An introspective algorithm for the integer determinant

Jean-Guillaume Dumas, Anna Urbanska

An introspective algorithm for the integer determinant

Jean-Guillaume Dumas Anna Urbańska

Abstract

We present an algorithm computing the determinant of an integer matrix A. The algorithm is *introspective* in the sense that it uses several distinct algorithms that run in a concurrent manner. During the course of the algorithm partial results coming from distinct methods can be combined. Then, depending on the current running time of each method, the algorithm can emphasize a particular variant. With the use of very fast modular routines for linear algebra, our implementation is an order of magnitude faster than other existing implementations. Moreover, we prove that the expected complexity of our algorithm is only $O\left(n^3(\log(n)+\log(\|A\|))^2\log(n)\right)$ *bit operations*, where $\|A\|$ is the largest entry in absolute value of the matrix.

1 Introduction

One has many alternatives to compute the determinant of an integer matrix. Over a field, the computation of the determinant is tied to that of matrix multiplication via block recursive matrix factorizations [12]. On the one hand, over the integers, a naïve approach would induce a coefficient growth that would render the algorithm not even polynomial. On the other hand, over finite fields, one can nowadays reach the speed of numerical routines [9]. The classical approach is thus to reduce the computation modulo some primes of constant size and to recover the integer determinant from the modular computations. For this, at least two variants are possible: Chinese remaindering and p-adic lifting. The first variant requires either a good *a priori* bound on the size of the determinant or an early termination probabilistic argument [10, §4.2]. It thus achieves an *output dependant* bit complexity of $O\left(n^\omega \log(|det(A)|)\right)$ where ω is the exponent of matrix multiplication (3 for the classical algorithm, and 2.375477 for the Coppersmith-Winograd method). Of course, with the coefficient growth, the determinant size can be as large as $\Omega(n\log(n))$ (Hadamard's bound) thus giving a large worst case complexity.

Now the second variant uses system solving and p-adic lifting [4] to get an approximation of this determinant with a $O\left(n^3(\log(n)+\log(\|A\|))^2\right)$ bit complexity [16]. Indeed, every integer matrix is unimodularly equivalent to a diagonal matrix $S = diag\{s_1,\ldots,s_n\}$ with $s_i|s_{i+1}$. This means that there exist integer matrices U,V with $\det U, \det V = \pm 1$, such that $A = USV$. The s_i are called the invariant factors of A. Then, solving a system with a random right hand side will reveal s_n as the common denominator of the solution vector entries with high probability.

The idea of [1] is thus to combine both approaches, i.e. to approximate the determinant by p-adic lifting and recover only the remaining part ($\det(A)/s_n$) via Chinese remaindering. Then G. Villard remarked that at most $O(\sqrt{n})$ invariant factors can be distinct and that, in general, only the last $O(\log(n))$ of those are nontrivial [11]. This remark, together with a preconditioned p-adic solving computing the i-th invariant factor enable them to produce a $O^\sim(n^{2+\omega/2})$ worst case algorithm, where $O^\sim$ hides some logarithmic factors, and an algorithm with an expected $O(n^3(\log(n) + \log(\|A\|))^2 \log^2(n))$ complexity. Note that the actual best worst case complexity algorithm is $O^\sim(n^{2.697263} \log(\|A\|))$, which is $O^\sim(n^{3.2} \log(\|A\|))$ without fast matrix multiplication, by [14]. Unfortunately, these last two worst case complexity algorithms, though asymptotically better, are not the fastest for the generic case or for the actual matrix sizes. The best expected complexity algorithm is a Las Vegas algorithm of Storjohann [18] which uses an expected number of $O^\sim(n^\omega \log \|A\|)$ bit operations. In section 5 we compare the performance of this algorithm to ours, based on experimental results of [19].
In this paper, we propose a new way to extend the idea of [17, 20] to get the last consecutive invariant factors with high probability in section 3.2. Then we combine this with the scheme of [1]. This combination, is made in an adaptive way. This means that the algorithm will choose the adequate variant at run-time, depending on discovered properties of its input. More precisely, in section 4, we propose an algorithm which uses timings of its first part to choose the best termination. This particular kind of adaptation was somewhat introduced in [15] as introspective ; we use here the more specific definition of [3]. This enables us to prove in section 4.1 an expected complexity of $O\left(n^3(\log(n) + \log(\|A\|))^2 \log(n)\right)$ bit operations, gaining a $\log(n)$ factor and improving the constants from [11]. Moreover, we are able to detect the worst cases during the course of the algorithm thus enabling us to switch to the asymptotically fastest method. In general this last switch is not required and we show in section 5 that when used with the very fast modular routines of [7, 9] and the LinBox library [8], our algorithm can be an order of magnitude faster than other existing implementations.

2 Base Algorithms and Procedures

In this section we present the procedures in more detail and describe their probabilistic behavior. We start by a brief description of the properties of the Chinese Remaindering loop (CRA) with early termination (ET) (see [5]), then proceed with the *LargestInvariantFactor* algorithm to compute s_n (see [1, 11, 17]). We end the section with a summary of ideas of Abbott *et al.* [1], Eberly *et al.* and Saunders *et al.* [17].

2.1 Output dependant Chinese Remaindering Loop (CRA)

CRA is a procedure based on the Chinese remainder theorem. Determinants are computed modulo several primes p_i. Then the determinant is reconstructed modulo $p_0 \cdots p_{n-1}$ in the symmetric range via the Chinese reconstruction. The integer value of the determinant is thus computed as soon as the product of the p_i exceeds $2|\det(A)|$. We know that the product is big enough if it exceeds some upper bound on this value or, probabilistically,

if the reconstructed value remains identical for several successive additions of modular determinants. The principle of early termination (ET) is thus to stop the reconstruction before reaching the upper bound, as soon as the determinant remains the same for several steps [5].
Algorithm 2.1 is an outline of a procedure to compute the determinant using CRA loops with early termination, correctly with probability $1-\epsilon$. We start with a lemma.

