A BLAS Based C Library for Exact Linear Algebra on Integer Matrices

Zhuliang Chen
http://www.uwaterloo.ca/~z4chen

Arne Storjohann
http://www.uwaterloo.ca/~astorjoh

School of Computer Science, U. Waterloo
Waterloo, Ontario, N2L 3G1 Canada

ABSTRACT

Algorithms for solving linear systems of equations over the integers are designed and implemented. The implementations are based on the highly optimized and portable ATLAS/BLAS library for numerical linear algebra and the GNU Multiple Precision library (GMP) for large integer arithmetic.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; G.4 [**Mathematical Software**]: Algorithm design and analysis, Efficiency

General Terms

Algorithms

Keywords

integer matrix, linear system solving

1. INTRODUCTION

The fundamental problem in exact linear algebra is to compute a particular solution to a system of linear equations. Linear solving can be divided into two cases. The first case – *nonsingular solving* – is to compute the unique solution to a nonsingular system $Ax = b$, where $A \in \mathbb{Z}^{n\times n}$ and $b \in \mathbb{Z}^{n\times 1}$. The second and more general case – *certified solving* – is to compute a solution with minimal denominator to a system $Ax = b$, or to certify that the system has no solution, where $A \in \mathbb{Z}^{n\times m}$ has arbitrary shape and rank.

Nonsingular solving is the main building block of many recently proposed algorithms for other problems, including Diophantine solving [13, 20], the certified solving problem mentioned above [21], determinant [1, 10], Smith form [10, 23], and special cases of Hermite form [28]. Nonsingular solving is the main computational task driving the cost of all the algorithms in [1, 10, 13, 20, 21, 23, 28].

The two main contributions of this paper are as follows. First, we describe an efficient implementation of the well-know p-adic lifting algorithm [5, 19] for nonsingular solving. Second, we design and report on an implementation of a new algorithm, based on ideas from [13, 14, 21], for certified solving. A feature of the certified solver is that lattice basis reduction may optionally be used to reduce the size of the particular solution. The implementations of these exact solvers are included in the recently released free library of C source code called IML — Integer Matrix Library[1].

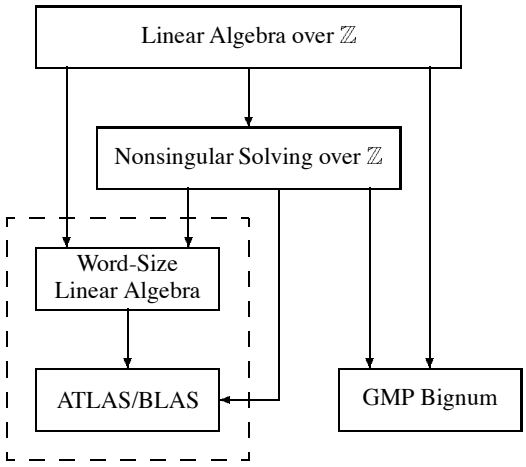

Figure 1: Organization of IML

Below we discuss these two contributions in more detail, but first consider the organization of the IML library shown in Figure 1. The modules in the lowest level refer to two highly optimized and portable software libraries: the Automatically Tuned Linear Algebra Software library [29] for numerical linear algebra, and the GNU Multiple Precision library [16] for large integer arithmetic.

Now consider the dashed box in Figure 1. The module Word-Size Linear Algebra refers to the computation of matrix invariants (e.g., determinant, inverse, rank, left and right nullspace, row echelon form) over a small prime field. Section 2 discusses our implementation of this module. We use the standard representation for elements of $\mathbb{Z}/(p)$ — nonnegative integers between 0 and $p-1$. In IML we always choose the moduli p small enough to allow direct use of the numerical BLAS routines for basic arithmetic operations on matrices and vectors. For example, by stipulating that $n(p-1)^2 \le 2^{53}-1$ (the size of a double mantissa), the multiplication over $\mathbb{Z}/(p)$ of two matrices with inner dimension n can be performed with single matrix multiplication over $\mathbb{Z}$ followed by reduction of entries modulo p. A more general purpose BLAS interface for matrix arithmetic over finite fields is already described

[1] http://www.scg.uwaterloo.ca/~z4chen/iml.html

ISSAC'05, July 24–27, 2005, Beijing, China.

in [8], where a wider class of fields (i.e., larger, nonprime) are handled efficiently by using various techniques (e.g., lazy reduction, blocking to preserve exactness, alternative field representations). See also [6, 9].

Nonsingular solving. The goal is to solve a system $Ax = b$, where $A \in \mathbb{Z}^{n\times n}$ is nonsingular and $b \in \mathbb{Z}^{n\times 1}$. A key feature of this problem is expression swell. We expect the unique solution vector $A^{-1}b \in \mathbb{Q}^{n\times 1}$ to require about as much space to write down as the entire input system.

The three main computations in the variation of p-adic lifting that we use are: (i) a matrix inverse modulo p, $\log p = O(\log n + \log ||A||)$; (ii) $O(n)$ matrix-vector products with entries of bitlength about $\log p$; (iii) $O(n)$ operations with integers of bitlength about $n \log p$. Stage (iii) is accomplished with GMP. Our implementation reduces stages (i) and (ii) to Level 3 BLAS *Gemm* (word-size matrix-matrix multiplication) and Level 2 BLAS *Gemv* (word-size matrix-vector multiplication) by working over a well chosen residue number system. For the practitioner and implementor, the most important section of this paper is Section 3, where all the various optimization techniques used in our p-adic solver are described.

The goal of our implementation is both performance and generality. For example, transposed systems $A^t x = b^t$ are directly handled, the right hand side b may be multiple columns, and the integers in A and b may be arbitrarily large. However, if b is a single column and/or entries in A are all word-size (i.e., 32 or 64 bit) then the solver is optimized to take advantage.

All timings we give in this paper use the same environment: Intel® Itanium® 2 @ 1.3GHz, 50Gb RAM, Intel® C++ compiler 8.0, GMP v. 4.1.3, ATLAS v. 3.6.0. Our code will solve a system of dimension 500, 1000 and 2,000 with single decimal digit entries in about 1.2 seconds, 7 seconds, and 42 seconds (see Table 2). While integers in these input systems were chosen randomly in the range $-7 \ldots 7$, the numerators and denominators of entries in the solution vectors have 141, 1904, and 4110 decimal digits, respectively. Our implementation is designed for dense matrices, and doesn't take advantage of any structure or sparseness that may be present in the input system.

