LIFTING TOEPLITZ/HANKEL COMPUTATIONS

by

BRIAN JOSEPH MURPHY

A dissertation
submitted to the Graduate Faculty in Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
The City University of New York

2007

UMI Number: 3283176

UMI Microform 3283176

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript
has been read and accepted for the Graduate Faculty in Computer Science
in satisfaction of the dissertation requirement for the degree of
Doctor of Philosophy.

APRIL 23, 2007	VICTOR PAN
Date	Chair of Examining Committee
APRIL 24, 2007	THEODORE BROWN
Date	Executive Officer

Dr. Victor Pan, Adviser and Chair
The Graduate School and University Center, Lehman College

Dr. Theodore Brown
The Graduate School and University Center, Queens College

Dr. Michael Anshel
The Graduate School and University Center, The City College

Dr. Stathis Zachos
The Graduate School and University Center, Brooklyn College

Dr. Mohammad Tabanjeh
Virginia State University

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

LIFTING TOEPLITZ/HANKEL COMPUTATIONS

by

BRIAN JOSEPH MURPHY

Adviser: Dr. Victor Pan

The arithmetic cost bounds for solving a Toeplitz or Hankel linear system of equations is $O(n \lg^2 n)$. The progression of algorithmic development that led to this bound includes what have come to be referred to as "fast" and "superfast" algorithms. First came the "fast" algorithms crafted by Levinson in 1947, Durbin in 1959, and Trench in 1964 and 1965. The "fast" algorithms each perform $O(n^2)$ arithmetic operations. Later "superfast" algorithms were devised, in particular the BYG algorithm by Brent, Gustavson, and Yun in 1980 [BGY80] and the MBA divide and conquer algorithm by Morf in 1974 [M74] and 1980 [M80] and Bitmead and Anderson in 1980 [BA80]. These "superfast" algorithms both require $O(n \lg^2 n)$ arithmetic operations and can be implemented numerically with finite precision.

Unfortunately, numerical instability plagues these Toeplitz and Hankel "superfast" numerical linear solvers (see Bunch 1985 [B85]) and some large and important classes of Toeplitz and Hankel matrices are ill-conditioned (see Tyrtyshnikov 1994 [T94]). Therefore, absent application of exceedingly high precision these "superfast" numerical linear solvers produce invalid results for many highly significant Toeplitz and Hankel linear systems. This dilemma led to the development of algebraic (or symbolic) techniques to simultaneously bound the arithmetic cost and the precision of computation. Such techniques have resulted in implementations of the algorithms that are slower, but error free for ill-conditioned Toeplitz and Han-

kel input matrices. Algebraic (or symbolic) implementations typically rely upon utilization of the *Chinese remainder algorithm* (see [GG99]], [PW02], and [WP03]).

Herein Hensel's p-adic lifting has been leveraged in alternative algebraic methods for implementing "superfast" Toeplitz/ Hankel linear solvers. This approach holds important advantages over those using the *Chinese remainder algorithm*. In addition, Hensel's p-adic lifting has been extended to apply q-adic lifting for $q = 2^s$ where s is an integer, herein refered to as Binary lifting. This has allowed many of the expensive modular computations required in the lifting steps to be carried out often implicitly and practically for "free" as a result of the efficiencies inherent in this regard by the binary nature of today's computer hardware.

Acknowledgements

I would like to thank the following:

My adviser, Dr. Victor Pan, for offering me the opportunity to work on many fascinating projects with a world class Mathematician and Computer Scientist who is also one of the most genuinely kind persons I've come to know.

The faculty of the Doctoral Program in Computer Science at The Graduate School and University Center of The City University of New York. Expressly:

Dr. Theodore Brown, Executive Officer. You provided the stimulus that led ultimately to completion of my task.

Dr. Stanley Habib, Executive Officer when I was admitted to the program. Your dedication to your students is something I will strive to replicate in my carrer.

The faculty of the Department of Mathematics and Computer Science at Lehman College of The City University of New York. So many of you made a significant impact in my life. Expressly:

Dr. Robert Feinerman, Chairman. One day 14 years ago when it was the furthest thing from my mind, you asked me if I would consider teaching a class at Lehman. This is where it's led me. Your support was invaluable along the way.

Dr. Connor Lazarov, the graduate adviser, during my time in Lehman's Bachelors and Masters degree programs and at the start of my teaching carrer there. You taught my first undegergraduate course and my last graduate course at Lehman. In between and thereafter we had many interesting conversations. I miss our chats. I am saddened that you're no longer with us.

My entire extended family and all of my friends. Expressly:

My Wife Betsy. Because of you the last twenty-one years are the most precious of my life. As the next twenty-one unfold and each twenty-one thereafter, I look forward to having you by my side. I can't imagine it any other way. You're a part of me. I love you. You're my best friend and always will be.

My Daughter Virginia. I really appreciate the sacrifice you've made these last few months. I'm sorry that our time together recently was limited more than either you or I cared for it to be. Someday, maybe I can return the favor. Perhaps as you prepare your dissertation and for its defense, though I hope you'll manage your time better than I, so that the limit need not be as acute. I love you Virginia. I'm so glad that you are my daughter. You are my sunshine.

My parents, Richard and Marlene. I always knew how much you loved me. You left no room for doubt. I always recognized the sacrifices you gladly made for me. They were plain for all to see. I always had faith that I could count on you. You never let me down. Only now, I find, I always underestimated all you did for me. Thank you so much for everything you've done for me throughout my life. You put me on the right path and helped me to stay there. I love you both very very much.

Finally, my grandfather, Harry Lamm. Even though you were only able to be here for my first nine years, so that I knew you only from childhood's perspective, no other person influenced and inspired me throughout my life nearly as much nor quite the way you did. I can imagine how you would have delighted in this moment. More than thirty years have past, yet as I write this, I miss you as if you left us yesterday.

Contents

List of Tables

List of Figures

1 Introduction

Algebraic (or symbolic) algorithms for sparse and/or structured linear systems of equations with integer coeficients are examined herein. Hensel's lifting, yields the solution in nearly optimum Boolean time. Furthermore, lifting is performed modulo a reasonably bounded power of two to implement the algorithms in binary within a fixed computer precision, (e.g., the IEEE standard double precision), while minimizing the number of the word operations involved. The study is specific to solving Toeplitz linear systems and readily extends to some fundamental computations with various sparse and/or structured matrices as well as computing univariate polynomial gcds, lcms, and resultants. It also relates lifting to some popular numerical computations.

This paper combines techniques for lifting and other algebraic (or symbolic) computations modulo integers as well as computations with sparse and structured matrices. Included are the details from both areas, e.g., some basic results on Toeplitz and Hankel matrices and on the reconstruction of the rational solution from the p-adic solution.

Algebraic (or symbolic) algorithms are a customary tool for the solution of integer linear systems of equations $M\mathbf{x} = \mathbf{b}$. One of the basic techniques is Hensel's p-adic lifting, proposed for solving general linear systems by Moenck and Carter 1979 in [MC79] and Dixon 1982 in [D82] and having strong similarity to Wilkinson's popular algorithm for numerical iterative refinement in [S80], [GL96, Section 3.5.3], [H02] (see Remark 7.1). Lifting is performed with precision of the order of $\lceil \lg p \rceil$ bits, but the exact rational multi-precision solution can be readily reconstructed from its output.

The basic operation in each recursive step for lifting of a system of linear equations is multiplication of an input matrix M by a vector and multiplication

of the precomputed inverse of M modulo a fixed prime p by a vector. This makes lifting attractive for parallel processing and particularly effective for sparse and/or structured linear systems for which these multiplications are fast. Such linear systems can be solved with lifting in time optimal within a polylogarithmic factor (see Section 8). Here the optimality is in terms of the numbers of both Boolean (bit) operations involved as well as arithmetic operations performed with the precision bounded by the length of a computer word, (e.g., with the IEEE standard double precision).

A boolean time bound nearly as small is supported by Newton's p-adic lifting, which has a strong similarity to Newton's iteration for matrix inversion (see, e.g., [P01, chapter 6] and the biliography therein), but each Newton's lifting step doubles the precision of computing, and substantial additional effort is needed to reverse this expansion to ensure fixed (e.g., double) precision operation.

To enable binary computations obviating the need for many expensive division and modulus operations for each iteration and to keep computations within computer precision throughout (except, at the last stage during reconstruction of the exact rational solution, represented by long binary numbers), one should apply *binary lifting*, that is initialize Hensel's as well as Newton's lifting modulo a properly bounded power of two. The policy adopted herein is saturated initialization, that is, lifting is initialized modulo 2^d for an integer d chosen to be a little smaller than the length of a computer word. Compared to the customary initialization with random primes p, this means stepping on new ground where lifting is done in rings, instead of fields. In this paper, algorithms are adjusted respectively and still they are nearly optimal for sparse and/or structured integer linear systems of equations. Furthermore, this new binary lifting is still simple to analyze and to implement.

Exemplified in this study is the highly important case of Toeplitz linear systems of equations which are treated in some detail, but the algorithms and complexity estimates can actually be applied to a number of fundamental computations with various important classes of sparse and/or structured matrices.

The well known correlation between computations with Toeplitz and Hankel matrices and polynomials enables further extension of these algorithms and their nearly optimal cost estimates to computing the greatest common divisor (hereafter gcd), the least common multiple (hereafter lcm), resultant, and Padé approximations of a univariate polynomial as well as interpolating rational functions.

Lifting is linked to matrix determinants in two ways. It facilitates their algebraic (or symbolic) computations and where the determinant is known the recovery of the exact rational solution from the output of lifting can be simplified.

Finally, comparison and possibly extension of binary lifting to numerical solution of ill conditioned linear systems of equations leads to some interesting research challenges (see Section 13).

2 Definitions and Preliminaries

2.1 Foundation

Definition 2.1. $\mathbb{Z}$ *is the ring of integers.*

Definition 2.2. $\mathbb{Q}$ *is the field of rational numbers.*

Definition 2.3. $\lg$ *is the base 2 log, i.e.* $\log_2$.

Definition 2.4. $\operatorname{ord}_q(m)$, *the order of* q *in* m, *is the maximal integer* l *such that* q^l *divides* m.

Definition 2.5. $\nu(y)$ *is the numerator, and* $\delta(y) \geq 1$ *is the denominator in the ratio* $y = \nu(y)/\delta(y)$ *of two coprime integers* $\nu(y)$ *and* $\delta(y)$.

2.2 Matrices

Definition 2.6. $M = (m_{i,j})_{i,j=1}^{k,l}$ *is a* $k \times l$ *matrix with rational or integer entries* $m_{i,j}$, $M \in \mathbb{Q}^{k \times l}$ *or* $M \in \mathbb{Z}^{k \times l}$, *respectively.* $\mathbf{v} = (v_i)_{i=1}^{k}$ *is a vector.* I *is the identity matrix of a proper size.* I_l *is the* $l \times l$ *identity matrix.* M^T *is the transpose of* M.

Definition 2.7. $\det M$ *and* $\operatorname{adj} M = (d_{i,j})_{i,j=1}^{n}$ *denote the determinant and the adjoint of an* $n \times n$ *matrix* $M = (m_{i,j})_{i,j=0}^{n-1}$ *where* $d_{i,j} = \det M_{i,j}$ *and the submatrix* $M_{i,j}$ *is obtained by deleting the* $\imath$*-th row and the* $\jmath$*-th column of* M. $\operatorname{adj} M = M^{-1} \det M$ *if* M *is nonsingular.*

Definition 2.8. *For a matrix* $M = (m_{i,j})_{i,j=1}^{k,l}$ *and vectors* $\mathbf{b} = (b_i)_{i=0}^{k}$, $\mathbf{x} = (x_i)_{i=0}^{l}$, $\mathbf{u} = (u_i)_{i=0}^{l}$, *and* $\mathbf{v} = (v_i)_{i=1}^{l}$, *where* m_{ij}, b_i, u_i, v_i *are constants,* x_j *are unknown values,* $i = 1, \ldots, k$; $j = 1, \ldots, l$; $k, l \in \mathbb{Z}$; *and* $k, l \geq 1$, *define the dot product of two vectors* $\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_l v_l = \sum_{i=1}^{l} u_i v_i$, *a matrix-by-vector product* $M\mathbf{x} = (\sum_{j=1}^{l} a_{ij} x_j)_{i=1}^{k}$, *and the solution* $\mathbf{x}$ *to a linear system* $M\mathbf{x} = \mathbf{b}$ *of* k *equations with* l *unknowns as the set of values* s_j *such that the substitution* $x_j = s_j$, *for* $j = 1, \ldots, n$ *simultaneously satisfies all equations in the system.*

Definition 2.9. *A linear system* $M\mathbf{x} = \mathbf{b}$ *for* $\mathbf{b} \neq \mathbf{0}$ *has exactly one solution and is called nonsingular if and only if its coefficient matrix* M *has inverse* M^{-1} *such that* $M^{-1}M = MM^{-1} = I$. M *is nonsingular if and only if all its rows (columns) are linearly independent, i.e., no row (column) is a linear combination of other rows (columns). In this case the matrix* M *must be a square* $n \times n$ *matrix and* $\det M \neq 0$.

Clearly, if M is nonsingular, $M\mathbf{x} = \mathbf{b}$ is satisfied by $\mathbf{x} = M^{-1}\mathbf{b}$, but generally solving a nonsingular linear system $M\mathbf{x} = \mathbf{b}$ is a simpler task than computing M^{-1}.

Definition 2.10. $|M|$ *denotes the column norm of a matrix* $M = (m_{i,j})_{i,j}$, $|M| = ||M||_1 = \max_j \sum_i |m_{i,j}|$. $|\mathbf{v}| = \sum_i |v_i|$ *denotes the* ℓ_1*-norm of a vector* $\mathbf{v} = (v_i)_i$. $\alpha(M) = \max_{i,j} |m_{i,j}|$. $\beta(\mathbf{v}) = \max_i |v_i|$.

Definition 2.11. $v_M \leq 2n^2 - n$ *and* i_M *are the minimum numbers of arithmetic operations sufficient to multiply a given* $n \times n$ *matrix* M *by a vector and to invert it, respectively.*

Hadamard's estimate below is known to be sharp in the worst case, but is an overestimate on the average according to [ABM99].

Fact 2.1. Hadamard's Bound

$$|\det M| \leq \prod_j (\Sigma_i m_{i,j}^2)^{1/2} \leq (\alpha(M)\sqrt{n})^n,$$
$$|\det M| \leq |M|^n$$

and

$$|\operatorname{adj} M| \leq (n-1)\alpha(\operatorname{adj} M),$$
$$|\operatorname{adj} M| \leq (\alpha(M)\sqrt{n-1})^{n-1}(n-1),$$
$$|\operatorname{adj} M| \leq (n-1)|M|^{n-1}$$

for an $n \times n$ matrix $M = (m_{i,j})_{i,j}$.

Hereafter let $\mathbf{b} \neq 0$, $n > 2$, $|M| > 2$, and so $\lg n > 1$, $\lg |M| > 1$.

Definition 2.12. *For two integers $q > 0$ and $s > 1$, a matrix M in $\mathbb{Z}_{qs}^{n \times n}$ is* factor-q nonsingular *modulo qs if there exists a matrix Q in $\mathbb{Z}_{qs}^{n \times n}$ such that*

$$MQ \bmod (qs) = qI. \tag{2.1}$$

2.3 Modular Arithmetic

Definition 2.13. *$\mathbb{Z}_m$ is the set of integers modulo m, $m \in \mathbb{Z}$, $m > 1$.*

Fact 2.2. *For $r, n, m \in \mathbb{Z}$, $0 \leq r < m$, $r = n \bmod m$ is read "r is the remainder when n is divided by m." Equivalently, $n \equiv r \pmod m$ is read "n and r are congruent modulo m." Each of these implies that $m|(n - r)$, therefore $n = r + tm$ for $t \in \mathbb{Z}$, $t \geq 0$.*

Theorem 2.1. *The mapping from $\mathbb{Z} \to \mathbb{Z}_m$, $m \in \mathbb{Z}$ is a homomorphism, implying:*

$$(x + y) \bmod m = ((x \bmod m) + (y \bmod m)) \bmod m,$$
$$(x - y) \bmod m = ((x \bmod m) - (y \bmod m)) \bmod m,$$
$$(xy) \bmod m = ((x \bmod m)(y \bmod m)) \bmod m.$$

Theorem 2.2. The Chinese Remainder Theorem. *$x \in \mathbb{Z}$ can be uniquely represented $\pmod{m_1 m_2 m_3 \cdots m_n}$ given pair-wise relatively prime moduli m_1, m_2, m_3, ..., m_n and integer remainders r_1, r_2, r_3, ..., r_n such that $r_i = x \bmod m_i$, for all $i = 1, \ldots, n$.*

If $2x < m_1 \ldots m_n$, the *Chinese Remainder Algorithm* provides a method for determining x given the r_i and m_i, for all $i = 1, \ldots, n$. One of the many applications of the *Chinese Remainder Theorem* allows the result of integer computations

to be determined by carrying out multiple instances of the computation modulo different relatively prime moduli and for the result to be constructed from the resulting pairs of remainders and moduli via the *Chinese Remainder Algorithm.*

The *Euclidean Algorithm* finds the greatest common divisor (gcd) of two integers without need of factoring. It is based on the simple recursion:

Theorem 2.3. $g = \gcd(x, y) = \begin{cases} \gcd(y, x \bmod y) & if \quad x \bmod y \neq 0 \\ y & otherwise. \end{cases}$

The *Extended Euclidean Algorithm*, by adding a small amount of bookkeeping to the computations of the *Euclidean Algorithm*, not only finds gcds, but as a by-product of calculating those gcds also determines cofactors s and t such that $g = sx + ty$, $|t| < x$, $|s| < y$. These cofactors are modular multiplicative inverses of $x \bmod y$ and of $y \bmod x$ if x and y are relatively prime, that is, if $g = 1$.

Definition 2.14. *The modular multiplicative inverse of an integer* $n \bmod m$*, is that integer* $n^{-1} \bmod m$ *such that* $(n * n^{-1}) \bmod m = 1$*. If the modulus* m *and an integer* n*,* $0 < n < m - 1$*, are relatively prime, i.e.,* $\gcd(m, n) = 1$*, then the modular multiplicative inverse of* $n \bmod m$ *does in fact exist and can be found using the* Extended Euclidean Algorithm*. Otherwise it does not exist.*

Clearly then, a modulus p, a prime number, has the advantage that each element $x \in \mathbb{Z}_p$ other than zero has a modular multiplicative inverse $(\bmod p)$.

2.4 p-adic Integers

Definition 2.15. p-adic expansion of an integer: *Every integer can be represented as a polynomial in* p *with coefficients* $0, \ldots, p-1$ *represented by a vector. Rationals can also be represented as p-adics if the denominator is relatively prime to* p*.*

2.5 The Issue of Numerical Instability

Due to their finite nature, computers are limited in terms of the precision with which they are capable of representing numeric values. Floating point numbers, which stand as substitutes for real numbers in computer computations, suffer inaccuracies due to precision issues. Calculations done on floating point numbers can compound the problem both because the floating point inputs to the calculation may already be rounded, by necessity, and/or because computations using those floating point parameters may generate rounding errors of their own. This makes it quite possible for all bits in a significand, generated by multiple floating point operations, to fall within the error and therefore be of no significance in expressing the solution to the problem.

Definition 2.16. *The condition number of a matrix is the ratio of its minimal singular value to its maximal singular value*

$$cond_2(M) = \sigma_1/\sigma_n = ||A||||A^{-1}||.$$

A matrix M is considered "ill-conditioned" if its condition number is large.

The condition number of a matrix helps gauge the likelihood of success in application of numerical methods for solving linear systems of equations. The rule of thumb is that the output error is the product of the input error and the condition number. Furthermore, in 1961–65, J. H. Wilkinson proved that for most fundamental matrix computations, the rounding errors accumulate to an equivalent of the input error. It follows that the accuracy of the output of numerical matrix computations is poor where the input matrix has a large condition number.

Conditioning problems can be addressed by using greater machine word size or wider floating point values and their operations implemented in software. The hardware solution is prohibitively expensive. The software solution slows compu-

tations. Yet, each can still fail to provide reliable results, given that the precision provided by the machine is still finite.

Clearly, it is desirable to bound the precision of computing in such cases. One of the common means is through modular arithmetic. Computing modulo an integer m bounds the precision of computing by $\lceil \lg m \rceil$.

2.6 Cost Estimates for Convolution and Multiplication

Let $m(n)$ denote the number of field operations needed to convolve two n-vectors, or equivalently to multiply two polynomials. $m(n) \geq 2n-1$ (the informational lower bound), $m(n) = O(n \lg n)$ over the fields or rings that support FFT, and more generally,

$$m(n) \leq c_{class} n^2, \quad m(n) \leq c_k n^{\lg 3}, \quad m(n) \leq (c_{ck} n \lg n) \lg \lg n, \tag{2.2}$$

over any field or ring with unity. Here $\lg 3 = 1.58496\ldots$, c_{class}, c_k, and c_{ck} are three constants, $0 < c_{class} < c_k < c_{ck}$, and the above bounds are supported by the classical, Karatsuba's [K90], and Cantor and Kaltofen's algorithms; practical choice among them depends on the degree n (see Bernstein, [B03], and [GG03]).

