\ -2 & 3 & -176554567 \\ 0 & 1 & -41655398 \end{bmatrix} \tag{3.5}$$

and

$$\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 81659997 \\ 3 & 3 & -81024481 \\ 0 & 0 & -394319 \\ 1 & 1 & -242809128 \end{bmatrix}^T . \tag{3.6}$$

Because A_{well} only has three nontrivial invariant factors, the outer product $v_1 u_1$ (comprised of zero vectors because entries are reduced modulo 1) is not included in the outer product adjoint. ◊

Example 3.7. Example input matrix A_{ill} in (1.2) has $\mathrm{snf}(A_{\mathsf{ill}}) = \mathrm{Diag}(s_1, s_2, s_3, s_4) = \mathrm{Diag}(1, 3, 3, 2934)$ and

$$A_{\mathsf{ill}}^{-1} = \begin{bmatrix} -3285673 & 2076890 & -2559046 & -812713 \\ -2227201 & 1407824 & -1734670 & -550897 \\ -3007820 & 1901254 & -2342648 & -743978 \\ -3502023 & 2213652 & -2727564 & -866229 \end{bmatrix}.$$

An outer product adjoint for A_{ill} is given by

$$v_4 u_4 + (798) v_3 u_3 + (798) v_2 u_2$$
$$\equiv \begin{bmatrix} -1105 & -1102 & 140 & -1147 \\ -781 & 152 & 980 & -277 \\ -956 & 418 & 1078 & 556 \\ 399 & -798 & -798 & 399 \end{bmatrix} \pmod{s_4}.$$

Valid choices for the v_* and u_* are

$$\begin{bmatrix} v_2 & v_3 & v_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & -1019 \\ 0 & -1 & -340 \\ 0 & 0 & -399 \end{bmatrix} \tag{3.8}$$

and

$$\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ -1105 & 494 & 938 & -1147 \end{bmatrix}. \tag{3.9}$$

Note that for this example, which is ill conditioned, expanding the outer product adjoint only yields $(2934)A_{\mathsf{ill}}^{-1} \bmod 2934$. $\Diamond$

The goal of the remainder of this section is to describe a procedure to compute an outer product adjoint. Instead of directly giving the procedure, we first develop in Section 3.1 a procedure that takes as input a nonsingular $B \in \mathsf{R}^{n\times n}$ with

$$\mathrm{snf}(B) = \mathrm{Diag}(e_1, e_1e_2, \ldots, e_1e_2\cdots e_n)$$

and computes a decomposition for B as in (3.2): we call this a *Smith decomposition* of B. Then, Section 3.2 gives a procedure to compute an outer product adjoint for a nonsingular $A \in \mathsf{R}^{n\times n}$ by adapting the Smith decomposition procedure to the special case when $B = s_nA^{-1}$.

3.1. Computing a Smith decomposition. Let a nonsingular $B \in \mathsf{R}^{n\times n}$ be given. We will show how to compute column vectors $v_n, \ldots, v_1 \in \mathsf{R}^{1\times n}$ and row vectors $u_n, \ldots, u_1 \in \mathsf{R}^{n\times 1}$ such that

$$\begin{aligned} B &= (e_1)v_nu_n + (e_1e_2)v_{n-1}u_{n-1} + \cdots + (e_1\cdots e_n)v_1u_1 \\ &= e_1(v_nu_n + e_2(v_{n-1}u_{n-1} + \cdots + e_n(v_nu_n)\cdots)), \end{aligned} \tag{3.10}$$

where $\mathrm{snf}(B) = \mathrm{Diag}(e_1, e_1e_2, \ldots, e_1e_2\cdots e_n)$. The e_i do not need to be known in advance but are computed during the course of the procedure. The procedure works by constructing matrices $B_1, B_2, \ldots, B_{n+1} \in \mathsf{R}^{n\times n}$ such that

$$B_j = \frac{1}{e_1e_2\cdots e_{j-1}}(B - e_1(v_nu_n + e_2(v_{n-1}u_{n-1}+ \cdots + e_{j-1}(v_{n-j+2}u_{n-j+2})\cdots))) \tag{3.11}$$

with

$$\operatorname{snf}(B_j) = \operatorname{Diag}(e_j, e_je_{j+1}, \cdots, e_j, e_{j+1}, \cdots e_n, 0, 0, \ldots, 0). \tag{3.12}$$

Considering (3.12), $\operatorname{snf}(B_{n+1})$ will be the zero matrix, showing that (3.10) will follow from (3.11) for $j = n+1$.

A key feature of the procedures that we develop here is that the matrix B_j does not need to be represented explicitly, that is, as a single $n \times n$ matrix filled with elements of R. Rather, we only need construct explicitly the scalars e_i and vectors v_{n-i+1} and u_{n-i+1}, $1 \le i \le j-1$. Then, if B_j needs to be applied to a vector, an algorithm based on the definition of B_j in (3.11) can be used; the design and analysis of such an algorithm is deferred to Section 4.

We now explain how to construct the B_j by induction on j, $1 \le j \le n+1$, with base case $j = 1$. Initialize $B_1 = B$. Then B_1 satisfies (3.11) and (3.12). Suppose B_j satisfying (3.11) and (3.12) have been computed for $1 \le j \le i$, for some i with $1 \le j \le n$. We show how to compute B_{i+1} satisfying (3.11)) and (3.12). Because R is a principal ideal ring, there exist vectors $x \in \mathsf{R}^{1\times n}$ and $y \in \mathsf{R}^{n\times 1}$ such that $e_i := \gcd(B_i) = xB_iy$, the gcd of all entries of B_i. The nonzero invariant structure of a matrix does not change if we embed into a larger matrix banded with zeroes. Let $v := B_iy$ and $u := xB_i$, and consider the following unimodular transformation of the matrix obtained from B_i by augmenting it with an initial row and column of zeroes.

$$\left[\begin{array}{c|c} 1 & x \\ \hline & I_n \end{array}\right] \left[\begin{array}{c|c} 0 & \\ \hline & B_i \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline y & I_n \end{array}\right] = \left[\begin{array}{c|c} e_i & u \\ \hline v & B_i \end{array}\right] \tag{3.13}$$

Let $v_{n-i+1} := v/e_i \in \mathsf{R}^{n\times 1}$ and $u_{n-i+1} := u/e_i \in \mathsf{R}^{1\times n}$, and apply another unimodular transformation to the matrix on the right hand side of (3.13) to zero out entries below and to the right of e_i.

$$\left[\begin{array}{c|c} 1 & \\ \hline -v_{n-i+1} & I_n \end{array}\right] \left[\begin{array}{c|c} e_i & u \\ \hline v & B_i \end{array}\right] \left[\begin{array}{c|c} 1 & -u_{n-i+1} \\ \hline & I_n \end{array}\right] = \left[\begin{array}{c|c} e_i & \\ \hline & B_i - e_i v_{n-i+1} u_{n-i+1} \end{array}\right] \tag{3.14}$$

All entries of the matrix on the right hand side of (3.14) are divisible by e_i. Let $B_{i+1} = (1/e_i)(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathsf{R}^{n \times n}$, matching (3.11) for $j = i+1$. Since the matrix on the right of (3.14) has the same nonzero invariant factors as B_i, we conclude that (3.12) also holds for $j = i+1$. The procedure just described is summarized by Recipe 3.15, correctness of which follows by induction.

Recipe 3.15. Computing a Smith decomposition.
Input: A nonsingular $B \in \mathsf{R}^{n \times n}$.

```
 1.  Let $B_1 = B$.
 2.  For $i = 1, \ldots, n$ do 3–10
 3.      Find $x \in \mathsf{R}^{1 \times n}$ and $y \in \mathsf{R}^{n \times 1}$ such that $xB_iy = \gcd(B_i)$.
 4.      $v \leftarrow B_i y$.
 5.      $u \leftarrow x B_i$.
 6.      $e_i \leftarrow xv$.
 7.      $v_{n-i+1} \leftarrow v/e_i$.
 8.      $u_{n-i+1} \leftarrow u/e_i$.
 9.      Let $B_{i+1} = \frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathsf{R}^{n \times n}$.
10.      assert: Eqs. (3.11) and (3.12) hold for $j = i+1$.
11.  assert: $B = (e_1)v_n u_n + (e_1 e_2) v_{n-1} v_{n-1} + \cdots + (e_1 \cdots e_n) v_1 u_1$.
```

We defer until Sections 5 and 6 to describe how to compute the row and column vectors x and y in Step 3 in Recipe 3.15.

