COMPUTING THE SMITH FORMS OF INTEGER MATRICES AND SOLVING RELATED PROBLEMS

by

Zhendong Wan

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Summer 2005

COMPUTING THE SMITH FORMS OF INTEGER MATRICES AND SOLVING RELATED PROBLEMS

by

Zhendong Wan

Approved: _____

Henry R. Glyde, Ph.D.
Interim Chair of the Department of Computer and Information Sciences

Approved: _____

Thomas M. Apple, Ph.D.
Dean of the College of Arts and Sciences

Approved: _____

Conrado M. Gempesaw II, Ph.D.
Vice Provost for Academic and International Programs

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

B. David Saunders, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

John Case, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Bob Caviness, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Erich Kaltofen, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Qing Xiang, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to those who have guided, encouraged, and helped me during my Ph.D. study.

First, I would like to thank my Ph.D thesis adviser, Prof. B. David Saunders, for his guiding and encouragement. This thesis could not be done without his guiding and encouragement.

I would like to thank Mrs. Nancy Saunders for her encouragement and hospitality, thank all committee members, Prof. John Case, Prof. Bob Caviness, Prof. Erich Kaltofen, and Prof. Qing Xiang, for their guiding, encouragement, and help.

I would like to thank my friends for all kinds of help. Especially thank all linbox members for their encouragement and help, thank Mr. Leon La Spina and Mr. Danial Rochel for reading my thesis and correcting my writing.

Finally, I would like to thank my wife, Fei Chen (Mrs. Wan), for her patience, encouragement, and help with my English. Also, I would like to thank my parents, brother and sister for their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Smith form of an integer matrix plays an important role in the study of algebraic group theory, homology group theory, systems theory, matrix equivalence, Diophantine systems, and control theory. Asymptotic complexity of the Smith form computation has been steadily improved in the past four decades. A group of algorithms for computing the Smith forms is available now. The best asymptotic algorithm may not always yield the best practical run time. In spite of their different asymptotic complexities, different algorithms are favorable to different matrices in practice. In this thesis, we design an algorithm for the efficient computation of Smith forms of integer matrices. With the computing powers of current computer hardware, it is feasible to compute the Smith forms of integer matrices with dimension in the thousands, even ten thousand.

Our new "engineered" algorithm is designed to attempt to combine the best aspects of previously known algorithms to yield the best practical run time for any given matrix over the integers. The adjective "engineered" is used to suggest that the structure of the algorithm is based on both previous experiments and the asymptotic complexity. In this thesis, we also present lots of improvements for solving related problems such as determining the rank of a matrix, computing the minimal and characteristic polynomials of a matrix, and finding the exact rational solution of a non-singular linear system with integer coefficients.

Chapter 1

INTRODUCTION

Number systems provide us the basic quantitative analysis tool to understand the world and predict the future, and algebra is the abstract study of number systems and operations within them. Computer algebra which designs, analyzes, implements and applies algebraic algorithms has risen as an inter-disciplinary study of algebra and computer science. The power of computers has increased as predicted by Moore's law. Thus symbolic (exact) computation has become more and more important. Symbolic computation, which is often expensive in computation time, delivers the answer without any error, in contrast to numerical linear algebra computation.

In my thesis, I address the efficient computation of the Smith normal form (Smith form, in short) of a matrix over the integers, a specific problem in computer algebra. I will also address some related problems, including computing the matrix rank, exactly solving integer linear systems, and computing the minimal and characteristic polynomials of a matrix. For an $n \times n$ integer matrix, the number of non-zero diagonal entries of the Smith normal form is equal to the rank of the matrix and the Smith form of the matrix can be obtained by computing $\mathrm{O}^{\sim}(\log n)$ characteristic polynomials or by exactly solving $\mathrm{O}^{\sim}(\sqrt{n})$ linear systems (usually with integer coefficients). In each chapter, I discuss each problem in detail.

My main contribution to computer algebra can be summarized as follows:

1. Algorithms with good asymptotic complexity may not be efficient in practice. In order to achieve high performance, it is necessary to optimize them from a practical

point of view. A high performance package needs both good algorithms and good implementations. There are many different Smith form algorithms. If asymptotic complexity is used as the only criteria to compare them, it is easy to sort these algorithms. But in practice, it's hard to tell which is best. Asymptotically best algorithms may not always yield the best practical run time. Each algorithm may yield the fastest run time for certain cases. We have developed an "engineered" algorithm that attempts to utilize the best aspects of the previous known methods. We called it "engineered" because its structure is based on both theoretical and experimental results. In particular, we discovered the importance of separating the rough part and smooth part of invariant factors. Different algorithms are suitable for different parts. Thus we could divide the problem into two sub-problems of computing the rough part and the smooth part. For each subproblem, we used adaptive algorithms to solve it based on thresholds, which are determined by both theoretical and experimental results. We implemented the engineered algorithm in LinBox, which is accessible at `www.linalg.org`. Experiments showed significant success. The engineered algorithm yields the nearly best practical run time for all cases, and guarantees the worst case asymptotic complexity of the Smith normal form algorithm [EGV00]. A preliminary version of the engineered Smith form algorithm was discussed in our paper [SW04]. A more complete version is planned. Details can also be found in chapter 5

2. In [EGV00], the Smith normal form computation of an integer matrix can be reduced to solving $O^{\sim}(\sqrt{n})$ integer linear systems of dimension $n \times n$. For a non-singular $n \times n$ integer matrix A and a randomly chosen integer vector b, the least common multiple of all denominators of the solution of the linear equation $Ax = b$ is very likely to be equal to the largest invariant factor (the largest diagonal entry of the Smith normal form) Using perturbation matrices U and V, where U is an $n \times i$ integer matrix and V is an $i \times n$ integer matrix, the largest invariant factor of

$A + UV$ is very likely to be the ith invariant factor of A (the ith diagonal entry of the Smith form of A). For this perturbation, a number of repetitions are required to achieve a high probability of correctly computing the ith invariant factor. Each distinct invariant factor can be found through a binary search. We have proven that the probability of success in computing the ith invariant factor is better than previously known results, thereby allowing the resulting algorithm to terminate after fewer steps. We also improved the preconditioning process by the following: We replaced the additive preconditioner by a multiplicative preconditioner. Namely, to compute the ith invariant factor we show that the preconditioner $[I_i, U]$ and $[I_i, V]^T$ (U is an $i \times (n-i)$ random integer matrix and V is an $(n-i) \times i$ random integer matrix) is sufficient and is more efficient because the resulting system is $(n-i) \times (n-i)$ rather than $n \times n$. Details can be found both in chapter 5 and in [SW04].

3. Finding the rational solution to a non-singular system with integer coefficients is a classical problem. Many other linear algebra problems can be solved via rational solutions. p-adic lifting [MC79, Dix82] yields the best asymptotic complexity for exactly solving integer linear systems. Numerical linear methods often deliver approximate solutions but much faster than exact linear algebra methods. Many high performance tools have been developed, for example the BLAS (Basic Linear Algebra Subprograms) implementations []. In order to get an approximate solution with high accuracy, numerical methods such as in [GZ02] depend on high precision software floating point operations. that are expensive in both asymptotic and practical run time costs. Using floating point machine arithmetic, BLAS can quickly deliver a solution accurately up to a few bits in a well-conditioned case. In practice, integer matrices are often well-conditioned. It is well known that numeric methods can quickly deliver the solution accurately up to a few bits in well-conditioned cases. In practice, we often meet well-conditioned cases. For a well-conditioned case, we found that by the repeated use of the BLAS to compute low precision approximate

solutions to linear systems that are amplified and adjusted at each iteration, we can quickly compute an approximate solution to the original linear system to any desired accuracy. We show how an exact rational solution can be constructed with a final step after a sufficiently accurate solution has been completed with the BLAS. The algorithm has the same asymptotic complexity as p-adic lifting. Experiment results have shown significant speedups. Details can be found both in chapter 4 and in [Wan04].

4. In linear algebra, many problems such as finding the rational solution of a non-singular integer linear system and computing the characteristic polynomial of an integer matrix are solved via the modular methods. The answer is a vector of integers or a vector of rationals. Working with a single prime via Hensel lifting or with many different primes via the Chinese remainder algorithm is the common way to find the solution modulo a large integer. A priori bounds like Hadamard's bound for the solution of a linear system can be used to determine a termination bound for the Hensel or the Chinese remainder method, that is when the power of the single prime in the Hensel method or the product of the primes in the Chinese remainder method meets or exceeds the termination bound, the exact solution over the integers or rationals, as appropriate, can be constructed. But in practice, the exact solution can often be constructed before the termination bound is met. If the modulus answers are the same for two contiguous iterations of the Hensel or the Chinese remainder method, then it is highly likely that the exact desired result can be constructed at this point thereby yielding an efficient probabilistic algorithm that eliminates the need for additional repetitions to reach the a priori termination bound of the deterministic method. - see e.g [BEPP99, Ebe03]. However, computing all the entries of a solution vector at each step in the Hensel or the Chinese remainder method is quite expensive. We have developed a new method - at each step, only a single number instead of a vector of numbers is reconstructed to decide if more

steps are needed. The number, which is called the ”certificate”, is a random linear combination of the entries of the modular answer. With high probability, when an early termination condition is met, the exact solution can be constructed from the modular answer at this point. Details can be found both in chapter 2 and in [ASW05].

5. The rank of an integer matrix can be found via computation modulo one or more random primes. The computation of the rank of a matrix over a finite field can be done by elimination or by a black box method. With preconditioners as in [EK97, KS91, CEK$^+$02], it most likely has no nilpotent blocks of size greater than 1 in its Jordan canonical form and its characteristic polynomial charpoly(x) is likely to be equal to its minimal polynomial except missing a factor of a power of x. If a matrix has no nilpotent blocks of size greater than one in its Jordan canonical form, then its rank can be obtained from its characteristic polynomial. Since in this case, the multiplicity of the roots in its characteristic polynomial is equal to the dimension of its null space and the rank is equal to the difference of its dimension and the dimension of its null space. For a large sparse system, a sparse elimination method which tries to minimize the fill-in can be very efficient in practice. A version of SuperLU [Sup03] over the finite fields is very efficient in practice - see [DSW03]. Also a blackbox method can be used to efficiently determine the rank. Each method is better for certain cases. Which to use in practice is hard to answer. We found an adaptive algorithm that starts an elimination first, and switches to a blackbox method if it detects a quick rate of fill-in. This adaptive algorithm turns out to be efficient in practice. Experimental results have shown that the run time of the adaptive algorithm is always close to the minimum run time of the two. The approach can be easily adopted to other linear problems such as computing the determinant. Details can be found both in chapter 3 and in [DSW03, LSW03].

6. I have implemented my algorithm in the LinBox library, a C++ template library

for exact, high-performance linear algebra computation. I have improved the LinBox interface design and data structures for sparse matrices, improved the previous implementations, and added many functionalities. All my algorithms I have discussed above are available in LinBox. The beta version is available at `http://www.linalg.org`.

The thesis consists of four parts. Each part addresses a different problem and discusses the history of the problem and my contributions. In each part, I present different methods for the sparse and dense cases. Due to memory efficiency, even run time efficiency, it is often not a good idea to treat large sparse cases just like dense cases. There are many good algorithms for sparse cases of different problems. I start with the problem of computing the characteristic and the minimal polynomials. Two distinct methods for computing a minimal polynomial over a finite field are presented. One is Wiedemann's method [Wie86], which is better for sparse matrices. The other is the Krylov subspace based elimination which is favorable to dense matrices. Wiedemann's method can be adopted to determine the rank of a matrix and the two methods can be adopted to compute the valence, which will be introduced in the chapter about Smith forms. Characteristic polynomials can be computed based on the corresponding minimal polynomials. Computation of minimal and characteristic polynomials over the integers cases are also discussed. Then the matrix rank problem is addressed. For an integer matrix, its rank gives information about the number of non-zeroes invariant factors. The rank of an integer matrix can be found via computation modulo random primes. Over a finite field, (sparse) elimination can be used to determine the rank. For a sparse matrix over a finite field, Wiedemann's method with good preconditioners as in [EK97, KS91, CEK$^+$02] can determine the rank efficiently in practice. Sparse elimination and Wiedemann's method are favored by different cases. Also a practically efficient adaptive algorithm based on the two methods is presented. After that, exact rational solutions of non-singular linear systems are discussed. Exact rational solutions of non-singular linear systems can be

used to determine the largest invariant factors of integer matrices. With perturbation as in [EGV00, SW04], exact rational solutions can also be used determine any indexed invariant factors. p-adic lifting [MC79, Dix82] is a common way to find exact solutions of non-singular linear systems. In the third part, I discuss a new way to find exact rational solutions of non-singular linear systems using numeric linear methods. This new way yields efficient practical run times. Following that, an adaptive algorithm based on p-adic lifting and my new method is presented. Finally, I will discuss the Smith form computation problem. A history of Smith form algorithms, followed by our improvement, is presented. The algorithms discussed in the first three parts, up to this point, are the basis for an "engineered" algorithm for the Smith form.

We use the following definition and notations in the thesis. We say $f(n) = \mathrm{O}^{\sim}(g(n))$, if $f(n) = \mathrm{O}(g(n)^{1+\mathrm{o}(1)})$. Thus $\mathrm{O}^{\sim}()$ can be used to capture these escaped log factors. We use $M(l)$ to denote the number of bit operations required to multiply two integers with bit length at most l. By the Schönhage and Strassen algorithm [vzGG99, Theorem 8.24], $M(l) = O(l \log l \log\log l)$. We use $\log_2(\|A\|)$ to denote the bit length of the entry with largest absolute value of an integer matrix A. We use ω to denote the exponent for matrix multiplication. Using standard matrix multiplication, $\omega = 3$, while the best known Coppersmith and Winograd algorithm [CW87] has $\omega = 2.376$. We use $\mathbb{N}$ to denote the set of the natural numbers, $\mathbb{Z}$ to denote the set of the integers, $\mathbb{Q}$ to denote the set of the rationals, and $\mathbb{R}$ to denote the set of the real numbers. We use $\#S$ to denote the cardinality of the set S.

Chapter 2

COMPUTATION OF MINIMAL POLYNOMIAL AND CHARACTERISTIC POLYNOMIALS

2.1 Introduction

In this chapter, we discuss computation of the minimal polynomial and characteristic polynomial over both finite fields and the integers. Over a finite field, the minimal polynomial can be computed by Krylov subspaces. For dense matrices over a finite field, the Krylov matrix whose columns span a Krylov subspace can be computed. After that a Gaussian elimination can be used to compute the minimal polynomial. For sparse matrices over a finite field, Wiedemann's algorithm can be used. It works with the minimal polynomial of a linearly generated sequence, which can be computed by the Berlekamp-Massey method. The characteristic polynomial of a sparse matrix over a finite field can computed by a blackbox method [Vil00] and the characteristic polynomial of a dense matrix can be computed by an elimination based algorithm such as the Keller-Gehrig algorithm [KG85]. Over the integers, the minimal polynomial and characteristic polynomial can be found by computing the result mod many different primes and using the Chinese remainder algorithm to construct the desired answer. An early termination technique may be used to reduce the practical run time. In practice, computing the minimal polynomial takes less run time than that of the characteristic polynomial. Hensel lifting can be used to compute the characteristic polynomial based on the minimal polynomial. One of our main contributions here is the implementation of these different algorithms for minimal

and characteristic polynomials. These functionalities are now available in the LinBox library. The other is that we discovered a new method that allows the allow efficient binary tree structured Chinese remainder algorithm to be used with an early termination strategy. This new method yields practical run time improvement and even possibly asymptotic improvements.

A portion of the work in this chapter were presented at ISSAC 2003 [PW03] and ISSAC'05 [ASW05].

2.2 Minimal polynomials over a finite field

DEFINITION 2.1. The minimal polynomial of an $n \times n$ matrix A over a field $\mathbb{F}$ is the monic polynomial $p(x)$ over $\mathbb{F}$ of least degree such that $p(A) = 0$.

Any other non-zero polynomial f with $f(A) = 0$ is a multiple of p, the minimal polynomial. We use minpoly(A) to denote the minimal polynomial of a matrix A. In the following, I present two different algorithms for computing the minimal polynomials over a finite field. One is a blackbox method, the Weidman's method. The other is an elimination based method. Both methods are based on the Krylov space.

DEFINITION 2.2. Given an $n \times n$ matrix over a finite field $\mathbb{F}$, and a column vector $v \in \mathbb{F}^n$, the subspace of $\mathbb{F}^n$ spanned by $\{v, Av, \ldots, A^i v, A^{i+1} v, \ldots\}$ is called the Krylov subspace of A and v.

2.2.1 Wiedemann's method for minimal polynomial

In 1986, Wiedemann published a land-mark randomized Las-Vegas algorithm for solving a sparse system over a finite field by working with the minimal polynomial of the matrix. This method is based on Krylov subspaces and can be easily adopted to compute the minimal polynomial of a sparse matrix. This method can compute the minimal polynomial of an n-dimensional blackbox matrix. It requires linear space, quadratic number

of field operations, and matrix-by-vector product no more than $2n$ times. With preconditioning [EK97, CEK$^+$02], Wiedemann's method can be modified to compute the rank of a matrix over a finite field - see algorithm 3.4. Wiedemann's method is based on the minimal polynomial of a linearly generated sequence. Let V be a vector space over a field $\mathbb{F}$, and let $\{a_i\}_{i=0}^{\infty}$ be an infinite sequence with elements $a_i \in V$.

DEFINITION 2.3. The sequence $\{a_i\}_{i=0}^{\infty}$ is linearly generated (recurrent) over $\mathbb{F}$ if there exists $n \in \mathbb{N}$ and $f_0, \cdots, f_n$ with $f_n \neq 0$ such that,

$$\sum_{0 \leq j \leq n} f_j a_{i+j} = f_n a_{i+n} + \ldots + f_0 a_i = 0$$

for all $i \in \mathbb{N}$. The polynomial $f = \sum_{0 \leq j \leq n} f_j x^j \in \mathbb{F}[x]$ of degree n is called a characteristic (or annihilating, or generating) polynomial of $\{a_i\}_{i=0}^{\infty}$.

The set of all characteristic polynomials for $\{a_i\}_{i=0}^{\infty}$ together with the zero polynomial forms a principal ideal in $\mathbb{F}[x]$. The unique monic polynomial generating that ideal is called the *minimal polynomial* of a linearly generated sequence $\{a_i\}_{i=0}^{\infty}$. Every characteristic polynomial is a multiple of the minimal polynomial.

Let A be an $n \times n$ matrix over a field $\mathbb{F}$, then the sequence $\{A^i\}_{i=0}^{\infty}$ is linearly generated, and its minimal polynomial is the minimal polynomial of A, minpoly(A). For any column $v \in \mathbb{F}^n$, the sequence $\{A^i v\}_0^{\infty}$ is also linearly generated. Its minimal polynomial, denoted by minpoly(A, v), must be a divisor of minpoly(A) and can be a proper divisor. For any row vector $u \in \mathbb{F}^{1 \times n}$, the sequence $\{uA^i v\}$ is linearly generated as well, and its minimal polynomial denoted by minpoly$_u(A, v)$ is again a divisor of minpoly(A, v). In [KS91, Lemma 2 and its remark], Kaltofen and Saunders have proven the following two lemmas:

LEMMA 2.4. *Let $A \in \mathbb{F}^{n \times n}, v \in \mathbb{F}^n$, and S be a finite subset of $\mathbb{F}$. If we randomly and uniformly select a row $u \in S^{1 \times n}$, then the probability*

$$\text{Prob}(\text{minpoly}(A, v) = \text{minpoly}_u(A, v)) \geq 1 - \frac{\deg(\text{minpoly}(A, v))}{\#S},$$

where $\#S$ means the cardinality of the set S.

LEMMA 2.5. *Let $A \in \mathbb{F}^{n\times n}$, and S be a finite subset of $\mathbb{F}$. If we randomly and uniformly select a column $v \in S^n$, then the probability*

$$\text{Prob}(\text{minpoly}(A) = \text{minpoly}(A, v)) \geq 1 - \frac{\deg(\text{minpoly}(A))}{\#S}.$$

Thus, the minimal polynomial of A can be computed by picking a random row vector u and a random column vector v. That is accomplished by first constructing the sequence $\{uA^iv\}_0^\infty$ and then finding its minimal polynomial. Then, the minimal polynomial of the sequence $\{uA^iv\}_0^\infty$ can be computed by the Berlekamp-Massey method [Mas69] in $\text{O}(m \deg(\text{minpoly}_u(A, v)))$ field operations.

ALGORITHM 2.6. *Wiedemann's algorithm for minimal polynomial*
Input:

- *A, an $n \times n$ matrix over a finite field $\mathbb{F}$.*

Output:

- *The minimal polynomial of A.*

Procedure:

Step 1: Pick a random row vector u and a random column vector v.

Step 2: Compute the sequence

$$a_0 = uv, a_1 = uAv, \cdots, a_i = uA^iv, a_{i+1} = uA^{i+1}v, a_{2n-1} = uA^{2n-1}v.$$

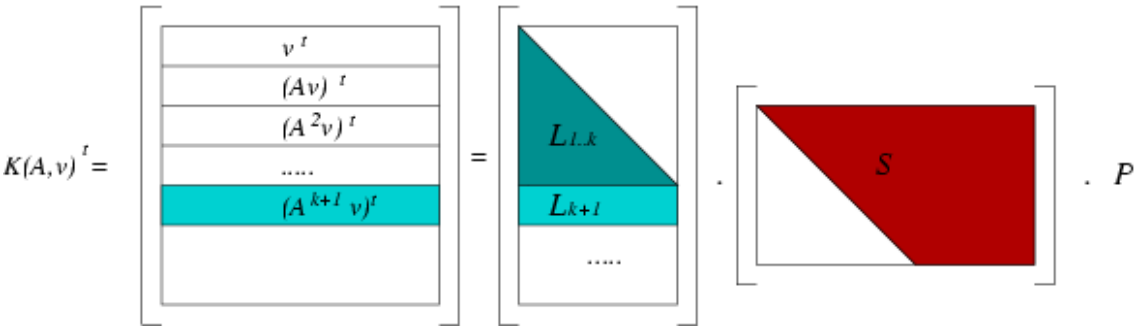
Step 3: Compute $\text{minpoly}_u(A, v)$ *by the Berlekamp-Massey algorithm [Mas69] and return it.*

The algorithm above will compute the minimal polynomial of A with probability at least $1 - \frac{2n}{\#\mathbb{F}}$. It requires $\text{O}(n^2)$ field operations and $\text{O}(n)$ matrix-by-vector product of A. An early termination technique [Ebe03] can be used here to reduce practical run time.

2.2.2 An elimination based minimal polynomial algorithm

Given a matrix A, if we randomly choose a vector V, then the minimal polynomial of the sequence $\{v, Av, A^2, \ldots, A^n v, A^{n+1}v, \ldots\}$ is likely to be equal to the minimal polynomial of A by lemma 2.4. The minimal polynomial of the sequence $\{v, Av, A^2, \ldots, A^n v, A^{n+1}v, \ldots\}$ can be computed via elimination on the corresponding $n \times n$ Krylov matrix $K(A, v)$, whose ith column is the vector $A^i v$. More precisely, one computes the LSP factorization of $K(A, v)^T$ - see [IMH82] for a description of the LSP factorization. Let k be the degree of minpoly(A, v). Then the first k rows of $K(A, v)^T$ are linearly independent, and the remaining $(n - k)$ columns are a linear combination of the first k ones. Therefore in its LSP factorization, S is triangular with its last $(n - k)$ rows being equal to 0. The LSP factorization of $K(A, v)^T$ can be viewed as in figure 2.1.

