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 $O\left(n^3 \log^{2.5}(n\|A\|)\right)$ *bit operations* in the dense case and $O\left(\Omega n^{1.5} \log^2(n\|A\|) + n^{2.5} \log^3(n\|A\|)\right)$ in the sparse case, where $\|A\|$ is the largest entry in absolute value of the matrix and Ω is the cost of matrix-vector multiplication in the case of a sparse 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 [17]. 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 [10].

Therefore, the classical dense approach over the integers is 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 [11, §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 $O(n\log(n))$ (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 the determinant with a $O\left(n^3(\log(n)+\log(\|A\|))^2\right)$ bit complexity [22]. 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 system solving and recover only the remaining part $(\det(A)/s_n)$ via Chinese remaindering.

Then G. Villard remarked that at most $O(\sqrt{\log(|\det(A)|)})$ invariant factors can be distinct and that we can expect that only the last $O(\log(n))$ of those are nontrivial [15]. This remark, together with a preconditioned p-adic solving to compute the i-th invariant factor lead to a $O^\sim(n^{2+\omega/2})$ worst case algorithm [15], 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 for dense matrices is $O^\sim(n^{2.7})$, which is $O^\sim(n^{3.2})$ without fast matrix multiplication, by [19]. Unfortunately, these last two worst case complexity algorithms, though asymptotically better than [15], are not the fastest for the generic case or for the actually attainable matrix sizes. The best expected complexity algorithm is the Las Vegas algorithm of Storjohann [24] which uses an expected number of $O^\sim(n^\omega)$ bit operations. In section 5 we compare the performance of this algorithm to ours, based on the experimental results of [25].

For sparse matrices the classical approach using Chinese remaindering gives the complexity of $O(\Omega n\log(|\det(A)|))$.

In this paper, we propose a new way to extend the idea of [23, 26] 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 introduced in [21] as introspective; here we use the more specific definition

of [12]. Furthermore, we show how to modify the algorithm in the case of a sparse matrix. In section 4.2 we prove that the expected complexity of our algorithm is

$$O(n^3(\log(n) + \log(\|A\|))^2\sqrt{\log(n)}$$

bit operations in the case of dense matrices, gaining a $\log^{1.5}(n)$ factor compared to [15]. For the sparse case, using a sparse linear solver of [16], we claim the expected complexity

$$O\left(\Omega n^{1.5}(\log(n) + \log(\|A\|))\log(n) + n^{2.5}(\log(n) + \log(\|A\|))\log(\|A\|)\log(n)\right),$$

assuming that the cost of one matrix-vector product is Ω. Our analysis leads to the conclusion that the ideas of [1, 15] can also be applied to the sparse case. By moving to the sparse solver of [16] inside the algorithm of [15] we could obtain an algorithm with a worst case time complexity of $O\left(n^3(\log(n) + \log(\|A\|))^{1.5}\log(\|A\|)\log^2(n)\right)$ and the expected complexity of $O^{\approx}\left(n^{2.5}(\log(n) + \log(\|A\|))\log(\|A\|)\log^2(n)\right)$, where $O^{\approx}$ hides some $(\log(n) + \log(\|A\|))$ factors.

Moreover, we are able to detect the worst cases during the course of the algorithm and thus 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, 10] and the LinBox library [8], our algorithm can be an order of magnitude faster than other existing implementations.

A preliminary version of this paper was presented in the Transgressive Computing 2006 conference [13]. Here we give better asymptotic results for the dense case, adapt our algorithm to the sparse case and give more experimental evidences.

2 Base Algorithms and Procedures

In this section we present the procedures in more details 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, 15, 23]). We end the section with a summary of ideas of Abbott *et al.* [1], Eberly *et al.* and Saunders *et al.* [23].

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 $p_0 \cdots p_n$ 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 sufficiently big 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 this 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. *Let H be an upper bound for the determinant (e.g. H can be the Hadamard's bound: $|det(A)| \leq (\sqrt{n}\|A\|)^n$). Suppose that primes p_i greater than $l \geq 4$ are randomly sampled form a set P with $|P| \geq \lceil 2\log_l(H)\rceil$. Then let r_t be the value of the determinant modulo $p_0 \cdots p_t$ computed in the symmetric range. We have :*

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

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

(iii) if $r_t = r_{t+1} = \cdots = r_{t+k}$ *and* $k \geq \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\log_l(H))}\rceil$, *where* $P' = |P| - t - 1$, *then* $P(r_t \neq \det(A)) < \epsilon$.