Bit operation costs are estimated, reflecting arithmetic cost and precision of computation. To each arithmetic operation performed over integers modulo q, that is, with d-bit precision, for $d = \lceil \lg q \rceil$, assign the cost of $O(\mu(d))$ bit operations, where $\mu(d)$ denotes the bit operation complexity of multipication of two integers modulo q, $\mu(d) \geq 2d-2$ (an information lower bound),

$$\mu(d) \leq C_{class} d^2, \quad \mu(d) \leq C_k d^{\lg 3}, \quad \mu(d) \leq (C_{ss} d \lg d) \lg \lg d, \tag{2.3}$$

C_{class}, C_k, and C_{ss} are three constants, $0 < C_{class} < C_k < C_{ss}$, and the above bounds are supported by the classical algorithm and those of Karutsuba 1963 and Schönhage and Strassen 1971; all these algorithms are practically used, depending on the precision d (see [K98], [B03], [GG03]).

3 Solving Linear Systems

3.1 Gausian Elimination

Direct methods for solving linear systems of equations and finding matrix inverses apply only arithmetic operations coupled with comparisons and matrix row (column) interchange. They output exact results assuming error-free computaions. The most commonly used direct algorithm for handling linear systems of equations is Gaussian Elimination. It requires $\frac{2}{3}n^3 + O(n^2)$ arithmetic operations for a linear system of n equations with n unknowns.

3.2 Complete Recursive Triangular Factorization

Divide and conquer algorithms for matrix computations trace their roots back at the very least to [S69] and [AHU74]. Such algorithms solve the given matrix problem by solving "easier" subproblems and build on those results to determine the sought after solution. In the complete recursive triangular factorization of a matrix, multiple applications of matrix multiplication are substitued for directly inverting a matrix. This recursive factorization was used by V. Strassen in [S69] to prove that the exponent of the arithmetic complexity of matrix inversion is not greater than that of matrix multiplication.

Definition 3.1. *Given an $n \times n$ matrix M, $M^{(k)}$ where $k \leq n$ is a submatrix of M composed of rows and columns $1, \ldots, k$ of matrix M, i.e., $M^{(k)}$ is a northwestern (leading principle) submatrix of M. If all submatrices $M^{(k)}$ for $k = 1, \ldots,$ rank(M) are nonsingular, $M^{(k)}$ is said to* have generic rank profile. *If a matrix M in addition to having generic rank profile is itself nonsingular then M is a* strongly nonsingular matrix.

An $n \times n$ strongly nonsingular matrix M can be represented as a 2×2 block

matrix,

$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}, \tag{3.1}$$

where, by Definition 3.1, the $k \times k$ submatrix of M, $M_{00} = M^{(k)}$, is also strongly nonsingular.

The 2×2 block matrix above can be factored through block Gaussian elimination into the lower triangular, diagonal, and upper triangular 2×2 block matrices

$$M = \begin{pmatrix} I & 0 \\ M_{10}M_{00}^{-1} & I \end{pmatrix} \begin{pmatrix} M_{00} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix} \tag{3.2}$$

where the $(n-k) \times (n-k)$ matrix,

$$S = M_{11} - M_{10}M_{00}^{-1}M_{01}, \tag{3.3}$$

is called the *Schur complement of* M_{00} *in* M and is denoted by $S(M_{00}, M)$.

Theorem 3.1. *If a matrix* M *is strongly nonsingular, then its Schur complement* $S = S(M_{00}, M)$ *is strongly nonsingular.*

Given the factoriztion of M in (3.2) and its strongly nonsingular M_{00} and S, it is easy to see that

$$M^{-1} = \begin{pmatrix} I & -M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix} \begin{pmatrix} M_{00}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -M_{10}M_{00}^{-1} & I \end{pmatrix}. \tag{3.4}$$

Since the matrix M_{00} is strongly nonsingular, its Schur complement S can be obtained in $n-k$ steps of Gaussian elimination. Expanding (3.3) obtains

$$M^{-1} = \begin{pmatrix} M_{00}^{-1} + M_{00}^{-1}M_{01}S^{-1}M_{10}M_{00}^{-1} & -M_{00}^{-1}M_{01}S^{-1} \\ -S^{-1}M_{10}M_{00}^{-1} & S^{-1} \end{pmatrix}. \tag{3.5}$$

Theorem 3.2. S^{-1} *is the southeastern* $(n-k) \times (n-k)$ *submatrix of* M^{-1}.

Due to Theorem 3.1 and given that M is strongly nonsingular, M_{00} and S are strongly nonsingular. Therefore, the same block factorizations applied to M in (3.2) and (3.4) can be applied to both M_{00} and S. Obviously this downward pattern

can continue recursively until it ends with the inversion of the 1×1 submatrix $M^{(1)}$ which is a scalar. At this stage the inverse of $M^{(1)}$, its Schur complement S_1, and the inverse of the Schur complement S_1^{-1} are determined. Actual computation begins with the 1×1 matrix $M^{(1)}$ and recursively extends its inverse to the inverses of the matrices $M^{(2)}$, $M^{(3)}$, ... For each recursive step of the return sequence, the inverse of $M^{(2i)}$ is determined using $(M^{(i)})^{-1}$ and S_i^{-1} as indicated in (3.4) and S_{2i}^{-1} is determined via complete recursive triangular factorization.

Fact 3.1. *Complete recursive triangular factorization can be extended to compute the solution* $\mathbf{x} = M^{-1}\mathbf{b}$ *to a linear system* $M\mathbf{x} = \mathbf{b}$ *(of course) and* $\det M$ *for any strongly nonsingular matrix* M*. Observe that* $\det M = (\det M_{00})(\det S)$ *and successively compute* $\det M_{00}$*,* $\det S$*, and* $\det M$*.*

Theorem 3.3. *If* M *is a real nonsingular matrix, then* $M^T M$ *and* MM^T *are strongly nonsingular.*

Fact 3.2. $M^{-1} = (M^T M)^{-1} M^T = M^T (MM^T)^{-1}$.

Fact 3.3. $(\det M)^2 = \det(M^T M) = \det(MM^T)$.

Theorem 3.3 and Facts 3.1, 3.2, and 3.3 imply that complete recursive triangular factorization can be used to find M^{-1}, the solution $\mathbf{x} = M^{-1}\mathbf{b}$ to a linear system $M\mathbf{x} = \mathbf{b}$ and $\det M$ for any real nonsingular matrix M.

Algorithm 3.1. *The Balanced CRTF of a Strongly Nonsingular Matrix.*

INPUT: *A strongly nonsingular $n \times n$ matrix M.*

OUTPUT: *M^{-1} and the BCRTF of M and M^{-1}.*

COMPUTATIONS: *Define the BCRTF tree $T(M) = T_{n,n}$, then recursively compute and invert all matrices associated with its nodes according to the following rules:*

1. *The left child of any node is a leading principle block of its parent.*
2. *Invert each of the leaves (1-by-1 matrices) directly, then recursively invert all other nodes based on factorization* (3.4) *and its recursive extension. That is, in each recursive step:*
   a. *first invert the left child;*
   b. *then compute and invert the right child;*
   c. *and finally invert their parent based on* (3.4) *or its extension.*
3. *Terminate when the root M is inverted.*

Algorithm 3.1 can be extended to compute the solution $\mathbf{x} = M^{-1}\mathbf{b}$ to a linear system $M\mathbf{x} = \mathbf{b}$ and $\det M$.

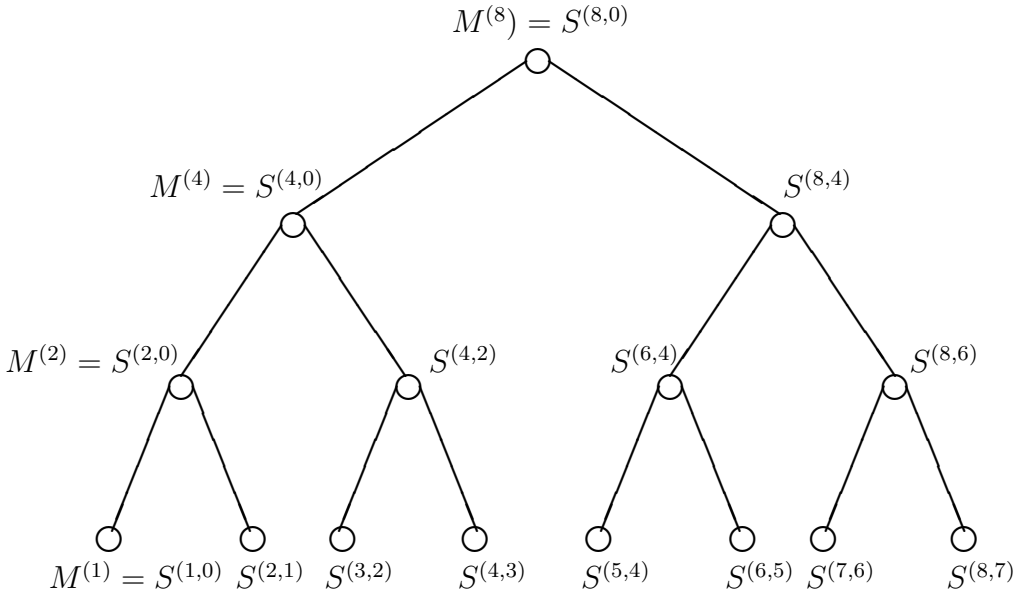

Figure 1: Generic BCRTF tree $T_{8,8} = T(M)$ for an 8×8 matrix M.

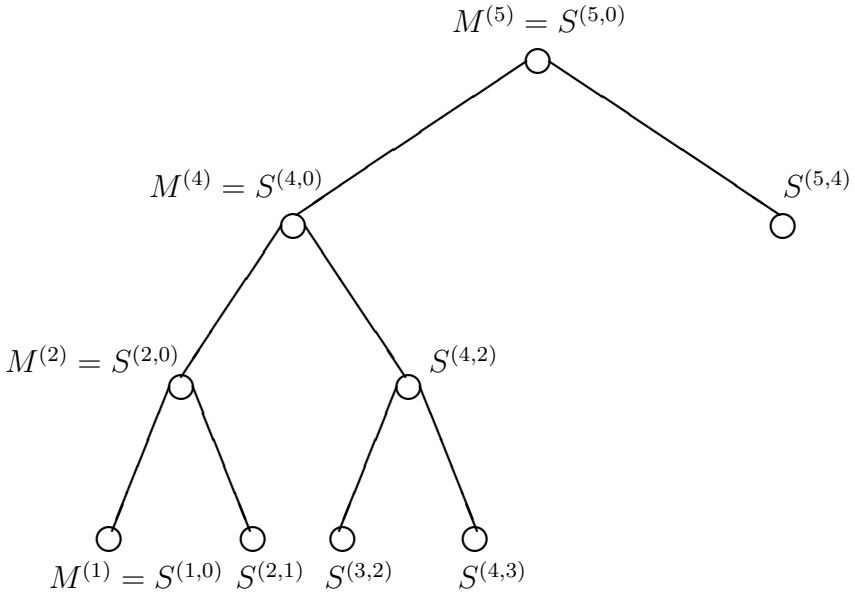

Figure 2: Generic BCRTF tree $T_{5,5} = T(M)$ for a 5×5 matrix M.

4 Toeplitz and Hankel

4.1 Toeplitz and Hankel Matrices

Clearly an $n \times n$ square matrix M can be defined by n^2 entries, but there are highly important cases where the entries of the matrix exhibit a pattern that allows a relatively small number of parameters (on the order of n) to define all entries. For structured matrices, in particular for Toeplitz and Hankel matrices, the performance of the divide and conquer algorithm of Section 3.2 improves dramatically as multiplying a matrix by a vector upon which they depend is simplified.

Definition 4.1. *$T = (t_{i,j})_{i,j}$ is a* Toeplitz matrix *if $t_{i,j} = t_{i+1,j+1}$ for every pair of its entries $t_{i,j}$ and $t_{i+1,j+1}$, i.e., entries of a Toeplitz matrix are invariant along each diagonal. A Toeplitz matrix is defined by the $2n-1$ entries of its first row and first column. $Z(\mathbf{v})$ is the lower triangular Toeplitz matrix defined by its first column $\mathbf{v}$. For any scalar f and vector $\mathbf{v} = (v_i)_{i=0}^{n-1}$, define the $n \times n$ unit f-*circulant matrix $Z_f = (z_{i,j})$, $z_{i,i-1} = 1$, $i = 2, \ldots, n$; $z_{1,n} = f$, $z_{i,j} = 0$ *for other pairs (i, j), and the f-circulant matrix with the first column $\mathbf{v}$, $Z_f(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_f^i$. (Note that $Z_f^n = fI$.)*

Generating f-Circulant Matrix Z_f **Toeplitz Matrix**

$$\begin{pmatrix} 0 & 0 & \cdots & f \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{pmatrix}$$

Definition 4.2. *$H = (h_{i,j})_{i,j}$ is a* Hankel matrix *if $h_{i,j} = h_{i-1,j+1}$ for every pair of its entries $h_{i,j}$ and $h_{i-1,j+1}$, i.e., entries of a Hankel matrix are invariant along each anti-diagonal. Define the* reflection matrix *$J = (j_{g,h})$, $j_{g,n+1-g} = 0$ for $g = 0$,*

..., $n-1$, $j_{g,h}=0$ for $h+g\neq n$. ($J\mathbf{v}=(v_{n-i-1})_{i=0}^{n-1}$ for any vector $\mathbf{v}=(v_i)_{i=0}^{n-1}$, $J^2=I$.) (Multiplying the matrix J by any vector reverses that vector $\mathbf{v}=(v_i)_{i=0}^{n-1}$, that is, $J\mathbf{v}=(v_{n-i-1})_{i=0}^{n-1}$, $J^2=I$.)

Hankel Matrix

$$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_1 & h_2 & \iddots & h_n \\ \vdots & \iddots & \iddots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-2} \end{pmatrix}$$

Theorem 4.1. *TJ and JT are Hankel matrices if T is a Toeplitz matrix, and HJ and JH are Toeplitz matrices if H is a Hankel matrix.*

With Toeplitz and Hankel linear systems immediately reduced to one another attention will be focused on Toeplitz matrices only.

Fact 4.1. *For $n\times n$ Toeplitz matrices there are $2n-1$ defining entries.*

The following well-known results (see, e.g., [P01, Chapter 2]) reduce matrix-by-vector multiplications for both Toeplitz matrices T and their inverses T^{-1} to polynomial multiplication or equivalently vector convolutions. Similar complexity results can be obtained by relying on the factor-circulant representations of the matrices T and T^{-1} (see [P01, Section 2.6 and Exercise 2.24c]).

Theorem 4.2. *Multiplication of an $n\times n$ Toeplitz matrix T by a vector is a subproblem of vector convolution between a $(2n-1)$ dimensional vector and an n dimensional vector whose entries are given by the entries of the input matrix and vector, respectively. If the Toeplitz matrix T is triangular then both vectors are n dimensional vectors.*

Corollary 4.1. *An $n \times n$ Toeplitz matrix T can be multiplied by a vector in $m(k)$ arithmetic operations for $m(n)$ in (2.2) and $k = 3n - 2$; the bound decreases to $m(k)$ for $k = 2n - 1$ if T is a triangular Toeplitz matrix.*

The next theorem of Heinig 1979 [H79] extends the Gohberg–Semencul formula of 1972.

Theorem 4.3. *Let $T = (t_{i,j})_{i,j=0}^{n-1}$ be a nonsingular Toeplitz matrix. Let t_{-n} be any scalar (e.g., $t_{-n} = 0$), and write $t_{i-j} = t_{i,j}$ for $i, j = 0, \ldots, n-1$. Let $p_n = -1$, $\mathbf{t} = (t_{i-n})_{i=0}^{n-1}$, $\mathbf{p} = (p_i)_{i=0}^{n-1} = T^{-1}\mathbf{t}$, $\mathbf{q} = (p_{n-i})_{i=0}^{n-1}$, $\mathbf{v} = T^{-1}\mathbf{e}_1$, $\mathbf{e_1} = (1, 0, \ldots, 0)^T$, $\mathbf{u} = ZJ\mathbf{v}$. Then $T^{-1} = Z(\mathbf{p})Z^T(\mathbf{u}) - Z(\mathbf{v})Z^T(\mathbf{q})$.*

Hereafter the $n \times 2$ matrix $(\mathbf{v}, \mathbf{p})$ for the above vectors $\mathbf{p} = p(t_{-n})$ (for a fixed t_{-n}) and $\mathbf{v}$ is called a *generator* for T^{-1}.

The next theorem is a corollary of Theorems 4.2 and 4.3.

Theorem 4.4. *$4m(k) + n$ arithmetic operations for $m(n)$ in (2.2) and $k = 2n - 1$ suffice to multiply the matrix T^{-1} by a vector provided that T is a nonsingular Toeplitz matrix and T^{-1} is given with its generator, that is, the vectors $\mathbf{p}$ and $\mathbf{v}$ in Theorem 4.3.*

4.2 Displacement Representation of Matrices

Definition 4.3. Displacement operators L of Stein and Sylvester types *map a matrix M into its displacements $L(M) = \nabla_{A,B}(M) = AM - MB$ and $L(M) = \Delta_{A,B}(M) = M - AMB$, respectively. A and B are called* operator matrices, *$r = \operatorname{rank} L(M)$ is called the L-*rank *or the* displacement rank *of M. If*

$$L(M) = GH^T, \tag{4.1}$$

where

$$G = (\mathbf{g}_1, \ldots, \mathbf{g}_l) \quad \textit{and} \quad H = (\mathbf{h}_1, \ldots, \mathbf{h}_l) \tag{4.2}$$

are $l \times n$ matrices and M and $L(M)$ are $n \times n$ matrices, then the matrix pair (G, H) is called a generator *of length l for $L(M)$ and an L-*generator *(or a* displacement generator*) of length l for M, $l \geq r$. (A pair G, H in (4.1) is nonunique for a fixed $L(M)$.) If M is an $n \times n$ matrix and l is small relative to n (say, if $l = O(1)$ as $n \to \infty$), then M is said to have L-*structure *or to be an L-*structured matrix. *(Both Sylvester and Stein type operators ∇ and Δ would suffice herein [P01]; Sylvester type operators will be used.)*

Theorem 4.5. *[P01, Theorem 4.6.4]. Given an L-generator (G_l, H_l) of length l for a matrix M having L-rank $r \leq l$, it is sufficient to use $O(l^2 n)$ arithmetic operations to compute an L-generator (G, H) of length r for the matrix M.*

The following results are immediately verifiable.

Theorem 4.6. *Let M be a nonsingular matrix, then*

$$\nabla_{A,B}(M^{-1}) = -M^{-1}\nabla_{B,A}(M)M^{-1}.$$

Theorem 4.7. *For two pairs of scalars a and b and matrices M and N of the same size and any linear operator L (in particular, for $L = \nabla_{A,B}$ for any pair of matrices A and B),*

$$L(aM + bN) = aL(M) + bL(N).$$

Theorem 4.8. *[P01, Theorem 1.5.4]. For a 5-tuple $\{A, B, C, M, N\}$ of matrices of compatible sizes,*

$$\nabla_{A,C}(MN) = \nabla_{A,B}(M)N + M\nabla_{B,C}(N).$$

Based on the results in this subsection, L-structured $n \times n$ matrices M with n^2 entries may be represented in compressed form via $2nl$ entries of their short L-generators and operate with them according to the following sequence:

COMPRESS $\longrightarrow$ OPERATE $\longrightarrow$ DECOMPRESS

Theorems 4.5–4.8 support the OPERATE stage, in which, in addition to saving memory space, usually dramatically decrease computational time. For the DECOMPRESS stage, see Theorem 4.10, [P01, Chapter 6], and [PW03].

4.3 Toeplitz-like and Hankel-like Matrices

Theorem 4.9. *For any pair of distinct scalars e and f and any Toeplitz matrix T, there exist nonunique pairs $(Z_e(\mathbf{u}), Z_f(\mathbf{v}))$ and $(Z_0(\mathbf{w}), Z_0(\mathbf{x}))$ such that $T = Z_e(\mathbf{u}) + Z_f(\mathbf{v}) = Z_0(\mathbf{w}) + Z_0^T(\mathbf{x})$. (Every matrix $Z_0(\mathbf{v})$ is lower triangular.)*

Definition 4.4. *An $n \times n$ matrix M is Toeplitz-like if $r = \operatorname{rank} L(M)$ is small relatively to n ($r = O(1)$ as $n \to \infty$), and if M is given with its L-generator of length $l = O(r)$, where $L = \nabla_{Z_e,Z_f}$, $L = \nabla_{Z_e^T,Z_f^T}$, $L = \Delta_{Z_e,Z_f^T}$, $L = \Delta_{Z_e^T,Z_f}$ for a fixed pair of scalars e and f. A matrix M is Hankel-like if MJ or JM is Toeplitz-like (and so the problems of solving Toeplitz-like and Hankel-like linear systems are reduced to each other).*

$n \times n$ Toeplitz-like matrices T generalize $n \times n$ Toeplitz and Hankel matrices. They can be represented in a compact form with their *displacement generators* made up of $O(n)$ parameters and, like the matrix T^{-1} in Theorem 4.4, can be multiplied by vectors fast. The same properties hold for the inverses of nonsingular Toeplitz-like matrices and for Hankel-like matrices $J\{T\} = \{T\}J$ where the matrix J is from Definition 4.2 and $\{T\}$ denotes the class of Toeplitz-like matrices (see some relevant bibliography in Kailath and Sayed 1999 [KS99], Pan 2000 [P00], and [P01]).

Here is the displacement of T with $l \le 2$ [P01, page 120]:

$$\nabla_{Z_f,Z_e}(T) = Z_f T - T Z_e = (Z_0 J\mathbf{t}_- - e\mathbf{t})\mathbf{e}_{n-1}^T + \mathbf{e}_0(fJ\mathbf{t} - Z_0^T\mathbf{t}_-)^T \qquad (4.3)$$

for $T = (t_{i-j})_{i,j=0}^{n-1}$, $\mathbf{t} = (t_i)_{i=0}^{n-1}$, $\mathbf{t}_- = (t_{-i})_{i=0}^{n-1}$, and any pair of scalars e and f.