Certified solving. A reduction of Diophantine solving to nonsingular solving is given in [13]; an integer solution (if one exists) of a system $Ax = b$, where $A \in \mathbb{Z}^{n\times m}$ has arbitrary shape and rank profile, can be computed with high probability by combining a few random rational solution vectors. The ideas in [13] are taken up in [20] and extended in [21] to get an algorithm for *certified solving*: compute either a minimal denominator solution or prove that the system is inconsistent. Section 6 describes our new algorithm for the certified solving problem; the definition of certified solving is recalled at the start of the section.

Suppose for now that $A \in \mathbb{Z}^{n\times m}$ has full row rank. On the one hand, the approach of [21] is to solve multiple nonsingular systems of the form $ABx = b$, where $B \in \mathbb{Z}^{m\times n}$ is chosen randomly. On the other hand, the algorithm we describe here computes a single compression of the form $C = AB$, for random $B \in \mathbb{Z}^{m\times(n+k)}$, and then computes a certified solution of the compressed system $Cx = b$. The matrix C can be written as $C = [C_1 \,|\, C_2]$ where C_1 is nonsingular and the column dimension k of C_2 is a small constant. Thus, the compressed system is almost square. Section 4 sketches our (lengthy) proof that the compression will be successful with high probability even when $k = 25$ (i.e., that the compressed system $Cx = b$ has the same minimal denominator as the original system $Ax = b$). Section 5 gives a deterministic algorithm for computing a certified solution of the compressed system from the single nonsingular system solution $C_1^{-1}[C_2 \,|\, b]$, which is then easily extended to get a solution of the original system. The nonsingular system solution $C_1^{-1}[C_2 \,|\, b]$ is computed using the fast nonsingular solver discussed above. Asymptotically, the expected cost of our certified solver is the same as the algorithm in [21], but there are two important advantages in practice.

First, the approach in [21] needs to solve multiple systems of the form $ABy = b$, for a sequence of random dense matrices B. The algorithm here solves a single nonsingular system $C_1 y = [C_2 \,|\, b]$ with right hand side constant number of columns. It is considerably more efficient to solve one nonsingular system with right hand side k columns than to solve k different nonsingular systems with right hand side one column. Just consider that only one as opposed to multiple matrix inverses modulo p need be computed.

Second, lattice reduction may be used to optionally reduce the bitlength of numerators in the solution vector. Our algorithm for solving the compressed system $Cx = b$ recovers at the same time an integer right kernel basis for C which has dimension k (e.g., $k = 10$). This kernel can be used to reduce the solution size using lattice basis reduction, based on an idea described in [25, Page 381]. Although a reduction is not guaranteed, this method works well in practice. For example, consider a system $Ax = b$ with A of dimension 1000×2000 and all entries in A and b chosen randomly in the range $-7 \ldots 7$. Without lattice reduction, a particular (in this case Diophantine) solution is computed in 52s and has entries with about 3097 decimal digits. With basis reduction the time increases to 89s but entries in the improved solution have only 311 decimal digits. The key to this approach is that lattice reduction is applied to a vector space of dimension k (in this example $k = 10$). Applying the idea in [25, Page 381] directly would require computing and reducing the right kernel of dimension 1000 of the original matrix A; we expect this (since the system is random) to produce a solution vector with very small entries (i.e., single decimal digit) but with current techniques the lattice reduction of this large kernel is prohibitively expensive in terms of both time and space.

For a matrix or vector A over $\mathbb{Z}$, we denote by $||A||$ the maximum magnitude of entries in A.

2. WORD-SIZE LINEAR ALGEBRA

Almost all vector space invariants of a given $A \in (\mathbb{Z}/(p))^{n\times m}$ can be recovered by computing an echelon transform: a structured nonsingular U and a permutation P such that UPA is in echelon form. The following example has rank profile $(1,4,5)$.

$$\overset{U}{\left[\begin{array}{c|c} U_1 & \\ \hline U_2 & dI_3 \end{array}\right]} \overset{PA}{\left[\begin{array}{c} **********\\ **********\\ **********\\ \hline **********\\ **********\\ ********** \end{array}\right]} = \overset{R}{\left[\begin{array}{c} 1** \quad *** \\ \quad 1 \quad **** \\ \quad\quad 1**** \\ \hline \end{array}\right]}$$

A recursive reduction of echelon transform computation to matrix multiplication is described in [27, Section 2.1]. We have implemented an optimized version of the echelon transform algorithm in IML. The functionality is also available in Maple® 10 as the command `Modular[RowEchelonTransform]`.

All matrix multiplications arising in the algorithm have inner dimension at most $\lceil n/2 \rceil$, and thus can be performed directly with *Gemm* provided that $\lceil n/2 \rceil (p-1)^2 + (p-1) \leq 2^{53} - 1$. The additive term $(p-1)$ arises because the algorithm takes advantage of an extra matrix addition operation allowed to be performed by *Gemm*.

On the one hand, two $n \times n$ matrices can be multiplied with $2n^3 - n^2$ arithmetic operations. On the other hand, the computation of the inverse of A using the echelon transform also has cost

$2n^3 + O(n^2)$ arithmetic operations. Of course, the real advantage of using a recursive reduction to matrix multiplication is that we can exploit Level 3 BLAS, which is typically orders of magnitude faster than an equivalent computation using Level 2. To evaluate our implementation in IML we use the *ratio test*: the time for an invariant computation (e.g., inverse or determinant) of an $n \times n$ matrix is divided by the time for computing the product of two matrices. Table 1 shows some timings for a call to $Gemm(A,A)$ for computing AA for different dimensions of A. In the same table, we show the ratios of the times to compute $A^{-1} \bmod p$ and $\det A \bmod p$ with the time for the call to $Gemm(A,A)$. For dimension five thousand an inverse computation takes about one and a half times as long as a matrix multiplication, while the determinant can be computed in about half the time. Similar results are obtained for the computation of rank, nullspace, and reduced row echelon form. We conclude that the echelon transform gives a highly effective (in practice) reduction of matrix (vector space) invariant computation to matrix multiplication.

Dimension	$Gemm(A,A)$	$\frac{A^{-1} \bmod p}{Gemm(A,A)}$	$\frac{\det A \bmod p}{Gemm(A,A)}$
300	0.01 s	5	2
500	0.06 s	3	1.33
1000	0.46 s	2.3	0.98
2000	3.58 s	1.9	0.74
3000	11.56 s	1.7	0.66
5000	0.89 m	1.4	0.54

Table 1: Ratio test for word-size inverse and determinant.

3. NONSINGULAR SOLVING

Let a nonsingular matrix $A \in \mathbb{Z}^{n \times n}$ and a right hand side $b \in \mathbb{Z}^{n \times m}$ be given. The first phase of p-adic lifting is to compute the inverse of A modulo p, where $p \perp \det A$.

