

# An Algorithm to Solve Integer Linear Systems Exactly Using Numerical Methods

Zhendong Wan

*Department of Computer Science, Drexel University, Philadelphia, PA*

---

## Abstract

In this paper, we present a new algorithm for the exact solutions of linear systems with integer coefficients using numerical methods. It terminates with the correct answer in well-conditioned cases or quickly aborts in ill-conditioned cases. Success of this algorithm on a linear equation requires the linear system must be sufficiently well-conditioned for the numeric linear algebra method being used to compute a solution with sufficient accuracy. Our method is to find an initial approximate solution by using a numerical method, then amplify the approximate solution by a scalar, adjust the amplified solution and corresponding residual to integers so that they can be computed without large integer arithmetic involved and can be stored exactly. Then we repeat these steps to refine the solution until sufficient accuracy is achieved, and finally reconstruct the rational solution. Our approximating, amplifying, and adjusting idea enables us to compute the solutions without high precision software floating point operations involved in the whole procedure or big integer arithmetic involved except at the final rational reconstruction step. We will expose the theoretical cost and show some experimental results.

*Key words:* linear systems, numerical linear algebra methods, rational solvers

---

## 1 Introduction

Both symbolic methods and numerical methods can be used to solve linear systems. But they use different techniques and have been developed independently. Symbolic methods for solving linear systems are based on modular methods via solving the linear system modulo a large integer and finally reconstructing the rational solution. Either  $p$ -adic lifting [Dixon (1982); Moenck and Carter (1979)] or computation modulo many different primes and using

---

*Email address:* wan@cs.drexel.edu (Zhendong Wan).

Chinese remainder theorem are the usual way to compute solutions of linear systems modulo a large integer. Numerical methods use either direct methods like Gaussian Elimination (with or without pivoting),  $QR$  factorization, or iterative methods such as Jacobi's method, Lanczos' method, or the GMRES method - see [Demmel (1997); Saad (2003); Trefethen and D. Bau (1997)] for details. Symbolic methods for solving linear systems can deliver the correct answer without any error, though they are often more expensive in computation time than numerical methods. But numerical linear algebra methods for solving linear systems are subject to the limitation of floating point precision and iterative methods are subject to additional convergence problems.

In this paper, we combine the two approaches. We describe a new algorithm to exactly solve well-conditioned integer linear systems using numerical methods. It aborts quickly in ill-conditioned cases. Success of this algorithm requires the system to be sufficiently well-conditioned for the numeric linear algebra method being used to compute a solution with sufficient accuracy. Though this algorithm has the same asymptotic cost as  $p$ -adic lifting, it yields practical efficiency. Since over the past few decades, hardware floating point operations have been sped up dramatically, from a few hundred FLOPS in 1940s to a few GFLOPS now, even in PCs. Also many high performance numerical linear algebra packages are developed using fast BLAS implementation for dense systems.

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

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

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

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

procedure or big integer arithmetic involved except at the final rational reconstruction step. The details will be discussed in section 3. The scaling idea has been used for a long time in numerical linear methods in a different way. See e.g. (Forsythe and Moler, 1967, Chapter 2). Numerical methods often simply use it as preconditioners.

In this paper, in discussion of performance, we use  $M(l)$  to denote the bit operations required to multiply two integers with bit length at most  $l$ . By Schönhage and Strassen algorithm (von zur Gathen and Gerhard, 1999, Theorem 8.24),  $M(l) = O(l \log l \log \log l)$ . We use  $\mathcal{L}(A)$  to denote the maximum bit length of entries of integer matrix  $A$ . Also we use  $O^\sim(f(n))$  to denote  $O(f(n)^{1+o(1)})$ , thus ignoring log factors.

The next section is a classic result about the rational reconstruction. The continued fraction which gives the best approximation of a real number enables us to reconstruct a rational number with certain constraints from a real number. Then in the following section we describe a way how to achieve arbitrary accuracy using numerical linear methods, on the condition that inputs are integers and matrices are well-conditioned. Finally the potential usage of our new algorithms for sparse linear systems is demonstrated with a challenge problem.