Herein, it is sufficient to use e and f in the set $\{-1, 0, 1\}$ (see [P04, Appendix A]).

Theorem 4.10. *[P01, Examples 4.4.2]. Given* (4.1)*, then*

$$(e-f)M = \sum_{j=1}^{l} Z_e(\mathbf{g}_j) Z_f(J\mathbf{h}_j) \qquad \text{if } L = \nabla_{Z_e, Z_f},\ e \neq f,$$

$$(e-f)M = \sum_{j=1}^{l} Z_e^T(J\mathbf{g}_j) Z_f^T(\mathbf{h}_j) \qquad \text{if } L = \nabla_{Z_e^T, Z_f^T},\ e \neq f.$$

Theorems 4.1, 4.5, and 4.10 together imply the following corollary.

Corollary 4.2. *An $n \times n$ Toeplitz-like or Hankel-like matrix M of displacement rank r given with its displacement generator of length l can be multiplied by a vector by using $O(m(n)l)$ ring operations or alternatively $O(m(n)r + l^2 n)$ field operations for $m(n)$ in* (2.2)*.*

4.4 Compression of a Toeplitz-like Matrix

Definition 4.5. *Let $m \in \mathbb{Z}$, $U \in \mathbb{Q}^{k \times l}$, and $V \in \mathbb{Z}^{k \times l}$. V has the order $s = \mathrm{ord}_m(V)$ in m if s is the maximal integer such that $m^{-s}V \in \mathbb{Z}^{k \times l}$ for $V \neq 0$, $s = \infty$ for $V = 0$. A prime p is relatively prime to V if $\mathrm{ord}_p(V) = 0$. Let every entry of U be either 0 or the ratio of two relatively prime numbers. Then $\delta(U)$ is the least common denominator of the entries of $U \neq 0$, $\delta(0) = 1$. For $\tilde{U} = \delta(U)U \in \mathbb{Z}^{k \times l}$, $g_m(U) = \mathrm{ord}_m(\delta(U))$, then $\mathrm{ord}_m(U) = \mathrm{ord}_m(\tilde{U}) - g_m(U)$.*

Problem 4.1. *Given a generator G_V, H_V of length l for a matrix $V = L(M) \in \mathbb{Z}^{n \times k}$ of rank r, compute a generator G, H of length r for V such that*

$$g_p(G) \geq 0, \quad g_p(H) \geq 0. \tag{4.4}$$

The next algorithm solves Problem 4.1, where $G_V = V$, $H_V = I$, $l = n$.

Algorithm 4.1. *Basic Compression via Gaussian Elimination.*

INPUT: *A prime p, a matrix $V \in \mathbb{Z}^{n\times k}$ of rank r, given by its $n \times k$ entries.*

OUTPUT: *A generator G, H of length r for V.*

COMPUTATIONS: *Apply Gaussian elimination with column pivoting to compute lower triangular matrices U^T and L and a permutation matrix P such that $V = LUP$ and at every elimination step the order in p of the pivot element is minimum in its row. The output matrices G and H are defined as the $n \times r$ submatrices formed by the first r columns of L and P^TU, respectively.*

Correctness follows because all diagonal entries of U are relatively prime to p (by construction of U), U is nonsingular, $\operatorname{rank} L = \operatorname{rank} V = r$, and all but the first r columns of L vanish, so $LUP = V$.

The algorithm performs $O(nkr)$ arithmetic operations with ratios η/δ of pairs of relatively prime numbers η and $\delta > 0$ such that $\operatorname{ord}_p(\delta) = 0$, and if $r = O(1)$, then $\lg(|\eta|\delta) = O(\lg|V|)$. Here and hereafter, $|V| = |(v_{i,j})_{i,j})| = 1 + \max_{i,j}|v_{i,j}|$.

Theorem 4.11. *For $r = O(1)$, Algorithm 4.1 requires $O(nk\mu(\lg|V|))$ bit operations for $\mu(d)$ in* (2.3).

4.5 Compression of Short Generators

The following techniques compress a matrix V represented by its short although not shortest generator (cf. the proof of Theorem 1.1 in [P01]).

Algorithm 4.2. *Mulitiplication with Basic Compression.*

INPUT: *A prime p, a matrix $V \in \mathbb{Z}^{n\times k}$ of an unknown rank r for $k = O(n)$, given by its rational generator G_V, H_V of length $l \geq r$.*

OUTPUT: *A shortest rational generator G, H for V satisfying* (4.4).

COMPUTATIONS:

1. *Compute* $V = G_V H_V^T$.
2. *Apply Algorithm 4.1.*

The algorithm uses $O(kln)$ arithmetic operations. To decrease the precision of computing at stage 1, first compute the integers

$$t_G = g_p(G_V), \quad t_H = g_p(H_V), \quad t = t_G + t_H,$$

and

$$u = 1 + \max\{t, \lceil \log_p(2|V|) \rceil\}, \tag{4.5}$$

then compute in $\mathbb{Z}_{p^u}$ the matrices $p^{t_G} G_V$, $p^{t_H} H_V$ (both are relatively prime to p), $p^t V = (p^{t_G} G_V)(p^{t_H} H_V^T)$, and V; finally recover V in $\mathbb{Z}^{n \times k}$ from $V \bmod p^u$. This yields the precision of $\lceil u \lg p \rceil$ bits, so the bit cost in Stage 1 is in

$$B_+ = O(nkl\mu(u \lg p)). \tag{4.6}$$

The latter bound dominates the bound at stage 2 (based on Theorem 4.11).

The next algorithm applies to the same input as Algorithm 4.1 and also computes a shortest generator for V. For $l = O(1)$, it is faster by the factor of k but instead of (4.4) supports only the weaker bounds of

$$g_p(G) + g_p(H) \geq g_p(G_V) + g_p(H_V). \tag{4.7}$$

Algorithm 4.3. *Compression via repeated multiplication and basic compression.*

INPUT/OUTPUT: *As in Algorithm 4.2 but with* (4.7) *replacing* (4.4).

COMPUTATIONS:

1. *Apply Algorithm 4.1 to the input matrix* $\tilde{G_V} = \delta(G_V) G_V$ *to compute a generator* $\bar{G}$, $\bar{H}$ *of length* $\bar{r} = \operatorname{rank} G_V \leq l$.

2. *Compute the* $\bar{r} \times l$ *matrix* $W = \bar{H}^T H_V$ *and apply Algorithm 4.1 to the matrix* $\tilde{W} = \delta(W)W$ *to compute its shortest generator* $\hat{G}$, H.

3. *Compute the matrix* $G = \bar{G}\hat{G}/(\delta(G_V)\delta(\bar{H}^T H_V))$. *Output* G, H.

Correctness follows from correctness of Algorithm 4.1 and the observation that the matrices G and H have full rank. The arithmetic cost is $O(l^2 n)$. For $l = O(1)$, the precision of computing is $O(u \lg p)$. This yields the bit cost bound

$$B = O(n\mu(u \lg p)). \tag{4.8}$$

Remark 4.1. *Algorithm 4.3 supports Theorem 4.5 for* $G_1 = G_V$, $H_1 = H_V$ *complemented with bounds* (4.5) *and* (4.7).

Remark 4.2. *Both Algorithms 4.2 and 4.3 can be applied to scaled integer generator* $\tilde{G_V}$, $\tilde{H_V}$ *and performed in* $\mathbb{Z}_{p^u}$. *From the (scaled) output generator* G, H *in* $\mathbb{Z}_{p^u}$, *the (scaled) output in* $\mathbb{Z}$ *can be recovered because* u *is large enough.*

4.6 The MBA Algorithm

The MBA algorithm [M74], [M80], [BA80] is in essence an extension of the Complete Recursive Triangular Factorization of a matrix specifically redesigned to take advantage of the aformentioned faster matrix operations available for structured matrices of Toeplitz type. As noted in Section 3.2, as the recursion fans out and inverted matrices are determined for each $M^{(k)}$, $k = 2, \ldots, n$, the computational effort is concentrated on finding the inverse of the Schur complement S through an additional recursive step coupled with matrix multiplication. While the Schur complement of a Toeplitz matrix S and its inverse S^{-1} are neither Toeplitz matrices nor the inverses of Toeplitz matrices, they do exhibit structure and have a displacement rank of $r = 2$, which allows for the efficient use of displacement generators when inverting S at each stage of the recursion. This process requires

$O(n \lg^2 n)$ arithmetic operations, which is a significant improvement over Gaussian Elimination's $O(n^3)$.

However, in numerical implementation the MBA algorithm suffers a significant weakness. That weakness is numerical instability. Thus, since the numerical MBA algorithm is not always able to accurately represent the results of intermediate computations the final results for a large class of matrices will be in error. Significant bits are lost during computations, so that the output bears little or no resemblance to the solution sought. The bane of the MBA algorithm is ill-conditioned matrices, as it is just such matrices on which MBA performs poorly (see Bunch 1985 [B85], Tyrtyshnikov 1994 [T94]). For those cases where numerical instability wreaks havoc with the MBA algorithm, one can resort to an algebraic (or symbolic) implementation. *Chinese Remaindering* is typically leveraged to provide this algebraic alternative. The linear system in question, $M\mathbf{x} = \mathbf{b}$, is scaled to achieve integral input matrix M' and integer input vector $\mathbf{b}'$. The MBA algorithm is then performed multiple times, once for each of the prime moduli $p_1, \ldots, p_s$. When all remainders have been computed, the *Chinese Remainder Algorithm* is employed to determine the result $\mathbf{x}'$ modulo $p = p_1 \cdots p_s$. For integer inputs M' and $\mathbf{b}'$, the solution vector $\mathbf{x}'$ will have all rational entries. Conversion from the modular integer output $\mathbf{x}' \bmod p$ to the rational solution $\mathbf{x}'$ is simple. Since the MBA algorithm computes the $detM'$ as a by-product and since $(detM')\mathbf{x}'$ is an integer vector, the reconstruction from $(detM')\mathbf{x}' \bmod p$ requires only a division by $detM'$ for each entry.

5 Lifting Structured Linear Systems

5.1 Hensel's p-adic Lifting

Algebraic (or symbolic) algorithms are a customary tool for the solution of integer linear systems of equations $M\mathbf{x} = \mathbf{b}$. One of the basic techniques is Hensel's p-adic lifting, proposed for solving general linear systems by Moenck and Carter 1979 in [MC79] and Dixon 1982 in [D82] and having a strong similarity to Wilkinson's popular algorithm for numerical iterative refinement in [S80], [GL96, Section 3.5.3], [H02] (see Remark 7.1). Lifting is performed with precision of the order of $\lceil \lg p \rceil$ bits, but the exact rational multiprecision solution can be readily reconstructed from its output.

The basic operation in each recursive step for lifting of a system of linear equations is multiplication of an input matrix M by a vector and multiplication of the precomputed inverse of M modulo a fixed prime p by a vector. This makes lifting attractive for parallel processing and particularly effective for sparse and/or structured linear systems for which these multiplications are fast. Such linear systems can be solved with lifting in time optimal within a polylogarithmic factor (see Section 8). Here the optimality is in terms of the numbers of both Boolean (bit) operations involved as well as arithmetic operations performed with the precision bounded by the length of a computer word (e.g., with the IEEE standard double precision).

Fact 5.1. Hensel's lifting *begins with x_0, the solution x modulo a prime p, and for a fixed h, in h lifting steps computes $\mathbf{x}^{(h)}$, the solution modulo p^h. $\mathbf{x}^{(h)}$ is represented as $\mathbf{x}^{(h)} = x_0 + (x_1)p + (x_2)p^2 + \cdots + (x_{h-1})p^{h-1}$ where the $x_i \in \mathbb{Z}$, $0 \le x_i < p$, p a prime and $h \in \mathbb{Z}$, $h \ge 1$.*

Because Hensel's lifting is an algebraic method its computations are done error free.

Given a prime p, a nonsingular integer matrix M, an integer vector $\mathbf{b}$, and the first term Q in the p-adic expansion of the matrix M^{-1}, the next algorithm recursively computes the p-adic expansion of $\mathbf{x}^{(h)} = M^{-1}\mathbf{b} \bmod p^h$.

Algorithm 5.1. Hensel's lifting *for a linear system [MC79], [D82].*

INPUT: $M \in \mathbb{Z}^{n\times n}$, *an integer* p *relatively prime with* $\det M$, $\mathbf{b} \in \mathbb{Z}^n$, *an integer* $h > 1$, *and* $Q = M^{-1} \bmod p$.

OUTPUT: $\mathbf{x}^{(h)} = M^{-1}\mathbf{b} \bmod p^h$.

INITIALIZATION: $\mathbf{r}^{(0)} = \mathbf{b}$.

COMPUTATIONS:

1. *For* $i = 0, 1, \ldots, h-1$,
   (a) *Compute* $\mathbf{u}^{(i)} = Q\mathbf{r}^{(i)} \bmod p$
   (b) *Compute* $\mathbf{r}^{(i+1)} = (\mathbf{r}^{(i)} - M\mathbf{u}^{(i)})/p$
2. *Compute* $\mathbf{x}^{(h)} = \sum_{i=0}^{h-1} \mathbf{u}^{(i)} p^i$.

Herein *Hensel's lifting* is utilized for solving Toeplitz linear systems $T\mathbf{x} = \mathbf{b}$. From Algorithm 5.1 it is plain that matrix-by-vector multiplication is the dominant operation driving the operation count for solving linear systems with *Hensel's lifting*. For this reason, as with the *Complete Recursive Triangular Factorization* of a matrix, the substitution of the accelerated $O(n \lg n)$ matrix-by-vector multiplication available for Toeplitz matrices T and their inverses T^{-1} in place of general $O(n^2)$ matrix-by-vector multiplication provides for a significant speed up to *Hensel's lifting* when applied to finding the solution to a Toeplitz linear system.

Hensel's lifting has several advantages over the coupled MBA and *Chinese Remainder Algorithm*. Each Hensel's iteration is essentially reduced to two matrix-by-vector multiplications for the two matrices T and $Q = T^{-1} \bmod p$. For Toeplitz

matrices these require $O(n \lg n)$ arithmetic operations. Furthermore, *Hensel's lifting* maintains constant dimension throughout all computations, rather than working with the many various sized auxiliary matrices of the MBA algorithm. This allows for better parallelization based upon well known parallel implementations of the *fast Fourier transform* (FFT). Because T is a Toeplitz matrix and Q the inverse of a Toeplitz matrix, simple Toeplitz-by-vector multiplication for T and use of the Gohberg-Semencul formula for Q will suffice. Further, T need only be nonsingular mod p for *Hensel's lifting* rather than strongly nonsingular mod p as required by the MBA algorithm. *Hensel's lifting* also has the advantage of working modulo a single prime whereas *Chinese remaindering* requires multiple primes. This implies that with *Hensel's lifting* degeneracy only occurs if matrix T is singular mod p for a single prime p, whereas MBA with *Chinese remaindering* suffers degeneracy if matrix T is not strongly nonsingular for any of several primes.

In order to exploit *Hensel's lifting* toward this end, the lifting algorithm must among its inputs include matrix $Q = T^{-1} \bmod p$ for p a prime number. One option that readily provides Q utilizes an algebraic MBA algorithm. Here though *Hensel's lifting* is applied to a single MBA output for a single prime moduli as opposed to applying the *Chinese Remainder Algorithm* to multiple MBA outputs for various prime moduli. Except at the initialization stage, when coupled with MBA for initialization, *Hensel's lifting* retains all the above benefits except that matrix T must now be strongly nonsingular mod p and of course, at the initialization stage only, the MBA algorithm's working space issue arises.

Because of the digital nature of computer hardware, a significant speed up of lifting can be attained when the prime chosen is $p = 2$. This eliminates the need to perform the modulo and division operations of each step and replaces the divisions with right shifts. $x \bmod 2^h$, $h \in \mathbb{Z}_+$, is equivalent to the bit-wise *and* operation

x *and* $(2^h - 1)$. The bit-wise *and* is often utilized in this way as a bit mask in place of *mod* when the modulus is a power of two. No additional effort is required for general computer hardware to carry out integer operations utilizing all bits defining integer values up to word size. Bits outside the mask have no influence on the result. Therefore, it is unnecessary to mask off the extra bits during lifting iterations for a modulus smaller than word size. The masking can instead be done once, after lifting is complete.

Clearly though not all nonsingular matrices are nonsingular mod 2. Obviously half can be expected to fail this test. Since lifting on computer hardware with any choice of prime other than 2 requires the more expensive modulo and division operations in every iteration, it was well worth the effort to find new methods that allow the above mentioned savings while avoiding the above mentioned degeneracy problem associated with $p = 2$.

The obvious direction this implied for the research effort was to begin lifting not from mod p for $p = 2$, but rather mod q for $q = 2^k$, where $k \in \mathbb{Z}$, $k > 1$. One such method, that comes to mind quickly, is to employ MBA mod 2^k for a k less than or equal to the bit size of a machine word. On MBA's output employ lifting. In essence "skipping" $k - 1$ lifting steps.

5.2 Generalized Hensel's Lifting

Herein, the Hensel's lifting algorithm in [MC79], [D82] is generalized to perform in the rings $\mathbb{Z}_{qs}$ for two integers $q > 0$ and $s > 1$. The focus herein will lie mainly on the case where q and s equal powers of two. In this case, *generalized Hensel's lifting* will be refered to as *binary Hensel's lifting*. It is assumed that M is a factor-q nonsingular matrix in $\mathbb{Z}_{qs}^{n\times n}$ (see Definition 2.12) and that the matrix Q satisfies (2.1) is given. It is sufficient if this matrix is given with a black box for its multiplication by a vector or with its generator ($\mathbf{v}$,$\mathbf{p}$) in the case of a Toeplitz

matrix T (see Theorem 4.4). The following algorithm computes the first h terms in the vector expansion $M^{-1}\mathbf{b} = \sum_{i=0}^{\infty} \mathbf{u}^{(i)} s^i$, $\mathbf{u}^{(i)} \in \mathbb{Z}_{qs}^n$, $i = 0, 1, \ldots$.

Algorithm 5.2. Generalized Hensel's Lifting. (See Examples 5.1–5.3 below.)

INPUT: *a matrix* $M \in \mathbb{Z}^{n\times n}$, *a vector* $\mathbf{b} \in \mathbb{Z}^n$, *three positive integers* h, q, *and* s, *and a matrix* $Q \in \mathbb{Z}_{qs}^{n\times n}$ *satisfying (2.1).*

OUTPUT: *the vector* $\mathbf{x}^{(h)} \in \mathbb{Z}^n$ *such that* $M\mathbf{x}^{(h)} = (q\mathbf{b}) \bmod (qs^h)$.

INITIALIZATION: $\mathbf{r}^{(0)} = \mathbf{b}$.

COMPUTATIONS:

1. *For* $i = 0, 1, \ldots, h-1$,
   (a) *Compute* $\mathbf{u}^{(i)} = Q\mathbf{r}^{(i)} \bmod (qs)$
   (b) *Compute* $\mathbf{r}^{(i+1)} = (q\mathbf{r}^{(i)} - M\mathbf{u}^{(i)})/(qs)$
2. *Compute* $\mathbf{x}^{(h)} = \sum_{i=0}^{h-1} \mathbf{u}^{(i)} s^i$.

Example 5.1. $M = \left(\begin{smallmatrix} 2 & 1 \\ 3 & 2 \end{smallmatrix}\right)$, $\mathbf{b} = \binom{3}{4}$. *So* $\mathbf{x} = \binom{2}{-1}$. *By applying Algorithm 5.2 for* $q = 1$, $s = 2$, $\mathbf{r}^{(0)} = \mathbf{b}$, *successively compute* $Q = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\mathbf{u}^{(0)} = \binom{0}{1}$, $\mathbf{r}^{(1)} = \binom{1}{1}$, $\mathbf{u}^{(1)} = \binom{1}{1}$, $\mathbf{r}^{(2)} = \binom{-1}{-2}$, $\mathbf{u}^{(2)} = \binom{0}{1}, \ldots$ *to define* $\mathbf{x}^{(3)} = \mathbf{x} \bmod 8 = \binom{0}{1} + 2\binom{1}{1} + 4\binom{0}{1}$.

Example 5.2. $M = \left(\begin{smallmatrix} 4 & 1 \\ 6 & 2 \end{smallmatrix}\right)$, $\mathbf{b} = \binom{3}{4}$. *So* $\mathbf{x} = \binom{1}{-1}$. *By applying Algorithm 5.2 for* $q = s = 2$, $\mathbf{r}^{(0)} = \mathbf{b}$, *successively compute* $Q = \left(\begin{smallmatrix} 0 & 1 \\ 2 & 0 \end{smallmatrix}\right)$, $\mathbf{u}^{(0)} = \binom{0}{2}$, $\mathbf{r}^{(1)} = \binom{1}{1}$, $\mathbf{u}^{(1)} = \binom{1}{2}$, $\mathbf{r}^{(2)} = \binom{-1}{-2}$, $\mathbf{u}^{(2)} = \binom{2}{2}, \ldots$. *So,* $\mathbf{x}^{(3)} = 2\mathbf{x} \bmod 8 = \binom{0}{2} + 2\binom{1}{2} + 4\binom{2}{2}$, $(M\mathbf{x}^{(h)} - 2\mathbf{b}) \bmod 2^{h+1} = 0$ *for* $h = 1, 2, 3$.