Figure 2.1: The principle of computation of the minimal polynomial



Now the trick is to notice that the vector $m = L_{k+1}L_{1\ldots k}^{-1}$ gives the negations of the coefficients of minpoly(A, v). Indeed, if we define $X = K_{A,v}^t$

$$X_{1\ldots n,k+1} = (A^k v)^t = \sum_{i=0}^{k-1} m_i (A^i v)^t = m \cdot X_{1\ldots n,1\ldots k},$$

then

$$\text{minpoly}(A, v)(X) = X^k - m_k X^{k-1} - \cdots - m_1 X - m_0.$$

The algorithm itself is straightforward:

ALGORITHM 2.7. *Elimination-based* minpoly(A, v)

Input:

- A*, an* $n \times n$ *matrix over a field* $\mathbb{F}$
- $v \in \mathbb{F}^n$*, a column vector.*

Output:

- minpoly(A, v)*, the minimal polynomial of the sequence* $\{v, Av, A^2, \ldots, A^n v, A^{n+1} v, \ldots\}$

Procedure:

1. $K_{1\ldots n,1} = v$
2. *for* $i = 1$ *to* $\log_2 n$[Compute $K(A, v)$]
   $K_{1\ldots n,2^i\ldots 2^{i+1}-1} = A^{2^{i-1}} K_{1\ldots n,1\ldots 2^i-1}$
3. $(L, S, P) =$ LSP$(K^t), k =$ rank(K)
4. $m = L_{k+1}.L^{-1}_{1\ldots k}$
5. return minpoly$(A, v)(X) = X^k + \sum_{i=0}^{k-1} m_i X^i$

The dominant operation in this algorithm is the computation of the Krylov matrix, in $\log_2 n$ matrix multiplications, i.e. in O$(n^\omega \log n)$ algebraic operations. The LSP factorization requires O(n^ω) operations and the triangular system resolution, O(n^2). Thus the complexity of this algorithm is O$(n^\omega \log n)$. When using classical matrix multiplications (assuming $\omega = 3$), it is preferable to compute the Krylov matrix by n successive matrix vector products. Then the number of field operations is O(n^3).

2.3 Characteristic polynomials over a finite field

DEFINITION 2.8. For an $n \times n$ matrix A over a field $\mathbb{F}$, the characteristic polynomial of A is the polynomial $p(x)$ defined by

$$p(x) = \text{det}(xI_n - A).$$

We use charpoly(A) to denote the characteristic polynomial of a matrix A. We will present a blackbox method and two elimination based methods for characteristic polynomial over a finite field below.

2.3.1 A blackbox method

We use charpoly(A) to denote the characteristic polynomial of a matrix A.

DEFINITION 2.9. Given a monic polynomial $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ the companion matrix of $p(x)$, denoted $\mathcal{C}_{p(x)}$, is defined to be the $n \times n$ matrix with 1's down the the first sub-diagonal and minus the coefficients of $p(x)$ down the last column, or alternatively, as the transpose of this matrix.

$$\mathcal{C}_{p(x)} = \begin{pmatrix} 0 & 0 & \dots & \dots & \dots & -a_0 \\ 1 & 0 & \dots & \dots & \dots & -a_1 \\ 0 & 1 & \dots & \dots & \dots & -a_2 \\ 0 & 0 & \ddots & & & -a_3 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \dots & 1 & -a_{n-1} \end{pmatrix}.$$

For a monic polynomial of $p(x)$, the characteristic polynomial and the minimal polynomial of the companion matrix $\mathcal{C}_{p(x)}$ is equal to $p(x)$.

A matrix over a field can be decomposed to an equivalent diagonal block matrix, whose diagonal entries are companion matrices.

DEFINITION 2.10. If M is an $n \times n$ matrix over a field $\mathbb{F}$, there is an invertible matrix P such that $P^{-1}MP = F$ where

$$F = M_1 \oplus M_2 \oplus \cdots \oplus M_t$$

where the Ms are the companion matrices of what are known as the invariant polynomials of M. If M_i is the companion matrix of a polynomial $f_i(x)$, and the characteristic polynomial of M is $f_i(x)$, then

$$p(x) = f_1(x)f_2(x)\cdots f_t(x).$$

Each f_i divides f_{i+1}, if one writes the companion matrices M_i such that as i increases, so does the number of rows/columns of M_i. The polynomial f_t then is the minimal polynomial of M.

For these companion matrices in the Frobenius normal form of A, their characteristic polynomials are also invariant factors of $xI_n - A$ over $\mathbb{F}[x]$. And the minimal polynomial is the largest invariant factor of $xI_n - A$. All the invariant factors can be computed by binary search - see [Vil00] for details.

2.3.2 Two elimination based methods

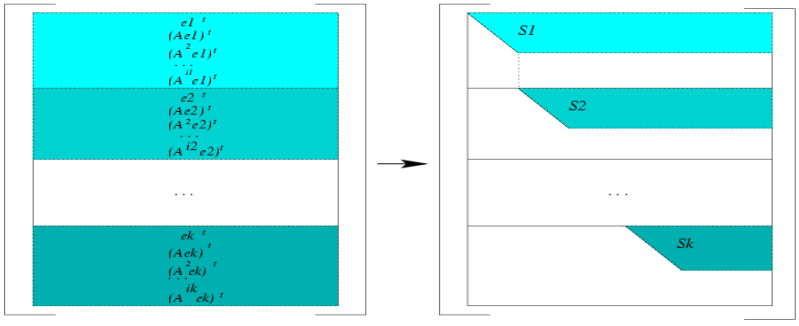
In [KG85], Keller-Gehrig gives an $O(n^\omega \log(n))$ algorithm for characteristic polynomial over a finite field $\mathbb{F}$. It works as follows:

- Find e_{i_j}, $1 \leq j \leq k$, such that $\mathbb{F}^n$ is equal to the direct sum of all the Krylov Spaces of A and e_{i_j}. This involves a complicated step-form elimination routine designed for this application.
- Construct $U = \{e_{i_1}, Ae_{i_1}, \ldots, A^{k_1}e_{i_1}, e_{i_2}, Ae_{i_2}, \ldots\}$ a basis of $\mathbb{F}^n$.
- $U^{-1}AU$ is block upper triangular, and the diagonal blocks are companion matrices. Then, characteristic polynomial of A is equal to the product the characteristic polynomial of these companion blocks.

In implementation, we replace the step-form elimination by a simple block elimination LSP. Also algorithm 2.7 can be adjusted to get each factor. Moreover, the LSP factorization of each $K(A, e_i)$ is already available, since it has been computed to determine the

linear dependencies with LSP. Therefore, the expensive part of minpoly(A, e_i) can be avoided. We can replace the complex step form elimination of Keller-Gehrig by the simpler LSP factorization. Then, we have a straightforward method for finding the factors: apply only the third step of the minpoly(A, v) algorithm, which is a triangular system solve. The first two steps have already been computed. We have a straightforward diagram for our implementation.

Figure 2.2: Diagram for Keller-Gehrig's algorithm



2.3.3 LU-Krylov method for characteristic polynomial

Krylov methods, described for instance in [Hou64], are based on the similarity transformation of the matrix A. The first step is to reduce A to an upper block triangular matrix

$$U^{-1}AU = \begin{bmatrix} F & Y \\ 0 & A_2 \end{bmatrix}$$

where F is the companion matrix of minpoly(A, v). Recursively calling the algorithm on A_2 makes the matrix block upper triangular to compute the characteristic polynomial. The characteristic polynomial is the product of the companion block polynomials.

The above method is a global outline. Let us specify some steps in more detail. Several algorithms, such as Danilevskii's, Samuelson's and others in [Hou64], can be viewed as variants of Krylov's method. Their complexity is $O(n^3)$ in the worst case.

We propose a variation of Krylov's method, similar to Danilevskii's algorithm. Whereas Danilevskii's algorithm uses elementary Gauss-Jordan elimination matrices, we instead use the block LU factorization to similarly transform the matrix into a block triangular form. Although this does not provide any theoretical improvement of the time complexity, the use of blocks makes it of practical interest.

ALGORITHM 2.11. *A variation of Krylov method*
Input:

- A, *an* $n \times n$ *matrix over a field* $\mathbb{F}$

Output:

- charpoly(A), *characteristic polynomial of* A

Procedure:

1. *Pick a random vector* v
2. *Compute* $X = K(A, v)^t$
3. *Compute* $(L, S, P) = LSP(X^t)$
4. *Compute* $M(x) =$ minpoly(A, v) *with the algorithm 2.7.*
5. *if (*$deg(M) = n$*) then return* charpoly$(A) = M(x)$
6. *else*
7. *Compute* $A' = PA^tP^t = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix}$, *where* A_{11} *is* $k \times k$.
8. *return* charpoly$(A) = M(x)$charpoly$(A'_{22} - A'_{21}S_1^{-1}S_2)$

THEOREM 2.12. *Given an* $n \times n$ *matrix over a field* $\mathbb{F}$*, the algorithm 2.11 above correctly computes the characteristic polynomial of* A*.*

Proof. The LSP factorization computes:

$$X = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} [S_1|S_2]P$$

Let us deduce the invertible matrix

$$\overline{X} = \underbrace{\begin{bmatrix} L_1 & 0 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{L}} \underbrace{\begin{bmatrix} S_1 & S_2 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{S}} P = \begin{bmatrix} X_{1..k} \\ \begin{bmatrix} 0 & I_{n-k} \end{bmatrix} P \end{bmatrix}$$

Now $X_{1..k}A^t = C^t X_{1..k}$, where C is the companion matrix associated to minpoly(A, v). Therefore

$$\begin{aligned} \overline{X}A^t\overline{X}^{-1} &= \left[\begin{array}{c|c} C^t & 0 \\ \hline \begin{bmatrix} 0 & I_{n-k} \end{bmatrix} PA^tP^t\overline{S}^{-1}\overline{L}^{-1} & \end{array}\right] = \left[\begin{array}{c|c} C^t & 0 \\ \hline \begin{bmatrix} A'_{21} & A'_{22} \end{bmatrix} \overline{S}^{-1}\overline{L}^{-1} & \end{array}\right] \\ &= \left[\begin{array}{c|c} C^t & 0 \\ \hline Y & X_2 \end{array}\right] \end{aligned}$$

where $X_2 = A'_{22} - A'_{21}S_1^{-1}S_2$. So we know the algorithm 2.11 returns the correct answer.

□

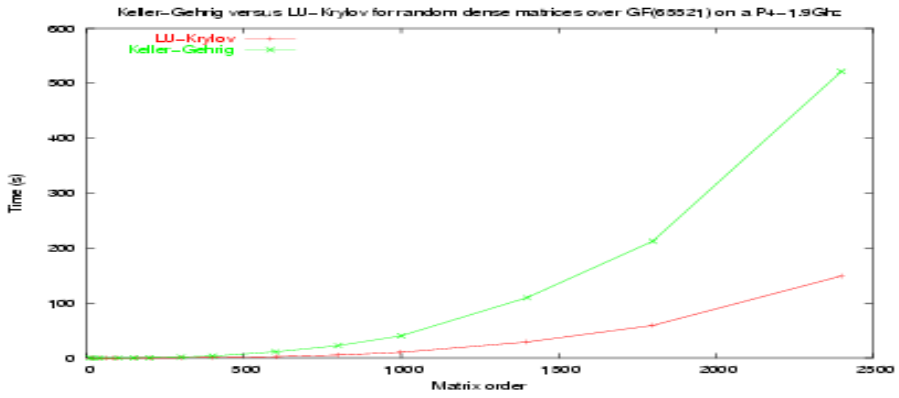
2.3.4 Experiments: LU-Krylov vs. Keller-Gehrig

For our experiments, we implement these algorithms in C++ routines, using a template field representation, and the standard BLAS interface. We use the modular finite field representation of Givaro and the portable BLAS implementation, ATLAS.

Firstly, we use random matrices. For these matrices, the characteristic polynomial equals the minimal polynomial. The LU-Krylov algorithm only computes one LSP factorization for the minimal polynomial, whereas the Keller-Gehrig algorithm needs $\log(n)$ LSP factorizations. That means LU-Krylov algorithm is faster. This agrees with what Figure 2.3 shows.

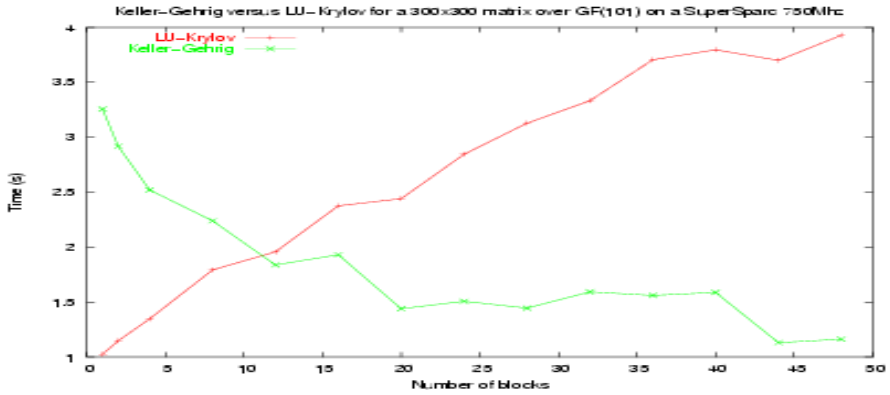
After that, we compare the timing for a given order with a variant number of companion matrices in the Frobenius normal form. As shown in figure 2.4, LU-Krylov is

Figure 2.3: Over random matrices



faster when the number of invariant factors is small, and after a cross-over point, Keller-Gehrig is faster. The reason for this is that LU-Krylov needs almost as many LSP factorization as the number of blocks, whereas Keller-Gehrig needs $\log(d)$ of them, where d is the degree of minimal polynomial.

Figure 2.4: Over matrices with rich structures



From these comparisons, it appears that neither algorithm is always better. The user may choose between them, depending on the number of companion matrices in the Frobenius normal form. The two algorithms could be combined into an adaptive one:

LU-Krylov is used for the computation of the first block. Then if the degree of the corresponding minimal polynomial is low (i.e. under a given threshold), Keller-Gehrig is used for the rest of the computation.

2.4 Minimal and characteristic polynomials over the integers

Minimal polynomial of integer matrices can be obtained by computing modulo many different primes and using the Chinese remainder algorithm. For a $n \times n$ integer matrix A and a prime p, we define A_p to be the image matrix of A over the field $\mathbb{Z}_p$. The minimal polynomial of A_p is well defined and may or may not be equal to the minpoly(A) $(\mathrm{mod}\ p)$. If they are equal, we call p a "good" prime, otherwise we call p a "bad" prime. For example, $A = \begin{bmatrix} 3 & 0 \\ 0 & 6 \end{bmatrix}$. We know that

$$\text{charpoly}(A)(x) = \text{minpoly}(A)(x) = x^2 - 9x + 18.$$

For prime 5,

$$\text{minpoly}(A_5)(x) = x^2 - 4x + 3 = \text{minpoly}(A) \pmod 5,$$

thus 5 is a good prime. While for prime 3,

$$\text{minpoly}(A_3)(x) = x \neq \text{minpoly}(A)(x) \pmod 3,$$

thus 3 is a bad prime. For an $n \times n$ integer matrix A, the number of bad primes is at most $\mathrm{O}^{\sim}(n^2 \log_2(\|A\|))$ - see [DSV00, page 10]. An early termination technique can be used to reduce the number of primes required, and therefore reduce the practical run time. There is already some discussion about the early termination, for example [Emi98]. But here, we use an enhanced and more efficient termination technique. Only one integer, called "certificate", is constructed instead of the actual answer mod modulus so far at each step. This technique can be easily adopted to other cases, for example, solving a non-singular integer systems over the rationals. The following is a special case of the generic one - see [SW05].

ALGORITHM 2.13. *Minimal polynomial with early termination*

Input:

- A, *an* $n \times n$ *integer matrix.*
- $\mathcal{P}$, *a set of odd primes of size* M, *with minimal prime* β.
- Ω, *the random sample size.*

Output:

- v, *coefficients of the minimal polynomial of* A.

Procedure:

1. *Set* $l := 0$, *the length of the output vector.*
2. *Set list* $\mathcal{L} := \emptyset$, *list of pair (good prime, minimal polynomial of* A *mod it).*
3. *Choose a random vector* x *of length* n *with entries independently and uniformly chosen from* $[0, \Omega - 1]$.
4. *Uniformly choose a prime* p *from* $\mathcal{P}$ *which has not been used before. Mark* p *as being used.*
5. *Compute* v = *minpoly*$(A \mod p)$
   If its degree is lower than l, *goto statement 3.*
   If its degree is larger than l, *set* l *to be this length, empty* $\mathcal{L}$, *append pair (*p, v_p*) to* $\mathcal{L}$, *and goto statement 2.*
   If its degree is equal to l, *append pair (*p, v_p*) to* $\mathcal{L}$. *Use the Chinese remainder algorithm to construct a certificate* $c^{(i)}$ *where* i *the size of* $\mathcal{L}$, *such that* $c^{(i)} = x \cdot v_q \pmod q$, *for each pair* (q, v_q) *in* $\mathcal{L}$.

6. *If $c^{(i)} \neq c^{(i-1)}$, then goto statement 3. Otherwise $c^{(i)} = c^{(i-1)}$, i.e., the termination condition is met. Return the vector v, which is constructed from pairs in $\mathcal{L}$ by the Chinese remainder algorithm, such that $v = v_q \pmod q$, for every pair (q, v_q) in $\mathcal{L}$. This construction can be done by using a divide-and-conquer method.*

REMARK. 1. In order to capture negative numbers, we normalize the final number a to lie in $[-(m-1)/2, (m-1)/2]$, where m=$\prod_{1 \leq i \leq n} p_i$.

2. The pre-selected primes idea in [Kal02] may be used here also. It works with preselected prime stream so that one can pre-compute the constants which are independent of the actual answer, for instance $(p_1 \ldots p_i)^{-1} \mod p_{i+1}$ (The precomputed constant will help if the Newton interpolation algorithm is applied to construct the final answer). Such moduli are not random; additional post-check of the result at one additional random moduli will guarantee the correctness of the final answer with very high probability. Please see [Kal02] for more details.

THEOREM 2.14. *Given an $n \times n$ integer matrix, if we choose the set of odd primes to be odd primes of magnitude at most $n^\gamma \log n(\log n + \log_2(\|A\|))$ (γ is a real constant bigger than 2), then algorithm 2.13 above computes the minimal polynomial with error probability at most*

$$\frac{2}{\Omega} + \mathrm{O}(n^{2-\gamma}).$$

It requires $\mathrm{O}^\sim(n \log_2(\|A\|))$ minimal polynomial computation over finite fields.

Note: For a typical problem, such as characteristic polynomial, or minimal polynomial, it is easy to choose a reasonable set of primes and random sample size such that the error probability is tiny. Runs are independent. If we repeat the algorithm, the error probability is squared.

Proof. Let vector α denote the correct answer and c denote $x \cdot \alpha$. We assume the early termination condition is met at some number n, i.e. $c^{(n)}$ is equal to $c^{(n-1)}$. If both $|c| \geq ||\alpha||_\infty$ and $c = c^{(n)}$ are true, then the algorithm returns the correct answer. This is true

since the modulus, which is the product of the primes in $\mathcal{L}$, is at least $2||\alpha||_\infty$ under these hypotheses. Therefore, there are only two sources of errors. One is the bad certificate. The other is the premature termination.

The next statement explores the probability of a bad certificate. If x is a random vector with entries independently and uniformly chosen from the integer set $[0, \Omega - 1]$, then

$$\text{Prob}(|x \cdot \alpha|) < ||\alpha||_\infty) \leq \frac{2}{\Omega}.$$

This statement is true since there is at least one entry of α, whose absolute value is equal to $||\alpha||_\infty$. Without loss of generality, we assume $|\alpha_0| = ||\alpha||_\infty$. Then for any $x_1, \ldots, x_{l-1}$, there are at most two integers x_0, such that $|x \cdot \alpha| < ||\alpha||_\infty$. Therefore

$$\text{Prob}(|x \cdot \alpha|) < ||\alpha||_\infty) \leq \frac{2}{\Omega}.$$

Let us look at the probability of a premature termination. On the condition that the early termination condition $c^{(n)} = c^{(n-1)}$ is met, the probability that $c \neq c^{(n)}$ is at $\text{O}(n^{2-\gamma})$. This can be followed from the theorem [Kal02, Theorem 1.]. So the total error probability is at most

$$\frac{2}{\Omega} + \text{O}(n^{2-\gamma}).$$

The number of the minimal polynomial calls over finite fields can be obtained by estimating the maximal bit length for coefficients of minimal polynomial A - see e.g. [KV03]. □

Combining Wiedemann's algorithm and baby/giant step over the dense case, the algorithm [Kal02, KV03] can compute the minimal polynomial with asymptotic improvement from $\text{O}^\sim(n^4 \log_2(\|A\|))$ to $\text{O}^\sim(n^{3+1/3} \log_2(\|A\|))$ with standard matrix multiplication.

In practice, it is often cheaper to compute minimal polynomial than characteristic polynomial over the integers. In [KV03], Kaltofen and G. Villard point out that the characteristic polynomial can be computed by using Hensel lifting on the minimal polynomial. In [DPW05, section 4.4], the implementation detail has been presented.

2.5 An application

A key step in the computation of the unitary dual of a Lie group is to determine if certain rational symmetric matrices are positive semi-definite. The computation of signatures of certain matrices which are decided by root systems and facets(see e.g. [ASW05]) can help classify the unitary representations of Weyl groups. The most interesting case is that of $E8$, which is a Weyl group with order $696,729,600$. $E8$ has 112 representations, the largest of which has dimension $7,168$. Though results on $E8$ are known, our successive computation will encourage to calculate the unitary dual of more general cases such as Hecke algebras, real and p-adic Lie groups.

The minimal polynomial can be used to determine positive (negative) definiteness and positive (negative) semi-definiteness. By Descartes' rule, the characteristic polynomial can be used to determine the signature which is the difference of positive eigenvalues and negative ones. These special rational matrices from Lie groups representation can be scaled to integer matrices by multiplying by the lcm of the denominators of all entries. For these specially constructed matrices from Lie group representation, the lcm of the denominators of all entries is just a little larger than each individual one. This transformation keeps the signature.

On a challenge facet $1/40(0,1,2,3,4,5,6,23)$ cases, all constructed rational matrices are non-singular and we verified that all these rational matrices of dimension up to a few thousand are positive definite. On another special facet $(0,1/6,1/6,1/4,1/3,5/12,7/12,23/12)$, all constructed rational matrices have low ranks and we verified all matrices (largest dimension $7,168$) are positive semi-definite. Our success experiments shows the computation for classifying the representation of Lie groups is feasible.

Chapter 3

COMPUTATION OF THE RANK OF A MATRIX

3.1 Introduction

Determining the rank of a matrix is a classical problem. The rank of an integer matrix can be probabilistically determined by computation modulo a random prime. The rank of an integer matrix can also be certified by using its trace - see [SSV04] for details. It can be made to be deterministic if the rank is determined by computation modulo a sufficient number of distinct primes.

