8 Integer matrix determinant and nonsingular rational system solving

Consider the problem of solving linear systems over the rational numbers. In this domain we face the added complexity of coefficient growth. Given nonsingular $A \in \mathbb{Z}^{n\times n}$ and $w \in \mathbb{Z}^{n\times 1}$, we wish to find $v \in \mathbb{Q}^{n\times 1}$ such that $Av = w$, or in other words, compute $A^{-1}w$. Note that v is a vector of rational numbers, and not just integers, which adds to the complexity of the problem. Here's an example of a nonsingular linear system solution:

$$\overset{A}{\begin{bmatrix} -81 & -98 & -76 & -4 & 29 \\ -38 & -77 & -72 & 27 & 44 \\ -18 & 57 & -2 & 8 & 92 \\ 87 & 27 & -32 & 69 & -31 \\ 33 & -93 & -74 & 99 & 67 \end{bmatrix}} v = \overset{w}{\begin{bmatrix} -16 \\ -9 \\ -50 \\ -22 \\ 45 \end{bmatrix}} \text{ has solution } v = \begin{bmatrix} \frac{1615537}{4562102} \\ -\frac{10907346}{11405255} \\ \frac{25212277}{22810510} \\ \frac{2035412}{11405255} \\ \frac{2893133}{22810510} \end{bmatrix}.$$

Of course, we could just do an LU decomposition over $\mathbb{Q}$ and solve the system directly. The problem is that the numbers can get very large. The main tool to see this is Hadamard's (1893) bound on the size of the determinant of A. This essentially says that the absolute value of the determinant is less than the product of the euclidean length of its rows (or columns). An easy consequence is that $\det A \le n^{n/2}||A||^n$, where $||A||$ is the maximum magnitude of entries in A.

To bound the size of a solution to a system of (integer) linear equations $Av = w$, we employ Cramer's rule, which says that if $Av = w$, and $v = (v_1, \ldots, v_n)^t \in \mathbb{Q}^{n\times 1}$, then $v_i = \det A_i / \det A$, where A_i is A with the ith column replaced by w. How does this help?

(1) The numerator and denominator of entries in $v \in \mathbb{Q}^{n\times 1}$ all have absolute value less than $n^{n/2}||A||^{n-1}||w||$.

(2) All denominators in the entries of v are divisors of $\det A$. In other words, $(\det A)v \in \mathbb{Z}^{n\times 1}$.

8.1 Finding the determinant with Chinese remaindering

A classic example of using the Chinese remainder algorithm is to compute the determinant of an integer matrix. To take into account the *size* of the entries involved, we count *word operations* rather than simply operations in $\mathbb{Z}$. We use the standard algorithms for integer arithmetic, which allow us to multiply two integers of length k words using $O(k^2)$ word operations.

The idea is to compute $\det A \bmod p_i$ for a collection of small primes $p_1, \ldots, p_k$. First, note that $(\det A) \bmod p = \det(A \bmod p)$. This follows since $\det A$ is a polynomial in the entries of A (see text, Sections 5.5 and 25.4).

The first task in employing Chinese Remaindering is to establish a bound β on the magnitude of the output integers. Let $A \in \mathbb{Z}^{n\times n}$, and recall that $||A||$ denotes the maximum magnitude of

all entries. Then Hadamard's bound gives that $|\det A| \leq \beta := n^{n/2}||A||^n$. Let $p_1, \ldots, p_k \in \mathbb{Z}_{>0}$ be primes such that

$$M := p_1 p_2 \cdots p_k > 2\beta.$$

We know that $-M/2 < \det A < M/2$, so if we know $\det A \bmod M$, we can recover $\det A \in \mathbb{Z}$. This requires that we use the "symmetric" representation of an integer modulo M. For example, if M is odd, then the numbers modulo M are reduced in the range $[-(M-1)/2, (M-1)/2]$, which allows us to represent positive and negative numbers. In Maple we would use `mods` instead of the default `modp`. Look it up!