(i) $B := \mathrm{mod}(A^{-1}, p)$;

The second phase computes vectors c_i such that $\mathrm{mod}(A^{-1}b, p^{i+1}) = c_0 + c_1 p + \cdots + c_i p^i$ for $i = 0,1,\ldots,k-1$, k to be determined.

(ii) $r := b$;
for i **from** 0 **to** $k-1$ **do**
  $c_i := \mathrm{mod}(B \bmod(r,p), p)$;
  $r := (r - Ac_i)/p$
od;

Hadamard's inequality bounds the numerators and denominators of entries in $A^{-1}b$ by $N := \lfloor n^{n/2} \|A\|^{n-1} \|b\| \rfloor$ and $D := \lfloor n^{n/2} \|A\|^n \rfloor$, respectively. Thus, if k is chosen to satisfy $p^k > 2ND$ we can recover $A^{-1}b \in \mathbb{Q}^{n \times 1}$ using radix conversion and rational number reconstruction, see [12, Section 5.10].

(iii) $c := c_0 + c_1 p + \cdots + c_{k-1} p^{k-1}$; # radix conversion
$x := \mathrm{RatRecon}(c, N, D, p^k)$;

Phase (iii) is accomplished using GMP. The optimized implementation of phase (iii) will be discussed later. First consider phases (i) and (ii). Our implementation reduces all of the matrix arithmetic in these stages to a number of calls to the highly efficient BLAS routines *Gemm* (matrix-matrix products) and *Gemv* (matrix-vector products). This is accomplished by working in a residue number system. On the one hand, in order to be able to apply *Gemm* and *Gemv* directly we need to work with moduli that are bounded in magnitude by B, where $n(B-1)^2 \le 2^{53} - 1$, or, equivalently, $B \le 2^t$ where $t := \log_2(1 + \sqrt{(2^{53}-1)/n})$. On the other hand, to maximize the amount of work in each BLAS call the moduli should be chosen as large as possible. For $n = 20{,}000$ we have $t > 19.35$, so for all practical purposes there will be sufficiently many distinct primes in the range $2^{t-1} \ldots 2^t$. Thus, we may assume that all our moduli are about t bits in length.

The obvious approach is to choose p to be a single t-bit prime such that $p \perp \det A$ (such a prime can be found easily with a random choice.) Then stage (i) is accomplished with a single application of the echelon transform algorithm described in Section 2. Moreover, each computation of c_i in stage (ii) can be accomplished with a single call to *Gemv*.[2] Since $\|Ac_i\| \le n\|A\|\|c_i\| < n\|A\|(p-1)$, the computation of Ac_i in stage (ii) can be accomplished by working over a residue number system with about $1 + (\log_2 n + \log_2 \|A\|)/t$ moduli. Each iteration of the loop in stage (ii) makes t bits of progress (the bitlength of p), so the number of calls to *Gemv* per t bits of progress is about $2 + (\log_2 n + \log_2 \|A\|)/t$. Experiments with the approach just described revealed that much more time is spend in phase (ii) than in phase (i). This is not unexpected since a single matrix-matrix multiplication using Level 3 BLAS is orders of magnitude faster than n matrix-vector multiplications using Level 2 BLAS.

To better balance the cost between phases (i) and (ii) we modify the above approach by choosing p to be a multiple of t-bit primes: $p = p_1 p_2 \cdots p_l$, l to be determined. Then the computation of c_i in phase (ii) requires l calls to *Gemv*. Computing the updated r in phase (ii) by first computing Ac_i requires a residue number system with about $l + (\log_2 n + \log_2 \|A\|)/t$ primes. But we can optimize this by taking advantage of the fact that $(r - Ac_i)/p$ is integral. Using GMP, compute the division with remainder of each entry in r by p to obtain $r = pQ + R$, where $\|R\| < p$. Then $(r - Ac_i)/p$ is equal to $Q + (R - Ac_i)/p$. Since $\|(R - Ac_i)/p\| \le 1 + n\|A\|$, we can compute $(R - Ac_i)/p$ directly in a residue number system with only about $(\log_2 n + \log_2 \|A\|)/t$ moduli. This optimized approach requires only $l + (\log_2 n + \log_2 \|A\|)/t$ calls to *Gemv* per lt bits of lifting progress, compared to the previous estimate of $2l + (\log_2 n + \log_2 \|A\|)l/t$ calls. Of course, as l increases the cost of phase (i) increases, since l inverses need to be computed. Therefore, we need to find a balance between these two concerns. Our experiments suggest the choice $l \approx 2(\log n + \log_2 \|A\|)$ if entries in A are small and can fit into signed long. Otherwise, we choose $l \approx (\log_2 n + \log_2 \|A\|)$. In the next two subsections we discuss two more optimizations, aimed at minimizing the cost of phase (iii) and minimizing the number of lifting steps in phase (ii).

3.1 Fast solution reconstruction

The first part of phase (iii) is to compute the sum $c = c_0 + c_1 p + \cdots + c_{k-1} p^{k-1}$. Initially we used a Horner scheme but discovered this was too slow. Instead, we implemented recursive algorithm for the radix conversion which incorporates large integer multiplication (see [12, Exercise 9.20]) to take better advantage of GMP. The next step is rational reconstruction. The goal is to compute a common denominator d such that $d < D$ and $\|\mathrm{mod}(dc, p^k)\| < N$. The naive approach to accomplish this is to apply rational reconstruction on each entry of c independently and then set d to be the lcm of the n denominators. In practice, this is slow since each entry in the solution vector typically has denominator a large factor of d (in many cases equal to d.) In particular, GMP doesn't currently offer an optimized function for rational number reconstruction. Our implementation[3] is very sensitive to the size of the denominator.

[2] We will assume here that the right hand side has column dimension $m = 1$. If $m > 1$ then a single call to *Gemm* should be used.
[3] We implemented the standard iterative algorithm based on the ex-

Instead, we use a trick that is implemented in Shoup's NTL library [24]. Initialize $d := 1$, and for $i = 1,2,\ldots,n$ reconstruct the ith entry of the solution vector as follows. Let e denote the ith entry in c. Compute $\bar{e} := \mathrm{mod}(de, p^k)$ and then apply rational number reconstruction on $\bar{e}$ to get $n/\bar{d}$. Then $n/(d\bar{d})$ is the ith entry in the solution vector. Update d to be $d := d\bar{d}$ and proceed to the next i. At the end d will be the lcm of all the denominators. The advantage of this approach is that the product of all n denominators being reconstructed is equal to d, instead of being as large as d^n. Thus, the net cost of all n calls to the rational number reconstruction routine is about the same as one call with a number having denominator d.

