Algorithms for Solving Linear Systems over Cyclotomic Fields

Liang Chen and Michael Monagan

Department of Mathematics
Simon Fraser University
Burnaby, B.C. V5A 1S6, CANADA.

Abstract

We consider the problem of solving a linear system $Ax = b$ over a cyclotomic field. What makes cyclotomic fields of special interest is that we can easily find a prime p that splits the minimal polynomial $m(z)$ for the field into linear factors. This makes it possible to develop very fast modular algorithms.

We give two output sensitive modular algorithms, one using multiple primes and Chinese remaindering, and the other using linear $p-$adic lifting. Both of our algorithms use rational reconstruction to recover the rational coefficients in the solution vector. We have implemented both algorithms in Maple with key parts of the implementation implemented in C for efficiency. A complexity analysis and experimental timings both show that Chinese remaindering is competitive with $p-$adic lifting.

We also give a third algorithm which computes the solution x as a ratio of two determinants modulo $m(z)$ using Chinese remaindering. This representation is a factor of $d = \deg m$ more compact in general, and because of this, we can compute it the fastest in general.

1. Introduction

In this paper we consider the problem of how to efficiently solve a linear system $Ax = b$ over an algebraic number field $\mathbb{Q}(\zeta)$ where ζ is a k'th primitive root of unity. These number fields, which include the complex rationals, are called the cyclotomic fields. The minimal polynomial $m(z)$ for ζ is $\Phi_k(z)$, the k'th cyclotomic polynomial. This is a monic irreducible polynomial over $\mathbb{Z}$ of degree $d = \phi(k)$ where ϕ is Euler's function. The first few cyclotomic polynomials are shown in Table 1..

Our motivation for considering linear systems over cyclotomic fields arose from problems given to us by Vahid Dabbaghian from computational group theory – from the search for a matrix representation over $\mathbb{C}$ for a finite group.

⋆ This work was supported by NSERC of Canada and the MITACS NCE of Canada
Email addresses: lchenb@cecm.sfu.ca (Liang Chen), mmonagan@cecm.sfu.ca (Michael Monagan).

k	$\Phi_k(z)$	k	$\Phi_k(z)$
3	z^2+z+1	7	$z^6+z^5+z^4+z^3+z^2+z+1$
4	z^2+1	8	z^4+1
5	$z^4+z^3+z^2+z+1$	9	z^6+z^3+1
6	z^2-z+1	10	$z^4-z^3+z^2-z+1$

Table 1. Cyclotomic polynomials of order 3–10

Finding efficient algorithms for solving a linear system $Ax = b$ over $\mathbb{Q}$ is a classical problem in computer algebra. One approach is to solve $Ax = b$ modulo a sequence of primes $p_1, p_2, ...$, and recover the rational solutions in x using Chinese remaindering and rational number reconstruction. For a linear system of dimension n with $A_{i,j}, b_i \in \mathbb{Z}$ where $|A_{i,j}|, |b_i| < 10^c$, that is, the size of the integers in the input are bounded by c digits in length, in general, the size of integers in the solution vector x are n times longer than those in A and b. This means that if we use machine primes, primes of constant bit length, this method will need $O(cn)$ primes in general. If ordinary Gaussian elimination is used to solve the $O(cn)$ linear systems modulo the primes, the complexity of this multiple prime approach is dominated by $cn \cdot n^3$ if ordinary Gaussian elimination is used.

By using linear $p-$adic lifting one can reduce this to $c \cdot n^3$. The $p-$adic approach was first applied to linear systems by Dixon in (4) and Moenck and Carter in (9). The recent paper of Chen and Storjohann (1) describes an implementation of the this approach which reduces the matrix inversion modulo p to floating point matrix multiplications so that level 3 BLAS can be used. We also cite the work of Storjohann (12) which looks at the complexity of solving $Ax = b$ over $\mathbb{Q}$ and contains an extensive bibliography on the problem.

In principle, the same two basic approaches, Chinese remaindering and linear $p-$adic lifting, with rational number reconstruction, can be applied to linear systems over a number field $\mathbb{Q}(\alpha)$. What makes the cyclotomic fields of special interest is the following well known property.

Lemma 1. Let $m(z)$ be a cyclotomic polynomial of degree d and let p be a prime. The probability that $m(z)$ splits into distinct linear factors over $\mathbb{Z}_p$ is asymptotically $1/d$.

Lemma 1 means that there are many primes that split $m(z)$ available. If $\mathbb{Q}(\alpha)$ is an algebraic number field with minimal polynomial $f(z)$ of degree d, then, in general, the probability that $f(z)$ splits into linear factors over $\mathbb{Z}_p$ is $1/d!$ which is too low to try to split $f(z)$. Furthermore, since we can efficiently factor $m(z)$ into linear factors over $\mathbb{Z}_p$ we can solve $Ax = b \bmod p$ at each root of $m(z)$, potentially in parallel, then interpolate the n polynomials in $x_i \in \mathbb{Z}_p[z]$, again, potentially in parallel. The advantage is that we reduce the solving of $Ax = b \bmod m(z)$ to solving d linear systems over $\mathbb{Z}_p$ where, if we choose the prime(s) p appropriately, all the arithmetic in $\mathbb{Z}_p$ that needs to be done can be done directly by the hardware of the computer.

Our paper is organized as follows. In section 2 we first look at the problem of finding a prime p that splits a cyclotomic polynomial into linear factors over $\mathbb{Z}_p$. We then present and analyze the running time of three modular algorithms. The first uses Chinese

remaindering and rational reconstruction. The second uses linear p-adic lifting and rational reconstruction. The third uses Chinese remaindering only to reconstruct the solutions as ratios of determinants over $\mathbb{Z}$.

We have implemented the algorithms in Maple 10. In Section 3 we present timings comparing the algorithms on different problem sets including random inputs and real systems given to us by Vahid Dabbaghian. Unlike the case of solving linear systems over the rationals, where the linear p-adic method is clearly superior, when solving $Ax = b$ over a cyclotomic field, computation of the "error" makes the linear p-adic method more expensive. Furthermore, the size of the integers in the solution vector x may give Chinese remaindering an advantage. Chinese remaindering also has the advantage of being more easily parallelized.

We assume the reader is familiar with rational reconstruction. Rational reconstruction was invented by Paul Wang in (13). A more accessible description of the rational reconstruction problem and the solution using Euclid's algorithm can be found in (2). We use the algorithm of Monagan in (10) because it allows us to control the failure probability.

2. Solving Systems Involving Roots of Unity

We present three modular algorithms for solving a linear system $Ax = b$ modulo $m(z)$ where $m(z)$ is a cyclotomic polynomial of degree d. We restrict to the case A is non-singular. We assume fractions in the input system $Ax = b$ have been cleared and powers of z have been reduced modulo $m(z)$ so that $A_{i,j}, b_i$ are polynomials in $\mathbb{Z}[z]$ of degree less than d.