The algorithm is as follows:

Algorithm: `ModularDeterminant`
Input: ‣ $A \in \mathbb{Z}^{n\times n}$
Output: ‣ $\det A \in \mathbb{Z}$
(1) Let $\beta := n^{n/2}||A||^n$
(2) Find small primes $p_1, \ldots, p_k \in \mathbb{Z}_{>0}$ such that $M := p_1 p_2 \cdots p_k > 2\beta$
(3) For $i = 1$ to k do
(4) Compute $A_i = A \bmod p_i$
(5) Compute $d_i = \det A_{p_i}$ over $\mathbb{Z}_{p_i}$
End For
(6) Using the Chinese Remainder Algorithm compute $d \in \mathbb{Z}$ in the symmetric range modulo M, with $d \equiv d_{p_i} \bmod p_i$ for $1 \leq i \leq k$

Finding primes in Step 2 is straightforward (though we will not address it completely here). We could simply use $2, 3, 5, \ldots$. However, it is usually more beneficial to actually have all our primes the same length, say ℓ-bits, i.e., with $2^\ell < p_i < 2^\ell$. In general, if we require the product of our primes to be larger than 2β, some results from number theory ensure us that we can choose $\ell = 6 + \ln\ln\beta$. For the choice of β in the algorithm above we have $\log\beta \leq n(\log n + \log||A||)$. Thus, in the above algorithm, the smallest choice for ℓ is

$$\ell \in O(\log n + \log\log||A||) \text{ and } k \in O(n(\log n + \log||A||)/\ell).$$

To simplify our analysis somewhat we will choose ℓ somewhat larger, with

$$\ell \in \Theta(\log n + \log||A||) \text{ and } k \in \Theta(n). \tag{1}$$

In practice, $\ell = 32$ will be more than sufficient for any reasonable problem, in which case $\ell = O(1)$ and $k = \Theta(\log M)$. However, to be completely correct in our analysis we will use the bounds given in (1).

The cost of the above algorithm is now easily analyzed. Computing A_i in Step 4 can be done with $O(n^2(\log n + \log||A||))$ word operations. Computing d_i can be done with $O(n^3(\log n + \log||A||)^2)$ word operations, since every operation in $\mathbb{Z}_p$ can be done with $O(\ell^2)$ word operations. Since $k = O(n)$, the cost without Step 6 is $O(n^4(\log n + \log||A||)^2)$ word operations. But the Chinese remainder algorithm is done using standard algorithms in time $O(n^2(\log n + \log||A||))^2)$, so the total cost is $O(n^4(\log n + \log||A||)^2)$ word operations, or about quartic in the input dimension.

8.2 Solving nonsingular systems with Chinese remaindering

We can use the same approach as above for computing determinants to solve a nonsingular linear systems $Av = w$ for $A \in \mathbb{Z}^{n\times n}$, and $w = \mathbb{Z}^{n\times 1}$. We will actually compute $d = \det A$ and $u \in \mathbb{Z}^{n\times 1}$ such that $Au = dw$ at the same time. The solution to $Av = w$ is then $(1/d)\cdot u$.

Choose primes $p_1, p_2, \ldots, p_k$ such that $M := \prod p_i > 2\beta := 2n^{n/2}||A||^{n-1}\max(||A||, ||w||)$. By Hadamard's bound, d and each entry of u has absolute value less than β. We will choose our primes to have ℓ bits, where $\ell \in \Theta(\log n + \log||A|| + (\log||w||)/n)$, in which case $k = O(n)$.

For $1 \le i \le k$ compute

$$\begin{aligned} A^{(i)} &= A \bmod p_i, \\ d^{(i)} &= \det A \bmod p_i, \\ u^{(i)} &= d^{(i)}A^{-1}w \bmod p_i. \end{aligned}$$

Now use the Chinese remainder algorithm to recover d in $\mathbb{Z}$, and $u \in \mathbb{Z}^{n\times 1}$.

The cost of computing all the $d^{(i)}$ and $u^{(i)}$ is $O(k \times n^3 \times \ell^2)$ word operations. This simplifies to $O(n^4(\log n + \log||A||)^2 + n^2(\log||w||)^2)$ word operations. This bounds the cost of the modular reductions and the Chinese remaindering as well.

8.3 Faster nonsingular solving using p-adic lifting

Next we will consider a faster method which requires $O(n^3(\log n + \log\beta)^2)$ word operations, where β is a bound for the magnitude of entries in A and w. This should be fairly surprising. First we require a bit of information about "p-adic expansions" and the conversion between different representations of raional numbers.