Example 5.3. $M = \left(\begin{smallmatrix} 32 & 2 \\ 48 & 4 \end{smallmatrix}\right)$, $\mathbf{b} = \binom{24}{32}$. *So,* $\mathbf{x} = \binom{1}{-4}$, $s_1(M) = 2$, $s_2(M) = 16$. *Algorithm 5.2 can be applied to* M *and* $\mathbf{b}$ *for* $q = 3$, $s = 2$.

The following theorem shows correctness of the algorithm (see part b) and bounds the precision of its computations. For $q = 1$ and a prime s, Algorithm 5.2 and the theorem have appeared in [D82].

Theorem 5.1. *For* $\mathbf{r}^{(i)}$ *and* $\mathbf{x}^{(h)}$ *in Algorithm 5.2,*

a) $\mathbf{r}^{(i)} \in \mathbb{Z}^n$ *for all* i*;*

b) $M\mathbf{x}^{(h)} = q\mathbf{b} \bmod (qs^h)$*;*

c) all components $r_j^{(i)}$ *of all vectors* $\mathbf{r}^{(i)} = (r_j^{(i)})_j$ *satisfy the bounds*
$|r_j^{(i)}| \le |b_j|/s^i + \alpha n \frac{qs-1}{q} \sum_{k=1}^{i} s^{-k} < \beta/s^i + \alpha n(qs-1)/(qs-q) < \gamma$
where $M = (m_{i,j})_{i,j=1}^n$, $\mathbf{b} = (b_j)_{j=1}^n$,

$$\begin{aligned} \beta = \beta(\mathbf{b}) &= \max_j |b_j|, \\ \alpha = \alpha(M) &= \max_{i,j} |m_{i,j}|, \\ \gamma &= 2\alpha n + \beta. \end{aligned} \tag{5.1}$$

Proof.

a) $(q\mathbf{r}^{(i)} - M\mathbf{u}^{(i)}) \bmod (qs) = (qI - MQ)\mathbf{r}^{(i)} \bmod (qs)$, and the claim follows because $MQ = qI \bmod (qs)$.

b) $M\mathbf{x}^{(h)} = \sum_{i=0}^{h-1} M\mathbf{u}^{(i)} s^i = \sum_{i=0}^{h-1} (q\mathbf{r}^{(i)} - qs\mathbf{r}^{(i+1)})s^i = q\mathbf{b} - qs^h\mathbf{r}^{(h)} = q\mathbf{b} \bmod (qs^h)$.

c) By definition, all components $u_j^{(i)}$ of all vectors $\mathbf{u}^{(i)}$ satisfy $|u_j^{(i)}| \le qs - 1$, and so $qs|r_j^{(i+1)}| \le q|r_j^{(i)}| + \alpha n \max_k |u_k^{(i)}| \le q|r_j^{(i)}| + (qs-1)\alpha n$. The claim now follows by induction on i.

□

Clearly, the arithmetic computational cost of a lifting step is in $m_M + m_Q + O(n)$. Here is an upper bound on the precision of computing.

Lemma 5.1. *Algorithm 5.2 operates with integers in the range* $[-2^{d_1}, 2^{d_1})$ *where*

$$d_1 \le \lceil \lg(2qs\gamma) \rceil \tag{5.2}$$

for γ *in (5.1).*

Proof. The lemma follows from Theorem 5.1 a) and c) since the vectors $\mathbf{u}^{(i)}$ are computed in $\mathbb{Z}_{qs}$. □

The bit precision of computing in Algorithm 5.2 is at most d_1 and is only $\lceil \lg(qs) \rceil$ at the stages of computing the vectors $\mathbf{u}^{(i)}$. Therefore, each lifting step requires $(m_M + O(n))\mu(d_1) + m_Q\mu(\lg(qs))$ bit operations. If λ is the length of a computer word and $d_1 \le \lambda$, then all arithmetic operations in the algorithm are word operations, that is performed within computer precision. To save lifting steps and word operations, *saturated initialization* is used, that is choose q and s so as to maximize the value d_1 subject to the bound $d_1 \le \lambda$.

In Section 6, generalized lifting is initialized, that is, the matrix Q satisfying equation (2.1) is computed. In Section 7, the rational solution $\mathbf{x}$ is reconstructed from the vector $\mathbf{x}^h$.

5.3 Matrix Inversion via Generalized Newton's Lifting

Extending generalized *Hensel's lifting* to matrix inversion actually accelerates it. Recursively compute the matrices

$$X_0 = qM^{-1} \bmod (qs), \quad X_i = X_{i-1}(2qI - MX_{i-1}) \bmod (qs^{2^i}), \tag{5.3}$$

$i = 1, 2, \ldots, h$. Assuming the reduction modulo qs^{2^i}, deduce that

$$qI - MX_i = (qI - MX_{i-1})^2 = (qI - MX_0)^{2^i} = 0,$$

that is, $qI = MX_i \bmod (qs^{2^i})$. For $q = 1$, this is Newton's lifting for matrix inversion [MC79], which has obvious similarity to Newton's iteration for the inversion

of a matrix M numerically [P01, Chapter 6]:

$$X_i = X_{i-1}(2I - MX_{i-1}), \quad i = 1,\ 2,\ \ldots. \tag{5.4}$$

The i-th step of (5.4) squares the residual matrix $I - MX_{i-1}$, thus implying quadratic convergence of the approximations X_i to M^{-1}.

Generalized Newton's lifting for matrix inversion recursively computes the matrices X_0, X_1, ... such that

$$MX_0 = qI \bmod (qs), \quad qX_i = X_{i-1}(2qI - MX_{i-1}) \bmod (qs^{2^i}), \tag{5.5}$$

$i = 1,\ 2,\ \ldots,\ h$. Deduce that $q^{2^i-1}(qI - MX_i) = (q^{2^{i-1}-1}(qI - MX_{i-1}))^2 = (qI - MX_0)^{2^i} = 0 \mod (qs)^{2^i}$ and therefore $qI - MX_i = 0 \bmod (qs^{2^i})$. For $q = 1$, this is Newton's lifting for matrix inversion [MC79], whose ı-th step also squares the residual matrix $I - MX_{i-1}$, thus implying quadratic convergence of the approximations X_i to M^{-1}.

Remark 5.1. *A striking similarity can also be observed between the algebraic Algorithm 5.2 for* Hensel's lifting *and the celebrated algorithm for iterative improvement of the numerical solution of linear systems of equations. (Compare Golub and Van Loan 1996 [GL96, Section 3.5.3], Skeel 1980 [S80], Higham 1996 [H96], and our Algorithm 9.2.) This similarity was exploited in Pan 1992 [P92] and Emiris et al. 1998 [EPY98] to improve the solution algorithm. The improvement relies on performing modular arithmetic with binary rational numbers to avoid computations with the vanishing leading bits of the residuals.*

In h steps, generalized Newton lifting (5.5) achieves as much as generalized Hensel's in 2^h steps, but the precision of computing is roughly doubled in every Newton's step, reaching the level of $(2^h \lg s + \lg q)$-bit precision in h steps.

Every Newton's step (5.5) is essentially reduced to performing $n \times n$ matrix

multiplication twice, which is generally an expensive operation, although allowing a substantial acceleration on multiprocessors.

A boolean time bound nearly as small as that of Hensel's lifting is supported by Newton's p-adic lifting, which has a strong similarity to Newton's iteration for matrix inversion (see, e.g., [P01, chapter 6] and the bibliography therein), but each Newton's lifting step doubles the precision of computing, and substantial additional effort is needed to reverse this expansion to ensure fixed (e.g., double) precision operation.

For a Toeplitz matrix $T = (t_{k-j})_{k,j} = M/q$, the iteration is substantially simplified, because the inverse $X_i = qM^{-1} \bmod (qs^{2^i})$ in $\mathbb{Z}_{qs^{2^i}}$, $i = 0, 1, \ldots$, can be represented with their $n \times 2$ generators $X_i(\mathbf{e}_1, \mathbf{t}) = (X_i\mathbf{e}_1, X_i\mathbf{t})$ where $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$ and the vector $\mathbf{t}$ is defined in Theorem 4.3.

Thus the iteration (5.5) takes the following form,

$$\begin{aligned} X_0(\mathbf{e}_1, \mathbf{t}) &= qM^{-1}(\mathbf{e}_1, \mathbf{t}) \bmod (qs), \\ X_i(\mathbf{e}_1, \mathbf{t}) &= X_{i-1}(2qI - MX_{i-1})(\mathbf{e}_1, \mathbf{t}) \bmod (qs^{2^i}), \end{aligned} \tag{5.6}$$

$i = 1, 2, \ldots$. Its every step is reduced essentially to the multiplication of the matrix T by the $n \times 2$ matrix $X_{i-1}(\mathbf{e}_1, \mathbf{t})$ and of the matrix X_{i-1} by the resulting $n \times 2$ matrix. This is still $O(m(n))$ arithmetic operations (see Theorems 4.2 and 4.3), which is much less than the complexity of $n \times n$ matrix multiplication.

Likewise, if T is a Toeplitz matrix, represent the iterates X_i in (5.4) with their generators and rewrite iteration (5.4) as follows,

$$X_i(\mathbf{e}_1, \mathbf{t}) = X_{i-1}(2I - MX_{i-1})(\mathbf{e}_1, \mathbf{t}). \tag{5.7}$$

For $q = 1$ the iteration processes (5.5) and (5.6) are similar to Newton's iteration for numerical inversion of a matrix M [P01, Chapter 6], which takes the forms

$$X_i = X_{i-1}(2I - MX_{i-1}), \quad i = 1, 2, \ldots, \tag{5.8}$$

for a general matrix M and

$$X_i(\mathbf{e}_1, \mathbf{t}) = X_{i-1}(2I - MX_{i-1})(\mathbf{e}_1, \mathbf{t}), \quad i = 1, 2, \ldots, \tag{5.9}$$

for a Toeplitz matrix M.

In h steps, generalized Newton's lifting computes the solution modulo qs^{2^h}, which generalized Hensel's lifting yields in 2^h steps, but the precision of computing is roughly doubled in every Newton's step and reaches the level of $(2^h \lg s + \lg q)$-bit precision in h steps, so that extended precision is required in quite a small number of Newton's steps. The overall asymptotic bit–operation cost estimate slightly increases versus generalized Hensel's lifting even where $\mu(d) = O((d \lg d) \lg \lg d)$. The Newton's approach, however, enables some savings of word operations in the case where the ratio $\lceil \lg(2qs\gamma) \rceil / \lambda$ is small initially. Indeed, apply generalized Newton's steps as long as the precision of computing stays within a fixed bound w not exceeding the length λ of a computer word. As soon as it is observed that the precision at the next Newton's step would exceeds w, switch to generalized Hensel's lifting, thus yielding its saturated initialization.

5.4 Extention to Toeplitz-like

Hensel's lifting can be easily extended to a Toeplitz/Hankel-like nonsingular input. If the input matrix is nonsingular, the computations do not change except that the matrices are represented by their short generators of length in $O(1)$. If the matrix can be singular, choose the MBA algorithm in [P04] to handle the initialization stage for these singular matrices. The algorithm performed modulo a single moderately large integer. It begins with its multiplicative preconditioning of the input and then inverts a nonsingular submatrix of the largest size. At the MBA stage, many auxiliary matrices are processed and thus there are some additional problems of degeneracy, but otherwise the advantages of computing the rational

solution with lifting versus the MBA algorithm are preserved in this approach. Furthermore, lifting can be initialized with the MBA algorithm. The bit complexity of the initialization is strongly dominated at the subsequent lifting stage. [P04] elaborates upon some technical details of this approach such as solving the singularity problems in $\mathbb{Z}_q$ (via recursive scaling) and the compression of displacement generators in $\mathbb{Z}_q$ with a low bit complexity. The algorithm is immediately extended to solving consistent singular linear systems as well as computing the matrix rank and a vector from (or a basis for) the null space of a Toeplitz/Hankel-like matrix and hence to computing the gcd, the lcm, and the resultant of univariate polynomials with integer coefficients as well as their Padé approximations (cf. [BP94], [P96], [P01]).

The algorithms can be readily extended to various other classes of structured input matrices M. The computational cost estimates are proportional to the arithmetic cost of the multiplication of M and M^{-1} by a vector, which is small also for several other important classes of structured matrices, e.g., of Vandermonde and Cauchy–Pick types. The matrices of Cauchy–Pick type, however, have non-integral (rational) entries, and thus are more likely to degenerate in the reduction modulo a fixed prime power. In this case, it is most promising to begin with their displacement transformation to the Toeplitz/Hankel-like case [P90], [P01]. Then with rounding and scaling we arrive at an integer input.

Hensel's lifting can be replaced by *Newton's* (see subsection 5.3). In this case the number of lifting steps decreases by the factor of $h/\lg h$ for h in Fact 5.1, but a higher precision of computing is needed, in particular the precision is doubled in each *Newton's* lifting step. This can be an advantage for parallel acceleration [P00], and enables rapid initial increase of the precision of the lifting computation until it reaches or nearly reaches the length λ of a computer word (see subsection

5.3).

While this paper addresses application of *Hensel's lifting* to Toeplitz linear systems, extension to Hankel and more generally to Toeplitz-like and Hankel-like linear systems is straightforward. Displacement transformations [P90] immediately allow for extensions to other classes of structured matrices types.

5.5 Extension to the Singular Case and Aplications to Polynomial Computations

For a singular input matrix M of a rank r, the inverse of its nonsingular $r \times r$ submatrix M_r is sought. Algorithm 6.1 can be immediately extended to compute the rank r, submatrix M_r, and the matrix Q_r such that $Q_r M_r \bmod (qs) = qI_r$ for appropriate q and s. Then by applying generalized lifting and the customary techniques in [BP94, page 110], a solution to a consistent linear system $M\mathbf{x} = \mathbf{b}$ and vectors from (as well as the basis for) the null space of the matrix M can be computed. The asymptotic bounds on the overall computational costs do not increase.

The MBA algorithm in $\mathbb{Z}_{p^w}$, can be extended as well provided the input matrix M is a Toeplitz matrix of a rank r and has generic rank profile. The generic rank profile property can be ensured with high probability by shifting to the Toeplitz-like matrix UML where U^T and L are random unit lower triangular Toeplitz matrices (see Kaltofen and Saunders 1991 [KS91]).

The transition from M to UML involves the generation of $2n$ random entries that define the first columns of the matrices U^T and L, but otherwise its computational cost is dominated at the stages of lifting and the reconstruction of the rational solution. The computations at the latter stages are also more expensive than the verification that the $r \times r$ leading principal submatrix is nonsingular. Cost estimates for the rank computation, however, are the Monte Carlo random-

ized bounds (which do not cover the verification that rank $M = r$).

The latter algorithms and complexity estimates can be extended to the problems of computing the gcds, lcms, Padé approximations, and rational interpolation functions where the input is given by univariate polynomials with integer coefficients. On the extension techniques see [BGY80], [BP94], [P96], [P01, Chapters 2 and 3].

6 Initialization of Generalized Lifting

In this section, generalized lifting is initialized assuming that both integers q and s are powers of a fixed prime p. To yield the saturated initialization, the integer $\lg_p(qs)$ is maximized subject to the inequality $d_1 \leq \lambda$ for d_1 in (5.2) and λ denoting the length of a computer word.

Problem 6.1. Initialization of generalized lifting.

INPUT: *a nonsingular matrix $M \in \mathbb{Z}^{n \times n}$, a prime p, and three positive integers λ, the length of a computer word, γ in (5.1), and w such that*

$$\lceil \lg(2p^w \gamma) \rceil = \lambda. \tag{6.1}$$

OUTPUT: *either FAILURE or two integers $q > 0$ and $s > 1$, both powers of p such that $\log_p(qs) = w$, and a matrix Q satisfying (2.1).*

Due to Lemma 5.1, the subsequent lifting steps require a precision within λ. Clearly, with a smaller value of w, some extra lifting steps and word operations would have been needed, whereas with a larger value of w the length of a computer word would have been exceeded and would have required extended precision.

6.1 For General Input Matrices

Gaussian elimination with column pivoting enables computation of the solution to Problem 6.1 for a general matrix M. Block Gaussian elimination would have yielded some implementation benefits (see, e.g., [GL96, Sections 1.3 and 1.4]) but at the price of complicating both pivoting and the exploitation of matrix structure and sparseness.

Algorithm 6.1. Initialization via Gaussian elimination. *For a fixed prime p, compute the integer w in (6.1), and apply Gaussian elimination to invert the matrix*

M. Perform the computations in the ring $\mathbb{Z}_{p^w}$. Apply partial pivoting to avoid divisions by multiples of p, that is at every elimination step interchange the rows to minimize the order of p in the pivot entry (cf. Definition 2.4 on the order of p). If at some step the order exceeds w, output FAILURE and stop. (This occurs where $s_n(M) \mod p^w = 0$ for $s_n(M)$ denoting the Smith leading invariant factor of the matrix M, which is a divisor of $\det M$ [N72] and is typically equal to $\det M$ [ABM99], [EGV00, Corollary 6.2].) Otherwise continue the elimination until the matrix M is diagonalized. Then choose the integer v equal to the maximal order in p among all pivot entries (which are the diagonal entries of the output diagonal matrix). (Under this choice $v = \mathrm{ord}_p(s_n(M))$.) Finally fix the positive integer $u = w - v$ and compute the matrix Q satisfying (2.1) for $q = p^v$ and $s = p^u$.

The algorithm does not fail if and only if

$$w > \mathrm{ord}_p(s_n(M)) \tag{6.2}$$

for w in (6.1). Due to (2.2) it is sufficient if $w > n \log_p |M|$. The reader is refered to [PWa] and the bibliography therein on estimating the failure probability for this computation under a random choice of p or M.

Algorithm 6.1 uses the order of n^3 arithmetic operations performed with the precision of $\lceil \lg(qs) \rceil$. If (6.1) holds, they are word operations. This cost bound is the same as or is dominated at the lifting stage.

6.2 For Toeplitz Input Matrices

In the case of a strongly nonsingular $n \times n$ Toeplitz or Toeplitz-like input matrix M, Algorithm 6.1 is replaced by adapting the MBA divide-and-conquer algorithm by Morf 1974 and 1980 and Bitmead and Anderson 1980 [M74], [M80], and [BA80]. (See Pan 2004 [P04] and the bibliography therein for analysis of this algorithm in $\mathbb{Z}_{p^w}$ and cf. also [P01, Chapter 5].) The adaptation to computing in $\mathbb{Z}_{p^w}$ includes

the low cost deterministic algorithm in [P01, Section 4.6.2] that compresses the dispacement generators in $\mathbb{Z}_p$ wherever they involve extraneous parameters. In the rings $\mathbb{Z}_{p^w}$, pivoting is directed to avoid degeneration in $\mathbb{Z}_{p^w}$, caused by divisions with divisors divisible by p.

It is known that the MBA algorithm works if and only if the input matrix M is strongly nonsingular. In the case under consideration herein, this means strong nonsingularity in the field $\mathbb{Z}_p$. The following are some relevant results (see [P01] and [PWa] on the respective probability estimates).

- A random $n \times n$ integer Toeplitz matrix is likely to be strongly nonsingular modulo any fixed prime $p >> n$.
- If M is not strongly nonsingular in $\mathbb{Z}_p$ for a random prime p sampled from a large range, then M is unlikely to be strongly nonsingular even in $\mathbb{Z}$.
- The matrices $M^T M$ and MM^T are strongly nonsingular in $\mathbb{Z}$ if M is nonsingular in $\mathbb{Z}$ and are likely to remain strongly nonsingular in $\mathbb{Z}_p$ for a larger random prime p. They are Toeplitz-like if the matrix M is as well.

Having the matrices $M^T M$ or MM^T inverted allows

$$M^{-1} = (M^T M)^{-1} M^T = M^T (MM^T)^{-1}.$$

The MBA algorithm uses $O(m(n) \lg n)$ arithmetic operations to compute $O(n)$ parameters that define a displacement generator of its inverse. As a by-product the algorithm also computes $O(n \lg n)$ parameters that define recursive triangular factorization of a strongly nonsingular input matrix, whose determinant is readily available from the factorization.

If a Toeplitz input matrix M is strongly nonsingular in $\mathbb{Z}_p$, lifting can be initialized by applying the Levinson–Durbin algorithm (see Levinson 1947 [L47]

and Durbin 1959 [D59]). It uses the order of n^2 operations in $\mathbb{Z}_p$. This is more operations than required by the MBA algorithm, but still translates into a bit–complexity bound dominated at the lifting stage.

7 Recovery of the Rational Solution

The rational solution of a linear system can recoverd from its value modulo a sufficiently large integer. Basic recovery techniques are known (and are traced back to Pan 1987 and 1988 [P87, Appendix] and [P88]), but some details and specific estimates are provided. The representation of a component of the rational solution requires up to the order of $n \lg \gamma$ bits for γ in equation (5.1). For a larger dimension n this is a multiprecision representation, more naturally treated under the Boolean (bit) model rather than the word model.

7.1 Rational Number Reconstruction

In computer algebra where computations are performed with rational numbers modulo a large integer q (a prime, prime power, or product of several selected primes) the last step is to reconstruct the rational output from its value modulo q [GG99]. Modular rational number reconstruction is the final stage in determining the solution of a nonsingular linear system of n equations by means of *Chinese Remaindering* and p-adic lifting [MC79], [D82], [P02] (see [GG99], [S86], [UP83], [Z93], [HW60] for other important applications).