For the purpose of determining the complexity of our algorithms we use $n = \dim A$, $d = \deg m(z)$, and suppose that largest integer appearing in the input A, b is bounded by 10^c and the largest integer appearing in the numerators and denominators of the rational coefficients in the solution vector x is bounded by 10^e. We also use the following notation and lemma in our analyses.

Let $f(z) = \sum_{i=0}^{l} a_i z^i$ with $a_i \in \mathbb{Z}$. Let $||f||_\infty = \max_i |a_i|$ denote the height of $f(z)$ and let $||f||_1 = \sum_i |a_i|$ denote the one-norm of $f(z)$. For the matrix A and vector b of polynomials in $\mathbb{Z}[z]$ let

$$||A|| = \max_{i,j} ||A_{i,j}||_\infty \quad \text{and} \quad ||b|| = \max_i ||b_i||_\infty.$$

Lemma 2. Let $m(z) = x^d + \sum_{i=0}^{d-1} a_i z^i$ with $a_i \in \mathbb{Z}$. Let $f(z) = \sum_{i=0}^{l} b_i z^i$ with $b_i \in \mathbb{Z}$. Let r be the remainder of f divided m. Then $r \in \mathbb{Z}[x]$ (because m is monic) and $||r||_\infty \leq (1 + ||m||_\infty)^\delta ||f||_\infty$ where $\delta = l - d + 1$.

Proof. The quotient of f divided m has degree $l-d$, hence, there are at most $l-d+1 = \delta$ subtractions in the division algorithm. The first subtraction is $f_1 := f - b_l x^{l-d} m$. We have $||b_l m||_\infty \leq ||f||_\infty ||m||_\infty$, hence,

$$||f_1||_\infty \leq ||f||_\infty + ||m||_\infty ||f||_\infty = (1 + ||m||_\infty)||f||_\infty.$$

For the purpose of bounding $||r||_\infty$ we assume $\deg f_1 = l - 1$. The next subtraction is $f_2 := f_1 - \mathrm{lc}(f_1) x^{l-1-d} m$. Bounding $|\mathrm{lc}(f_1)| \leq ||f_1||_\infty$ we have

$$||f_2||_\infty \leq ||f_1||_\infty + ||f_1||_\infty ||m||_\infty = (1 + ||m||_\infty)^2 ||f||_\infty.$$

Repeating this argument the result is obtained. □

2.1. Splitting $m(z)$ into linear factors.

Let $m(z) = \Phi_k(z)$ be the k'th cyclotomic polynomial of degree $d = \phi(k)$. The following fact (see Huang (8) which mentions this result and other useful results about cyclotomic polynomials and their factorization modulo primes) characterizes precisely which primes split $m(z)$ into distinct linear factors and tells us how to find them. For completeness we give a proof.

Lemma 3. Let p be a prime and let $m(z) = \Phi_k(z)$ be the k^{th} cyclotomic polynomial. If $p \nmid k$ then $m(z)$ has distinct roots in $\mathbb{Z}_p$ if and only if $p \equiv 1 \pmod{k}$.

Proof. Recall that if p is a prime then Fermat's little theorem says $a^p \equiv a \bmod p$ for all integers a, hence, $0, 1, 2, ..., p-1$ are roots of the polynomial $z^p - z$ over $\mathbb{Z}_p$. Since $m(z) | z^k - 1$, to prove the Lemma it suffices to show $z^k - 1 | z^{p-1} - 1$ over $\mathbb{Z}_p$ if and only if $k | p-1$. The easiest way to see this is to verify that if $p - 1 = kq$ then

$$z^{p-1} - 1 = z^{kq} - 1 = (z^k - 1)(z^{k(q-1)} + z^{k(q-2)} + ... + z^k + 1)$$

and if $p - 1 = kq + r$ with remainder $r \neq 0$ then the remainder of $z^{kq+r} - 1$ divided by $z^k - 1$ is $z^r - 1$ which is not zero over $\mathbb{Z}_p$. $\square$

Suppose we have found a prime p satisfying the condition in Lemma 3. Then $m(z)$ has d roots in $\mathbb{Z}_p$. To compute the roots we use use the probabilistic algorithm of Rabin in (11) which is based on the following observation. For a prime $p > 2$, the polynomial $z^p - z = z(z^{(p-1)/2} - 1)(z^{(p-1)/2} + 1)$ has as roots $0, 1, 2, ..., p-1$ in $\mathbb{Z}_p$. Therefore for $\alpha \in \mathbb{Z}$, the polynomial $(z + \alpha)^{(p-1)/2} - 1$ has $(p-1)/2$ of the integers $0, 1, ..., p-1$ as roots in $\mathbb{Z}_p$. Thus for $d > 1$ if one computes

$$g = \gcd((z + \alpha)^{\frac{p-1}{2}} - 1, m(z))$$

one will likely get a non-trivial factor of $m(z)$. By repeating this gcd computation for different α one eventually splits $m(z)$. Then one splits g and m/g recursively.

If α is chosen at random then one can show (see (6) Ch. 8) that

$$\Pr(g \neq 1 \text{ and } g \neq m) > \frac{1}{2} - \frac{1}{2p^2} \geq 4/9.$$

In order to compute g efficiently for large p one needs to use the repeated squaring with remainder algorithm – one computes the remainder $r(z)$ of $(z + \alpha)^{(p-1)/2}$ divided $m(z)$ using repeated squaring modulo $m(z)$ then computes $g = \gcd(r(z) - 1, m(z))$.

An analysis of the time complexity for factorization of univariate polynomials over finite fields can be found in Gerhard and von zur Gathen's book (5). Adapting Theorem 14.1 of their book to the case where $m(z)$ is a product of linear factors, we have the following result.

Theorem 4. The expected number of arithmetic operations in $\mathbb{Z}_p$ that Rabin's algorithm takes to split $m(z)$ into linear factors over $\mathbb{Z}_p$ is $O(\log d(\log p + \log d)M(d))$ where $M(d)$ is the cost of multiplying two polynomials of degree d over $\mathbb{Z}_p$.

Note, the first contribution to the cost, $\log d \log p\, M(d)$, is the repeated squaring cost and the second contribution, $\log^2 d\, M(d)$, is the gcd computation cost. If one uses classical quadratic algorithms for univariate polynomial multiplication and gcd, the expected running time is $O(\log p\, d^2 \log d)$.

Gerhard and von zur Gathen also point out in their text that the basic ideas behind this probabilistic algorithm, in particular, the gcd used to split $m(z)$, were already known to Legendre in 1785!

It is natural to ask whether, for cyclotomic $m(z)$, we can compute the roots faster than Theorem 4. Suppose we can find one root α of $m(z)$. Then since α is a primitive root of unity, $\alpha^k = 1$, and hence the other roots of $m(z)$ are simply the d powers α^i for $0 < i < k$ with $\gcd(i, k) = 1$ which can be computed in $O(k)$ multiplications in $\mathbb{Z}_p$. So the question now is, how fast can be compute one root of $m(z)$?