3.2. Computing an outer product adjoint. Now consider using Recipe 3.15 with an input matrix B that is equal to $s_n A^{-1}$ with $\mathrm{snf}(A) = \mathrm{Diag}(s_1, s_2, \ldots, s_n)$. Note that the Smith form of B is given by

$$\mathrm{snf}(B) = \mathrm{Diag}(\overbrace{e_1}^{s_n/s_n}, \overbrace{e_1 e_2}^{s_n/s_{n-1}}, \ldots, \overbrace{e_1 e_2 \cdots e_n}^{s_n/s_1}).$$

Given as input such a B, loop iteration i of Recipe 3.15 computes the first invariant factor of B_i which has

$$\operatorname{snf}(B_i) = \operatorname{Diag}(e_i, e_ie_{i+1}, \ldots, e_ie_{i+1}\cdots e_n, 0, 0, \ldots, 0).$$

By Lemma 2.2 we can recover the Smith form of B_i by working modulo its largest invariant factor $e_ie_{i+1}\cdots e_n = s_{n-i+2}$. (For convenience, define $s_{n+1} := s_n$.)

Recipe 3.16 adapts Recipe 3.15 to compute an outer product adjoint of A while keeping intermediate quantities reduced modulo the largest invariant factor.

Recipe 3.16. Computing an outer product adjoint.

Input: A nonsingular $A \in \mathsf{R}^{n\times n}$ and $s_n \in \mathsf{R}$, the largest invariant factor of A.

```
1.  Let $B_1 = s_nA^{-1}$ and $s_{n+1} = s_n$.
2.  For $i = 1, \ldots, n$ do 3–11
3.      Find $x \in \mathcal{R}(\mathsf{R}, s_{n-i+2})^{1\times n}$ and $y \in \mathcal{R}(\mathsf{R}, s_{n-i+2})^{n\times 1}$ such
        that $\operatorname{Rem}(xB_iy, s_{n-i+2}) = \operatorname{Rem}(\gcd(B_i, s_{n-i+2}), s_{n-i+2})$.
4.      $v \leftarrow \operatorname{Rem}(B_iy, s_{n-i+2})$.
5.      $u \leftarrow \operatorname{Rem}(xB_i, s_{n-i+2})$.
6.      $e_i \leftarrow \operatorname{Rem}(xv, s_{n-i+2})$.
7.      If $e_i = 0$ then break.
8.      $v_{n-i+1} \leftarrow v/e_i$.
9.      $u_{n-i+1} \leftarrow u/e_i$.
10.     $s_{n-i+1} \leftarrow s_{n-i+2}/e_i$.
11.     Let $B_{i+1} = \frac{1}{e_i}(B_i - e_iv_{n-i+1}u_{n-i+1}) \in \mathsf{R}^{n\times n}$.
12. assert: $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\le j\le i-1}$ is an outer product
    adjoint for $A$.
```

Recipe 3.16 requires as input the largest invariant factor s_n of A, and computes s_{n-i+1} at loop iteration $2, 3, \ldots, n$. There two subtleties to be aware of. First, at iteration i the matrix B_i is only correct modulo s_{n-i+2}. By Lemma 2.2, we need to compute e_i as the gcd of all entries of B_i over $\mathsf{R}/\langle s_{n-i+2}\rangle$: over R we compute the gcd of s_{n-i+2} with all entries in B_i, and then reduce modulo s_{n-i+2}. Second, if there exists a minimal $i \le n$ such that $e_i = 0$, then B_i is congruent modulo s_{n-i+2} to the zero matrix: this is handled by exiting the loop early since the trailing part of the sum $(s_n/s_{n-i+1})v_{n-i+1}u_{n-i+1} + \cdots + (s_n/s_1)v_1u_1$ in (3.10) is known to be congruent to zero modulo s_n in this case.

Sections 5 and 6 specialize Recipe 3.16 to the rings $\mathsf{R} = \mathbb{Z}$ and $\mathsf{R} = \mathsf{K}[z]$. 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 R-linear combination of B_iY for a Y that is randomly chosen so that $\gcd(B_iY, s_{n-i+2}) = \gcd(B_i, s_{n-i+2})$ with high probability. Once v is known, x is computed as the solution to an extended euclidean problem.

The key computational step that Recipe 3.16 leaves unspecified is how to compute the product B_iy efficiently when B_i is represented implicitly as s_nA^{-1} plus a sum of outer products; this is the topic of the next section.

4. Matrix multiplication with a partial outer product adjoint

Let $(s_{n-i+1}, v_{n-i+1}, u_{n-i+1})_{1\le i\le k}$ be an outer product adjoint for a nonsingular $A \in \mathsf{R}^{n\times n}$, such as computed by Recipe 3.16. If we define

$$T_i := \frac{s_n}{s_n}v_nu_n + \frac{s_n}{s_{n-1}}v_{n-1}u_{n-1} + \cdots + \frac{s_n}{s_{n-i+2}}v_{n-i+2}u_{n-i+2} \in \mathsf{R}^{n\times n}, \tag{4.1}$$

then at loop iteration $i = 1, 2, \ldots, k$, Recipe 3.16 needs to pre- and post-multiply a vector by the matrix

$$B_i = \frac{s_{n-i+2}}{s_n}\left(s_nA^{-1} - T_i\right) \in \mathsf{R}^{n\times n}, \tag{4.2}$$

working modulo s_{n-i+2}, where s_{n+1} is defined to be s_n. Recall that that B_i is represented implicitly as the original input matrix A and the scalars s_* and vectors v_* and u_* in (4.1). In this section we consider the problem of computing $\mathrm{Rem}(B_iY, s_{n-i+2})$ for a given $Y \in \mathcal{R}(\mathsf{R}, s_{n-i+2})^{n\times m}$.

Our approach is to compute $A^{-1}Y$ and $\mathrm{Rem}(T_iY, s_n)$ separately and then take a linear combination to arrive at the *target matrix*

$$T := s_{n-i+2}A^{-1}Y - (s_{n-i+2}/s_n)\,\mathrm{Rem}(T_iY, s_n) \in \mathsf{R}^{n\times m}. \tag{4.3}$$

The final result is obtained as $\mathrm{Rem}(T, s_{n-i+2})$. In general, both $s_{n-i+2}A^{-1}Y$ and $(s_{n-i+2}/s_n)\mathrm{Rem}(T_iY, s_n)$ will be over the fraction field of R with denominators being divisors of s_{n-i+2}/s_n, but the linear combination T in (4.3) is guaranteed to be over R by construction of the outer product adjoint. To avoid computing $A^{-1}Y$ at full precision we will work modulo an N that is relatively prime to s_n and that is large enough to capture exactly the entries in T in (4.3). The approach can be summarized as follows.

1. Choose a modulus N such that $N \perp s_n$ and the target matrix T in (4.3) is contained in $\mathcal{R}(\mathsf{R}, N)^{n\times m}$.

2. Compute $W_1 := \mathrm{Rem}(A^{-1}Y, N)$.

3. Compute $W_2 := \mathrm{Rem}(T_iY, s_n)$.

4. Compute $W := \mathrm{Rem}(s_{n-i+2}W_1 - (s_{n-i+2}/s_n)W_2, N)$.

5. Return $\mathrm{Rem}(W, s_{n-i+2})$.

We call this recipe to compute $\mathrm{Rem}(B_iY, s_{n-i+2})$ the *double modulus* approach since we alternately use the relatively prime moduli N, s_n, N and $s_{n-i+2}|s_n$ in steps 2, 3, 4 and 5, respectively.

Next we give two examples of the double modulus approach. Example 4.4, using A_{well} from (1.1), explains how the double modulus approach leads to a good complexity, at least in the case of a well conditioned input matrix. Example 4.8, using A_{ill} from (1.2), explains the sensitivity of the computational cost on the condition number of the input matrix.

