Exactly Solve Integer Linear Systems Using Numerical Methods

Zhendong Wan

Department of Computer Science, University of Delaware, Newark, DE, USA

Abstract

We present a new fast way to exactly solve non-singular linear systems with integer coefficients using numerical methods. This leads to high performace in practice and also leads to asymptoticly fast algorithms for some special family of linear systems. Our method is to find an initial approximate solution by using a fast numerical method, then amplify the approximate solution by a scalar, adjust the amplified solution and corresponding residual so that they can be stored exactly, and repeat these steps to refine the solution until high enough accuracy is achieved, and finally reconstruct the rational solution. We will expose the theoretical cost and show some experimental results.

Key words: linear system, numerical linear methods, rational solution

1 Introduction

Both symbolic linear methods and numerical linear methods can be used to solve linear systems. But they use different techniques and algorithms, and have been developed independently. Symbolic linear methods often use either p-adic lifting Dixon (1982); Moenck and Carter (1979) or the Chinese remainder algorithm or both. They all are based on solving the linear system modulo a big integer and finally reconstructing the rational solution. 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. Symbolic linear methods can deliver the accurate answer without any error, though they are often more

Email address: wan@mail.eecis.udel.edu (Zhendong Wan).

expensive in computation time than numerical linear methods, But numerical linear methods are subject to convergence problems and the limitation of floating point precision.

In this paper, we combine the two methods. We describe a new way to exactly solve non-singular integer linear systems using numerical methods. It is effective in practice 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 packages are developed using fast BLAS implementation for dense systems. Also this approach leads to asymptoticy faster algorithms for some special sparse systems. The sparse systems are discussed in section 3.2.

The motivation of this paper is the high performance of numerical packages and this simple fact: 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 we can compute the solution with very high accuracy, then we can reconstruct the rational solution.

Generally numerical linear methods are inexact when carried out on a computer: one hopes for answers accurate up to machine precision (or software floating point precision), no better. In order to achieve more accuracy than machine precision, our idea is simple, approximation, amplification and adjustment in short words. 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, repeat these steps until a desired accuracy is achieved. The approximating, amplifying, and adjusting idea enable us to compute the solution with arbitrarily high accuracy without any high precision software floating point arithmetic involved. The details are discuss 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, 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 $\log ||A||$ to denote the maximum of bit length of entries of integer matrix A.

Next section is about the rational reconstruction. It is a classsic result. Best approximation of the continued fraction enables us to reconstruct a rational number with certain constraint 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 integer and matrices are well conditioned. After that these ideas lead to an asymptoticly fastest algorithm for a special sparse family. 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 a quick reminder, a brief decription of the continiued fraction is given. For a real number r, a simple continued fraction is an expression in the form

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

where all a_i are integers. From now on, we assume that if $r \geq 0$, then $a_0 \geq 0$, all $a_i > 0$ for $i \geq 1$, and if $r < 0$, then $a_0 \leq 0$, all $a_i < 0$ for $i \geq 1$. A more convenient notation is $r = [a_0; a_1, a_2, \cdots]$ Intuitively, we can apply extended Euclid's algorithm for finding the greatest common divisor 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, $3796/1387 = 2 + 1022/1387 = 2 + 1/(1387/1022) = 2 + 1/(1 + 365/1022) = 2 + 1/(1 + 1/(1022/365)) = 2 + 1/(1 + 1/(2 + 292/365) \cdots = 2 + 1/(1 + 1/(2 + 1/(1 + 4))) = [2; 1, 2, 1, 4]$.

For a simple continued fraction for r (either finite or infinite) one defines a family of finite segments $s_k = [a_0; a_1, a_2, ..., a_k]$, each s_k being a rational number: $s_k = p_k/q_k$ with $q_k > 0$ and $\gcd(p_k, q_k) = 1$. There are properties about simple continued fraction from number theory. Following we list some, assuming $r \geq 0$. For $r < 0$, there are similar properties about them.