Modular rational roundoff is the recovery of a rational number x/y from three integers k, l, and $r = (x/y) \bmod l$ provided l and y are relatively prime, x and y are relatively prime unless $x = 0$, $|x| < k \le l$, and $0 < y \le l/k$. $\rho(\lg l)$ denotes the bit–operation complexity of this recovery. Clearly, $x = r$, $y = 1$ if $k > |r|$. The pair (x, y) is unique under the additional assumption that $2|x| < k$ [GG03].

Theorem 7.1.

$$\rho(d) \le cd^2, \ \rho(d) \le C\mu(d) \lg d \tag{7.1}$$

for $\mu(d)$ in (2.3) *and two positive constants C and c, $c < C$.*

Proof. To support the theorem, it is sufficient to apply the algorithms in any of the papers by Pan and Wang 2002 [PW02], 2003 [WP03], 2004 [PW04], or Monagan 2004 [M04]. □

The rational coordinates x/y of the solution to a linear system of equations will be recovered where the pairs of relatively prime integers $|x|$ and y are bounded from above based on Fact 2.1. Further practical gain can rely on heuristics and early termination techniques (see [ABM99] and Dumas et al. 2001 [DSV01]). Choose l and k such that $2|x| < k$ and the solution is unique.

The recovery problem may be solved by applying the Extended Euclidean Algorithm to the input pair $(r_0, r_1) = (l, r)$. Here $|r_1| < |r_0|$, and the algorithm stops for the smallest positive i such that $r_i < k$ in the remainder sequence r_0, r_1, r_2, ... The remainder sequence can be complemented by two dual sequences of *convergents*, denoted s_0, s_1, s_2, ..., and t_0, t_1, t_2, ... in [GG03] (cf. also Schrijver 1986 [S86] and Zippel 1993 [Z93]). With modular rational roundoff, the desired rational solution can be readily obtained from the two triples $(r_{i-1}, s_{i-1}, t_{i-1})$ and (r_i, s_i, t_i), each made up of a remainder and two convergents.

7.2 Deterministic Reconstruction of Rational Components

The unique vector $\mathbf{x} = qM^{-1}\mathbf{b}$ can be recovered from the output vector $\mathbf{x}^{(h)}$ of Algorithm 5.1 if h is sufficiently large. An estimate on how large can be found as follows:

Theorem 7.2. *Let* $\mathbf{x} = qM^{-1}\mathbf{b}$ *denote a unique solution to the linear system* $M\mathbf{x} = q\mathbf{b}$. *Assume* $\rho(d)$ *in (7.1),*

$$d = \lceil \lg(2(\alpha\sqrt{n})^{2n-1}n\beta q)\rceil = O(n\lg\gamma + \lg q), \tag{7.2}$$

and α, β *and* γ *in (5.1). Suppose that in*

$$h = 1 + \lfloor \log_s(2(\alpha\sqrt{n})^{2n-1}n\beta)\rfloor \tag{7.3}$$

steps Algorithm 5.2 computes the vector $\mathbf{x}^{(h)} = \sum_{i=0}^{h-1} \mathbf{u}^{(i)} p^i = \mathbf{x} \bmod (qs^h)$. *Then it is sufficient to perform* B *bit operations for*

$$B = n\rho(d) \tag{7.4}$$

and ρ *defined in Section 7.1 to recover the vector* $\mathbf{x}$ *from the vector* $\mathbf{x}^{(h)}$.

Proof. Suppose that the pairs of coprimes $\nu_j = \nu(x_j)$ and $\delta_j = \delta(x_j)$ define the rational components $x_j = \nu_j/\delta_j$ of the vector $\mathbf{x} = (x_j)_j = qM^{-1}\mathbf{b}$. Fix the smallest integer $k > 2(\alpha\sqrt{n-1})^{n-1} n\beta q$. Note that $s^h > 2(\alpha\sqrt{n})^{2n-1} n\beta$ for h in (7.3). Recall that $M^{-1} \det M = \operatorname{adj} M$ and deduce from Fact 4.1 that $l = qs^h > 2|\nu_j|\delta_j$ and $2|\nu_j| < k \le qs^h$. Then according to Section 7.1 every component x_j can be uniquely recovered from $qx_j \bmod (qs^h)$, and the claimed bit–complexity bound for the recovery follows. □

7.3 Deterministic Recovery with Lifting for a Triangular Factorization

Suppose that lifting of a Toeplitz-like matrix M with the MBA algorithm has been initialized. As a by-product it computes the recursive triangular factorization of $M \mod (qs)$, which immediately defines $(\det M) \mod (qs)$ (cf. [P01, Chapter 5]). By combining the MBA algorithm with recursive application of h lifting steps of Algorithm 5.2 to all auxiliary matrices defining the factorization, this factorization modulo qs^h can be computed and thus obtains $(\det M) \mod (qs^h)$ and the integer vector $\mathbf{y} = (\det M)\mathbf{x} = (\det M)\mathbf{x} \mod (qs^h)$ where $M\mathbf{x} = q\mathbf{b}$ [P00], [P04]. For the integer components y_i of the vector $\mathbf{y}$, $|y_i| \le nq|M|^{n-1}|\mathbf{b}|$. If $qs^h > 2|M|^n$, $qs^h > 2nq|M|^{n-1}|\mathbf{b}|$, then these components and the value $\det M$ can be immediately reconstructed, and the vector $\mathbf{x} = \mathbf{y}/\det M$ can be output.

Since $\det(MM^T) = \det(M^T M) = (\det M)^2$, $\det(UML) = \det M$ for unit triangular matrices U and L, the above technique also covers the case of matrices

M preconditioned according to Section 5.5.

With this technique the multiplications and divisions by the integer $\det M$ are the only operations in the entire computational process where the operands do not generally fit computer precision.

The price for the above simplification of the reconstruction stage is that operations are performed with matrix blocks (rather than just a matrix itself), slightly increasing the complexity of the computations by at most a factor of $\lg n$. These minor complications essentially disappear in the case where similar techniques are applied to general integer linear systems. (In that case lifting is combined with Gaussian elimination using pivoting rather than with the MBA algorithm.)

Remark 7.1. *Assume binary lifting in Algorithm 5.1 for $q = 1$ and for $s = 2^v$ being a power of two and examine the products $\sigma_i(\mathbf{x})$ of $\det M$ with the terms $\mathbf{u}^{(i)}s^i$ of the computed s-adic approximation $\mathbf{x}^{(i)}$ to the solution vector $\mathbf{x}$. These products are the segments in the binary representation of the integer vector $(\det M)\mathbf{x}$. Binary lifting defines them recursively, from right to left. It begins with the trailing segment $\sigma_0(\mathbf{x}) = (\det M)\mathbf{x} \mod s$, obtained by the truncation of the most significant bits up to but not including the v rightmost bits. Numerical iterative refinement defines the same segments from left to right. It begins with the leading segment, obtained by the truncation of the least significant bits and keeping only a certain segment of the leading bits. In both lifting and iterative refinement, one yields the next binary segment of the vector $(\det M)\mathbf{x}$ by reapplying the algorithm to a linear system with the same matrix and properly modified right-hand side.*

7.4 Las Vegas Versus Monte Carlo Randomization

Unlike *deterministic algorithms*, which always produce correct output, *randomized algorithms* produce correct output with a probability of at least $1 - \epsilon$ for a fixed tolerance ϵ. The randomized complexity estimates are of *Las Vegas* type if they

cover the cost of the correctness verification, that is if at the estimated cost one either fails with a low probability or outputs the correct solution. The other randomized complexity estimates are of *Monte Carlo* type. They show the cost bound under which the output can be erroneous, although with a bounded low probability.

7.5 Randomized and Heuristic Recovery of the Rational Solution

By following [P87, Appendix] and [P88] and using Las Vegas randomization, the recovery of the rational solution is accelerated. The bit–complexity bound in (7.4) is decreased by a factor of $\lg d$ provided $\mu(d) = O(d^{\lg 3})$ or $\mu(d) = O((d \lg d) \lg \lg d)$ and $\rho(d)$ is bounded in (7.1), but with heuristic extensions the progress is more significant (cf. the Victor Shoup's recipe in part (b) below).

The acceleration relies on two observations:

(a) The vector $\mathbf{y} = \delta\mathbf{x}$ is filled with integers provided

$$\delta = \mathrm{lcm}_j\, \delta(x_j), \quad 1 \le j \le n \tag{7.5}$$

(for $\delta(y)$ in Definition 2.5), that is δ is the least common multiple of the denominators in all rational coordinates x_j of the solution $\mathbf{x} = (x_j)_j$ to the system $M\mathbf{x} = q\mathbf{b}$. Due to the integrality of the vector $\mathbf{y}$, its recovery from the vector $\mathbf{y} \bmod (qs^h)$ is immediate if $qs^h > 2\delta|\mathbf{x}| = 2|\mathbf{y}|$. Since $\delta \le s_n(M) \le |\det M| \le (\alpha(M)\sqrt{n})^n$ (see Fact 4.1), it is sufficient to use h of (7.3). Multiplication of the vector $\mathbf{x}$ by δ requires the order of $n\mu(d)$ bit operations, thus limiting the theoretical gain versus the estimate $B = n\rho(d)$ in (7.4). The practical gain can be significant, however, because the constant bounding the ratio $\rho(d)/(\mu(d)(\lg d))$ from above can be quite large.

(b) Computation of the value δ can be accelerated with randomization because this value is likely to equal the least common multiple of the denominators in a smaller number K of random linear combinations $\mathbf{c}_k^T\mathbf{x}$ of the coordinates $x_1, \ldots, x_n$, $k = 1, \ldots, K$. Moreover, according to the recipe by Victor Shoup and to empirical evidence, one can typically save further computational work by using the denominators of some selected coordinates themselves, e.g., the first, the second, etc., instead of their random linear combinations.

This approach (originated in [P88]) was recently studied in [ABM99], Cooperman et al. 1999 [CFG99], Eberly et al. 2000 [EGV00], and Mulders and Storjohann 2004 [MS04]. Elaborating upon it:

Algorithm 7.1. *Randomized recovery of the rational solution.*

INPUT: *The same as in Algorithm 5.2 and in addition a positive $\varepsilon < 1$, an integer*

$$h = 1 + \lfloor \log_s(2n(\alpha\sqrt{n})^{2n-1}\eta\beta)\rfloor \tag{7.6}$$

of equation (7.3)*, and the vector* $\mathbf{x}^{(h)} = (x_i^{(h)})_{i=1}^n = qM^{-1}\mathbf{b} \bmod (qs^h)$.

OUTPUT: *FAILURE with a probability of at most ε or a positive integer δ and an integer vector $\mathbf{y}$ such that*

$$M\mathbf{y} = \delta q\mathbf{b}. \tag{7.7}$$

INITIALIZATION: *Compute*

$$K = 2\lceil \lg(1/\varepsilon)\rceil, \tag{7.8}$$

$$\eta = \lceil 6 + 2n\lg(n\alpha)\rceil \tag{7.9}$$

for α and β in (5.1). Then sample K pseudo random vectors

$$\mathbf{c}_k = (c_{jk})_{j=1}^n \in \mathbb{Z}_\eta^n, \quad k = 1, \ldots, K. \tag{7.10}$$

COMPUTATIONS:

1. *Compute the K integers*

$$w_k = \mathbf{c}_k^T \mathbf{x}^{(h)} = \sum_{j=1}^{n} c_{jk} x_j^{(h)}, \quad k = 1, \ldots, K.$$

2. *Recover a unique set of pairs of coprime integers ν_k and δ_k such that*

$$(\nu_k/\delta_k) \bmod (qs^h) = w_k, 1 \le 2\delta_k|\nu_k| \le qs^h,\ 2|\nu_k| < qs^h,\ k = 1, \ldots, K. \tag{7.11}$$

3. *Compute the least common multiple of the denominators*

$$\delta_{lcd} = \operatorname{lcm}_k \delta_k, \quad 1 \le k \le K. \tag{7.12}$$

4. *Compute the integer vector $\mathbf{y} = (y_j)_{j=1}^n$ such that $\mathbf{y} \bmod (qs^h) = \delta_{lcd}\mathbf{x}^{(h)}$ and $2|y_j| < qs^h$ for all j. If $M\mathbf{y} = q\delta_{lcd}\mathbf{b}$, output $\mathbf{y}$ and $\delta = \delta_{lcd}$; otherwise output FAILURE.*

Combining equations (7.3), (7.9), and (7.10) with Fact 4.1 implies (7.11). Correctness of Algorithm 7.1 is implied by the following simple result.

Theorem 7.3. *δ_{lcd} in (7.12) divides δ in (7.5). Furthermore,*

$$\text{Probability}(\delta_{lcd} \ne \delta) \le \varepsilon.$$

Theorem 7.3 is deduced similarly to Theorem 2.1 in [EGV00] based on equations (7.3), (7.8)–(7.12), and the following lemma.

Lemma 7.1. *For a prime p, integers K in (7.8), k such that $1 \le k \le K$, δ in (7.5), η in (7.9), and δ_k in* (7.11), Probability$(\operatorname{ord}_p(\delta_k) < \operatorname{ord}_p(\delta))$ *is equal to $1/\eta$ for $p \ge \eta$ and to $\lfloor \eta/p \rfloor/\eta \le 1/p$ for $p \le \eta$.*

Proof. Let $l = \mathrm{ord}_p(\delta) = \max_j \mathrm{ord}_p(\delta(x_j))$ for $1 \le j \le n$. W.l.o.g., let $l = \mathrm{ord}_p(\delta(x_1))$ and let c denote the first coordinate of the vector $\mathbf{c} = \mathbf{c}_k$. Then

$$\mathbf{c}^T\mathbf{x} = \frac{cu}{ap^l} - \frac{v}{p^h b} = \frac{cub - avp^{l-h}}{abp^l}$$

where $\mathbf{x} = M^{-1}\mathbf{b}$, $l \ge h$, and a, b, u, and v are four integers coprime with p. Clearly, $\mathrm{ord}_p(\delta_k)$ for δ_k in (7.11) never exceeds l; it equals l if and only if $cub - avp^{l-h}$ is coprime with p. Since ub is coprime with p and since c is random in $\mathbb{Z}_\eta$, the probability bound follows. □

To estimate the bit complexity of performing Algorithm 7.1 in terms of $d = O(n \lg \gamma + \lg q)$ in (7.2), $\mu(d)$ in (2.3), $\rho(d)$ in (7.1), and K in (7.8), the following auxiliary result is needed.

Lemma 7.2. *Let j and k be positive integer parameters, $j \to \infty$. Then $O(\mu(j)k)$ bit operations are sufficient to multiply two positive integers u and v such that $u < 2^j$ and $v < 2^{j+k}$.*

Proof. Represent v as $\sum_{i=0}^{k-1} v_i 2^{ij}$, $0 \le v_i < 2^j$ for all i. Compute the products $w_i = uv_i$ for $i = 0, 1, \ldots, k-1$. This takes $O(\mu(j)k)$ bit operations. Now compute the sum $uv = \sum_{i=0}^{k-1} w_i 2^{ij}$. This takes $O(jk)$ bit operations. □

Algorithm 7.1 involves:

- $O(Kn\mu(d))$ bit operations at Stage 1;
- $O(K\rho(d))$ at Stage 2;
- $O(K\mu(d) \lg d)$ at Stage 3, and
- $O(n\mu(d))$, $O(n\mu(\lg \beta)d/\lg \beta)$, and $O(m(n)\mu(\lg \gamma)d/\lg \gamma)$ for computing the vectors $\delta_{lcd}\mathbf{x}^{(h)}$, $q\delta_{lcd}\mathbf{b}$, and $M\mathbf{y}$ at Stage 4, respectively.

(The two latter bounds are deduced based on Lemma 7.2.) Summarizing, the following estimate is obtained.

Theorem 7.4. *Algorithm 7.1 generates nK random elements in $\mathbb{Z}_\eta$ for η in (7.9) and $K = 2\lceil \lg(1/\epsilon) \rceil$ in (7.8). It either fails (this occurs with a probability of at most ϵ) or computes the solution $\mathbf{y}$, δ to linear system (7.7). The algorithm involves*

$$B_1 = O(Kn\mu(d) + K\rho(d) + m(n)\mu(\lg \gamma)d/\lg \gamma)$$

bit operations for $d = O(n \lg \gamma + \lg q)$ in (7.2), $\rho(d)$ in (7.1), γ in (5.1), $m(n)$ in (2.2), and $\mu(d)$ in (2.3); it involves $o(B_1)$ bit operations for generating nK pseudo random elements in $\mathbb{Z}_\eta$.

8 Computational Complexity Estimates

The complexity of the Hensel-based algorithms is surpassed only slightly by the complexity of the respective Newton-based algorithms. The refined bit (Boolean) estimates are presented in a theorem and simplified in a table. It is assumed that the recovery of the rational solution can be obtained by means of the algorithms in Sections 7.2 and 7.5. The resulting estimates can be adjusted assuming the techniques of Section 7.3.

Theorem 8.1. *For a prime p and its powers q and s, let a matrix $M \in \mathbb{Z}^{n\times n}$ be q-factor nonsingular modulo qs. Let $Q \in \mathbb{Z}_s^{n\times n}$ satisfy* (2.1)*. Let $\mathbf{b} \in \mathbb{Z}^n$. Then the rational solution $\mathbf{x}$ to the linear system $M\mathbf{x} = \mathbf{b}$ can be computed by applying the algorithms in the previous sections at a bit–operation cost within the following bounds:*

a) $O(n^3\mu(\lg(qs)))$ *at the initialization stage for a general matrix M;*

b) $O((m(n)\lg n)\mu(\lg(qs)))$ *at the initialization stage for a Toeplitz matrix M;*

c) $(m_Q\mu(\lg(qs)) + (m_M + O(n))\mu(\lg(\gamma qs)))h$ *at the lifting stage;*

d) $n\rho(d) = O(n\mu(d)\lg d)$, $d = O(n\lg\gamma + \lg q)$ *for the deterministic recovery of the n coordinates x_i of the rational solution $\mathbf{x}$ where all x_i are recovered independently of each other;*

e) $O((hm(n)\lg n)\mu(\lg(\gamma qs))) + n\mu(d)$ *for the deterministic recovery of the n coordinates x_i based on lifting the recursive triangular factorization of the (preconditioned) input matrix;*

f) $O(\lg(\frac{1}{\varepsilon})(n\mu(d)+\rho(d)+m(n)\frac{\mu(\lg\gamma)}{\lg\gamma}d)$ *for the Las Vegas recovery which involves $n\lceil\lg\frac{1}{\varepsilon}\rceil\lceil\lg(6+2n\lg(n\alpha))\rceil$ random bits and may fail with a probability of at most $\varepsilon > 0$ but otherwise is certified to be correct.*

Here $m(n)$, $\mu(d)$, and $\rho(d)$ are defined in (2.2), (2.3), *and* (7.1), *γ and α in* (5.1), *$d = O(n \lg \gamma + \lg q)$ in* (7.2), *$h = O(n \lg(\gamma n))$ in* (7.3). *Futhermore, randomized Toeplitz preconditioning $M \rightarrow UML$ in Section 5.5 enables the extension of the above asymptotic bit–operation complexity bounds to the solution of a singular consistent system $M\mathbf{x} = \mathbf{b}$ and to the computation of a nonzero vector $\mathbf{v}$ from the null space $\mathbb{N}(M) = \{\mathbf{v} : M\mathbf{v} = \mathbf{0}\}$ for a singular matrix M. These are Las Vegas bounds, which also cover the certification of the correctness of the solution.*

Remark 8.1. *(Cf. Bini and Pan 1994 [BP94].) The randomized complexity estimates of Theorem 8.1 also apply to computing the rank r of the matrix M and an $r \times r$ nonsingular submatrix of the matrix UML in Section 5.5. In this case, however, they are Monte Carlo estimates. For a Toeplitz input matrix M, the asymptotic Las Vegas estimates for the bit complexity of computing a vector from the null space can be extended to the Monte Carlo estimates for computing a shortest displacement generator of a Toeplitz-like matrix whose columns form a basis for the null space of M.*

Table 8.1 summarizes the bit–complexity estimates in Theorem 8.1. To make the estimates more observable, the notation "$\tilde{O}$" (which means "O" up to the factors in $(\lg \lg n)^{O(1)}$) and the following simplifying assumptions are employed,

$$\log_s \gamma = O(1), \quad \lg(qs) = O(\lg n), \quad \lg(1/\epsilon) = O(\lg n). \tag{8.1}$$

Here ϵ is the error probability in the randomized rational reconstruction of the output.

The bound of $\tilde{O}(n^2 \lg^2 n)$ bit operations on the overall randomized complexity of the initialization, lifting and rational solution reconstruction is record low. Furthermore the bound is nearly optimal assuming a Toeplitz input matrix M and equations (8.1) because $n^2 \lg n$ bits are required to represent the n rational output

Table 8.1: The bit complexity of lifting (for general and Toeplitz input matrices M) and rational reconstruction (deterministic and randomized), under (8.1), (2.2), (2.3), (7.1), and (5.1).

Initialization complexity B_{-1} (for a general matrix M)	$O(n^3\mu(\lg n)) = \tilde{O}(n^3 \lg n)$
Initialization complexity B_{-1} (for a Toeplitz matrix M)	$O((m(n)\lg n)\mu(\lg n)) = \tilde{O}(n \lg^3 n)$
Lifting complexity B_0 (for a general matrix M)	$O(n^3\mu(\lg n)) = \tilde{O}(n^3 \lg n)$
Lifting complexity B_0 (for a Toeplitz matrix M)	$O(nm(n)\mu(\lg n)) = \tilde{O}(n^2 \lg^2 n)$
Reconstruction complexity B_1 (deterministic)	$O(n\rho(n \lg n)) = \tilde{O}(n^2 \lg^3 n)$
Reconstruction complexity B_1 (randomized)	$O(n\mu(n \lg n) + \rho(n)(\lg n)) = \tilde{O}(n^2 \lg^2 n)$
Reconstruction complexity B_1 (with lifting the factorization)	$O(nm(n)(\lg n)\mu(\lg n)) = \tilde{O}(n^2 \lg^3 n)$

values $x_1, \ldots, x_n$, and therefore at least as many bit operations are required to compute these values.