For a dense matrix over a finite field, its rank can be found by using Gaussian elimination. Moreover, pivoting in Gaussian elimination is not as important in exact computation as it is in numerical Gaussian elimination. In the numeric linear method, the answer is approximate due to floating point rounding errors, and a pivoting strategy is used to improve the accuracy of the answer and make the algorithm backward stable. But in symbolic computation, only exact field operations are used, and the algorithm always gives the correct answer. A block implementation such as [DGP04] speeds up the practical run time of Gaussian elimination.

However, there are choices of algorithms for large sparse matrices. For a sparse matrix over a finite field, its rank can be computed by either Krylov space methods such as Wiedemann's method [Wie86, KS91], and block Wiedemann's algorithm [Cop95, Kal95], or sparse elimination such as SuperLU. Each algorithm is favorable to certain cases. In practice, it is hard to make the choice of algorithms for a given sparse matrix. We contribute an adaptive (introspective) algorithm to solve this puzzle. This adaptive

method is obtained by carefully studying each algorithm and comparing these two distinct methods. It works by starting with an elimination and switching to a Krylov space method if a quick rate of fill-in is detected. We have studied two typical methods. One method is GSLU (Generic SuperLU) [DSW03], an adaptation of the SuperLU method [Sup03], and a fundamentally Gaussian elimination method with special ordering strategies. The second method, called here BB for "black box", is a variant of the Krylov space-based Wiedemann's algorithm [Wie86]. Our experiment shows that our adaptive method is feasible and very efficient. The adaptive idea may be easily adopted to other problems such as computing determinants and solving linear systems. The observations made here apply quite directly to those problems.

A portion of the work in this chapter were presented at SIAMLA 2003 [DSW03] and ACA2003[LSW03].

3.2 Wiedemann's algorithm for rank

DEFINITION 3.1. The rank of a matrix over a field is the dimension of the range of the matrix. This corresponds to the number of linearly independent rows or columns of the matrix.

Algorithm 2.6 can be used to compute the minimal polynomial of a matrix over finite fields. With preconditioning as in [EK97, KS91, CEK$^+$02], it likely has no nilpotent blocks of size greater than 1 in its Jordan canonical form (i.e. x^2 doesn't divide its minimal polynomial) and its characteristic polynomial will be equal to the shifted minimal polynomial with high probability. We define the *shifted minimal polynomial* to be the polynomial which is the product of the minimal polynomial and a power of x and has degree of the matrix's order. In the case that a matrix has no nilpotent blocks of size greater than 1 in its Jordan canonical form, the rank can be determined from the characteristic polynomial and is equal to the degree of its characteristic polynomial minus the

multiplicity of its zero roots. The next theorem is from [CEK$^+$02, Theorem 4.7] about diagonal preconditioning for the symmetric case.

THEOREM 3.2. *Let A be a symmetric $n \times n$ matrix over a field $\mathbb{F}$ whose characteristic is 0 or greater than n and let S be a finite subset of $\mathbb{F} \backslash 0$. If $d_1, \ldots, d_n$ are chosen uniformly and independently from S and $D = \text{diag}(d_1, \ldots, d_n)$, then the matrices A and DAD have the same rank, the minimal polynomial of DAD is square free, and the characteristic polynomial is the production of the minimal polynomial and a power of x, with probability at least $1 - 4n^2/\#S$. This probability increases to $1 - 2n^2/\#S$ if the squares of elements of S are distinct.*

Note that if the minimal polynomial is square free, then the matrix has no nilpotent blocks of size greater than 1 in its Jordan canonical form. For the non-symmetric case, an additional random diagonal D may be used. We have the following lemma:

LEMMA 3.3. *Let A be a non-symmetric $n \times n$ matrix over a field $\mathbb{F}$ whose characteristic is 0 or greater than n and let S be a finite subset of $\mathbb{F}$. If $d_1, \ldots, d_n$ are chosen uniformly and independently from S and $D = \text{diag}(d_1, \ldots, d_n)$, then the matrices A and $A^T DA$ has the same rank with probability at least $1 - n/\#S$.*

Proof. This is a simple case in [EK97, lemma 6.1]. Note that the matrix $A^T DA$ can be viewed as a polynomial matrix over $\mathbb{F}[d_1, \ldots, d_n]$ and has the same rank as A. So there is a non-singular sub-matrix of $A^T DA$ with dimension of A's rank. So the determinant of this sub-matrix is a multi-variable polynomial over $\mathbb{F}[d_1, \ldots, d_n]$ with total degree at most n. By the Schwartz-Zippel lemma [Sch80], the probability that this determinant is equal to 0 is at most $n/\#S$. □

Wiedemann's algorithm for the rank using diagonal preconditioning is straightforward.

ALGORITHM 3.4. *Wiedemann's algorithm for rank*
Input:

- *A, a $n \times n$ matrix over a finite field $\mathbb{F}$*

Output:

- *The rank of A*

Procedure:

1. *Choose $n \times n$ diagonal matrices D_1 and D_2 with diagonal entries uniformly and independently from $\mathbb{F} \backslash 0$.*
2. *Compute the minimal polynomial of $D_2 A^T D_1 A D_2$*
3. *If the constant coefficient of the minimal polynomial is non-zero, return its degree. Otherwise, return its degree -* 1.

Note that $D_2 A^T D_1 A D_2$ can be viewed as a composition of the blackbox matrices D_2, A^T, D_1, A, D_2 and can be used as a blackbox. That is, $D_2 A^T D_1 A D_2$ doesn't need to be computed explicitly and its matrix-by-vector product can be computed via a sequence of matrix-by-vector products of D_2, A, D_1. Other preconditioning like Toeplitz matrices, sparse matrices may be used also - see [EK97, KS91, CEK$^+$02]

3.3 GSLU: a sparse elimination

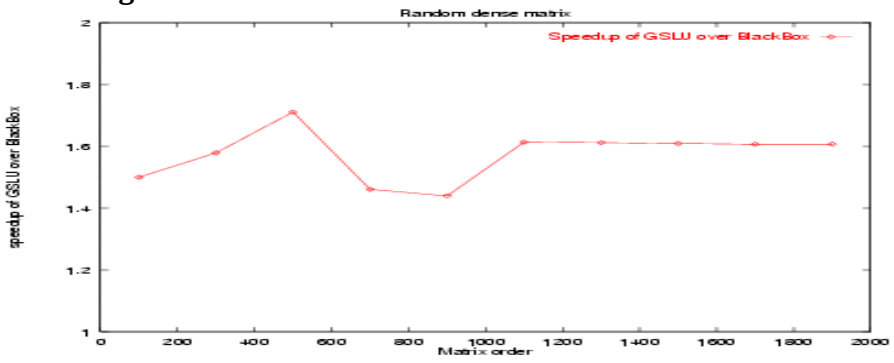
GSLU (Generic SuperLU) is adapted to LinBox library from SuperLU version 2.0 [Sup03] and can be used with arbitrary fields including finite field representations. Basic GSLU is an elimination method, and uses pre-ordering to keep sparsity of L and U during elimination. There are five possible ordering choices (natural ordering, multiple minimum degree applied to structure $A^T A$, multiple minimum degree applied to structure $A^T + A$, column approximation minimum degree, and supplied by user as input) for pre-ordering the columns of a matrix A. After that, a column-based Gaussian elimination is used. It works like the incremental rank method, first performing LUP in the first k columns, adding the $(k+1)^{th}$ column and updating L, U and P. We call time updating L, U and P *the $(k+1)^{th}$ step time*.

3.4 Complexity comparison

The run time of algorithm BB is quite reliably computed a priori. Suppose the matrix has order n, rank r, and E nonzero entries. Then each matrix vector product costs E multiply-adds in general. With an early termination technique [BEPP99, Ebe03], the algorithm requires $2r$ matrix-by-vector products with the matrix A and $\tilde{O}(rn)$ additional field operations consisting of dot products, matrix-by-vector products of the preconditioner, and the Berlekamp-Massey step. In contrast, with all elimination methods, the run time of GSLU is quite variable. It depends on the rate of fill-in, which in turn depends on the success of the fill-in avoidance method used.

Let us count the field operations for each method. For simplicity, we will add multiply-add to the traditional field operations consisting of adding, subtracting, multiplying, dividing. For an $n \times n$ sparse matrix of rank r with E nonzero entries, we have the following: BB requires $2rE + rn$ field operations and the number of field operations required by GSLU varies from about E to $\frac{1}{3}rn^2$ field operations. Let us apply it to the special $n \times n$ dense matrix case with rank r: BB requires about $2rn^2$ arithmetic operations and GSLU requires about $\frac{1}{3}rn^2$ arithmetic operations. Thus for the dense case, the ratio $time(BB)/time(GSLU)$ is roughly 6 in theory. The practical ratio is 1.6 as

Figure 3.1: Blackbox VS GSLU over random dense matrices



Test was run on a P-4 1.9G HZ with 1G RAM

demonstrated in figure 3.1, due to our special optimal implementation for matrix-by-vector product.

For the general sparse case, it is very difficult to predict which method will run faster. Generally speaking, BB is superior for very large matrices, in particular when fill-in causes the matrix storage to exceed machine main memory and even exceed the available virtual memory. In practice, fill-in is hard to detect. Evidently, there are some small dimension matrices (less then 200×200) for which BB is faster than GSLU, while there are some very huge dimension sparse matrices, close to $100,000 \times 100,000$, for which GSLU is much faster than BB.

3.5 An adaptive algorithm

Due to the uncertain run time of algorithms for computing the rank of a sparse matrix, it is desirable to have an algorithm that can intelligently choose among various available algorithms for computing the rank and the given matrix. We call such an algorithm an adaptive algorithm. For computing the rank of a sparse matrix, our adaptive algorithm starts with a sparse elimination algorithm, but may switch to a blackbox algorithm under certain appropriate conditions. Let us explore more properties about the sparse elimination algorithm, GSLU. The GSLU algorithm is a "left looking" elimination, which has the consequence that the cost of elimination steps tends to be an increasing function of step number. After testing on many examples, we observe that GSLU will increase the time for each step dramatically at a very early stage, then step time will increase very slowly and never decrease. This gives us a chance to recognize rapid growth of step cost and switch to the BB method at a relatively early stage. At each step we can estimate the remaining cost by assuming the same cost for all future steps (the rectangular region in the figure). If it exceeds BB cost, we switch.

3.6 Experimental results

The adaptive algorithm is based on a simple idea which starts with a sparse elimination and may switch to a blackbox method if the estimated remaining run time exceeds the predicted blackbox method run time. We have implemented the adaptive algorithm in the LinBox library. We have chosen our examples from various sources, such as J. Dumas's sparse integer matrices collection* and matrix market†. In following discussion of performance, we use "relative efficiency". The relative efficiency is defined by the ratio of the run time by one method over the run time by the best method. We choose these examples based on matrix orders. If the sparse elimination method is better than the blackbox method on a given matrix, then the adaptive algorithm computes the rank of the matrix via the sparse elimination directly without switching to the blackbox method. So in this case, the adaptive algorithm only takes as long as the minimum run times of the sparse elimination and the blackbox method. That is, the relative efficiency of the adaptive algorithm is 1. We leave out these cases where the sparse elimination is better than the blackbox method in fig 3.2. The main point in figure 3.2 is to show that the overhead of the adaptive algorithm is light and the adaptive algorithm can run efficiently even in the "worst" cases. We have a generic race method in which two methods parallel run in two machines with the same input arguments and if one finishes first, immediately kill the other. So the parallel run time of the race is always equal to the minimum run time of these two. The work is twice as much as the parallel run time. So the relative efficiency of the race method is guaranteed to be $1/2$.

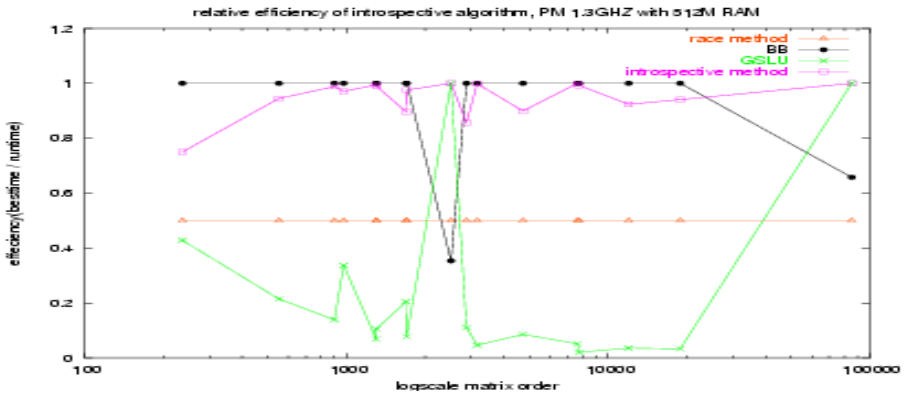
In the performance figure, the x-axis is the matrix order, the y-coordinate is relative efficiency. We clearly see that the relative is above 0.8 for our adaptive method when the matrix order is over 1000, and above 0.7 when matrix order is small. This means that our

---

* `http://www-lmc.imag.fr/lmc-mosaic/Jean-Guillaume.Dumas/simc.html`

† `http://math.nist.gov/MatrixMarket/`

Figure 3.2: Experimental result of adaptive algorithm



adaptive algorithm of elimination and Blackbox methods is advisable and very effective in practice.

Chapter 4

COMPUTATION OF EXACT RATIONAL SOLUTIONS FOR LINEAR SYSTEMS

4.1 Introduction

Finding the rational solution of a non-singular integer linear system is a classic problem. In contrast to numerical linear methods, the solution is computed symbolically (exactly) without any error. Rational solving can be applied to solve the Smith form problem, Diophantine equations, etc. By Cramer's rule, there is a unique solution to a non-singular integer linear system. The modular method is a good way to find the exact solution. The solution can be found either by computation modulo many distinct primes and the Chinese remainder algorithm or by p-adic lifting [Dix82, MC79], a special case of Hensel lifting [vzGG99, section 15.4]. The latter is better, since the asymptotic complexity of p-adic lifting is cheaper by a factor of $\mathrm{O}^{\sim}(n)$ than the first one. It is not a good idea to find the rational solution of a non-singular integer linear system by using Gaussian elimination over the rationals. For an $n \times n$ non-singular integer matrix A, and a column integer vector b, the solution of $Ax = b$ is a column vector with rational entries whose numerators and denominators have $\mathrm{O}^{\sim}(n(\log_2(\|A\|) + ||b||_\infty))$ bits. Thus the cost using Gaussian elimination over the rationals will be $\mathrm{O}^{\sim}(n^4(\log_2(\|A\|) + ||b||_\infty))$. This method costs by an $\mathrm{O}^{\sim}(n)$ factor more expensive than p-adic lifting and is not memory efficient.

We contribute a new algorithm which works in well-conditioned cases with the same asymptotic complexity as the p-adic lifting and aborts quickly in ill-conditioned

cases. Success of this algorithm on a linear equation requires the linear system to be sufficiently well-conditioned for the numerical linear algebra method being used to compute a solution with sufficient accuracy. This approach will lead to high performance in practice. Another contribution is an adaptive algorithm for computing the exact solution of a non-singular linear system. The idea is to start with our new algorithm, and if it aborts, switch to p-adic lifting.

4.2 p-adic lifting for exact solutions

Working with a prime p, developing solutions mod p^k (i.e. in p-adic arithmetic), and finally reconstructing the rational solution can be used to find the exact solutions of linear equations. The rational reconstruction can be done using the Euclidean algorithm for the gcd of two integers - see [vzGG99, section 5.7]. The following is the outline of this method:

ALGORITHM 4.1. *p-adic lifting for exact solutions*
Input:

- A, *an* $n \times n$ *non-singular integer matrix*
- $b \in \mathbb{Z}^n$, *a column vector*
- p, *a prime such that* $p \not| \det(A)$

Output:

- $x \in \mathbb{Q}^n$, *the rational solution of* $Ax = b$.

Procedure:

1. *Compute* $\delta = \beta^n n^{n/2}$, *which bounds absolute values of numerators and denominators of solution to* $Ax = b$. *Where* β *is a bound on the absolute values of entries of* A *and* b.

2. *Compute* C *such that* $CA = I_n \pmod{p}$

3. *Set* $b_0 = b$

4. *Compute* $x_0 = Cb_0 \pmod{p}$

5. *For* $i = 0$ *to* $2\lceil \log \delta / \log p \rceil$ *do*

   - *Compute* $b_{i+1} = p^{-1}(b_i - Ax_i)$
   - *Compute* $x_{i+1} = Cb_{i+1} \pmod{p}$

6. *Rationally reconstruct* x

THEOREM 4.2. *The algorithm 4.1 above correctly computes the rational solution and requires* $\mathrm{O}^{\sim}(n^3 \log_2(\|A\|))$ *bit operations.*

Proof. See [Dix82] for details. □

Algorithm 4.1 can be modified to work with sparse matrices by replacing the method of solving the linear system modulo p by Wiedemann's method (see algorithm 2.6). In step 5, each solution will be involved with degree(minpoly(A)) matrix-by-vector products. It requires totally $n^2 \log_2(\|A\|)$ matrix-by-vector products in the worst case. Thus the algorithm will be no more of efficient time, but it can save space.

4.3 Using numerical methods

Both symbolic methods and numerical methods can be used to solve linear systems. But the tow methods use different techniques and have been developed independently. Symbolic methods for solving linear systems are based on modular methods via solving the linear system modulo a large integer and finally reconstructing the rational solution. Either p-adic lifting [Dix82, MC79] or computation modulo many different primes and using the Chinese remainder algorithm are the widely used methods to compute solutions of linear systems modulo a large integer. Numerical methods use either

direct methods like Gaussian Elimination (with or without pivoting), QR factorization, or iterative methods such as Jacobi's method, Lanczos' method, or the GMRES method - see [Dem97, Saa03, TB97] for details. Symbolic methods for solving linear systems can deliver the correct answer without any error, though they are usually more expensive in computation time than numerical methods. But numerical linear algebra methods for solving linear systems are subject to the limitation of floating point precision and iterative methods are subject to additional convergence problems.

In this section, we describe a new algorithm to exactly solve well-conditioned integer linear systems using numerical methods. A portion of this work has been submitted to the journal of symbolic computation [Wan04]. A secondary benefit is that it aborts quickly in ill-conditioned cases. Success of this algorithm requires the system to be sufficiently well-conditioned for the numeric linear algebra method being used to compute a solution with sufficient accuracy. Though this algorithm has the same asymptotic cost as p-adic lifting, it yields practical efficiency. The practical efficiency results from the following two things. First, over the past few decades, hardware floating point operations have been sped up dramatically, from a few hundred FLOPS in 1940s to a few GFLOPS now, even in PCs. Second, many high performance numerical linear algebra packages have been developed using fast BLAS implementation for dense systems.

The motivation for this algorithm is the high performance of numerical linear algebra packages and the simple fact below:

FACT 4.3. If two rational numbers $r_1 = \frac{a}{b}$, $r_2 = \frac{c}{d}$ are given with $\gcd(a, b) = 1$, $\gcd(c, d) = 1$, and $r_1 \neq r_2$, then $|r_1 - r_2| \geq \frac{1}{bd}$.

That is, rational numbers with a bounded denominators are discrete, though it is well known that all rational numbers are dense in the real line. Because of this simple fact, if a solution with very high accuracy can be computed, then the rational solution can be reconstructed.

In general, numerical methods are inexact when carried out on a computer: answers are accurate up to at most machine precision (or software floating point precision). In order to achieve more accuracy than machine precision, our idea is straightforward: approximation, amplification, and adjustment. We first find an approximate solution with a numerical method, then amplify the approximate solution by a chosen suitable scalar, adjust the amplified approximate solution and corresponding residual so that they can be stored exactly as integers of small magnitude. Then repeat these steps until a desired accuracy is achieved. The approximating, amplifying, and adjusting idea enables us to compute the solution with arbitrarily high accuracy without any high precision software floating point arithmetic involved in the whole procedure or big integer arithmetic involved except at the final rational reconstruction step. Detail will be discussed in section 4.5.

The next section is a classic result about the rational reconstruction. Continued fractions which give the best rational approximations of real numbers enable us to reconstruct a rational number with certain constraints from a real number. In the following section, we describe a way to achieve arbitrary accuracy using numerical linear methods, on the condition that inputs are integers and matrices are well-conditioned. Finally the potential usage of our new algorithms for sparse linear systems is demonstrated with a challenging problem.

4.4 Continued fractions

The best approximation with a bounded denominator of a real number is a segment of its continued fraction. Just as a quick reminder, a brief description of the continued fraction is given. For a real number r, a simple continued fraction is an expression in the form

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}},$$

where all a_i are integers. From now on, we assume that if $r \geq 0$, then $a_0 \geq 0$, all $a_i > 0$ for $i \geq 1$, and if $r < 0$, then $a_0 \leq 0$, all $a_i < 0$ for $i \geq 1$. A more convenient notation is $r = [a_0; a_1, a_2, \cdots]$. Intuitively, we can apply the extended Euclidean algorithm to compute the simple continued fraction of a rational number.

For example, let $r = \frac{3796}{1387}$. We can compute $\gcd(3796, 1387)$ using Euclid's algorithm, $3796 = 1387 \cdot 2 + 1022$; $1387 = 1022 \cdot 1 + 365$; $1022 = 365 \cdot 2 + 292$; $365 = 292 \cdot 1 + 73$; $292 = 73 \cdot 4$. We re-write these equations, $3796/1387 = 2 + 1022/1387 = 2 + 1/(1387/1022) = 2 + 1/(1 + 365/1022) = 2 + 1/(1 + 1/(1022/365))$ $= 2 + 1/(1 + 1/(2 + 292/365) \cdots = 2 + 1/(1 + 1/(2 + 1/(1 + 4))) = [2; 1, 2, 1, 4]$.

For a simple continued fraction for r (either finite or infinite) one defines a family of finite segments $s_k = [a_0; a_1, a_2, ..., a_k]$, each s_k being a rational number: $s_k = p_k/q_k$ with $q_k > 0$ and $\gcd(p_k, q_k) = 1$. Below we list a few properties of the simple continued fraction in $r \geq 0$ cases. There are similar properties in $r < 0$ cases. Further detail about these properties can be found on the internet, for example
`http://www.cut-the-knot.org/do_you_know/fraction.shtml` and
`http://mathworld.wolfram.com/ContinuedFraction.html`.

1. Every rational number can be associated with a finite continued fraction. Irrational numbers can also be uniquely associated with simple continued fractions. If we exclude the finite fractions with the last quotient equal to 1, then the correspondence between rational numbers and finite continued fractions becomes one to one.

2. For all $k \geq 2$, $p_k = a_k p_{k-1} + p_{k-2}$, $q_k = a_k q_{k-1} + q_{k-2}$.

3. $q_k p_{k-1} - p_k q_{k-1} = (-1)^k$. And $s_k - s_{k-1} = (-1)^{k-1}/(q_k q_{k-1})$.

4. $s_0 < s_2 < s_4 < s_6 < \cdots < r < \cdots < s_7 < s_5 < s_3 < s_1$.