Proof. For (i), notice that $-\lfloor \frac{p_0 \cdots p_t}{2} \rfloor \leq r_t < \lceil \frac{p_0 \cdots p_t}{2} \rceil$. Then $r_t = \det(A)$ as soon as $p_0 \cdots p_t \geq 2|\det(A)|$. With l being the lower bound for p_i this reduces to $t \geq \lceil \log_l |\det(A)| \rceil$ when $\det(A) \neq 0$.
For (ii), we observe that $\det(A) = r_t + K p_0 \ldots p_t$ and it suffices to estimate the number of primes greater than l dividing K.
For (iii) we notice that k primes dividing K are to be chosen with probability less than $\frac{\binom{R}{k}}{\binom{|P|-(t+1)}{k}}$. The latter is bounded by $(\frac{R}{P'})^k$ since $R \leq \lceil \log_l(\frac{2H}{2})\rceil \leq |P'|$. Solving for k for the latter to be less than ϵ gives the result. □

To compute the modular determinant in algorithm 2.1 we use the LU factorization procedure in the dense case (complexity $O(n^\omega)$).

In the sparse case, sparse elimination can be used in practice e.g. for extremely sparse matrices [9]. In general, black box method are prefered.

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: $H = (\sqrt{n}\|A\|)^n$; $P' = |P| - \lceil \log_l(H) \rceil$; $i = 0$; // Hadamard's bound
2: **repeat**
3: Get a prime p_i from the set P;
4: $P = P \backslash \{p_i\}$
5: Compute $\det(A) \bmod p_i$;
6: Reconstruct r_i, the determinant modulo $p_0 \cdots p_i$; // by Chinese remaindering
7: $k = \max\{t : r_{i-t} = \cdots = r_i\}$; $R = \lceil \log_l \frac{H+|r_i|}{p_0 p_1 \cdots p_{i-k}} \rceil$
8: Increment i;
9: **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 idea is to precondition the matrix so that its characteristic polynomial equals its minimal polynomial [14, 2]; and then to compute the minimal polynomial via Wiedemann's algorithm [27]. The complexity of the sparse modular determinant computation is then $O(\Omega n)$ [9, Table 4].

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 [22] and later in the form of the *LargestInvariantFactor* procedure (LIF) in [1, 15, 5, 23]. 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 $d(x) = d(y)|s_n(A)$.

Thus, solving $Ax = b$ enables us to get $s_n(A)$ with high probability. The cost of solving using Dixon p-adic lifting [4] is $O\left(n^3(\log(n) + \log(\|A\|))^2 + n\log^2(\|b\|)\right)$

as stated by [20]. For the sparse matrix case a sparse linear solver of [16] can be used and by similar reasoning as in [20], the cost of one solving is that of $O(n^{1.5}(\log(n)+\log(\|A\|))+n^{0.5}\log(\|b\|))$ matrix-vector products and $O(n^2(\log(n)+\log(\|A\|))(n^{0.5}+\log(\|A\|))+n^2\log(\|b\|)(\log(n)+\log(\|A\|))+n\log^2(\|b\|)$ additional arithmetic operations.

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, i.e. a bound for $\|b\|$. 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, $l \geq 1$, then $P(p^l | \frac{s_n(A)}{\tilde{s}_n}) \leq \frac{1}{\beta}\lceil\frac{\beta}{p^l}\rceil$;

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 $s_n(A) = \tilde{s}_n$ with probability at least 1/3;

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

v) if $r = \lceil \log(\log(H)) + \log(\frac{1}{\epsilon})\rceil$, $2 \mid \beta$ and $\beta \geq 2$ 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 [15]. To prove (v) we slightly modify the proof of (iv) in the following manner. From (i) we notice that for every prime p dividing $s_n(A)$, the probability that it divides the missed part of $s_n(A)$ satisfies:

$$P\left(p \mid \frac{s_n(A)}{\tilde{s}_n}\right) \leq \left(\frac{1}{2}\right)^r.$$

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

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

□

Remark 2.3. Theorem 2.2 enables us to produce a LIF procedure, which gives an output $\tilde{s}(n)$ close to $s_n(A)$ with a time complexity $O(n^3(\log(n) + \log(\|A\|))^2)$ in the case of dense matrices (see (ii), notice that β is large). Then for the sparse case we can produce an algorithm with $O(n^2(\log(n) + \log(\|A\|))(n^{0.5} + \log(\|A\|)\log(\log(H)))$ complexity (see (iv)). In this case $\log(\log(H))$ repetitions are performed and $log(\beta)$ can be at most $O(n^{0.5})$ in order not to interfere in the complexity.