## 2 Continued fraction

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

$$r = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}},$$

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

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

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

- (1) Every rational number can be associated with a finite continued fraction. Irrational numbers can also be uniquely associated with simple continued fractions. If we exclude the the finite fractions with the last quotient equal to 1, then the correspondence between rational numbers and finite continued fractions becomes one to one.
- (2) For all  $k \geq 2$ ,  $p_k = a_k p_{k-1} + p_{k-2}$ ,  $q_k = a_k q_{k-1} + q_{k-2}$ .
- (3)  $q_k p_{k-1} - p_k q_{k-1} = (-1)^k$ . And  $s_k - s_{k-1} = (-1)^{k-1} / (q_k q_{k-1})$ .
- (4)  $s_0 < s_2 < s_4 < s_6 < \dots < r < \dots < s_7 < s_5 < s_3 < s_1$ .

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

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

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

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

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

Then

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

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

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

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

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

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

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

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

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

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

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

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

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

If  $r = \frac{n}{d}$ , the  $(p_k, q_k)$  can be computed by the half gcd algorithm - please see e.g. (von zur Gathen and Gerhard, 1999, Corollary 11.10). It needs  $O(M(l) \log l)$  bit operations.

### 3 Exact solutions of integer linear systems

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

It is well known that numerical linear algebra methods are inexact and the accuracy of the solutions is limited to machine precision (or the software floating point precision), when carried out on a computer. Iterative refinement methods (see e.g. [Forsythe and Moler (1967); Demmel (1997); Geddes and Zheng

(2002)) can be used to refine the solution. The refinement, [Geddes and Zheng (2002)] in which a portion, not all, of high precision floating point arithmetics is replaced by lower precision floating point arithmetics, is very efficient in computing a solution with a pre-set high precision. It works this way: for input  $A$ ,  $b$ , and a pre-set precision, initially solve  $Ax = b$  in the hardware floating point precision (lower than the pre-set high precision), repeat the following steps enough times to refine the answer until the desired accuracy is achieved,  $r = b - Ax$  in a higher precision proportional to the step count, solve  $A \cdot \Delta x = r$  in the hardware floating point precision, update  $x = x + \Delta x$  in a higher precision proportional to the step count. These refinement methods help compute solutions with a reasonable high accuracy at affordable practical run time cost and worst case complexity cost. While in symbolic linear algebra, the high accuracy of solutions required in order to reconstruct the exact answer makes these iterative methods impractical. All these refinement methods above require floating point operations in high precision. And the cost of one software floating point operation increases rapidly with respect to the precision, which has been illustrated in (Geddes and Zheng, 2002, Figure 1). Also these refinement methods lead to more expensive complexity for the worst case than the  $p$ -adic lifting method. Thus for symbolic computation, we need a better refinement method. Our approximating, amplifying and adjusting idea works this way: for input integer matrix  $A$  and integer vector  $b$ , initialize the solution  $x = \frac{1}{d} \cdot n$  with  $d = 1$  and vector  $n = 0$ , initialize  $r = b$ , repeat the following steps enough times to achieve any desired accuracy: find an approximate solution  $A \cdot \Delta x = r$  in the hardware floating point arithmetic, choose a suitable scalar  $\alpha$ , amplify and adjust the  $\Delta x$  with rationals  $\frac{1}{\Delta d} \cdot \Delta n$  by setting  $\Delta d = \alpha$ ,  $\Delta n = (\approx \alpha \cdot \Delta x_1, \dots, \approx \alpha \cdot \Delta x_n)$ , update answer:  $n = \alpha \cdot n + \Delta n$  and  $d = \Delta d \cdot d$ , update residual  $r = \alpha \cdot r - A \cdot \Delta n$ . In each refinement iteration, the components of the amplified and adjusted residual are integers of magnitude at most  $\|b\|_\infty + \|A\|_\infty$ . Thus our method can be used to achieve the arbitrary high accuracy of the answers without high precision software floating point arithmetic involved in the whole procedure or big integer arithmetic involved except the final rational reconstruction step. After sufficient accuracy is achieved, the final rational solution can be reconstructed.