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

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

THEOREM 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:

(1) This is a classic result. I put it here just as a refreshing remindar.
(2) If $B = 1$, then there is either no solution or $\frac{a}{b} = \frac{\text{the nearest integer to } r}{1}$.

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 $0 < b_1, b_2 \leq B$, $\gcd(a_1, b_1) = \gcd(a_2, b_2)$, $\frac{a_1}{b_1} \neq \frac{a_2}{b_2}$. Then $|\frac{a_1}{b_1} - \frac{a_2}{b_2}| \leq |\frac{a_1}{b_1} - r| + |\frac{a_2}{b_2} - r| < \frac{1}{2Bb_1} + \frac{1}{2Bb_2} \leq \frac{1}{b_1 b_2}$, while $|\frac{a_1}{b_1} - \frac{a_2}{b_2}| = |\frac{a_1 b_2 - b_1 a_2}{b_1 \cdot b_2}| \geq \frac{1}{b_1 b_2}$. Contradiction! So there is at most one solution.

If $\frac{a}{b}$ is a solution with $0 < b \leq B$, and $\gcd(a, b) = 1$. Then we need to prove $\frac{a}{b} = \frac{p_k}{q_k}$, for some k, such that either $p_k/q_k = r$ or $q_k \leq B < q_{k+1}$. By way of contradiction, suppose $\frac{a}{b} \neq \frac{p_k}{q_k}$. If $p_k/q_k = r$, then $\frac{p_k}{q_k}$ is another rational solution. Contradiction! Now we assume $q_k \leq B < q_{k+1}$. We know by property 3, $|\frac{p_k}{q_k} - \frac{p_{k+1}}{q_{k+1}}| = \frac{1}{q_k q_{k+1}}$. Also $|\frac{a}{b} - \frac{p_k}{q_k}| \geq \frac{1}{bq_k} > \frac{1}{q_k q_{k+1}}$. So $\frac{a}{b}$ doesn't lie between $\frac{p_k}{q_k}$ and $\frac{p_{k+1}}{q_{k+1}}$. By property 4, we know r must lie between $\frac{p_k}{q_k}$ and $\frac{p_{k+1}}{q_{k+1}}$. So $|\frac{a}{b} - r| \geq \min(|\frac{a}{b} - \frac{p_k}{q_k}|, |\frac{a}{b} - \frac{p_{k+1}}{q_{k+1}}|) \geq \frac{1}{bq_{k+1}}$. Therefore $\frac{1}{2Bb} > \frac{1}{bq_{k+1}}$, $q_{k+1} > 2B$.

Thus $|\frac{p_k}{q_k} - r| \leq |\frac{p_k}{q_k} - \frac{p_{k+1}}{q_{k+1}}| = \frac{1}{q_k q_{k+1}} < \frac{1}{2q_k B}$. so $\frac{p_k}{q_k}$ is another solution. Contradiction!

If $r = \frac{n}{d}$, the (p_k, q_k) can be computed by half gcd algorithm (see (von zur Gathen and Gerhard, 1999, Algorithm 11.4)). It needs $O(M(l) \log l)$ bit operations.

3 Rational solver for integer linear systems

In the section, we present a new way to exactly compute the solution of a non-singular linear system with integer coefficients, repeatedly using a numerical method. It is well known that numerical linear methods are inexact when carried out on a computer. Iterative refinement methods (see Forsythe and

Moler (1967); Demmel (1997); Geddes and Zheng (2002)) can be used to refine the solution. The refinement Geddes and Zheng (2002) is a recent result, very effective 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. Iterative refinement methods such as this one help improve the accuracy of the answers, but the accuracy is limited to machine precision (or the software floating point precision). That is, the answers are accurate up to machine precision (or the software floating point precision), no better. Also each refinement iteration requires a computation in high precision. The cost of one software floating point operation increases rapidly with respect to the precision. It has been illustrated in (Geddes and Zheng, 2002, Figure 1). 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 α, amplify and adjust the Δx with rationals $\frac{1}{\Delta d} \cdot \Delta n$ by setting $\Delta d = \alpha$, $\Delta n = (\approx \alpha \cdot \Delta x_1, \ldots, \approx \alpha \cdot \Delta x_n)$, update answer: $n = \alpha \cdot n + \Delta n$ and $d = \Delta d \cdot d$, update residual $r = \alpha \cdot r - A \cdot \Delta n$. In each refinement iteration, the amplified and adjusted residual can be stored exactly, and its bit length can be well controlled. Thus our method can be used to achieve the arbitrary high accuracy of the answers without high precision software floating point arithmetic involved. After sufficient accuracy is achieved, the final rational solution can be reconstructed.