If we modify Rabin's algorithm to compute only one root of $m(z)$ in the obvious way, namely, split recursively either g or m/g, whichever has smaller degree, then we can compute one root in time $O((\log p + \log d)\, M(d))$. And since the computation of the other roots is trivial, we have the following result which improves Lemma 4 by a factor of $\log d$.

Theorem 5. Let $m(z) = \Phi_k(z)$ be the k^{th} cyclotomic polynomial and $d = \phi(k)$ and p be a prime such that $p \equiv 1 \bmod k$. The d roots of $m(z)$ which are in $\mathbb{Z}_p$ can be computed in $O((\log p + \log d)M(d))$ arithmetic operations in $\mathbb{Z}_p$ (on average).

2.2. Chinese Remaindering

2.2.1. The Algorithm

Algorithm 1 CRT Approach

Input: $A \in \mathbb{Z}^{n\times n}[z]$, $b \in \mathbb{Z}^n[z]$, $m \in \mathbb{Z}[z]$ a cyclotomic polynomial of degree d.
Output: $x \in \mathbb{Q}^n[z]$ which satisfies $A.x \equiv b \pmod m$.

1: Set $x^{(0)} = 0$ and $P = 1$.
2: **for** k = 1, 2, 3, ... **do**
3: Find a new prime p_k s.t. $m(z)$ has d distinct roots $\alpha_{k1}, .., \alpha_{kd}$ in $\mathbb{Z}_{p_k}$ and compute them.
4: Let $A_k = A \bmod p_k$ and $b_k = b \bmod p_k$
5: **for** $i = 1$ to d **do**
6: Evaluate A_k and b_k at $z = \alpha_{ki} \bmod p_k$.
7: Solve $A_k(\alpha_{ki}).x_{ki} \equiv b_k(\alpha_{ki}) \bmod p_k$ for x_{ki}.
8: If $A_k(\alpha_{ki})$ is not invertible modulo p **goto** Step 3.
9: **end for**
10: Interpolate $x_k(z)$ using $(\alpha_{k1}, x_{k1}), .., (\alpha_{kd}, x_{kd})$.
11: Apply Chinese remaindering to recover $x^{(k)}$ from $x^{(k-1)} \bmod P$ and $x_k \bmod p_k$ and set $P = p_k \times P$.
12: **if** $k \in \{1, 2, 4, 8, 16, \ldots\}$ **then**
13: Let x be the output of applying rational reconstruction to the integer coefficients of $x^{(k)} \bmod P$.
14: If rational reconstruction succeeded **and** $m(z)|(A.x - b)$ **then** output x.
15: **end if**
16: **end for**

Algorithm 1 as stated assumes that A is invertible over $\mathbb{Q}$. In order to prove that Algorithm 1 is correct, we need to show that all images of the solutions used in the reconstruction of the solution x over $\mathbb{Q}$ are correct. Consider the 1 by 1 linear system

$$[10z + 15]x = [1]$$

where $m(z) = z^2 + z + 1$. The solution is

$$x = [-2/35z + 1/35].$$

Looking at the solution we see that our algorithm cannot work if it uses primes 5 or 7. It is clear that the matrix $A = [10z + 15]$ is singular mod 5 and Algorithm 1 detects this in step 8. But what about the prime 7? The determinant $D = \det A = 10z + 15$ is not 0 modulo 7 but D^{-1} does not exist mod 7 and hence A is not invertible mod 7. Does Algorithm 1 also eliminate the prime 7? Lemma 6 below proves that it does. First a definition.

Definition 2.1. Let $D = \det(A) \in \mathbb{Z}[z]$. A prime p chosen by Algorithm 1 is said to be *unlucky* if D is invertible modulo $m(z)$ but D is not invertible modulo $m(z)$ modulo p.

Lemma 6. Let p be a prime chosen in Algorithm 1 so that $m(z) = \Pi_{i=1}^{d}(z - \alpha_i)$ for distinct $\alpha_i \in \mathbb{Z}_p$. Then p is unlucky $\Rightarrow$ $A(\alpha_i)$ is not invertible modulo p for some i.

Proof. Let $D = \det A \in \mathbb{Z}[z]$. Then p is unlucky $\Rightarrow$ D is not invertible modulo $(m(z), p)$ $\Rightarrow$ $\deg_z \gcd(D \bmod p, m \bmod p) > 0 \Rightarrow z - \alpha_i | D \bmod p$ for some $i \Rightarrow D(\alpha_i) = 0 \bmod p$ $\Rightarrow$ $A(\alpha_i)$ is not invertible mod p (for some i). $\square$

From the proof we can see also that the unlucky primes are precisely the primes that divide the resultant

$$R = \mathrm{res}_z(D(z), m(z)).$$

It follows that for given inputs A, b and $m(z)$ with A invertible in characteristic 0, there are finitely many unlucky primes, and therefore, if the primes chosen by Algorithm 1 are chosen from a sufficiently large set, Algorithm 1 will rarely encounter an unlucky prime. Lemma 11 in Section 2.4 bounds the size of the integer R and can be used to bound the probability that Algorithm 1 chooses an unlucky prime. It can also be used to modify Algorithm 1 to detect whether A is singular in characteristic 0.

In our analysis of the running time of Algorithm 1 below we have assumed that unlucky primes are rare, and hence, do not affect the running time. Our implementation of Algorithm 1 uses machine primes, 31 bit primes on a 64 bit machine, and 25 bit floating point primes on a 32 bit machine, and consequently unlucky primes are rare in practice.

2.2.2. Analysis

We state the running time of Algorithm 1 in terms of n, d, c which quantify the size of the input and L, the number of primes needed by Algorithm 1 to reconstruct x. Because we use machine primes, primes of constant bit-length that fit into a machine word, L is linear in e, the length of the largest integer appearing in the rational coefficients in x.

In general, the length of the rationals appearing in the output can be slighty more then nd times longer than those in the input (see Lemma 11). But in section 3.1 our linear systems arising in practice illustrate that L can be much much smaller. Thus we state the running times for L and also for $L \in O(cnd)$.

Theorem 7. The running time for Algorithm 1, assuming no unlucky primes are encountered, and not counting the cost of the trial divisions $m(z)|A.x - b$ (in the next section we will show that the trial divisions can be eliminated from Algorithm 1), is

$$O(n^3dL + n^2d^2L + n^2dLc + ndL^2).$$

Moreover, if $L \in O(cnd)$ then the cost is

$$O(n^4d^2c + n^3d^3c^2).$$

The first term n^3dL is the cost of the linear solves modulo p, the second n^2d^2L is for evaluating A at the d roots modulo p, the third n^2dLc is for reducing the input matrix A modulo p, and the fourth ndL^2 is the cost of the Chinese remaindering and the rational reconstruction.