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 expected 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.

Then Saunders and Wan [23, 26] 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.

Also, Eberly, Giesbrecht and Villard have shown that for the dense case the expected number of non trivial invariant factors is small, namely less than $\lceil 3log_\lambda(n)\rceil + 29$ if the entries of the matrix are chosen in a set of λ consecutive integers [15]. As they also give a way to compute any $s_i(A)$, this leads to an algorithm with expected complexity $O(n^3(\log(n) + \log(\|A\|))^2 \log(n)\log_\lambda(n))$.

First, our analysis yields that the expected number of invariant factors for a random sparse matrix can be bounded by $O(\log(n))$ as well, while for a dense matrix the bound can be refined as $O(\log^{0.5}(n))$.

Second, our idea is to extend the method of Saunders and Wan to get the last invariant factors of A slightly faster than by [15]. Thanks to the adaptive approach it is possible to remove the $\log(n)$ factor form expected complexity of [15]. Moreover, we will show in the following sections that it enables us to build an adaptive algorithm solving a minimal number of systems.

Note that the analysis yields that it should be possible to change a $\log(n)$ factor in the expected complexity of [15] 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 $\log^\alpha(n)$ 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 [15] 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 $\lceil 3\log_\lambda(n)\rceil + 29$. In search for some sharpening of this result we prove the following theorems.

Theorem 3.1. *Let A be a dense matrix with entries chosen randomly and uniformly from a set of size $\lambda \geq 3$. Let p be a prime. The expected number of non-trivial invariant factors of A divisible by p is at most 3.*

Theorem 3.2. *Let A be a dense matrix with entries chosen randomly and uniformly from a set of size $\lambda \geq 3$. The expected number of non trivial invariant factors of A is at most $\left\lceil \sqrt{2\log_\lambda(n)} \right\rceil + 2$.*

Theorem 3.3. *Let A be a non-singular sparse matrix with Ω non-zero entries which are chosen randomly and uniformly from a set of size $\lambda \geq 3$. Then the expected number of nontrivial invariant factors of A is at most $\lceil \log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 2)\rceil + 7$.*

In order to prove the theorems stated above, we need first the following lemma.

Lemma 3.4. *For $\lambda \geq 12$, the sum over the primes p: $\sum_{8<p<\lambda} \left(\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil\right)^j$ can be bounded by $(\frac{1}{2})^j$.*

Proof. We will consider primes from the interval $\frac{\lambda}{2^{k+1}} \leq p < \frac{\lambda}{2^k}$, $k = 0, 1, \ldots \max\{\lceil\log(\lambda)\rceil - 3, 2\}$ separately. For the kth interval, $\lceil\frac{\lambda}{p}\rceil$ equals 2^{k+1}. In each interval there are at most $\lceil\frac{\lambda}{2^{k+2}}\rceil$ odd numbers and at most $\frac{\lambda}{2^{k+2}}$ primes: if in the interval there are more than 3 odd numbers, at least one of them is divisible by 3 and is therefore composite. For this to happen it is enough that $\lambda \geq 12$. We may therefore calculate:

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

□

Remark 3.5. For $\lambda = 2^l$, the sum over k can be made from 0 up to $l-2$, instead of $\lceil \log(\lambda) \rceil - 3$ and we can therefore include more primes in the sum: $\sum_{\mathbf{4}<p<2^l} \left(\frac{1}{2^l} \lceil \frac{2^l}{p} \rceil \right)^j \leq \left(\frac{1}{2}\right)^j$.

Proof. (Theorem 3.1) The main idea of the proof is similar to that of [15].