Lemma 2.1. *Suppose that primes p_i greater than $l \geqslant 4$ are randomly sampled form a set P, and let r_n be the value of the determinant modulo $p_0 \cdots p_n$ computed in the symmetric range. Then*

(i) $r_n = det(A)$, *if* $n \geqslant N = \begin{cases} \lceil \log_l(|\det(A)|) \rceil & \text{if } \det(A) \neq 0 \\ 0 & \text{whenever } \det(A) = 0 \end{cases}$;

(ii) if $r_n \neq \det(A)$ *then there are at most* $R = \lceil \log_l(\frac{|\det(A) - r_n|}{p_0 \cdots p_n}) \rceil$ *primes* p_{n+1} *such that* $r_n = \det(A)$ *mod* $p_0 \cdots p_n p_{n+1}$;

(iii) if $r_n = r_{n+1} = \cdots = r_{n+k}$ *then* $Prob(r_n \neq \det(A)) < \epsilon$ *if only* $k \geqslant \lceil \log(1/\epsilon) / \log(\frac{P'}{\log_l(H)}) \rceil$, *where* $P' = |P| - \lceil \log(H)/\log(l) \rceil$ *and* H *is an upper bound for the determinant (e.g.* H *can be the Hadamard's bound:* $|det(A)| \leqslant (\sqrt{n}\|A\|)^n$*).*

Proof. For (i), notice that $-\lfloor \frac{p_0 \cdots p_n}{2} \rfloor \leqslant r_n < \lceil \frac{p_0 \cdots p_n}{2} \rceil$. Then $r_n = \det(A)$ as soon as $\lfloor \frac{p_0 \cdots p_n}{2} \rfloor \geqslant |\det(A)|$. With l being the lower bound for p_i this reduces to $n \geqslant \lceil \log_l |\det(A)| \rceil$ in the case when $\det(A) \neq 0$.
For (ii), we observe that $\det(A) = r_n + K p_0 \ldots p_n$ and it suffices to estimate the number of primes greater or equal l that divide K.
For (iii) we notice that k primes dividing K can be chosen with probability $\binom{R}{k} / \binom{|P| - n + 1}{k}$, which can be bounded by $(\frac{R}{P'})^k$ Since $R \leqslant \lceil \log_l(\frac{2H}{2}) \rceil$ we get the result. □
To compute the modular determinant in algorithm 2.1 we use the LU factorization and we refer to it as LU iteration. Early termination is particularly useful in the case when the computed determinant is much smaller than the *a priori* bound. The running time of this procedure is output dependant.

2.2 Largest Invariant Factor

A method to compute s_n for integer matrices was first stated by V. Pan [16] and later in the form of the *LargestInvariantFactor* procedure (LIF) in [1, 11, 5, 17]. The idea is to obtain a divisor of s_n by computing a rational solution of the linear systems $Ax = b$. If b is chosen at random from a sufficiently large set, then the computed divisor can be as close as possible to s_n with high probability. Indeed, with $A = USV$, we can equivalently solve $SVx = U^{-1}b$ for $y = Vx$, and then solve for x. As U and V are unimodular, the least common multiple of the denominators of x and y, d(x) and d(y) satisfies $d(x) = d(y)|s_n$. Thus, solving $Ax = b$ via p-adic lifting [4], enables us to get s_n with high probability at the cost of $O(n^3(\log(n) + \log(\|A\|))^2)$ independently of the size of s_n.

Algorithm 2.1 Early Terminated CRA

Require: An integer matrix A.
Require: $0 < \epsilon < 1$.
Require: A set P of random primes greater than l.
Ensure: The integer determinant of A, correct with probability at least $1 - \epsilon$.

1: $H = (\sqrt{n}\|A\|)^n$; $P' = |P| - \lceil \log(H)/\log(l) \rceil$; $i = 0$; // Hadamard's bound
2: **repeat**
3: Get a prime p_i from the set P; $P = P - \{p_i\}$
4: Compute $\det(A) \bmod p_i$; //via LU factorization of A modulo p_i.
5: Reconstruct r_i, the determinant modulo $p_0 \cdots p_i$; // by Chinese remaindering
6: $k = \max\{t : r_{i-t} = \cdots = r_i\}$; $R = \lceil \log_l \frac{H+|r_i|}{p_0 p_1 \ldots p_{i-k}} \rceil$
7: i++;
8: **until** $\frac{R(R-1)\ldots(R-k+1)}{(|P|-n)(|P|-n-1)\ldots(|P|-n-k+1)} < \epsilon$ or $\prod p_i > 2H + 1$

The algorithm takes as input parameters β and r which are used to control the probability of correctness. r is the number of successive solvings and β is the size of the set from which the values of a random vector b are chosen. With each system solving, the output $\tilde{s}_n$ of the algorithm is updated as the lcm of the current solution denominator $d(x)$ and the result obtained so far.
The following theorem characterizes the probabilistic behavior of the LIF procedure.

Theorem 2.2. *Let A be a $n \times n$ matrix, H its Hadamard's bound, r and β be defined as above. Then the output $\tilde{s}_n$ of Algorithm* LargestInvariantFactor *of [1] is characterized by the following properties.*

i) Let $r = 1$, p be a prime, $l \geqslant 1$, then $P(p^l | \frac{s_n(A)}{\tilde{s}_n}) \leqslant \frac{1}{\beta} \lceil \frac{\beta}{p^l} \rceil$;

ii) if $r = 2$, $\beta = \lceil (n+1)H \rceil$ then $\mathbf{E}\left(\log(\frac{s_n(A)}{\tilde{s}_n})\right) = O(1)$;

iii) if $r = 2$, $\beta = 6 + \lceil 2\log(\log(H)) \rceil$ then $s_n = \tilde{s}_n$ with probability at least 1/3;

iv) if $r = \lceil 2\log(\log(H)) \rceil$, $\beta \geqslant 2$ then $\mathbf{E}\left(\log(\frac{s_n(A)}{\tilde{s}_n})\right) = O(1)$;

v) if $r = \log(\log(H)) + \log(\frac{1}{\epsilon})$, $2 \mid \beta$ and $\beta \geqslant 2$ then $s_n(A) = \tilde{s}_n$ with probability at least $1 - \epsilon$;

Proof. The proofs of (i), (ii) and (iv) are in [1]. The proof of (iii) is in [11]. To prove (v) we slightly modify the proof of (iv) in the following manner. From (i) we notice that for every prime p dividing s_n, the probability that it divides the missed part of $s_n(A)$ satisfies:

$$P(p \mid \frac{s_n}{\tilde{s}_n}) \leqslant (\frac{1}{2})^r.$$

As there are at most $\log(H)$ such primes, we get

$$P(s_n = \tilde{s}_n) \geqslant 1 - \log(H)(1/2)^r \geqslant 1 - \log(H)2^{-\log(\log(H))-\log(\frac{1}{\epsilon})} = 1 - \log(H)\frac{1}{\log(H)}\epsilon.$$

□

2.3 Abbott-Bronstein-Mulders, Saunders-Wan and Eberly-Giesbrecht-Villard ideas

Now, the idea of [1] is to combine both the Chinese remainder and the LIF approach. Indeed, one can first compute s_n and then reconstruct only the remaining factors of the determinant by reconstructing $\det(A)/s_n$. The complexity of this algorithm is $O\left(n^3 \log(|\det(A)/s_n(A)|)\right)$ which is unfortunately $O^{\sim}(n^4)$ in the worst case. However, nothing is known about the algorithm expected complexity.
Now Saunders and Wan [17, 20] proposed a way to compute not only s_n but also s_{n-1} (which they call a bonus) in order to reduce the size of the remaining factors $d/(s_n s_{n-1})$. The complexity doesn't change.
Then, Eberly, Giesbrecht and Villard have shown that the expected number of non trivial invariant factors is small, namely less than $\lceil 3log_\lambda(n)\rceil + 29$ in general if the entries of the matrix are chosen in a set of λ consecutive integers [11]. As they also give a way to compute any $s_i(A)$ which leads to an algorithm with expected complexity $O(n^3(\log(n) + \log(\|A\|))^2 \log(n)) \log_\lambda(n))$.
Our idea is to extend the method of Saunders and Wan to get the last $O(\log_\lambda n)$ invariant factors of A slightly faster than by [11]. Then, we are able to remove one of the $\log(n)$ factors of the expected complexity. Moreover, we will show in the following sections that this enables us to build an adaptive algorithm solving a minimal number of systems.
We should also mention, that it should be possible to change a $\log(n)$ factor in the expected complexity of [11] to a $\log\log(n)$ employing the bound for the expected number of invariant factors twice. Indeed their extra $\log(n)$ factor comes from the algorithm where n non trivial invariant factors are to be computed. But in the expected case, as they have only $\log(n)$ of those, this extra factor could be consequently reduced.