To estimate the number of word operations in the lifting algorithm, including its initialization stage, but not its recovery stage (which involves long integers), assume that $\lg(\gamma qs) = O(\lg \lambda)$, to have a constant bound on the word complexity of an operation in $\mathbb{Z}_t$ where $\lg t = O(\lg(\gamma qs))$. Then the bounds (a)-(c) in Theorem 8.1 turn into the following word–complexity estimates:

a) $\tilde{O}(n^3)$

b) $\tilde{O}(m(n) \lg n)$

c) $\tilde{O}((m_Q + m_M)h)$.

The above estimates show the decrease is only roughly by a factor of λ versus the bit–complexity estimates in Theorem 8.1. Furthermore, the bounds in part

c) decrease as s increases, and when ensuring the saturated initialization, s is maximized.

9 Nearly Optimal Solution of Structured and Sparse Linear Systems

Besides the extension to singular matrices in Section 5.5, the lifting algorithms can be readily extended to any integer or rational input matrix provided it is defined, nonsingular in $\mathbb{Z}_q$, and both it and its precomputed inverse can be multiplied by vectors fast. Toeplitz/Hankel-like matrices and sparse and/or other structured matrices are obvious examples. The same initialization and rational reconstruction algorithms and the study in [PWa] of the degeneration in $\mathbb{Z}_q$ can be applied to these matrices as well.

The extension is quite straightforward for structured matrices defined by their short displacement or semiseparable (rank structured) generators. (See [KKM79], [GO94], [BP94], [KS99], [P01], [EG02], [BGP03/05], and the bibliography therein and in [VVGM05] on these classes of structured matrices.) Below are further comments on the extension to the large and important class of sparse linear systems (e.g., linear systems arising from the discretization of the PDE's), to which the solution algorithms and the complexity estimates are readily extended. Namely, linear systems whose $n \times n$ coefficient matrices M are associated with graphs having an $s(n)$ separator families for $s(n) = O(n^3)$ are discussed. (See the formal definitions in [LRT79], [P93], or [PR93].) In particular $s(n) = O(n^h)$, $h = 1 - \frac{1}{d}$ for a d-dimensional grid graph and $s(n) = 4\lfloor k/2 \rfloor \sqrt{n}$ for an n-vertex finite element graph with at most k boundary vertices per element.

For such matrices recursive block triangular factorization into sparse factors is defined with the techniques of generalized nested dissection in Lipton, Rose and Tarjan 1979 [LRT79], Gilbert and Hafsteinsson 1990 [GH90], Gilbert and Schreiber 1992 [GS92], Pan 1993 [P93], and Pan and Reif 1993 [PR93]. This factorization provides preconditioning for the subsequent rapid multiplication of the matrix M^{-1}

by a vector. Preconditioning enables the initialization modulo a basic integer, and then the matrix and its inverse can be multiplied by a vector using lifting.

The algorithms in the above papers compute the factorization and then multiply the matrix M^{-1} by a vector at the arithmetic cost in $O(M(s(n))) = O(s(n))^3$ and $s(n)^2 + F$, respectively, provided a pair of $n \times n$ matrices can be multiplied in $M(n)$ arithmetic operations and the matrix M has F nonzero entries. This implies the bit–operation cost bounds in $\tilde{O}(M(s(n)) \lg n) = \tilde{O}(s(n)^3 \lg n)$ at the initialization stage and $\tilde{O}((s(n)^2 + F)n \lg n)$ at the lifting stage, whereas the stage of the reconstruction of the rational solution is simplified because the factorization produces the determinant. For the important class of sparse linear systems associated with planar grid graphs, solutions are obtained in $\tilde{O}(n^2 \lg n)$ bit operations, which is a nearly optimal bound.

10 Computing Determinants of General, Structured, and Sparse Matrices

In this section is a review of the computation of the determinants of general, sparse, and structured matrices M and an observation of its other links to lifting besides the application at the reconstructon stage in Section 7.3.

First, note that this computation benefits from *arithmetic filtering*, that is from applying the faster numerical algorithms first and then, in the rare cases where these algorithms fail, backing them up with slower, but more reliable algebraic (or symbolic) algorithms.

Some effective numerical and algebraic (or symbolic) algorithms for determinants rely on the LUP or QR factorization of the input matrix (cf. [C92], [BEPP99], [PY01], and the bibliography therein). The algebraic (or symbolic) algorithms use low precision to compute the determinant modulo sufficiently many smaller primes. Then they reconstruct the integer output by applying the Chinese remainder algorithm. The Wiedemann and block Wiedemann algebraic (or symbolic) algorithms in Wiedemann 1986 [W86] and Coppersmith 1994 [C94] compute $\det M$ modulo such primes as the x-free term of the charcteristic polynomial $\det(M - xI)$ obtained from Krylov sequence.

In [P87, Appendix] and [P88] it was proposed to compute the determinant $\det M$ of an $n \times n$ general integer matrix M by solving the linear system $M\mathbf{x} = \mathbf{b}$ for a random integer vector $\mathbf{b}$ and to employ Hensel's lifting for the solution. This approach was ressurected and advanced in [ABM99] and [EGV00] to support the randomized Monte Carlo computation of the determinant by using $(n^d \lg |M|)^{1+o(1)}$ bit operations for $d = 3$ for the average input matrix M and $d = 7/2$ for the worst case input matrix.

The worst case Monte Carlo exponent $d = 7/2$ was decreased to the Las Vegas

exponents $d = 16/5$, based on an adapted block Wiedemann algorithm, and finally $d = 3$, in Storjohann 2005 [S05], based on the high order Newton's lifting.

The MBA algorithm and lifting have been applied to yield the exponent $d = 16/5$ as well. Kaltofen and Villard 2001 in [KV01] have adapted the block Wiedemann algorithm and ended with the exponent $d = 10/3$ in their main Theorem 2. The final stage of the (block) Wiedemann algorithm is the solution of a (block) Toeplitz/Hankel linear system of equations. By incorporating the MBA algorithm and/or lifting at this stage one can easily decrease the exponent d to 16/5 (see Pan 2002 and 2004 [P02a, Theorem 5.1] and [P04a]). Current progress in lifting enables amelioration of the resulting determinant algorithm although with no affect on the exponent d.

One can decrease all of the above exponents d by incorporating the asymptotically fast algorithms in [CW90] for $n \times n$ matrix multiplication and/or the LKS block half-gcd algorithm (due to Lehmer, Knuth, and Schönhage [B03]), but herein this option is ignored as practically invalid due to the huge overhead constants.

The more customary numerical and algebraic (or symbolic) algorithms based on the LUP or QR factorization of the input matrix only support the exponent $d = 4$, but they are deterministic and for a matrix of a smaller or even moderate size perform faster and can be the practical methods of choice.

For computing the determinants of sparse and/or structured matrices M one can obtain dramatic and practically valid acceleration based on either the MBA or the Wiedemann algorithms, which both output $\det M$ and the vector $M^{-1}\mathbf{b}$. The Wiedemann algorithm supports the exponent $d = \nu + 1$ where the input matrix can be multiplied by a vector in $n^{\nu+o(1)}$ arithmetic operations; in particular $\nu = 1$ for Toeplitz input matrices. The MBA algorithm requires the latter property for the inverses of the input matrix and its leading principal submatrices as well.

The MBA algorithm loses its efficiency where the structure of the input matrix is not readily extended to its inverse, which is the case for the multivariate polynomial resultants. In this case faster algebraic (or symbolic) solution (supporting the record randomized bit–complexity estimates for computing the determinants) is by means of an adapted (block) Wiedemann algorithm (see [EP03/05]).

11 Degeneration in the Rings $\mathbb{Z}_m$

11.1 The Probability of Degeneration in $\mathbb{Z}_{p^v}$ for a Random Prime p

For a fixed nonsingular matrix M, the condition for nondegeneration depends on the prime p. Assume a random prime p, fix its power v, and estimate the probability that p^v divides $\det M$, recalling that $s_n(M)$ is a divisor of $\det M$.

Hereafter $\ln = \log_e$ stands for the natural logarithms (with the base $e = 2.718281\ldots$) and $\pi(y)$ denotes the number of primes not exceeding y.

Lemma 11.1. *(See also (10.4).) If $y > 114$, then $1 < \frac{\pi(y)}{y} \ln y < 1.25$.*

Proof. See Rosser and Schoenfeld 1962 [RS62]. □

Lemma 11.2. *Let $y \geq 114$, then $\pi(y) - \pi(\frac{y}{20}) > (1/\tilde{\beta})\frac{y}{\ln y}$ for*

$$\tilde{\beta} = \frac{1}{1-\tilde{\alpha}} = 1.2049303\ldots, \quad \tilde{\alpha} = \frac{\ln 114}{16 \ln 5.7} = 0.17007650\ldots. \tag{11.1}$$

Proof. By Lemma 11.1, we have $\pi(y) - \pi(\frac{y}{20}) > \frac{y}{\ln y} - \frac{1.25y}{20\ln(y/20)}$. Observe that $\frac{\ln(y/20)}{\ln y}$ is monotone increasing as y grows. So $\frac{1.25}{20\ln(y/20)} \leq \frac{\tilde{\alpha}}{\ln y}$ for $\tilde{\alpha}$ in (11.1) and $y \geq 114$. Combine the above estimates. □

Lemma 11.3. *(Cf. Corollary 7.8.2 in [P01].) Let y, v, h, and k be positive integers such that*

$$y \geq 114, \quad 0 < h^{1/k} \leq y/20. \tag{11.2}$$

Let p be a random prime selected in the range $(y/20, y]$ under the uniform probability distribution. Then Probability$(h \bmod p^v = 0) < \frac{\tilde{\beta} k \ln y}{vy}$ *for $\tilde{\beta}$ in (11.1).*

Proof. Suppose that in the above range there are exactly l distinct primes whose v-th powers divide h. Then the product of these powers also divides h, and therefore we have $h \geq (\frac{y}{20})^{vl}$ because each of the l primes lying in the range $[y/20, y]$ is at

least as large as $\frac{y}{20}$. On the other hand, $h \le (\frac{y}{20})^k$ by assumption. Therefore, $vl \le k$, that is, $l \le k/v$. Compare the latter upper bound on l with the lower bound in Lemma 11.2 on the overall number of primes in the range $(\frac{y}{20}, y]$. □

Theorem 11.1. *(Cf. Corollary 7.8.3 in [P01].) Suppose that ξ is a positive number, v is a positive integer, $M \in \mathbb{Z}^{n\times n}$ is nonsingular, and a prime p is randomly sampled from the range $(y/20, y]$ under the uniform probability distribution in this range where $y = \frac{n\xi \ln|M|}{v\epsilon} \ge 114$ and $\xi = \frac{16\ln 114}{16\ln 5.7 - \ln 114} = 16\tilde{\alpha}\tilde{\beta} = 3.278885\ldots$ for $\tilde{\alpha}$ and $\tilde{\beta}$ in (11.1). Then we have*

$$P = \text{Probability}((\det M) \bmod p^v = 0) < \epsilon. \tag{11.3}$$

Proof. Write $h = |\det M|$, $k = \frac{n\ln|M|}{\ln(y/20)}$, so that $h \le |M|^n$ and $k\ln\frac{y}{20} \ge \ln h$, which implies (11.2). Apply Lemma 11.3 and deduce that

$$P < \frac{\tilde{\beta}k\ln y}{uy} = \frac{\tilde{\beta}n\ln|M|}{v\ln(y/20)}\frac{\ln y}{y} = \frac{v\epsilon\tilde{\beta}n\ln|M|}{vn\ln|M|}\frac{\ln y}{\xi\ln(y/20)} = \frac{\epsilon\tilde{\beta}\ln y}{\delta\ln(y/20)}.$$

Note that

$$\frac{\ln y}{\ln(y/20)} \le \frac{\ln 114}{\ln 5.7}$$

for $y \ge 114$. Therefore

$$P < \epsilon\frac{\tilde{\beta}\ln 114}{\xi\ln 5.7} = \frac{16\tilde{\alpha}\tilde{\beta}}{\xi}\epsilon = \epsilon.$$

□

To extend the above results to smaller y, one may exploit the known extensions of Lemma 11.1, e.g.,

$$1 + \frac{1}{2\ln y} < \pi(y)\frac{\ln y}{y} < 1 + \frac{3}{2\ln y} \tag{11.4}$$

for $y \ge 59$ [GG03, Theorem 18.7]. Refined estimates for $\pi(y)$ can be found in Karatsuba 1990 [K90].

Extend Theorem 11.1 to any integer q instead of $q = p^v$. Relying on the following observation:

Lemma 11.4. *Let p and q be coprime and let u, v, and h be three positive integers. Then $p^u q^v$ divides h if and only if both p^u and q^v divide h.*

Corollary 11.1. *Let $p_1, \ldots, p_h$ be h distinct primes sampled randomly and independently in the ranges $(y_i/20, y_i]$, $i = 1, \ldots, h$, respectively, under the uniform probability distribution. Here $y_i = \frac{\xi n}{2u_i\epsilon} \ln |M| \geq 114$ for ξ in Theorem 11.1 and $i = 1, \ldots, h$; the matrix $M \in \mathbb{Z}^{n\times n}$ is nonsingular; $v_1, \ldots, v_h$ are positive integers, and*

$$2h - 2 \leq \frac{y_i}{\tilde{\beta} \ln y_i} \tag{11.5}$$

for $\tilde{\beta}$ in Lemma 11.2 and for all i. Then

$$P = \text{Probability}(p_1^{v_1} \cdots p_h^{v_h} \text{ divides } \det M) \leq \epsilon^h.$$

Proof. Corollary 11.1 follows from Lemma 11.4 and Theorem 11.1 for $y = y_i$ and $v = 2v_i$. The primes $p_1, \ldots, p_{i-1}$ are excluded from the range $(y_i/20, y_i]$ for every i; this decreases the overall number of primes in this range but less than by twice for $i \leq h$ because of (11.5) and Lemma 11.2. The effect of this decrease on the probability estimates is overweighed by the increase of v from v_i to $2v_i$. □

Remark 11.1. *A random integer matrix M is strongly nonsingular in $\mathbb{R}_q^{n\times n}$ for $q = p^v$ or $q = p_1^{v_1} \cdots p_k^{v_k}$ with a probability which is within the factor of n from the respective bounds in Theorem 11.1 and Corollary 11.1.*

11.2 The Probability of Degeneration for a Fixed Prime p

Suppose that for a fixed basic prime p, a random integer matrix M, and two appropriate integers $q = p^v$ and $s = p^u$, it is wished to estimate the probability that the computations can be performed with a precision within the word length λ.

In the study of computations with general matrices, the following analytic estimate by Brent and McKay 1987 for the proportion of singular matrices in $\mathbb{Z}_{p^u}^{n\times n}$ is useful. (They also supply similar estimates in $\mathbb{Z}_q^{n\times n}$ for any integer $q > 1$.)

Theorem 11.2. *[BMK87, Corollary 2.2]. Write* $P_k(r) = (1-r)(1-r^2)\cdots(1-r^k)$, $r = 1/p$. *Then the nonsingular matrices make up the fraction* $\frac{P_{n+u-1}(r)}{P_{u-1}(r)}$ *of all matrices in* $\mathbb{Z}_{p^u}^{n\times n}$.

Brent and McKay show specific estimates for their ratios as $n \to \infty$ and $q = 1$, ..., 16. Our Table 10.4 in Section 10.4 shows some statistics of nonsingularity of random integer matrices in $\mathbb{Z}_q$, for $n = 5$, 10, 50, 100, $q = 2^g$, and $g = 0$, 1, ..., 20.

They are in reasonable agreement with the analytic estimates in [BMK87].

For Toeplitz versus general matrices M, the known analytic estimates and the results of our experiments in Tables 10.1–10.3 in Section 10.4 show a little higher proportion of nonsingular matrices in $\mathbb{Z}_q^{n\times n}$. In Daykin 1960 [D60] and Kaltofen and Lobo 1996 [KL96] the case of a prime q is studied.

Theorem 11.3. *For any pair of a prime* p *and a positive integer* n, *the fraction of* $1/p$ *of all Toeplitz matrices in* $\mathbb{Z}_p^{n\times n}$ *are singular.*

The following corollary is independently interesting.

Corollary 11.2. *For any pair of a prime* p *and a positive integer* n, *consider the space of the pairs of polynomials* $u(x)$ *and* $v(x)$ *over* $\mathbb{Z}_p$ *such that* $\deg v(x) = n$,

$\deg u(x) < n$. *Then the pairs of coprime polynomials make up a fraction of* $1-1/p$ *in this space.*

Proof. The corollary follows by combining the latter theorem with Proposition 9.1 on page 159 in the book [BP94]. This proposition defines a bijection map of all pairs (h, H) of $h \in \mathbb{Z}_p$ and nonsingular Hankel matrices H in $\mathbb{Z}_p^{n\times n}$ to all pairs of coprime polynomials $u(x)$ and $v(x)$ over $\mathbb{Z}_p$ where $v(x)$ is monic, $\deg v(x) = n$, and $\deg u(x) < n$. Combine the bijection $J : H \leftrightarrow T = HT$ with Theorem 11.3 to count the number of pairs (h, H) where H is nonsingular in $\mathbb{Z}_p^{n\times n}$ and extend this count to the number of pairs of coprime polynomials $u(x)$ and $v(x)$ over $\mathbb{Z}_p$ to obtain the corollary. □

Theorem 11.4. *[KL96, Theorem 5]. For any pair of a prime* p *and a natural* n*, the strongly nonsingular matrices (that is, nonsingular with all their leading principal submatrices) make up a fraction of* $(1-\frac{1}{p})(1-\frac{p-1}{p^2})^{n-1}$ *in the space of all Toeplitz matrices in* $\mathbb{Z}_p^{n\times n}$.

There are no known extensions of the above analytic estimates to the rings $\mathbb{Z}_q$ for any integer $q > 1$. The next results may partly fill this void.

Theorem 11.5. *The fraction of at least* $1 - n/q$ *Toeplitz matrices in* $\mathbb{Z}_q^{n\times n}$ *are nonsingular.*

Proof. There are q^{2n} pairs of univariate polynomials u, v over $\mathbb{Z}_q$ where $\deg u < n$, $\deg v = n$, v is monic. These polynomials are not coprime if and only if their resultant vanishes in $\mathbb{Z}_q$. In virtue of the celebrated lemma in Demillo and Lipton 1978 [DL78] (also in Zippel 1979 [Z79] and Schwartz 1980 [S80]), this occurs for the fraction of at most n/q pairs because the resultant is a polynomial of degree of at most n in the coefficients of u and v. This means at least $(q-n)q^{2n-1}$ pairs of coprime polynomials u and v over $\mathbb{Z}_q$. Due to the bijection in Proposition 9.1 on

page 159 in [BP94], already cited in the proof of Corollary 10.2, we have as many pairs (h, H) in $(\mathbb{Z}_q, \mathbb{Z}_q^{n\times n})$ where H is a nonsingular Hankel matrix. Therefore, there are at least $(q-n)q^{2n-2}$ nonsingular matrices among a total of q^{2n-1} Hankel matrices in $\mathbb{Z}_q^{n\times n}$. The bijection $J : H \leftrightarrow T = HJ$ extends this count to Toeplitz matrices. □

Corollary 11.3. *The fraction of at least* $1 - \frac{(n+1)n}{2q}$ *Toeplitz matrices in* $\mathbb{Z}_q^{n\times n}$ *are strongly nonsingular.*

Proof. There are at most iq^{2n-2} Toeplitz matrices in $\mathbb{Z}_q^{n\times n}$ with singular $i\times i$ leading principal submatrix for $i = 1, \ldots, n$, due to Theorem 11.5. This makes up at most $\sum_{i=1}^{n} iq^{2n-2} = q^{2n-2}n(n+1)/2$ submatrices which are not strongly nonsingular in the set of all q^{2n-1} Toeplitz matrices in $\mathbb{Z}_q^{n\times n}$. □

According to the latter results as well as the results of experimental tests for nonsingularity of random integer Toeplitz and general matrices in $\mathbb{Z}_{q^w}^{n\times n}$ for $q = 2$, $w \le 20$, and $n \le 100$ presented in Section 11.3, the transition to the rings $\mathbb{Z}_{p^w}$ for larger p^w keeps the chances of degeneration quite remote on the average.

11.3 Experimental Computations: How frequently is a random integer Toeplitz matrix non-singular modulo a fixed power of two?

In these tests $n \times n$ integer Toeplitz matrices $M = (t_{i-j})_{i,j}$ were randomly generated. Their entries $t_{1-n}, \ldots, t_{n-1}$ were chosen to be independent of one another under the uniform random distribution on $\mathbb{Z}_q$ for $q = 2^w$ and for a positive integer w. The first column in each of Tables 11.2, 11.4, 11.6 and 11.7 shows how frequently in these tests a random $n \times n$ integer Toeplitz matrix M was nonsingular in $\mathbb{Z}_q$.

Whenever the test showed singularity, it was repeated recursively (up to at most four times), each time adding the outer product of two random vectors to the input matrix. The $(1 + i)$-th column of each table, for $i = 0, 1, 2, 3, 4$, shows for how many out of 100,000 samples the results were positive for the matrices $M - U_j V_j^T$, for some $j \leq i$ where U_j, $V_j^T \in \mathbb{Z}_q^{n \times l}$, $M \in \mathbb{Z}_q^{n \times n}$, $q = 2^w$.

