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.

Submitted January 2008 to the Journal of Symbolic Comptuation

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 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 use rational reconstruction to recover the rational coefficients in the solution vector. We also give a third algorithm which computes the solution x as a ratio of two determinants modulo $m(z)$ using Chinese remaindering only. In general, because this representation for x is a factor of $d = \deg m$ more compact, we can compute it the fastest in general.

We have implemented the algorithms in Maple with key parts of the implementation implemented in C for efficiency. A complexity analysis and experimental timings show that on inputs with random matrix entries, the third method is fastest. However, on real inputs arising from problems in computational group theory, the first two methods are must faster than the third method because the solution vectors have very small rational coefficients.

We also give a fast probabilistic algorithm for computing the roots of a cyclotomic polynomial modulo a prime p and some new data on the height of the cyclotomic polynomials of order up to 100 million.

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 primitive k'th root of unity. These number fields, which include the complex rationals, are called the cyclotomic fields. The

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

minimal polynomial $m(z)$ for ζ is $\Phi_k(z)$, the k'th cyclotomic polynomial. It is a monic irreducible polynomial in $\mathbb{Z}[z]$ of degree $d = \phi(k)$ (where ϕ is Euler's function) whose roots are the primitive k'th complex roots of unity. The first few cyclotomic polynomials are shown in Table 1.

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

Table 1. Cyclotomic polynomials of order 1–10

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. What is special about these linear systems is that the size of the rationals in their solution vectors are much smaller than they would be the input coefficients were randomly generated. This means that our modular algorithms need to be output sensitive, sensitive to the size of the rational numbers in x.

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 a term or order n^4.

By using linear $p-$adic lifting one can reduce this to n^3. The $p-$adic approach was first applied to linear systems by Dixon in (5) and Moenck and Carter in (14). The recent paper of Chen and Storjohann (2) 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 (18) which looks at the complexity of solving $Ax = b$ over $\mathbb{Q}$ and contains an extensive bibliography on the problem. Progress has also been made on the complexity of sparse linear systems. In (6), Eberly et. al. show that if one can multiply a matrix A of dimension n by a vector in $O(n)$ field operations, then one can compute the dense solution of $Ax = b$ in $O(n^{2.27})$ field operations.

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 fact (which follows from Lemma 4).

Lemma 1. *Let $m(z) = \Phi_k(z)$ 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, 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. And if we choose the prime(s) p appropriately, so that arithmetic in $\mathbb{Z}_p$ can be done directly by the hardware of the computer, then the overall algorithm will be efficient in practice.

1.1. Organization of paper.

Our paper is organized as follows. In section 2 we look at the problem of finding a prime p that splits a cyclotomic polynomial $\Phi_k(z)$ into linear factors over $\mathbb{Z}_p$ and how to compute the roots of $\Phi_k(z)$ in $\mathbb{Z}_p$ efficiently. We then present and analyze the running time of three modular algorithms for solving $Ax = b$ over a cyclotomic field. The first uses Chinese remaindering and rational reconstruction. The second uses linear p-adic lifting and rational reconstruction. The third uses Chinese remaindering only.

We assume the reader is familiar with rational reconstruction. Rational reconstruction was invented by Paul Wang in (19). A more accessible description of the rational reconstruction problem and the solution using Euclid's algorithm can be found in (3). We use the algorithm of Monagan in (15) because it allows us to control the failure probability.

We have implemented the algorithms in Maple. 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. We find that the Chinese remaindering approach is competitive and our third algorithm is the fastest in general.

1.2. Dividing by $m(z)$.

We use the following notation and lemma in our analysis. 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

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

Thus $\| [A|b] \|$ is the magnitude of the largest integer appearing in the coefficients of the polynomials in A and b.

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*

$$||r||_\infty \leq (1 + ||m||_\infty)^\delta ||f||_\infty \;\; \text{where} \;\; \delta = l - d + 1.$$

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 $||b_l m||_\infty \le ||f||_\infty ||m||_\infty$, hence,

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

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 $|\mathrm{lc}(f_1)| \le ||f_1||_\infty$ we have

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

Repeating this argument the result is obtained. □

1.3. The height of the cyclotomic polynomials.

Let $m(z) = \Phi_k(z)$ and let $H_k = \| \Phi_k(z) \|_\infty$. Thus H_k is height of the cyclotomic polynomial of order k. In section 2, because powers of $(1+ \| m(z) \|_\infty)$ appear in bounds that affect the complexity of our algorithms, we are interested in how large H_k can be. In Table 1, the reader may see that $H_k = 1$ for $1 \le k \le 10$. The first cylcotomic polynomial with height $H_k > 1$ is $\Phi_{105}(z)$ which has height 2. The first with height $H_k > 2$ is $\Phi_{385}(z)$ which has height 3. There are three cyclotomic polynomials of order less than 1000 with height 3 and 37 with height 2.