### 3.1 Rational solver for dense integer linear systems

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

**Algorithm 1** *Rational solver for dense case*

*Input:*

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

Output:

- $x$ , a rational vector, solution of  $Ax = b$ . Or it aborts with message "insufficient numerical accuracy".

Procedure:

- (1) Compute a LUP factorization of  $A$  in floating point arithmetic using a backward stable numeric method. [Other factorizations may be used here too.]
- (2) Set integer  $d^{(0)} := 1$ . [The common denominator of the answer.]
- (3) Set integer vector  $r^{(0)} := b$ . [The residual.]
- (4) Set integer  $i := 0$ . [Step counter]
- (5) Compute integer  $B$ , the Hadamard bound of  $A$ , which bounds the determinant and all  $(m - 1) \times (m - 1)$  minors of  $A$ .
- (6) repeat the following steps until  $d^{(i)} > 2m \cdot B^2(2^{-i}\|b\|_\infty + \|A\|_\infty)$ .
  - 6.1  $i := i + 1$ .
  - 6.2 Compute  $\bar{x}^{(i)} = A^{-1}r^{(i-1)}$  in floating point arithmetic using the LUP factorization from step 1. [An approximate solution  $Ax = r^{(i-1)}$ .]
  - 6.3 Compute the integer scalar,  $\alpha^{(i)} := \min(2^{30}, 2^{\lfloor \log_2(\frac{\|r^{(i-1)}\|_\infty}{\|r^{(i-1)} - A\bar{x}^{(i)}\|_\infty}) - 1 \rfloor})$  in floating point arithmetic; [For the purpose of high performance,  $\alpha$  is better chosen to a power of 2. Since in this case, operations of integers multiplying the  $\alpha$  can be replaced by more efficient operations of shifting bits. For small size systems, the approximate maybe equals to the exact answer. So that the constant will make the algorithm not abort, since in that case  $\alpha$  becomes positive infinity. Also 30 is chosen so that the amplified and adjusted solution and residual can be exactly represented by hardware double floating point numbers.]
  - 6.4 If  $\alpha^{(i)} < 2$ , abort with error message "insufficient numerical accuracy". [This situation happens only if  $A$  is ill-conditioned. The author rarely encountered with the ill-conditioned cases in practice.]
  - 6.5 Exactly compute integer vector  $x^{(i)} := (\approx \alpha^{(i)} \cdot \bar{x}_1^{(i)}, \dots, \approx \alpha^{(i)} \cdot \bar{x}_m^{(i)})$ .  $x^{(i)}$  is the nearest integer of  $\alpha^{(i)} \cdot \bar{x}^{(i)}$  component-wise, such that  $\|x^{(i)} - \alpha^{(i)} \cdot \bar{x}^{(i)}\|_\infty \leq 0.5$ . [Amplify and adjust]
  - 6.6 Exactly compute integer  $d^{(i)} := d^{(i-1)} \cdot \alpha^{(i)}$ .
  - 6.7 Exactly compute integer vector  $r^{(i)} := \alpha^{(i)} \cdot r^{(i-1)} - Ax^{(i)}$ . [Amplify the residual by a scalar.]
- (7) Set  $k := i$
- (8) Compute integer vector  $n^{(k)} = \sum_{1 \leq i \leq k} \frac{d^{(k)}}{d^{(i)}} \cdot x^{(i)}$ , noting  $\frac{d^{(k)}}{d^{(i)}} = \prod_{i < j \leq k} \alpha^{(j)}$ .
- (9) Reconstruct rational solution  $x$  from  $\frac{1}{d^{(k)}} \cdot n^{(k)}$  using Theorem 1 with denominators bounded by  $B$ .
- (10) Return  $x$ .