We mention an anecdotal timing result. For a nonsingular system with dimension 3000 and single decimal digit entries, the new technique reconstructs the solution within five seconds. Applying rational reconstruction to each entry independently used twenty-five minutes. Thus, a three hundred times speedup is gained. The reason for this is that the GMP library provides highly optimized multiplication and division subroutines (i.e., for the computation of $\bar{e} := \mathrm{mod}(de, p^k)$).

3.2 Output sensitive lifting

In the worst case, the number of lifting steps k needs to be chosen satisfying $p^k > 2ND$ in order to reconstruct the solution correctly, where $N = \lfloor n^{n/2}\|A\|^{n-1}\|b\|\rfloor$ and $D = \lfloor n^{n/2}\|A\|^n\rfloor$ come from Hadamard's bound. However, Hadamard's bound is often a pessimistic [2] and we can take advantage of this by performing the lifting in an output sensitive manner.

1. Initialize k_0 to be zero and k to be a small positive integer (e.g., $k = 10$).
2. Perform $k - k_0$ iterations in phase (ii) to compute the lifting coefficients $c_{k_0}, \ldots, c_{k-1}$.
3. Compute $c := c_0 + c_1 p + \ldots + c_{k-1}p^{k-1}$.
4. Use the rational reconstruction scheme of the previous section to attempt to compute a common denominator d and $y := \mathrm{mod}(dc, p^k)$ such that $\max(d, \|y\|) < \lfloor\sqrt{p^k/2}\rfloor$.
5. If the reconstruction succeeds and $Ay = db$ then return y/d. Otherwise, assign k to k_0 and increase k by a small positive integer[4] and goto step 2.

In our implementation we optimize the above scheme by making the following two changes. First, we merge steps 3 and 4. As soon as we compute the ith entry e of c using radix conversion, we perform the rational reconstruction on e before starting to compute the $(i+1)$st entry of c. If the reconstruction of e fails, we avoid computing entries $i+1,\ldots,n$ of c. In practice, if k is not large enough a failure is reported quickly. Second, instead of assaying correctness of y in step 5 with an expensive matrix-vector multiplication (note that entries in y may have bitlength n times larger than entries in A) we verify the solution by checking the magnitude bound according to the following lemma, due to [3].

LEMMA 1. *If* $d\|b\|, n\|A\|\|y\| < (p^k - 1)/2$ *then* $Ay = bd$.

3.3 Timings

Table 2 gives some timings to solve the nonsingular system $Ax = b$ for different dimensions of A, where b is a vector and entries of A and b are relatively small. Although not shown in the table, we point out that the time spent for the matrix inverse computations is almost the same as the time spent for the lifting iteration, due to our optimized choice of size of the lifting basis. A more detailed breakdown of the timing can be found in [4].

Dimension	Time	Digits	Dimension	Time	Digits
100	0.2 s	141	3500	3.1 m	7620
500	1.2 s	877	4000	4.7 m	8821
1000	6.6 s	1905	4500	6.5 m	10040
1500	19 s	2991	5000	9.5 m	11271
2000	42 s	4110	6000	14.7 m	13761
2500	1.3 m	5257	7000	23.4 m	16288
3000	2.2 m	6430	8000	35.4 m	18849

Table 2: Timings to solve a nonsingular system $Ax = b$, where b is a vector and entries of A and b are randomly chosen in the range $-7\ldots7$. The Digits column is the number of decimal digits in entries of the solution vector.

Dimension	IML	NTL	Maple
200	0.2 s	0.1 s	5.8 s
500	1.5 s	17 s	87 s
1000	6.6 s	37 s	23 m
2000	42 s	5.6 m	5.7 h

Table 3: Comparison of timings to solve a nonsingular system $Ax = b$ using IML *NonSingSolve*, NTL *solve1*, and Maple `Modular[LinIntSolve]`. Entries in A and column vector b are randomly chosen between -7 and 7.

Table 3 compares our implementation with function *solve1* in NTL [24] v. 5.3.2 and `Modular[LinIntSolve]` in Maple® v. 9.5. NTL also makes full use of GMP for the large integer arithmetic, but except for the fast rational reconstruction technique described in Section 3.1, the implementation in NTL doesn't incorporate the optimizations described in this section. The speedup of IML over NTL is obtained in large part because NTL is not using the ATLAS/BLAS library to perform the matrix-vector and matrix-matrix multiplications. The solver in Maple is actually BLAS based but uses a homomorphic imaging scheme instead of lifting.

Giorgi [15] has implemented a BLAS based version of lifting for nonsingular solving in the LinBox[5] library [7]. For input matrices with small entries (i.e., in the range $-7\ldots7$) the timings obtained are very similar to those in Table 2. For matrices with larger entries the IML implementation using multiple primes for lifting gives an improvement (e.g., the solution to a 2000×2000 input matrix with 100 bit entries takes about twice as long in LinBox as with IML).

Finally, we mention a technical optimization based on a feature of BLAS. If the right hand side b has column dimension $m > 1$, the computation of r and c_i in phase (ii) of the algorithm should make use of single calls to *Gemm* instead of m calls to *Gemv*. For example, Table 4 shows that solving a system with right hand side ten columns takes less than five times as long as solving a system with right hand side a single column.

4. LATTICE COMPRESSION

Let $A \in \mathbb{Z}^{m\times n}$ with full row rank m be given, $n > m$. Let $B \in \mathbb{Z}^{n\times(m+k)}$, with entries chosen uniformly and randomly from a finite set $\Lambda = \{0,1,\ldots,\lambda\}$, where $\lambda \geq 2$. We call the action of post-

tended euclidean scheme, see [12, Section 9.10].

[4]Our implementation uses $k := k + 10$, which has not been optimized.

[5]`http://www.linalg.org`

Dimension	$m=1$	$m=10$	Ratio
500	1.2 s	4.9 s	4.1
2000	42.0 s	3.4 m	4.9
4000	4.7 m	22.3 m	4.7

Table 4: Comparison of timings to solve a nonsingular system $Ax=b$ with $m=1$ and $m=10$, where m is the column dimension of b and the entries of A and b are randomly chosen between -7 and 7.

multiplication of A by B *lattice compression*: $C=AB$ where C has k more columns than rows. Let $A \equiv_R C$ denote that A and C have the same Hermite column basis (i.e., the set of all $\mathbb{Z}$-linear combinations of columns of A is equal to the set of all $\mathbb{Z}$-linear combinations of columns of C). In this section we sketch the result in [4] which gives a lower bound on the probability that $A \equiv_R C$; the bound exponentially converges to one with respect to k. The lattice compression technique will be used by the algorithm in next section.