Based on the nice properties about continued fraction above, now we can prove:

THEOREM 4.4. *Given r, $B > 0$ there is at most one rational solution $\frac{a}{b}$ such that $|\frac{a}{b} - r| < \frac{1}{2Bb}$, $0 < b \leq B$, and $\gcd(a, b) = 1$. Moreover, if there is one rational solution $\frac{a}{b}$, then for some k, $\frac{a}{b} = \frac{p_k}{q_k}$, where (p_k, q_k) is a segment of the simple continued fraction of r, such that either $p_k/q_k = r$ or $q_k \leq B < q_{k+1}$. Moreover, if $r = \frac{n}{d}$, then there is an algorithm to compute (p_k, q_k), which requires $O(M(l) \log l)$ bit operations, , where l is the maximum bit length of n and d.*

Note: if $B = 1$, then there is either no solution or $\frac{a}{b} = \frac{\text{nearest integer to r}}{1}$.

Proof. First we prove there is at most one solution. By way of contradiction, if there are two different rational solutions $\frac{a_1}{b_1}$ and $\frac{a_2}{b_2}$ with

$$0 < b_1, b_2 \leq B, \gcd(a_1, b_1) = \gcd(a_2, b_2) = 1, \frac{a_1}{b_1} \neq \frac{a_2}{b_2}.$$

Then

$$|\frac{a_1}{b_1} - \frac{a_2}{b_2}| \leq |\frac{a_1}{b_1} - r| + |\frac{a_2}{b_2} - r| < \frac{1}{2Bb_1} + \frac{1}{2Bb_2} < \frac{1}{b_1 b_2}.$$

This is a contradiction to $|\frac{a_1}{b_1} - \frac{a_2}{b_2}| \geq \frac{1}{b_1 b_2}$ from Fact 4.3. So there is at most one solution.

For the given real number r, we assume k is the unique integer number, such that

$$\text{either } p_k/q_k = r \text{ or } q_k \leq B < q_{k+1}.$$

If $\frac{a}{b}$ is a solution with $0 < b \leq B$, and $\gcd(a, b) = 1$. Then we need to prove $\frac{a}{b} = \frac{p_k}{q_k}$. By way of contradiction, suppose $\frac{a}{b}$ is a solution and $\frac{a}{b} \neq \frac{p_k}{q_k}$. If $p_k/q_k = r$, then $\frac{p_k}{q_k}$ is another rational solution, and we have a contradiction. If $q_k \leq B < q_{k+1}$, then by property 3,

$$|\frac{p_k}{q_k} - \frac{p_{k+1}}{q_{k+1}}| = \frac{1}{q_k q_{k+1}}.$$

And by fact 1 and the assumption, $q_k \leq B < q_{k+1}$,

$$|\frac{a}{b} - \frac{p_k}{q_k}| \geq \frac{1}{bq_k} > \frac{1}{q_k q_{k+1}}.$$

So $\frac{a}{b}$ doesn't lie between $\frac{p_k}{q_k}$ and $\frac{p_{k+1}}{q_{k+1}}$. On the other hand, r must lie between $\frac{p_k}{q_k}$ and $\frac{p_{k+1}}{q_{k+1}}$ by property 4. Thus

$$|\frac{a}{b} - r| \geq \min(|\frac{a}{b} - \frac{p_k}{q_k}|, |\frac{a}{b} - \frac{p_{k+1}}{q_{k+1}}|) \geq \frac{1}{bq_{k+1}}.$$

And by the assumption $\frac{a}{b}$ is a solution, $|\frac{a}{b} - r| < \frac{1}{2Bb}$. Therefore $\frac{1}{2Bb} > \frac{1}{bq_{k+1}}$. Thus $q_{k+1} > 2B$. So,

$$|\frac{p_k}{q_k} - r| \leq |\frac{p_k}{q_k} - \frac{p_{k+1}}{q_{k+1}}| = \frac{1}{q_k q_{k+1}} < \frac{1}{2q_k B}.$$

Therefore $\frac{p_k}{q_k}$ is another solution. This is a contradiction to what we have proven that there is at most one solution.

If $r = \frac{n}{d}$, the (p_k, q_k) can be computed by the half gcd algorithm - please see e.g.[vzGG99, Corollary 11.10]. It needs O$(M(l) \log l)$ bit operations. □

4.5 Exact solutions of integer linear systems

In this section, we present a new way to exactly compute the solution of a non-singular linear system with integer coefficients, repeatedly using a numerical linear algebra method. As is common in numerical analysis, we will use the infinity-norm in the performance analysis. Specifically, for a $n \times m$ matrix $A = (a_{ij})$, we define $||A||_\infty$ = $\max_{1 \leq i \leq n} \sum_{1 \leq j \leq m} |a_{ij}|$, and for a vector b of length n, we define $||b||_\infty$ = $\max_{1 \leq j \leq n} |b_j|$.

It is well known that numerical linear algebra methods are inexact and the accuracy of the solutions is limited to machine precision (or the software floating point precision), when carried out on a computer. Iterative refinement methods (see e.g. [FM67, Dem97, GZ02]) can be used to refine the solution. The refinement, [GZ02] in which a portion, not all, of high precision floating point arithmetics is replaced by lower precision floating point arithmetics, is very efficient in computing a solution with a pre-set high precision. It works this way: for input A, b, and a pre-set precision, initially solve $Ax = b$ in the hardware floating point precision (lower than the pre-set high precision), repeat the following steps enough times to refine the answer until the desired accuracy is achieved: $r = b - Ax$ in a higher precision proportional to the step count, solve $A \cdot \Delta x = r$ in the hardware floating point precision, update $x = x + \Delta x$ in a higher precision proportional to the step count. These refinement methods help compute solutions with a reasonable high accuracy at affordable practical run time cost and worst case complexity cost.

In symbolic linear algebra, however, the high accuracy of solutions required in order to reconstruct the exact answer makes these iterative methods impractical. All the refinement methods above require floating point operations in high precision. And the cost of one software floating point operation increases rapidly with respect to the precision, which has been illustrated in [GZ02, Figure 1]. Also, these refinement methods lead to more expensive complexity for the worst case than the p-adic lifting method. Thus for symbolic computation, we need a better refinement method. Our approximating, amplifying and adjusting idea works as follows: For input integer matrix A and integer vector b. First initialize the solution $x = \frac{1}{d} \cdot n$ with $d = 1$ and vector $n = 0$ and initialize the residual $r = b$. Then repeat the following steps until we achieve a desired accuracy: find an approximate solution $A \cdot \Delta x = r$ in the hardware floating point arithmetic, choose a suitable scalar α, amplify and adjust the Δx with rationals $\frac{1}{\Delta d} \cdot \Delta n$ by setting $\Delta d = \alpha$, $\Delta n = (\approx \alpha \cdot \Delta x_1, \ldots, \approx \alpha \cdot \Delta x_n)$, update answer: $n = \alpha \cdot n + \Delta n$ and $d = \Delta d \cdot d$, update residual $r = \alpha \cdot r - A \cdot \Delta n$. In each refinement iteration, the components of the amplified and adjusted residual are integers of magnitude at most $||b||_\infty + ||A||_\infty$. Our method can be used to achieve any arbitrary high accuracy of the answers without high precision software floating point arithmetic involved in the whole procedure or big integer arithmetic involved except the final rational reconstruction step. After sufficient accuracy is achieved, the final rational solution can be reconstructed.

4.5.1 An exact rational solver for dense integer linear systems

We apply our main idea described above to dense linear systems in detail.

ALGORITHM 4.5. *An exact rational solver for dense case*
Input:

- A, *a non-singular* $m \times m$ *integer matrix.*
- b, *a right hand side integer vector.*

Output:

- $x \in \mathbb{Q}^m$*, the rational solution of* $Ax = b$*. Or it aborts with message "insufficient numerical accuracy".*

Procedure:

1. *Compute a* LUP *factorization of* A *in floating point arithmetic using a backward stable numeric method.* [Other factorizations may be used here too.]

2. *Set integer* $d^{(0)} := 1$. [The common denominator of the answer.]

3. *Set integer vector* $r^{(0)} := b$. [The residual.]

4. *Set integer* $i := 0$. [Step counter]

5. *Compute integer* B*, the Hadamard bound of* A*, which bounds the determinant and all* $(m-1) \times (m-1)$ *minors of* A*.*

6. *repeat the following steps until* $d^{(i)} > 2m \cdot B^2(2^{-i}||b||_\infty + ||A||_\infty)$.

   6.1 $i := i + 1$.

   6.2 *Compute* $\bar{x}^{(i)} = A^{-1}r^{(i-1)}$ *in floating point arithmetic using the* LUP *factorization from step 1.* [An approximate solution $Ax = r^{(i-1)}$.]

   6.3 *Compute the integer scalar,* $\alpha^{(i)} := min(2^{30}, 2^{\lfloor \log_2(\frac{||r^{(i-1)}||_\infty}{||r^{(i-1)} - A\bar{x}^{(i)}||_\infty}) - 1 \rfloor}$ *in floating point arithmetic;* [For the purpose of high performance, α should be chosen to be a power of 2, because in this case, operations of integers multiplying the α can be replaced by more efficient operations of shifting bits. For small size systems, the approximate answer might equal the exact answer. The constant makes the algorithm work in the small size cases. Also, 30 is chosen so that the amplified and adjusted solution and residual can be exactly represented by hardware double floating point numbers.]

6.4 *If* $\alpha^{(i)} < 2$*, abort with error message "insufficient numerical accuracy".* [This situation happens only if A is ill-conditioned.]

6.5 *Exactly compute integer vector* $x^{(i)} := ([\alpha^{(i)} \cdot \bar{x}_1^{(i)}], \ldots, [\alpha^{(i)} \cdot \bar{x}_m^{(i)}])s$*, where* [] means to round to the nearest integer.

6.6 Exactly compute integer $d^{(i)} := d^{(i-1)} \cdot \alpha^{(i)}$.

6.7 Exactly compute integer vector $r^{(i)} := \alpha^{(i)} \cdot r^{(i-1)} - Ax^{(i)}$. [Amplify the residual by a scalar.]

7. *Set* $k := i$

8. *Compute integer vector* $n^{(k)} = \sum_{1 \le i \le k} \frac{d^{(k)}}{d^{(i)}} \cdot x^{(i)}$*, noting* $\frac{d^{(k)}}{d^{(i)}} = \prod_{i < j \le k} \alpha^{(j)}$.

9. *Reconstruct rational solution* x *from* $\frac{1}{d^{(k)}} \cdot n^{(k)}$ *using Theorem 4.4 with denominators bounded by* B.

10. *Return* x.

Note:

1. This algorithm requires the input matrix to be sufficiently well-conditioned for the numerical solver being used to work successfully. If the matrix is ill-conditioned, it is a nice and important property of our algorithm that we quickly detect the failure rather than continuing a nonsensical successive refinement.

2. In solving a linear system, estimating the residual usually is easier than estimating the error. Obviously, the infinity norm of the error is bounded by the product of the infinity norms of the inverse of the input matrix and the residual. In the ith iterator, $r^{(i)}$, the infinity norm of the residual is approximately $\frac{1}{\alpha^{(i)}}$ times the previous one, by the choice of $\alpha^{(i)}$. Thus the infinity norm of the residual is approximately $\frac{1}{d^{(i)}}$. So $e^{(i)}$, the infinity norm of the error is at most approximately $\frac{||A^{-1}||_\infty}{d^{(i)}}$. By Cramer's rule, we know $||A^{-1}||_\infty \le mB$. Thus roughly speaking, $e^i \le \frac{mB}{d^{(i)}}$. Careful analysis

shows that $r^i \leq \frac{\frac{||b||_\infty}{2^i}+||A||_\infty}{d^{(i)}}$. Thus if the loop ends without abortion, the error will be at most $\frac{1}{2Bd^{(k)}}$, and then by theorem 4.4, the correct solution can be constructed from the approximate solution $\frac{n^{(k)}}{d^{(k)}}$.

THEOREM 4.6. *If the $\alpha^{(i)}$ in step 6.3 is not over computed in each iteration, that is, $\alpha^{(i)}$ is no larger than the actual value of $\frac{||r^{(i-1)}||_\infty}{2||r^{(i-1)}-A\bar{x}^{(i)}||_\infty}$ due to floating point approximation, then the algorithm above will either abort or terminate with the correct rational solution in the i^{th} iteration,*

$$||r^{(i)}||_\infty = ||d^{(i)}(b - A\frac{1}{d^{(i)}} \cdot n^{(i)})||_\infty \leq 2^{-i}||b||_\infty + ||A||_\infty.$$

Proof. On the input of A and b, if the algorithm aborts, the statement is true. Otherwise, we need to show the algorithm terminates with the correct rational answer. First we show the algorithm will terminate. Since it doesn't abort, each $\alpha^{(i)}$ is no less than 2. Let us estimate $d^{(i)}$,

$$d^{(i)} = \prod_{1\leq j\leq i} \alpha^{(j)} \geq \prod_{1\leq j\leq i} 2 = 2^i.$$

Thus the loop inside the algorithm runs through finitely many iterations. The algorithm will terminate.

Now we prove the correctness. Let $n^{(i)} = \sum_{1\leq j\leq i} \frac{d^{(i)}}{d^{(j)}} \cdot x^{(j)}$, the numerator of the accumulated solution after the first i iterations. We need to estimate $e^{(i)} = ||\frac{1}{d^{(i)}} \cdot n^{(i)} - A^{-1}b||_\infty$, the norm of the absolute error of the solution in each iteration. By induction, we can prove that

$$r^{(i)} = d^{(i)}(b - A\frac{1}{d^{(i)}} \cdot n^{(i)}). \text{ So}$$

$$e^{(i)} = ||\frac{1}{d^{(i)}} \cdot n^{(i)} - A^{-1}b||_\infty = \frac{1}{d^{(i)}}||A^{-1}r^{(i)}||_\infty.$$

Now we need to estimate $||r^{(i)}||_\infty$. In each iteration, by the hypotheses of the theorem, we have $||A\bar{x}^{(i)} - r^{(i-1)}||_\infty \leq \frac{1}{2\alpha^{(i)}} \cdot ||r^{(i-1)}||_\infty$. By the definition of $x^{(i)}$, we know $||x^{(i)} - \alpha^{(i)} \cdot \bar{x}^{(i)}||_\infty \leq 0.5$. So

$$\begin{aligned} ||r^{(i)}||_\infty &= ||Ax^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}||_\infty \\ &\leq ||\alpha^{(i)} \cdot A\bar{x}^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}||_\infty + ||Ax^{(i)} - \alpha^{(i)} \cdot A\bar{x}^{(i)}||_\infty \\ &\leq \tfrac{1}{2}||r^{(i-1)}||_\infty + \tfrac{1}{2}||A||_\infty. \end{aligned}$$

Therefore $||r^{(i)}||_\infty \leq \frac{1}{2^i}||b||_\infty + ||A||_\infty$. Thus

$$e^{(i)} = \frac{1}{d^{(i)}}||A^{-1}r^{(i)}||_\infty \leq \frac{1}{d^{(i)}}||A^{-1}||_\infty(\frac{1}{2^i}||b||_\infty + ||A||_\infty), \forall i \geq 1.$$

Let k be the value of i when the loop stops. Let us estimate $2B\text{det}(A)e^{(k)}$. So far, we know

$$2B\text{det}(A)e^{(k)} < \frac{2}{d^{(k)}}||B\text{det}(A) \cdot A^{-1}||_\infty(2^{-k}||b||_\infty + ||A||_\infty).$$

It is well known that $\text{det}(A)A^{-1}$ is the adjoint matrix of A. That is, each entry of $\text{det}(A)A^{-1}$ is $(m-1) \times (m-1)$ minor of A. Thus

$$e^{(k)}2B\text{det}(A) \leq \frac{2mB^2(2^{-k}||b||_\infty + ||A||_\infty)}{d^{(k)}} < 1.$$

So we have $e^{(k)} < \frac{1}{2B\text{det}(A)}$. Thus $||\frac{n^{(k)}}{d^{(k)}} - A^{-1}b||_\infty < \frac{1}{2Bdet(A)}$. And also by Cramer's rule we know $\text{det}(A) \cdot A^{-1}b$ is an integer vector. Therefore the reconstructed rational solution must be equal to $A^{-1}b$ by Theorem 4.4. □

Remarks:

1. The asymptotic time complexity is comparable with the Dixon lifting algorithm [Dix82]. For an $n \times n$ well-conditioned matrix with entries of bit length of at most a fixed number, both algorithms require $O^\sim(n^3)$ bit operations.

2. The idea of [PW02, section 4] can be used to accelerate the final rational reconstruction step. In practice, often only a few rational reconstructions instead of n rational reconstructions are done.

3. In implementation, it is possible to choose all $\alpha^{(i)}$ the same, which usually depends on the numerical linear algorithm and the condition number of the matrix.

4. In implementation, in each iteration, we may detect if the $\alpha^{(i)}$ is over computed by checking if the infinity norm of the residual computed in step 6.7 is as small as expected in theory. If the $\alpha^{(i)}$ is over computed, we can reduce $\alpha^{(i)}$ to half and try. In practice, we haven't encountered this case.

5. This algorithm doesn't have to be completed. Just a few iterations can be used to achieve a desired accuracy. Since from the proof above we know, after i iterations, if we return the accumulated answer $\frac{1}{d^{(i)}}n^{(i)}$ as the solution, then the infinity norm of the absolute residual $||b - A \cdot \frac{1}{d^{(i)}}n^{(i)}||_\infty$ is less than or equal to $\frac{1}{\prod_{1 \le j \le i} \alpha^{(i)}}(2^{-i}||b||_\infty + ||A||_\infty)$, which implies that the absolute residual will decrease exponentially.

4.5.1.1 Total cost for well-conditioned matrices

In practice, integer matrices are often well-conditioned. If the matrix is well-conditioned, a backward stable (see [TB97] for details) factorization such as Gaussian elimination with (partial) pivoting or QR factorization is suitable to be used in this algorithm. For this case, the algorithm doesn't abort, but terminates with the correct rational solution. The entry of the amplified and adjusted residual in each iteration will be bounded by $||b||_\infty + ||A||_\infty$. In order to estimate the cost, we need to estimate $||\bar{x}^{(i)}||_\infty$. If the non-singular integer matrix A is well-conditioned (i.e., $||A||_\infty \cdot ||A^{-1}||_\infty$ is reasonable small), then $||A^{-1}||_\infty$ is small since $||A||_\infty$ is larger than or equal to 1. In each iteration,

$$||\bar{x}^{(i)}||_\infty \approx ||A^{-1}r^{(i-1)}||_\infty \le ||A^{-1}||_\infty \cdot ||r^{(i-1)}||_\infty = O(||r^{(i-1)}||_\infty).$$

So each iteration needs $\mathrm{O}^{\sim}(m^2)$ floating point operations and $\mathrm{O}^{\sim}(m^2(\log_2(\|A\|) + \log ||b||_\infty))$ machine word size integer operations. Now we estimate the number of iterations. We

know the Hadamard bound B and $\log B = \mathrm{O}^{\sim}(m \log_2(\|A\|))$. So the number of iterations required is $\mathrm{O}^{\sim}(m \log_2(\|A\|) + \log ||b||_\infty)$. Thus the total loop costs

$$\mathrm{O}^{\sim}(m^3(\log_2(\|A\|))(\log_2(\|A\|) + \log ||b||_\infty)).$$

The computation of A^{-1} costs $O(m^3)$ floating point operations. The computation of $n^{(k)}$ costs $\mathrm{O}^{\sim}(m^2(\log_2(\|A\|) + \log ||b||_\infty))$ bit operations by using divide-and-conquer method [CLRS01, Section 2.3.1] and FFT-based fast integer arithmetic. The final rational reconstruction in Theorem 4.4 will cost $\mathrm{O}^{\sim}(m^2(\log_2(\|A\|) + \log ||b||_\infty))$ bit operations. So the asymptotic cost for the worst case is

$$\mathrm{O}^{\sim}(m^3(\log_2(\|A\|))(\log_2(\|A\|) + \log ||b||_\infty)).$$

4.5.1.2 Experimentation on dense linear systems

The following table is the comparison of the running time of three different methods. All are implemented in C/C++. Plain_Dixon is an implementation of Dixon lifting without calling BLAS in LinBox*. Dixon_CRA_BLAS is an implementation by Z. Chen and A. Storjohann [CS04]. This method uses the idea of FFLAS [DGP02] and a mixture of Dixon lifting and the Chinese remainder algorithm. Dsolver is our implementation of algorithm 4.5, using LAPACK routines implemented in ATLAS†.

Table 4.1: Run time of different algorithms for exact solutions

order	100	200	300	400	500	600	700	800
Plain_Dixon	0.91	7.77	29.2	78.38	158.85	298.81	504.87	823.06
Dixon_CRA_BLAS	0.11	0.60	1.61	3.40	6.12	10.09	15.15	21.49
Dsolver	0.03	0.20	0.74	1.84	3.6	6.03	9.64	14.31

* LinBox is an exact computational linear algebra package under development, `www.linalg.org`

† ATLAS is a linear algebra software and provides C and Fortran77 interfaces to a portably efficient BLAS implementation, as well as a few routines from LAPACK and is available at `http://math-atlas.sourceforge.net`

In the table above, times are in seconds. Tests are sequentially run on a server with 3.2GHZ Intel Xeon processors and 6GB memory. All entries are integers randomly and independently chosen from $[-2^{20}, 2^{20}]$. Clearly, both Dsolver and Dixon_CRA_BLAS benefit from high performance BLAS routines implemented in ATLAS and are much faster than Plain_Dixon. Our algorithm is easier to be implemented than the idea used in Dixon_CRA_BLAS. Also, Dsolver is faster than Dixon_CRA_BLAS. The reason can be explained as follows. Dixon lifting and our method need near $\frac{2\log(\det(A))}{\log p}$ and $\frac{2\log(\det(A))}{\log \alpha}$ iterations, respectively, where p is the base of p-adic lifting and α is the geometric average of $\alpha^{(i)}$. For a well-conditioned matrix, the $\alpha^{(i)} \geq 2^{30}$ in each iteration. That is, in each iteration, Dsolver can get at least 30 binary leading bits of the exact solution of $Ax = r^{(i)}$. While in Dixon_CRA_BLAS, using Dixon lifting and FFLAS, each iteration in Dixon lifting can only get a p-adic digit, $p < 2^{27}$. This is because arithmetic operations such as multiplication in $\mathbb{Z}_p$ must be done exactly in double precision floating point arithmetic. So our method is expected to use fewer iterations. Also, our method can directly call BLAS routines without any extra cost for conversion between integer and floating point representation. Then it is not surprising that Dsolver is faster than Dixon_CRA_BLAS.

4.5.2 An exact rational solver for sparse integer linear systems

The main idea is to apply algorithm 4.5 by replacing the numerical solver with a successful sparse linear system solver. In the case of sparse matrices (including structured matrices and general sparse matrices), the algorithm can be especially fast if a sparse numerical solver works for the matrix at hand. Iterative methods such as Lanczos method, or Jacobi's method, the generalized minimum residual method (with restarts), conjugate gradient method, and bi-conjugate gradient method (- see e.g. [Saa03, TB97]]) are good candidates for numerically solving sparse linear systems. If the matrices are general sparse matrices with many zero entries, sparse elimination methods such as SuperLU are good candidates too. It is not our purpose to examine these issues, such as choice

of preconditioner, associated with sparse numerical solvers. But we point out that when the chosen numerical solver does succeed, then our method can get the exact solution extremely rapidly. In next section, our example from Trefethen's challenge suite shows a case where an iterative method works after applying a custom preconditioner adapted to the specific matrix.