3 Computing the product of $O(\log(n))$ last invariant factors

3.1 On the number of invariant factors

The result in [11] says that a $n \times n$ matrix with entries chosen randomly and uniformly from a set of size λ has the expected number of invariant factors bounded by $\lceil 3\log(n)\rceil + 29$. In search for a sharpening of this result we prove the following theorems.

Theorem 3.1. *Let p be a prime. The expected number of non-trivial invariant factors divisible by p is at most 6.*

Theorem 3.2. *The expected number of nontrivial invariant factors is at most* $\log_\lambda(n) + \log_\lambda(\log_\lambda(n)+1) + 9$.

Both proofs can be found in the appendix A.

3.2 Extended Bonus Idea

In his thesis [20], Z. Wan introduces an idea of computing the penultimate invariant factor (i.e. s_{n-1}) of A while computing s_n using 2 system solvings. The additional cost is comparatively small, therefore s_{n-1} is referred to as a bonus. Here, we extend this idea to the computation of the $(n-k)$th factor with $(k+1)$ solvings.
Let $x^{(j)}$ be the rational solution to the equation $Ax^{(j)} = b^{(j)}$, where $b^{(j)}$ is a random vector. Then $x^{(j)}$ coordinates have a common denominator $\tilde{s}_n$ and we let $n^{(j)}$ denote the vector of numerators of $x^{(j)}$. Then $x^{(j)} = \frac{1}{\tilde{s}_n} n^{(j)}$ and $\gcd(n^{(j)}, \tilde{s}_n) = 1$.
Let B denote the $n \times (k+1)$ matrix $[b^{(j)}]_{j=1,\dots,k+1}$. Following Wan, we notice that $s_n(A)A^{-1}$ is an integer matrix, the Smith form of which is equal to

$$diag(\frac{s_n(A)}{s_n(A)}, \frac{s_n(A)}{s_{n-1}(A)}, \dots, \frac{s_n(A)}{s_1(A)}).$$

Therefore, we may compute $s_{n-k}(A)$ when knowing $s_{k+1}\left(s_n(A)A^{-1}\right)$. The trick is that the computation of A^{-1} is not required: we can perturb A^{-1} by right multiplying it by B. Then, $s_{k+1}(s_n(A)A^{-1}B)$ is a multiple of $s_{k+1}(s_n(A)A^{-1})$. Instead of $s_n(A)A^{-1}B$ we would prefer to use $\tilde{s}_n A^{-1}B$ which is already computed and equal to N, where $N = [n^{(j)}]_{j=1,\dots,k+1}$ is the matrix of numerators. The relation between A and N is as follows.

Lemma 3.3. *Let* $\tilde{s}_n^{-1}N$, $\gcd(\tilde{s}_n, N) = 1$ *be a solution to the equation* $AX = B$, *where* B *is* $n \times k$ *and the entries of* B *are uniformly and randomly chosen from the set* $\{0, 1, \dots \beta - 1\}$. *Then*

$$\frac{\tilde{s}_n}{\gcd(s_{i+1}(N), \tilde{s}_n)} | s_{n-i}(A), i = 1, 2 \dots, k.$$

Proof. The Smith forms of $s_n(A)A^{-1}B$ and N are connected by the relation $\frac{s_n(A)}{\tilde{s}_n} s_i(N) = s_i(s_n(A)A^{-1}B)$, $i = 1, \dots, (k+1)$. Therefore the quotient $\frac{s_n(A)}{s_{k+1}(s_n(A)A^{-1}B)}$ equals $\frac{\tilde{s}_n}{s_{k+1}(N)}$, and by taking $\frac{\tilde{s}_n}{\gcd(s_{k+1}(N), \tilde{s}_n)}$ one obtain an (integer) factor of $s_{n-k}(A)$. Moreover, the under-approximation is solely due to the choice of B.

□

Remark 3.4. Taking $\gcd(s_{k+1}(N), \tilde{s}_n)$ is necessary as $\frac{\tilde{s}_n}{s_{k+1}(N)}$ may be a rational number. Particularly, it happens when $s_1(N) = \gcd(N_{ij}) > 1$, and in this case, since $\gcd(\tilde{s}_n, N) = 1$, the impact of $s_1(N)$ on s_{n-k} is neglected. Moreover, this allows us to consider $p | \tilde{s}_n$ in all probability consideration throughout the paper.

In fact we are interested in computing the product $\pi_k = s_n s_{n-1} \cdots s_{n-k}(A)$ of the invariant factors of A. Then, following the idea of Abbott [1], we would like to reduce the computation of the determinant to the computation of $\frac{\det(A)}{\tilde{\pi}_k}$, where $\tilde{\pi}_k$ is a factor of π_k we have obtained.

We can compute $\tilde{\pi}_k$ as $\tilde{s}_n^{k+1}/\gcd(s_1 s_2 \cdots s_{k+1}(N), \tilde{s}_n^{k+1})$. The product of the $(k+1)$ first invariant factors of a matrix is equal to the gcd of all its $(k+1) \times (k+1)$ minors. In our approach, it suffices to compute $\lfloor n/(k+1) \rfloor$ of those. In the following lemmas we show that by repeating the choice of matrix B twice, we will omit only a finite number of bits in π_k. We start with a technical lemma, the proof of which is in Appendix B.

Lemma 3.5. *For $n \times n$ matrix A with entries chosen randomly and uniformly from the set $\{0, 1 \dots S-1\}$, the probability that $p^l < S$ divides the determinant $\det(A)$ is at most $\frac{3}{p^l}$ for $p \neq 2, 3, 5$ and $\frac{4}{p^l}$ for $p = 2, 3, 5$ and $S \geqslant 81$.*

In the next lemma we discuss the impact of choosing only a few minors in $s_1 \dots s_{k+1}(N)$ calculation. Here, $\mathrm{ord}_p(x)$ denotes the higher power of p dividing x.