The strongest form of lattice compression has $k=0$. For example, let V be a unimodular matrix ($\det V = \pm 1$) such that all but the first m columns of AV are zero: $AV = \left[\begin{array}{c|c} C & \\ \end{array}\right]$, where $C \in \mathbb{Z}^{m\times m}$ is necessarily nonsingular. If we take B as the submatrix of V comprised of the first m columns, then $C=AB$ and $A \equiv_R C$. However, the most efficient algorithm to compute such a V is too expensive by factor of m and, moreover, guarantees only the bound $\|B\| \leq m^{3m/2+3}\|A\|^{3m}$ (see [27, Proposition 8.10]) which causes this approach to be very inefficient. Alternatively, by choosing $B \in \mathbb{Z}^{n\times(m+k)}$ to be a random matrix with k a small constant, we have $A \equiv_R C$ with high probability.

THEOREM 2. *Let $A \in \mathbb{Z}^{m\times n}$ be given, where $\mathrm{rank}(A)=m$ and $n>m$. Let $B \in \mathbb{Z}^{n\times(m+k)}$ have entries uniformly and independently chosen at random from $\Lambda=\{0,1,\ldots,\lambda-1\}$. If $\lambda \geq \max(m+1, \log_2 n, \log_2\|A\|)$, then the probability that $A \equiv_R AB$ is at least $1-16(1/2)^{k/5}$. In particular, the probability is $1/2$ if $k=25$.*

Now we give the idea of our proof for Theorem 2. Refer to [4] for the complete proof. Let $C=AB$. To bound the probability of $A \equiv_R C$, an equivalent conversion is to assume that $A \equiv_R I_m$ and to bound the probability that $C \equiv_R I_m$. Since $A \equiv_R I_m$ if and only if for all primes p, $A \bmod p$ has full row rank over $\mathbb{Z}/(p)$, our goal is to derive a lower bound on the probability that $C \bmod p$ has full row rank over $\mathbb{Z}/(p)$ for all primes p.

Let $N \in \mathbb{Z}^{n\times(n-m)}$ denote a right kernel for A. Then $N \equiv_L I_{n-m}$ (i.e., N has the Hermite row basis an identity matrix). Thus, for any prime p, $N \bmod p$ has full column rank over $\mathbb{Z}/(p)$. From [21, Lemma 15] we obtain the following.

LEMMA 3. *Let $A \in \mathbb{Z}^{m\times n}$ and $B \in \mathbb{Z}^{n\times l}$ be given. Let N be a right kernel for A. Then for any prime p, $\mathrm{rank}(AB \bmod p) = \mathrm{rank}(\left[\begin{array}{c|c} N & B \end{array}\right] \bmod p) - \mathrm{rank}(N \bmod p)$ over the finite field $\mathbb{Z}/(p)$.*

Lemma 3 implies that $C \bmod p$ has full row rank m over $\mathbb{Z}/(p)$ for any prime p if and only if $\left[\begin{array}{c|c} N & B \end{array}\right] \bmod p$ has full row rank n over $\mathbb{Z}/(p)$. Since $\left[\begin{array}{c|c} N & B \end{array}\right]$ is equivalent to its Smith form, $\left[\begin{array}{c|c} N & B \end{array}\right] \bmod p$ has full row rank over $\mathbb{Z}/(p)$ for any prime p if and only if all the diagonal entries in the Smith form of $\left[\begin{array}{c|c} N & B \end{array}\right]$ are equal to 1.

Partition B as $B = \left[\begin{array}{c|c|c} B_1 & B_2 & B_3 \end{array}\right]$ and introduce two indeterminants t_1 and t_2 such that the column dimensions of B_1, B_2 and B_3 is $m-t_1, t_1+t_2$ and $k-t_2$ respectively. There exist unimodular matrices U and V which separately apply row operations and column operations on $\left[\begin{array}{c|c} N & B \end{array}\right]$ and transform the submatrix $\left[\begin{array}{c|c} N & B_1 \end{array}\right]$ to Smith form. Such U and V can be partitioned using a conformal block decomposition as

$$\left[\begin{array}{c} U \\ \hline U_1 \\ \hline U_2 \end{array}\right] \left[\begin{array}{c|c|c|c} N & B_1 & B_2 & B_3 \end{array}\right] \left[\begin{array}{c|c|c|c} * & * & & \\ \hline * & * & & \\ \hline & & I_2 & \\ \hline & & & I_3 \end{array}\right] = \left[\begin{array}{c|c|c|c} 1 & & & \\ & \ddots & & \\ & & s_1 & \\ & & & \ddots \\ & & & s_{m-t_1} \\ \hline & & & \end{array}\right] \left[\begin{array}{c|c} * & * \\ \hline & \\ \hline S_2 & S_3 \end{array}\right] \quad (1)$$

where $\left[\begin{array}{c|c} S_2 & S_3 \end{array}\right] = U_2 \left[\begin{array}{c|c} B_2 & B_3 \end{array}\right]$ and the $(t_1+1)\times n$ submatrix U_2 is in the nullspace for N. The first $n-m$ invariant factors are 1 since $N \equiv_L I_{n-m}$.

Having set up the transform as above, we can see that a sufficient condition for all the diagonal entries of the Smith form of $\left[\begin{array}{c|c} N & B \end{array}\right]$ to equal one is that $s_1 = s_2 = \ldots = s_{m-t_1-1} = 1$ and $\left[\begin{array}{c|c} S_2 & S_3 \end{array}\right] \equiv_R I_{t_1+1}$. The first step of our proof is to bound the probability that $s_1 = \ldots = s_{m-t_1-1} = 1$ using the technique from [10, Section 6]. A problem of adapting the technique is that it can only provide a useful bound on the probability that $s_1, \ldots, s_{m-t_1-1}$ are equal to one, without determining the value of s_{m-t_1}. However, if $\left[\begin{array}{c|c} S_2 & S_3 \end{array}\right] \equiv_R I_{t_1+1}$, then all the entries on the diagonal of the Smith form of $\left[\begin{array}{c|c} N & B \end{array}\right]$ are necessarily 1. So, the second step of our proof is to bound the probability that $\left[\begin{array}{c|c} S_2 & S_3 \end{array}\right] \equiv_R I_{t_1+1}$ using the results from [21, Section 3]. In the final step, we derive the result of Theorem 2 by fixing the values of t_1 and t_2 and combining the previous two results.