As in most numerical analysis, we will use the infinity-norm in the performance analysis. Specifically, for a $n \times m$ matrix $A = (a_{ij})$, we define $||A||_\infty = \max_{1 \le i \le n} \sum_{1 \le j \le m} |a_{ij}|$. For a vector b of length n, we define $||b||_\infty = \max_{1 \le j \le n} |b_j|$;

3.1 Dense integer linear system solver

We apply our main idea to the dense linear system in detail: algorithm, theorem, and remarks.

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 $Ax = b$.*

Procedure:

(1) *Compute floating point matrix A^{-1}, the inverse of A using floating point arithmetics. [A^{-1}, the inverse of A, may be computed using any backward stable numerical method. Even more, it doesn't need to be computed explicitly and can simply be stored in its factorization form, because it will be used as a blackbox.]*
(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;* [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 high performance, α is better chosen to a power of 2 since in this case any integer multiplying the α can be replaced by shifting bits. 30 is a constant number, which is better chosen to be a little smaller than the bit length of the hardware integer for high performance reason, 30 for 32-word machines. For small size systems, the approximate maybe equals to the exact anser. So that the constant 30 will make the algorithm not abort, since in that case α becomes positive infinity.]
  6.4 *if $\alpha^{(i)} < 2$, abort with error message "insufficient numerical accuracy".*
  6.5 *Exactly compute integer vector $x^{(i)} := (\approx \alpha^{(i)} \cdot \bar{x}_1^{(i)}, \ldots, (\approx \alpha^{(i)} \cdot \bar{x}_m^{(i)})$. $x^{(i)}$ is the nearst integer of $\alpha^{(i)} \cdot \bar{x}^{(i)}$ component-wise, obviously $||x^{(i)} - \alpha^{(i)} \cdot \bar{x}^{(i)}||_\infty \leq 0.5$.* [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 each time.]
(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.*

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

PROOF.On the input of A, b, if the algorithm aborts, the statement is true. Otherwise, we need to show the algorithm terminates with correct rational answer. First we show the algorithm will terminate. Let us estimate $d^{(i)} = \prod_{1\le j\le i}\alpha^{(j)} \ge \prod_{1\le j\le i} 2 \ge 2^i$, since it doesn't abort, that is $\alpha^{(i)} >= 2$. We know $\frac{2^i}{i+1}$ increases rapidly, so the loop inside the algorithm runs finitely many iterations. The algorithm will terminate. Now we prove the correctness. Let $n^{(i)} = \sum_{1\le j\le i}\frac{d^{(k)}}{d^{(j)}} \cdot x^{(j)}$, 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 $r^{(i)} = d^{(i)}(b - A\frac{1}{d^{(i)}} \cdot n^{(i)})$, 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 , we have $||A\bar{x}^{(i)} - r^{(i-1)}||_\infty \le \frac{1}{2\alpha^{(i)}} \cdot ||r^{(i-1)}||_\infty$. By the definition of $x^{(i)}$, we know $||x^{(i)} - \alpha^{(i)} \cdot \bar{x}^{(i)}||_\infty \le 0.5$. So

$$\begin{aligned}
||r^{(i)}||_\infty &= ||Ax^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}||_\infty \\
&\le ||\alpha^{(i)} \cdot A\bar{x}^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}||_\infty + ||Ax^{(i)} - \alpha^{(i)} \cdot A\bar{x}^{(i)}||_\infty \\
&\le 0.5||r^{(i-1)}||_\infty + 0.5||A||_\infty.
\end{aligned}$$