Lemma 3.6. *Let N and B be as defined in lemma 3.3. Suppose that B is a random matrix with entries chosen uniformly form the set $\{0, 1, \dots S-1\}, S \geqslant \max(H, 81)$ and $k = O(\log(n))$. Let $N = [N_1 | \dots N_\mu | N']^T$ where N_i are $(k+1) \times (k+1)$ matrices, $\mu = \lfloor n/(k+1) \rfloor$. Then*

$$\sum_{l=1}^{\infty} \sum_{p^l | \tilde{s}_n} Prob\Big(\mathrm{ord}_p(\frac{\gcd_{i=1\dots\mu}(\det(N_i))}{\gcd(minors(N))}) = l \Big) l \log(p) \ \in \ O(1). \tag{1}$$

Proof. The bound S on $\|B\|$ is chosen in the way that following remark 3.4 we can only consider $p \leqslant S$. Therefore from Lemma 3.5 we get that for $l \geqslant 1$

$$Prob(p^l | \det(N_i)) \leqslant \frac{\alpha}{p^l},$$

where $\alpha = 3$ for $p > 5$ and $\alpha = 4$ for $p = 2, 3, 5$.
Now the sum (1), which represents the size of the over-approximation due to partial gcd computation can be bounded by

$$\sum_{l=1}^{\infty} \sum_{p^l | \tilde{s}_n} Prob\Big(\mathrm{ord}_p(\frac{\gcd_{i=1\dots\mu}(\det(N_i))}{\gcd(minors(N))}) = l \Big) l \log(p) \leqslant \sum_{l=1}^{\infty} \sum_{p^l | \tilde{s}_n} Prob\Big(p^l | \gcd_{i=1\dots\mu}(\det(N_i)) \Big) \log(p)$$

$$\leqslant \log 2 (2 + \sum_{l=3}^{\infty} (\frac{4}{2^l})^\mu) + \log 3 (1 + \sum_{l=2}^{\infty} (\frac{4}{3^l})^\mu) + \log 5 (1 + \sum_{l=2}^{\infty} (\frac{4}{5^l})^\mu) + \sum_{p>5} \sum_{l=1}^{\infty} \log(p) (\frac{3}{p^l})^\mu$$

$$\leqslant 3 + 4 + 6 + 1 = 14$$

With μ being about $\frac{n}{O(\log(n))}$, the expected size of overestimation due to partial gcd calculation is $O(1)$. □

To consider the impact of the choice of B on our method we start with a remark, which is a small modification of [20, Lem. 5.17].

Remark 3.7. For every matrix M there exist a full rank $k \times n$, $k \leqslant n$, matrix V, such that $\mathrm{ord}_p(\frac{s_1 \cdots s_k(MB)}{s_1 \cdots s_k(M)})$ is less or equal $\mathrm{ord}_p(\det(VB))$.

Lemma 3.8. *Let A be an $n \times n$ integer matrix, $S \geqslant \max(H, 81)$ and B_i, $i = 1, 2$ be $n \times k$ matrices with the entries uniformly and randomly chosen from the set $\{0, 1, \ldots S\}$. Then for $M = s_n(A)A^{-1}$*

$$\log\Big(\frac{\pi_k(A)}{s_n^k}\gcd(s_2 \cdots s_{k+1}(MB_1), s_2 \cdots s_{k+1}(MB_2))\Big) \ \in \ O(1).$$

Proof. From Remark 3.7 and Lemma 3.5, we have $Prob(B_i : \mathrm{ord}_p(\frac{s_1 \cdots s_k(MB_i)}{s_1 \cdots s_k(M)}) \leqslant l) \leqslant \frac{\alpha}{p^l}$, with $\alpha = 3$ for $p > 5$ and $\alpha = 4$ for $p = 2, 3, 5$. Now the expected size of the under-estimation is less than or equal to

$$\begin{aligned}\log(2)(1 + \sum_{l=2}^{\infty}(\frac{4}{2^l})^2) + \log(3)(1 + \sum_{l=2}^{\infty}(\frac{4}{3^l})^2) + \log(5)(1 + \sum_{l=2}^{\infty}(\frac{4}{5^l})^2) + \sum_{5<p\leqslant H}\sum_{l=1}^{\infty}\log(p)(\frac{3}{p^l})^2 \\ \leqslant 3 + 4 + 6 + \int_6^{\infty}\log(x)\frac{9}{x^2-1}dx,\end{aligned}$$

which is $O(1)$. □

It is worth noting that the above mentioned schemes could lead to an algorithm to compute several last (or first) invariant factors with possibly better probabilistic behavior and expected complexity than that of [11].

4 Introspective Algorithm

Now we should incorporate algorithm 2.1 and the ideas presented in sections 2.2 and 3.2 in the form of an introspective algorithm. Indeed, we give a recipe for an auto-adaptive program that implements several algorithms of diverse space and time complexities for solving a particular problem. The best path is chosen at run time, from a self-evaluation of the dynamic behavior (e.g. timings) while processing a given instance of the problem. This kind of auto-adaptation is called introspective in [3].

In the following, CRA loop refers here to algorithm 2.1, slightly modified to compute $\det(A)/K$. If we re-run the CRA loop, we use modular determinant results already computed to recover $\det(A)/K \bmod p$.

Theorem 4.1. *Algorithm 4.1 correctly computes the determinant with probability $1 - \epsilon$.*

Proof. Termination is possible only by the early terminated CRA loop or by the determinant algorithm used in the last step. The choice of k from theorem 2.1(iii) and the choice of the determinant algorithm from [13, 19] ensures that $1 - \epsilon$ probability is obtained. □

4.1 Complexity

The following theorem gives the complexity of the algorithm.

Algorithm 4.1 Extended Bonus Determinant Algorithm

Require: An integer $n \times n$ matrix A.
Require: $0 < \epsilon < 1$, an error tolerance.
Require: A stream S of random integers uniformly chosen from the set $\{0, 1 \ldots, \max(\lceil (n+1)H \rceil, 81)\}$, H - Hadamard's bound for A.
Require: A set P of random primes greater than l.
Ensure: The integer determinant of A, correct with probability at least $1 - \epsilon$.