Let A be a random matrix with entries chosen uniformly and randomly from the set $\{0, 1, 2 \ldots \lambda - 1\}$. Let $MDep_i(p,k)$ denote the event that the first i columns of A mod p have rank at most $i-k$ over $\mathbf{Z}_p$. Surely, $MDep_k(p,k)$ means that the first k columns of A are 0 mod p, and thus the probability is $\left(\frac{1}{\lambda} \lceil \frac{\lambda}{p} \rceil \right)^{kn}$. Now, $\frac{1}{\lambda} \lceil \frac{\lambda}{p} \rceil$ denotes the probability that an entry of the matrix is determined modulo p and is equal to λ^{-1} if $p \geq \lambda$ or less than $\frac{2}{p+1}$ in the case $p < \lambda$. We are now going to find $P(MDep_i(p,k) \mid \neg MDep_{i-1}(p,k))$ for $i > k$. Since the event $MDep_{i-1}(p,k)$ did not occur, A_{i-1} has p-rank at least $(i-k)$ and at most $(i-1)$. For $MDep_{i,k}$ to occur it must be $(i-k)$. Thus there exists a set of $i-k$ rows L_{i-k} which is full rank mod p. Consider any row v_j that is left. As v_j is a combination of L_{i-k}, then say its first $i-k$ entries can be chosen randomly and its remaining k entries are determined mod p. More precisely, for $\lambda \geq p$ the probability that v_j is in the span of L_{i-k} is at most λ^{-k}. For $p < \lambda$ this probability is $(\frac{1}{\lambda} \lceil \frac{\lambda}{p} \rceil)^k$ which is always less than $(\frac{2}{p+1})^k$. As there are $n-i+k$ rows outside L_{i-k}, the probability that none of them is linearly independent of L_{i-k} over $\mathbf{Z}_p$ is at most $(\frac{2}{p+1})^{k(n-i+k)}$ for $p < \lambda$ and $(\frac{1}{\lambda})^{k(n-i+k)}$ for $p \geq \lambda$.

Now, since $P(MDep_i(p,k) \wedge \neg MDep_{i-1}(p,k)) \leq P(MDep_i(p,k) \mid \neg MDep_{i-1}(p,k))$, we can bound $P(MDep_i(p,k)) \leq P(MDep_k(p,k)) + P(\bigcup_{j=k+1}^{i} (MDep_j(p,k) \wedge \neg MDep_{j-1}(p,k)))$ by

$$P(MDep_i(p,k)) \leq \left(\frac{2}{p+1} \right)^{k(n-i+k)} \frac{(p+1)^k}{(p+1)^k - 2^k} \quad \text{for } p < \lambda$$

and

$$P(MDep_i(p,k)) \leq \left(\frac{1}{\lambda} \right)^{k(n-i+k)} \frac{\lambda^k}{\lambda^k - 1} \quad \text{for } p \geq \lambda.$$

Now, suppose that the number of invariant factors of A divisible by p is at least j (event $I_j(p)$). This implies that the first columns of A have rank at most $n-j$ mod p, or that $MDep_{n-j+k}(p,k)$ has occurred for all $k = 0..j$. This proves in particular that $P(I_j(p)) \leq MDep_n(p,j)$.

Thus, we have that the expected number of invariant factor divisible by

$p < \lambda$ satisfies:

$$\sum_{j=0}^{n} j\left(P(I_j(p)) - P(I_{j+1}(p))\right) = \sum_{j=1}^{n} P(I_j(p)) \leq \sum_{j=1}^{n} MDep_n(p,j) \leq \sum_{j=1}^{n} \left(\frac{2}{p+1}\right)^{jj} \frac{(p+1)^j}{(p+1)^j - 2^j}$$

The latter is decreasing in p and is therefore less than its value at $p = 2$, which is lower than 2.4. For $p \geq \lambda$ the result is even sharper:

$$\sum_{j=0}^{n} j\left(P(I_j(p)) - P(I_{j+1}(p))\right) \leq \sum_{j=0}^{n} \left(\frac{1}{\lambda}\right)^{jj} \frac{\lambda^j}{\lambda^j - 1} \leq \sum_{j=1}^{n} \left(\frac{1}{3}\right)^{jj} \frac{3}{2}$$

the latter being lower than 0.52. □

Proof. (Theorem 3.2) In addition to $MDep_i(p,k)$ introduced earlier, let Dep_i denote an event that the first i columns of A are linearly independent and $MDep_i(k)$, an event that either of $MDep_i(p,k)$ occurred. Recall from [15, §6] that $P(Dep_1 \vee MDep_1(k)) \leq \lambda^{-n}$, and $P(Dep_i \mid \neg(Dep_{i-1} \vee MDep_{i-1}(k))) \leq \lambda^{-n+i-1}$.

To bound $P(MDep_i(k) \mid \neg(Dep_{i-1}(k) \vee MDep_{i-1}(k)))$ we sum the results for all primes.