4.6 An application to a challenge problem

We will demonstrate that our new method can run about 720 times faster than previous methods. In 2002, Prof. L. N. Trefethen posted "A Hundred-dollar, Hundred-digit Challenge" at SIAM News, Volume 35, Number 1. We are particularly interested in problem 7. Here is the original problem:

> Let A be the $20,000 \times 20,000$ matrix whose entries are zero everywhere except for the primes $2, 3, 5, 7, ..., 224737$ along the main diagonal and the number 1 in all the positions a_{ij} with $|i - j| = 1, 2, 4, 8, ..., 16384$. What is the $(1, 1)$ entry of A^{-1}?

Though only the first 10 decimal digits are required for the original problem, in 2002, an *exact*(symoblic) solution was computed by Jean-Guillaume Dumas of LMC-IMAG in Grenoble, France. It is a fraction with exactly 97,389 digits each for relatively prime numerator and denominator. He ran 182 processors for four days using Wiedemann's algorithm [Wie86] and the Chinese remainder algorithm - see [DTW02] for details. A few months later, we verified the result on one processor supporting 64-bit architecture with a large main memory(8GB) using p-adic lifting [Dix82] after explicitly computing the inverse of A mod a word size prime by Gaussian elimination and storing it as a dense matrix - see [DTW02] for details. Now, armed with the main idea presented here, the exact answer of this problem can be computed in 25 minutes with a PC with Linux, 1.9GHZ Pentium processor, 1GB memory, or in 13 minutes with a better machine with dual 3.2GHZ Intel Xeon processors and 6GB memory. And only a few MB of memory at run time are required. Our new method has won us an entry in the update followup of

a book [BLWW04] discussing the solutions to the challenge problems. This method is an application of our main idea with a custom approximate inverse of the input matrix. The general $n \times n$ matrix with the same pattern as A is a sparse matrix with $O(n \log n)$ non-zero entries and is almost a row diagonally dominant matrix except in the first few rows. For the special 20000×20000 matrix A, there are at most 29 non-zero entries in each row (column). So, if we represent the matrix as a block matrix:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where $A_{11}, A_{12}, A_{21}, A_{22}$ are 500×500, 500×19500, 19500×500, and 19500×19500 matrices, respectively. Then A_{22} is a *strongly diagonally dominant* matrix. Let D be the diagonal part of A_{22}, E be the rest. Therefore,

$$\begin{aligned} ||A \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I||_\infty &= ||(\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix} + \begin{bmatrix} 0 & A_{12} \\ 0 & E \end{bmatrix}) \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I||_\infty \\ &= || \begin{bmatrix} -A_{12}D^{-1}A_{21}A_{11}^{-1} & A_{12}D^{-1} - ED^{-1}A_{21}A_{11}^{-1} & ED^{-1} \end{bmatrix} ||_\infty \end{aligned}.$$

We know all diagonal entries of A_{22} are greater than 3571 (which is the 500th prime), all off-diagonal entries of A_{22} are 0 or 1, and there are at most 28 non-zero off-diagonal entries of A_{22} in each row or column, so $||ED^{-1}||_\infty \leq \frac{3571}{28} < \frac{1}{127.5}$. Also A_{12} and A_{21} are very sparse, and every diagonal entry of D is larger than 3571. Therefore it is reasonable to estimate that $||A \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I||_\infty$ is less than $\frac{1}{64}$, near twice of $\frac{1}{127.5}$, and follow-up computation shows that $\frac{1}{64}$ is not overestimated. So we can use $\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1}$ as an approximate inverse of A. Note that it is a lower triangle block matrix. Thus we can quickly solve

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix} x = y, \text{ for any } y.$$

We call LAPACK routines to compute the inverse of A_{11} explicitly, and store it as a dense matrix. Other parts can be stored as sparse matrices and be used as blackboxes. Here, in

order to simplify the explanation and estimate the condition number of A_{11}, an explicit inverse is computed instead of just storing its factorization, and the time computing the inverse is negligible, compared to the total run time for computing the symbolic answer. Computation shows that A_{11} is well-conditioned. In the experimentation, we solve $Ax = e_1 = (1, 0, \cdots, 0)$. With algorithm 4.5, We choose all scalars $\alpha^{(i)}$ equal to 64 and add a watchdog for the residual, that is, if the norm of the residual is not smaller than the theoretical result in each iteration, it would abort with an error message. Only digits for x_1 are stored and only x_1 is reconstructed. This method is asymptotically faster than previous methods, and requires a chunk of memory almost as small as when totally treating the matrix as a blackbox. Here is a brief summary of the three different successful approaches above.

Table 4.2: Comparison of different methods for solving a challenge problem

Methods	Complexity	Memory	Run time
Quotient of two determinants Wiedemann's algorithm The Chinese remainder algorithm	O($n^4 \log^2 n$)	a few MB	Four days in parallel using 182 processors, 96 Intel 735 MHZ PIII, 6 1GZ 20 4 $\times$ 250MHZ sun ultra-450
Solve $Ax = e_1 = (1, 0, \cdot, 0)$ by plain Dixon lifting for the dense case Rational reconstruction	O($n^3 \log n$)	3.2 GB	12.5 days sequentially in a Sun Sun-Fire with 750 MHZ Ultrasparcs and 8GB for each processor
Solve $Ax = e_1 = (1, 0, \cdot, 0)$ by our methods above Rational reconstruction	O($n^2 \log^2 n$)	a few MB	25 minutes in a pc with 1.9GHZ Intel P4 processor, and 1 GB memory

Clearly our new method is quite efficient in memory and computing time.

4.7 An adaptive algorithm

Success of our hybrid algorithm requires the system to be sufficiently well-conditioned for the numerical linear algebra method being used to compute a solution with sufficient

accuracy. It aborts quickly on ill-conditioned cases. p-adic lifting [Dix82, MC79] works on both well-conditioned and ill-conditioned cases. In practice, it runs slower than our hybrid algorithm. An adaptive algorithm can be formed easily, starting with our hybrid algorithm and switching to p-adic lifting if it aborts. This adaptive algorithm is implemented in the LinBox library.

Chapter 5

COMPUTATION OF SMITH FORMS OF INTEGER MATRICES

5.1 Introduction

Smith normal form (Smith form, in short) plays an important role in the study of algebraic group theory, homology group theory, systems theory, matrix equivalence, Diophantine systems, theory of control. In 1861, H. J. Smith introduced the Smith normal form. Since then, the Smith form problem has attracted the attention of many researchers due to its wide application. Many algorithms have been discovered and algorithmic complexity has been improved. With computer power nowadays, the Smith form of dimension one thousand can be computed in a few hours. One of our main contributions to Smith form computation is that we discovered an engineered algorithm for Smith normal form which yields very good practical run time and asymptotic complexity. Asymptotically fast algorithms may run slowly in practice. There are choices of algorithms and each is favored by certain cases. Moreover, different parts of Smith form are favored by different algorithms. Based on both theoretical and experimental results, we propose an "engineered" Smith form algorithm. Another contribution of ours is that we have improved the probability bound of an important Monte-Carlo algorithm. Also, we have discovered an efficient preconditioner for computing the Smith normal form.

This chapter will start with a history of Smith form algorithms. Algorithms in [KB79, Ili89, HM91, Sto96, Gie96, EGV00, KV03] will be discussed. Then in section 5.6, we will present our algorithm for dense matrix Smith form, which primarily takes the form of a suite of improvements to the probabilistic algorithm in [EGV00]. Finally, an "engineered" Smith form algorithm will be introduced.

5.2 The definition of Smith form

The following definition of Smith form is based on the existence proof by H. Smith in [Smi61].

DEFINITION 5.1. For every integer matrix A, there exist unimodular matrices U and V, and a diagonal matrix S with non-negative diagonal entries s_i, for $i = 1, \cdots, n.$ such that $A = USV$, and $s_i | s_{i+i}$. The matrix S is called the Smith Normal Form or simply Smith Form of A, the s_i are called the invariant factors of A. Here an integer matrix is called unimodular if and only if its determinant is 1 or -1.

REMARK. Note that the choices of U and V are not unique, though the Smith Normal Form S of an integer matrix A is unique.

The following is a simple fact about Smith form.

FACT 5.2. If A is a given $n \times n$ integer matrix with invariant factors $(s_1, ..., \ldots, s_n)$, then

$$\det(A) = \pm \prod_{0 \le i \le n} s_i, \text{ and } s_i = 0, \text{ if } i > \text{rank}(A).$$

For an integer matrix A, we define $s_i(A)$ to be the i^{th} invariant factor of A. In spite of the abstract definition of Smith form, the Smith form can be computed via a series of elementary column and row operations consisting of (see e.g. [New72]):

1. Adding an integer multiple of one row (column) to another.
2. Multiplying a row (column) by -1.
3. Swapping two rows (columns).

For example, a 2×2 matrix:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}.$$

We can perform a series of elementary row and column operations below to transform A into its Smith form.

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix} \xrightarrow{2^{nd} \text{ col - } 1^{st} \text{ col}} \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \xrightarrow{2^{nd} \text{ row - } 1^{st} \text{ row}} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}.$$

$1, 2$ are the invariant factors of A. In this example, $\det(A) = 1 * 3 - 1 * 1 = 2 = 1 * 2$.

The following theorem gives an upper bound of the determinant.

THEOREM 5.3. *Hadamard's inequality*

Let $A \in \mathbb{R}^{n \times n}$, and $B \in \mathbb{R}$ such that all entries of A are at most B in absolute value. Then

$$|\det(A)| \leq \sqrt{\prod_{i=1}^{n} (\sum_{j=1}^{n} A_{ij}^2)} \leq n^{n/2} B^n.$$

Proof. See e.g. [vzGG99, THEOREM 16.6]. □

In the worst case, the determinant is equal to the Hadamard bound. Since the determinant is equal to the product of all invariant factors up to a sign, all invariant factors are bounded by the Hadamard bound. In the worst case, the largest is equal to the Hadamard bound. Therefore, the largest invariant factor contains $\tilde{O}(n \log_2(\|A\|))$ bits in the worst case. This large number of bits in invariant factors contributes difficulty to the Smith form computation. Let us look a random example: a random 5×5 integer matrix generated in Maple9:

$$\begin{bmatrix} -66 & -65 & 20 & -90 & 30 \\ 55 & 5 & -7 & -21 & 62 \\ 68 & 66 & 16 & -56 & -79 \\ 26 & -36 & -34 & -8 & -71 \\ 13 & -41 & -62 & -50 & 28 \end{bmatrix}.$$

Its Smith form S by SmithForm in LinearAlgebra package in Maple9 is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4820471082 \end{bmatrix}.$$

Though the random matrix itself has order 5 and small entries, its Smith form has a large entry.

In the following, a number of Smith form algorithms will be discussed. The following notations will be used. We use A_i to denote the i^{th} column of a matrix A and use COL(i)(ROW(i)) to denote the ith column(row) of matrix.

5.3 Kannan and Bachem's algorithm

As pointed out by Frumkin in [Fru77], none of the algorithms given before 1979 for computing the Smith Normal Form of an integer matrix is known to be a polynomial time algorithm. In 1997, X. Fang and G. Havas [FH97] showed there is an elimination algorithm for the Smith form of an integer matrix with an exponential time complexity and exponential space required in a worst case.

In 1979, R. Kannan and A. Bachem [KB79] proved that there was a polynomial time algorithm for the Smith form of an integer matrix. Their method successively controls the size of the entries of the integer matrix after each elimination step by using the extended greatest common divisor and special elimination.

Here I will explain the algorithm in non-singular cases with every leading principal minor nonsingular. The general case can be transformed into this case by matrix permutation and a standard row reduction technique - see [KB79].

The essential part of Kannan and Bachem's method is a special extended gcd-based row (column) elimination, which eliminates an off-diagonal entry in a column (row).

SUBPROCEDURE 1. Column Elimination Step[used in Kannan and Bachem's algorithm]

Input:

- A, an $n \times n$ integer matrix
- i, an integer, $1 \leq i \leq n$
- j, an integer, $1 \leq j \leq n$, $j \neq i$

Output:

- A, modified but equivalent to input A, with $A_{ij} = 0$.

Procedure:

1. Use extended GCD[1] algorithm to compute g, p and q, such that

$$g = \gcd(A_{ii}, A_{ij}), g = p \cdot A_{ii} + q\dot{A}_{ij} \text{ and } |p|, |q| \leq max(A_{ii}, A_{ij}).$$

2. Perform the elementary column operations on A so that A_{ij} becomes zero by replacing i^{th} and j^{th} columns of A by the two columns of the product

$$\begin{bmatrix} A_i & A_j \end{bmatrix} \begin{bmatrix} p & -A_{ij}/g \\ q & A_{ii}/g \end{bmatrix}.$$

Note that

$$\det(\begin{bmatrix} p & -A_{ij}/g \\ q & A_{ii}/g \end{bmatrix}) = 1.$$

A similar procedure can perform a row elimination operation. The following is Kannan and Bachem's algorithm for the Smith form.

ALGORITHM 1. Kannan and Bachem's algorithm

Input:

[1] short for Greatest Common Divisor

- A, an $n \times n$ non-singular integer matrix with every leading principal minor non-singular.

Output:

- The Smith form of input matrix A.

Procedure:

1. for $i = 1$ to n do Repeat

   (a) [Eliminate all off-diagonal entries in ith column] for $j = i + 1$ to n do Row Elimination(A,i, j)

   (b) [Transform the bottom-right submatrix consisting of the last $(n + 1 - i)$ rows and columns into a lower triangular matrix]
   for $j = i$ to $n - 1$ do

      i. for $k = i$ to j do

         - Column Elimination(A, k, $j + 1$).
         - If $A_{kk} < 0$ then set $A_k = -A_k$.
           For $z = 1$ to $k - 1$ do set $A_z = A_z - \lceil A_{kz}/A_{kk} \rceil A_k$.

      ii. If $A_{i+1,i+1} < 0$ then set $A_{i+1} = -A_{i+1}$.
      For $z = 1$ to i do set $A_z = A_z - \lceil A_{i+1,z}/A_{i+1,i+1} \rceil A_{i+1}$.

   Until A_{ii} is the only non-zero entry in i^{th} column of A and A_{ii} divides every element A_{jk}, where $i \leq j \leq n$ and $i \leq k \leq n$

2. Return A.

A typical step of the algorithm can be demonstrated as follows: a 7×7 matrix, and $i = 3$ case,

$$\xrightarrow{i=3 \text{ step}} \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & 0 & 0 & 0 \\ 0 & 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & * & 0 \\ 0 & 0 & * & * & * & * & * \end{bmatrix} \xrightarrow{\text{row eliminations}} \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \end{bmatrix}$$

$$\xrightarrow{\text{col eliminations}} \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & 0 & 0 & 0 \\ 0 & 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & * & 0 \\ 0 & 0 & * & * & * & * & * \end{bmatrix} \xrightarrow{\text{row eliminations}} \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \end{bmatrix}$$

$$\longrightarrow \cdots \longrightarrow \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & * & * & * & * \end{bmatrix} \xrightarrow{i=4 \text{ step}}$$

The algorithm 1 successively controls the maximal bit length of intermediate results during elimination, which has been shown to be polynomial. Based on that, they have proven the following theorem - see [KB79, Theorem 4].

THEOREM 5.4. *The algorithm 1 above correctly computes the answer and is polynomial.*

In 1982, T. Chou and G. Collins in [CC82] showed that one can obtain an even better bound by changing the normalization order of Kannan and Bachem's algorithm from top to bottom and left to right to right to left and top to bottom. Interestingly, E. Kaltofen, M. Krishnamoorthy and D. Saunders in [KKS90] described a parallel algorithm for the Smith form based on Kannan and Bachem's algorithm.

5.4 Iliopoulos' algorithm

In 1989, C. Iliopoulos [Ili89] discovered that Smith form can be computed via the determinant. At that time, it was known that the determinant could be found by computing mod many different primes and can run in $\mathrm{O}^{\sim}(n^4 \log_2(\|A\|))$ bit operations. For recent results about determinant computation, see e.g. [KV03]. Via mod the determinant, Ililoupos in his paper proved that the Smith form can be computed in $\mathrm{O}^{\sim}(n^5(\log \|A\|)^2)$ bit operations. I will use a non-singular case to demonstrate Iliopoulos' idea. C. Iliopoulos has extended this idea to general cases.

For a given $n \times n$ non-singular integer matrix A, Iliopoulos' algorithm firstly computes d, the determinant of A, then computes the Smith form mod d, i.e computes the Smith form of

$$\begin{bmatrix} A \\ dI_n \end{bmatrix}.$$

Note that every invariant factor of A divides d from fact 5.2. Iliopoulos' algorithm for the Smith form is based on two subroutines, ELIMINATEROW and ELIMINATECOL.

SUBPROCEDURE 2. ELIMINATEROW [Eliminate all off diagonal entries in a row]
Input:

- A, an $n \times n$ integer matrix
- d, the modulus

- ρ, a row index, and $1 \leq \rho \leq n$.

Output:

- A, modified but equivalent, with properties, $A_{\rho i} = 0$, for $\rho < i \leq n$.

Procedure:

1. [Make $A_{\rho\rho} = 0$] If $A_{\rho\rho} \neq 0$ then

   - compute r, x_1, x_2, such that

   $$r = \gcd(A_{\rho\rho}, A_{\rho,\rho+1}), r = x_1\, A_{\rho\rho} + x_2\, A_{\rho,\rho+1}, |x_1| \leq |a_{\rho,\rho+1}|/2,$$

   $$\text{and } |x_2| \leq |a_{\rho\rho}|/2.$$

   - Update the $\rho^{th} and (\rho+1)^{th}$ columns to become

   $$\begin{bmatrix} A_\rho & A_{(\rho+1)} \end{bmatrix} \begin{bmatrix} a_{\rho,\rho+1}/r & x_1 \\ -a_{rho\rho}/r & x_2 \end{bmatrix}.$$

2. compute s, and t_j, $\rho + 1 \leq j \leq n$ such that

$$s = \gcd(A_{\rho,\rho+1}, \cdots, A_{\rho n}), \text{ and } s = \sum_{j=\rho+1}^{n} t_j A_{\rho j}.$$

3. set

$$t_j := t_j \pmod{d}, \text{ for } \rho + 1 \leq j \leq n.$$

4. update

$$\text{COL}(\rho) \leftarrow \text{COL}(\rho) + \sum_{j=\rho+1}^{n} t_i \text{COL}(j).$$

5. For $\rho + 1 \leq j \leq n$, update

$$\text{COL}(j) \leftarrow \text{COL}(j) - (A_{\rho j}/s)\text{COL}(\rho).$$

6. Normalize $A_{j\rho}$ by mod d, for $\rho + 1 \leq j \leq n$

REMARK. This subroutine eliminates off diagonal entries in a row. The transformation is equivalent to multiplying $\begin{bmatrix} \text{A} \\ dI_n \end{bmatrix}$ by a unimodular integer matrix on the left.

There is a similar subroutine ELIMINATECOL, which eliminates off diagonal entries in a column.

ALGORITHM 2. Iliopoulos' algorithm

Input:

- A, an $n \times n$ non-singular matrix.

Output:

- The Smith form of A.

Procedure:

1. [Compute the determinant] $d \leftarrow$ the determinant of A.

2. [Transform A to an equivalent diagonal matrix] For $\rho = 1$ to n do Repeat[Finding a right pivoting]

   - ELIMINATEROW (A, d, ρ)

   - ELIMINATECOL (A, d, ρ)

   Until

   $$a_{\rho\rho} \mid a_{\rho j}, \text{ for all } \rho + 1 \leq j \leq n.$$

3. [Compute the Smith form of the diagonal matrix] for $p = 1$ to n do for $q = p + 1$ to n do

   - $h \leftarrow A_{pp}$

- $a_{pp} \leftarrow GCD(A_{pp}, h)$

- $a_{qq} \leftarrow A_{qq}h/a_{pp}$

4. Return A.

REMARK. The Smith form of a diagonal matrix can be efficiently computed via a special sorting network - see section 5.7.1.

Iliopoulos' algorithm above uses the determinant to control the maximal bit length of entries of intermediate results during elimination.

THEOREM 5.5. *For an $n \times n$ non-singular matrix A, the algorithm 2 above correctly computes its Smith form and requires* $\mathrm{O}^{\sim}(n^5(\log_2(\|A\|))^2)$ *bit operations.*

REMARK. The exponent 5 in the complexity can be explained as follows: 3 comes from the exponent part in Gaussian elimination, 1 comes from the loop in order to find the right pivoting, and another 1 comes from the bit length of the modulus. Thus, an elimination-based optimal algorithm will require $\mathrm{O}^{\sim}(n^4 \log_2(\|A\|))$ bit operations to compute the Smith form of an $n \times n$ integer matrix.

Proof. See [Ili89, corollary 5.3]. □

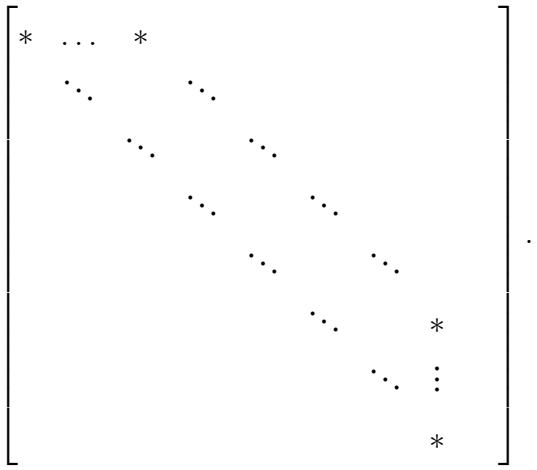
In 1991, J. Hafner and K. McCurley [HM91] independently discovered an idea similar to Iliopoulos' above for the Smith form computation.

5.5 Storjohann's near optimal algorithm

In 1996, A. Storjohann [Sto96] improved the complexity substantially. He showed that the Smith form can be computed in $\mathrm{O}^{\sim}(n^{(\omega+1)} \log_2(\|A\|))$ bit operations. This algorithm computes the Smith form modulo the determinant by using matrix multiplication.

The important step of computing Smith form via elimination is the choice of U and V. This algorithm emphasizes the choice of U and V. The essential part of this method is banded matrix reduction.

A square matrix A is called upper b-banded if $A_{ij} = 0$ for $j < i$ or $j \geq i + b$. A banded matrix visually looks like:



Storjohann showed that an upper b-banded matrix can be reduced to an upper $(\lfloor b/2 \rfloor + 1)$ -banded matrix in O$(n^2 b^{(\omega-2)})$ ring operations.

We use $sub_A[i, k]$ to denote the $k \times k$ principal submatrix of A consisting of rows and columns $i + 1, i + 2, \cdots, i + k$.

SUBPROCEDURE 3. Band Reduction

Input:

- A, an upper b-banded matrix in $\mathbb{Z}_d^{n \times n}$ with $b > 2$.

Output:

- An upper $(\lceil b/2 \rceil + 1)$-banded matrix, which is unimodularly equivalent to A.