$k = \log(1/\epsilon)/\lceil \log(\frac{P'}{\log_l(H)}) \rceil$; see Lem. 2.1(iii)
for $i = 1$ to k **do**
  run the CRA loop for $\det(A)$; //see Alg. 2.1
  if early terminated **then**
    Return determinant;
  end if
end for
$i_{max} = \log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 1) + 9)$; //see Theorem 3.2
for $j = 1, 2$ **do**
  $i = 0; \tilde{\pi}_{-1} = 1; K = 1;$
  while $i < i_{max}$ **do**
    Generate b_i a random vector of dimension n from the stream S;
    Compute $\tilde{s}_n$ by solving $Ax_i = b_i$; //see Section 2.2
    if $i = 0$ **then**
      $i = 1; \tilde{\pi}_0 = \tilde{s}_n;$
    else
      $N := \tilde{s}_n X$, where $X = [x_l]_{l=0,\ldots i}$; //see Section 3.2;
      $\tilde{\pi}_i = 0;$
      for $l = 1, \ldots \lfloor n/i \rfloor$ **do**
        $\tilde{\pi}_i = \gcd(\tilde{\pi}_i, \det(N_l))$, where N_l is the lth minor of N;
      end for
      i=i+1;
    end if
    $K = \mathrm{lcm}(\tilde{\pi}_{i-1}, K)$; $\tilde{\pi}_{i-1} = K;$
    Resume CRA looping on $d = \det(A)/K$; for at most the time of one system solving;
    if early terminated **then**
      Return $d \cdot K$;
    end if
    if $\tilde{\pi}_{i-1} = \tilde{\pi}_{i-2}$ **then**
      Resume CRA looping on $d = \det(A)/K$; for at most the time of $(i_{max} - i)$ system solvings;
      if early terminated **then**
        Return $d \cdot K$;
      else
        $i = i_{max};$
      end if
    end if
  end while
end for
run an asymptotically better integer determinant algorithm;

Theorem 4.2. *The expected complexity of Algorithm 4.1 is*

$$\mathrm{O}^{\approx}\left(n^{\omega}\log(1/\epsilon)+n^3(\log n+\log(\|A\|))^2\log(n)\right)$$

where $\mathrm{O}^{\approx}$ *hides some* $\log(\log(n))$ *factors. The pessimistic complexity depends on the algorithm used in the last step.*

Proof. To analyze the complexity of the algorithm we would consider the complexity of each step. With k defined as in the algorithm, the complexity of initial CRA iteration is $O\left(n^{\omega}\log(1/\epsilon)\right)$. The system solving in the LIF algorithm is performed $2i_{max}$ times with $\log(\|B\|)=O(n\log(n))$, which results with a complexity of $O(i_{max}n^3(\log(n)+\log(\|A\|)^2)$. Considering the time limit, this is also the time of all CRA loop iterations. To compute $\tilde{\pi}_i$ by means of the CRA determinant algorithm, we need $O\Big(\lfloor n/i\rfloor i^4(\log(i)+n(\log(n)+\log(\|A\|)+\log(\|B\|))\Big)$ bit operations, which for $i=2,\ldots,i_{max}$ with i_{max} being $O(\log(n))$ is $\mathrm{O}^{\sim}(n^2)$ and thus negligible.
With the expected number of invariant factors bounded by $i_{\max}$ (see Thm.3.2), it is expected that the algorithm will return the result before the end of the *while* loop, provided that the under-estimation of $\tilde{\pi}_{i_{max}}$ is not too big. But by updating $\tilde{s}_n$ $O(\log(n))$ times and updating the product $\tilde{\pi}_{i_{max}}$ twice, it is expected that the overall under-estimation will be $O(1)$ (see Theorem 2.2(ii) and Lemma 3.8), thus it is possible to recover it by several CRA loop iterations.

□

5 Experiments and Further Adaptivity

The described algorithm is implemented in the LinBox exact linear algebra library [8]. In a preliminary version i_{max} is set to 2 or 1 and the switch in the last step is not implemented. This is however enough to evaluate the performance of the algorithm and to introduce further adaptive innovations.
Comparing the data from table 5 we notice that the algorithm with $i_{max}=1$ (which is in fact a slightly modified version of Abbott's algorithm [1]) runs better for small n. Those timings have been evaluated on a set of matrices which have the same Smith form as $diag\{1,2,..,n\}$ and the number of invariant factors of about $\frac{n}{2}$. For every matrix, with each step, the size of s_{n-i} decreases whilst the cost of its computation increases. This accounts for better performance of Abbott's algorithm, which computes only s_n, in the case of small n. For bigger n calculating s_{n-1} starts to pay out. The same situation repeats at each step.
The switch between winners can be explained by the fact that in some situations, obtaining s_{n-i} by LU-factorization (which costs $\frac{\log(s_{n-i})}{\log(l)}$ the time of LU) outperforms system solving. Then, this also holds for all consecutive factors and the algorithm basing on CRA wins. The condition can be checked *a posteriori* by approximating the time of LUs needed to compute the actual factor. We can therefore construct a condition that would allow us to turn to the CRA loop in the appropriate moment. This can be done by changing the condition in

n	$i_{max}=1$	$i_{max}=2$	n	$i_{max}=1$	$i_{max}=2$
100	0.17	0.22	300	5.65	5.53
120	0.29	0.33	350	9.76	9.64
140	0.48	0.55	400	14.99	14.50
160	0.73	0.78	600	57.21	54.96
180	1.07	1.16	800	154.74	147.53
200	1.49	1.51	1000	328.93	309.61
250	2.92	3.00	2000	3711.26	3442.29

Table 1: Comparison of the performance of Algorithm 4.1 with $i_{\max}$ set to 1 and 2 on engineered matrices.

line 24 ($\tilde{\pi}_{i-1} = \tilde{\pi}_{i-2}$) to

$$\log(\frac{\tilde{\pi}_{i-1}}{\tilde{\pi}_{i-2}}) \leqslant \frac{time(solving)}{time(LU)} \log(l),$$

if the primes used in the CRA loop are greater than l. This would result with a performance close to the best and yet flexible. If, to some extend, s_{n-i} could be approximated *a priori*, this condition could be checked before its calculation. This would require a partial factorization of s_{n-i+1} and probability considerations as in the appendix A and [11].
For a generic case of random dense matrices our observation is that the bound for the number of invariant factors is quite crude. Therefore the algorithm 4.1 is constructed in the way that minimizes the number of system solving to at most twice the actual number of invariant factors for a given matrix. Under the assumption that the approximations $\tilde{s}_n$ and $\tilde{\pi}_i$ are sufficient, this leads to a quick solution.
Indeed for random matrices, the algorithm nearly always stopped with early termination after one system solving. This together with fast underlying arithmetics of FFLAS [7] accounted for the superiority of our algorithm as seen in figure 5 where comparison of timings for different algorithms is presented.

6 Conclusions

In this paper we presented an algorithm computing the determinant of an integer matrix which expected time complexity is $O\left(n^3(\log(n)+\log(||A||))^2\log(n)\right)$. Our algorithm uses an introspective approach so that its actual running time is only $O\left(n^3(\log(n)+\log(||A||))^2 k\right)$ if the number k of invariant factors is smaller than *a priori* expected. Moreover, the adaptive approach allows us to switch to the algorithm with best worst case complexity if it happens that the number of nontrivial invariant factors is unexpectedly large. This adaptivity, together with very fast modular routines, allows us to produce an algorithm, to our knowledge, faster by at least an order of magnitude than other implementations.
Ways to further improve the running time are to reduce the number of iterations in the solvings or to group them in order to get some block iterations as is done e.g. in [2]. A