5. CERTIFIED SOLVING WITH KERNEL BASIS

In this section we present a deterministic algorithm to certified solve a linear system $Cx=b$. We consider a special case of the problem: we assume that C may be decomposed as $C = \left[\begin{array}{c|c} A & B \end{array}\right] \in \mathbb{Z}^{n\times(n+k)}$ where $A \in \mathbb{Z}^{n\times n}$ is nonsingular. The algorithm here is applicable for arbitrary B, but is designed for the case when the column dimension k of B is small (e.g., $k=O(1)$). In Section 6 we show how to use the lattice compression technique of Section 4 to certified solve an input system with arbitrarily column dimension by reducing to the special case we describe here, with $k=25$.

Before describing the algorithm we recall the difference between a right nullspace (over $\mathbb{Q}$) and right kernel (over $\mathbb{Z}$). The matrix C is over the principal ideal domain $\mathbb{Z}$, but we may also consider C to be over the field $\mathbb{Q}$. On the one hand, a matrix $N \in \mathbb{Q}^{(n+k)\times k}$ is a right nullspace for C over $\mathbb{Q}$ if $\mathrm{rank}(N)=k$ and $CN=0$. On the other hand, for a matrix $K \in \mathbb{Z}^{(n+k)\times k}$ to be a right kernel for C over $\mathbb{Z}$ the following additional condition must be satisfied: every integer vector in the right nullspace of C must be generated by an integer linear combination of columns of K. Computing a right kernel is a more subtle problem than computing a right nullspace. For example, scaling a nullspace by multiplying by the least common multiple of the denominators of entries will produce a nullspace with integer entries, but this is unlikely to be a kernel.

The first step of our algorithm is to compute $A^{-1}\left[\begin{array}{c|c} B & b \end{array}\right]$. Then the algorithm constructs a right kernel $K \in \mathbb{Z}^{(n+k)\times k}$ for C over $\mathbb{Z}$,

a minimal denominator solution $y \in \mathbb{Q}^{(n+k)\times 1}$ such that $Cy = b$, and a certificate vector $z \in \mathbb{Q}^{1\times n}$ which proves the minimality of the denominator of y. For clarity, we first consider the computation of only K, then show how extend the method to compute y and z. For our size estimates in this section we assume that $\log\|b\| = O(n(\log n + \log\|C\|))$.

Finally, we recall a technique for optionally reducing the the solution using lattice reduction.

Constructing a right kernel K

We will compute K to be the last k columns of the unique unimodular matrix of dimension $n+k$ such that

$$\left[\begin{array}{c|c} A & B \\ \hline & I_k \end{array}\right] \overset{K}{\left[\begin{array}{c|c} * & * \\ \hline * & * \end{array}\right]} = \left[\begin{array}{c|c} * & \\ \hline * & H \end{array}\right],$$

where the matrix on the right hand side is in column Hermite form. Solving gives $K = \text{StackMatrix}(-A^{-1}BH, H)$. We now show how to construct H efficiently from $A^{-1}B$. Let s be the denominator of $A^{-1}\left[\begin{array}{c|c} B & b \end{array}\right]$ and set $N = \text{StackMatrix}(-sA^{-1}B, sI_k) \in \mathbb{Z}^{(n+k)\times k}$. Then N is a basis for the right nullspace of C over $\mathbb{Q}$. The algorithm of [17] can be used to compute the upper triangular row Hermite basis $T \in \mathbb{Z}^{k\times k}$. The algorithm is easily modified to compute, instead of T, a left equivalent matrix $\bar{T} \in \mathbb{Z}^{k\times k}$ that is lower triangular with off-diagonal entries in each column reduced modulo the positive column leader (just an alternative definition of the Hermite row basis). This algorithm uses $O(nk^2)$ integer operations and takes advantage of the rows sI_k to keep all intermediate entries during the computation of T and $\bar{T}$ reduced modulo s. Then $N\bar{T}^{-1}$ is also a right kernel of C over $\mathbb{Z}$, so the last k rows of K and $N\bar{T}^{-1}$ are right equivalent (i.e., H is the column Hermite form of $s\bar{T}^{-1}$). The following code fragment shows how to compute H from $\bar{T}$, keeping all off-diagonal entries in H reduced modulo s.

$H := sI_k$;
for i **from** 1 **to** k **do**
  for j **from** 1 **to** $i-1$ **do**
    for l **from** 1 **to** j **do**
      $H_{il} := (H_{il} - \bar{T}_{ij}H_{jl}) \bmod s$
    od
  od;
  for j **from** 1 **to** i **do**
    $H_{ij} := H_{ij}/\bar{T}_{ii}$
  od
od

Once H has been recovered, compute $K := NH(1/s)$.

By Hadamard's bound and Cramer's rule, all of s and $\|N\|$ are bounded in length by $O(n(\log n + \log\|C\|))$ bits. Assuming $A^{-1}B$ has already been computed, the dominant step is to compute $\bar{T}$ at a cost of $O(nk^2)$ operations with integers bounded in absolute value by s.

Constructing a minimal denominator solution y

Now we extend the algorithm just described to compute a minimal denominator solution y. Recall that $N = \text{StackMatrix}(-sA^{-1}B, sI_k)$ is a basis for the right nullspace of C. A particular solution of $Cx = b$ is $x_1 := \text{StackMatrix}(A^{-1}b, 0) \in \mathbb{Q}^{(n+k)\times 1}$, and the set of all rational solutions is $\{x_1 + Nv \mid v \in \mathbb{Q}^{n\times 1}\}$. Our goal is to find a $v \in \mathbb{Q}^{k\times 1}$ such that $x_1 + Nv$ has minimal denominator. Let

$$\left[\begin{array}{c|c} N & sx_1 \\ \hline & s \end{array}\right] \text{ have Hermite form } \left[\begin{array}{c|c} T & c \\ \hline & e \end{array}\right]. \tag{2}$$

Two rational matrices that are left equivalent necessarily have the same denominator. Thus, our goal is to find a $v \in \mathbb{Q}^{k\times 1}$ such that the vector

$$\left[\begin{array}{c|c} N & sx_1 \\ \hline & s \end{array}\right] \left[\begin{array}{c} sv \\ \hline 1 \end{array}\right] \frac{1}{s}, \text{ or, equivalently, } \left[\begin{array}{c|c} T & c \\ \hline & e \end{array}\right] \left[\begin{array}{c} sv \\ \hline 1 \end{array}\right] \frac{1}{s} \tag{3}$$

has minimal denominator. The equivalent presentation in (3), together with the choice $v = -T^{-1}c(1/s)$, reveals that the minimal denominator is s/e. But since sT^{-1} is integral, we can choose $v = (-sT^{-1}c \bmod se)(1/s^2)$, giving the minimal denominator solution $y = (s^2x_1 - N\bar{v})/s^2$, where $\bar{v} = -sT^{-1}c \bmod se \in \mathbb{Z}^{k\times 1}$ can be computed by first computing $sT^{-1} \bmod se$ using similar method as used for the computation of H from $s\bar{T}^{-1} \bmod s$. The cost estimate remains the same as before.