Therefore $||r^{(i)}||_\infty \le \frac{1}{2^i}||b||_\infty + ||A||_\infty$. Thus $e^{(i)} = \frac{1}{d^{(i)}}||A^{-1}r^{(i)}||_\infty \le \frac{1}{d^{(i)}}||A^{-1}||_\infty$ $(\frac{1}{2^i}||b||_\infty + \cdot||A||_\infty)$, for $i \ge 1$. Let k be the value of i when the loop stops. Let us estimate $2B\det(A)e^{(k)}$. So far, we know $2B\det(A)e^{(k)} < \frac{2}{d^{(k)}}\ ||B\det(A) \cdot A^{-1}||_\infty(2^{-k}||b||_\infty + ||A||_\infty)$. We know $\det(A)A^{-1}$ is the adjoint matrix of A. Obviously, for non-singular integer matrices A, the Hadamard bound bounds very minor of A. That is, each entry of $\det(A)A^{-1}$ is $(m-1)\times(m-1)$ minor of A. Therefore $||\det(A)A^{-1}||_\infty \le mB$. Thus $e^{(k)}2B\det(A) \le \frac{2mB^2(2^{-k}||b||_\infty + ||A||_\infty)}{d^{(k)}}$ < 1. So we have $e^{(k)} < \frac{1}{2B\det(A)}$. Since $||\frac{1}{d^{(k)} \cdot n^{(k)}} - A^{-1}b||_\infty < \frac{1}{2Bdet(A)}$, and by Cramer's rule we know $\det(A) \cdot A^{-1}b$ is an integer vector, by Theorem 1, the reconstructed rational solution must be equal to $A^{-1}b$.

Remarks:

(1) The asymptotic time complexity is comparable with the Dixon lifting algorithm Dixon (1982).
(2) The idea of (Pan and Wang, 2002, section 4) can be used to accelerate the final rational reconstruction step.
(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 residual computed in step 6.7 is as small as expected in theory.

(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 accumulated answer $\frac{1}{d^{(i)}}n^{(i)}$ as the solution, then the inifity norm of the absolute residual, $||b - A \cdot \frac{1}{d^{(i)}}n^{(i)}||_\infty$, is less than $\frac{1}{\prod_{1 \le j \le i} \alpha^{(i)}}(2^{-1}||b||_\infty + ||A||_\infty)$, 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. For the well conditioned matrices, the algorithm doesn't abort, but terminates with the correct rational solution if a backward stable numerical method is used. The entry of the amplified residual will be bound by $||b||_\infty + ||A||_\infty$. But in estimation of the cost, we need to estimate $||\bar{x}^{(i)}||_\infty$, If A is well conditioned, then $||A^{-1}||_\infty$ is small. In each iteration, $||\bar{x}^{(i)}||_\infty \approx ||A^{-1}r^{(i-1)}||_\infty \le ||A^{-1}||_\infty \cdot ||r^{(i-1)}||_\infty = O(||r^{(i-1)}||_\infty)$. So each iteration needs $O(m^2)$ floating point operations and $O^{\sim}(m^2(\log ||A|| + \log ||b||_\infty))$ 32-bit integer operations. Now we estimate the number of iterations. We know the Hadamard bound B and $\log B = O^{\sim}(m \log ||A||)$. So the number of iteration required is $O^{\sim}(m \log ||A|| + \log ||b||_\infty)$. Thus the loop costs $O^{\sim}(m^3(\log ||A||)(\log ||A|| + \log ||b||_\infty))$.

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

3.1.2 Dense linear system experimentation

The following table is the comparison of the running time of three different methods. Plain_Dixon is an implementation of Dixon lifting without calling BLAS in LinBox [1] . 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, in C, using LAPACK routines implemented in ATLAS [2] .