But the situation for $k < 1000$ is misleading. Erdos has shown that for any constant $c > 0$, $H_k > k^c$ for infinitely many k and Maier showed in (12) that this holds for a set of positive density. In Table 2 below we have listed the values of $k < 10^6$ and H_k for which $H_k > H_j$ for $0 < j < k$, that is, the orders k with increasing height H_k. Note, in constructing this table, it is not hard to show that if $H_k > 1$ then k must be a product of three or more primes, and secondly, if $H_k > H_j$ for $0 < j < k$ then k must be odd and square-free.

k	H_k	k	H_k
105 = (3)(5)(7)	2	26,565 = (3)(5)(7)(11)(23)	59
385 = (5)(7)(11)	3	40,755 = (3)(5)(11)(13)(19)	359
1,365 = (3)(5)(7)(13)	4	106,743 = (3)(7)(13)(17)(23)	397
1,785 = (3)(5)(7)(17)	5	171,717 = (3)(7)(13)(17)(37)	434
2,805 = (3)(5)(11)(17)	6	255,255 = (3)(5)(7)(11)(13)(17)	532
3,135 = (3)(5)(11)(19)	7	279,565 = (5)(11)(13)(17)(23)	585
6,545 = (5)(7)(11)(17)	9	285,285 = (3)(5)(7)(11)(13)(19)	1,182
10,465 = (5)(7)(13)(23)	14	327,845 = (5)(7)(17)(19)(29)	31,010
11,305 = (5)(7)(17)(19)	23	707,455 = (5)(7)(17)(29)(41)	35,111
17,255 = (5)(7)(17)(29)	25	886,445 = (5)(7)(19)(31)(43)	44,125
20,615 = (5)(7)(19)(31)	27	983,535 = (3)(5)(7)(17)(19)(29)	59,518

Table 2. Increasing heights of cyclotomic polynomials of order $k < 10^6$

The data for $k < 10^6$ suggests that the growth of H_k is accelerating. There is a jump in the height at $k = 327,845$. What got us interested in this problem was the discovery of a large jump at $k = 1,181,895$ where $H_k = 14,102,773$. This is first k where the height $H_k \geq k$. This motivated us to develop and implement an asymptotically fast algorithm (described below) for computing $\Phi_k(z)$ of large order so we could search for $\Phi_k(z)$ with large height. We have computed the following data for $10^6 < k < 10^8$ where the third column shows the bit-length of H_k.

k	H_k	$\log_2 H_k$	k	H_k	$\log_2 H_k$
983,535	59518	15.86	13441645	1475674234751	40.42
1181895	14102773	23.75	15069565	1666495909761	40.60
1752465	14703509	23.81	30489585	2201904353336	41.00
3949491	56938657	25.76	37495115	2286541988726	41.06
8070699	74989473	26.16	40324935	2699208408726	41.30
10163195	1376877780831	40.32	43730115	862550638890874931	59.58

Table 3. Increasing heights of cyclotomic polynomials of order $10^6 < k < 10^8$.

One sees obvious "jumps" in the data. Note the height $H_k = 862,550,638,890,874,931$ for $k = 43,730,115 = (3)(5)(11)(13)(19)(29)(37)$. This is the first k for which $H_k \geq k^2$.

Since all these k have small prime factors, an obvious approach to try to find $\Phi_k(z)$ with large height is to consider those k which are products of small primes. In (11), Koshiba computed $H_k = 669,606$ for $k = (3)(7)(11)(13)(17)(19)$, the product of the first 7 primes. Using our fast code, we have computed $H_k = 8,161,018,310$ for $k = 111,546,435$, the product of the first 8 primes and $H_k = 2,888,582,082,500,892,851$ for $k = 3,234,846,615$, the product of the first 9 primes. We are still computing cyclotomic polynomials of larger heights. We are maintaining our current results at

`http://www.cecm.sfu.ca/~mmonagan/research/Heights.html`

1.4. Computing the cyclotomic polynomials.

To compute H_k we first compute $\Phi_k(z)$ explicitly. Our algorithm uses the following two facts (see Gallian (7)).

Lemma 3. For k square free let $k = p_1 \times p_2 \times ...p_m$ be the prime factorization of k. If $m = 1$ then $\Phi_k(z) = 1 + z + ... + z^{k-1}$. If $m > 1$ then for any $p|k$,

$$\Phi_k(z) = \frac{\Phi_{k/p}(z^p)}{\Phi_{k/p}(z)}.$$

These two facts give us an algorithm for computing $\Phi_k(z)$ which does a sequence of exact divisions of increasing degree. We show an example

Example 1. To compute $\Phi_{15}(z)$ we first compute $\Phi_3(z) = 1 + z + z^2$. Then

$$\Phi_{15}(z) = \frac{\Phi_3(z^5)}{\Phi_3(z)} = \frac{z^{10} + z^5 + 1}{z^2 + z + 1} = z^8 - z^7 + z^5 + z^3 - z + 1.$$

The algorithm is well known. It is used, for example, by Maple 11 to compute cyclotomic polynomials. Noting that the divisor $\Phi_{k/p}(z)$ is dense of degree $\phi(k/p)$, and that the quotient $\Phi_k(z)$ is dense, even though the dividend $\Phi_{k/p}(z^p)$ is p sparse, using classical division, this last division, which dominates the cost, does $O(\phi(k)\phi(k/p)) = O(p\phi(k/p)^2)$ arithmetic operations in $\mathbb{Z}$. For $k = (3)(5)(7)(11)(13)(17)$, the product of the first 7 primes, this works out to about 1.2 billion arithmetic operations in $\mathbb{Z}$.

To speed up the algorithm, we apply the Fast Fourier Transform to do the polynomial exact division modulo primes and hence to reduce the complexity to $O(\phi(k)\log\phi(k))$ arithmetic operations in $\mathbb{Z}_q$ where q is a machine prime.