Constructing a certificate z

Finally, we show how to construct a certificate vector $z \in \mathbb{Q}^{1\times n}$ such that zC is integral and zb has denominator s/e. Our approach is to construct a $q \in \mathbb{Z}^{1\times n}$ such that $qA^{-1}B$ is integral and $qA^{-1}b$ has denominator s/e, and then set $z := qA^{-1}$. Let u be equal to row $k+1$ of a unimodular matrix that transforms the first matrix in (2) to it's Hermite form. Let q be the vector comprised of the first n entries of u reduced modulo s, so that $u = \left[\begin{array}{c|c|c} q & * & \end{array}\right] \bmod s$. Recall that $N = \text{StackMatrix}(-A^{-1}B, sI_k)$ and $x_1 = \text{StackMatrix}(A^{-1}b, 0)$. Then

$$\left[\begin{array}{c|c|c} q & * & * \end{array}\right] \left[\begin{array}{c|c} -sA^{-1}B & sA^{-1}b \\ \hline sI_k & \\ \hline & s \end{array}\right] = \left[\begin{array}{c|c} & e \end{array}\right] \bmod s$$

which shows that q is as desired.

It remains to show how to compute u, row $k+1$ of a unimodular matrix effecting the transformation in (2). The transformation to Hermite form can be accomplished with a sequence of $O(nk)$ unimodular transformations, each transformation on one or two rows (cf. [27, page 55]). Multiplying these transformations together would yield a unimodular matrix $U = U_{O(nk)} \cdots U_3U_2U_1$, but this would be too expensive ($O(n^2k)$ arithmetic operations) and the U produced might be dense, with $O(n^2)$ nonzero entries. Instead, store all these transformations and then, working modulo s, apply them in reverse order to row $k+1$ of the identity matrix. As a result, u can be computed at the same time. Finally, compute $z := qA^{-1}$.

If $k = O(1)$ the nonsingular systems $A^{-1}\left[\begin{array}{c|c} B & b \end{array}\right]$ and qA^{-1} can be solved with $O(n^3(\log n + \log\|C\|)^2)$ bit operations (see [20]). Additional $O(n)$ operations with integers of bitlength $O(n(\log n + \log\|C\|))$ bits is required. This gives the following.

THEOREM 4. *Suppose* $\log\|b\| = O(n(\log n + \log\|C\|))$ *and* $k = O(1)$. *Given a prime* p *such that* $p = O(\log n + \log\|C\|)$ *and* $p \perp \det A$, *the algorithm described above computes* K, y *and* z *with* $O(n^3(\log n + \log\|C\|)^2)$ *bit operations.*

Incorporating lattice reduction

Let y be a minimal denominator solution for $Cx = b$ and K be a right kernel for C. If y and K are computed using the algorithm

supporting Theorem 4 the bitlength of entries will be $O(n(\log n + \log \|A\|))$. We can try to find a minimal denominator solution $\bar{y}$ with improved bitlength using the following approach described in [25, Page 381].

Use lattice basis reduction [18] to compute a reduced kernel $\bar{K}$. Then compute $u \in \mathbb{Z}^k$ so that the vector $\mathrm{d}(y)\bar{y} := \mathrm{d}(y)y + Ku$ is *size reduced* with respect to the vectors in $\bar{K}$. In all our experiments with random matrices, this produces a minimal denominator solution $\bar{y}$ which has numerators with bitlength about a factor of k smaller than those in y.

The main cost of the above approach is to reduce the lattice K of dimension k in in $(n+k)$-dimensional space. This lattice has a very special shape since k is so small compared to $n+k$ (e.g., $k = O(1)$). Moreover, the lattice is very skew since the norms of column vectors in K are large (i.e., $O(n(\log n + \log \|A\|))$). Using directly the LLL algorithm [18] would be too expensive. Instead, we use the modification in [26] that works in three stages: (1) compute $K^T K \in \mathbb{Z}^{k \times k}$; (2) compute, modulo an integer $M = O(kn\|K\|)$, a unimodular matrix $U \in \mathbb{Z}^{k \times k}$; (3) set $\bar{K} = KU \bmod M$. Step (2) dominates the cost with $O(k^4 n(\log n + \log A))$ arithmetic operations involving integers bounded in length by $O(kn(\log n + \log \|A\|))$ bits. The algorithm in [22], which is less sensitive to $\|K\|$, may also be well suited for this task.

6. CERTIFIED LINEAR SYSTEM SOLVING

Let $A \in \mathbb{Z}^{n \times m}$ and $b \in \mathbb{Z}^{n \times 1}$ be given. We first recall the properties of *minimal denominator*. Let $\mathrm{d}(y)$ denote the denominator of a solution vector y to the system $Ax = b$, where b can be a vector or a matrix. The set of denominators of all the solutions to the system $Ax = b$ generates an ideal I of $\mathbb{Z}$. Let $\mathrm{d}(A,b)$ denote the generator of the ideal I. Then $\mathrm{d}(A,b)$ divides all the elements in I and hence is the minimal denominator.

Our algorithms for *certified linear system solving* has the same functionality as the algorithms in [21], which take as input A and b and return as output one of the following:

1. (y,z), where
   - $y \in \mathbb{Q}^{m \times 1}$ with $Ay = b$,
   - $z \in \mathbb{Q}^{1 \times n}$ with $zA \in \mathbb{Z}^{1 \times m}$, and
   - zb and y have the same denominator.
2. ("no solution", q), where
   - $q \in \mathbb{Q}^{1 \times n}$ with $qA = (0,\ldots,0) \in \mathbb{Q}^{1 \times m}$ and $qb \neq 0$.

In the first case, y is a solution with minimal denominator and z serves as a certificate for the minimality of the denominator of y. The idea of minimal denominator certificate is a generalization of the integer version of Farkas' lemma in [11]. In the second case, vector q certifies that the system is inconsistent and has no rational solution. The idea for certifying inconsistency is due to [14]. Refer to [21] for explanations in detail.

There are three stages to our certified solver. The first is to either prove that the input system $Ax = b$ is inconsistent or to reduce to an equivalent system which has full row rank. Our implementation of this first stage is accomplished using a similar approach as in [21, *CertifiedSolver*, Page 506]. The description of this phase is omitted here since it is so similar. Henceforth we will assume that the input matrix $A \in \mathbb{Z}^{n \times m}$ has full row rank n.