8.3.1 p-adic expansions and radix conversion

Let $p \in \mathbb{Z}_{>0}$ (not necessarily prime for now). Then every element of

$$S = \{a/b : a, b \in \mathbb{Z}, \gcd(b, p) = 1\}$$

has a unique (possibly infinite) p-adic expansion:

$$a/b = c_0, c_1, c_2, c_3, c_4, \cdots$$

such that $c_0 + c_1 p + .. + c_{i-1}p^{i-1} \equiv a/b \bmod p^i$ for all $i > 0$. For example

$$\begin{aligned} 17/21 \bmod 10 &= 7 \\ 17/21 \bmod 100 &= 77 \\ 17/21 \bmod 1000 &= 477 \\ 17/21 \bmod 10000 &= 477 \\ 17/21 \bmod 100000 &= 90477 \end{aligned}$$

This works much as a Taylor expansion of a rational function works. For example, we can write

$$\frac{x}{2x+1} = x - 2x^2 + 4x^3 - 8x^4 + 16x^5 + \cdots$$

In class we saw a divide and conquer algorithm for "radix conversion" that could compute the p-adic expansion of an integer given in usual 2^{64}-ary representation, and vice versa, in time $O(\mathsf{B}(n))$ word operations, n the word length of the integer. See the text, Section 9.2, for details. Recall that B is our function for bounding the cost of some algorithms, like the fast EEA, that reduce to fast integer multiplication and have running time $O(\mathsf{M}(n)\log n)$ word operations if pseudo-linear integer arithmetic is used.

8.3.2 Rational Number Reconstruction

The algorithm we describe for solving linear systems of equations will actually solve the system modulo p^k, for sufficiently large k. An important question is how do we go from this modular representation to a standard (fractional) representation of the answer in $\mathbb{Q}$?

Suppose there is a rational number n/d (for relatively prime $n, d \in \mathbb{Z}$), which we only know modulo m. That is, we only know $u \in \mathbb{Z}$ such that $u \equiv n/d \bmod m$. We assume that $\gcd(d,m) = 1$. How do we find n and d from u and m?

First, note that $du \equiv n \bmod m$, or in other words $du + qm = n$ for some $q \in \mathbb{Z}$. This looks a little like the Euclidean algorithm. In fact, consider the Euclidean algorithm on u and m: set $r_0 = m$, $r_1 = u$, $s_0 = 0$, and $s_1 = 1$, and then for $i = 1, 2, 3, \ldots$ until $i = k$ where $r_{k+1} = 0$ compute

$$\begin{aligned} q_i &= r_{i-1} \text{ quo } r_i \\ r_{i+1} &= r_{i-1} - q_i r_i \\ s_{i+1} &= s_{i-1} - q_i s_i \end{aligned}$$

At each step we have $s_i u + t_i m = r_i$ or $s_i u \equiv r_i \bmod m$ or $u \equiv r_i/s_i \bmod m$. In other words, at each step of the Euclidean algorithm we get r_i, $s_i \in \mathbb{Z}$ such that $u \equiv r_i/s_i \bmod m$.

Now suppose we know that $|n| < N$ and $d < m/N$ for some bound $N \in \mathbb{Z}_{>0}$. Then let i be such that $r_i < N < r_{i-1}$. Then $u \equiv r_i/s_i \bmod m$, and this is essentially unique. This can be computed with the same cost as computing the extended Euclidean algorithm on d and m, that is, in $\mathsf{B}(\log m)$ word operations. Please see the text, Section 5.10, for details.

8.3.3 Dixon's algorithm

We next present a method for solving $Av = w$, $A \in \mathbb{Z}^{n\times n}$ and $w \in \mathbb{Z}^{n\times 1}$, for $v \in \mathbb{Q}^{n\times 1}$. It will be faster in asymptotic complexity sense and a practical sense. The algorithm is known as Dixon's algorithm (from a paper by Dixon in 1985). It is also closely related to so-called iterative refinement in numerical analysis, and actually to the linearly-convergent Newton iteration you developed in Assignment 2.

Let p be a prime (or small prime power) such that $\gcd(p, \det A) = 1$ and $\log p \in \Theta(\log n + \log \beta)$. Let $\mathbb{Z}_p = \{0, \ldots, p-1\}$. For $i = 0, 1, 2, ...$, suppose $v_i \in \mathbb{Z}_p^{n \times 1}$ is such that

$$A^{-1} w = v_0 + p v_1 + p^2 v_2 + p^3 v_3 + \cdots \tag{2}$$

The left hand side of (2) is an $n \times 1$ vector of rational numbers with denominators relatively prime to p. The right hand side of (2) is the $n \times 1$ vector of p-adic expansions of these rational numbers. We know this exists, because we know that all the denominators in v are factors of $\det A$, and we know that $\det A \not\equiv 0 \bmod p$.