Example 4.4. Consider the matrix $A_{\mathsf{well}} \in \mathbb{Z}^{4\times 4}$ from (1.1). From Example 3.4 we know that $s_4 = 6$, $s_5 = 486192114$, and we can take $T_3 = v_4u_4 + (81032019)v_3u_3$ where $v_4, v_3 \in \mathbb{Z}^{4\times 1}$ and $u_4, u_3 \in \mathbb{Z}^{1\times 4}$ are as in (3.5) and (3.6), respectively. Let $Y := \begin{bmatrix} -1 & 1 & -1 & -2 \end{bmatrix}^T$, with entries reduced modulo $s_3 = 6$. Consider the computation of $\mathrm{Rem}(B_3Y, s_4)$ using the approach summarized above. For now, we will omit the $\mathrm{Rem}(\cdot, N)$ operations in steps 2 and 4 to better illustrate the utility of the double modulus technique. First compute the nonsingular rational system solution

$$A_{\mathsf{well}}^{-1}Y = \begin{bmatrix} \frac{150002}{81032019} \\ -\frac{366661}{54021346} \\ \frac{2388827}{486192114} \\ -\frac{896440}{243096057} \end{bmatrix}. \tag{4.5}$$

Step three computes

$$W_2 := \mathrm{Rem}(T_3Y, s_4) = \begin{bmatrix} -161164026 \\ 158764089 \\ -159675211 \\ -1792880 \end{bmatrix}. \tag{4.6}$$

Next compute the target vector

$$\begin{aligned} T &:= s_3A_{\mathsf{well}}^{-1}Y - (s_3/s_4)W_2 \\ &= \begin{bmatrix} \frac{300004}{27010673} \\ -\frac{1099983}{27010673} \\ \frac{2388827}{81032019} \\ -\frac{1792880}{81032019} \end{bmatrix} - \begin{bmatrix} -\frac{53721342}{27010673} \\ \frac{52921363}{27010673} \\ -\frac{159675211}{81032019} \\ -\frac{1792880}{81032019} \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ -2 \\ 2 \\ 0 \end{bmatrix}. \end{aligned} \tag{4.7}$$

For this example, the target vector T in (4.7) is already reduced modulo $s_3 = 6$, so step 5 is not required. Note that integers in the target vector T in (4.7) have small bitlength compared to the intermediate results in (4.5) and (4.6).

To avoid the computation of $A_{\mathsf{well}}^{-1}Y$ at full precision we can work modulo $N = 97$ and in step 2 compute

$$W_1 := \mathrm{Rem}(A_{\mathsf{well}}^{-1} Y, 97) = \begin{bmatrix} -15 \\ 43 \\ 48 \\ 20 \end{bmatrix}.$$

And then in step 4 we have

$$W := \mathrm{Rem}(s_3 W_1 - (s_4/s_3) W_2, 97) = \begin{bmatrix} 2 \\ -2 \\ 2 \\ 0 \end{bmatrix}.$$

Note that $N = 97$ is large enough to capture exactly in the symmetric range modulo N the entries in T in (4.7). $\Diamond$

In general, we expect entries in the rational system solution $A^{-1}Y$ to be ratios of integers with bitlength $O(n(\log n + \log ||A||))$. For comparison, entries in the target matrix T in (4.3) will have bitlength only $O(\max(\log s_{n-i+2}, \log s_{n-i+2} + \log n + \log ||A^{-1}||))$. The next example shows how the $\log ||A^{-1}||$ term in this bound can dominate for an ill conditioned matrix.

EXAMPLE 4.8. Consider the matrix $A_{\mathsf{ill}} \in \mathbb{Z}^{4\times 4}$ from (1.2). From Example 3.7 we know that $s_3 = 3$, $s_4 = 2394$, and we can take $T_3 = v_4 u_4 + (798) v_3 u_3$ where $v_4, v_3 \in \mathbb{Z}^{4\times 1}$ and $u_4, u_3 \in \mathbb{Z}^{1\times r}$ are as in (3.8) and (3.9), respectively. Let $Y := \begin{bmatrix} -1 & -1 & 1 & -1 \end{bmatrix}^T$, with entries reduced modulo $s_3 = 3$. As in the previous example, we will omit the $\mathrm{Rem}(\cdot, N)$ operations in steps 2 and 4. First compute the nonsingular rational system solution

$$A_{\mathsf{ill}}^{-1} Y = \begin{bmatrix} -\frac{268775}{1197} \\ -\frac{182198}{1197} \\ -\frac{246052}{1197} \\ -\frac{718}{3} \end{bmatrix}$$

Step three computes

$$W_2 := \mathrm{Rem}(T_3 Y, s_4) = \begin{bmatrix} -496 \\ 1088 \\ 1060 \\ -798 \end{bmatrix}$$

Next compute the target vector

$$\begin{aligned} T &:= s_3 A_{\mathsf{ill}}^{-1} Y - (s_4/s_3) W_2 \\ &= \begin{bmatrix} -\frac{268775}{399} \\ -\frac{182198}{399} \\ -\frac{246052}{399} \\ -718 \end{bmatrix} - \begin{bmatrix} -\frac{248}{399} \\ \frac{544}{399} \\ \frac{530}{399} \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} -673 \\ -458 \\ -618 \\ -717 \end{bmatrix}. \end{aligned} \tag{4.9}$$

Finally, step 5 reduces the result modulo s_3.

$$\mathrm{Rem}(T, 3) = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Our final result is reduced modulo 3, but to apply the double modulus approach for this example, which is ill conditioned, we need to choose an N that is large enough to capture in the symmetric range modulo N the entries in the target vector T in (4.9). $\diamondsuit$

The rest of this section is divided into two subsection. In Section 4.1 we give an algorithm to compute $\mathrm{Rem}(T_i Y, s_n)$. In Section 4.2 a lower bound for the size of N is established to ensure that $\mathrm{Rem}(TY, N) = T$.

4.1. Application of scaled outer products. Given a $Y \in \mathcal{R}(\mathsf{R}, s_n)^{n\times m}$, the product $\mathrm{Rem}(T_iY, s_n)$ can be computed efficiently by using nested multiplication.

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

Algorithm 4.10 implements the above scheme. In the first loop, iteration j does $O(nm)$ arithmetic operations modulo s_{n-j+1} with operands from $\mathcal{R}(\mathsf{R}, s_{n-j+2})$ and $\mathcal{R}(\mathsf{R}, s_{n-j+1})$. Iteration j of the second loop does $O(nm)$ operations modulo s_{n-j+1} with operands from $\mathcal{R}(\mathsf{R}, s_{n-j+1})$.

ALGORITHM 4.10. $\mathtt{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\le j\le i-1}, Y)$.

Input: ◦ nonzero $s \in \mathcal{A}(\mathsf{R})$
◦ $s_{n-i+2}|s_{n-i+3}|\cdots|s_n = s$ and s_{n-i+2} a nonunit if $i > 1$
◦ $v_{n-j+1} \in \mathcal{R}(\mathsf{R}, s_{n-j+1})^{n\times 1}$, $1 \le j \le i-1$
◦ $u_{n-j+1} \in \mathcal{R}(\mathsf{R}, s_{n-j+1})^{1\times n}$, $1 \le j \le i-1$
◦ $Y \in \mathcal{R}(\mathsf{R}, s)^{n\times m}$

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

```
1.  If $i \le 1$ then
2.      Return the $n \times m$ zero matrix.
3.  $Y_{n+1} \leftarrow Y$.
4.  $s_{n+1} \leftarrow s_n$.
5.  For $j = 1, \ldots, i-1$ do 6–7
6.      $Y_{n-j+1} \leftarrow \mathrm{Rem}(Y_{n-j+2}, s_{n-j+1})$.
7.      $X_{n-j+1} \leftarrow \mathrm{Rem}(v_{n-j+1}(u_{n-j+1}Y_{n-j+1}), s_{n-j+1})$.
8.  $V \leftarrow X_{n-i+2}$.
9.  For $j = i-2, i-3, \ldots, 1$ do
10.     $V \leftarrow \mathrm{Rem}(X_{n-j+1} + (s_{n-j+1}/s_{n-j})V, s_{n-j+1})$.
11. Return $V$.
```

Lemma 4.11. *Algorithm 4.10 is correct. The running time is*

$$\mathsf{R} = \mathbb{Z}: \quad O(nm\,\mathsf{M}(\log \textstyle\prod_{j=1}^{i-1} s_{n-j+1}))\ \textit{bit operations.}$$

$$\mathsf{R} = \mathsf{K}[z]: \quad O(nm\,\mathsf{M}(\textstyle\sum_{j=1}^{i-1} \deg s_{n-j+1}))\ \textit{field operations from}\ \mathsf{K}.$$

Proof. The claimed running time bounds follow from the superlinearity of the cost function M. □

4.2. The double modulus approach. Our goal is to compute $\mathrm{Rem}(B_iY, s_{n-i+2})$. From the definition of B_i in (4.2) we have that

$$\mathrm{Rem}(B_iY, s_{n-i+2}) = \mathrm{Rem}(T, s_{n-i+2})$$

where T is as in (4.3). Let $N \in \mathsf{R}$ be relatively prime to s_n. If $\mathrm{Rem}(T, N) = T$ then also

$$\mathrm{Rem}(\mathrm{Rem}(T, N), s_{n-i+2}) = \mathrm{Rem}(T, s_{n-i+2})$$

and we can apply Recipe 4.12 to compute $\mathrm{Rem}(B_iY, s_{n-i+2})$.