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

[2] 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

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
dslover	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 much 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 α is the geometric average of $\alpha^{(i)}$. For a well conditioned matrix, the $\alpha^{(i)} > 2^{32}$ in each iteration. That is, in each iteration, dsolver can get at least 32 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 Sparse integer linear system solver

The main idea is to compute floating point solutions of linear systems using iterative methods such as Lanczos method, the GMRES method, or Jacobi's method, using a few cheap floating point matrix-by-vector products. These iterative methods can successfully solve many problems from discretization of partial differential equations. But from our experiments, we know that iterative methods like the GMRES and CG and BICG don't work well for random sparse matrices. More precisely, these iterative methods don't converge in a small number of iterations for random sparse linear systems. For diagonally dominant matrices, Jacobi's method can be used to compute the floating point solution with a few iterations.

We focus on diagonally dominant matrices in the rest of this section. The

and is available at `http://math-atlas.sourceforge.net` similar idea can be used more generally, as illustrated in the following section. An $n \times m$ matrix $A = (a_{ij})$ is called row diagonally dominant, if for each $1 \leq k \leq n$, $|a_{kk}| > \sum_{j \neq k} |a_{kj}|$. We call A an β-rdd matrix, if, for each $1 \leq k \leq n$, $|a_{kk}| > \beta \sum_{j \neq k} |a_{kj}|$. For row diagonally dominant matrices, Jacobi's method is a good method to compute a floating point solution using a few matrix-by-vector productions. This approach leads an algorithm aymptotic faster than any previous one.

Algorithm 2 *Rational solver for rdd case*
Input:

- *A, an $m \times m$ β-rdd integer matrix, $\beta > 1$*
- *b, a right hand side integer vector.*

Output:

- *x, the rational solution of $Ax = b$.*

Procedure:

```
(1) Set integer $s := \lceil \frac{(\beta^2+\beta) max(|A_{ii}|, 1 \leq i \leq m)}{\beta - 1} \rceil$. [Note: $\frac{1}{\beta} + \frac{\beta-1}{\beta^2+\beta} = \frac{2}{\beta+1}$.]
(2) Compute diagonal integer matrix $P$, such that $P_{ii} = \lfloor \frac{s}{A_{ii}} \rfloor$. [P is the
    preconditioning matrix. Solve $PAx = Pb$.]
(3) Set integer $d^{(0)} := 1$. [The common denominator]
(4) Set integer vector $r^{(0)} := P \cdot b$; [The residual]
(5) Compute $B$, the Hadamard bound of $\det(A)$, which bounds the determinat
    and all $(m-1) \times (m-1)$ minors of $A$.
(6) Set $i := 0$.
(7) Set integer $\alpha := max(2, \lfloor \frac{\beta+1}{2} \rfloor)$.
(8) Set integer $l := \lceil \frac{\log(2\alpha)}{\log(\frac{\beta+1}{2})} \rceil$.
(9) Set $E := sI_m - PA$. [$Ex = sx - P \cdot Ax$. We may use $E$ as a blackbox.]
(10) Repeat the following steps until $d^{(i)} > 2mB^2(2^{-i}||Pb||_\infty + ||PA||_\infty)||P^{-1}||_\infty$
  10.1 Set $i := i + 1$;
  10.2 Compute rational vector $\bar{x}^{(i)} = \sum_{0 \leq j < l} \frac{1}{s^{j+1}} E^j r^{(i-1)}$. [Note that $s^l \cdot \bar{x}^{(i)}$
       is an integer vector.]
  10.3 Compute integer vector $x^{(i)} := (\approx \alpha \cdot \bar{x}_1^{(i)}, \ldots, \approx \alpha \cdot \bar{x}_m^{(i)})$. [ $||x^{(i)} - \alpha \cdot$
       $\bar{x}^{(i)}||_\infty \leq 0.5$.]
  10.4 Compute integer vector $r^{(i)} := \alpha \cdot r^{(i-1)} - P \cdot Ax^{(i)}$.
  10.5 Set integer $d^{(i)} := \alpha \cdot d^{(i-1)}$
(11) Set $k := i$
(12) Compute integer vector $n^{(k)} = \sum_{1 \leq i \leq k} \alpha^{k-i} \cdot x^{(i)}$.
(13) Reconstruct rational solution $x$ from $\frac{1}{d^{(k)}} \cdot n^{(k)}$ using Theorem 1 with the
     denominators bounded by $B$.
(14) return $x$.
```