Multiply both sides of (2) by A, so

$$w = A(v_0 + p v_1 + p^2 v_2 + p^3 v_3 + \cdots).$$

We can now take this modulo p^k, so

$$w \equiv A(v_0 + p v_1 + p^2 v_2 + \cdots p^{k-1} v_{k-1}) \bmod p^k.$$

Subtract the right hand side from the left hand side:

$$w - A(v_0 + p v_1 + p^2 v_2 + \cdots + p^{k-1} v_{k-1}) \equiv 0 \bmod p^k.$$

The left hand side of the above equation is thus divisible by p^k. Define

$$\mathrm{Res}(A, w, p^k) = \frac{w - A(v_0 + p v_1 + \cdots + p^{k-1} v_{k-1})}{p^k},$$

Now compute

$$v_k \equiv A^{-1} \mathrm{Res}(A, w, p^k) \bmod p,$$

so $A v_i \equiv \mathrm{Res}(A, w, p^k) \bmod p$ and equivalently

$$p^k A v_i \equiv p^k \mathrm{Res}(A, w, p^k) \bmod p^{k+1}.$$

Thus, with a little bit of manipulation, we find

$$A\left(v_0 + p v_1 + \cdots + p^{k-1} v_{k-1} + p^k v_k\right) \equiv w \bmod p^{k+1},$$

and with the solution to a single linear system modulo p, we have added a new p-adic digit to each entry in the solution vector. Moreover, note that the system we solve is always using $A \bmod p$. I.e., we are evaluating $A^{-1} \bmod p$ once to compute each set of p-adic digits.

Algorithm: `DixonSolverPadic`
Input: ‣ $A \in \mathbb{Z}^{n \times n}$ and $w \in \mathbb{Z}^{n \times 1}$, and $\ell \in \mathbb{Z}_{>0}$
Output: ‣ $v \equiv A^{-1} w \bmod p^\ell$
(1) Compute $A \bmod p$
(2) Compute $\overline{A} = A^{-1} \bmod p$
(3) $r \leftarrow w$
(4) For i from 0 to $\ell - 1$ do
(5) $\quad v_i \leftarrow \overline{A} r \bmod p$
(6) $\quad r \leftarrow (r - A v_i)/p$
End For
(7) Return $v \leftarrow v_0 + v_1 p + v_2 p^2 + \cdots v_{\ell-1} p^{\ell-1}$

Note that we compute $\overline{A} = A^{-1} \bmod p$ *once*, and then use it at every iteration of the loop. This is because computing it costs $O(n^3)$ operations modulo p, whereas to compute $\overline{A}r$ for any $r \in \mathbb{Z}_p^{n\times 1}$ costs only $O(n^2)$ operations. What is the overall cost of this algorithm? Assume that all entries in A and w have absolute value less than β. In the following analysis we also assume that we have chosen our prime p to satisfy $\log p \in \Theta(\log n + \log \beta)$.

- Step (1): Compute $A \bmod p$ one time: $O(n^2(\log n + \log \beta)^2)$
- Step (2): Compute $\overline{A} = A^{-1} \bmod p$ one time $O(n^3(\log n + \log \beta)^2)$
- Step (5): Compute $\overline{A}r \bmod p$, and do it ℓ times: $O(\ell n^2(\log n + \log \beta)^2)$
- Step (6): Compute Av_i directly in $\mathbb{Z}$ with $O(n^2(\log n + \log \beta)^2)$ word operations, and r can be computed with this cost as well.
- Step (7): Using radix conversion, compute in $\mathbb{Z}$ with $O(n(\ell(\log n + \log \beta))^2)$ operations.

The total cost of the algorithm is thus $O((n^3 + n\ell^2 + \ell n^2)(\log n + \log \beta)^2)$ machine operations. We can then use `DixonSolverPadic` and rational number reconstruction (see §8.3.2) to convert the solution to $\mathbb{Q}^{n\times 1}$. For this we must choose an ℓ of sufficient size to guarantee the solution can be reconstructed. For this we again use the Hadamard bound. The entries $v = A^{-1}w$ all have numerator and denominator of absolute value less than $n^{n/2}\beta^n$. Thus, if we assume that $p^\ell > 2(n^{n/2}\beta^n)^2 = 2n^n\beta^{2n}$ we can recover all numerators and denominators from their values modulo p^ℓ. Since we have chosen p such that $\log p \in \Theta(\log n + \log \beta)$, we will have $\ell \in \Theta(n)$. Doing rational ecovery on numbers of this size (using the standard Euclidean algorithm) has cost $O((n(\log n + \log \beta))^2)$, and we have n such numbers in the solution vector. Thus the total cost to find $v = A^{-1}w$ is $O(n^3(\log n + \log \beta)^2)$ machine operations. Note that this is approximately n times faster than using the Chinese remainder method of §8.2!