Recipe 4.12. Application of B_i.
Input: $Y \in \mathcal{R}(\mathsf{R}, s_{n-i+2})^{n\times m}$.

```
1.    $W_1 := \mathrm{Rem}(A^{-1}Y, N)$.
2.    $W_2 := \mathtt{OPM}(s_n, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\le j\le i-1}, Y)$.
3.    $W := \mathrm{Rem}((s_{n-i+2}W_1 - (s_{n-i+2}/s_n)W_2, N)$.
4.    **return** $\mathrm{Rem}(W, s_{n-i+2})$
```

Theorem 4.13. *If $Y \in \mathcal{R}(\mathsf{R}, s_{n-i+2})^{n\times m}$ then Recipe 4.12 returns $\mathrm{Rem}(B_iY, s_{n-i+2})$ if*

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

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

Proof. The matrices $s_nA^{-1}Y$ and $\mathrm{Rem}(T_iY, s_n)$ are over 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 2.1, with B_iY playing the role of a.

Over $\mathsf{K}[z]$ we have

$$\begin{aligned} \deg s_n A^{-1} Y &\leq \deg s_n A^{-1} + \deg Y \\ &\leq \deg s_n A^{-1} + \deg s_{n-i+2} - 1 \end{aligned}$$

and $\deg \operatorname{Rem}(T_i Y, s_n) \leq \deg s_n - 1$. 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

$$\begin{aligned} \|s_n A^{-1} Y\| &\leq n s_n \|A^{-1}\| \, \|Y\| \\ &\leq n s_n \|A^{-1}\| s_{n-i+2}/2 \end{aligned}$$

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

Corollary 4.14. *Consider the case* $\mathsf{R} = \mathsf{K}[z]$. *If* $Y \in \mathsf{K}^{n \times m}$ *and* $\deg s_n A^{-1} - \deg s_n \leq 0$, *then Recipe 4.12 returns* $\operatorname{Rem}(B_i Y, s_{n-i+2})$ *if* $\deg N \geq 2 \deg s_{n-i+2}$.

5. Integer matrix inversion

Our Las Vegas algorithm for integer matrix inversion consists of three phases. To begin, we give a high level overview of each phase.

Phase 1 uses randomization to probabilistically compute the the following quantities.

- A modulus $p \in \mathbb{Z}_{>0}$ such $p \perp \det A$ and $\log p \in \Theta(\log n + \log \|A\|)$, together with the inverse $C = \operatorname{Rem}(A^{-1}, p)$.
- An upper bound α for $\|A^{-1}\|$.
- The largest invariant factor s of A.

Phase 2 adapts Recipe 3.16 for computing an outer product adjoint to the integer case. Recipe 3.16 did not specify how to find vectors $v \in \mathbb{Z}^{n \times 1}$ and $u \in \mathbb{Z}^{1 \times n}$ such that $e_i := \operatorname{Rem}(u B_i v, s_{n-i+2})$

is the first invariant factor of B_i. Here, we use randomization to compute e_i probabilistically as the gcd of all entries in B_iV and s_{n-i+2} for a randomly chosen V. Also, because some numbers (e.g., s and α from phase 1) are only computed probabilistically, some checks need to be added to ensure the algorithm stops and returns Fail in case the upper bound on the running time would be exceeded.

Phase 3 computes the explicit inverse of A, if necessary computing the extra component R in the decomposition $s_nA^{-1} = \mathrm{Rem}(s_nA^{-1}, s_n) + s_nR$. The inverse is assayed for correctness and either Fail or the correct inverse is returned.

In Section 5.1 we recall some known results from the literature and specialize them to our needs here. Probabilities of correctness, complexity bounds, and implementation details for phases 1, 2 and 3 are given in Sections 5.2, 5.3 and 5.4, respectively.

Algorithm 5.1. `IntInverse`(A).
Input: Nonsingular $A \in \mathbb{Z}^{n\times n}$.
Output: A^{-1}.
Comment: Fail may be returned with probability $< 1/2$.

```
        [Phase 1: Initialization]
 1.     Choose $\bar{p}$ uniformly and randomly from among the first
        $12\lfloor \log_2(n^{n/2}||A||^n)\rfloor$ primes.
 2.     $p \leftarrow$ the smallest positive integer power of $\bar{p}$ such that
        $\log_2 p \geq \log_2 \sqrt{n} + \log_2 ||A||$.
 3.     If $\bar{p} \nmid \det A$ Return Fail Else $C \leftarrow \mathrm{Rem}(A^{-1}, p)$.
 4.     $\alpha \leftarrow ||A^{-1}X||$, where $X \in \{-1,1\}^{n\times 4}$ is chosen uniformly
        and randomly.
 5.     $M \leftarrow 6 + \lceil 2n(\log_2 n + \log_2 ||A||)\rceil$.
 6.     $L \leftarrow \{0,\ldots,M-1\}$.
 7.     $s \leftarrow$ the denominator of $A^{-1}X$, where $X \in L^{n\times 12}$ is chosen
        uniformly and randomly.
 8.     $m \leftarrow 2\lceil (2+\log_2 n)/\log_2 3\rceil$.

        [Phase 2: Compute Outer Product Adjoint]
 9.     Let $B_1$ denote $sA^{-1}$ and $s_{n+1} = s$.
10.     For $i = 1,\ldots,n$ do 11–24
```

```
11.         Choose $Y \in L^{n\times m}$ uniformly and randomly.
12.         $Y \leftarrow \mathrm{Rem}(Y, s_{n-i+2})$.
13.         $N \leftarrow p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal such that $p^k \geq$
            $s_{n-i+2}(n\alpha s_{n-i+2}+1)+2$.
14.         Compute $V \leftarrow \mathrm{Rem}(B_iY, s_{n-i+2})$ using Recipe 4.12.
15.         Use Lemma 5.5 to compute $x \in \mathcal{R}(\mathbb{Z}, s_{n-i+2})^{1\times n}$ and
            $y \in \mathcal{R}(\mathbb{Z}, s_{n-i+2})^{m\times 1}$ such that $\mathrm{Rem}(xVy, s_{n-i+2}) =$
            $\mathrm{Rem}(\gcd(V, s_{n-i+2}), s_{n-i+2})$.
16.         $v \leftarrow \mathrm{Rem}(Vy, s_{n-i+2})$.
17.         Compute $u \leftarrow \mathrm{Rem}(B_i^Tx^T, s_{n-i+2})^T$ using Recipe 4.12.
18.         $e_i \leftarrow \mathrm{Rem}(xv, s_{n-i+2})$.
19.         If $e_i = 0$ then break Else $s_{n-i+1} \leftarrow s_{n-i+2}/e_i$.
20.         If $(i = 1$ and $s_n \neq s)$ then Return Fail.
21.         If $\prod_{j=1}^{i} s_{n-j+1} > n^{n/2}||A||^n$ or $e_i \not| u$ then Return Fail.
22.         $u_{n-i+1} \leftarrow u/e_i$.
23.         $v_{n-i+1} \leftarrow v/e_i$.
24.         Let $B_{i+1}$ denote $\frac{1}{e_i}(B_i - e_iv_{n-i+1}u_{n-i+1}) \in \mathbb{Z}^{n\times n}$.

      [Phase 3: Compute Inverse and Assay Correctness]
25.   If $\mathtt{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\leq j\leq i-1}, A) \neq 0_{n\times n}$ then
26.       Return Fail.
27.   $C_0 \leftarrow \mathtt{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\leq j\leq i-1}, I_n)$.
28.   $N_1 \leftarrow p^k$, where $k \in \mathbb{Z}_{\geq 0}$ is minimal such that $p^ks \geq 2\alpha s+2$.
29.   $C_1 \leftarrow \mathrm{Rem}(s\,\mathrm{Rem}(A^{-1}, N_1), N_1)$, computed using $p$-adic
      lifting.
30.   $C \leftarrow \mathrm{Rem}(N_1\,\mathrm{Rem}(1/N_1, s)C_0 + s\,\mathrm{Rem}(1/s, N_1)C_1, N_1s)$.
31.   $N_2 \leftarrow p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal such that $p^k \geq$
      $(2n/s)||A||||C|| + 2$.
32.   If $\mathrm{Rem}(A\,\mathrm{Rem}((1/s)C, N_2), N_2) \neq I_n$ then Return Fail.
33.   Return $(1/s)C$, with fractions reduced.
```

5.1. Computational tools. 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 5.2 (Eberly *et al.* 2000, 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* $L := \{0, 1, \ldots, M-1\}$ *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* $1/3$.