THEOREM 3 *The algorithm above will correctly compute the rational solution. It requires* $O^{\sim}(\frac{m}{\log(\frac{\beta+1}{2})}(\log ||A|| + \log ||b||_\infty))$ *matrix-by-vector products of* A *and* $O^{\sim}(m^2(\log ||A|| + \log ||b||_\infty))$ *extra bit operations, and the total cost is* $O^{\sim}(\frac{mN}{\log(\frac{\beta+1}{2})}(\log ||A|| + \log ||b||_\infty)^2)$ *bit operations, where* N *is the number of non-zero entries of* A.

PROOF.The algorithm above solves $Ax = b$ by solving $P \cdot Ax = Pb$. The proof is similar to the proof of theorem 2. Since $d^{(i)} \geq 2^i$, the loop doesn't run forever. Now we prove the correctness. We need to estimate the absolute error in each iteration. $e^{(i)} = ||\frac{1}{d^{(i)}} \cdot n^{(i)} - A^{-1}b||_\infty$, where $n^{(i)} = \sum_{1 \leq j \leq i} \alpha^{i-j} \cdot x^{(j)}$ is the numerator of the accumulated solution of the first i iterations. By induction, we have $r^{(i)} = d^{(i)}(Pb - P \cdot A \frac{1}{d^{(i)}} n^{(i)})$. Therefore, $e^{(i)} = \frac{1}{d^{(i)}} ||A^{-1}P^{-1}r^{(i)}||_\infty$. We need to estimate $||r^{(i)}||_\infty$. Let $\bar{A} = PA$. We know $||r^{(i)}||_\infty = ||\alpha r^{(i-1)} - \bar{A}x^{(i)}||_\infty$ $\leq ||\alpha r^{(i-1)} - \alpha \bar{A}\bar{x}^{(i)}||_\infty + ||\bar{A}\alpha\bar{x}^{(i)} - \bar{A}x^{(i)}||_\infty$ By the choice of $x^{(i)}$, obviously $||x^{(i)} - \alpha\bar{x}^{(i)}||_\infty \leq 1$. Thus $||r^{(i)}||_\infty \leq \alpha||r^{(i-1)} - \bar{A}\bar{x}^{(i)}||_\infty + ||A||_\infty$. So now let us estimate $||\bar{A}\bar{x}^{(i)} - r^{(i-1)}||_\infty$ $||\bar{A}\bar{x}^{(i)} - r^{(i-1)}||_\infty = ||\bar{A}\sum_{0 \leq j < l} \frac{1}{s^{j+1}} E^j r^{(i-1)} i - r^{(i-1)}||_\infty \leq ||\bar{A}\sum_{0 \leq j < l i \frac{1}{s^{j+1}}} E^j - I_m||_\infty ||r^{(i-1)} i||_\infty = ||(sI_m - E)\sum_{0 \leq j < l} \frac{1}{s^{j+1}} E^j - I_m||_\infty ||r^{(i-1)}||_\infty \leq ||\frac{1}{s^l} E^l||_\infty ||r^{(i-1)}||_\infty \leq (\frac{||E||_\infty}{s})^l ||r^{(i-1)}||_\infty$. Now we need to estimate the $||E||_\infty$. Since $E = sI_m - \bar{A}$, for an $1 \leq i \leq m$, $\sum_{1 \leq j \leq m} |E_{ij}| = |E_{ii}| + \sum_{1 \leq j \leq m, j \neq i} |E_{ij}| \leq |s - P_{ii}A_{ii}| + |\frac{1}{\beta}P_{ii}A_{ii}| \leq |A_{ii}| + |\frac{1}{\beta}s| \leq |\frac{\beta-1}{\beta^2+\beta}s| + |\frac{s}{\beta}| \leq \frac{2s}{\beta+1}$ Thus, $||E||_\infty \leq \frac{2s}{\beta+1}$. So $||\bar{A}\bar{x}^{(i)} - r^{(i-1)}||_\infty \leq (\frac{2}{\beta+1})^l ||r^{(i-1)}||_\infty \leq \frac{1}{2\alpha}||r^{(i-1)}||_\infty$. So $||r^{(i)}||_\infty = ||\alpha r^{(i-1)} - \bar{A}x^{(i)}||_\infty \leq ||0.5r^{(i-1)}||_\infty + 0.5||PA||_\infty$. Therefore $||r^{(i)}||_\infty \leq 2^{-i}||Pb||_\infty + ||PA||_\infty$. Thus $e^{(i)} \leq ||A^{-1}||_\infty ||P^{-1}||_\infty \frac{1}{d^{(i)}}(2^{-1}||Pb||_\infty + ||PA||_\infty)$, for $i >= 1$. Let us estimate e^k, k is the final iteration number. Now we know $e^{(k)} 2B\det(A) \leq \frac{2}{d^{(k)}} \cdot B||\det(A)A^{-k}||_\infty ||P^{-1}||_\infty (2^{-k}||Pb||_\infty + ||PA||_\infty)$ $\leq \frac{2mB^2}{d^{(k)}} \cdot ||P^{-1}||_\infty (2^{-k}||Pb||_\infty + ||PA||_\infty) < 1$. Thus $||A^{-1}b - \frac{1}{d^{(k)}} \cdot n^{(k)}||_\infty = e^{(k)} < \frac{1}{2B\det(A)}$. and by Cramer's rule we know $\det(A) \cdot A^{-1}b$ is an integer vector, by Theorem 1, the reconstructed rational solution must be equal to $A^{-1}b$.