Note, rational reconstruction is not attempted at each step because, unlike Chinese remaindering, it cannot be done efficiently incrementally. Thus our description of the algorithm implies that Algorithm 1 may use up to twice as many primes (so $O(L)$ primes) as are necessary to reconstruct x.

Proof. In step 3 the cost of finding a new machine primes p_k which split the minimal polynomial $m(z)$ into linear factors is negligible. The cost of computing the roots is $O(\log d(\log p + \log d)M(d))$ where $M(d)$ is the cost of multiplication of polynomials of degree d over $\mathbb{Z}_p$. Since our primes are machine primes, of constant bitlength, this reduces to $O(\log^2 dM(d))$. If Karatsuba's multiplication algorithm, or a faster multiplication algorithm, is used, this is in $O(d^2)$. This is smaller than the other terms in the cost.

In step 4 we reduce the integer coefficients in the inputs modulo p before the evaluations in step 6 to reduce the complexity. To reduce the coefficients in A and b takes $O(n^2dLc)$ arithmetic operations in $\mathbb{Z}_p$ since there are n^2 entries in A and n in b to reduce and each entry is a polynomial with at most d non-zero terms. This needs to be done for each of the L primes that we choose. The evaluation step evaluates $A_{i,j}(z)$ and $b_i(z)$ at each root $\alpha_{k1}, \cdots, \alpha_{kd}$ modulo p_k. This costs $O(n^2d^2L)$ because there are n^2+n polynomials to be evaluated, where each requires $O(d)$ arithmetic operations in $\mathbb{Z}_p$ using Horner's rule. This operation needs to be done for all L primes. Solving the system $A_k(\alpha_{ki}).x_{ki} \equiv b_k(\alpha_{ki}) \bmod p_k$ for x_{ki} takes $O(n^3)$ operations using Gaussian Elimination and this is done over all the roots modulo all primes for a total of $O(n^3dL)$ arithmetic operations in $\mathbb{Z}_p$. Interpolation takes $O(nd^2L)$ arithmetic operations since we only need to interpolate the solution vector which has n elements over the d roots. Interpolation is dominated by the evaluations.

Chinese remaindering is applied to integer coefficients of $x^{(k-1)} \bmod P$ and $x_k \bmod p_k$. There are at most nd integers to reconstruct. The *incremental* cost at step k is O(k) per coefficient since P is a product of $k-1$ primes. Summing $O(knd)$ for $k = 1..O(L)$, the total cost is $O(ndL^2)$.

If classical Euclid's algorithm is used for rational reconstruction, rational reconstruction from an integer modulo P, a product of k machine primes, primes of size $O(1)$, costs $O(k^2)$. Since we attempt rational reconstruction after $k = 1, 2, 4, 6, 16, ...$ primes, the final successful rational reconstruction will dominate total the cost of rational reconstruction. Since the solution vector x has at most nd rational coefficients, the total cost of the final successful reconstruction is $O(ndL^2)$. Adding the above contributions gives the running time as stated. □

2.2.3. Rational Reconstruction

In our implementation of Algorithm 1, we found, consistent with (1), that for dense inputs with integer coefficients in A and b chosen uniformly at random, the rationals in the solution vector x are much longer than the integers in A and b. For such inputs rational reconstruction and Chinese remaindering can dominate the cost.

Obviously, one may employ asymptotically fast algorithms for Chinese remaindering and rational reconstruction to reduce the theoretical complexity of Algorithm 1. From a practical viewpoint, however, all one needs is fast integer multiplication and division. This is relevant because Maple is using the GMP integer arithmetic package which has fast integer multiplication and long division but no fast Euclidean algorithm, hence, no fast Chinese remaindering and rational reconstruction are available.

One can effectively reduce the cost of Chinese remaindering to that of integer multiplication without using asymptotically fast Chinese remaindering as follows. At step $k = 2^{j+1}$ suppose we have obtained u satisfying $Au \equiv b \bmod m(z) \mod P$ where $P = p_1 \times p_2 \times ... \times p_{2^j}$ from step $k = 2^j$. Suppose we next compute v satisfying $Av \equiv b \mod m(z) \bmod Q$ where $Q = p_{2^j+1} \times p_{2^j+2} \times ... \times p_{2^{j+1}}$. We then need to solve for x_k satisfying $Ax_k \equiv b \mod m(z) \mod PQ$. If we write $x_k = u + wP$ we have $w = (v-u)P^{-1} \mod Q$. This requires inverting P modulo Q which costs $O((2^j)^2)$ using the classical Euclidean algorithm. But this is done once and then the scalar multiplication of the vector of $(v-u)$ by $P^{-1} \mod Q$ and the vector w by P costs $O(ndM(2^j))$ where $M(k)$ is the cost of multiplying and dividing integers of length k. If a fast algorithm is used here the total cost of Chinese remaindering can be reduced from $O(ndL^2)$ to $O(L^2 + nd \log LM(L))$.

The cost of the successful rational reconstruction of the $\leq nd$ rational coefficients in x can similarly be reduced to roughly one rational reconstruction and $O(nd)$ long multiplications and divisions using a clever trick. Suppose we are reconstructing a rational from an image $u \bmod P$ and b is the LCM of the denominators of all rationals reconstructed so far. The idea is to apply rational reconstruction to $b \times u \bmod P$ instead. We refer the reader to (1) for details. Assuming fast integer multiplication and division are available, these improvements effectively reduce the cost of Chinese remaindering and rational reconstruction to that of fast multiplication, that is, from $O(ndL^2)$ to $O(L^2 + ndM(L))$ where $M(L)$ is the cost of multiplication of integers of length L digits and the L^2 term is the cost of the classical Euclidean algorithm which we use for computing inverses and rational reconstruction.

For rational reconstruction we use the Maximal Quotient Rational Reconstruction algorithm of Monagan (10) because it will fail with high probability when the modulus P is not large enough yet to reconstruct the rational number. One applies rational reconstruction to the coefficients in x sequentially until it fails. When the next rational reconstruction attempt is made, one should start in x where the last failure occurred.

2.3. Linear p-adic Lifting

2.3.1. The Algorithm

2.3.2. Analysis

We state the running time of Algorithm 2 in terms of n, d, c and L, the number of lifting steps that Algorithm 2 takes. For p^L to be large enough to reconstruct rationals of length e in x, $L \in O(e)$.