Note:

- (1) This algorithm requires the input matrix to be sufficiently well-conditioned for the numeric solver being used to work successfully. When adequate condition is in doubt, it is a nice and important property of our algorithm that we quickly detect the failure rather than continuing a nonsensical successive refinement.
- (2) In solving a linear system, usually estimating the residual is easier than estimating the error. Obviously the infinity norm of the error is bounded by the product of the infinity norms of the inverse of the input matrix and the residual. In the  $i$ th iterator,  $r^{(i)}$ , the infinity norm of the residual is approximately  $\frac{1}{\alpha^{(i)}}$  of the previous one, by the choice of  $\alpha^{(i)}$ . Thus the infinity norm of the residual is approximately  $\frac{1}{d^{(i)}}$ . So  $e^{(i)}$ , the infinity norm of the error is at most approximately  $\frac{\|A^{-1}\|_\infty}{d^{(i)}}$ . By Cramer's rule, we know  $\|A^{-1}\|_\infty \leq mB$ . Thus roughly speaking,  $e^i \leq \frac{mB}{d^{(i)}}$ . Careful analysis shows that  $r^i \leq \frac{\frac{\|b\|_\infty + \|A\|_\infty}{2^i}}{d^{(i)}}$ . Thus if the loop ends without abortion, the error will be at most  $\frac{1}{2Bd^{(k)}}$ , and then by theorem 1, the correct solution can be constructed from the approximate solution  $\frac{n^{(k)}}{d^{(k)}}$ .

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remarks:

- (1) The asymptotic time complexity is comparable with the Dixon lifting algorithm Dixon (1982). For an  $n \times n$  well-conditioned matrix with entries of bit length of at most a fixed number, both algorithms requires  $O^\sim(n^3)$  bit operations.
- (2) The idea of (Pan and Wang, 2002, section 4) can be used to accelerate the final rational reconstruction step. In practice, often only a few rational reconstructions instead of  $n$  rational reconstructions are done.
- (3) In implementation, it is possible to choose all  $\alpha^{(i)}$  the same, which usually depends on the numerical linear algorithm and the condition number of the matrix.
- (4) In implementation, in each iteration, we may detect if the  $\alpha^{(i)}$  is over computed by checking if the infinity norm of the residual computed in step 6.7 is as small as expected in theory. If the  $\alpha^{(i)}$  is over computed, we can reduce  $\alpha^{(i)}$  to half and try. In practice, we haven't encountered this case.

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

### 3.1.1 Total cost for well-conditioned matrices

In practice, the matrices are often well-conditioned. And ill-conditioned matrices are rarely encountered in the author's experience. If the matrix is well-conditioned, a backward stable (see [Trefethen and D. Bau (1997)] for details) factorization such as Gaussian elimination with (partial) pivoting and QR factorization is suitable to be used in this algorithm. For this case, the algorithm doesn't abort, but terminates with the correct rational solution. The entry of the amplified and adjusted residual in each iteration will be bound by  $\|b\|_\infty + \|A\|_\infty$ . In order to estimate the cost, we need to estimate  $\|\bar{x}^{(i)}\|_\infty$ . If the non-singular integer matrix  $A$  is well-conditioned (i.e.,  $\|A\|_\infty \cdot \|A^{-1}\|_\infty$  is reasonable small), then  $\|A^{-1}\|_\infty$  is small since  $\|A\|_\infty$  is larger than or equal to 1. In each iteration,

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

So each iteration needs  $O^\sim(m^2)$  floating point operations and  $O^\sim(m^2(\mathcal{L}(A) + \log \|b\|_\infty))$  machine word size integer operations. Now we estimate the number of iterations. We know the Hadamard bound  $B$  and  $\log B = O^\sim(m\mathcal{L}(A))$ . So the number of iteration required is  $O^\sim(m\mathcal{L}(A) + \log \|b\|_\infty)$ . Thus the total loop costs

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

The computation of  $A^{-1}$  costs  $O(m^3)$  floating point operations, the computation of  $n^{(k)}$  costs  $O^\sim(m^2(\mathcal{L}(A) + \log \|b\|_\infty))$  bit operations by using divide-and-conquer method (Cormen et al., 2001, Section 2.3.1) and FFT based fast integer arithmetic. The final rational reconstruction in Theorem 1 will cost  $O^\sim(m^2(\mathcal{L}(A) + \log \|b\|_\infty))$  bit operations. So the asymptotic cost for the worst case is

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

### 3.1.2 Experimentation on dense linear systems

The following table is the comparison of the running time of three different methods. All are implemented in C/C++. Plain\_Dixon is an implementation

of Dixon lifting without calling BLAS in LinBox<sup>1</sup>. Dixon\_CRA\_BLAS is an implementation by Z. Chen and A. Storjohann [Chen and Storjohann (2004)]. This method uses the idea of FFLAS [Dumas et al. (2002a)] and a mixture of Dixon lifting and the Chinese remainder algorithm. Dsolver is our simple implementation of algorithm 1, using LAPACK routines implemented in ATLAS<sup>2</sup>.