An inspection of (Eberly *et al.* 2000, Proof of Theorem 2.1) reveals that the definition of M in Lemma 5.2 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 (Eberly *et al.* 2000, Theorem 2.1).

Corollary 5.3. *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 (Dixon 1982), used to compute $\mathrm{Rem}(A^{-1}X, p^k)$ for a given $X \in \mathbb{Z}^{n\times m}$ and k. The presentation and analysis in Dixon (1982) 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. For a detailed derivation of parts (i) and (ii) of the following lemma we refer to Mulders & Storjohann (2004, Section 5).

Lemma 5.4. *Let* $A \in \mathbb{Z}^{n\times n}$ *be nonsingular. Suppose we are given*

- *a* $p \in \mathbb{Z}_{>0}$ *with* $p \perp \det A$ *and* $\log p \in \Theta(\log n + \log ||A||)$,
- $C := \mathrm{Rem}(A^{-1}, p)$,
- $N := p^k$ *for an integer* $k > 0$.

(i) Suppose $Y \in \mathbb{Z}^{n\times m}$ *satisfies* $\log ||Y|| \in O(\log N)$. *Then* $\mathrm{Rem}(A^{-1}Y, N)$ *can be computed using linear* p*-adic lifting in*

$$O(n^2 km\,\mathsf{B}(\log n + \log ||A||) + nm\,\mathsf{B}(k(\log n + \log ||A||)))$$

bit operations.

(ii) *Suppose* $Y \in \mathbb{Z}^{n\times m}$ *satisfies* $\log ||Y|| \in O(n(\log n + \log ||A||))$. *Then* $A^{-1}Y$ *can be computed with* $O(n^3 m\, \mathsf{B}(\log n + \log ||A||))$ *bit operations.*

(iii) *Cost of Recipe 4.12. If* $\prod_{j=1}^{i-1} s_{n-i+1} \le n^{n/2}||A||^n$ *and* $\log N \in O(\max(\log p, \log(s_{n-i+2}(ns_{n-i+2}||A||^{-1}+1)+2)))$, *then the recipe can be applied with*

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

bit operations, where

$$k_{n-i+2} := \max\left(1, \frac{\log s_{n-i+2}}{\log n + \log ||A||}, \frac{\log ||A^{-1}||}{\log n + \log ||A||}\right).$$

Proof. To understand the cost bound for part (iii) it is helpful to recall the cost analysis for part (i). 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 $O(n^2 km\, \mathsf{B}(\log n + \log ||A||))$ 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 (iii). The computation of W_2 in Recipe 4.12 costs $O(nm\, \mathsf{M}(n(\log n + \log ||A||)))$ bit operations (Lemma 4.11). This also bounds the cost of the arithmetic operations performed in the return statement of Recipe 4.12. The cost of computing $W_1 := \mathrm{Rem}(A^{-1}Y, N)$ is as stated in part (i) with $k = \log_p N$. The definition of k_{n-i+2} is such that $k_{n-i+2} \in \Theta(\log_p N)$. The cost estimate stated in part (iii) simplifies the second term of the cost estimate from part (i) 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 5.5. *Given $V \in \mathcal{R}(\mathbb{Z},s)^{n\times m}$, we can compute vectors $x \in \mathcal{R}(\mathbb{Z},s)^{1\times n}$ and $y \in \mathcal{R}(\mathbb{Z},s)^{m\times 1}$ such that* $\mathrm{Rem}(xVy,s) = \mathrm{Rem}(\gcd(V,s),s)$ *in $O(nm\,\mathsf{B}(\log s))$ bit operations.*

PROOF. Storjohann (1997, Section 4.1) derives a method to compute a y such that $\gcd(Vy,s) = \gcd(V,s)$. The cost of the method is $O(nm\,\mathsf{B}(\log s))$ 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 $O(\mathsf{B}(\log s))$ bit operations using operation Stab from Storjohann & Mulders (1998). Once y has been computed, compute x to be a solution to the standard extended euclidean problem with input Vy. $\square$

5.2. Phase 1: Initialization. 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 (i) of the following lemma gives some properties that are guaranteed to hold by construction, while part (ii) 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 5.6. *Phase 1 of Algorithm 5.1 makes use of $O(n(\log n + \operatorname{loglog}||A||))$ random bits and completes in $O(n^3\,\mathsf{B}(\log n + \log||A||))$ bit operations. Fail will be returned with probability at most $1/12$. If Fail is not returned, then (i) will hold, and (ii) will hold with overall probability at least $1 - 1/6$.*

(i) $p \perp \det A$, $\log p \geq \log\sqrt{n} + \log||A||$ 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.

(ii) $\alpha \geq ||A^{-1}||$, 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 (i). From the prime number theorem we know that $\log \bar{p} \in \Theta(\log n + \operatorname{loglog} ||A||)$; the upper bound and lower bounds for $\log p$ follow by construction. The upper bound for α uses $||X|| = 1$: $||A^{-1}X|| \leq n||A^{-1}||\,||X|| = n||A^{-1}||$. Abbott *et al.* (1999) show that s is a factor of s_n.

Now consider part (ii). 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 (ii) will hold with probability at least $1 - 1/6$ will follow.

To see that $\alpha < ||A^{-1}||$ with probability at most 1/16, let $x = \left[\begin{array}{c|c} x_1 & \bar{x} \end{array}\right]^T \in \{-1, 1\}^{n\times 1}$ be chosen uniformly and randomly, and let $a = \left[\begin{array}{c|c} a_1 & \bar{a} \end{array}\right] \in \mathbb{Q}^{1\times n}$ 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 $\operatorname{sign}(a_1x_1) = \operatorname{sign}(\bar{a}\bar{x})$; 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 5.2 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 (Knuth 1981, Section 4.5.4). The modular inverse C can be computed in the allotted time using Gaussian elimination. By (ii), the rational system solutions $A^{-1}X$ can be computed in the allotted time using p-adic lifting. $\square$

5.3. Phase 2: Compute Outer Product Adjoint Formula. Phase 2 of Algorithm 5.1 adapts the algorithm for outer product adjoint shown in Recipe 3.16 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 $\gcd(B_iY, s_{n-i+2})$ 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. Lemma 5.6(ii)). 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 5.7. *Phase 2 of Algorithm 5.1 uses* $O(n^2(\log n)(\log n + \operatorname{loglog}||A||))$ *random bits and completes in*

$$O(n^2(\log n)\,\mathsf{B}(n(\log n + \log||A||))) \tag{5.8}$$

plus

$$O(n^3 \max(\log n, \log||A^{-1}||)\,\mathsf{B}(\log n + \log||A||)) \tag{5.9}$$

bit operations.

PROOF. Since the algorithm will stop and return Fail if the bound $\prod_{j=1}^{i} s_{n-j+1} \le n^{n/2}||A||^n$ is not satisfied, 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 4.12, the running time bound in (5.8) for the entire phase follows from the superlinearity of B: $\sum_{j=1}^{i-1} \mathsf{B}(\log s_{n-j+1}) \in O(\mathsf{B}(\sum_{j=1}^{i-1} \log s_i))$.

The cost of the two calls to Recipe 4.12 for single loop iteration is given by Lemma 5.4(iii). Let k_{n-i+2} be defined as in Lemma 5.4. Then

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

The part of the cost bound related to the lifting, shown in (5.9), incorporates the simplification $\log n + \log||A|| \ge \log n$. □

LEMMA 5.10. *If s is the largest invariant factor of A and $\alpha \ge ||A^{-1}||$, then phase 2 of Algorithm 5.1 will not return Fail and computes a correct outer product adjoint for A with probability at least* $1 - 1/4$.