Algorithm 2 Linear p-adic Lifting Approach

Input: $A \in \mathbb{Z}^{n\times n}[z]$, $b \in \mathbb{Z}^n[z]$, $m \in \mathbb{Z}[z]$ a cyclotomic polynomial of degree d.
Output: $x \in \mathbb{Q}^n[z]$ which satisfies $A.x = b \pmod m$

1: Find a machine prime p s.t. m splits linearly over $\mathbb{Z}_p$, and compute the roots $\alpha_1, .., \alpha_d$ of $m(z) \bmod p$
2: Let $e_0 = b$, $x^{(0)} = 0$
3: Invert $A(\alpha_i) \bmod p$ for all roots.
If any $A(\alpha_i)$ is not invertible mod p then **Goto** Step 1.
4: **for** $k = 0, 1, 2, \ldots$ **do**
5: Reduce $e_k \bmod p$
6: **for** $i = 1$ to d **do**
7: Evaluate the error e_k at $z = \alpha_i \bmod p$.
8: Compute $x_{ki} \equiv A(\alpha_i)^{-1}.e_{ki} \bmod p$
9: **end for**
10: Interpolate $x_k(z)$ from (α_1, x_{k1}), ...,(α_d, x_{kd}).
11: Set $e_{k+1} = (e_k - A.x_k \bmod m(z)) \ / \ p$
12: $x^{(k+1)} = x^{(k)} + x_k \times p^k$
13: **if** $k + 1 \in \{1, 2, 4, 8, 16, ..., \}$ **then**
14: Let x be the output of applying rational reconstruction to $x^{(k+1)} \bmod p^{k+1}$.
15: If rational reconstruction succeeds and $m(z)|(A.x - b)$ **then** output x.
16: **end if**
17: **end for**

Theorem 8. The running time for Algorithm 2, not counting the cost of the trial divisions in $m(z)|A.x - b$, is

$$O(n^3d + n^2d^2Lc + ndL^2).$$

Moreover, if $L \in O(cnd)$ then the cost is

$$O(n^3d + n^3d^3c^2 + n^3d^3c^2) = O(n^3d^3c^2).$$

The first contribution, n^3d, is the cost of the d matrix inverses. The second, n^2d^2Lc, is the total cost of computing the error e_k. The third, ndL^2, is the cost of step 12 which is a conversion from the p-adic representation of the solution to an integer representation. The last, ndL^2, is the cost of rational reconstruction.

Proof. In this algorithm, we only need one prime p such that the minimal polynomial splits linearly over $\mathbb{Z}_p$. The time for computing this can be ignored. In Step 3 we precompute the inverse of the input matrix A at each root d modulo p using Gaussian Elimination. This costs $O(n^3d)$ arithmetic operations in $\mathbb{Z}_p$ in total. To reduce the error e_k modulo p in Step 5 costs $O(ndcL)$ operations since e_k is a vector of n polynomials of degree $< d$ with coefficients of length c digits and this is done $O(L)$ times. The substitution of all d roots into e_k costs $O(nd^2L)$ arithmetic operations. Computing the solution vector x_{ki} is just a matrix vector multiplication modulo p which costs $O(n^2dL)$ in total. Interpolation costs $O(nd^2L)$ which is the same as in Algorithm 1. To compute the error e_k in Step 11 we need to do a matrix vector multiplication of polynomials over $\mathbb{Z}$ then divide by $m(z)$. This is dominated by the matrix vector multiplication which requires n^2 multiplications of polynomials of degree less than d. Now the integer

coefficients of the polynomials in A are of size $O(c)$ but the integers in x_k are modulo p, that is, of size $O(1)$. Consequently fast integer multiplication is not applicable here. This costs $O(n^2d^2Lc)$ in total using classical polynomial multiplication. In Step 12, adding x_kp^k to $x^{(k)}$ costs $O(ndk)$ operations for each lifting step. In total this is $O(ndL^2)$ which note is the same as the cost of the incremental Chinese remaindering in Algorithm 1. The rational reconstruction cost is the same as for Algorithm 1, namely $O(ndL^2)$. Therefore, the total running time for this algorithm is $O(n^3d + n^2d^2Lc + ndL^2)$. $\square$

2.3.3. Computing the error.

In our implementation of Algorithm 2, one of the most expensive components are the computation of the error in step 11, in particular, the matrix vector multiplication in $A.x_k$ which needs to be computed over $\mathbb{Z}$. This requires n^2 polynomial multiplications. It has complexity $O(n^2d^2c)$ assuming classical multiplication. Since the length of the vector $A.x_k$ is more than $O(n^2dc)$ in general, we cannot reduce the complexity of computing the error by more than a factor of d. We have attempted to speed this up, by a factor of d, by choosing primes $p_1, p_2, ...$ such that $m(z)$ has d roots α_{ij} in $\mathbb{Z}_{p_i}$, evaluating $A(\alpha_{ij}) \bmod p_i$, caching these primes, α_{ij} and matrices for the next lifting step, multiplying $A(\alpha_{ij}).x_k(\alpha_{ij}) \bmod p_i$, interpolating, and then Chinese remaindering. But, the result was somewhat disappointing; it was sometimes slower than computing the $A.x_k \bmod m(z)$ over the integers.

Because the computation of the error is expensive, Algorithm 1 is often better despite the n^3dL term in Algorithm 1's complexity. The error computation involves polynomial arithmetic but the linear solves in Algorithm 1 are done using machine arithmetic.

Reconstruction cost.

The other most expensive component of Algorithm 2 is the reconstruction cost $O(ndL^2)$ when L is large. If a fast multiplication is available, we have already mentioned how the rational reconstruction cost can be reduced to $O(nd \log LM(L) + L^2)$ where $M(L)$ is the cost of multiplying integers of size L. The cost of Step 11 can also be reduced similarly from $O(ndL^2)$ to $O(L^2 + nd \log LM(L))$, as follows. Noting that the algorithm only attempts rational reconstruction for
$k \in \{1, 2, 4, 8, ..., 2^j, ...\}$, that is, we only need to compute $x^{(k)}$ for these values of k. Now

$$x^{(k)} = x_0 + x_1p + ... + x_{k-1}p^{k-1}$$

thus

$$x^{(2k)} = x_0 + x_1p + ... + x_{2k-1}p^{2k-1} = x^{(k)} + p^k\Delta_k$$

where $\Delta_k = x_k + x_{k+1}p + ... + x_{2k-1}p^{k-1}$. To compute $x^{(2k)}$, if we first compute $x^{(k)}$ then Δ_k then the scalar multiplication of Δ_k by p^k costs $O(ndM(k))$ where $M(k)$ is the integer multiplication cost. Now if one computes both $x^{(k)}$ using the same method recursively, and also Δ_k using the same method recursively, the cost of computing $x^{(2k)}$ is the cost of computing $x_0, x_1, ..., x_{2k-1}$ plus

$$T(2k) \leq 2T(k) + ndO(M(k)).$$

Solving for $T(L)$ we obtain a total cost of $O(nd \log LM(L))$.

2.4. Trial Division

Algorithms 1 and 2 both terminate when rational reconstruction of x succeeds and $m(z)|b - Ax$ over $\mathbb{Q}$. This trial division can be sped up if one avoids arithmetic with the fractions that appear in x. One computes D the least common multiple of the denominators of all fractions appearing in the coefficients of the polynomials in x, then computes Dx to clear fractions in x, and tests if $m(z)|Db - A(Dx)$. Here all arithmetic is over $\mathbb{Z}$ since $m(z)$ is monic over $\mathbb{Z}$. In our experiments, the time spent doing trial divisions this way is a few percent or less of the total time.