We pick the largest primes of the form $q = 2^{27}r+1$ and $q = 2^{26}s+1$ satisfying $q^2 < 2^{63}$ so that multiplications in $\mathbb{Z}_q$ can be done using signed 64 bit machine arithmetic. The largest is $q = 2^{27} \times 17 + 1$. Let p be the largest prime dividing k and let n be the first integer of the form $n = 2^k$ greater than $p\phi(k/p)$, the degree of the dividend in the last division. Let ω be a primitive n'th root of unity in $\mathbb{Z}_q$. Using the discrete fast Fourier transform (DFFT) (see (9), Ch. 4) in $\mathbb{Z}_q$ we compute

$$A = [\Phi_{k/p}(\omega^{ip}) \text{ for } i = 0, 1, ..., n-1] \in \mathbb{Z}_q^n \text{ and}$$

$$B = [\Phi_{k/p}(\omega^{i}) \text{ for } i = 0, 1, ..., n-1] \in \mathbb{Z}_q^n$$

in $O(n \log_2 n)$ arithmetic operations in $\mathbb{Z}_q$. Next we divide pointwise; we compute

$$C = [\frac{A_i}{B_i} \text{ for } i = 0, 1, ..., n-1] \in \mathbb{Z}_q^n.$$

This requires n inverse calculations in $\mathbb{Z}_q$ each of which has constant cost. Note that because the DFFT uses ω of order 2^k, that is, of even order, and because $\Phi_{k/p}(z)|z^{k/p}-1$ and k/p is odd, that is, the roots of $\Phi_{k/p}(z)$ in $\mathbb{Z}_q$ must have odd order, it cannot happen that $B_i = 0$ for any i.

Oberserve that $C_i = \Phi_k(\omega^i)$. Thus we apply the inverse discrete FFT in $\mathbb{Z}_q$ to C to obtain the coefficients of $\Phi_k(z)$ modulo our prime q. We do this for two primes $q_1 = 2^{27}\times 17+1$ and $q_2 = 2^{27}\times 15+1$ and obtain the integer coefficients of $\Phi_k(z)$ using Chinese remaindering. This assumes that $H_k < \lfloor q_1q_2/2 \rfloor < 2^{61}$. To check that we have computed $\Phi_k(z)$ correctly, that is, that H_k is not greater than $\lfloor q_1q_2/2 \rfloor$, we test if $\Phi_{k/p}(z^p) = \Phi_k(z)\Phi_{k/p}(z)$ modulo additional primes using the discrete FFT.

2. Solving Systems Involving Roots of Unity

Let $m(z) = \Phi_k(z)$ be the cyclotomic polynomial of degree $d = \phi(k)$. We present three modular algorithms for solving a linear system $Ax = b$ modulo $m(z)$ for 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 will also assume that the length of the largest integer in $m(z) = \Phi_k(z)$ is bounded by a constant to simplify our analysis. To be precise, assume $||m||_\infty < 10^6$. By Table 2, this is satisfied for all orders $k < 10^6$.

Thus the size of the input $[A|b]$ and $m(z)$ is in $O(n^2dc)$ and the size of the output x is $O(nde)$.

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

The following Lemma (see Huang (13)) characterizes those primes which split $m(z)$ into distinct linear factors. For completeness we give a proof.

Lemma 4. 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 $\mathbb{Z}_p$. $\square$

For algorithms which use Chinese remaindering, we will need to obtain a sequence of primes of the form $p = qk + 1$ for which arithmetic in $\mathbb{Z}_p$ can be done by the hardware of the machine. In our Maple implementations of our algorithms on a 64 bit machine, we use 31 bit primes. The simplest way to do this is to start with the largest integer of the form $qk+1 < 2^{31}$ and to test the integers in the sequence $S = (qk+1, (q-1)k+1, (q-2)k+1, ...)$ for primality. Let $\pi(x)$ be the number of primes less than x and let $\pi(x; k)$ be the number of primes less than x of the form $p = ik + 1$. Direchlet's theorem (see Chapter 9 of (1)) states that $\Pi(x; k) \sim x/(\phi(k) \ln x)$. Thus there should be lots of primes in the sequence S of the required form. But this is an asymptotic result. We seek a result that guarantees a minimum number of primes of the form $qk + 1$ in a range like $[x, 2x]$ so that we could pick a range to guarantee that our algorithms do not run out of primes of the required form. A very useful theorem of Rosser and Schoenfeld (17) states that

$$3/5 \frac{x}{\ln x} < \pi(2x) - \pi(x) < 7/5 \frac{x}{\ln x}.$$

We know of no analagous result for primes in arithmetic progressions.

Now suppose we have found a prime p satisfying the condition in Lemma 4. Then $m(z)$ has $d = \phi(k)$ roots in $\mathbb{Z}_p$. To compute the roots of $m(z)$ one could use a polynomial factorization algorithm. Alternatively, one could use the probabilistic algorithm of Rabin (see (16)) for finding the roots – Maple uses this algorithm. [1]

But for cyclotomic polynomials, it is much faster to construct the roots directly, without need for polynomial arithmetic. Let $m(z) = \Phi_k(z)$ be the k^{th} cyclotomic polynomial and $d = \phi(k)$. Let p be a prime of the form $p = kq + 1$ and let α be a primitive element in $\mathbb{Z}_p$. Then $\beta = \alpha^q$ is a primitive k'th root of unity and $\{\beta^i \text{ for } 0 < i \leq k \text{ such that } \gcd(i, k = 1)\}$ are all primitive k'th roots of unity and hence these are the desired roots of $m(z)$. This observation gives an algorithm for finding the roots of $m(z)$ as follows. Pick $1 < \alpha < p - 1$ at random until we get a primitive element. To test if α is a primitive element we will need the factorization of $p - 1$. This is easy for p a machine prime, and since the density of primitive elements is $\phi(p-1)/(p-1)$, this approach is efficient. Better, however, is the following method which eliminates the need for integer factorization and has a higher probability of success.