PROOF. Phase 2 is identical to algorithm for outer product adjoint shown in Recipe 3.16 except that e_i is computed probabilistically. By Theorem 4.13, Recipe 4.12 will correctly compute V. Corollary 5.3 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. □

5.4. Phase 3: Compute Inverse and Assay Correctness.

LEMMA 5.11. *If s is the largest invariant factor of A, $\alpha \geq ||A^{-1}||$, and the $(s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\leq j\leq i-1}$ computed in phase 2 comprise an outer product adjoint for A, then phase 3 of Algorithm 5.1 will not return Fail.*

PROOF. Assume the conditions of the lemma are satisfied. Then the first call to Algorithm 4.10 (`OPM`) computes $\mathrm{Rem}(sA^{-1}A, s)$ which will be the zero matrix and the second call computes $C_0 = \mathrm{Rem}(sA^{-1}, s)$. The matrix C_1 is equal to $\mathrm{Rem}(sA^{-1}, N_1)$. The matrix C is obtained by Chinese remaindering together C_0 and C_1 to get $C = \mathrm{Rem}(sA^{-1}, N_1 s)$. Since $||sA^{-1}|| \leq s\alpha$, and $N_1 s \geq 2\alpha s + 2$, by Lemma 2.1 we have $sA^{-1} = \mathrm{Rem}(sA^{-1}, N_1 s)$ 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. □

LEMMA 5.12. *If phase 3 of Algorithm 5.1 does not return Fail, then the correct inverse of A is returned.*

PROOF. If the first call to `OPM` yields the zero matrix then C_0 as computed in the next line has the property that $\mathrm{Rem}(AC_0, s)$ is the zero matrix. By construction of C, $\mathrm{Rem}(C, s) = C_0$, so $\mathrm{Rem}(AC, s)$ 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 2.1, the choice of N_2 guarantees that $(1/s)AC = \mathrm{Rem}((1/s)AC, N_2)$. If $(1/s)AC = I_n$ then $(1/s)C$ must be the inverse of A. □

LEMMA 5.13. *Phase 3 of Algorithm 5.1 completes in*

$$O(n^2\,\mathsf{B}(n(\log n + \log ||A||))) \tag{5.14}$$

plus

$$(5.15)\quad O(n^3(\log ||A||^{-1})/(\log n + \log ||A||)\,\mathsf{B}(\log n + \log ||A||))$$

bit operations.

Proof. Algorithm 4.10 bounds the cost of the two calls to `OPM` by (5.14) bit operations. Since $\alpha \leq n||A^{-1}||$ and $s \leq n^{n/2}||A||^n$, all of N_1, $N_1 s$ and N_2 will have bitlength bounded by $O(n(\log n + \log ||A||))$. Thus, the cost of computing C, $\mathrm{Rem}((1/s)C, N_2)$, and reducing the fractions in $(1/s)C$ is bounded by (5.14) bit operations as well.

The costs of computing $\mathrm{Rem}(A^{-1}, N_1)$ via lifting and computing the matrix product $A\,\mathrm{Rem}((1/s)C, N_2)$ are sensitive to α. In particular, both $\log N_1$ and $\log N_2$ are $O(\log n + \log ||A^{-1}||)$. By Lemma 5.4, C_1 can be computed in (5.15) bit operations. Now consider the computation of the product $A\,\mathrm{Rem}((1/s)C, N_2)$. To begin, convert $\mathrm{Rem}((1/s)C, N_2)$ to a p-adic expansion with $O((\log ||A^{-1}||)/(\log n + \log ||A||))$ terms. Multiplying each term by A has cost bounded by (5.15). Convert back to standard representation. The cost of the radix conversion is bounded by (5.14) bit operations. □

Theorem 5.16. *Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular. Algorithm 5.1 uses $O(n^2(\log n)(\log n + \mathrm{loglog}\,||A||))$ random bits and completes in*

$$O(n^2(\log n)\mathsf{B}(n(\log n + \log ||A||)))$$

plus

$$O(n^3 \max(\log n, \log ||A^{-1}||)\mathsf{B}(\log n + \log ||A||))$$

bit operations. The algorithm either returns Fail or A^{-1}. Fail is returned with probability less than 1/2.

Proof. Lemma 5.12 guarantees that an incorrect result will not be returned. By Lemma 5.11, 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. Summing the probabilities from Lemma 5.6 and 5.10 gives $1/12 + 1/6 + 1/4 \leq 1/2$. Phase 2 dominates (Lemma 5.7). □

Recall that $\kappa(A) = n||A||\,||A||^{-1}$ where $||\cdot||$ denote the max-norm. The following corollary of Theorem 5.16 follows from the fact that $\kappa(A) \geq 1$.

Corollary 5.17. *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 $(n^3(\log||A|| + \log \kappa(A)))^{1+o(1)}$ bit operations.*

6. Polynomial matrix inversion

Let $A \in \mathsf{K}[z]^{n\times n}$ be nonsingular with degree $d > 0$, K a field. Recall that the degree of the determinant of A is bounded by nd, while entries in the adjoint matrix $A^{\mathsf{adj}} := (\det A)A^{-1}$ 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 for A will be sensitive to the difference between $\deg A^{\mathsf{adj}}$ and $\deg\det A$. Generically, $\deg A^{\mathsf{adj}} - \deg\det A = (n-1)d - nd < 0$, but we may have $\deg A^{\mathsf{adj}} - \deg\det A \leq (n-1)d$, 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 the following input matrix over $\mathsf{K}[z]$ where $\mathsf{K} = \mathbb{Z}/\langle 97\rangle$.

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

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

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

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 $\alpha \in \mathsf{K}$. These ideas are summarized in the following lemma, the proof of which is elementary.

Lemma 6.1. *Let $A \in \mathsf{K}[z]^{n\times n}$ with degree bounded by d and let $\alpha \in \mathsf{K}$. 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 $\det \mathrm{Rem}(A_1, z) \neq 0$ then $\deg\det A_2 = nd$.*

In Section 6.1 we recall some know results from the literature and specialize them to our needs here. Probability of correctness, complexity bound, and implementation details for phases 1, 2 and 3 are given in Sections 6.2, 6.3, and 6.4, respectively.

Algorithm 6.2. `PolyInverse`(A, Λ).
Input: ∘ nonsingular $A \in \mathsf{K}[z]^{n\times n}$ of degree $d > 0$, K a field
∘ a set Λ of elements of K
Output: A^{-1}.
Comment: Fail may be returned with probability $\leq 4nd/\#\Lambda$.

```
     [Phase 1: Initialization]
1.   If $\det \mathrm{Rem}(A, z) \neq 0$ then
2.       $\alpha \leftarrow 0$.
3.       $A_1 \leftarrow A$.
4.   Else
5.       Choose $\alpha$ uniformly and randomly from $\Lambda$.
6.       $A_1 \leftarrow A\,|_{z=z+\alpha}$.
7.       If $\det \mathrm{Rem}(A_1, z) = 0$ then Return Fail.
8.   $A_2 \leftarrow z^d\left(A_1\,|_{z=1/z}\right)$.
9.   Choose $\beta$ uniformly and randomly from $\Lambda$.
10.  If $\det \mathrm{Rem}(A_2, z-\beta) = 0$ then Return Fail.
11.  $p \leftarrow (z-\beta)^d$.
12.  $C \leftarrow \mathrm{Rem}(A_2^{-1}, p)$.
13.  Choose $y \in \Lambda^{n\times 1}$ uniformly and randomly.
14.  $s \leftarrow$ the denominator of $A_2^{-1}y$.
```

```
       [Phase 2: Compute Outer Product Adjoint]
15.    Let $B_1$ denote $sA_2^{-1}$ and $s_{n+1} = s$.
16.    For $i = 1, \ldots, n$ do 17–28
17.        If $i > 1$ then choose $y \in \Lambda^{n\times 1}$ uniformly and randomly.
18.        Set $N \leftarrow p^k$, where $k \in \mathbb{Z}_{>0}$ is minimal such that
           $\deg p^k \geq 2 \deg s_{n-i+2}$.
19.        Compute $v \leftarrow \mathrm{Rem}(B_i y, s_{n-i+2})$ using Recipe 4.12.
20.        Compute $x \in \mathcal{R}(\mathsf{K}[z], s_{n-i+2})^{1\times n}$ such that
           $\mathrm{Rem}(xv, s_{n-i+2}) = \mathrm{Rem}(\gcd(v, s_{n-i+2}), s_{n-i+1})$.
21.        Compute $u \leftarrow \mathrm{Rem}(B_i^T x^T, s_{n-i+2})^T$ using Recipe 4.12.
22.        $e_i \leftarrow \mathrm{Rem}(xv, s_{n-i+2})$.
23.        If $e_i = 0$ then break Else $s_{n-i+1} \leftarrow s_{n-i+2}/e_i$.
24.        If ($i = 1$ and $s_n \neq s$) then Return Fail.
25.        If $\sum_{j=1}^{i} s_{n-j+1} > nd$ or $e_i \not| u$ then Return Fail.
26.        $u_{n-i+1} \leftarrow u/e_i$.
27.        $v_{n-i+1} \leftarrow v/e_i$.
28.        Let $B_{i+1} = \frac{1}{e_i}(B_i - e_i v_{n-i+1} u_{n-i+1}) \in \mathsf{K}[z]^{n\times n}$.
29.    If $\sum_{j=1}^{i-1} \neq nd$ then Return Fail.

       [Phase 3: Compute Inverse and Assay Correctness]
30.    If $\mathtt{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\leq j\leq i-1}, v_2) \neq 0_{n\times n}$ then
31.        Return Fail.
32.    $C \leftarrow \mathtt{OPM}(s, (s_{n-j+1}, v_{n-j+1}, u_{n-j+1})_{1\leq j\leq i-1}, I_n)$.
33.    If $\mathrm{Rem}(A_2 \,\mathrm{Rem}((1/s)C, p), p) \neq I_n$ then Return Fail.
34.    Return $\left((z^d/s)C\right)|_{z=1/(z-\alpha)}$, with fractions reduced.
```