This data motivates using Algorithm 5.2 for smaller i_+ and j_+. They can be compared with similar statistics for general, tridiagonal, and five-diagonal matrices. Table 11.4 shows such statistics, although without small rank perturbations. According to these tests in the case where $q = 2^w$, degeneration is more likely for five-diagonal than general matrices, and is even more likely for tridiagonal matrices, but even for the latter matrices it is quite rare for larger w. It can also be seen that for tridiagonal matrices degeneration is substantially less likely where a shift from $q = 2^w$ to $q = 5^w$ is made.

Table 11.1: Number of times the matrix $M + A_i$ for a random 20×20 Toeplitz matrix M and a random 20×20 matrix A of rank at most i is strongly nonsingular in the ring $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	210	210	210	210	210
2	1165	1335	1418	1480	1528
3	7456	9948	11865	13660	15438
4	22631	32762	40789	47702	53884
5	44823	62750	74087	82002	87421
6	64272	83083	91681	95959	98044
7	79170	93568	97868	99303	99771
8	88441	97860	99522	99907	99976
9	93658	99262	99895	99988	99997
10	96702	99761	99971	100000	100000
11	98312	99922	99994	100000	100000
12	99130	99975	99997	100000	100000
13	99558	99990	100000	100000	100000
14	99759	99997	100000	100000	100000
15	99882	99998	100000	100000	100000
16	99938	99999	100000	100000	100000
17	99977	100000	100000	100000	100000

Table 11.2: Number of times the matrix $M + A_i$ for a random 20×20 Toeplitz matrix M and a random 20×20 matrix A of rank at most i is nonsingular in the ring $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	50173	59450	66672	72514	77452
2	68814	80808	87785	92256	95133
3	82971	92311	96197	98164	99136
4	90559	96899	98862	99567	99852
5	95079	98809	99671	99907	99973
6	97333	99557	99907	99981	99997
7	98643	99859	99973	99998	100000
8	99302	99948	99993	99999	100000
9	99639	99983	100000	100000	100000
10	99816	99997	100000	100000	100000
11	99903	99999	100000	100000	100000
12	99955	100000	100000	100000	100000

Table 11.3: Number of times the matrix $M + A_i$ for a random 50×50 Toeplitz matrix M and a random 50×50 matrix A of rank at most i is strongly nonsingular in the ring $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	1	1	1	1	1
2	4	4	4	4	4
3	186	202	209	214	220
4	2544	3422	4041	4516	5050
5	13528	19755	24892	29560	33979
6	33003	48884	60242	69138	76040
7	55505	75580	86295	92274	95684
8	72981	90415	96482	98685	99532
9	84788	96769	99219	99837	99968
10	91787	98954	99857	99979	99998
11	95671	99700	99971	99997	100000
12	97708	99909	99992	100000	100000
13	98793	99970	99999	100000	100000
14	99413	99992	100000	100000	100000
15	99721	99998	100000	100000	100000
16	99853	100000	100000	100000	100000

Table 11.4: Number of times the matrix $M + A_i$ for a random 50×50 Toeplitz matrix M and a random 50×50 matrix A of rank at most i is nonsingular in the ring $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	50054	59383	66661	72665	77581
2	68781	80792	87812	92341	95151
3	82842	92263	96282	98203	99139
4	90507	96868	98877	99589	99844
5	95132	98846	99695	99915	99976
6	97440	99597	99912	99981	99994
7	98667	99857	99972	99994	99998
8	99315	99953	99989	99997	99999
9	99653	99985	100000	100000	100000
10	99829	99997	100000	100000	100000
11	99917	99999	100000	100000	100000
12	99967	100000	100000	100000	100000

Table 11.5: Number of times the matrix $M + A_i$ for a random 100×100 Toeplitz matrix M and a random 100×100 matrix A of rank at most i is strongly nonsingular in $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	0	0	0	0	0
2	0	0	0	0	0
3	1	1	1	1	1
4	67	70	74	76	78
5	1810	2412	2828	3213	3603
6	10805	16448	21113	25586	29685
7	30625	46536	58189	67402	74571
8	53119	74270	85699	92149	95693
9	71774	90161	96417	98732	99545
10	84225	96749	99266	99839	99954
11	91521	98982	99859	99978	99998
12	95583	99677	99974	99996	100000
13	97764	99904	99995	99999	100000
14	98882	99980	99999	99999	100000
15	99447	99996	100000	100000	100000
16	99721	99998	100000	100000	100000
17	99863	100000	100000	100000	100000

Table 11.6: Number of times the matrix $M + A_i$ for a random 100×100 Toeplitz matrix M and a random 100×100 matrix A of rank at most i is nonsingular in $\mathbb{Z}_q$ for $q = 2^g$ out of 100,000 samples

g \ i	0	1	2	3	4
1	50170	59672	66652	72460	77368
2	68969	80960	87833	92188	95130
3	82799	92261	96240	98128	99122
4	90498	96935	98884	99570	99845
5	94975	98837	99662	99893	99971
6	97255	99547	99898	99970	99991
7	98591	99827	99966	99994	99998
8	99249	99931	99989	99998	99998
9	99616	99976	99997	100000	100000
10	99804	99994	100000	100000	100000
11	99898	99998	100000	100000	100000
12	99948	100000	100000	100000	100000

Table 11.7: Number of times a random $n \times n$ general matrix M is nonsingular in the ring $\mathbb{Z}_q$ out of $100,000$ samples for $q = 2^g$

	$n = 5$	$n = 10$	$n = 50$	$n = 100$
$g = 0$	29,986	28,781	28,940	28,781
$g = 1$	58,637	57,679	57,884	57,782
$g = 2$	77,650	76,817	77,047	77,104
$g = 3$	88,399	87,916	88,000	88,080
$g = 4$	94,102	93,888	93,943	93,921
$g = 5$	97,046	96,911	96,963	96,937
$g = 6$	98,519	98,414	98,483	98,452
$g = 7$	99,245	99,180	99,212	99,235
$g = 8$	99,634	99,598	99,590	99,620
$g = 9$	99,820	99,791	99,783	99,806
$g = 10$	99,911	99,894	99,892	99,899
$g = 11$	99,956	99,957	99,950	99,953
$g = 12$	99,977	99,977	99,978	99,980
$g = 13$	99,985	99,992	99,991	99,992
$g = 14$	99,992	99,996	99,993	99,995
$g = 15$	99,993	99,997	99,996	99,998
$g = 16$	99,995	99,999	99,999	99,998
$g = 17$	99,998	99,999	99,999	99,998
$g = 18$	99,999	100,000	99,999	99,999
$g = 19$	99,999	100,000	100,000	100,000
$g = 20$	99,999	100,000	100,000	100,000

Table 11.8: Number of times a random $n \times n$ general matrix M is strongly non-singular in the ring $\mathbb{Z}_q$ out of $100,000$ samples for $q = 2^g$

	$n = 5$	$n = 10$	$n = 50$	$n = 100$
$g = 0$	3,139	93	0	0
$g = 1$	18,018	3,001	0	0
$g = 2$	41,353	15,653	5	0
$g = 3$	63,254	37,406	566	1
$g = 4$	78,891	59,785	6,440	374
$g = 5$	88,518	76,762	23,843	5,681
$g = 6$	94,042	87,455	47,873	22,908
$g = 7$	96,948	93,386	68,784	46,996
$g = 8$	98,402	96,580	82,731	68,572
$g = 9$	99,164	98,253	90,944	82,738
$g = 10$	99,577	99,121	95,323	90,927
$g = 11$	99,789	99,553	97,669	95,353
$g = 12$	99,886	99,769	98,792	97,582
$g = 13$	99,934	99,877	99,403	98,779
$g = 14$	99,965	99,947	99,707	99,371
$g = 15$	99,978	99,967	99,854	99,662
$g = 16$	99,983	99,984	99,917	98,836
$g = 17$	99,986	99,990	99,963	99,914
$g = 18$	99,987	99,994	99,982	99,947
$g = 19$	99,987	99,994	99,988	99,970
$g = 20$	99,988	99,995	99,992	99,983

Analysis of the Results of the Experiments

For fixed q and n, assume that M is singular over $\mathbb{Z}_q$ with a probability p. Next estimate p. Let x be a random variable such that

$$x = \begin{cases} 1 & \text{if } \det M = 0 \bmod q; \\ 0 & \text{if } \det M \neq 0 \bmod q. \end{cases}$$

Let $x_1, \ldots, x_m$ be the observed values of x. By the Central Limit Theorem,

$$\lim_{m\to\infty} \frac{(x_1 + \ldots + x_m) - mp}{\sqrt{mp(1-p)}} = N(0,1)$$

where $N(0,1)$ is the standard normal probability distribution. Therefore, a confidence interval of probability $1-\alpha$ for p is

$$\left(\bar{x} - Z_{\alpha/2}\sqrt{\bar{x}(1-\bar{x})/m},\ \bar{x} + Z_{\alpha/2}\sqrt{\bar{x}(1-\bar{x})/m}\right)$$

where $\bar{x} = \frac{1}{m}(x_1 + \ldots + x_m)$, Z_α is defined by Probability$(N(0,1) > Z_\alpha) = \alpha$.

Example 11.1. *For $g = 8$, $n = 50$, one can be "99.9%" sure that*

Probability(*Toeplitz matrix M is non-singular*)	$= 0.993 \pm 0.001$;
Probability(*Toeplitz matrix M is strongly non-singular*)	$= 0.731 \pm 0.005$;
Probability(*general matrix M is non-singular*)	$= 0.992 \pm 0.001$;
Probability(*general matrix M is strongly non-singular*)	$= 0.688 \pm 0.005$.

12 Alternative Initializations of Generalized Toeplitz–Hensel's Lifting Modulo a Power of a Prime

In this section two alternative algorithms for the initialization of generalized lifting for factor-q nonsingular Toeplitz linear systems are presented. These algorithms solve a linear system $M\mathbf{x} = q\mathbf{f}$ modulo qs. The integers q and s, both powers of a fixed prime p, are computed in the process of performing the algorithms. The bit complexity of these algorithms is estimated and the algorithms are extended to inverting the matrix Q in (2.3). Finally, these algorithms are compared with the MBA initialization in Section 6.2.

Given a prime p, its power $m = p^b$, a matrix M, and a vector $\mathbf{f} = (f_i)_i$, both of the algorithms first compute the rational vector $M_0^{-1}\mathbf{f}$ for the matrix $M_0 = aM + mI$ and a fixed integer a coprime with m ($a = 1$ in the second algorithm). At the final stage of the algorithms, this is extended to computing the vector $qM^{-1}\mathbf{f} \bmod (qs)$ for appropriate q and s, both powers of p.

12.1 Solving a Linear System with Modular Continuation

Algorithm 12.1. *Initialization of Toeplitz–Hensel's lifting with modular continuation.*

INPUT: *A nonsingular matrix $M \in \mathbb{Z}^{n\times n}$, a vector $\mathbf{f} \in \mathbb{Z}^n$, a prime p, and two integers $m = p^b$ for a positive integer b and $\lambda > 0$, the length of a computer word. (If this length is not bounded, write $\lambda = \infty$.)*

OUTPUT: *FAILURE if $(\det M) \bmod (qs) = 0$ or two positive integers, q and s, both being powers of p such that $qs < 2^\lambda$, and the vector $\mathbf{y} = (qM^{-1}\mathbf{f}) \bmod (qs)$.*

INITIALIZATION: *Choose an integer $a > 1$ coprime with p such that*

$$\gamma^+ \quad = \quad \beta(\mathbf{f}) + 2(m + a\alpha(M))n \quad < \quad 2^\lambda. \tag{12.1}$$

(It is assumed that the values of m and a are sufficiently small to have this bound.)

COMPUTATIONS:

1. *Compute the integer $r = m^{-1} \bmod a$ and the matrix $M_0 = mI + aM$; note that $Q = M_0^{-1} \bmod a = rI$.*

2. *Let $\alpha = \alpha(M_0)$, $\beta = \beta(\mathbf{f})$ and choose h in (7.3) or (7.6) to support deterministic or randomized recovery of the vector $M_0^{-1}\mathbf{f}$ according to Section 7. Specifically, in the deterministic case we write*

$$h = 1 + \lfloor \log_a(2((a|M| + m)\sqrt{n})^{2n-1} n\beta(\mathbf{f}) \rfloor. \tag{12.2}$$

Apply Algorithm 5.2 (for $q = 1$, M replaced by M_0, $\mathbf{b}$ by $\mathbf{f}$, and s by a) to compute the vector $M_0^{-1}\mathbf{f} \bmod a^h$; recover the rational vector $M_0^{-1}\mathbf{f}$.

3. *Compute $d = \max_j \mathrm{ord}_m(\delta((M_0^{-1}\mathbf{f})_j))$. If $2d \le b$, output the integers $q = p^d$ and $s = p^{b-2d} = m/q^2$; compute and output the vector*

$$\mathbf{y} = (aqM_0^{-1}\mathbf{f}) \bmod (qs) = (qM^{-1}\mathbf{f}) \bmod (qs).$$

Otherwise output FAILURE.

Correctness of the algorithm follows because as soon as the equation $q^2 s = m$ is obtained at Stage 3, $M_0/q = (m/q)I + (a/q)M = qsI + (a/q)M = (a/q)M \bmod (qs)$, which implies the desired equations

$$M\mathbf{y} \quad = \quad aqMM_0^{-1}\mathbf{f} \quad = \quad q\mathbf{f} \bmod (qs).$$

The bit operation complexity of performing the algorithm is clearly dominated at its Stage 2. The estimates in Theorem 8.1 can be applied for

$$q = 1, s = a, \gamma = 2\alpha n + \beta, \beta = \beta(\mathbf{f}), \alpha = \alpha(M_0), \tag{12.3}$$

so that $\alpha \le \alpha^+ = m + a\alpha(M)$, $\gamma \le \gamma^+$ for γ^+ in (12.1).

Theorem 12.1. *The bit operation complexity of Algorithm 12.1 applied to a Toeplitz matrix M is bounded according to Theorem 8.1 where q, s, α, β, and γ are defined in (12.3).*

The following properties should guide us in choosing the integers a and b.

(a) As a grows larger, the number of lifting steps decreases at Stage 2 of Algorithm 12.1.

(b) As b grows larger, the number of bit operations increases in Algorithm 12.1.

(c) As a and/or b grows larger, the number of bits of precision of the computations increases at Stage 2, but the bound (12.1) is sufficient to keep the precision below $\lambda + 1$.

(d) (12.1) holds for a positive integer b if

$$b \le b^+ \ = \ \lceil \log_p \Delta \rceil \ , \quad \Delta \ = \ \frac{2^\lambda - 1 - \beta(\mathbf{f})}{2n} - a\alpha(M) \ > \ 1. \tag{12.4}$$

(e) If the integer b^+ is fixed, the word complexity can be minimized by applying Algorithm 12.1 for $b = b^+$. If $b^+ \ge 2d$ for d in Stage 2, the algorithm produces the desired output integers q and s and vector $\mathbf{y}$. Otherwise, the algorithm fails, but the computations can be repeated for distinct a and/or p.

Here are two adverse results of increasing the integer d.

(f) Algorithm 12.1 fails if $2d$ exceeds b^+.

(g) The number h of lifting steps defined in (7.3) and (7.6) for $\alpha = \alpha(M_0)$, $\beta = \beta(\mathbf{f})$ (cf. (12.2)) is roughly proportional to $\log_a \alpha(M_0)$ and $\log_a(a\alpha(M)+m)$. Therefore h is roughly proportional to $d/\log a$ if $m = p^b = p^{2d}$ dominates $a\alpha(M)$.

The following theorem estimates d.

Theorem 12.2. *$d = \max_j \mathrm{ord}_p((\delta(M_0^{-1})_j) = \mathrm{ord}_p(s_n(M_0)) = \mathrm{ord}_p(s_n(M)) \le \mathrm{ord}_p(\det M)$, and so $b \ge 2d$ at Stage 2 of Algorithm 12.1 if $b \ge 2\,\mathrm{ord}_p(\det M)$. The latter bound holds if $b \ge 2\log_p|\det M|$.*

Proof. The theorem is easily deduced from the definitions of M_0, $\delta(x/y)$, and $s_n(M)$ and from the bounds (2.2) since a and p are coprime. □

In addition to (2.2), recall that for a larger random prime p and/or a random integer matrix M, $\mathrm{ord}_p(s_n(M))$ tends to be within a small factor from $\mathrm{ord}_p(\det M)$ (see Theorem 11.1) and therefore within a small factor from $n\log_p(\sqrt{n}\alpha(M))$. Then, by virtue of Theorem 12.2, the integers b and d should be of at most the order of $n\log_p(\sqrt{n}\alpha(M))$. This means that for a moderate bound λ and a larger integer n, Algorithm 12.1 should fail, whereas for a larger λ, that is, for computations with the extended precision, the number h of lifting steps at Stage 2 of this algorithm should grow by roughly the factor of $n/\log a$ versus the estimates in (7.3) and (7.6). Due to this growth caused by the term $mI = p^bI$ in M_0, the arithmetic, word, and bit complexity estimates for the initialization with Algorithm 12.1 should exceed by roughly the factor of n the respective estimates in Theorem 8.1 for the complexity of the subsequent solution of a Toeplitz linear system. Decreasing h is avoided by means of increasing the value $\log a$ because of the high price for increasing $\alpha(M_0)$ and Δ in (12.4).

The above comments apply to the worst case input p and M. For a larger random prime p and/or a random Toeplitz matrix M, however, the chances for the failure of Algorithm 12.1 dramatically decrease, because the integers $\mathrm{ord}_p(\det M)$ and d tend to be in $O(\log_p n)$ according to the estimates in Sections 11.1 and 11.2. In this case $\log(p^b) = O(\log n)$, $\log \alpha(M_0) = O(\log(a\alpha(M) + n))$, and adding the complexity estimates for the initialization with Algorithm 12.1 would not affect the overall asymptotic estimates for solving Toeplitz linear systems.

12.2 Solving a Linear System with Variable Diagonal

In Algorithm 12.1 it is not possible to keep the computation of the vector $\mathbf{x} = M^{-1}\mathbf{f}$ in binary form because a and s are coprime and thus cannot both equal powers of two. The next algorithm does not have this deficiency and still uses about as many lifting steps and bit operations as Algorithm 12.1. The lifting stage of this second algorithm can be performed numerically with bounded precision. Only deterministic recovery is specified at Stage 2, although the recipes of Section 7 for randomized or heuristic acceleration can be immediately extended.

At Stage 2 of this algorithm, *numerical rational roundoff* is applied: a unique rational number x/y is recovered from three integers ν, δ, and k provided $1 \le y \le k$, $|x| < k$, $|x|$ and y are coprime unless $x = 0$; $|x/y - \nu/\delta| < 1/(2k^2)$, and $|\nu| < \delta$. The bound (7.1) for $d = \delta$ is applied to the bit-operation complexity of this recovery as well [WP03].

Algorithm 12.2. *Initialization of Toeplitz–Hensel's lifting using the variable diagonal technique (cf. [P00]).*

INPUT: *as in Algorithm 12.1 and $c > 1$ such that $m \ge c|M|$.*

OUTPUT: *as in Algorithm 12.1.*

INITIALIZATION: *Write* $\mathbf{z}_0 = \mathbf{0}$, $\mathbf{r}_0 = \mathbf{f}$.

COMPUTATIONS: *(cf. Definition 2.10):*

1. *Compute the matrices* $M_0 = M + mI$ *and* $Q = m^{-1}I$.
2. *Recursively compute the vectors* $\mathbf{z}_{i+1} - \mathbf{z}_i = Q\mathbf{r}_i = m^{-1}\mathbf{r}_i$ *and* $\mathbf{r}_{i+1} = \mathbf{f} - M_0\mathbf{z}_{i+1} = \mathbf{r}_i - M_0Q\mathbf{r}_i = -m^{-1}M\mathbf{r}_i$ *for* $i = 0, 1, \ldots, h-1$ *and*

$$h = \lfloor (2n-1)\log_c(|M| + m) + \log_c(2|\mathbf{f}|^2/(c-1)) \rfloor \tag{12.5}$$

(cf. (12.2)).

3. *Recover the vector* $\mathbf{z} = M_0^{-1}\mathbf{f}$ *from* $\mathbf{z}_h$ *deterministically, by using the numerical rational roundoff algorithms.*
4. *Proceed as in Stage 3 of Algorithm 12.1 for* $a = 1$ *and* $\mathbf{y} = \mathbf{z}$.

Stage 2 can be implemented numerically as the customary residual correction algorithm for iterative improvement of the computed approximations to $\mathbf{z}$ where the initial approximation is given by the scaled identity matrix $Q = m^{-1}I$ (see [S80a], [GL96, Section 3.5.3], [H96]). This algorithm is employed in lieu of Hensel's auxiliary lifting.

It has been shown that

$$M_0Q - I = QM_0 - I = m^{-1}M ,$$

$$\mathbf{z} - \mathbf{z}_h = M_0^{-1}(\mathbf{f} - M_0\mathbf{z}_h) = M_0^{-1}\mathbf{r}_h ,$$

$$\mathbf{r}_h = -m^{-1}M\mathbf{r}_{h-1} = (-m^{-1}M)^h\mathbf{r}_0 = (-m^{-1}M)^h\mathbf{f} .$$

Furthermore,

$$|M_0^{-1}| = m^{-1}|(I + M/m)^{-1}| \le m^{-1}\sum_{i=0}^{\infty}(|M|/m)^i,$$

and so

$$|M_0^{-1}| \le \frac{1}{(c-1)m}$$

since $m \geq c|M|$.