[1] 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.

Step 1: Pick $0 < a < p-1$ at random and compute $\beta = a^q$.

Now β is necessarily a k'th root of unity but it may not be primitive. To test if β is primitive we do the following.

Step 2: Compute $S = \{\beta^i \text{ for } 0 < i < k\}$ and check if the elements of S are distinct. If they are then $\beta = a^q$ is a primitive k'th root of unity and the roots of $m(z)$ can be selected from S.

Steps 1 and 2 can be done in $O(k + \log_2 q)$ multiplications and $O(k \log_2 k)$ comparisons of integers modulo p. The following lemma tells us the probability that β is primitive.

Lemma 5. Let $p = kq + 1$ be a prime. Let a be chosen at random such that $0 < a < p$ and let $\beta = a^q$. Then the probability that β is a primitive k'th root of unity is $\phi(k)/k$.

Proof. $\beta = a^q$ is a primitive k'th root of unity implies $\beta^m \not\equiv 1 \bmod p$ for all $0 < m < k$. We will count N the number of integers $0 < m < p$ satisfying $\beta^m \not\equiv 1 \bmod p$. Let α be a primitive element. Then $a = \alpha^i$ for some $0 \le i < p-1$ and $\beta^m = a^{qm} = \alpha^{iqm}$. Thus

$$\beta^m \not\equiv 1 \bmod p \implies \alpha^{iqm} \not\equiv 1 \bmod p.$$

But α is a primitive element so

$$\alpha^{iqm} \not\equiv 1 \bmod p \implies p-1 \nmid iqm \implies qk \nmid iqm \implies k \nmid im.$$

Let $g = \gcd(k, i)$. If $g = 1$ then $k \nmid im$ for all $0 < m < k$ since $m < k$. If $g > 1$ then we have $\frac{k}{g} | \frac{i}{g} m$ for $0 < m = \frac{k}{g} < k$. The the number of integers $0 < i < k$ satisfying $\gcd(i, k) = 1$ is simply $\phi(k)$ and since $p = kq$, $N = \frac{p-1}{k}\phi(k) = q\phi(k)$. Thus the desired probability $N/(p-1) = q\phi(k)/(p-1) = \phi(k)/k$ as required. $\square$

The probability $\phi(k)/k$ is high. If k is prime it is $(k-1)/k$. The lowest it can be is when k is a product of distinct small primes, 2, 3, 5, 7, ... For example, the lowest value of $\phi(k)/k$ for $k < 2000$ is 0.23 for $k = 210$. Observe also that

$$\frac{\phi(k)}{k} \ge \frac{\phi(k)}{k}\frac{\phi(q)}{q} = \frac{\phi(kq)}{kq} = \frac{\phi(p-1)}{p-1},$$

the probability that $0 < \alpha < p$ is a primitive element. The difference can be significant. For example, for $k = 11, q = 6, p = 67$, we have $(\phi(k)/k)/(\phi(p-1)/(p-1)) = 3$. The following result (see Chapter 4 of (1)) bounds how low $\phi(k)/k$ can be.

Theorem 6 (Edmund Landeau, 1903)**.** There exists a constant $c > 0$ such for all $k > 1$

$$\frac{\phi(k)}{k} \ge \frac{ck}{\ln(2 + \ln k)}.$$

Thus we expect to try $O(\ln \ln k)$ values $\beta = a^q$ before we get find one which is a primitive k'th root of unity. Thus we have established the following result.

Theorem 7. For a prime p of the form $p = kq + 1$, the roots of $m(z) = \Phi_k(z)$ in $\mathbb{Z}_p$ can be computed in $O((k + \log_2 q) \log\log k)$ multiplications in $\mathbb{Z}_p$ and $O(k \log k \log\log k)$ comparisons on average.

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 satisfying A is non-singular (mod $m(z)$).

Output: $x \in \mathbb{Q}^n[z]$ which satisfies $Ax \equiv b$ (mod $m(z)$).

```
1:  Set $x^{(0)} = 0$, $P = 1$, and $U = 1$.
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 $p_k$, set $U = p_k \times U$ and 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
```

2.2. Chinese Remaindering

2.2.1. The Algorithm

Algorithm 1 as stated assumes that A is invertible over $\mathbb{Q}$. However, A may not be invertible modulo a prime chosen in Step 3. 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 8 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 8. *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

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

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 14 in Section 2.5 bounds the size of the integer R. This bound 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 – A is proven singular over $\mathbb{Q}$ when the integer U in the algorithm satisfies $U \geq 2|R|$.

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.

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 normally linear in e, the length of the largest integer appearing in the rational coefficients in x. But, 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 can use (up to twice as many) than are necessary to reconstruct the rationals in x. Note also, depending on how the trial divisions in step 14 are implemented, we may need additional primes (see section 2.4).