6.1. Computational tools. In addition to a nonsingular input $A \in \mathsf{K}[z]^{n\times n}$, our algorithm for computing the inverse will take as input a subset Λ of elements of K. Probability estimates will depend on $\#\Lambda$. The following result is the polynomial analogue of Corollary 5.3.

LEMMA 6.3. *Let $s \in \mathsf{K}[z]$ be nonzero. For any $B \in \mathsf{K}[z]^{n\times n}$, 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 $\mathsf{K}[z]/\langle p\rangle$ 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 6.3 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 $\gcd((s_nA^{-1})y, s_n) = 1$.

Corollary 6.4. *Let $A \in \mathsf{K}[z]^{n\times n}$ 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 Mulders & Storjohann (2004, Section 5) for a derivation of Lemma 6.5(i). The derivation of part (ii) is similar to Lemma 5.4(iii). The $n\,\mathsf{B}(nd)$ term comes from the application of algorithm `OPM`. The $n^2k_{n-i+2}\,\mathsf{B}(d)$ term captures the cost of performing $O(k_{n-i+2})$ lifting steps to obtain $\mathrm{Rem}(A^{-1}y, N)$.

Lemma 6.5. *Let $A \in \mathsf{K}[z]^{n\times n}$ be nonsingular of degree d. Suppose we have $p \in \mathsf{K}[z]_{>0}$ with $p \perp \det A$ and $\deg p = d$, together with $C := \mathrm{Rem}(A^{-1}, p)$.*

(i) If $y \in \mathsf{K}[z]^{n\times 1}$ satisfies $\deg y \in O(nd)$, then $A^{-1}y$ can be computed with $O(n^3\,\mathsf{B}(d))$ field operations from K.

(ii) Cost of Recipe 4.12. If $\sum_{j=1}^{i-1} \deg s_{n-i+1} \le nd$ and $\deg N \in O(\deg s_{n-i+2})$, then the recipe can be applied with

$$O(n^2k_{n-i+2}\,\mathsf{B}(d) + n\,\mathsf{B}(nd))$$

field operations from K, where $k_{n-i+2} := 1 + (\deg s_{n-i+2})/d$.

6.2. Phase 1: Initialization.

Lemma 6.6. *Phase 1 of Algorithm 6.2 completes in $O(n^3\,\mathsf{B}(nd))$ 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 $nd/\#\Lambda$. The same upper bound holds for the probability that $\det \operatorname{Rem}(A_2, t - \beta) = 0$.

Now assume that Fail is not returned. That $\deg \det A_2 = nd$ follows from Lemma 6.1. Corollary 6.4 gives the probability estimate for the correctness of s. By Lemma 6.5, the computation of $A^{-1}y$, which dominates the cost of the phase, can be accomplished in the allotted time. □

Note that the second probability estimate in Lemma 6.6 is conditional: the probability that phase 1 does not return Fail and s is computed correctly is at least $1 - 2nd/\#\Lambda - nd/\#\Lambda$.

6.3. Phase 2: Compute Outer Product Adjoint Formula.

Lemma 6.7. *Phase 2 of Algorithm 6.2 completes in $O(n^2\,\mathsf{B}(nd))$ 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 4.12, a total cost bound of $O(n\,\mathsf{B}(nd))$ field operations for the entire phase follows from the superlinearity of B.

The cost of the two calls to Recipe 4.12 for a single loop iterations is given by Lemma 6.5(ii). Let k_{n-i+2} be as defined in Lemma 6.5(ii). Then $\sum_{j=1}^{i-1} k_{n-i+2} \in O(n)$. The overall cost of all calls to Recipe 4.12 is thus $O(n^2\,\mathsf{B}(nd))$ field operations. □

Lemma 6.8. *If s is the largest invariant factor of A_2, then phase 2 of Algorithm 6.2 computes an outer product adjoint for A with probability at least $1 - nd/\#\Lambda$.*

Proof. Phase 2 is identical to the algorithm for outer product adjoint shown in Recipe 3.16 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 $s_{n+1}, s_n, \ldots, s_{n-i+2}$ and hence $B_1, B_2, \ldots, B_i$ are computed correctly up to some i. By Corollary 4.14, Recipe 4.12 will correctly compute v. By Lemma 6.3, 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 $nd/\#\Lambda$. $\square$

6.4. Phase 3: Compute Inverse and Assay Correctness.

LEMMA 6.9. *If s is the largest invariant factor of A, and phase 2 computes a correct outer product adjoint for A, then phase 3 of Algorithm 6.2 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 for A_2, as specified in the lemma. The first call to `OPM` computes $\mathrm{Rem}(sA^{-1}, s)$ which will be the zero matrix. The second call computes $C = \mathrm{Rem}(sA_2^{-1}, s)$. Since $\deg\det A_2 = nd$, we have $\deg sA_2^{-1} < \deg s$ and by Lemma 2.1 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 6.10. *If phase 3 of Algorithm 6.2 does not return Fail, then the correct inverse of A is returned.*

PROOF. If the first call to `OPM` does not return fail, then C as computed in the next line has the property that $\mathrm{Rem}(A_2C, s)$ 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 the inverse of A_2. $\square$

THEOREM 6.11. *Let $A \in \mathsf{K}[z]^{n\times n}$ be nonsingular with degree d. Algorithm 6.2 completes in $O(n^2\,\mathsf{B}(nd))$ field operations. The algorithm either returns Fail or A^{-1}. Fail is returned with probability $4nd/\#\Lambda$.*

To ensure a positive probability of success we require that $\#\mathsf{K}$ is large enough. If K is too small we can work over an algebraic extension of degree $O(\log nd)$ over K. The cost bound of Theorem 6.11 increases by some factors of $\log nd$.

Corollary 6.12. *Let K be a field and z be an indeterminate. There exists a Las Vegas algorithm that computes the exact inverse of a nonsingular $A \in \mathsf{K}[z]^{n\times n}$ using an expected number of $(n^3 \deg A)^{1+o(1)}$ field operations from K.*

7. Conclusions

We have demonstrated a Las Vegas randomized algorithm to compute A^{-1} where $A \in \mathbb{Z}^{n\times n}$ in $(n^3(\log ||A|| + \log \kappa(A)))^{1+o(1)}$ bit operations. The main outstanding problem is to remove the dependence of the running time on the condition number $\kappa(A) := n||A||\,||A^{-1}||$. A promising approach may be to use additive preconditioning as described by Pan *et al.* (2008). For example, even if $A \in \mathbb{Z}^{n\times n}$ is ill conditioned, the rank one perturbation $A + vu$ for a randomly chosen column vector v and row vector u may be well conditioned.