First, we bound this probability by

$$P\left(MDep_i(p,k) \mid \neg(Dep_{p,i-1}(k) \vee MDep_{i-1}(p,k))\right) \leq P(MDep_i(p,k) \mid \neg MDep_{i-1}(p,k))$$

and the latter has been shown to be $(\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^{k(n-i+k)}$ in the proof of theorem 3.1. Thus we have

$$\sum_{p<\lambda} P\left(MDep_i(p,k) \mid \neg(Dep_{p,i-1}(k) \vee MDep_{i-1}(p,k))\right) \leq \sum_{p<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^{k(n-i+k)}$$

and then we treat separately the smallest primes to get

$$\begin{aligned}\sum_{p<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^{k(n-i+k)} &\leq \left(\frac{2}{3}\right)^{k(n-i+k)} + \left(\frac{2}{4}\right)^{k(n-i+k)} + \left(\frac{2}{6}\right)^{k(n-i+k)} + \left(\frac{2}{8}\right)^{k(n-i+k)} + \sum_{8<p<\lambda} (\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)^{k} \\ &\leq \left(\frac{2}{3}\right)^{k(n-i+k)} + \left(\frac{1}{2}\right)^{k(n-i+k)} + \left(\frac{1}{3}\right)^{k(n-i+k)} + \left(\frac{1}{4}\right)^{k(n-i+k)} + \left(\frac{1}{2}\right)^{k(n-i+k)}\end{aligned}$$

thanks to lemma 3.4.

For primes $p \geq \lambda$ 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 $p \geq \lambda$ dividing the minor is at most $\frac{i-1}{2}(\log_\lambda(i-1)+2))$. Summarizing,

$$P((MDep_i(k) \wedge Dep_i) \mid \neg(Dep_{i-1} \vee MDep_{i-1}(k))) \leq \left(\frac{1}{\lambda}\right)^{n-i+1} + \left(\frac{2}{3}\right)^{k(n-i+k)} + \left(\frac{1}{2}\right)^{k(n-i+k)} + \left(\frac{1}{3}\right)^{k(n-i+k)} + \left(\frac{1}{4}\right)^{k(n-i+k)} + \left(\frac{1}{2}\right)^{k(n-i+k)} + \frac{i-1}{2}\left(\log_\lambda(i-1)+2\right)\left(\frac{1}{\lambda}\right)^{k(n-i+k)}$$

for $2 \leq i \leq k$.

Now, as in the previous proof, we use the union of the cases to get rid of the negative terms and the facts that $\frac{(p+1)^k}{(p+1)^k-2^k} \leq \frac{p+1}{p-1}$ and $\frac{\lambda^k}{\lambda^k-1} \leq \frac{\lambda}{\lambda-1}$ to get:

$$P(MDep_i(k)) \leq \lambda^{-n} + \frac{\lambda}{\lambda-1}\left(\frac{1}{\lambda}\right)^{n-i+1} + 3\left(\frac{2}{3}\right)^{k(n-i+k)} + 2\left(\frac{1}{2}\right)^{k(n-i+k)} + \frac{3}{2}\left(\frac{1}{3}\right)^{k(n-i+k)} + \frac{4}{3}\left(\frac{1}{4}\right)^{k(n-i+k)} + 2\left(\frac{1}{2}\right)^{k(n-i+k)} + \frac{i-1}{2}\left(\log_\lambda(i-1)+2\right)\left(\frac{1}{\lambda}\right)^{k(n-i+k)}\frac{\lambda}{\lambda-1}. \quad (1)$$

We can now compute the expected number of non trivial invariant factors.

Once again, the probability that the number of non trivial invariant factors is at least j is lower than $P(MDep_{n-j+k}(k))$ for $j > k$.

Now, let us fix $h = \left\lceil \sqrt{2\log_\lambda(n)} \right\rceil$. Then, we have that $h^2 \geq 2\log_\lambda(n) \geq \log_\lambda(n)+\log_\lambda(\log_\lambda(n)+2)$ since $\lambda, n \geq 3$. This gives also $\lambda^{h^2} > n(\log_\lambda(n)+2)$ and

$$1 > n\left(\log_\lambda(n)+2\right)\left(\frac{1}{\lambda}\right)^{h^2}\frac{\lambda^2}{2\lambda^h(\lambda-1)^2}.$$