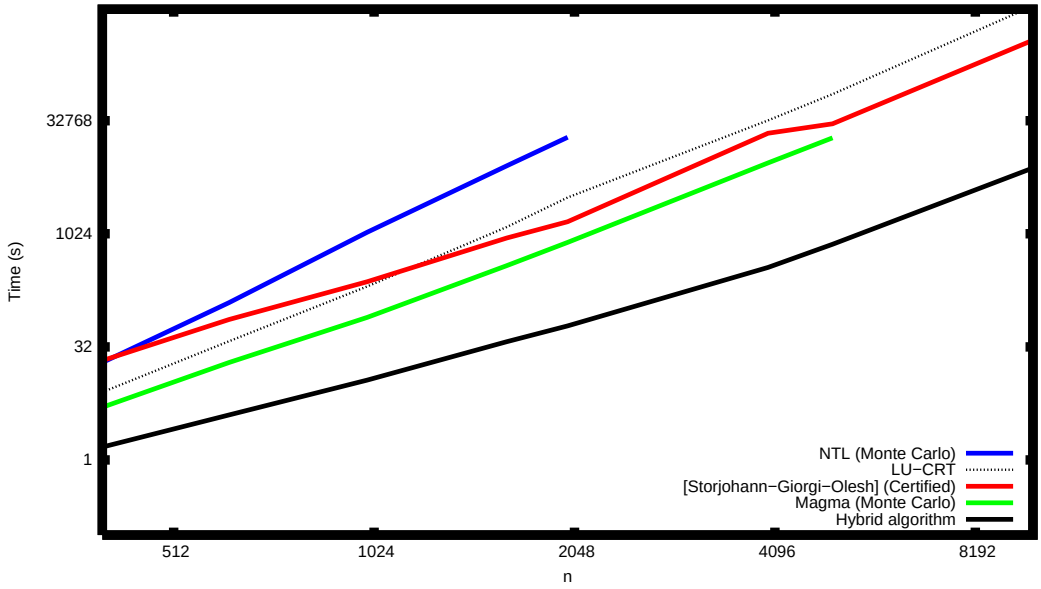

Figure 1: Comparison of our algorithm with other existing implementation. Tested on random dense matrices of the order 400 to 10000, with entries {-8,-7,. . . ,7,8} Using fast modular routines puts our algorithm several times ahead of the others. Scaling is logarithmic.

modification to be tested, is to try to reconstruct s_n with only some entries of the solution vector $x = \mathbf{n}/d$. Parallelization can also be considered to further modify the algorithm. Of course, all the LU iterations in one CRA step can be done in parallel. An equivalently efficient way is to perform several p-adic liftings in parallel, but with less iterations [6]. There the issue is to perform an optimally distributed early termination.

References

[1] J. Abbott, M. Bronstein, T. Mulders. Fast deterministic computation of determinants of dense matrices. In *Proc. of ACM International Symposium on Symbolic and Algebraic Computation (ISAAC'1999)*, 197-204, ACM Press, 1999.

[2] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In *Proc. of ACM International Symposium on Symbolic and Algebraic Computation (ISAAC'2005)*, 92–99, ACM Press, 2005.

[3] V.-D. Cung, V. Danjean, J.-G. Dumas, T. Gautier, G. Huard, B. Raffin, C. Rapine, J.-L. Roch, D. Trystram, Adaptive and hybrid algorithms: classification and illustration on triangular system solving, in: *Proceedings of Transgressive Computing 2006, Granada, España.*

[4] J. Dixon. Exact Solution of Linear Equations Using P-Adic Expansions. In *Numer.Math.* 40(1), 137-141, 1982.

[5] J.G. Dumas, D. Saunders, G. Villard. On Efficient Sparse Integer Matrix Smith Normal Form Computations. In *Journal of Symbolic Computations.* 32 (1/2), 71-99, 2001.

[6] J.G. Dumas, W. Turner, Z. Wan. Exact Solution to Large Sparse Integer Linear Systems. *ECCAD'2002 : The 9th Annual East Coast Computer Algebra Day*, 2002.

[7] J.G. Dumas, T. Gautier, C. Pernet. FFLAS: Finite field linear algebra subroutines. *ISSAC'2002.* 2002.

[8] J.G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, D. Saunders, W. Turner, G. Villard. LinBox: A Generic Library for Exact Linear Algebra. *ICMS'2002 : International Congress of Mathematical Software* 2002.

[9] J.G. Dumas, P. Giorgi, C. Pernet. FFPACK: finite field linear algebra package. *ISSAC'2004.* 2004.

[10] J.G. Dumas, C. Pernet, Zhendong Wan. Efficient Computation of the Characteristic Polynomial. *ISSAC'2005.* 2005.

[11] W. Eberly, M. Giesbrecht, G. Villard. On computing the determinant and smith form of an integer matrix. In *Proc. 41st FOCS*, 675-687, 2000.

[12] O.H. Ibarra, S. Moran, R.Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45Ž01356, Mar.1982.

[13] E. Kaltofen, G. Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. In *Journal of Computational and Applied Mathematics* 164(2004), 133-146 2004.

[14] E. Kaltofen, G. Villard. On the complexity of computing determinants. *Computational Complexity*, 31(3-4), pp 91–130, 2005.

[15] D. Musser. Introspective Sorting and Selection Algorithms. *Software—Practice and Experience*, 8(27), pp 983–993, 1997.

[16] V. Pan. Computing the determinant and the characteristic polynomial of a matrix via solving linear systems of equations. *Inform. Process. Lett.* 28(1988) 71-75. 1988.

[17] D. Saunders, Z. Wan. Smith Normal Form of Dense Integer Matrices, Fast Algorithms into Practice. *ISSAC 2004* 2004.

[18] A.Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21(4), pp 609–650, 2005.

[19] A. Storjohann, P. Giorgi, Z. Olesh. Implementation of a Las Vegas integer Matrix Determinant Algorithm. *ECCAD'05: East Coast Computer Algebra Day*, 2005.

[20] Z. Wan. Computing the Smith Forms of Integer Matrices and Solving Related Problems. Ph.D. Thesis, U. of Delaware, USA, 2005.

A Proofs of theorems 3.1 and 3.2

In order to prove theorems stated in section 3.1, we will start with the following lemma.

Lemma A.1. *For $\lambda > 11$ the sum over primes p: $\sum_{8<p<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^j$ can be bounded by $(\frac{1}{2})^j$.*

Proof. We will consider primes from the interval $\frac{\lambda}{2^{k+1}} \leqslant p < \frac{\lambda}{2^k}$, $k = 0, 1, \ldots \max\{\lceil\log(\lambda)\rceil - 3, 2\}$ separately. For the kth interval $\lceil\frac{\lambda}{p}\rceil$ equals 2^{k+1}. In each interval there are at most $\lceil\frac{\lambda}{4}\rceil$ odd numbers and at most $\frac{\lambda}{4}$ primes. The reasoning goes as follows: if in the interval there are more than 3 odd numbers, at least one of them is divided by 3 and so does not count. For this to happen it is enough that $\lambda \geqslant 12$. We may therefore calculate:

$$\sum_{8<p<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^j \leqslant \sum_{k=0}^{\lceil\log(\lambda)\rceil-3} \frac{\lambda}{2^{k+2}}(\frac{2^{k+1}}{\lambda})^j \leqslant \frac{1}{2\lambda^{j-1}}(2^{\lceil\log(\lambda)\rceil-2})^{j-1} \leqslant (\frac{1}{2})^j.$$

□

Remark A.2. For $\lambda = 2^l$ we may consider primes $p > 4$.