The inversion algorithm in this paper is based on computing an outer product adjoint, a representation of $\mathrm{Rem}((\det A)A^{-1}, \det A)$ as the sum of scaled outer products. Since the size of the outer product adjoint is about the same as that required to represent the input matrix, a natural question is if fast matrix multiplication can be incorporated for its computation. We can partially answer this question. Let k be the number of nontrivial invariant factors of an input matrix $A \in \mathbb{Z}^{n\times n}$. If high-order lifting (Storjohann 2005) is used to compute the required rational system solutions, then running phases 1 and 2 of Algorithm 5.1 can be used to probabilistically compute an outer product adjoint of A in $k \times (n^\omega \log ||A||)^{1+o(1)}$ bit operations, independent of $\kappa(A)$. For a "random" integer matrix we can expect k to be small. In particular, let $\Lambda = \{a, a+1, a+2, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda \in \mathbb{Z}_{\geq 2}$, and suppose $A \in \Lambda^{n\times n}$ is chosen uniformly and randomly. Then Eberly, Giesbrecht & Villard (2000, Corollary 6.3) show that the expected number k of nontrivial invariant of A is bounded by $O(\log_\lambda n)$.

We have shown how to adapt our algorithms for integer matrix inversion to obtain an algorithm for polynomial matrix inversion. The Las Vegas randomized algorithm we obtain has cost

$(n^3 \deg A)^{1+o(1)}$ field operations from K for any nonsingular input matrix $A \in \mathsf{K}[z]^{n\times n}$. Recently, Zhou *et al.* (2014) give a variant of the inversion algorithm of Jeannerod & Villard (2005) that has no genericity requirements: the algorithm can deterministically compute the inverse of any input in $(n^3 \deg A)^{1+o(1)}$ field operations. For an exposition of applications of the $(n^3 \deg A)^{1+o(1)}$ inversion algorithms, including the fast computation of a sequence of scalar matrix powers, we refer to Zhou *et al.* (2014, Section 5).

For an input $A \in \mathsf{K}[z]^{n\times n}$ that is row reduced and whose determinant does not vanish modulo z, Gupta (2011) gives a Las Vegas algorithm to compute an outer product adjoint in $(n^\omega \deg A)^{1+o(1)}$ field operations.

Acknowledgements

The author thanks Burcin Erocal for pointing out some corrections to an earlier version of the polynomial matrix inversion algorithm, and the referees for their comments.

References

J. Abbott, M. Bronstein & T. Mulders (1999). Fast deterministic computation of determinants of dense matrices. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation ISSAC '99,* Vancouver, Canada, S. Dooley, editor, 197–204. ACM Press, New York. URL `http://dx.doi.org/10.1145/309831.309934`.

P. Bürgisser, M. Clausen & M. A. Shokrollahi (1996). *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften.* Springer-Verlag.

A. De, P. Kurur, C. Saha & R. Saptharishi (2013). Fast integer multiplication using modular arithmetic. *SIAM Journal on Computing* **42**(2), 658–699. URL `http://dx.doi.org/10.1137/100811167`.

J. D. Dixon (1982). Exact solution of linear equations using p-adic expansions. *Numerische Mathematik* **40**, 137–141. URL `http://dx.doi.org/10.1007/BF01459082`.

P. D. Domich, R. Kannan & L. E. Trotter, Jr. (1987). Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research* **12**(1), 50–59. URL `http://www.jstor.org/stable/3689672`.

W. Eberly, M. Giesbrecht & G. Villard (2000). Computing the determinant and Smith form of an integer matrix. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science,* Redondo Beach CA, 675–685. URL `http://dx.doi.org/10.1109/SFCS.2000.892335`.

M. Fürer (2009). Faster integer multiplication. *SIAM Journal on Computing* **39**(3), 979–1005. URL `http://dx.doi.org/10.1137/070711761`.

J. von zur Gathen & J. Gerhard (2013). *Modern Computer Algebra.* Cambridge University Press, 3rd edition.

S. Gupta (2011). *Hermite Forms of Polynomial Matrices.* Master's thesis, University of Waterloo. URL `http://hdl.handle.net/10012/6108`.

J. L. Hafner & K. S. McCurley (1991). Asymptotically fast triangularization of matrices over rings. *SIAM Journal on Computing* **20**(6), 1068–1083. URL `http://dx.doi.org/10.1137/0220067`.

C. S. Iliopoulos (1989). 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 on Computing* **18**(4), 658–669. URL `http://dx.doi.org/10.1137/0218045`.

C. P. Jeannerod & G. Villard (2005). Essentially optimal computation of the inverse of generic polynomial matrices. *Journal of Complexity* **21**, 72–86. URL `http://dx.doi.org/10.1016/j.jco.2004.03.005`.

E. Kaltofen (1992). On computing determinants of matrices without divisions. In *Proceedings of the 1992 International Symposium on Symbolic and Algebraic Computation ISSAC '92,* Berkeley CA, P. S. Wang, editor, 342–349. ACM Press, New York. URL `http://doi.acm.org/10.1145/143242.143350`.

E. Kaltofen & G. Villard (2004). On the complexity of computing determinants. *Computational Complexity* **13**(3–4), 91–130. URL `http://dx.doi.org/10.1007/s00037-004-0185-3`.

I. Kaplansky (1949). Elementary divisors and modules. *Transactions of the American Mathematical Society* **66**, 464–491.

D. E. Knuth (1981). *The Art of Computer Programming, Vol.2, Seminumerical Algorithms.* Addison–Wesley (Reading MA), 2nd edition.

T. Mulders & A. Storjohann (2004). Certified dense linear system solving. *Journal of Symbolic Computation* **37**(4), 485–510. URL `http://dx.doi.org/10.1016/j.jsc.2003.07.004`.

M. Newman (1972). *Integral Matrices.* Academic Press.

V.Y. Pan, R.E. Rosholt D. Ivolgin, B. Murphy, Y. Tang & X. Yan (2008). Additive preconditioning for matrix computations. In *Third International Computer Science Symposium in Russia, CSR 2008 Moscow, Russia, June 7-12, 2008 Proceedings*, LNCS 5010, 327–383. Springer Verlag. URL `http://dx.doi.org/10.1007/978-3-540-79709-8_37`.

A. Schönhage (1971). Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica* **1**, 139–144. URL `http://dx.doi.org/10.1007/BF00289520`.

A. Schönhage & V. Strassen (1971). Schnelle Multiplikation grosser Zahlen. *Computing* **7**, 281–292. URL `http://dx.doi.org/10.1007/BF02242355`.

A. Storjohann (1996). Near optimal algorithms for computing Smith normal forms of integer matrices. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation ISSAC '96,* Zürich, Switzerland, Y. N. Lakshman, editor, 267–274. ACM Press, New York. URL `http://doi.acm.org/10.1145/236869.237084`.

A. Storjohann (1997). A solution to the extended gcd problem with applications. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation ISSAC '97,* Maui HI, W. W. Küchlin, editor, 109–116. ACM Press, New York. URL `http://dx.doi.org/10.1145/258726.258762`.

A. Storjohann (2000). *Algorithms for Matrix Canonical Forms*. Ph.D. thesis, Swiss Federal Institute of Technology, ETH–Zurich. URL http://dx.doi.org/10.3929/ethz-a-004141007.

A. Storjohann (2005). The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity* **21**(4), 609–650. URL http://dx.doi.org/10.1016/j.jco.2005.04.002. Festschrift for the 70th Birthday of Arnold Schönhage.

A. Storjohann & T. Mulders (1998). Fast algorithms for linear algebra modulo N. In *Algorithms — ESA '98*, G. Bilardi, G. F. Italiano, A. Pietracaprina & G. Pucci, editors, LNCS 1461, 139–150. Springer Verlag. URL http://dx.doi.org/10.1007/3-540-68530-8_12.

W. Zhou, G. Labahn & A. Storjohann (2014). A deterministic algorithm for inverting a polynomial matrix. *Journal of Complexity*. URL http://dx.doi.org/10.1016/j.jco.2014.09.004.

Manuscript received January 7, 2009

Arne Storjohann
David R. Cheriton School of
Computer Science
University of Waterloo
Ontario, Canada
astorjoh@uwaterloo.ca
https://cs.uwaterloo.ca/~astorjoh/