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 14). But in section 3.1 our linear systems arising in practice illustrate that they can be much smaller. For this reason we state the running time in terms of L and also for $L \in O(cnd)$.

Theorem 9. The running time for Algorithm 1, assuming (i) no unlucky primes are encountered, (ii) $||m(z)||_\infty < 10^6$, and (iii) not counting the cost of the trial divisions $m(z)|(Ax - b)$ (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 n^3dL term is the cost of the linear solves modulo p, the n^2d^2L term is for evaluating A at the d roots modulo p, the n^2dLc term is for reducing the input matrix A modulo p, and the ndL^2 term is the cost of the Chinese remaindering and the rational reconstruction.

Proof. In step 3 the cost of finding a prime p, reducing $m(z)$ modulo p, and computing the roots of $m(z)$ in $\mathbb{Z}_p$ using our algorithm from section 2.1 is dominated by other steps in the algorithm.

In step 4, reduction of the integer coefficients in A and b modulo p 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.

Step 6 evaluates $A_{i,j}(z)$ and $b_i(z)$ at each root $\alpha_{k1}, \cdots, \alpha_{kd}$ modulo p. This costs $O(n^2d^2L)$ because there are n^2+n polynomials to be evaluated. Each requires $O(d)$ arithmetic operations in $\mathbb{Z}_p$ using Horner's rule. This needs to be done for all L primes.

For step 7, 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. Since this is done for each root and each prime, the total cost of step 7 is of $O(n^3dL)$ arithmetic operations in $\mathbb{Z}_p$.

Interpolation in step 10 takes $O(nd^2L)$ arithmetic operations since we only need to interpolate the solution vector which has n elements over the d roots. Notice that interpolation is dominated by the evaluations in step 6.

In step 11 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 $ndO(k)$ for $k = 1..O(L)$, the total cost is $O(ndL^2)$.

If classical Euclid's algorithm is used for rational reconstruction in step 13, 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 $k = 1, 2, 4, 8, 16, ...$ 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. $\square$

2.2.3. The Reconstruction Cost

In our implementation of Algorithm 1, we found, consistent with (2), 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. However, from a practical viewpoint, all one needs is fast integer multiplication and division. This is relevant because Maple 11 is using the GMP integer arithmetic package which has fast integer multiplication and long division but no fast Euclidean algorithm yet, hence, no fast Chinese remaindering and rational reconstruction are available.

One may essentially 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 $Av \equiv b \mod m(z) \mod Q$ 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 the integer 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 mod P and b is the LCM of the denominators of all rationals reconstructed so far. The idea is to apply rational reconstruction to $b \times u$ mod P instead. We refer the reader to (2) 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 Maple 11 uses for computing inverses and rational reconstruction.