However, we show that the trial division can be omitted entirely if the modulus $M = p_1 \times p_2 \times ... \times p_k$ in Algorithm 1 ($M = p^k$ in Algorithm 2) is sufficiently large. That is, by using additional primes (if necessary) in Algorithm 1, or doing additional lifting steps (if necessary) in Algorithm 2, we can omit the test. The idea is to bound the size of the integer coefficients in the remainder of $Db - A(Dx)$ divided $m(z)$ and require that the modulus M be greater than twice (allowing for positive and negative integers) the bound.

Let $N = ||Dx||$. First $||A(Dx)|| \leq ndN||A||$ since each entry in the vector $A(Dx)$ is obtained by adding n products of polynomials of degree at most $d-1$. Hence

$$||Db - A(Dx)|| \leq D||b|| + ndN||A|| = B.$$

Now we compute the remainder r_i of the i'th entry of the vector $Db - A(Dx)$ divided by $m(z)$. Applying Lemma 2 with $\delta = (2d-2) - d + 1 = d - 1$ we have, for all i,

$$||r_i||_\infty \leq (1 + ||m(z)||_\infty)^{d-1} B.$$

Hence we can state the following result.

Theorem 9. If rational reconstruction succeeds in Algorithms 1 and 2 and the modulus M satisfies

$$M > 2(1 + ||m(z)||_\infty)^{d-1}(D||b|| + ndN||A||)$$

then $m(z)|b - Ax$ over $\mathbb{Q}$.

Now we observe that if $m(z)$ is a cyclotomic polynomial, $||m||_\infty$ is small. There are five cyclotomic polynomials of order less than 1000 with height 3, 53 with height 2, and the rest have height 1 and the first cyclotomic polynomial with $||m|| > 1$ is $\Phi_{105}(x)$. In particular, for $||m(z)||_\infty = 1$ then Theorem 9 requires $M > 2^d(D||b|| + ndN||A||)$. So for cyclotomic $m(z)$, the length of M needed to satisfy Theorem 9 is not much longer than $D||b|| + N||A||$. We now argue that if rational reconstruction succeeds and x is correct then the value of M in Algorithms 1 and 2 likely satisfies Theorem 9 with no additional primes (lifting steps) needed.

When the rational number reconstruction of Monagan in (10) succeeds in Algorithms 1 and 2, with high probability, we have the correct solution vector x and thus $M > 2B \times 2^\Delta$ where $B \geq |ab|$ for all rational numbers a/b appearing in the coefficients of the polynomials in the solution vector x and Δ is the number of extra bits to make the probability of correctness high.

Normally, B is about the same size as ND but it can be smaller. The reader may now see that if $M > 2B2^\Delta$ and $||m|| = 1$ then indeed M will likely be greater than

$2^d(D||b|| + ndN||A||)$ when B, the size of the rationals in x, is larger than the $||b||$ and $||A||$, the size of the integers in the input, which is normally the case. This is confirmed by our experiments. For all linear systems that we tried, after rational reconstruction succeeded for the first time, Theorem 9 was satisfied immediately, that is, no additional primes (lifting steps) were needed.

2.5. A Bound for D

Recall that D is the LCM of the denominators of the fractions appearing in the solution vector x where $x = A^{-1}b \bmod m(z)$. Thus D divides the LCM of the denominators of the fractions appearing in the inverse of the polynomial $\det A$ modulo $m(z)$, that is, $D|\mathrm{res}_z(\det A, m(z)) \in \mathbb{Z}$. We have $\deg_z \det A \le n(d-1)$ since $\deg_z A_{i,j} < d$. We use the following result (see (7)) to bound $||\det(A)||_\infty$.

Lemma 10 (Goldstein and Graham, 1974). Let A be an n by n matrix of polynomials in $\mathbb{Z}[z]$. Let A' be the matrix of integers with $A'_{i,j} = ||A_{i,j}||_1$ that is, $A'_{i,j}$ is the one norm of $A_{i,j}$. Let H be Hadamard's bound for $\det A'$. Then $||\det A||_\infty \le H$.

Since $\deg_z A_{i,j} \le d-1$ we have $A'_{i,j} \le dC$. Applying Hadamard's bound to bound $|\det A'|$ we obtain

$$||\det A||_\infty \le \Pi_{i=1}^n \sqrt{\Sigma_{j=1}^n A'^2_{i,j}} = d^n n^{n/2} ||A||^n.$$

To calculate $\mathrm{res}_z(\det A, m(z))$, because $m(z)$ is monic

$$\mathrm{res}_z(\det A, m(z)) = \pm\, \mathrm{res}(r(z), m(z))$$

where $r(z)$ is the remainder of $\det A$ divided $m(z)$. Applying Lemma 2 to determine $||r||_\infty$ we have $\deg_z \det A \le n(d-1)$ thus $\delta \le n(d-1) - d + 1 = (n-1)(d-1)$ and

$$||r||_\infty \le (1 + ||m||_\infty)^{(n-1)(d-1)} d^n n^{n/2} ||A||^n.$$

Let $R = \mathrm{res}_z(r(z), m(z))$. Note that R is an integer. To bound $|R|$ recall that $R = \det S$ where S is Sylvester's matrix for the polynomials $r(z)$ and $m(z)$. Now $\deg_z r < d$ but for the purpose of bounding $|R|$ we assume $\deg_z r = d-1$. Then S is a $2d-1$ by $2d-1$ matrix of integers where the d coefficients of $r(z)$ are repeated in the first d rows of S and the $d+1$ coefficients of $m(z)$ are repeated in the last $d-1$ rows. Applying Hadamard's bound to the rows of S we obtain

$$|\det S| \le \sqrt{d||r||_\infty^2}^d \times \sqrt{(d+1)||m||_\infty^2}^{d-1}$$

from which we obtain the following result where we used $\sqrt{d+1}^{d-1} < \sqrt{d}^d$ for $d > 1$ to simplify the result.

Lemma 11. Let $R = \mathrm{res}_z(\det A, m(z))$. Then

$$|R| < d^{nd+d} ||m||_\infty^{d-1} (1 + ||m||_\infty)^{(n-1)(d-1)d} n^{dn/2} ||A||^{nd}.$$

The bound says the size of the denominators in $x = A^{-1}b$ can could more than nd times longer than $||A||$. Indeed if one constructs inputs A and b with polynomials of degree $d-1$ with coefficients chosen randomly from $[0, 10^c)$, so that $||A|| < 10^c$, $||b|| < 10^c$ and the bit-length of the input is $O(n^2 d \log 10^c) = O(cn^2 d)$, then one readily finds examples with $D > 10^{cnd}$.

The bound can also be used to bound the probability that our choice of primes in Algorithms 1 and 2 might result in a non-singular system.