Only the cost analysis is left. It requires at most $O^{\sim}(m \log ||A|| + \log ||b||_\infty)$ iterations for an input A and b, as in the dense case. In each iteration, we need to compute $\bar{x}^{(i)}$, $x^{(i)}$, $r^{(i)}$ and $d^{(i)}$. Since $s^l \cdot \bar{x}^{(i)}$ is a integer vector, it requires l calls of matrix-by-vector products of E, and extra integer addition to compute it. Each matrix-by-vector product of E requires only one matrix-by-vector product of A with other cheaper extra work. Thus it requires $O^{\sim}(\frac{m}{\log(\frac{\beta+1}{2})}(\log ||A|| + \log ||b||_\infty))$ matrix-by-vector products of A and $O^{\sim}(m^2(\log ||A|| + \log ||b||_\infty))$ extra bit operations. For updating $d^{(i)}$ and $r^{(i)}$ in each iteration, it requires $O^{\sim}(m(\log ||P||A + \log ||pb||))$ bit operations. The computation of $n^{(k)}$ and the final rational reconstruction requires $O^{\sim}(m^2(\log ||A|| + \log ||b||_\infty))$ bit operations. So the total cost is $O^{\sim}(\frac{mN}{\log(\frac{\beta+1}{2})}(\log ||A|| + \log ||b||_\infty)^2)$ bit operations.

The remarks in the dense case can apply here also.

3.2.1 Sparse linear system experimentation

We generate some random sparse and row diagonally dominant matrices, in which all diagonal entries are $100,000$, and each row has 10 extra randomly positioned non-zero entries which are randomly and independently chosen from $[80, 100]$. Choose $\beta = 100$. Again dsolver uses the implementation of Algorithm 1, while ssolver uses the implementation of Algorithm 2.

order	200	600	1000	1400	1800	2200	2800
dslover	0.14	4.04	18.30	49.37	105.21	190.93	312.94
ssolver	0.19	4.08	17.98	46.35	98.85	163.07	275.20

These tests are run sequentially in a server with 3.2GHZ Intel Xeon processors and 6GB memory. Clearly, algorithm 2 can take advantage of the sparsity, saving memory and even computing time for reasonably large matrix order.