2.3. Linear p-adic Lifting

2.3.1. The Algorithm

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 $A.x = b$ (mod $m$)
 1: Find a machine prime $p$ s.t. $m$ splits linearly over $\mathbb{Z}_p$, and compute the roots $\alpha_1, .., \alpha_d$
    of $m(z)$ mod $p$
 2: Let $e_0 = b$, $x^{(0)} = 0$
 3: Invert $A(\alpha_i)$ mod $p$ 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 $e_k$ mod $p$
 6:   for $i = 1$ to $d$ do
 7:     Evaluate the error $e_k$ at $z = \alpha_i$ mod $p$.
 8:     Compute $x_{ki} \equiv A(\alpha_i)^{-1}.e_{ki}$ mod $p$
 9:   end for
10:   Interpolate $x_k(z)$ from $(\alpha_1, x_{k1})$, ...,$(\alpha_d, x_{kd})$.
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 $x^{(k+1)}$ mod $p^{k+1}$.
15:     If rational reconstruction succeeds and $m(z)|(A.x - b)$ then output $x$.
16:   end if
17: end for
```

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. Assuming the primes used by Algorithms 1 and 2 are of the same length, L the number of lifting steps in Algorithm 2 is approximately equal to L the number of primes used by Algorithm 1.

Theorem 10. The running time for Algorithm 2, assuming (i) the prime p chosen in step 1 is not unlucky, (ii) $||m(z)||_\infty < 10^6$, and (iii) not counting the cost of the trial divisions $m(z)|(Ax-b)$, 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 n^3d term is the cost of computing the d matrix inverses and the n^2d^2Lc term is the cost of computing the error e_k in step 11. The ndL^2 term is the cost of step 12 which is a conversion from the p-adic representation of the solution to an integer representation. The ndL^2 term is the cost of rational reconstruction.

Before we give the proof we bound $||Ax_k \mod m(z)||$ that appears in step 11 and thus prove that and the size of the integers in the error e_k is bounded. Notice that the bound necessarily depends on $||m||_\infty$. Our simplifying assumption that $||m||_\infty < 10^6$ means that this factor in the bound may be ignored.

Lemma 11. Bounds on the error e_k.
(i) $||Ax_k \bmod m(z)|| \le (p-1)nd||A||(1+||m||_\infty)^{d-1}$
(ii) $||e_k|| \le nd||[A|b]||(1+||m||)^{d-1}$.

Proof. Since $x_k \in \mathbb{Z}_p^n[z]$ we have $||x_k|| < p$ and so $||Ax_k|| \le (p-1)nd||A||$ where the factor of n comes from the matrix vector multiplication and the factor d comes from multiplying polynomials of degree $< d$. Applying Lemma 2 to divide the polynomials in Ax_k by $m(z)$ gives (i).

We prove (ii) by induction on k. Since $e_0 = b$, (ii) holds at step $k = 0$. In step 11 the algorithm computes $e_{k+1} := (e_k - Ax_k \bmod m(z))/p$. Assuming (ii) is true for k, then

$$||pe_{k+1}|| \;=\; ||e_k - Ax_k \bmod m(z)|| \;\le\; ||e_k|| + ||Ax_k \bmod m(z)||.$$

Substituting for $||e_k||$ (induction assumption) and for $||Ax_k \bmod m(z)||$ from (i) have

$$\begin{aligned} ||pe_{k+1}|| &\le nd||[A|b]||(1+||m||_\infty)^{d-1} + (p-1)nd \parallel [A|b] \parallel (1+||m||_\infty)^{d-1} \\ &= pnd \parallel [A|b] \parallel (1+||m||_\infty). \end{aligned}$$

Dividing both sides by p we obtain (ii) is true by induction for all k. □

Proof of theorem 10. In Algorithm 2 we only need one prime p that splits $m(z)$ over $\mathbb{Z}_p$. The time for computing this may be ignored. In step 3 we pre-compute the inverse of the input matrix A at each root d modulo p using Gaussian elimination. This costs $O(dn^3)$ arithmetic operations in $\mathbb{Z}_p$. 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.

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 $O(nd^2L)$ – 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 $m(z)$. Under our assumption that $||m||_\infty < 10^6$, this is dominated by the matrix vector multiplication, and not division by $m(z)$, 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, not counting the cost of the trial divisions in step 15 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 parts is the computation of the error in step 11, in particular, the matrix vector multiplication in Ax_k which needs to be computed over $\mathbb{Z}$. This requires n^2 polynomial multiplications and n divisions by $m(z)$. It has complexity $O(n^2d^2c)$ assuming classical multiplication. Since the size of A is $O(n^2dc)$, we cannot reduce the complexity of computing the error by more than a factor of d. We have attempted to do this by choosing primes $p_1, p_2, ...$ such that $m(z)$ has d roots α_{ij} in $\mathbb{Z}_{p_i}$ – as we do for Algorithm 1 – evaluating $A(\alpha_{ij}) \bmod p_i$, caching these primes, the α_{ij} and the matrices $A(\alpha_{ij}) \bmod p_i$ for re-use in the next lifting step, multiplying $A(\alpha_{ij})x_k(\alpha_{ij}) \bmod p_i$, interpolating, and then Chinese remaindering to recover $e_{k+1} \in \mathbb{Z}[z]$. For this one needs a tight bound on $||e_{k+1}||$. In section 3 we will see that the improvement obtained is very good for randomly generated problems which have large rationals in the solution vector but not good on our real data sets where L the number of lifting steps is small.

2.3.4. The reconstruction cost.

Another expensive component of Algorithm 2 is the reconstruction cost $O(ndL^2)$ when L is large. We have already mentioned how the rational reconstruction cost can be reduced to $O(L^2 + ndM(L)\log L)$ where $M(L)$ is the cost of multiplying integers of size L. The cost of step 11 can similarly be reduced from $O(ndL^2)$ to $O(L^2 + ndM(L)\log L)$, as follows. First observe 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 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 $O(ndM(L)\log L)$.

2.4. Trial Division

Algorithms 1 and 2 both terminate when rational reconstruction of x succeeds and $m(z)|b - Ax$ over $\mathbb{Q}$. We use the rational reconstruction in (15), which builds in some redundancy so that if it succeeds, the output x is correct with high probability. One way to speed up the trial divisions is to avoid arithmetic with the fractions that appear in x. If we compute D the least common multiple of the denominators of all fractions appearing in the coefficients of the polynomials in x, Dx clears the fractions in x. We test if $m(z)|Db - A(Dx)$. Here all arithmetic is over $\mathbb{Z}$ since $m(z)$ is monic over $\mathbb{Z}$. This is quite effective in practice. In our experiments, the time spent doing trial divisions this way was always less than 10% and typically 1-2% 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 by 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 12. If rational reconstruction succeeds in Algorithms 1 and 2 and the modulus M satisfies

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

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

Remark: because $m(z)$ is a cyclotomic polynomial, $||m||_\infty$ is small as noted in the introduction. For $||m(z)||_\infty = 1$, Theorem 12 requires $M > 2^d(D||b|| + ndN||A||)$. This is not much longer than $D||b|| + N||A||$.

Now we consider the size of the integer D in Theorem 12. Consider $x = [1/p + z/q + z^2/r]$ with p, q, r distinct primes. Here $D = LCM(p, q, r) = pqr$ is three times longer than the size of the rationals in x and hence, if we apply Theorem 12, we may need additional primes in Algorithm 1. In practice, this situation does not normally happen – D is not significantly larger than the LCM of the denominators in x. But it does mean that using Theorem 12 may increase the number of primes needed by Algorithm 1 (the number of lifting steps needed by Algorithm 2).

In practice, on both the random data and real problems in section 3, because of the redundancy of the rational reconstruction algorithm in (15), after rational reconstruction succeeded, the value of M in Algorithms 1 and 2 always satisfied Theorem 12 with no additional primes (lifting steps resp.) needed. Even for those problems in data set 2 for which one 31 bit prime was sufficient to reconstruct the solution vector x – Theorem 12 was satisfied with zero additional primes (lifting steps resp.) 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 (10)) to bound $||\det(A)||_\infty$.

Lemma 13 (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 14. 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 $D > 10^{cnd}$.

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 the integer coefficients in $\det A \bmod m(z)$ and the $\det(A^{(j)}) \bmod m(z)$ using Chinese remaindering, we will need bounds on their height. We state these in the following lemma.

Lemma 15.

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

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 $||\det A \bmod m(z)||_\infty$ yields the first result. The second result follows by noting that Lemma 13, which is stated in terms of the rows of A, also applies to the columns of A. $\Box$

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 effectively reduces the integer Chinese remaindering cost 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).

In comparing Algorithm 1 and Algorithm 3, since Algorithm 3 needs to reconstruct integers (not rationals) of length d times smaller than Algorithm 1 in general, it needs a factor of approximately $2d$ fewer primes and hence will be $2d$ times faster than Algorithm 1. However, the size of the rationals in the solution vectors of real applications may be much smaller than the bound in Lemma 14. So Algorithm 3 is not necessarily faster than Algorithms 1 and 2.

3. Implementation and Timings

We have implemented Algorithms 1, 2 and 3 in Maple. Our implementation of Algorithms 1 and 2 includes the optimizations described in 2.2.3 and 2.3.3 for reducing the reconstruction cost, 2.3.4 for speeding up the computation of the error, and they apply Theorem 12 instead of doing trial division. We use 25 bit floating point primes on 32 bit machines, and 31 bit integer primes on 64 bit machines so that we can use the C codes

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 distinct primes such that $\Pi p_i > 2B$ and $m(z)$ splits
   into distinct linear factors modulo $p_i$.
3: Call Subroutine C with inputs $A, b, m$ and $P$.
```