2.6. Determinant Ratios

From Cramers rule, the solution vector x of the linear system $Ax = b \bmod m(z)$ may be expressed as

$$x_i = \frac{\det(A^{(j)})}{\det(A)} \bmod m(z)$$

where $A^{(j)}$ is the matrix A with the $j'th$ column replaced by b. The analysis in the previous section showed that the rationals in the solution vector x may be up to nd times longer than the integers in the input A, b, which means that x may be d times longer than the input. The factor of d comes from inverting $\det(A)$ modulo $m(z)$. If we choose instead to write the solutions in the form

$$x_j = \frac{\det(A^{(j)}) \bmod m(z)}{\det(A) \bmod m(z)},$$

in general, the integers in the determinants will be a factor of d times smaller. Moreover we can easily compute images of $\det(A^{(j)})$ and $\det A$ by modifying the solving of $A(\alpha_i)x = b(\alpha_i) \bmod p_k$ in Algorithm 1. One also computes the determinant $d = \det(A(\alpha_i)) \bmod p_k$ (at negligible additional cost) to obtain images of $\det A$ and then multiplies the scalars $x_j(\alpha_i) \bmod p_k$ by d (at negligible additional cost) to obtain images of $\det A^{(j)}$. In order to reconstruct $\det A \mod m(z)$ and the $\det(A^{(j)}) \bmod m(z)$, we will need bounds on their heights. We state these in the following lemma.

Lemma 12.

$$||\det A \bmod m(z)||_\infty \leq d^n ||A||^n (1 + ||m||_\infty)^{(n-1)(d-1)}$$

and

$$||\det A^{(j)} \bmod m(z)||_\infty \leq d^n ||A||^{n-1} ||b||_\infty (1 + ||m||_\infty)^{(n-1)(d-1)}.$$

Proof. In the previous section we have determined that $||\det A||_\infty \leq d^n ||A||^n$. Now $\det A \in \mathbb{Z}[z]$ has degree at most $n(d-1)$ in z. Applying Lemma 2 to bound bound $||\det A \bmod m(z)||_\infty$ yields the first result. The second result follows by noting that Lemma 10, which is stated in terms of the rows of A, also applies to the columns of A. □

We now give the algorithm. Since we use these bounds to determine the number of primes needed in advance, we recursively solve $Ax = b \bmod m(z)$ modulo half the primes, then modulo the other half, and Chinese remainder the two results. This reduces the integer Chinese remaindering cost (which can be the largest cost when the solutions have large integers) to the cost of integer multiplication (there is one inverse computed modulo the product of half the primes but $O(nd)$ multiplications of large integers). That is, asymptotically fast Chinese remaindering is not needed unless $c >> nd$ to get asymptotically fast reconstruction in practice.

In comparing Algorithm 1 and Algorithm 3, since Algorithm 3 needs to reconstruct integers (not rationals) of length d times smaller than Algorithm 1 it needs a factor of $2d$ fewer primes and hence will be $2d$ times faster than Algorithm 1 in general. However, the size of the rationals in the solution vectors of real applications may be much smaller than the bound in Lemma 11. Indeed, all the linear systems from Dhabbigian (3) have small solutions, some very small. Thus in our experiments, Algorithms 1 and 2 were much faster than Algorithm 3 in our real application.

Algorithm 3 Determinant Ratio.

Input: $A \in \mathbb{Z}[z]^{n\times n}$, $b \in \mathbb{Z}[z]^n$, $m \in \mathbb{Z}[z]$ a cyclotomic polynomial of degree d.
Output: $D \in \mathbb{Z}[z]$ and $x \in \mathbb{Z}[z]^n$ which satisfy
$D = \det A \bmod m(z)$ and $A.x \equiv Db \bmod m(z)$.

1: Let $B = d^n||A||^{n-1}\max(||A||, ||b||)(1 + ||m||)^{(n-1)(d-1)}$.
2: Let $P = \{p_1, ..., p_k\}$ be a set of primes such that $\Pi p_i > 2B$ and $m(z)$ splits into distinct linear factors modulo p_i.
3: Call Subroutine M with inputs A, b, m and P.

Algorithm 4 Subroutine M

Input: A, b, m in $\mathbb{Z}[z]$ and $P = \{p_1, p_2, ..., p_k\}$ a set of primes.
Output: (D, X, M) satisfying $D = \det A \bmod m(z) \bmod M$ and $m(z)|AX - Db \bmod M$.

1: **if** $k > 1$ **then**
2: Set $l = \lfloor k/2 \rfloor$.
3: Let D_1, X_1 be the output from Subroutine M applied to inputs A, b, M and $P = \{p_1, p_2, ..., p_l\}$.
4: Let D_2, X_2 be the output from Subroutine M applied to inputs A, b, M and $P = \{p_{l+1}, ..., p_k\}$
5: Let $M_1 = \Pi_{i=1}^{l} p_i$, $M_2 = \Pi_{i=l+1}^{k}$ and let $M = M_1 M_2$.
6: Compute $I = M_1^{-1} \bmod M_2$.
7: Set $\Delta_D = I(D_1 - D_2) \bmod M_2$ and $\Delta_X = I(X_1 - X_2) \bmod M_2$ using the symmetric range for the integers modulo M_2.
8: Set $D = D_1 + \Delta_D M_1 \in \mathbb{Z}_M[z]$.
Set $X = X_1 + \Delta_X M_1 \in \mathbb{Z}_M[z]^n$.
9: Ouput (D, X, M)
10: **else**
11: Compute the roots $\alpha_1, ..., \alpha_d$ of $m(z) \bmod p_1$.
12: Set $A = A \bmod p_1$ and $b = b \bmod p_1$
13: **for** $i = 1$ to d **do**
14: Evaluate A and b at $z = \alpha_i \bmod p_1$.
15: Solve $A(\alpha_i).x_i \equiv b(\alpha_i) \bmod p_1$ for $x_i \in \mathbb{Z}_{p_1}^n$ and compute also $D_i = \det A(\alpha_i) \bmod p_1$.
16: **if** $D_i \neq 0$ **then** set $x_i = D_i \times x_i \bmod p_1$
else pick a new prime $p > p_1$ s.t. $m(z)$ splits into linear factors and restart Algorithm M using $p_1 = p$.
17: **end for**
18: Interpolate $D \in \mathbb{Z}_p[z]$ from $(\alpha_1, D_1), ..., (\alpha_d, D_d)$.
19: Interpolate $x \in \mathbb{Z}_p[z]^n$ from $(\alpha_1, x_1), ..., (\alpha_d, x_d)$.
20: Output D, x, p_1.
21: **end if**

3. Implementation and Timings

We have implemented all three algorithms in Maple 10. In our programs, we used the Maple library routines `iratrecon` for rational number reconstruction, and our own routine `iscyclotomic` to find the order k of the given cyclotomic polynomial. We used the library routine `Roots(m) mod p` to find the roots of $m(z)$ in $\mathbb{Z}_p$. We use 25 bit

floating point primes on 32 bit machines, and 31 bit integer primes on 64 bit machines. If we choose the primes as stated, we can take advantage of the fast C code in the `LinearAlgebra:-Modular` package which provides fast polynomial evaluation, linear solving and matrix inversion over $\mathbb{Z}_p$.