Thus, the expected number of non trivial invariant factors is bounded by:

$$\sum_{j=1}^{h} 1 + \sum_{j=h+1}^{n} P(MDep_{n-j+h}(h))$$

which in turns is bounded by

$$h + \sum_{j=h+1}^{n} \Big(\lambda^{-n} + \frac{\lambda}{\lambda-1} \left(\frac{1}{\lambda}\right)^{j-h+1} + 3\left(\frac{2}{3}\right)^{jh} + 2\left(\frac{1}{2}\right)^{jh} + \frac{3}{2}\left(\frac{1}{3}\right)^{jh} + \frac{4}{3}\left(\frac{1}{4}\right)^{jh} + 2\left(\frac{1}{2}\right)^{jh} + \frac{n-j+h-1}{2}(\log_\lambda(n-j+h-1)+2)\left(\frac{1}{\lambda}\right)^{jh}\frac{\lambda}{\lambda-1}\Big)$$

$$\leq h + f(n,\lambda) + \frac{n}{2}(\log_\lambda(n)+2)\frac{\lambda}{\lambda-1}\sum_{j=h+1}^{n}\left(\frac{1}{\lambda}\right)^{jh} \leq \left\lceil \sqrt{2\log_\lambda(n)} \right\rceil + f(n,\lambda) + 1.$$

where $f(n,\lambda) \leq \frac{n-h}{\lambda^n} + \frac{1}{(\lambda-1)^2} + 3\left(\frac{2}{3}\right)^{h(h+1)} \frac{3^h}{3^h-2^h} + \frac{4}{2^{h^2}(2^h-1)} + \frac{3}{2\cdot 3^{h^2}(3^h-1)} + \frac{4}{3\cdot 4^{h^2}(4^h-1)} < 0.85$ as soon as $n \geq \lambda \geq 3$. □

Proof. (Theorem 3.3) We assume that the matrix A is a sparse $n \times n$ matrix, with Ω non-zero entries. The non-zero entries are chosen randomly and independently from a set $\{0, 1, \ldots \lambda - 1\}$. Furthermore, we assume, that A is nonsingular, i.e. in particular, in each row and column there is at least one non-zero entry, and also $P(Dep_i) = 0$ for all i.

Let U be a $n \times n$ matrix with all entries non-zero, $\det(U) = 1$. Then the Smith form of $B = AU$ is the same as the Smith form of A. The proof of the dense case in theorem 3.2 carries over the sparse case by choosing $h = 1$ on matrix B.

First, $MDep_i(p,1)$ means that all entries of the first column of B are 0 mod p. This translated to the entries of A gives a set of n linear equations with a total of Ω variables. The n sets of variables are distinct. This means that it suffices to take all but one entry randomly, and then fix the last one. This leads to the probability

$$P(MDep_i(p,1)) \leq \left(\frac{1}{\lambda}\left\lceil\frac{\lambda}{p}\right\rceil\right)^n.$$

While considering the case of $P(MDep_i(p,1) \mid \neg MDep_{i-1}(p,1))$ we first notice, that we can find a $(i-1) \times (i-1)$ non-zero minor, say the first $(i-1)$ rows of B. Then the first $(i-1)$ values in the remaining i-th column determine the constants such that the last column is a linear combination of the first $(i-1)$ columns. This again leads to the condition in a form of $(n-i+1)$ linear equations. By the same reasoning as above, the probability is

$$P(MDep_i(p,1) \mid \neg MDep_{i-1}(p,1)) \leq \left(\frac{1}{\lambda}\left\lceil\frac{\lambda}{p}\right\rceil\right)^{n-i+1}$$

Now, we separate the case where $j \geq \lceil \log_\lambda(n) + \log_\lambda(\log_\lambda(n)+2) \rceil = g$ and the lower values of j to get

$$\sum_{j=1}^{g} 1 + \sum_{j=g+1}^{n} \left(3\left(\frac{2}{3}\right)^j + 2\left(\frac{1}{2}\right)^j + \frac{3}{2}\left(\frac{1}{3}\right)^j + \frac{4}{3}\left(\frac{1}{4}\right)^j + 2\left(\frac{1}{2}\right)^j + \frac{n-j+1}{2}(\log_\lambda(n-j+1)+2) \right)$$

$$\leq g + f(n,\lambda) + \frac{n}{2}\frac{\lambda^2}{(\lambda-1)^2}\frac{1}{\lambda^{g+1}} \leq \lceil \log_\lambda(n) + \log_\lambda(\log_\lambda(n)+2) \rceil + f(n,\lambda) + \frac{1}{8}.$$

where $f(n,\lambda) \leq \frac{2^g}{3^{g-2}} + \frac{1}{2^{g-3}} + \frac{1}{4 \cdot 3^{g-2}} + \frac{1}{9 \cdot 4^{g-2}} \leq 6.49$ □