Algorithm 4 Subroutine C

```
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 $h = \lfloor k/2 \rfloor$.
3:     Set $(D_1, X_1, M_1) = C(A, b, m, \{p_1, p_2, ..., p_h\})$.
       Set $(D_2, X_2, M_2) = C(A, b, m, \{p_{h+1}, ..., p_k\})$.
4:     Set $M = M_1M_2$ – note, this is the product of the primes used which is not neces-
       sarily equal to $\Pi_{i=1}^k p_i$.
5:     Compute $i = M_1^{-1} \bmod M_2$.
6:     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 $M_2$.
7:     Set $D = D_1 + \Delta_D M_1 \in \mathbb{Z}_M[z]$.
       Set $X = X_1 + \Delta_X M_1 \in \mathbb{Z}_M[z]^n$.
8:     Ouput $(D, X, M)$
9: else
10:    Compute the roots $\alpha_1, ..., \alpha_d$ of $m(z) \bmod p_1$.
11:    Set $A = A \bmod p_1$ and $b = b \bmod p_1$
12:    for $i = 1$ to $d$ do
13:        Evaluate $A$ and $b$ at $z = \alpha_i \bmod p_1$.
14:        Solve $A(\alpha_i).x_i \equiv b(\alpha_i) \bmod p_1$ for $x_i \in \mathbb{Z}_{p_1}^n$ and compute $D_i = \det A(\alpha_i) \bmod p_1$.
15:        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 $p_1 = p$.
16:    end for
17:    Interpolate $D \in \mathbb{Z}_p[z]$ from $(\alpha_1, D_1), ..., (\alpha_d, D_d)$.
18:    Interpolate $x \in \mathbb{Z}_p[z]^n$ from $(\alpha_1, x_1), ..., (\alpha_d, x_d)$.
19:    Output $D, x, p_1$.
20: end if
```

in Maple's `LinearAlgebra:-Modular` package which provide fast BLAS based routines for doing linear algebra over $\mathbb{Z}_p$. These codes are for dense linear systems.

We compare our algorithms on three data sets described below. All timings were obtained using Maple 11 on an AMD® Opteron 150 processor running @ 2.4 GHz with 2GB of RAM. This is a 64 bit single core processor.

Data Set 1: Randomly Generated Inputs

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 data consists of systems of dimension $10, 20, 40, 80, 160$ 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 of degree 5 and coefficients uniformly chosen at random from $[0, 2^c)$.