The second stage is to compress the system $Ax = b$ with $A \in \mathbb{Z}^{n \times m}$ using lattice compression as described in Section 4. This give us an almost square system $Cx = b$ (e.g., $C = AB$ with $B \in \mathbb{Z}^{m \times (n+25)}$ chosen randomly). The matrix multiplication AB is accomplished by reducing to *Gemm* by using a residue number system. Note that if m is not too big with respect to n (e.g., $m \leq n+25$) then we can choose $B = I_m$ in which case $C = A$.

The third stage computes a certified solution $(\bar{y},z)$ to the compressed system $Cx = b$ using the algorithm described in the previous section. Using GMP, we assay if $(B\bar{y},z)$ is a certified solution to the original system $Ax = b$ by checking that zA is over $\mathbb{Z}$, and that zb and $B\bar{y}$ have the same denominator.

Dimension	Typical Solution		Reduced Solution	
$n \times m$	Digits	Time	Digits	Time
500×1000	1445	9 s	146	16 s
1000×2000	3097	52 s	311	89 s
3000×6000	10995	22 m	1102	33 m
6000×8000	23451	3.3 h	2347	4.8 h

Table 5: Timings to compute a minimal denominator solution to the system $Ax = b$, where $A \in \mathbb{Z}^{n \times m}$ and $b \in \mathbb{Z}^{n \times 1}$. Entries of A and b are randomly chosen between -7 and 7. The column Digits is the number of solution decimal digits.

k	Digits	Time
10	254	39 s
20	128	3.9 m
30	86	15 m
40	65	42 m
50	53	1.6 h

Table 6: Timings to compute a size-reduced minimal denominator solution to a system $Ax = b$ for different k, where k is the column dimension of the right kernel of the compressed system. Entries in $A \in \mathbb{Z}^{500 \times 1000}$ and $b \in \mathbb{Z}^{500 \times 1}$ are randomly chosen from -2^{10} to 2^{10}. The column Solution Digits is the number of decimal digits of the largest entry in the solution numerator.

6.1 Timings

The certified solver described in Section 5 and above was implemented in IML directly as described using GMP. All nonsingular solving uses the ATLAS/BLAS based algorithm described in Section 3. For the lattice compression phase we chose by default $\lambda = 2$ and $k = 10$ (the user has the option of choosing a larger k). Using this choice, the lattice compression $C = AB$ succeeded in all our experiments, and hence only a single nonsingular system $C_1^{-1} \left[\begin{array}{c|c} C_2 & b \end{array} \right]$ with right hand side eleven columns needed to be solved. Table 4 shows that the time for solving this system is only about five times as long as the time for solving a system with right hand side one column.

In Table 5 we give timings to compute a minimal denominator solutions to the system $Ax = b$ with entries in A and b randomly chosen. The table shows that the solution size can be approximately reduced by a factor of ten using lattice basis reduction. For a randomly chosen input system, we always observed this correspondence between the solution reduction and the kernel basis dimension. Table 6 considers the effect of increasing the value of k on the quality of the solution and the running time. Most of the increase in time is due to the lattice reduction phase.

Finally, we remark that an optimized BLAS based algorithm for diophantine solving has been implemented in LinBox [7]. See [15] for a description and detailed comparison with IML.

7. REFERENCES

[1] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '99*, pages 197–204. ACM Press, New York, 1999.

[2] J. Abbott and T. Mulders. How tight is Hadamard's bound? *Experimental Mathematics*, 10(3):331–336, 2001.

[3] S. Cabay. Exact solution of linear systems. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, pages 248—253, 1971.

[4] Z. Chen. A BLAS based C library for exact linear algebra on integer matrices. Master's thesis, School of Computer Science, University of Waterloo, 2005.

[5] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.

[6] J.-G. Dumas. Efficient dot product over word-size finite fields. *CoRR*, cs.SC/0404008, 2004.

[7] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and V. G. LinBox: A generic library for exact linear algebra. In A. J. Cohen and N. Gao, X.-S. andl Takayama, editors, *Proc. First Internat. Congress Math. Software ICMS 2002, Beijing, China*, pages 40–50, Singapore, 2002. World Scientific.

[8] J. G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '02*, pages 63–74. ACM Press, New York, 2002.

[9] J. G. Dumas, P. Giorgi, and C. Pernet. Finite field linear algebra package. In J. Gutierrez, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '04*, pages 119–126. ACM Press, New York, 2004.

[10] W. Eberly, M. Giesbrecht, and G. Villard. Computing the determinant and Smith form of an integer matrix. In *Proc. 31st Ann. IEEE Symp. Foundations of Computer Science*, pages 675–685, 2000.

[11] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.

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

[13] M. Giesbrecht. Efficient parallel solution of sparse systems of linear diophantine equations. In M. Hitz and E. Kaltofen, editors, *Second Int'l Symp. on Parallel Symbolic Computation: PASCO '97*, pages 1–10. ACM Press, New York, 1997.

[14] M. Giesbrecht, A. Lobo, and B. D. Saunders. Certifying inconsistency of sparse linear systems. In O. Gloor, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '98*, pages 113—119. ACM Press, New York, 1998.

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

[16] T. Granlund. The GNU multiple precision arithmetic library, 2004. Edition 4.1.4. `http://www.swox.com/gmp`.

[17] C. S. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM Journal of Computing*, 18(4):658–669, 1989.

[18] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[19] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proc. EUROSAM '79, volume 72 of Lecture Notes in Compute Science*, pages 65–72, Berlin-Heidelberg-New York, 1979. Springer-Verlag.

[20] T. Mulders and A. Storjohann. Diophantine linear system solving. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '99*, pages 281–288. ACM Press, New York, 1999.

[21] T. Mulders and A. Storjohann. Certified dense linear system solving. *Journal of Symbolic Computation*, 37(4):485–510, 2004.

[22] P. Nguyen and D. Stehlé. Floating-point LLL revisited. In *Proceedings of Eurocrypt '05*, 2005.

[23] D. Saunders and Z. Wan. Smith normal form of dense integer matrices, fast algorithms into practice. In J. Gutierrez, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '04*, pages 274–281. ACM Press, New York, 2004.

[24] V. Shoup. *NTL: A Library for Doing Number Theory*, 2005. `http://www.shoup.net/ntl/`.

[25] C. C. Sims. *Computation with finitely presented groups*, volume 48 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1984.

[26] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical Report 249, Departement Informatik, ETH Zürich, July 1996.

[27] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Swiss Federal Institute of Technology, ETH–Zurich, 2000.

[28] U. Vollmer. A note on the Hermite basis computation of large integer matrices. In R. Sendra, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '03*, pages 255–257. ACM Press, New York, 2003.

[29] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 27(1-2), 2001.