Procedure:

1. [Initialize]

$$s_1 \leftarrow \lfloor b/2 \rfloor$$

$$n_1 \leftarrow \lfloor b/2 \rfloor + b + 1$$

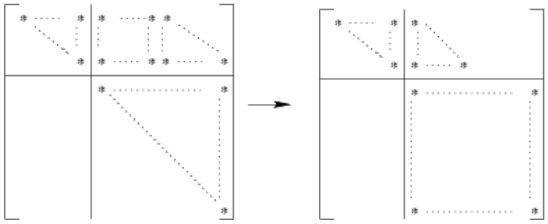
$$s_2 \leftarrow b - 1$$

$n_2 \leftarrow 2(b-1)$

2. [Apply a series of equivalence transformations]

   for $i = 0$ to $\lceil n/s_1 \rceil - 1$ do

   - Apply transformation in $sub_A[is_1, n_1]$:



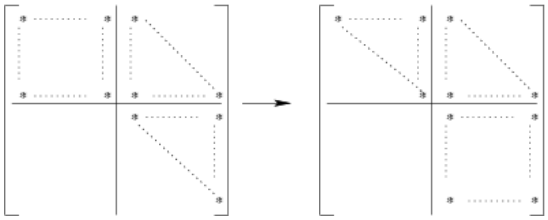
   By setting

$$sub_A[is_1, n_1] \leftarrow sub_A[is_1, n_1] \begin{bmatrix} I_{s_1} & 0 \\ 0 & U \end{bmatrix},$$

   where U is a special unimodular matrix over ring $\mathbb{Z}_d$, which upon post-multiplication, triangulates the $s_1 \times s_2$ upper hand block of $sub_A[is_1, n_1]$ into a lower triangular matrix.

   - for $j = 0$ to $\lceil (n - (i+1)s_1)/s_2 \rceil$

     Apply transformation in $sub_A[(i+1)s_1 + js_2, n_2]$:



     By setting

$$sub_A[(i+1)s_1 + js_2, n_2] \leftarrow \begin{bmatrix} U_1 & 0 \\ 0 & I_{s_2} \end{bmatrix} sub_A[(i+1)s_1 + js_2, n_2] \begin{bmatrix} I_{s_2} & 0 \\ 0 & U_2 \end{bmatrix},$$

where U_1, U_2 is an unimodular matrix over $\mathbb{Z}_d$, such that $C_1^T U_1^T$ is a lower triangular matrix and $(U_1 C_2)U_2$ is a lower triangular matrix, where

$$sub_A[(i+1)s_1 + js_2, n_2] = \begin{bmatrix} C_1 & C_2 \\ 0 & C_3 \end{bmatrix}.$$

3. Return A.

Please see [Sto96] for details.

By a sequence of banded matrix reductions, a banded matrix can be transformed into an equivalent bidiagonal matrix. The Smith form of an $n \times n$ bidiagonal matrix over a modular ring can be computed in $O(n^2)$ ring operations by a special elimination - see [Sto96, section 3.2]. Storjohann's algorithm computes the Smith form by first computing the determinant d, then transforming A to an upper triangular matrix over $\mathbb{Z}_d$, which is an upper n-banded matrix, after that transforming it to a bidiagonal matrix over $\mathbb{Z}_d$, and finally computing its Smith form.

ALGORITHM 3. Storjohann's algorithm

Input:

- A, an $n \times n$ integer matrix

Output:

- The Smith form of A

Procedure:

1. [Compute the determinant of a nonzero maximal rank submatrix of A]

   $d \leftarrow$ the determinant of a nonzero maximal rank submatrix of A.

2. [Compute the Smith form modulo d]

3. Compute a unimodular matrix U over $\mathbb{Z}_d$, such that $U^T \times A^T$ is a lower triangular matrix over $\mathbb{Z}_d$.

4. Set $A \leftarrow A \times U$ [Note that A is n-banded matrix now]

5. $b \leftarrow n$

6. while $b > 2$ do [Transform A into a bidiagonal matrix]

$$A \leftarrow \text{ Band Reduction}(A, b),$$

and

$$b \leftarrow \lceil b/2 + 1 \rceil.$$

7. [Compute the Smith form of Bidiagonal matrix over $\mathbb{Z}_d$.]

$$A \leftarrow \text{ the Smith Normal Form of } A.$$

8. Return A.

THEOREM 5.6. *For an $n \times n$ integer matrix, algorithm 3 above correctly computes its Smith form and requires $\mathrm{O}^{\sim}(n^4 \log_2(\|A\|))$ bit operations.*

Proof. See [Sto96, Theorem 14]. □

5.6 Eberly, Giesbrecht, and Villard's algorithm with improvements

All algorithms I discussed above are deterministic. Storjohann's algorithm 3 is the near optimal deterministic algorithm for Smith form. One question is whether the algorithmic complexity can be improved if randomization is allowed. The answer is *yes*. In the coming few sections, we will discuss a few probabilistic algorithms.

In 2000, W. Eberly, M. Giesbrecht and G. Villard [EGV00] discovered a new approach to compute the Smith form of an $n \times n$ integer matrix A in $\mathrm{O}^{\sim}(n^{3.5} \log \|A\|)$ or $\mathrm{O}^{\sim}(n^{2+\omega/2} \log_2(\|A\|))$ when fast matrix multiplication is available. The largest invariant

factor of a non-singular integer matrix can be computed by symbolically solving a few linear systems. the invariant factor of any index can be computed by means of random perturbations. A binary search can be used to find all distinct invariant factors.

5.6.1 Largest invariant factor with a "bonus" idea

This algorithm is a variation of the algorithm [EGV00, LargestInvariantFactor]. Our contribution is the "bonus" idea which can be used to find the second largest invariant factor with very little extra work.

ALGORITHM 4. LargestInvariantFactor
Input:

- A, a non-singular $n \times n$ integer matrix.
- ϵ, an error probability requirement.

Output:

- s_n, the largest invariant factor of A with probability at least $1 - \epsilon$.
- s_{n-1}, the second largest invariant factor of A.

Procedure:

1. $M := 6 + 2n(\log_2 n + \log_2(\|A\|)); \mathcal{L} := \{0, \ldots, M-1\}$.
2. Set $s_n^{(0)} := 1$; $s_{n-1}^{(0)} := 0$.
3. Set $N := \lceil \log(1/\epsilon)/\log 1.5 \rceil$.
4. For i from 1 to N do
   a. Choose random vectors, $\vec{b}^{(2i-1)}$, and $\vec{b}^{(2i)} \in \mathcal{L}^n$.
   b. Solve $A\vec{x}^{(2i-1)} = \vec{b}^{(2i-1)}$ and $A\vec{x}^{(2i)} = \vec{b}^{(2i)}$ over the rational field.

c. Compute

$$t^{(2i-1)} := \text{lcm}(\text{denom}(\vec{x}_1^{(2i-1)}), \ldots, \text{denom}(\vec{x}_n^{(2i-1)}))$$

and

$$t^{(2i)} := \text{lcm}(\text{denom}(\vec{x}_1^{(2i)}), \ldots, \text{denom}(\vec{x}_n^{(2i)})).$$

d. Compute $u^{(i)} = \frac{\gcd(t^{(2i-1)}, t^{(2i)})}{s_2(\left[t^{(2i-1)}\vec{x}^{(2i-1)} \quad t^{(2i)}\vec{x}^{(2i)}\right])}$. [$\left[t^{(2i-1)}\vec{x}^{(2i-1)} \quad t^{(2i)}\vec{x}^{(2i)}\right]$ is an $n \times 2$ integer matrix.]

e. Compute $s_n^{(i)} := \text{lcm}(s_n^{(i-1)}, t^{(2i-1)}, t^{(2i)})$.

f. Compute $s_{n-1}^{(i)} = \gcd(s_{n-1}^{(i-1)}, u^{(i)})$.

5. $s_n^{(N)}$ and $s_{n-1}^{(N)}$.

REMARK. In addition to the largest invariant factor, algorithm 4 also computes the second largest invariant with expected extra $\text{O}^{\sim}(n^2 \log_2(\|A\|))$ bit operations. The extra cost is negligible, comparing to the total cost. This is the reason we call it "bonus". This is important in practice, since big prime factors often occur only in the largest invariant factor. For example, the random matrices have trivial invariant factors, i.e. all invariant factors are 1 except the largest invariant. Also the (second) largest invariant factor can be used to improve Iliopoulos' algorithm and Storjohann's, by replacing determinant with (second) largest invariant factor. In practice, the largest invariant factor is often smaller than the determinant, and the second largest is smaller still.

THEOREM 5.7. *Given an $n \times n$ non-singular integer matrix A and an error bound ϵ, algorithm 4 above computes the largest invariant factor of A with probability at least $1 - \epsilon$ and the second largest factor of A with probability near $1 - \sqrt{\epsilon}$. The algorithm is expected to run in*

$$\text{O}^{\sim}(n^3 \log_2(\|A\|) \log(1/\epsilon))$$

bit operations.

Proof. By [EGV00, Theorem 2.1], we know in each iteration i, $t^{(2i-1)}$ and $t^{(2i)}$ are always factors of s_n and

$$\text{Prob}(\text{lcm}(t^{(2i-1)}, t^{(2i)}) \neq s_n) < 2/3.$$

After repeating N times, the error probability is at most ϵ.

The reason that $s_{n-1}^{(N)}$ is equal to the second largest invariant factor can be explained below. It is known that $s_n(A)A^{-1}$ is an integer matrix, and the Smith form of $s_n(A)A^{-1}$ is

$$\text{diag}(\frac{s_n(A)}{s_n(A)}, \frac{s_n(A)}{s_{n-1}(A)}, \cdots, \frac{s_n(A)}{s_1(A)}).$$

So $S_{n-1}(A) = \frac{s_n(A)}{s_2(s_n(A)A^{-1})}$. If we group the rational vectors $\vec{b}$ and solutions $\vec{x}$ in the loop into pairs,

$$A\vec{x}^{(2i-1)} = \vec{b}^{(2i-1)} \text{ and } A\vec{x}^{(2i)} = \vec{b}^{(2i)},$$

then

$$s_n(A)A^{-1}\left[\vec{b}^{(2i-1)}, \vec{b}^{(2i)}\right] = \left[s_n(A)\vec{x}^{(2i-1)}, s_n(A)\vec{x}^{(2i)}\right].$$

Thus

$$s_{n-1}(A) = \frac{s_n(A)}{s_2([s_n(A)\vec{x}^{(2i-1)}, s_n(A)\vec{x}^{(2i)}])} = \frac{\gcd(t^{(2i-1)}, t^{(2i)})}{s_2([t^{(2i-1)}\vec{x}^{(2i-1)}, t^{(2i)}\vec{x}^{(2i)}])}.$$

For any prime p, following from lemma 5.17, an approach like in theorem 5.14 shows that

$$\text{Prob}(p \nmid \frac{s_{n-1}(A)}{s_2([s_n(A)\vec{x}^{(2i-1)}, s_n(A)\vec{x}^{(2i)}])}) \geq (1 - \frac{1}{M}\lceil\frac{M}{p}\rceil)(1 - (\frac{1}{M}\lceil\frac{M}{p}\rceil)^2).$$

So Prob$(s_{n-1}^{(N)} \neq s_{n-1}(A))$ is near $\sqrt{\epsilon}$. Moreover the computed $s_{n-1}(A)$ is always a factor of the true answer.

Each iteration requires $\tilde{\text{O}}(n^3 \log_2(\|A\|))$ bit operations. So the number of expected bit operations for the algorithm is $\tilde{\text{O}}(n^3 \log_2(\|A\|) \log(1/\epsilon))$. □

5.6.2 The invariant factor of any index with a new efficient perturbation

Algorithm [EGV00, Algorithm:OneInvariantFactor] can be used to obtain the invariant factor of any index by means of perturbation. For $n \times i$ random matrix U and

$i \times n$ random matrix V, the largest invariant factor of $A + UV$ is likely to be the ith invariant factor of A. We propose a more efficient perturbation. Let L' and R' be random matrices with each entry randomly and independently chosen from a set of integers. Then the largest invariant factor of

$$\begin{bmatrix} I_i & L' \end{bmatrix} A \begin{bmatrix} I_i \\ R' \end{bmatrix}$$

is likely to be the ith invariant factor of A. Also the "bonus" idea in algorithm 4 can be extended here to compute the preceding invariant factor.

ALGORITHM 5. OneInvariantFactor

Input:

- A, an $n \times m$ integer matrix, w.l.g. $m \leq n$.
- i, an integer, $2 \leq i \leq$ rank(A).
- ϵ, an error probability requirement.

Output:

- s_i, the ith invariant factor of A with probability at least $1 - \epsilon$.
- s_{i-1}, the $(i-1)$th invariant factor of A.

Procedure:

1. Set $s_i^{(0)} := 0$.
2. Set $s_{i-1}^{(0)} := 0$.
3. Set $S :=$ $210^{\lceil \frac{i(6 \log n + 3 \log_2(\|A\|))}{210} \rceil}$; $\mathcal{L} = [0, \ldots, S-1]$.
4. Set $N := \log(10/\epsilon)/\log(24/23)$;
5. for $j = 1$ to N do

a. Choose random integer matrix $L \in \mathcal{L}^{i\times(n-i)}$.

b. Choose random integer matrix $R \in \mathcal{L}^{(m-i)\times i}$.

c. Compute $A' = \begin{bmatrix} I_i & L \end{bmatrix} A \begin{bmatrix} I_i \\ R \end{bmatrix}$.

d. Compute $(t^{(j)}, u^{(j)})$, the largest and the second largest invariant factors of A', by algorithm 4.

e. Compute $s_i^{(j)} := \gcd(s_i^{(j-1)}, t^{(j)})$; $s_{i-1}^{(j)} := \gcd(s_{i-1}^{(j-1)}, u^{(j)})$.

6. return $s_i^{(N)}$, and $s_{i-1}^{(N)}$.

REMARK. The algorithm is similar to algorithm in [EGV00, section 3]. But only one random set is used here and more efficient preconditioners are used. Also we are going to prove a better theoretical probability bound.

THEOREM 5.8. *For a given $n \times n$ integer matrix and $2 \le i \le rank(A)$, the algorithm 5 above computes the ith invariant factor of A with probability at least $1 - \epsilon$ and the $(i-1)$th invariant factor with probability near $1 - \sqrt{\epsilon}$. The algorithm is expected to run in*

$$\mathrm{O}^{\sim}(in^2 \log_2(\|A\|) \log^2(1/\epsilon))$$

bit operations.

We need a few lemmas and theorems described below in order to prove this theorem.

5.6.2.1 Non-uniformly distributed random variables

In this subsection, we focus on non-uniformly distributed random variables over a finite field $\mathbb{F}$ with additional bound condition,

$$\alpha \le \mathrm{Prob}(x = d) \le \beta, \forall d \in \mathbb{F}.$$

If $\alpha = \beta = \frac{1}{p}$, then x is a uniformly distributed random variable. The next lemma describes an example.

LEMMA 5.9. *If an integer x which is uniformly randomly chosen from $[0, \ldots, S-1]$, is mapped to a finite field $\mathbb{Z}_p$, then*

$$\frac{1}{S}\lfloor\frac{S}{p}\rfloor \leq \text{Prob}(x = d \mod p) \leq \frac{1}{S}\lceil\frac{S}{p}\rceil$$

for any $d \in \mathbb{Z}_p$. Thus $x \mod p$ is a non-uniformly distributed random variable over $\mathbb{Z}_p$.

Proof. Given an integer d and prime p, there is at most $\lceil\frac{S}{p}\rceil$ integers x such that,

$$0 \leq x \leq S-1, \text{ and } x = d \pmod p.$$

So

$$\text{Prob}(x = d \mod p) \leq \frac{1}{S}\lceil\frac{S}{p}\rceil.$$

By similarly reasoning,

$$\text{Prob}(x = d \mod p) \leq \frac{1}{S}\lfloor\frac{S}{p}\rfloor.$$

□

The next lemma characterizes the distribution of dot products for a random vector.

LEMMA 5.10. *Given a non-zero vector $\vec{t} \in \mathbb{F}$ and an element $d \in \mathbb{F}$, if $\vec{x} = (x_1, \ldots, x_n)$ is a random vector in $\mathbb{F}^n$ with each entry independently and non-uniformly randomly chosen from $\mathbb{F}$ with the probability that any element in $\mathbb{F}$ ranging from α to β is chosen, then*

$$\alpha \leq \text{Prob}(\vec{x} : \vec{t} \cdot \vec{x} = d) \leq \beta.$$

Proof. $\text{Prob}(\vec{x} : \vec{t} \cdot \vec{x} = d) \leq \beta.$ can be proved by using the variation of Schwartz-Zippel lemma in [Gie01, lemma]. Also it can be argued by a simple idea, which can be used to prove the inequality in the left side.

Since $\vec{t} = (t_1, \ldots, t_n) \neq 0$, without loss of generality we assume $t_1 \neq 0$.

$$
\begin{aligned}
&\text{Prob}(\vec{x} : \vec{t} \cdot \vec{x} = d) \\
&= \textstyle\sum_{j \in \mathbb{F}} \text{Prob}((x_2, \ldots, x_n) : \sum_{i=2}^{n} t_i x_i = j)\text{Prob}(x_1 : x_1 = \frac{d-j}{t_1} | \sum_{i=2}^{n} t_i x_i = j) \\
&= \textstyle\sum_{j \in \mathbb{F}} \text{Prob}((x_2, \ldots, x_n) : \sum_{i=2}^{n} t_i x_i = j)\text{Prob}(x_1 : x_1 = \frac{d-j}{t_1}) \\
&\leq \textstyle\sum_{j \in \mathbb{F}} \beta\text{Prob}((x_2, \ldots, x_n) : \sum_{i=2}^{n} t_i x_i = j) \\
&= \beta.
\end{aligned}
$$

By similarly reasoning, we know $\alpha \leq \text{Prob}(\vec{x} : \vec{t} \cdot \vec{x} = d)$. □

The lemma below characterizes the distribution of a random null space vector.

LEMMA 5.11. *Given a matrix $A \in \mathbb{F}^{n \times m}$ with rank r, if $\vec{x} = (x_1, \ldots, x_m)^T$ is a random vector in $\mathbb{F}$ with each entry independently and non-uniformly randomly chosen from $\mathbb{F}$ with the probability that any element in $\mathbb{F}$ ranging from α to β is chosen, then*

$$\alpha^r \leq \text{Prob}(\vec{x} : Ax = 0) \leq \beta^r.$$

Proof. We know $\exists L, U, P$, such that $A = LUP$, P is permutation matrix, L is non-singular, and U has the shape

$$\begin{bmatrix} I_r & U' \\ 0 & 0 \end{bmatrix}.$$

Thus,

$$
\begin{aligned}
&\text{Prob}(\vec{x} : Ax = 0) \\
&= \text{Prob}(\vec{x} : LUPx = 0) \\
&= \text{Prob}(\vec{x} : LUPx = 0) \\
&= \text{Prob}(\vec{x} : Ux = 0) \\
&= \text{Prob}((x_1, \ldots, x_r)^T = -U'(x_{r+1}, \ldots, x_n)^T) \\
&= \textstyle\prod_{j=1}^{r} \text{Prob}(x_j = -U_j'\dot{(}x_{r+1}, \ldots, x_n)^T | x_k = -U_k'\dot{(}x_{r+1}, \ldots, x_n)^T, \forall k, k < j) \\
&= \textstyle\prod_{j=1}^{r} \text{Prob}(x_j = -U_j'\dot{(}x_{r+1}, \ldots, x_n)^T) \\
&\leq \beta^r
\end{aligned}
$$

By similarly reasoning,

$$\alpha^r \leq \text{Prob}(\vec{x} : Ax = 0).$$

□

More generally, for a given $b \in \mathbb{F}^m$, if under the same hypotheses, then

$$\text{Prob}(\vec{x} : Ax = b) \leq \beta^r.$$

Furthermore, if $Ax = b$ has a solution. then

$$\text{Prob}(\vec{x} : Ax = b) \geq \alpha^r.$$

The next lemma characterizes the distribution of a random vector in a sub-space.

LEMMA 5.12. *Given a $k-$dimensional subspace of $\mathbb{F}^n$, $S = \text{Span}(S_1, \ldots, S_k)$. If $\vec{x} = (x_1, \ldots, x_n)^T$ is a random vector in $\mathbb{F}^N$ with each entry independently and non-uniformly randomly chosen from $\mathbb{F}$ with the probability that any element in $\mathbb{F}$ ranging from α to β is chosen, then*

$$\alpha^{n-k} \leq \text{Prob}(\vec{x} : x \in S) \leq \beta^{n-k}.$$

Proof. We extend $(S_1, \ldots, S_k)$ to be a basis of $\mathbb{F}^n$, $(S_1, \ldots, S_n)$. Let $B = (S_1, \ldots, S_n)$. Then

$$\text{Prob}(\vec{x} : \vec{x} \in S) = \text{Prob}(B^{-1}_{k+1,\ldots,n}\vec{x} = 0) \leq \beta^{n-k}.$$

By similarly reasoning,

$$\alpha^{n-k} \leq \text{Prob}(\vec{x} : \vec{x} \in S).$$

□

The next lemma characterizes the distribution of a random non-singular matrix.

THEOREM 5.13. *Given an $n \times n$ matrix B over $\mathbb{F}$, if R is an $n \times n$ random matrix over $\mathbb{F}$ with each entry independently and non-uniformly chosen from $\mathbb{F}$ with the probability that any particular element in $\mathbb{F}$ is chosen being no more than β, then*

$$\text{Prob}(\det(R - B) \neq 0) \geq \prod_{i=1}^{\infty}(1 - \beta^i).$$

Proof. We will only prove the case where $B = 0$ case, since $B \neq 0$ is result of this $B = 0$ case and lemma 5.10 If we let $R = (R_1, \ldots, R_n)$, then det$(A) \neq 0$ if and only if

$$a_i \notin \text{Span}(R_1, \ldots, R_{i-1}), \forall i, 1 \leq i \leq n.$$

Thus,

$$\begin{aligned}
&\text{Prob}(\det(R) \neq 0) \\
&= \textstyle\prod_{i=1}^{n} \text{Prob}(R_i \notin \text{Span}(R_1, \ldots, R_{i-1}) | R_1, \ldots, R_{i-1} \text{ are linearly independent.}) \\
&\geq \textstyle\prod_{i=1}^{n} (1 - \beta^{n+1-i}) \\
&\geq \textstyle\prod_{i=1}^{\infty} (1 - \beta^i).
\end{aligned}$$

□

For the case where $B = 0$, there is a similar result in [MS04]. Note that in the uniform case, we have $\beta = \frac{1}{p}$, thus

$$\text{Prob}(\det(R) \neq 0) \geq \prod_{i=1}^{\infty} (1 - (\frac{1}{p})^i).$$

If $\beta < 1$, then $\prod_{i=1}^{\infty}(1 - \beta^i)$ converges and is positive. Moreover,

$$\prod_{i=1}^{\infty}(1 - \beta^i) = e^{\sum_{i=1}^{\infty} \ln(1-\beta^i)} \geq e^{\sum_{i=1}^{\infty}(-\beta^i)} \geq e^{\frac{-\beta}{1-\beta}} \geq 1 - \frac{\beta}{1-\beta}.$$

The next lemma characterizes the distribution of a perturbation.