Table 1

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

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

### 3.2 Rational solver for sparse integer linear systems

The main idea is to apply the algorithm 1 by replacing the numeric solver with a successful sparse linear system solver. In the case of sparse matrices the al-

<sup>1</sup> LinBox is an exact computational linear algebra package under development, [www.linalg.org](http://www.linalg.org)

<sup>2</sup> ATLAS is a linear algebra software and provides C and Fortran77 interfaces to a portably efficient BLAS implementation, as well as a few routines from LAPACK and is available at <http://math-atlas.sourceforge.net>

gorithm can be especially fast if a sparse numeric solver works for the matrix at hand. Iterative methods using a few matrix-by-vector products could be used such as Lanczos method, or Jacobi’s method, the GMRES method (Generalized Minimum Residual method (with restarts)), CG method (Conjugate Gradient method), and BICG method (BiConjugate Gradient method) [Saad (2003); Trefethen and D. Bau (1997)]. If the matrices are general sparse matrices with many zero entries, sparse elimination methods such as SuperLU could also be used. It is not the purpose of this paper to examine these issues, such as choice of preconditioner, associated with sparse numeric solvers, but we point out that when the chosen numeric solver does succeed, then our method can then get to the exact solution extremely rapidly. In next section, our example from Trefethen’s challenge suite shows a case where an iterative method works after applying a custom preconditioner adapted to the specific matrix.

#### 4 Application to a challenge problem

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

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

Though only the first 10 decimal digits are required for the original problem, in year 2002, an *exact (symbolic)* solution was computed by Jean-Guillaume Dumas of LMC-IMAG in Grenoble, France. It is a fraction with exactly 97,389 digits each for relatively prime numerator and denominator. He ran 182 processors for four days using Wiedemann’s algorithm [Wiedemann (1986)] and the Chinese remainder theorem - see [Dumas et al. (2002b)] for details. A few months later, we verified the result on one processor supporting 64-bit architecture with a large main memory(8GB) using  $p$ -adic lifting [Dixon (1982)] after explicitly computing the inverse of  $A$  mod a word size prime by Gaussian elimination and storing it as a dense matrix - see [Dumas et al. (2002b)] for detail. Now with the main idea in this paper, the exact answer of this problem can be computed in 25 minutes with a PC with Linux, 1.9GHZ Pentium processor, 1GB memory, or in 13 minutes with a better machine with 3.2GHZ Intel Xeon processors and 6GB memory. And only a few MB of memory at run time is required. This method is an application of our main idea with a custom approximate inverse of the input matrix. The general  $n \times n$  matrix with the

same pattern as  $A$  is a sparse matrices with  $O(n \log n)$  non-zero entries and is an almost row diagonally dominant matrix except in the first few rows. For the special  $20000 \times 20000$  matrix  $A$ , there are at most 29 non-zero entries in each row (column). So, if we represent the matrix as a block matrix:

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

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

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

We know all diagonal entries of  $A_{22}$  are greater than 3571 which is the 500-th prime, all off diagonal entries of  $A_{22}$  are 0 or 1, and there are at most 28 non-zero off-diagonal entries of  $A_{22}$  in each row or column, so  $\|ED^{-1}\|_{\infty} \leq \frac{3571}{28} < \frac{1}{127.5}$ . Also  $A_{12}$  and  $A_{21}$  are very sparse, and every diagonal entry of  $D$  is larger

than 3571, therefore it is reasonable to estimate that  $\|A \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I\|_{\infty}$  is

less than  $\frac{1}{64}$ , near twice of  $\frac{1}{127.5}$ , and follow up computation shows that  $\frac{1}{64}$  is

not overestimated. So we can use  $\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1}$  as an approximate inverse of  $A$ .