3.2 Extended Bonus Ideas

In his thesis [26], Z. Wan introduces the idea of computing the penultimate invariant factor (i.e. s_{n-1}) of A while computing s_n using two 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 in the following manner:

1. The (matrix) solution of $AX = B$, where B is a $n \times k$ multiple right hand side can be written as $\tilde{s}_n^{-1} N$ where $\tilde{s}_n$ approximates $s_n(A)$ and the minors of N give some divisors of the last k invariant factors of A: see lemma 3.6.

2. We are actually only interested in getting the product of these invariant factors. This information is in the gcd of all $k \times k$ minors of N. We show that by taking the gcd of only a small subset of these minors, only very few extra factors are introduced: see lemma 3.9.

3. Then we show that repeating this solving twice with two distinct right-hand sides B_1 and B_2 is in general sufficient to remove those extra factors and to get a very fine approximation of the actual product of the last k invariants: see lemma 3.10.

3.2.1 The last k invariant factors

Let X be a (matrix) rational solution of the equation $AX = B$, where $B = [b_i], i = 1, \ldots, k+1$ is a random $n \times (k+1)$ matrix. Then the coordinates of X have a common denominator $\tilde{s}_n$ and we let $N = [n_i], i = 1, \ldots, k+1$ denote the matrix of numerators of X. Thus, $X = \tilde{s}_n^{-1} N$ and $\gcd(N_{ij}, \tilde{s}_n) = 1$.

Following Wan, we notice that $s_n(A)A^{-1}$ is an integer matrix, the Smith form of which is equal to

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

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. The relation between A and N is as follows.

Lemma 3.6. *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 $\{0, 1, \ldots \beta - 1\}$. Then*

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

Proof. The Smith forms of $s_n(A)A^{-1}B$ and N are connected by the relation $\frac{s_n(A)}{\tilde{s}_n} s_i(N) = s_i(s_n(A)A^{-1}B)$, $i = 1, \ldots, (k+1)$. Therefore $\frac{s_n(A)}{s_{k+1}(s_n(A)A^{-1}B)} = \frac{\tilde{s}_n}{s_{k+1}(N)}$, and thus $\frac{\tilde{s}_n}{\gcd(s_{k+1}(N), \tilde{s}_n)}$ is an (integer) factor of $s_{n-k}(A)$. Moreover, the under-approximation is solely due to the choice of B and the over-estimation it can introduce in $s_n(A)A^{-1}B$ compared to $s_n(A)A^{-1}$. □

Remark 3.7. 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. Moreover, this allows us to consider $p|\tilde{s}_n$ in all probability consideration throughout the paper.

3.2.2 Complexity reduction: using only a small subset of minors and reintroducing a few undesired factors

In fact we are interested in computing the product $\pi_k = s_n s_{n-1} \cdots s_{n-k+1}(A)$ 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 that we have obtained. We can compute $\tilde{\pi}_k$ as $\tilde{s}_n^k / \gcd(s_1 s_2 \cdots s_k(N), \tilde{s}_n^k)$. The product of the first k invariant factors of a matrix is equal to the gcd of all its $k \times k$ minors. In our approach it suffices to compute $\lfloor n/k \rfloor$ 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.

Lemma 3.8. *For $n \times n$ matrix M 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* $\frac{3}{p^l}$.

Proof. To check whether $\mathrm{ord}_p(\det(A)) \geq l$ we will consider a process of diagonalization for M mod p^l as described in Algorithm *LRE* of [5]. It consists of diagonalization and reduction steps. At the i-th 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 $(n-i+1, n-i+1)$ submatrix 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 M with entries chosen uniformly and randomly from the set $\{0, 1 \dots S-1\}$, the probability that one entry is determined mod $p^i, i \leq l$, is less than $\beta_i(0) = \frac{1}{S}\lceil \frac{S}{p^i} \rceil$, from [26, Lemma 5.9].