THEOREM 5.14. *Given an $n \times n$ matrix T and an $m \times n$ matrix B over $\mathbb{F}$ with $rank(\begin{bmatrix} T \\ B \end{bmatrix}) = n$, if R is an $n \times n$ random matrix over $\mathbb{F}$ with each entry independently and non-uniformly chosen from $\mathbb{F}$ with the probability that any element in $\mathbb{F}$ is chosen no more than β, then*

$$\text{Prob}(R : \det(T + RB) \neq 0) \geq \prod_{i=1}^{\infty}(1 - \beta^i).$$

Proof. If we let $r = rank(B)$, then there exists an $n \times n$ non-singular matrix Q, such that

$$BQ = \begin{bmatrix} B' & 0 \end{bmatrix},$$

where B' is of full column rank. Let us we rewrite

$$TQ = \begin{bmatrix} T_1 & T_2 \end{bmatrix},$$

where T_1 and T_2 are $n \times r$ and $n \times (n-r)$ matrices, respectively. Q is non-singular, so

$$n = \text{rank}(\begin{bmatrix} T \\ B \end{bmatrix}) = \text{rank}(\begin{bmatrix} TQ \\ BQ \end{bmatrix}) = \text{rank}(\begin{bmatrix} T_1 & T_2 \\ B' & 0 \end{bmatrix}).$$

Thus rank$(T_2) = n - r$. Therefore there exists non-singular M, such that,

$$MTQ = \begin{bmatrix} T_{11} & 0 \\ T_{21} & T_{22} \end{bmatrix} \text{ and } T_{22} \text{ is non-singular,}$$

where T_{11}, T_{21} and T_{22} are $r \times r$, $r \times (n-r)$, and $(n-r) \times (n-r)$ matrices, respectively. Let us look at

$$M(T + RB)Q = MTQ + MRBQ = \begin{bmatrix} T_{11} + M_1 RB' & 0 \\ * & T_{22} \end{bmatrix}.$$

Therefore

$$\det(T + RB) \neq 0 \text{ if and only } \det(T_{11} + M_1 RB') \neq 0,$$

since M, Q, and T_{22} are all non-singular.

Note that M_1 is fully row ranked and B' is fully column ranked. There exist permutation matrices, P_1 and P_2, such that the $i \times i$ leading principle minors of $M_1 P_1$ and $P_2 B'$ are non-singular. Let $R' = P_1^T R P_2^T$. Then $P_1 R B' = M_1 P_1 R' P_2 B'$. If we represent

$$M_1 P_1 = \begin{bmatrix} M_{11} & M_{12} \end{bmatrix}, P_2 B' = \begin{bmatrix} B_{11} & B_{12} \end{bmatrix}, R' = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}.$$

Then

$$T_{11} + M_1 RB' = M_{11}(R_{11} + Res(R_{12}, R_{21}, R_{22}, M_{11}, T_{11}, B_{11})B_{11}),$$

where

$Res(R_{12}, R_{21}, R_{22}, M_{11}, T_{11}, B_{11}) = M_{11}^{-1}(T_{11}+M_{12}B_{21}B_{11}+M_{11}R_{22}B_{12}+M_{12}R_{22}B_{12})B_{11}^{-1}$.

Since both M_{11} and B_{11} are non-singular, $T_{11} + M_1RB'$ is non-singular if and only if

$$\text{det}(R_{11} + Res(R_{12}, R_{21}, R_{22}, T_{11}, M_{11}, B_{11}) \neq 0.$$

So $\text{det}(T + RB) \neq 0$ if and only if

$$\text{det}(R_{11} + Res(R_{12}, R_{21}, R_{22}, T_{11}, M_{11}, B_{11}) \neq 0.$$

Thus

$$\begin{aligned}
&\text{Prob}(R : \text{det}(R_{11} + Res(R_{12}, R_{21}, R_{22}, T_{11}, M_{11}, B_{11}) \neq 0) \\
&= \textstyle\sum_{C_1,C_2,C_3} \text{Prob}(R_{11} : \text{det}(R_{11} + Res(C_1, C_2, C_3, T_{11}, M_{11}, B_{11}) \neq 0 \\
&\quad |R_{12} = C_1, R_{21} = C_2, R_{22} = C_3)\text{Prob}(R_{12} = C_1, R_{21} = C_2, R_{22} = C_3) \\
&= \textstyle\sum_{C_1,C_2,C_3} \text{Prob}(R_{11} : \text{det}(R_{11} + Res(C_1, C_2, C_3, T_{11}, M_{11}, B_{11}) \neq 0) \\
&\text{Prob}(R_{12} = C_1, R_{21} = C_2, R_{22} = C_3)
\end{aligned}$$

Therefore, by theorem 5.13,

$$\begin{aligned}
&\text{Prob}(R : \text{det}(R_{11} + Res(R_{12}, R_{21}, R_{22}, T_{11}, M_{11}, B_{11}) \neq 0) \\
&\geq \textstyle\sum_{C_1,C_2,C_3} \cdot \prod_{i=1}^{\infty}(1 - \beta^i)\text{Prob}((R_{12} = C_1, R_{21} = C_2, R_{22} = C_3) \cdot \\
&\geq \textstyle\prod_{i=1}^{\infty}(1 - \beta^i).
\end{aligned}$$

So

$$\text{Prob}(R : \text{det}(T + RB) \neq 0) \geq \prod_{i=1}^{\infty}(1 - \beta^i).$$

□

Note that if T and B are $n \times n$ and $m \times n$ matrices over $\mathbb{F}$, respectively, and if there exists R, such that $T + RB$ is non-singular, then

$$\text{rank}(\begin{bmatrix} T \\ B \end{bmatrix}) \geq \text{rank}(\begin{bmatrix} I & R \end{bmatrix} \begin{bmatrix} T \\ B \end{bmatrix}) \geq \text{rank}(T + RB) = n.$$

Thus

$$\text{rank}(\begin{bmatrix} T \\ B \end{bmatrix}) = n.$$

COROLLARY 5.15. *Given a prime p, under the same assumption in the theorem above about T and B, if R is an $n \times n$ random integer matrix with each entry independent and uniformly chosen from $[0, \ldots, S-1]$, then*

$$\text{Prob}(\det(T + RB) \neq 0 \pmod{p}) \geq \prod_{i=1}^{\infty}(1 - (\frac{1}{S}\lceil\frac{S}{p}\rceil)^i).$$

Proof. This follows from theorem 5.14 and fact 5.9. □

The lemma below explores the relationship between invariant factors of the perturbation and those of the original matrix.

LEMMA 5.16. *If L and A are $l \times n$ and $n \times m$ integer matrices, respectively, then $s_i(LA)$ is always a multiple of $s_i(A)$, for all $1 \leq i \leq \min(l, m, n)$.*

Proof. If $s_i(A) = 0$, then $\text{rank}(A) < i$. so

$$\text{rank}(LA) \leq \text{rank}(A) < i.$$

Thus $s_i(LA) = 0$.

If $s_i(A) = 1$, obviously the lemma is true.

Otherwise, let us have a look at the Smith form of LA and A mod $s_i(A)$. Suppose that A over $\mathbb{Z}_{s_i(A)}$ has the Smith form

$$S = \text{diag}(s_1, \ldots, s_r, 0, \ldots, 0), \text{ for some } r < i.$$

So there exist corresponding integer matrices U and V, such that both $\det(U)$ and $\det(V)$ are coprime to $s_i(A)$ and $A = USV$. Therefore,

$$s_i(LA) = s_i(LUSV) = s_i(\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix} V) \pmod{s_i(A)},$$

where L'_{11}, S_{11} are the first $r \times r$ principal sub-matrices of LU and S, respectively. Since det(V) is coprime to $s_i(A)$ and $r < i$, we know

$$s_i(\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix} V) = s_i(\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix}) = 0 \pmod{s_i(A)}.$$

This means $s_i(LA)$ is a multiple of $s_i(A)$. □

The lemma below explores the sufficient condition of $p \nmid \frac{s_i(LA)}{s_i(A)}$, where p is a prime.

LEMMA 5.17. *If A is an $n \times m$ integer matrix, p is a prime, and i is an integer, such that $s_i(A) \neq 0$, then there exists a full column ranked matrix M, such that for any $i \times n$ integer matrix L,*

$$p \nmid \frac{s_i(LA)}{s_i(A)}, \text{ if } \det(LM) \neq 0 \pmod{p}.$$

Proof. Let us look at the local Smith form of A at prime p. There exist integer matrices U, V and D, such that,

$$A = UDV, \det(U) \neq 0 \pmod{p}, \text{ and } \det(V) \neq 0 \pmod{p}, \text{ and}$$

$$D \text{ is the local Smith form of } A \text{ at } p.$$

We rewrite U, D as block matrices,

$$U = \begin{bmatrix} M & M' \end{bmatrix}, D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix},$$

where M is an $n \times i$ matrix, D_1 is an $i \times i$ matrix. Then

$$LA = \begin{bmatrix} LMD_1 & LM'D_2 \end{bmatrix} V.$$

Note that the entries of D_2 are multiples of those in D_1. So If $\det(LM) \neq 0 \mod p$, then D_1 is the local Smith form of LA at p, i.e. $p \nmid \frac{s_i(LA)}{s_i(A)}$. Also, M has full column rank. □

5.6.2.2 Proof of Theorem 5.8

Based on lemmas and theorems above, we are able to prove theorem 5.8.

Proof. If $s_i(A) = 0$, then

$$s_i(\begin{bmatrix} I_i & L \end{bmatrix} A \begin{bmatrix} I_i \\ R \end{bmatrix}) = 0, \text{ for any } L \text{ and } R.$$

Therefore the theorem is true.

Otherwise $s_i(A) \neq 0$. If L is a random $i \times (n-i)$ matrix with each entry uniformly and independently chosen from integer set $[0, ..., S-1]$, then by lemma 5.17 and corollary 5.15,

$$\text{Prob}(L : p \nmid \frac{s_i([I_i L]A)}{s_i(A)}) \geq \prod_{i=1}^{\infty}(1 - (\frac{1}{S}\lceil\frac{S}{p}\rceil)^i), \text{ for any prime } p.$$

If prime $p \geq S$, then

$$\frac{1}{S}\lceil\frac{S}{p}\rceil = \frac{1}{S}.$$

And if $11 \leq p < S$, then

$$\frac{1}{S}\lceil\frac{S}{p}\rceil \leq \frac{1}{p} + \frac{1}{S}.$$

Also, we choose S divisible by 2, 3, 5, and 7. Thus

$$\text{Prob}(L : p \nmid \frac{s_i([I_i L]A)}{s_i(A)}) \geq \begin{cases} 1 - \frac{1}{S-1}, & \text{if } p \geq S \\ 1 - 1.12(\frac{1}{p} + \frac{1}{S}), & \text{if } 11 \leq p < S \\ \frac{\sqrt{2}}{5}, & \text{if } p = 2 \\ \frac{1}{2}, & \text{if } p = 3 \\ \frac{3}{4}, & \text{if } p = 5 \\ \frac{5}{6}, & \text{if } p = 7 \end{cases}$$

Note that

$$\begin{aligned} \textstyle\prod_{i=1}^{\infty}(1 - (\frac{1}{p} + \frac{1}{S})^i) &\geq 1 - \frac{\frac{1}{p}+\frac{1}{S}}{1-\frac{1}{p}-\frac{1}{S}} \\ &> 1 - \frac{\frac{1}{p}+\frac{1}{S}}{1-\frac{1}{11}-\frac{1}{210}} \\ &> 1 - 1.12(\frac{1}{p} + \frac{1}{S}). \end{aligned}$$

Therefore, after $N = \log(10/\epsilon)/\log(24/23)$ number of iterations, the computed ith invariant factor of A,

$$\begin{aligned}\text{Prob}(s_i^{(N)} \neq s_i(A)) &\leq (\tfrac{23}{24})^N + (\tfrac{3}{4})^N + (\tfrac{7}{16})^N + (\tfrac{11}{36})^N + 5.5(\tfrac{4.5}{10})^{N-1} \\ &\leq 10(\tfrac{23}{24})^{N-1} \leq \epsilon.\end{aligned}$$

So algorithm 5 computes the ith invariant factor with probability at least $1-\epsilon$. By similarly reasoning, the second largest invariant factor is computed by the bonus idea with probability with probability near $1-\sqrt{\epsilon}$.

In algorithm 5, each iteration costs

$$\text{O}^{\sim}(in^2 + i^3(\log n + \log_2(\|A\|)) \cdot \log(1/\epsilon)).$$

So the total cost will be

$$\text{O}^{\sim}(in^2 + i^3(\log n + \log_2(\|A\|)) \cdot \log(1/\epsilon)) \log(1/\epsilon)).$$

□

5.6.3 Binary search for invariant factors

The next lemma gives an upper bound for the number of distinct invariant factors of an integer matrix.

LEMMA 5.18. *Let A be an $n \times n$ integer matrix. Then A has at most*

$$\text{O}(n^{1/2}(\log n + \log_2(\|A\|))^{1/2})$$

distinct invariant factors.

Proof. See e.g. [EGV00, Theorem 4.1]. Or the non-singular case can simply be proved by the argument below. Suppose A has invariant factors $s_1, \ldots, s_n$, and k distinct invariant factors $s_{i_1}, \ldots, s_{i_k}$ in increasing order. We know

$$\prod_{j=1}^{k} s_{i_j} \leq \prod_{j=1}^{n} s_n = |\det(A)|.$$

Also, $2s_{i_j} \leq s_{i_{j+1}}$, for all $1 \leq j \leq k-1$. So $S_{i_j} \geq 2^{j-1}$. Thus

$$|\det(A)| \geq \prod_{j=1}^{k} s_{i_j} \geq \prod_{j=1}^{k} 2^{j-1} = 2^{j(j-1)/2}.$$

Therefore, A has at most $O(\sqrt{\log |\det(A)|})$ distinct invariant factors. By the Hadamard bound, $|\det(A)| \leq n^{n/2}(\log_2(\|A\|))^n$. So A has at most

$$O(n^{1/2}(\log n + \log_2(\|A\|))^{1/2}).$$

The proof can be modified to handle the singular case as well. □

Algorithm 5 can be used to compute the invariant factor of any index. A binary search for every distinct invariant factor is feasible. Without loss of generality, we assume that A is an $n \times m$ integer matrix, and $m \leq n$.

THEOREM 5.19. *Given a $n \times n$ integer matrix and error bound ϵ, there exists a probabilistic algorithm which computes the Smith form of A with probability at least $1 - \epsilon$, and is expected to run in*

$$O^{\sim}(n^{3.5}(\log_2(\|A\|))^{1.5} \log^2(1/\epsilon))$$

bit operations.

Proof. See [EGV00, THEOREM 4.2]. □

5.7 Smith form algorithms for sparse matrices

Though the chapter focuses on the Smith form of dense matrices, some interesting algorithms for sparse cases are helpful to form an "engineered" Smith form algorithm. In 1996, M. Giesbrecht [Gie96] showed the Smith form can be computed in $O^{\sim}(n^2 E \log_2(\|A\|))$ bit operations, where E is the number of non-zero entries of A. This algorithm is based on computation of determinantal divisors d_i, which are the greatest common divisor of all the $i \times i$ minors of A. Determinantal divisors are computed by

the characteristic polynomials over the integers. Note that the i^{th} invariant factor of A is equal to d_i/d_{i-1} (let $d_0 = 1$), or 0 if $d_i = 0$.

In 2000, J. Dumas, B. Saunders and G. Villard [DSV00] showed an interesting valence algorithm, which requires $\tilde{O}(n^2 E \log_2(\|A\|))$ bit operations plus the time for factoring the valences. Firstly the valence v, which is the trailing nonzero coefficient of the minimal polynomial of $A^T A$, is computed. The valence will give information about the primes which may occur in invariant factors. Then the valence v is factored to find each prime which may occur in invariant factors. For each prime $p|v$, S_p, the Smith form of A over the local ring $\mathbb{Z}^{(p)} = \bigcup_{i=1}^{\infty} \mathbb{Z}_{p^i}$, is computed. Note that the Smith form of A = $\prod_{p|v} S_p$. The valence algorithm succeeds in many examples, especially those in computation of homology groups. Dumas, Heckenbach, Saunders and Welker developed a Gap homology package, which is available at `www.eecis.udel.edu/~linbox/gap.html`. In the next subsection, we will discuss the Smith form of a diagonal matrix which is a very special sparse matrix.

5.7.1 Smith forms of diagonal matrices

Iliopoulos' elimination produces an equivalent diagonal matrix. Computation of the Smith form of this diagonal matrix is rather straightforward using n^2 (extended) gcd computations[Ili89], a cost that is much lower than the elimination to diagonal form. Nonetheless, in this section we present a faster method to compute the Smith form of a diagonal matrix. It is soft linear in the size of the input. This speedup may be significant in some settings. In any case, it is an interesting and easily implemented method.

Given a matrix $A = \text{diag}(a_1, \ldots, a_n)$ we show how to compute its Smith form. Define the input size to be $N = \sum_{i=0}^{n}(\text{len}(a_i))$, where $\text{len}(a) = 1 + \log(|a|)$. The run time is soft linear in N, $\tilde{O}(N)$. A basic observation is that

$$\text{diag}(a, b) \text{ is equivalent to } \text{diag}(g, l), \text{ where}$$

$$g = \gcd(a, b) \text{ and } l = \text{lcm}(a, b).$$

Furthermore, the cost of this operation is O($G(\text{len}(a) + \text{len}(b))$) bit operations, where $G(m)$, the cost of gcd/lcm computation size m numbers, is O($m \log^2(m) \log\log(m)$) using the fast gcd [vzGG99]. Also note that the memory occupied by g and l, is the same as that for a and b since $ab = gl$, and thus

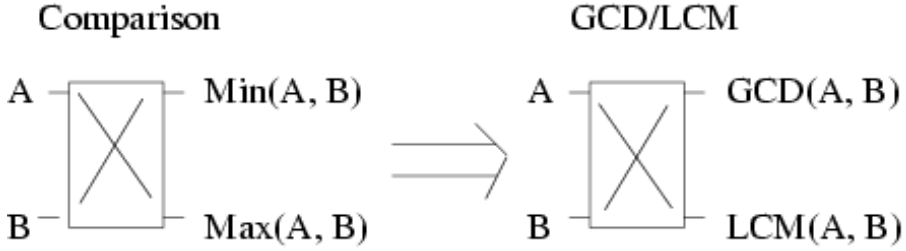
$$\log(a) + \log(b) = \log(g) + \log(l).$$

We will compute our Smith form by a series of such transformations, each replacing two elements by their gcd and lcm.

For diag$(a_1, \ldots, a_n)$, the n diagonal entries can be partitioned into $n/2$ pairs and the gcd/lcm replacement done on each pair independently. Using the fast methods, the cost of these $n/2$ operations is O($G(\sum_{i=1}^{n} \text{len}(a_i))$), i.e. softly linear in the input size.

The main idea is that sorting networks can be simply modified to compute the Smith form. Sorting networks are circuits consisting of comparator units wired together. Each comparator has two input wires and two output wires producing the min of the two inputs at one output wire and the max at the other. If the min/max comparators are replaced by gcd/lcm butterflies (gcd = min, lcm = max, and producing gcd at one output and lcm at the other), as shown in figure 5.1 a sorting network will compute the Smith form. To see this, observe that locally at each prime p, the gcd/lcm butterfly *is* a min/max comparator ($\gcd(p^e, p^f) = \min(p^e, p^f)$). As is well known, a matrix is in Smith form if and only if it is in Smith form locally at every prime.

Figure 5.1: A gcd/lcm sorting network



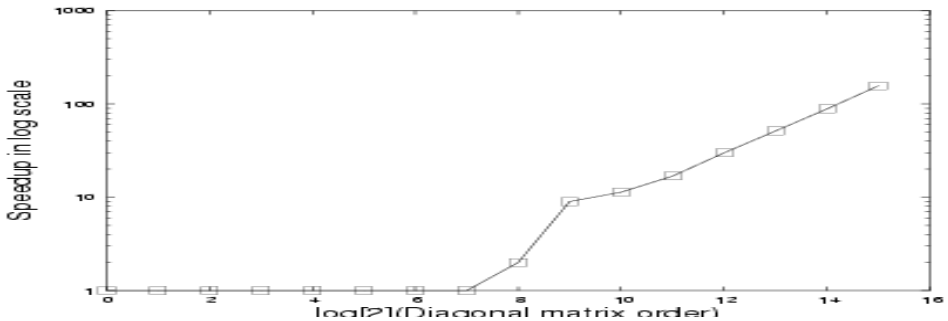
Without loss of generality, assume n be a power of two and let us consider staged sorting networks in which each stage is a circuit of depth one having n inputs feeding $n/2$ comparators, and n outputs feeding the next stage. Sorting networks such as Batcher's odd-even merge [Bat68], and the variant described in [CLRS01] require $\mathrm{O}(\log^2(n))$ stages. There is also a staged sorting network requiring only $\mathrm{O}(n\log(n))$ times [AKS83], but with large constant factor. Any such staged network may be used to compute diagonal Smith forms in soft linear time, since each stage costs soft linear time producing output of the same total size as that of the input, and there are logarithmically many stages. We have proven

THEOREM 5.20. *Let A be a diagonal matrix over a Principal Ideal Ring R. A staged sorting network with the min/max comparators replaced by gcd/lcm butterflies will compute the Smith form of A. If $R = Z$, the cost is $\mathrm{O}^{\sim}(N)$, where N is the sum of the lengths of the diagonal entries of A.*

If d is an upper bound for $\log(|a_i|)$, we may also observe that the run time is $\mathrm{O}^{\sim}(nd)$.

A softly linear method already exists as a consequence of Bernstein's "coprime base" algorithm[Berar]. Given $(a_1, \ldots, a_n)$, he computes the unique $(p_1, \ldots, p_m)$, such that the $p_i's$ are relatively prime to each other and each a_i is a product of powers of the p_i. He also computes $e_{i,j}$, the power of p_i in a_j, and by sorting the $e_{i,j}$ for each i the invariant factors may be obtained. The cost estimate given in [Berar, Theorem 18.2] for the coprime basis computation is $\mathrm{O}(N\log^4 N\log n\mu(N))$, where $\mu(k)$ is the cost of arithmetic on numbers of size k. The factoring of the a_i against the basis costs less[Berar, Theorem 21.3]. Clearly, so does the sorting, so that basis construction is the dominant cost. The algorithm presented here, when using the asymptotically fastest sorting network, has a runtime bound which is a factor of $\log^3(N)$ smaller. Next, we will present some performance data in graph. In figure 5.2 we compare experimentally the gcd/lcm sorting network approach to the standard n^2 gcd's approach. The latter method is, for diagonal

Figure 5.2: Diagonal matrix Smith form speedup (classical method time / sorting network time).



matrix $(d_1, \ldots, d_n)$, to do: for i from 1 to $n-1$, for j from $i+1$ to n or until $d_i = 1$, replace d_i and d_j by gcd(d_i, d_j) and lcm(d_i, d_j) respectively. Our measurements show that the speedup is substantial and eventually tends to be linear in n as expected.

5.8 An adaptive algorithm for Smith form

Though we do not have an efficient adaptive algorithm for Smith form of a sparse matrix, we have found a run-time efficient adaptive algorithm for the Smith form of a dense matrix. There are a number of Smith form algorithms. If asymptotic complexity is the only criterion, it is easy to sort them out as shown in the table below.