Note that it is a lower triangle block matrix. Thus we can quickly solve

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

We call LAPACK routines to compute the inverse of  $A_{11}$  explicitly, and store it as a dense matrix. Other parts can be stored as sparse matrices and be used as blackboxes. Here, in order to simplify the explanation and estimate the condition number of  $A_{11}$ , an explicit inverse is computed instead of just storing its factorization, and the time computing the inverse is negligible, compared to the total run time for computing the symbolic answer. Computation shows that  $A_{11}$  is well-conditioned. In the experimentation, we solve  $Ax = e_1 = (1, 0, \dots, 0)$ . With algorithm 1, We choose all scalars  $\alpha^{(i)}$  equal to 64 and add a watch dog for the residual, that is, if the norm of the residual is not smaller than the theoretical result in each iteration, it would abort with an error message. Only digits for  $x_1$  are stored and only  $x_1$  is reconstructed. This

method is asymptotically faster than previously, and requires almost a chunk of memory as small as when totally treating the matrix as a blackbox. Here is a brief summary of the three different successful approaches above.

Table 3

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

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

## 5 Future work and conclusion

Sparse numeric iterative methods have been most extensively successful on PDE problems. It remains to be seen whether fast numeric solvers can be found which succeed on the sorts of sparse matrix arising in integer linear algebra (for instance the homology matrices [Dumas et al. (2000)]). This seems an important avenue for further study.

In summary, we present a new practically fast algorithm to exactly (symbolically) solve well-conditioned dense linear systems with integer coefficients over the rational field using a hybrid of numerical methods and exact (symbolic) methods. It leads to high performance implementation in practice, and may lead to fast algorithms for general families of sparse linear systems.

## Acknowledgments

The author would like to thank B. David Saunders for his help with this paper. The paper couldn't be done without his help. Also thank Prof. Folkmar Bornemann and referees for many thoughtful suggestions.

## References

- Chen, Z., Storjohann, A., 2004. An implementation of linear system solving for integer matrices. In: Poster, ISSAC'04.
- Cormen, T., Leiserson, C., Rivest, R., Stein, C., 2001. Introduction to Algorithms, Second edition. MIT Press.
- Demmel, J. W., 1997. Applied numerical linear algebra. SIAM.
- Dixon, J. D., 1982. Exact solution of linear equations using  $p$ -adic expansion. Numer. Math., 137–141.
- Dumas, J.-G., Gautier, T., Pernet, C., 2002a. Finite field linear algebra subroutines. In: Proc. ISSAC'02. ACM Press, pp. 63 – 74.
- Dumas, J.-G., Saunders, B. D., Villard, G., 2000. Smith form via the valence : Experience with matrices from homology. In: Proc. ISSAC'00. ACM Press, pp. 95 – 105.
- Dumas, J.-G., Turner, W., Wan, Z., 2002b. Exact solution to large sparse integer linear systems. In: Poster, ECCAD 2002.
- Forsythe, G. E., Moler, C. B., 1967. Computer solution of linear algebraic systems. Prentice-Hall.
- Geddes, K., Zheng, W., December 2002. Exploiting fast hardware floating point in high precision computation. Tech. rep., School of Computer Science, University of Waterloo, CA.
- Moenck, R. T., Carter, J. H., 1979. Approximate algorithms to derive exact solutions to systems of linear equations. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. Springer-Verlag, pp. 65–73.
- Pan, V., Wang, X., 2002. Acceleration of Euclidean algorithm and extensions. In: Proc. ISSAC'02. ACM Press, pp. 207 – 213.
- Saad, Y., 2003. Iterative Methods for Sparse Linear Systems, 2nd Edition. SIAM.
- Trefethen, L. N., Bau, I., 1997. Numerical linear algebra. SIAM.
- von zur Gathen, J., Gerhard, J., 1999. Modern Computer Algebra. Cambridge University Press.
- Wiedemann, D., 1986. Solving sparse linear equations over finite fields. IEEE Transf. Inform. Theory 32, 54 – 62.