Therefore

$$|\mathbf{z} - \mathbf{z}_h| \leq (m^{-1}|M|)^h |\mathbf{f}||M_0^{-1}| \leq (m^{-1}|M|)^h |\mathbf{f}|/((c-1)m) \leq c^{-h}|\mathbf{f}|/((c-1)m) \tag{12.6}$$

for $m \geq 2|M|$.

To ensure correct recovery of the vector $\mathbf{z}$ from $\mathbf{z}_h$ with using the cited numerical rational roundoff algorithms, which extend the algorithms in Section 7.1, it is sufficient to approximate $\mathbf{z}$ by $\mathbf{z}_h$ within the error norm less than $1/(2|M_0|^{2n-1}|\mathbf{f}|)$. This bound is achieved in Algorithm 12.2 due to (12.5)–(12.6) and the inequality $|M_0| \leq |M| + m$.

The analysis in the previous subsection (for $a = 1$) (including Theorems 12.1 and 12.2) is immediately extended. b^+ in (12.6) increases since $a = 1$, and the parameter c (rather than a) plays the role of the lifting and logarithmic base (cf. (12.6)).

12.3 Extension from System Solving to Matrix Inversion and Newton's Acceleration

To initialize lifting, the matrix $Q = (qM^{-1}) \mod (qs)$ is sought. For general matrix M, this requires the solution of n linear systems of equations with the coefficient matrix M. In the Toeplitz case, we only solve the two linear systems $M\mathbf{x} = q_0\mathbf{t} \mod (q_0 s_0)$ and $M\mathbf{y} = q_1\mathbf{e}_1 \mod (q_1 s_1)$ where q_0, q_1, s_0 and s_1 denote the respective values of the integer parameters q and s for these two systems and where the two vectors $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$ and $\mathbf{t}$ define the generator of the matrix M^{-1} (see Theorem 4.3).

The same basic prime p is chosen for both systems and the choice of $q_0 = q_1$

and $s_0 = s_1$ is reconciled by computing

$$q = q_0 = q_1 = p^\sigma, \quad \sigma = \max_j(\mathrm{ord}_p(\delta(M_0^{-1}(\mathbf{t}, \mathbf{e_1}))_j))$$

and $s = s_0 = s_1 = \frac{m}{q}$ at Stage 3, which is common in Algorithm 12.1 (or 12.2) for both linear systems with $q\mathbf{t}$ and $q\mathbf{e}_1$ on the right-hand sides.

If the precision at the lifting steps in Stage 2 in Algorithms 12.1 or 12.2 is substantially less than λ, lifting can be accelerated by applying Newton's steps (5.3) or (5.9), respectively.

12.4 Extension to computations with singular matrices

As a by-product, both Algorithm 12.1 and 12.2 determine whether the matrix M is singular in $\mathbb{Z}$. Therefore combined with binary search, they can replace the MBA algorithm for computing the rank r and $r \times r$ nonsingular submatrix of a preconditioned matrix, UML, having generic rank profile (cf. [KS91] and the end of Section 5.5).

12.5 Comparison with the application of the MBA approach

Recall that Theorem 12.1 covers the bit complexity of performing both Algorithms 12.1 and 12.2 and implies that the estimated overall cost of Toeplitz solving increases versus Theorem 8.1 by a factor ranging from a moderate constant for the random average input matrix M to roughly n in the worst case.

Versus the MBA algorithm, Algorithms 12.1 and 12.2 have the advantage of avoiding divisions, so that they can be performed in the rings for any input matrix M which can be multiplied by a vector fast and do not fail in $\mathbb{Z}_{qs}$ unless the matrix $\frac{M}{q}$ is singular.

Initialization with algorithms of MBA type have lower asymptotic bit complexity than the subsequent stages of Toeplitz solving, but Algorithms 12.1 and

12.2 require simpler code than the MBA algorithm and in particular involve no auxiliary matrices of smaller sizes.

13 Final Remarks

Lifting can be applied to numerical solution of a linear system of equations. The real input values can be rounded to a fixed precision and then scaled to turn them into integers. This stage should be studied further, e.g., would it be effective to reapply and modify rounding if degeneration is encountered?

According to estimates in Section 8, the lifting-based overall solution cost is nearly optimal assuming the exact (long) representation of the rational output. In numerical computations the required precision of computing is defined by the tolerance to output errors and by the conditioning of the problem. For ill conditioned problems the required precision can approach the precision of the exact output, and in this case the algebraic (or symbolic) methods become most competitive.

A loosely related research challenge is the combination of lifting with iterative refinement and, more generally, of modular reduction with rounding, which truncate the binary numbers from two opposite sides (cf. Remark 7.1). Can effective algebraic (or symbolic) counterparts to various iterative algorithms for linear systems of equations be found? (See [P92], [EPY98] on some partial progress in this study.)

The algorithms in the rings $\mathbb{Z}_{qs}$ for $s = 2^u$ and $q = 2^v$ have been experimentally tested. Extensions to this work could include computing polynomial gcd, lcm, etc. for the experimental comparison with the alternative computations in $\mathbb{Z}_p$ for larger random primes p.

Is it reasonable to decrease the asymptotic bit–complexity estimates? Their factor of $m(n)$ comes from the basic operation of Toeplitz matrix-by-vector multiplication or equivalently polynomial multiplication. It is unlikely that any efficient algebraic computation scheme for the tasks could dispense with this operation. This informal argument suggests that improvement of the bit–complexity bounds

by the factor of $m(n)/n$ is unlikely. The basic operation can be viewed as multiplication of polynomials with bounded integer coefficients, and therefore the binary segmentation technique of Fischer and Paterson 1974 [FP74] (cf. [BP94, Section 3.9]) could yield theoretical acceleration by the factor of $(\lg\lg n)\lg\lg\lg n$. The resulting complexity bound in $O(n\mu(n\lg n))$, however, is not practically attractive unless n is huge. Indeed the overhead constant C_{ss} is large, whereas with C_{class} and C_k in (4.1) the overall bit–complexity bounds become as large as n^α for $\alpha > 2.5$.

Finally, lifting is a general technique of algebraic (or symbolic) computations, first proposed for polynomial factorization over the integers (cf. [GG03]). Would generalized and binary lifting be valuable in these applications?

References

[ABM99] J. Abbott, M. Bronstein, T. Mulders. Fast Deterministic Computation of the Determinants of Dense Matrices, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, 197–204, ACM Press, New York, 1999.

[AHU74] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Massachusetts, 1974.

[B85] J. R. Bunch, Stability of Methods for Solving Toeplitz Systems of Equations, *SIAM J. of Scientific and Statistical Computing*, **6(2)**, 349–364, 1985.

[B03] D. J. Bernstein, Fast Multiplication and Its Applications, to be printed in *Algorithmic Number Theory* (edited by Joe Buhler, Peter Stevenhagen), **44**, Mathematical Sciences Research Institute Publications, Cambridge University Press. Available from http://cr.yp.to/papers.html

[BA80] R. R. Bitmead, B. D. O. Anderson, Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations, *Linear Algebra and Its Applications*, **34**, 103–116, 1980.

[BEPP99] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science*, **210(1)**, 173–197, 1999.

[BGP03/05] D. A. Bini, L. Gemignani, V. Y. Pan, Fast and Stable QR Eigenvalue Algorithms for Generalized Companion Matrices and Secular Equation, *Numerische Math.*, **3**, 373–408, 2005. (Also Technical Report 1470, *Department of Math., University of Pisa*, Pisa, Italy, July 2003.)

[BGY80] R. P. Brent, F. G. Gustavson, D. Y. Y. Yun, Fast Solution of Toeplitz Systems of Equations and Computation of Padé Approximations, *J. Algorithms*, **1**, 259–295, 1980.

[BMK87] R. P. Brent, B. D. McKay, Determinants and Ranks of Random Matrices over $\mathbb{Z}_m$, *Discrete Math.*, **66**, 35–49, 1987.

[BP94] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, Volume. 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[C92] K. L. Clarkson, Safe and Effective Determinant Evaluation, *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, 387–395, IEEE Computer Society Press, Los Alamitos, California, 1992.

[C94] D. Coppersmith, Solving Homogeneous Linear Equations over *GF(2)* via Block Wiedemann Algorithm, *Math. of Computation*, **62(205)**, 333–350, 1994.

[CFG99] G. Cooperman, S. Feisel, J. von zur Gathen, G. Havas, GCD of Many Integers, *Computing and Combinatorics, Lecture Notes in Computer Science*, **1627**, 310–317, Springer, Berlin, 1999.

[CK91] D. G. Cantor, E. Kaltofen, On Fast Multiplication of Polynomials over Arbitrary Rings, *Acta Informatica*, **28(7)**, 697–701, 1991.

[CW90] D. Coppersmith, S. Winograd, Matrix Multiplicaton via Arithmetic Progressions, *J. of Symbolic Computation*, **9(3)**, 251–280, 1990.

[D59] J. Durbin, The Fitting of Time-Series Models, *Review of International Statistical Institute,* **28**, 229–249, 1959.

[D60] D. E. Daykin, Distribution of Bordered Persymmetric Matrices in a Finite Field, *J. Reine und Angewandte Math.,* **203**, 47–54, 1960.

[D82] J. D. Dixon, Exact Solution of Linear Equations Using p-adic Expansions, *Numerische Math.*, **40**, 137–141, 1982.

[DGP04] J.-G. Dumas, P. Giorgi, C. Pernet, FFPack: Finite Field Linear Algebra Package, *Proceedings of the International Symposium on Algebraic and Symbolic Computation (ISSAC'04)*, 119–126, ACM Press, New York, 2004.

[DL78] R. A. Demillo, R. J. Lipton, A Probabilistic Remark on Algebraic Program Testing, *Information Processing Letters*, **7(4)**, 193–195, 1978.

[DSV01] J.-G. Dumas, B. D. Saunders, G. Villard, On Efficient Sparse Integer Matrix Smith Form Computations, *J. of Symbolic Computation*, **32**, 71–99, 2001.

[EG02] Y. Eidelman, I. Gohberg, A Modification of the Dewilde–Van der Veen Method for Inversion Finite Structured Matrices, *Linear Algebra and Its Applications*, **343**–**344**, 419–450, 2002.

[EGV00] W. Eberly, M. Giesbrecht, G. Villard, On Computing the Determinant and Smith Form of an Integer Matrix, *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'2000)*, 675–685, IEEE Computer Society Press, Los Alamitos, California, 2000.

[EP03/05] I. Z. Emiris, V. Y. Pan, Improved Algorithms for Computing Determinants and Resultants, *J. of Complexity*, **21(1)**, 43–71, 2005. Proceedings version in *Proceedings of the 6th International Workshop on Computer Algebra in Scientific Computing (CASC '03)*, edited by E. W. Mayr, V. G. Ganzha, and E. V. Vorozhtzov, 81–94, Technische Univ. München, Germany, 2003.

[EPY98] I. Z. Emiris, V. Y. Pan, Y. Yu, Modular Arithmetic for Linear Algebra Computations in the Real Field, *J. of Symbolic Computation*, **26**, 71–87, 1998.

[FP74] M. J. Fischer, M. S. Paterson, String Matching and Other Problems, *SIAM-AMS Proceedings*, **7**, 113–125, 1974.

[GG99] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 1999.

[GG03] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 2003 (second edition).

[GH90] J. R. Gilbert, H. Hafsteinsson, Parallel Symbolic factorization of Sparse Linear Systems, *Parallel Computing*, **14**, 151–162, 1990.

[GL96] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996 (third addition).

[GO94] I. Gohberg, V. Olshevsky, Complexity of Multiplication with Vectors for Structured Matrices, *Linear Algebra and Its Applications*, **202**, 163–192, 1994.

[GS92] J. R. Gilbert, R. Schreiber, Highly Parallel Sparse Cholesky Factorization, *SIAM J. Scientific Computing*, **13**, 1151–1172, 1992.

[H79] G. Heinig, Beitrage zur spektraltheorie von Operatorbuschen und zur algebraischen Theorie von Toeplitzmatrizen, Dissertation **B**, *TH Karl-Marx-Stadt*, 1979.

[H96] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, PA, 1996.

[H02] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, PA, 2002 (second edition).

[HR84] G. Heinig, K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators, Operator Theory*, **13**, Birkhäuser, 1984.

[HW60] G. W. Hardy, E. M. Wright, An Introduction to the Theory of Numbers, 4th edition, Clarendon Press, Oxford, UK, 1960.

[K90] A. A. Karatsuba, The Distribution of Prime Numbers, *Russian Math. Surveys*, **45**, 99–171, 1990.

[K98] D. E. Knuth, *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Massachusetts, 1998.

[K04] I. Kaporin, The Aggregation and Cancellation Techniques As a Practical Tool for Faster Matrix Multiplication, *Theoretical Computer Science*, **315**, 469–510, 2004.

[KKM79] T. Kailath, S. Y. Kung, M. Morf, Displacement Ranks of Matrices and Linear Equations, *Journal of Mathematical Analysis and Applications*, **68(2)**, 395–407, 1979.

[KL96] E. Kaltofen, A. Lobo, On Rank Properties of Toeplitz Matrices over Finite Fields, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'96)*, 241–249, ACM Press, New York, 1996.

[KS91] E. Kaltofen, B. D. Saunders, On Wiedemann's Method for Solving Sparse Linear Systems, *Proceedings of AAECC–5, Lecture Notes in Computer Science*, **536**, 29–38, Springer, Berlin, 1991.

[KS99] T. Kailath, A. H. Sayed (editors), *Fast Reliable Algorithms for Matrices with Structure*, SIAM Publications, Philadelphia, PA, 1999.

[KV01] E. Kaltofen, G. Villard, On the Complexity of Computing Determinants, *Proceedings of the Fifth Asian Symposium on Computer Mathematics (ASCM 2001)*, (Shirayanagi, Kiyoshi and Yokoyama, Kazuhiro, editors), *Lecture Notes Series on Computing*, **9**, 13–27, World Scientific, Singapore, 2001.

[KV04] E. Kaltofen, G. Villard, Computing the Sign or the Value of the Determinant of an Integer Matrix, a Complexity Survey, *J. Computational Applied Math.*, **162(1)**, 133–146, 2004.

[L47] N. Levinson, The Wiener RMS (Root-Mean-Square) Error Criterion in the Filter Design and Prediction, *Journal of Mathematical Physics*, **25**, 261–278, 1947.

[LRT79] R. J. Lipton, D. Rose, R. E. Tarjan, Generalized Nested Dissection, *SIAM J. on Numerical Analysis*, **16(2)**, 346–358, 1979.

[M74] M. Morf, Fast Algorithms for Multivariable Systems, Ph.D. Thesis, *Department of Electrical Engineering, Stanford University*, Stanford, CA, 1974.

[M80] M. Morf, Doubling Algorithms for Toeplitz and Related Equations, *Proceedings of IEEE International Conference on ASSP*, 954–959, IEEE Press, Piscataway, New Jersey, 1980.

[M04] M. Monagan, Maximal Quotient Rational Reconstruction: an Almost Optimal Algorithm for Rational Reconstruction, *Proceedings of the International Symposium on Algebraic and Symbolic Computation (ISSAC'04)*, 243–249, ACM Press, New York, 2004.

[MC79] R. T. Moenck, J. H. Carter, Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equations, *Proceedings of EUROSAM, Lecture Notes in Computer Science*, **72**, 63–73, Springer, Berlin, 1979.

[MS04] T. Mulders, A. Storjohann, Certified Dense Linear System Solving, *J. of Symbolic Computation*, **37(4)**, 485–510, 2004.

[N72] M. Newman, *Integral Matrices*, Academic Press, New York, 1972.

[OP98] V. Olshevsky, V. Y. Pan, A Unified Superfast Algorithm for Boundary Rational Tangential Interpolation Problem and for Inversion and Factorization of Dense Structured Matrices, *Proceedings of the 39th Annual IEEE Symposium on Foundation of Computer Science (FOCS 98)*, 192–201, IEEE Computer Society Press, Los Alamitos, California, 1998.

[P84] V. Y. Pan, How Can I Speed up Matrix Multiplication?, *SIAM Review*, **26(3)**, 393–415, 1984.

[P87] V. Y. Pan, Complexity of Parallel Matrix Computations, *Theoretical Computer Science*, **54**, 65–85, 1987.

[P88] V. Y. Pan, Computing the Determinant and the Characteristic Polynomials of a Matrix via Solving Linear Systems of Equations, *Information Processing Letters*, **28**, 71–75, 1988.

[P90] V. Y. Pan, On Computations with Dense Structured Matrices, *Mathematics of Computation*, **55(191)**, 179–190, 1990.

[P92] V. Y. Pan, Can We Utilize the Cancellation of the Most Significant Digits?, Technical Report TR–92–061, *The International Computer Science Institute*, Berkeley, California, 1992.

[P93] V. Y. Pan, Parallel Solution of Sparse Linear and Path Systems, in *Synthesis of Parallel Algorithms* (J.H. Reif, editor), Chapter 14, 621–678, Morgan Kaufmann publishers, San Mateo, California, 1993.

[P96] V. Y. Pan, Parallel Computation of Polynomial GCD and Some Related Parallel Computations over Abstract Fields, *Theoretical Computer Science*, **162(2)**, 173–223, 1996.

[P00] V. Y. Pan, Parallel Complexity of Computations with General and Toeplitz-like Matrices Filled with Integers and Extensions, *SIAM J. Comput.*, **30(4)**, 1080–1125, 2000.

[P01] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/New York, 2001.

[P02] V. Y. Pan, Can We Optimize Toeplitz/Hankel Computations? *Proceedings of the Fifth International Workshop on Computer Algebra in Scientific Computing (CASC'02)*, Yalta, Crimea, September 2002 (E. W. Mayr, V. G. Ganzha, E. V. Vorozhtzov, Editors), 253–264, *Technische Universität München*, Germany, 2002.

[P02a] V. Y. Pan, Randomized Acceleration of Fundamental Matrix Computations, *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS)*, *Lecture Notes in Computing Science*, **2285**, 215–226, Springer, Heidelberg, Germany, 2002.

[P04] V. Y. Pan, Superfast Algorithms for Singular Integer Toeplitz/ Hankel-like Matrices, Technical Reports 2002 002, 2003 004 and 2004 015, *Ph.D. Program in Computer Science, The Graduate Center of the City University of New York*, New York, 2002/2003/2004.

[P04a] V. Y. Pan, On Theoretical and Practical Acceleration of Randomized Computation of the Determinant of an Integer Matrix, *Zapiski Nauchnykh Seminarov POMI* (in English), **316**, 163–187, St. Petersburg, Russia, 2004.

[PR93] V. Y. Pan, J. Reif, Fast and Efficient Parallel Solution of Sparse Linear Systems, *SIAM J. on Computing*, **22(6)**, 1227–1250, 1993.

[PW02] V. Y. Pan, X. Wang, Acceleration of Euclidean Algorithm and Extensions, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'02)*, (Teo Mora editor), 207–213, ACM Press, New York, 2002.

[PW03] V. Y. Pan, X. Wang, Inversion of Displacement Operators, *SIAM J. on Matrix Analysis and Applications*, **24**, **3**, 660–667, 2003.

[PW04] V. Y. Pan, X. Wang, On Rational Number Reconstruction and Approximation, *SIAM J. on Computing*, **33(2)**, 502–503, 2004.

[PWa] V. Y. Pan, X. Wang, Degeneration of Integer Matrices Modulo an Integer, preprint, 2006.

[PY01] V. Y. Pan, Y. Yu. Certification of Numerical Computation of the Sign of the Determinant of a Matrix, *Algorithmica*, **30**, 708–729, 2001.

[RS62] J. B. Rosser, L. Schoenfeld, Approximate Formulas of Some Functions of Prime Numbers, *Illinois J. of Math.*, **6**, 64–94, 1962.

[S69] V. Strassen, Gaussian Elimination Is Not Optimal, *Numerische Mathematik*, **13**, 354–356, 1969.

[S80] R. D. Skeel, Iterative Refinement Implies Numerical Stability for Gaussian Elimination, *Math. of Computation*, **35**, 817–832, 1980.

[S80a] J. T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *Journal of ACM*, **27(4)**, 701–717, 1980.

[S86] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, New York, 1986.

[S05] A. Storjohann, The Shifted Number System for Fast Linear Algebra on Integer Matrices, *J. of Complexity*, **21(4)**, 609–650, 2005.

[T94] E. E. Tyrtyshnikov, How Bad Are Hankel Matrices? *Numerische Mathematik*, **67(2)**, 261–269, 1994.

[UP83] S. Ursic, C. Patarra, Exact solution of systems of linear equations with iterative methods, *SIAM J. Algebraic Discrete Methods*, **4**, 111-115, 1983.

[VVGM05] R. Vandebril, M. Van Barel, G. Golub, N. Mastronardi, A Bibliography on Semiseparable Matrices, *Calcolo*, **42(3–4)**, 249–270, 2005.

[W86] D. Wiedemann, Solving Sparse Linear Equations over Finite Fields, *IEEE Trans. Inf. Theory*, **IT-32**, 54–62, 1986.

[WP03] X. Wang, V. Y. Pan, Acceleration of Euclidean Algorithm and Rational Number Reconstruction, *SIAM J. on Computing*, **32(2)**, 548-556, 2003.

[Z79] R. E. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proceedings of EUROSAM'79, Lecture Notes in Computer Science*, **72**, 216–226, Springer, Berlin, 1979.

[Z93] R. Zippel, *Effective Polynomial Computation*, Kluwer, Boston, 1993.