Table 5.1: Algorithms for computing Smith forms

Author(s)	Year	Citation	Time Complexity	Type
Smith	1861	[Smi61]	UNKNOWN	Deterministic
Kannan and Bachem	1979	[KB79]	Polynomial	Deterministic
Iliopoulos	1989	[Ili89]	$O^{\sim}(n^5(\log_2(\|A\|))^2)$	Deterministic
Hafner and McCurley	1991	[HM91]	$O^{\sim}(n^5(\log_2(\|A\|))^2)$	Deterministic
Storjohann	1996	[Sto96]	$O^{\sim}(n^{\omega+1}\log_2(\|A\|))$	Deterministic
Eberly, Giesbrecht and Villard	2000	[EGV00]	$O^{\sim}(n^{\omega/2+2}\log_2(\|A\|))$	Monte-Carlo
Kaltofen and Villard	2003	[KV03]	$O^{\sim}(n^{2.697263}\log_2(\|A\|))$	Monte-Carlo

However in practice, it's hard to tell which will perform best. The asymptotically fastest algorithms may not always yield the best practical run time. We estimate that the Smith form algorithm in [KV03] with the best asymptotic complexity turns out to be slow in the current hardware environment. The binary search algorithm in [EGV00] is also slow in the current hardware environment. These earlier elimination algorithms such as in [Ili89, HM91, Sto96] can be efficient in practice for certain cases. Also these algorithms can be improved by working mod the (second) largest invariant factor which can be computed either by algorithm 4 for the non-singular cases or by algorithm 5 for the singular cases. The computation of the largest invariant factor is practical. Based

certain matrices.

Both algorithm 4 and algorithm 5 are probabilistic algorithms and require a larger number of iterations of symbolically solving non-singular linear systems to correctly compute the largest invariant factor regarding small primes than they do regarding large primes. For example, for a non-singular matrix, 20 iterations of symbolically solving non-singular linear systems are required in order to compute the power of 2 occurring in the largest invariant factor with probability of $1 - \frac{1}{1000,000}$, while only 4 iterations are expected to correctly compute the largest invariant with the same probability regarding only primes bigger than 100. We'll do better to compute the contribution of small primes and large primes in separate ways. To facilitate the discussion of this, we borrow from number theory the terminology of smooth and rough numbers.

DEFINITION 5.21. Given a positive integer k, a number is *k-smooth* if it is a product of primes smaller than k and is *k-rough* if it is a product of primes no less than k.

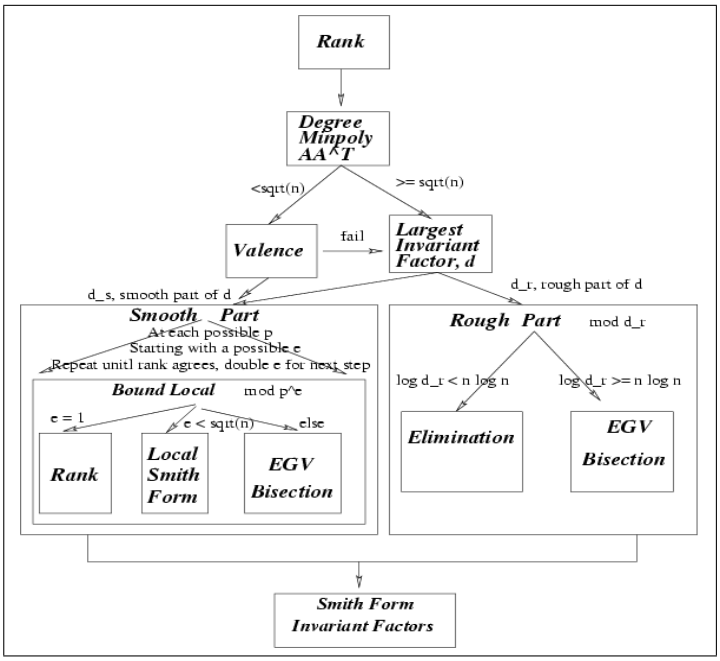
Given an integer k, every integer i can be written as a product of a k-smooth number and a k-rough number.

Based this knowledge above, an "engineered" algorithm for Smith form is proposed to deal with all cases. We called it *engineered* because its structure is based on both theoretical and experimental results. Different algorithms are suitable for different parts. In particular, we discovered the importance of separating the rough part and smooth part of invariant factors Therefore we divide the problem into smooth and rough sub-problems. For each subproblem, we use adaptive algorithms to solve it based on thresholds, which are determined by both theoretical and experimental results. The structure of the algorithm has been influenced by classes of matrices we have encountered in practice. We have designed it to recognize key patterns and go to the fastest practical algorithm for those cases, while avoiding cases where this could go badly awry.

The algorithm begins, for matrix A, by computing r, the rank of A, and d, the degree of the minimal polynomial of A or AA^T. These can be done quite rapidly. The

rank is needed in all cases, but d is computed just in order to decide between the valence approach and approaches which begin with the computation of the largest invariant factor. Next the smooth and rough parts of the valence or largest invariant factor are handled separately. The input to the smooth part includes a vector E of exponents for the first 25 primes. When coming from the valence, this is a zero-one vector with a zero indicating a prime known to be absent and a one indicating a prime which may occur. When coming from the largest invariant factor, the exponents are all positive and approximate the largest exponent of the prime occurring in the largest invariant factor. The organization of the adaptive Smith form algorithm is shown in figure 5.3.

Figure 5.3: An engineered algorithm to compute the Smith form.



In practice, we choose k, the smooth/rough threshold to be 100. Note that there are exactly 25 primes which are smaller than 100. Below is an outline of our engineered Smith form algorithm:

ALGORITHM 6. Engineered Smith form algorithm outline

Input:

- A, an $n \times n$ integer matrix.
- ϵ, an error probability requirement.

Output:

- S, vector of the invariant factors of A. Monte Carlo with error probability at most ϵ.

Procedure:

1. [rank] Compute r = rank of A. This can be done rapidly by computing mod a random prime from a suitable range. It is Monte Carlo with error probability at most $\epsilon/2$.
2. [minpoly degree] Compute the probable degree of the minimal polynomial of AA^T. Computing the minimal polynomial mod a single random prime suffices for this step. If this degree is less than a threshold go to the valence step, otherwise proceed to the largest invariant factor step. We want a threshold less than $\sqrt{n}$ for asymptotic complexity reasons. In practice the valence method is advantageous primarily when the minimal polynomial degree is very low, so the threshold can be set lower than that.
    (a) [valence] Any prime that occurs in the invariant factors of A occurs in the valence (nonzero coefficient of lowest degree term) of the minimal polynomial of A or AA^T . It has been found in practice that some matrices of interest have low degree minimal polynomials and rapidly computed valence for A or AA^T. See [DSV00] for further discussion of the valence method and for examples.

Compute one of these valences, v. If v is smooth, for i from 1 to 25, let E_i be 1 if the i-th prime divides v, and be 0 otherwise. Let $t \to 1$, and go to the smooth step. Otherwise the valence step fails, so proceed to largest invariant factor step.

(b) [largest invariant factor] Compute the largest invariant factor s_r and bonus, the precedent invariant factor, s_{r-1}, also using algorithm 5. The rough parts of the bonus and of the largest invariant factor agree with the true values with high probability. Thus the rough part of the bonus, often much smaller than the rough part of s_r, can be used in the rough part step below. Let s be the smooth part of the largest invariant factor for i from 1 to 25, let E_i be one greater than the exponent of the i-th prime in s_r, and let t be the remaining (rough) part of s_r.

3. [smooth part] Let $S_s \to \text{diag}(1, \ldots, 1)$, this will become the smooth part of the Smith form. For i from 1 to 25, if E_i is positive, compute the local Smith form, S_p at the i-th prime, and set $S_s \to S_s * S_p$. The local Smith form at p is computed by starting with the estimated exponent e. If the number of non-zero invariant factors in $A \pmod{p^e}$ is less than rank(A), double e, repeat the computation until the number of non-zero invariant factors mod p^e is equal to rank(A).

4. [rough part] Compute S_r the rough part of the Smith form by elimination if the modulus is small, or by a binary search if not. Correct the rough part of the largest invariant factor if necessary.

5. [return] Compute and return $S = S_s * S_r$. In practice, S, S_s, and S_r may be represented as vectors (of the diagonal elements) or even as lists of the distinct entries with their multiplicities.

The asymptotic complexity of the engineered algorithm above can be as good as the Smith form algorithm in [EGV00]. We have following theorem.

THEOREM 5.22. *There exist crossovers (thresholds) such that the engineered algorithm can probabilistically compute the Smith form of an $n\times$ integer matrices in $\mathrm{O}^{\sim}(n^{3.5}(\log_2(\|A\|))^{1.5})$ bit operations.*

Proof. In order to achieve $\mathrm{O}^{\sim}(n^{3.5}(\log_2(\|A\|))^{1.5})$, we need to choose all crossovers such that all parts can be computed in $\mathrm{O}^{\sim}(n^{3.5}(\log_2(\|A\|))^{1.5})$. We choose the crossover between the valence and the largest invariant factor to be $\mathrm{O}^{\sim}(n^{0.5})$. Also we may choose the crossovers between an elimination method and a binary search method for the local Smith form and the rough part of the Smith form such that each can be computed in $\mathrm{O}^{\sim}(n^{3.5}(\log_2(\|A\|))^{1.5})$ bit operations. □

These crossovers are important in practice. Performance of the engineered algorithm are influenced by these crossovers. In our implementation, these crossovers are chosen to favor our examples.

In the next section, we give some experimental results showing the effectiveness of this algorithm. Our experimentation is based on our implementation of the adaptive algorithm in the LinBox library. The code for random matrices used in our experiments is in the LinBox library. Other examples from other people are available on requests.

5.9 Experimental results

We have found that the Smith form algorithm [EGV00] is very inefficient in practice. We don't have an implementation of Storjohann's algorithm. The average runtime of Iliopoulos' algorithm is likely $\mathrm{O}^{\sim}(n^4\log_2(\|A\|))$ rather than the worst case $\mathrm{O}^{\sim}(n^5(\log_2(\|A\|))^2)$. This is borne out by statistical fitting to timings on matrices encountered in practice. The run time complexity is comparable to asymptotic complexity of Storjohann's algorithm. The bonus idea in algorithm 4 helps a lot in practice. We have found the elimination mod the largest invariant factor (second largest invariant factor) can run in reasonable time. And in some special sparse cases, the valence algorithm in [DSV00] can be efficient. We have tested on a number of different families of matrices.

All results show that our engineered algorithm is effective in practice. Here I will present two examples. The first one is a family of matrices which are randomly equivalent to

$$\text{diag}(1, 2, 3, \ldots, n),$$

where n is the dimension. For example, when $n = 100$, the invariant factors are: [(1 50), (2 17), (6 8), (12 5), (60 6), (420 2), (840 1), (2520 2), (27720 2), (360360 1), (720720 1), (232792560 1), (26771144400 1), (144403552893600 1), (309904450424599670640 0 1), (69720375229712477164533808935312303556800 1).] The Smith forms are described in compact form as a list of value, occurrence pairs. A pair (v, n) denotes n occurrences of invariant v on the Smith form diagonal.

In the table below, we compare the practical run time by our engineered algorithm with the elimination step time by Iliopoulos' algorithm ignoring the time for determinant computation. It shows that the bonus idea in algorithm 4 is very helpful in practice and our engineered algorithm is faster in practice.

Table 5.2: Run time of our engineered algorithm on constructed examples

Order	100	200	400	800	1600	3200
Engineered	0.93	5.31	52.9	643.99	7497.65	133487
Elimination	1.08	8.28	85.82	1278.73	12881.5	185382

Test was run sequentially on a server with dual 3.2GHZ Xero processors, 6GB main memory. The elimination indicates the elimination step time by Iliopoulos' algorithm ignoring the determinant computation time.

Another example is a collection of matrices from practice. We will describe their general nature next, and indicate the structure of their Smith forms. The Chandler's matrices are arising from combinatorial study. In our notation, the name of each matrix in

this family starts with C and the rest indicated its dimension. C1093 and C9841 are zero-one matrices of the indicated orders, each having about $1/3$ of the entries nonzero. In fact they are circulant, but we don't exploit that property here. They have Smith forms whose invariant factors are powers of 3 (except the last which have additional factors 364 and 3280 respectively. For example, The Smith form of C9841 = [(1 145), (3 1440), (3^2, 1572), (3^3, 1764), (3^4, 1764), (3^5, 1572), (3^6, 1440), (3^7, 143), ($2^4 * 3^7 * 5 * 41$, 1)], and C1093's is similar. C4369x70161 is again a zero-one matrix, expressing subspace incidences. It is quite sparse, with about 17 nonzero entries per column. The interest is in the power of 2 in the invariants, and, except for the last all are relatively small powers of 2, which makes for very fast computation the hardware overflow as free normalization mod 2^{32}. Its Smith from is [(1, 2801), (2, 640), (2^2, 416), (2^3, 256), (2^4, 255), ($2^4 \cdot 17$, 1)]. C5797 is a zero one incidence matrix with about 21 non-zeros per row. The invariant factors in this Smith form are primarily powers of 2, but we see also some structure at 5 and, as in other examples, extra primes occurring only in the largest invariant. [(1, 2226), (2, 430), (2^2, 801), (2^3, 410), (2^4, 1590), ($2^4 \cdot 5$, 170), ($2^5 \cdot 5$, 70), ($2^6 \cdot 5$, 99), ($2^6 \cdot 3 \cdot 5 \cdot 7$, 1)]. We remark that, for this matrix, the Smith form computation seems to be a fast way to get the determinant. Computation of the determinant via eliminations mod primes and the Chinese remaindering algorithm requires about twice as long as the hybrid Smith form algorithm.

Note that the bonus (second invariant factor) effectively removes the need for further computation with the extra factors in the last invariant beyond the initial step of computing it. This can be a huge savings, most especially when the remaining part is smooth as so often happens in our experience.

Krattenthaler's matrices K1 through K20 arise in some number theoretical computation to find formulas for π. $K\langle n\rangle$ has dimension $16n^2 + 8n + 1$ and the entries are mostly of modest size. However a few entries are large. For example, K20 has 163 large entries of lengths ranging from 275 up to 1245 digits. Here the need is to compute the

determinant (and factor it, which is easy since the factors are smooth). It turns out that these matrices also have rich structure in their Smith forms, meaning that numerous invariant factors are nontrivial and the largest invariant factor is significantly smaller than the determinant. For this reason, the smooth part of of our Smith form algorithm is the fastest way we know to compute these determinants.

The practical run time is shown in table 5.3. It turns out that the Chandler's matrices are favorable by the valence algorithm. In this case, the overhead of our engineered algorithm is negligible. As shown in the table, our engineer algorithm can yield great performance in these matrices.

Table 5.3: Run time of our engineered algorithm over practical examples

name	C1093	$K\langle 10\rangle$	C5797	C9841	C4369x70161
engineered	47.03	7600.83	6445.45	42639.8	67558
Elimination	153.41	13224.2	20807.4	NED	NEM

In the table above, NEM stands for "not enough memory". Test was run sequentially on a server with dual 3.2GHZ Xero processors, 6GB main memory. The elimination indicates the elimination step time by Iliopoulos' algorithm ignoring the determinant computation time.

CONCLUSION

In this thesis we have contributed new methods in exact linear algebra for the computation of minimal polynomial, characteristic polynomial, matrix rank, exact rational system solution, and Smith form. Our algorithms improve the state of the art both in theory (algorithm complexity) and (especially) in practice. Also, We have implemented all this in the system LinBox and contributed substantially to LinBox in other ways as well. In future we would like to make further improvements, especially in these areas: exact solutions of sparse systems and Smith forms of sparse matrices.

BIBLIOGRAPHY

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c\lg(n)$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[ASW05] J. Adams, B. D. Saunders, and Z. Wan. Signature of symmetric rational matrices and the unitary dual of Lie groups. In *Proc. of ISSAC'05*, 2005.

[Bat68] K.E. Batcher. Sorting networks and their applications. volume 32, pages 307–314, 1968.

[BEPP99] H. Bronnimann, I. Z. Emiris, V. Y. Pan, and S. Pion. Sign determination in residue number systems. *Theoret. Comput. Sci.*, 210:173197, 1999.

[Berar] D. J. Bernstein. Factoring into coprimes in essentially linear time. *Journal of Algorithms*, to appear.

[BLWW04] Folkmar Bornemann, Dirk Laurie, Stan Wagon, and Jórg Waldvogel. *The SIAM 100-Digit Challenge A Study in High-Accuracy Numerical Computing*. SIAM, 2004.

[CC82] T. W. Chou and G. E. Collins. Algorithms for the solution of systems of linear diophantine equations. *Siam J. Comput.*, 11:687–708, 1982.

[CEK+02] L. Chen, W. Eberly, E. Kaltofen, W. Turner, B. D. Saunders, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *LAA 343-344, 2002*, pages 119–146, 2002.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second edition*. MIT Press, 2001.

[Cop95] D. Coppersmith. Solving homegeneous linear equations over $GF(2)$ via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1995.

[CS04] Z. Chen and A. Storjohann. An implementation of linear system solving for integer matrices. In *Poster, ISSAC'04*, 2004.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th ACM STOC*, pages 1 – 6, 1987.

[Dem97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[DGP02] J-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In *Proc. ISSAC'02*, pages 63 – 74. ACM Press, 2002.

[DGP04] J-G. Dumas, P. Giorgi, and C. Pernet. FFPACK: Finite field linear algebra package. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain*. ACM Press, New York, 2004.

[Dix82] J. D. Dixon. Exact solution of linear equations using p-adic expansion. *Numer. Math.*, pages 137–141, 1982.

[DPW05] J-G. Dumas, C. Pernet, and Z. Wan. Efficient computation of the characteristic polynomial. In *Proc. of ISSAC'05*, 2005.

[DSV00] J-G. Dumas, B. D. Saunders, and G. Villard. Smith form via the valence: Experience with matrices from homology. In *Proc. ISSAC'00*, pages 95 – 105. ACM Press, 2000.

[DSW03] A. Duran, B. D. Saunders, and Z. Wan. Rank of sparse integer matrices. In *Siamla 2003*, 2003.

[DTW02] J-G. Dumas, W. Turner, and Z. Wan. Exact solution to large sparse integer linear systems. In *Poster, ECCAD 2002*, 2002.

[Ebe03] W. Eberly. Early termination over small fields. In *Proc. ISSAC'03*, pages 80–87. ACM Press, 2003.

[EGV00] W. Eberly, M. Giesbrecht, and G. Villard. On computing the determinant and Smith form of an integer matrix. In *Proc. 41st FOCS*, pages 675 – 687, 2000.

[EK97] W. Eberly and E. Kaltofen. On randomized Lanczos algorithms. In *Proc. ISSAC'97*, pages 176 – 183. ACM Press, 1997.

[Emi98] I. Z. Emiris. A complete implementation for computing general dimensional convex hulls. *Inter. J. Comput. Geom. Appl.*, 8:223–253, 1998.

[FH97] X. Fang and G. Havas. On the worst-case complexity of integer guassian elimination. In *Proc. ISSAC'97*, pages 28 –31. ACM Press, 1997.

[FM67] G. E. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, 1967.

[Fru77] M. A. Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. In M. Karpiński, editor, *Fundamentals of Computation Theory*, pages 386–392. Springer-Verlag, 1977.

[Gie96] M. Giesbrecht. Probabilistic computation of the Smith normal form of a sparse integer matrix. In *Proc. ANTS'96*, pages 173–186. Springer-Verlag, 1996.

[Gie01] M. Giesbrecht. Fast computation of the Smith form of a sparse integer matrix. *Computational Complexity*, 10:41–69, 2001.

[GZ02] K.O. Geddes and W.W. Zheng. Exploiting fast hardware floating point in high precision computation. Technical report, School of Computer Science, University of Waterloo, CA, December 2002.

[HM91] J. L. Hafner and K. S. McCurley. Asymptotically fast triangularization of matrices over rings. *Siam J. Comput.*, 20:1068–1083, 1991.

[Hou64] A. S. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell, Waltham, Mass., 1964.

[Ili89] C. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups of the Hermite and Smith normal forms of an integer matrix. *SIAM J. Comput.*, Vol. 18, No.4:658 – 669, 1989.

[IMH82] O. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, March 1982.

[Kal95] E. Kaltofen. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Mathematics of Computation*, 64(210):777–806, 1995.

[Kal02] E. Kaltofen. An out-sensitive variant of the baby steps / giant steps determinant algorithm. In *Proc. of ISSAC'02*, pages 138–144, 2002.

[KB79] R. Kannan and A. Bachem. Polynomial algorithm for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, Vol 8, No.4:499 – 507, 1979.

[KG85] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical computer science*, 36:309–317, 1985.

[KKS90] E. Kaltofen, M. Krishnamoorthy, and B. D. Saunders. Parallel algorithms for matrix normal forms. *Linear Algebra and its Applications*, pages 189 – 208, 1990.

[KS91] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. AAECC-9*, volume 539 of *Lect. Notes Comput. Sci.*, pages 29–38. Springer Verlag, 1991.

[KV03] E. Kaltofen and G. Villard. On the complexity of computing determinants. Technical Report 36, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, October 2003.

[LSW03] A. Lobo, B. D. Saunders, and Z. Wan. Rank and Smith form of extremely sparse matrices. *Abstract. ACA 2003*, 2003.

[Mas69] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15:122–127, 1969.

[MC79] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 65–73. Springer-Verlag, 1979.

[MS04] T. Mulders and A. Storjohann. Certified dense linear system solving. *Jounal of symbolic computation*, 37(4), 2004.

[New72] M. Newman. *Integral Matrices*. Academic Press, 1972.

[PW02] V. Pan and X. Wang. Acceleration of Euclidean algorithm and extensions. In *Proc. ISSAC'02*, pages 207 – 213. ACM Press, 2002.

[PW03] C. Pernet and Z. Wan. Lu based algorithms for the characteristic polynomial over a finite field. In *Poster, ISSAC'03*. ACM Press, 2003.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, pages 27:701–717, 1980.

[Smi61] H. M. S. Smith. On systems of indeterminate equations and congruences. *Philos. Trans.*, pages 293–326, 1861.

[SSV04] B. D. Saunders, A. Storjohann, and G. Villard. Matrix rank certification. *ELECTRONIC Journal of LINEAR ALGEBRA*, 11:16–23, 2004.

[Sto96] A. Storjohann. Near optimal algorithms for computing Smith normal forms of integer matrices. In *Proc. ISSAC'96*, pages 267 – 274. ACM Press, 1996.

[Sup03] *SuperLU User's Guide:* `http://www.nersc.gov/~xiaoye/SuperLU/`, 2003.

[SW04] B. D. Saunders and Z. Wan. Smith normal form of dense integer matrices fast algorithms into practice. In *Proc. ISSAC'04*, pages 274–281. ACM Press, 2004.

[SW05] B. D. Saunders and Z. Wan. Fast early termination technique in symbolic linear algebra. In *Poster, ECCAD 2005*, 2005.

[TB97] L. N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. SIAM, 1997.

[Vil00] G. Villard. Computing the frobenius normal form of a sparse matrix. In *The Third International Workshop on Computer Algebra in Scientific Computing*, pages 395–407, Oct, 2000.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Wan04] Z. Wan. An algorithm to solve integer linear systems exactly using numerical methods. *Journal of Symbolic Computation, submitted*, 2004.

[Wie86] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theory*, 32:54 – 62, 1986.