Remark A.3. If we exclude {2,3,5,7,8,16}, we get the same bound for $\sum_{k\in\mathbf{N}} \sum_{8<p^k<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p^k}\rceil)^j$.

Proof.[Theorem 3.1] The idea of the proof is similar to that of [11]. Let A be a random matrix with entries chosen uniformly and randomly from the set $\{0, 1, 2 \ldots \lambda - 1\}$. Let $MDep_i(p)$ denote an event that the submatrix A_i, including the first i columns of A mod p has rank at most $i - 2$ over $\mathbf{Z}_p$.
We are now going to find $P(MDep_i(p) \mid \neg MDep_{i-1}(p))$. Since the event $MDep_{i-1}(p)$ did not occur, A_{i-1} has p-rank $(i - 2)$ or $(i - 1)$. For $MDep_i$ it must be $(i - 2)$, thus, there exists a set of $(i-2)$ rows R_{i-2} which has the full rank. Consider any row v_j that is left. If v_j is a combination of R_{i-2} the last (ith) entry of v_j is determined mod p. For $\lambda \geqslant p$ this means that the probability that v_j is a combination of R_{i-2} is at most λ^{-1}. For $p < \lambda$ this probability is $\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil$ which is always less than or equal to $\frac{2}{p+1}$. As there are $n - i + 2$ vectors outside R_{i-2}, the probability that none of them is linearly independent with R_{i-2} over $\mathbf{Z}_p$ is at most $(\frac{2}{p+1})^{n-i+2}$ for $p < \lambda$ and $(\frac{1}{\lambda})^{n-i+2}$ for $p \geqslant \lambda$.
Since $P(MDep_i(p) \mid \neg MDep_{i-1}(p)) \geqslant P(MDep_i(p) \wedge \neg MDep_{i-1}(p))$, we have $P(MDep_i(p)) \leqslant P(\bigcup_{j=1}^{i}(MDep_j(p) \wedge \neg MDep_{j-1}(p)))$ which can be bounded by $(\frac{2}{p+1})^{n-i+2}\frac{p+1}{p-1}$ for $p < \lambda$ and $(\frac{1}{\lambda})^{n-i+2}\frac{\lambda}{\lambda-1}$ for $p \geqslant \lambda$.
Let the number of invariant factors divided by p be greater than j. The rank of A mod p is then at most $n-j$ over $\mathbf{Z}_p$. This in consequence means that for $j > 1$ the submatrix A_{n-j+2}

has the rank at most $n-j$, so the event $MDep_{n-j+2}(p)$ is fulfilled. Therefore matrix A has at least j invariant factors divided by p with probability at most

$$\begin{aligned}(\frac{2}{p+1})^j\frac{p+1}{p-1}, &\quad p<\lambda\\ (\frac{1}{\lambda})^j\frac{\lambda}{\lambda-1}, &\quad p\geqslant\lambda.\end{aligned}\tag{2}$$

Now the expected number of invariant factor divided by p is not greater than

$$\begin{aligned}3+3\sum_{j=3}^{j=n}(\frac{2}{3})^j=3+9(\frac{2}{3})^3\leqslant 6, &\quad p=2,\\ 1+\sum_{j=1}^{j=n}(\frac{2}{p+1})^j\frac{p+1}{p-1}=1+\frac{2(p+1)}{(p-1)^2}\leqslant 3, &\quad 2<p<\lambda,\\ 1+\frac{\lambda}{(\lambda-1)^2}<2, &\quad p\geqslant\lambda>2.\end{aligned}\tag{3}$$

□ *Proof.*[Theorem 3.2] In addition to $MDep_i(p)$ introduced earlier, let Dep_i denote an event that the first i columns of A are linearly independent and $MDep_i$, an event that either of $MDep_i(p)$ occurred. Recall that $P(Dep_1\vee MDep_1(p))\leqslant\lambda^{-n}$, and $P(Dep_i\mid\neg(Dep_{i-1}\vee MDep_{i-1}(p)))\leqslant\lambda^{-n+i-1}$.
To bound $P(MDep_i\mid\neg(Dep_{i-1}\vee MDep_{i-1}(p)))$ we sum the results for all primes. For $p<\lambda$, $i\leqslant n-1$ the sum can be bounded by

$$(\frac{2}{3})^{n-i+2}+\sum_{\lambda>p>8}(\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^{n-i+2}\leqslant(\frac{2}{3})^{n-i+2}+(\frac{1}{2})^{n-i+2},$$

thanks to the lemma A.1.
For primes $p\geqslant\lambda$ we should estimate the number of primes dividing the $(i-1)$th minor. By the Hadamard's bound (notice that Dep_{i-1} does not hold), the minors are bounded in absolute value by $((i-1)\lambda^2)^{\frac{i-1}{2}}$. Therefore the number of primes $p\geqslant\lambda$ dividing the minor is at most $\frac{i-1}{2}(\log_\lambda(i-1)+2))$. Summarizing,

$$\begin{aligned}&P((MDep_i\wedge Dep_i)\mid\neg(Dep_{i-1}\vee MDep_{i-1}(p)))\\ &\qquad\leqslant(\frac{1}{\lambda})^{n-i+1}+(\frac{2}{3})^{n-i+2}+(\frac{1}{2})^{n-i+2}+\frac{i-1}{2}\,(\log_\lambda(i-1)+2)\,(\frac{1}{\lambda})^{n-i+2}\end{aligned}$$

for $2\leqslant i\leqslant n-1$.
By the same argument as in the previous proof

$$P(MDep_{n-j+2})\leqslant\lambda^{-n}+(\frac{1}{\lambda})^{j-1}\frac{\lambda}{\lambda-1}+3(\frac{2}{3})^j+2(\frac{1}{2})^j+\frac{n-j+1}{2}\,(\log_\lambda(n-i+1)+2)\,(\frac{1}{\lambda})^j\frac{\lambda}{\lambda-1}.\tag{4}$$

Similarly, the probability that the number of invariant factors at least j is greater than $P(MDep_{n-j+2})$.
To calculate the expected number of invariant factors we first consider the case

$$\frac{n-j+1}{2}\big(\log_\lambda(n-j+1)+2\big)(\frac{1}{\lambda})^j\frac{\lambda}{\lambda-1}<1.$$