n	Coefficient length c in binary digits										Remark
	2	4	8	16	32	64	128	256	512	1024	
10	.046	.096	.223	.498	1.242	2.823	6.981	18.47	51.92	158.8	CRT
	.057	.081	.116	.258	.552	1.291	3.359	9.511	30.00	104.2	Lift 2
	.054	.118	.231	.451	.956	3.260	6.981	16.22	43.55	118.4	Padic
	.009	.011	.013	.020	.051	.090	.147	.320	.643	1.274	Cramer
20	.202	.414	.868	1.892	5.762	14.70	37.34	104.2	310.8	930.3	CRT
	.155	.250	.487	1.066	2.578	6.297	16.87	49.17	164.3	612.5	Lift2
	.314	.698	1.442	2.981	6.358	23.20	50.47	115.0	310.8	827.4	Padic
	.030	.040	.056	.081	.187	.357	.718	1.427	2.988	6.396	Cramer
40	.877	2.034	4.252	9.412	32.88	86.82	226.0	644.1	2038	–	CRT
	.630	1.270	2.539	5.291	13.74	33.74	93.29	291.2	1030	4142	Lift2
	2.104	4.706	10.08	21.67	46.85	190.7	411.8	916.7	2415	–	Padic
	.146	.219	.307	.438	1.121	2.340	4.711	9.405	20.41	45.77	Cramer
80	11.44	15.13	32.54	71.59	250.8	659.3	1780	–	–	–	CRT
	3.564	6.972	14.01	30.11	80.11	197.1	605.0	1998	–	–	Lift2
	16.04	36.36	79.89	177.4	383.0	1526	3203	–	–	–	Padic
	1.046	1.575	2.197	3.279	10.17	24.10	50.89	108.6	239.7	537.2	Cramer
160	91.23	120.9	264.8	581.1	2283	–	–	–	–	–	CRT
	54.14	53.47	110.4	239.3	660.0	1629	4584	–	–	–	Lift2
	251.4	318.3	706.8	1650	3528	–	–	–	–	–	Padic
	8.712	13.13	18.15	27.96	112.2	262.6	584.0	1197	2317	4546	Cramer
$m(z) := z^6 + z^5 + z^4 + z^3 + z^2 + z + 1$, $d = 6$											

Table 4. Runtimes (in CPU seconds) "–" denotes the running time is over 5000 seconds.

Table 4 shows the running time of dense random polynomial inputs for Algorithm 1 (CRT), Algorithm 2 (Padic), Algorithm 2 with the optimization for computing the error (Lift2), and Algorithm 3 (Cramer). We clearly see the advantage of Algorithm 3 here and that the optimized Padic lifting method is faster than Algorithm 1 as n gets larger.

Data Sets 2 & 3: Real Problems

The problems in these data sets were given to us by Vahid Dabbaghian. They include systems with various dimensions, coefficient lengths, and minimal polynomials where, note, the dimenion n of the system is indicated in the filename. The systems are available at `http://www.cecm.sfu.ca/CAG/code/VahidsSystems.zip`

file	sys49	sys100	sys100b	sys144	sys196	sys225	sys256	sys576	sys900
$deg_z(m)$	4	8	4	2	2	4	4	6	8
k	5	24	8	4	3	5	12	7	24
$\|\|A\|\|_\infty$	10	5	2	4	11	2	3	3	2
$\|\|x\|\|_\infty$	45	14	1	1	229	875	2	1	2
L	4	1	1	1	9	36	1	1	1
CRT	.144	.788	.029	.036	3.344	3.056	.155	.842	2.358
Padic	.109	.443	.030	.029	1.183	2.374	.174	.612	2.761
Lift 2	.111	.294	.100	.163	1.973	1.678	.640	3.022	7.627
Cramer	.293	4.159	.305	.147	6.206	4.644	3.748	53.69	338

Table 5. Runtime (in CPU seconds) on some of the systems given by Vahid Dabbaghian.

file	144Huge	196Huge	256Huge	256Huge2	324Huge	400Huge	484Huge
$deg_z(m)$	40	12	4	24	16	108	88
k	55	13	12	39	17	171	276
$\|\|A\|\|_\infty$	83	57	596	129	196	707	808
$\|\|x\|\|_\infty$	159	108	90010	255	573	2202	2504
L	8	7	4096	16	16	128	128
CRT	21.19	6.808	3904	64.01	67.97	12063	13653
Padic	22.62	5.850	1908	49.16	63.41	8536	8632
Lift 2	12.11	3.997	3349	60.68	60.59	26816	36529

Table 6. Runtime (in CPU seconds) on some of the $\underline{\text{huge}}$ systems given by Vahid Dabbaghian.

Tables 5 and 6 show running times for systems given to us by Vahid Dabbaghian. The labeling of the algorithms is the same as in Table 4. Most of the input systems in data set 2 are sparse. All solution vectors have small rationals, some very small. The problems in data set 3 are dense systems with larger integers in the input and solution vector x. To indicate this we have shown the maximum length of the coefficients in the inputs A and b and in the output x, in binary digits. In addition, we show L, the number of machine primes that are needed to construct the solution vector in the Chinese remaindering approach. For the given inputs, this is the same as the number of lifting steps needed in the linear p-adic lifting algorithm.

The data shows that Algorithm 3 performs very poorly on these inputs. It also shows that the optization for computing the error in Algorithm 2 is not always helpful and that

Algorithm 1 is competitive with both versions of Algorithm 2. Our conclusion is that running Algorithms 1 and 3 simultaneously (which is not difficult to implement) would be the best practical solution.

References

[1] Paul Bateman and Harold Diamond. *Analytic Number Theory – An Introductory Course.* World Scientific, 2004.

[2] 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.

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

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

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

[6] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computation using efficient block projections. To appear in *Proceedings of ISSAC '07*, ACM Press, 2007.

[7] Joseph Gallian. *Contemporary Abstract Algeba* 5th edition, Houghton Mifflin College Div., 2001.

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

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

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

[11] Y. Koshiba. On the calculations of the coefficients of the cyclotomic polynomials. Kagoshima Univ. Faculty of Science Report No. 31, 31–44, 1998.

[12] H. Maier. The size of the coefficients of cyclotomic polynomials. *Analytic Number Theory*, Birkhäuser, **2** 633–639, 1996.

[13] 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.

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

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

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

[17] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, **6** 64–94, 1962.

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

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