4 Quickly and exactly solve a challenge problem

I will demonstrate that our new method can be extremely much faster than previous methods. In year 2002, Prof. L. N. Trefethen posted "The SIAM 100-Dollar, 100-Digit Challenge".[3] Here is problem 7:

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

Though only the first 10 decimal digits is required for the original problem, an exact solution was computed by Jean-Guillaume Dumas of LMC-IMAG in Grenoble, France two years ago. It is a fraction with exactly 97,389 digits each for relatively prime numerator and denominator. He ran 182 processors for four days using LinBox software; the mathematics involved blackbox modular techniques, Wiedemann's algorithm Wiedemann (1986), and the Chinese remainder theorem. A couple of months later, we verified the result on one processor supporting 64-bit architecture with a large main memory(8GB) using Dixon lifting Dixon (1982) after explicitly computing the inverse of A over a word size prime by Gaussian elimination and storing it as a dense matrix. See Dumas et al. (2002b) for details. Now with the main idea in this paper, The exact answer of this problem can be computed in 25 minutes in a cheap

[3] `http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/hundred.html.`

PC with Linux, 1.9GHZ Pentium processor, 1GB memory, or in 13 minutes in a better machine with 3.2GHZ Intel Xeon processors and 6GB memory, and a few MB memory at run time is required. This method is a mixture of algorithm 1 and algorithm 2, The general $n \times n$ matrix with same pattern of A is a sparse matrices with $O^{\sim}(n \times \log_2(n))$ non-zero entries and is an almost row diagonally dominant matrix except the first few rows. For the special 2000×2000 matrix A, there are at most 29 non-zero entries in each row. It is known that the 500-th prime is 3571, and $\frac{3571}{28} \approx 127.5$. So, if we represent the matrix as a block matrix, $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where $A_{11}, A_{12}, A_{21}, A_{22}$ are 500×500, $500 \times 19,500$, $19,500 \times 500$, and $19,500 \times 19,500$ matrices, respectively. Then A_{22} is a 127.5-rdd matrix. Let D be the diagonal part of A_{22}, E be the rest. Let us estimate $||A \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I||_\infty = ||(\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix} + \begin{bmatrix} 0 & A_{12} \\ 0 & E \end{bmatrix}) \begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1} - I||_\infty$ $= || \begin{bmatrix} -A_{12}D^{-1}A_{21}A_{11}^{-1} & A_{12}D^{-1} \\ -ED^{-1}A_{21}A_{11}^{-1} & ED^{-1} \end{bmatrix} ||_\infty$. We estimate that it is less than $\frac{1}{64}$. So we can use $\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix}^{-1}$ to approximate the inverse of A. Note that it is a lower triangle block matrix. Thus we can solve $\begin{bmatrix} A_{11} & 0 \\ A_{21} & D \end{bmatrix} x = y$ quickly, 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. In the experimentation, we solve $Ax = e_1 = (1, 0, \cdots, 0)$. 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 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 asympotically faster than previously, and requires almost as a small chunk of memory as totally treating the matrix as a blackbox. Here is a brief summary of the three different successful approaches above.

Methods	Complexity	Memory	Run time
Quotient of two determinants Wiedemann's algorithm Chinese remainder theorem	$O^{\sim}(n^4)$	a few MB	Four days in parallel using 182 processors, 96 Intel 735 MHZ PIII, 6 1GZ 20 $4 \times$ 250MHZ sun ultra-450
Solve $Ax = e_1 = (1, 0, \cdot, 0)$ by plain Dixon lifting for the dense case Rational reconstruction	$O^{\sim}(n^3)$	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	$O^{\sim}(n^2)$	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 Conclusion

We present a new fast algorithm to solve well conditioned linear systems with integer coefficients over the rational field using numerical methods. It leads to high performance implemenation in practice, and may lead to fast algorithms for more general families of sparse linear systems.

Acknowledgement

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 for pointing one error and correcting it.

References

Chen, Z., Storjohann, A., 2004. An implementation of linear system solving for integer matrices. In: Poster, ISSAC'04.

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., Gautier, T., Pernet, C., 2002a. Finite field linear algebra subroutines. In: Proc. ISSAC'02. ACM Press, pp. 63 – 74.

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 Symposiumon 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.

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 IT-32, 54–62.