It suffices that $n(\log_\lambda(n)+1)\leqslant\lambda^j$, and therefore $\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+1)\leqslant j$. Consequently, the expected number of invariant factors is

$$\sum_{j=1}^{\lceil\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+1)\rceil}1+\sum_{j=\lceil\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+1)\rceil+1}^{n}\big(\lambda^{-n}+(\frac{1}{\lambda})^{j-1}\frac{\lambda}{\lambda-1}+3(\frac{2}{3})^j+$$
$$+2(\frac{1}{2})^j+\frac{n-j+1}{2}(\log_\lambda(n-j+1)+2))(\frac{1}{\lambda})^j\frac{\lambda}{\lambda-1}\big)=\lceil\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+1)\rceil+$$
$$+1+\frac{\lambda+1}{(\lambda-1)^2}+4+1\leqslant\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+1)+9.$$

□

B Modular determinant

Lemma B.1. *For $n\times n$ matrix A with entries chosen randomly and uniformly from the set $\{0,1\dots S-1\}$, the probability that $p^l<S$ divides the determinant* $\det(A)$ *is at most $\frac{3}{p^l}$ for $p\neq 2,3,5$ and $\frac{4}{p^l}$ for $p=2,3,5$ and $S\geqslant 81$.*

Proof. To check whether $\mathrm{ord}_p(\det(A))\geqslant l$ we will consider a process of diagonalization for A mod p^l as described in Algorithm *LRE* of [5]. It consists of diagonalization and reduction steps. At ith diagonalization step, if an invertible entry is found, it is placed in the (i,i) pivot position and the ith row and column are zeroed. If no invertible entry is found, we proceed with a reduction step i.e. we consider the remaining $(n-i+1,n-i+1)$ minor divided by p. The problem now reduces to determining whether ord_p of an $(n-i+1,n-i+1)$ matrix is greater than $l-n+i-1$.
In the probabilistic consideration we need to determine the distribution of entries mod p^i after each reduction step. First, for A with entries chosen uniformly and randomly from the set $\{0,1\dots S\}$, the probability that an entry is determined mod $p^i, i\leqslant l$, is less than or equal to $\beta_i(0)=\frac{1}{S}\lceil\frac{S}{p^i}\rceil$. After k reductions and m diagonalization steps we consider $i\leqslant l-2k$ (each reduction is performed on a matrix of order at least 2 and reduces the determinant by at least p^2) and a conditional probability of choosing the entries of a matrix A_k^m determined mod p^i with a probability at most $\beta_i(k)=\frac{1}{N_k}\lceil\frac{N_k}{p^i}\rceil$, where $N_k=\left\lceil\frac{\vdots}{p}\right\rceil^{\left\lceil\frac{\lceil\frac{S}{p}\rceil}{p}\right\rceil}$ (the division is repeated k times). Since k is less than or equal to $\lceil l/2\rceil-1$ and $l\leqslant log_p(S)$, we have $N_k\geqslant\frac{S}{p^k}\geqslant\sqrt{S}$ and $\beta_i\leqslant\frac{2}{p^i}$ throughout the diagonalization process.

We will estimate the probability inductively. The estimations are performed for $p > 5$. First, for $l = 1$, we recall the result of [20, p.62] that

$$P(p \mid \det(A)) \leqslant \sum_{i=1}^{n} \beta_1 \leqslant \frac{3}{p}.$$

Then for $n = 2, l = 2$,

$$P(p^2 \mid \det(A)) \leqslant (1 - P(p \mid a_{ij} \forall_{i,j}))\beta_2 + P(p \mid a_{ij} \forall_{i,j}) \leqslant \beta_2 + (\beta_1)^{2^2} \leqslant \frac{3}{p^2}.$$

Again for $n = 2, 1 < l < n$ this becomes

$$P(p^l \mid \det(A)) \leqslant (1 - P(p \mid a_{ij} \forall_{i,j}))\beta_l + P(p \mid a_{ij} \forall_{i,j}) P(p^{l-2} \mid \det(A_1)).$$

Notice, that we sum over all possible diagonalization/reduction steps combinations. β_i bounds the probability of choosing the last diagonal entry determined mod p^i. A_1 is equal to A/p. By induction for $p > 5$, $P(p^l \mid \det(A))$ can be bounded by $\frac{2}{p^l} + (\frac{2}{p})^4 \frac{3}{p^{l-2}} \leqslant \frac{3}{p^l}$.
Now we will consider $n > 2$. Again we can sum over all possible diagonalization/reduction steps combinations and the resulting bound for the probability is

$$\begin{aligned} P(p^l \mid \det(A)) \leqslant & \sum_{i=n}^{l} (1 - P(p \mid a_{ij} \forall_{i,j \leqslant n})) \dots (1 - P(p \mid a_{ij}^{n-i+1} \forall_{i,j \leqslant i+1})) \beta_1^{i^2} \\ & + \sum_{i=l-1}^{2} (1 - P(p \mid a_{ij} \forall_{i,j \leqslant n})) \dots (1 - P(p \mid a_{ij}^{n-i+1} \forall_{i,j \leqslant i+1})) \beta_1^{i^2} P(p^{l-i} \mid \det(A_1^i)) \\ & + (1 - P(p \mid a_{ij} \forall_{i,j \leqslant n})) \dots (1 - P(p \mid a_{ij}^{n-1} \forall_{i,j \leqslant 2})) \beta_l \end{aligned}$$

for $l \leqslant n$ and similarly for $l > n$

$$\begin{aligned} P(p^l, A) \leqslant & \sum_{i=n}^{2} (1 - P(p \mid a_{ij} \forall_{i,j \leqslant n})) \dots (1 - P(p \mid a_{ij}^{n-i+1} \forall_{i,j \leqslant i+1})) \beta_1^{i^2} P(p^{l-i} \mid \det(A_1^i)) \\ & + (1 - P(p \mid a_{ij}^{n-i+1} \forall_{i,j \leqslant n})) \dots (1 - P(p \mid a_{ij}^{n-1} \forall_{i,j \leqslant 2})) \beta_l. \end{aligned}$$

Again, we can use the induction to get

$$P(p^l \mid \det(A)) \leqslant \big(\sum_{i=2}^{l} (\frac{2}{p})^{i^2} \frac{3}{p^{l-i}} \big) + \frac{2}{p^l} \leqslant \sum_{i=0}^{\infty} (\frac{3 \cdot 2^4}{p^{l+2}} (\frac{2^5}{p^4})^i) + \frac{2}{p^l} \leqslant \frac{3 \cdot 2^4 p^4}{p^{l+2}(p^4 - 2^5)} + \frac{2}{p^l} \leqslant \frac{3}{p^l}$$

for $p > 5$. For $p = 2, 3, 5$ by similar calculations we can prove that, provided that $N_k \geqslant 9$, $P(p^l, A) \leqslant \frac{4}{p^l}$. The condition on N_k is satisfied as soon as $\sqrt{S} \geqslant 9$ i.e. $S \geqslant 81$.

□

Laboratoire de Modélisation et Calcul
Université Joseph Fourier, Grenoble I
BP 53X, 38041 Grenoble, FRANCE
{Jean-Guillaume.Dumas;Anna.Urbanska}@imag.fr
www-lmc.imag.fr/lmc-mosaic/{Jean-Guillaume.Dumas;Anna.Urbanska}