3.1. Timings

We compare our algorithms on three data sets. In the first data set the integer coefficients in A and b are generated at random and hence the size of the integers in the solution vector x are large. The second data set is a set of real problems which have small rationals in the solution vectors x and some of the systems are sparse. For the second data set, we also timed Maple's linear solver, the `LinearSolve` command in the LinearAlgebra package. Maple uses a sparse fraction-free Gaussian elimination algorithm to solve a linear system over an algebraic number field. Maple represents linear equations as a polynomials in $x_1, x_2, ..., x_n$, which is a sparse representation.

All timings we give in the following were obtained using Maple 10 on an AMD® Opteron 150 processor @ 2.4 GHz with 2GB of RAM. Our programs are designed for random dense inputs. They do not take advantage of any structure if the input systems are sparse.

Data Set 1:
For the first data set we use the $7th$ cyclotomic polynomial $m(z) = 1 + z + z^2 + z^3 + z^4 + z^5 + z^6$. The first data set consists of systems of dimension $5, 10, 20, 40, 80, 160$ where the entries of A and b were generated using the Maple command

```
> f := randpoly(z,dense,degree=5,coeffs=rand(2^c)):
```

for different values of c which specifies the lengths of the integer coefficients in binary digits. This Maple command will give us a dense polynomial in z with degree 5 and coefficients uniformly chosen at random from $[0, 2^c)$.

Table 1 shows the running time of dense random polynomial inputs for both algorithms. We observe that the linear p-adic lifting approach is faster than the Chinese remaindering approach when the dimension n of the input matrix and input vector gets larger as expected.

The first entry in each cell is the runtime (in CPU seconds) using linear p-adic lifting; the second one is for using Chinese remaindering, and "$-$" means the runtime is > 10000 seconds. We also put the number of primes that were used to recover the coefficients in the solution vector. It corresponds to the number of lifting steps in the linear p-adic lifting method.

Data Set 2:
The problems in this data set were given to us by Vahid Dabbaghian. They include systems with various dimensions, coefficient lengths, and minimal polynomials. The systems are available at

`http://www.cecm.sfu.ca/CAG/code/VahidsSystems.zip`

Table 2 shows the running times for the systems given to us by Vahid Dabbaghian. The line Linsolve in the table is the time for Maple's `LinearSolve` command from the LinearAlgebra package. One can see that the modular algorithms are typically 1000 times faster on these systems.

Most of the input systems in this problem set are sparse, and the solutions are small. That is, in most cases one prime number is sufficient to successfully reconstruct the

$n \setminus c$	4	16	64	256	1024	Remark
	.096	.446	3.071	38.77	707.1	Lin
5	.069	.447	2.766	28.782	565.7	CRT
	26	122	401	1601	6401	# primes
	.301	1.423	11.80	161.3	3084	Lin
10	.262	1.317	10.367	140.0	3109	CRT
	65	226	842	3250	12770	# primes
	1.067	5.308	49.87	725.7	–	Lin
20	.905	5.654	52.01	787.2	–	CRT
	122	442	1682	6401	n/a	# primes
	4.035	21.68	236.4	4120	–	Lin
40	4.699	31.99	327.6	5508	–	CRT
	226	842	3250	12997	n/a	# primes
	20.90	111.4	1505	–	–	Lin
80	33.64	235.0	2802	–	–	CRT
	485	1682	6562	n/a	n/a	# primes
	126.9	665.6	–	–	–	Lin
160	269.7	2061	–	–	–	CRT
	962	3365	n/a	n/a	n/a	# primes

Table 2. Runtime (in CPU seconds) of Random dense input with various dimensions and coefficients.

polynomial coefficients in the solution vector. In this problem set, both algorithms have similar performance because of the simplicity of the solutions. We state the maximum length of the coefficients in the inputs A and b and the output x in binary digits. The bottom row shows the number of primes used to successfully reconstruct the coefficients in the solutions. It also corresponds to the number of lifting steps.

References

[1] Zhuliang Chen and Arne Storjohann. A BLAS based C library for exact linear algebra on integer matrices. *Proceedings of ISSAC '05*, ACM Press, pp. 92–99, 2005.

[2] G. E. Collins and M. J. Encarnacion (1995). Efficient Rational Number Reconstruction. *J. Symbolic Computation* **20**, pp. 287–297.

[3] Polynomials systems from Vahid Dabbaghian.
See `http://www.cecm.sfu.ca/CAG/code/VahidsSystems.zip`

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

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

file	sys49	sys100	sys100b	sys144	sys196
$\deg_z(m)$	4	8	4	2	2
k	5	24	8	4	3
$\log_2 1 + \|\|A\|\|$	10	5	2	4	11
$\log_2 1 + \|\|x\|\|$	45	14	1	1	229
# primes	4	1	1	1	9
Lin	.109	.443	.030	.029	1.183
CRT	.144	.788	.029	.036	3.344
Linsolve	109.2	3080	30.15	10.49	4419
file	sys225	sys256	sys576	sys900	sys900b
$\deg_z(m)$	4	4	6	8	2
k	5	12	7	24	4
$\log_2 1 + \|\|A\|\|$	2	3	3	2	5
$\log_2 1 + \|\|x\|\|$	875	2	1	2	1
# primes	36	1	1	1	1
Lin	2.374	.174	.612	2.761	.462
CRT	3.056	.155	.842	2.358	1.458
Linsolve	769.1	848.5	2055	2265	1195

Table 3. Runtime (in CPU seconds) on some of the systems given by Vahid Dabbaghian.

[6] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*, Kluwer Academic, 1992.

[7] A. Goldstein and G. Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review* **1** 394–395, 1974.

[8] Ming-Deh A. Huang. Factorization of polynomials over finite fields and factorization of primes in algebraic number fields. *Proceedings of STOC '84*, ACM Press, pp. 175–182, 1984.

[9] R. Moenck and J. Carter. Approximate algorithms to doerive exact solutions to systems of linear equations. *Proceedings of EUROSAM '79*, Springer Verlag LNCS **72**, pp. 65–72, 1979.

[10] Michael Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. *Proceedings of ISSAC '04*, ACM Press, pp. 243–249, 2004.

[11] Michael Rabin. Probabilistic Algorithms in Finite Fields. *SIAM J. Computing* **9**(2) pp. 273–280, 1980.

[12] Arne Storjohann. The shifted number system for fast linear algebra on integer matrices. *J. Complexity*, Elsevier, **21** 605–650, 2005.

[13] P. Wang (1981). A p-adic Algorithm for Univariate Partial Fractions. *Proceedings of SYMSAC '81*, ACM Press, pp 212-217.