Now we look at the $(n-m) \times (n-m)$ remaining submatrix $M^{(m)}$ after k reductions and m diagonalization steps. We need only to consider that $i \leq l - 2k$ since each reduction is performed on a matrix of order at least 2 and reduces the determinant by at least p^2. Moreover, we observe that, apart from the reductions, the entries of this new matrix can also be considered as uniformly distributed. To see this one can think e.g. of $a_{22} - \frac{a_{21}}{a_{11}}a_{12}$ with a_{11} and a_{12} fixed by the last triangularization step. Then one has one degree of freedom, say for a_{21} and then a_{22} is fixed. Then, the probability that the free entry is determined mod p^i is at most $\beta_i(k) = \frac{1}{N_k}\lceil \frac{N_k}{p^i} \rceil$, where $N_k = \left\lceil \frac{\vdots}{p} \right\rceil$ (the division is repeated k times). Moreover, since k is less than $\lceil l/2 \rceil - 1$ and $l \leq log_p(S)$, we have $N_k \geq \frac{S}{p^k} \geq \sqrt{S}$ and since $p^i < N_k$ we have

$$\beta_i(k) \leq \frac{N_k + p^i - 1}{pN_k} \leq \frac{2p^i}{p^i(p^i+1)} = \frac{2}{p^i+1}$$

throughout the diagonalization process. We therefore now set $\beta_i = \frac{2}{p^i+1}$ and use it as a bound for $\beta_i(k)$ in our calculations.

The proof is inductive. First, for $l = 1$, [26, Thm 5.13] gives

$$P(p \nmid \det(M)) \geq \prod_{i=1}^{n}(1 - \beta_1^i).$$

This transforms to

$$P(p \mid \det(M)) \leq \sum_{i=1}^{n} \beta_1^i \leq \frac{\beta_1}{1-\beta_1}. \tag{2}$$

Thus, the probability can be bounded by $\min(1, \frac{2}{p-1})$ and therefore by $\frac{3}{p}$. Then for $n = 2, l = 2$:

$$P(p^2 \mid \det(M)) \leq (1-P(p \mid a_{ij}\forall_{i,j}))\beta_2 + P(p \mid a_{ij}\forall_{i,j}) \leq \beta_2 + \beta_1^4 \leq \frac{2}{p^2+1} + \left(\frac{2}{p+1}\right)^4 \leq \frac{3}{p^2}.$$

Now we suppose inductively that $P(p^i \mid \det(M)) \leq \frac{3}{p^i}$ for all $i < l$. Then for $n = 2, 1 < l < n$ the induction gives that the probability is

$$\begin{aligned} P(p^l \mid \det(M)) &\leq (1 - P(p \mid a_{ij}\forall_{i,j}))\beta_l + P(p \mid a_{ij}\forall_{i,j})P(p^{l-2} \mid \det(M/p)) \\ &\leq \beta_l(k) + \beta_1(k)^4 P(p^{l-2} \mid \det(M/p)) \leq \frac{2}{p^l+1} + \left(\frac{2}{p+1}\right)^4 \frac{3}{p^{l-2}}. \end{aligned}$$

Notice that we sum over all possible diagonalization/reduction steps combinations. The latter is less than $\frac{3}{p^l}$ when

$$\beta_1(k)^4 3p^2 \leq 1. \tag{3}$$

With $\beta_1(k) \leq \frac{2}{p+1}$ this means that $\frac{48}{(1+1/p)^2(p+1)^2} = \frac{48}{(p+2+1/p)^2} \leq 1$ which is fulfilled for $p > 3$. For primes $p = 2, 3$ we have to use a sharper bound for $\beta_l(k)$. Since $p^l < N_k$ and $l > 2$ we have $\beta_1(k) \leq \frac{p^l+1+p-1}{(p^l+1)p} \leq \frac{p^{l-1}+1}{p^l+1} < \frac{p+1}{p^2+1}$. This allows us to prove the inequality (3) for $p = 3$ since $(\frac{2}{5})^4 27 < 0.7$. For $p = 2$ and $l > 3$ also $(\frac{9}{17})^4 12 < 0.95$. For the remaining case $p = 2$, $l = 3$ we can bound $P(p \mid \det(M/p))$ by 1 instead of $\frac{3}{2}$ and then one can prove that $\beta_3(k) + \beta_1(k)^4 \leq \frac{2}{9} + (\frac{5}{9})^4 < 0.32 \leq \frac{3}{8}$.

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

$$\begin{aligned}P(p^l \mid \det(M)) &\leq \sum_{i=l}^{n}(1-P(p \mid a_{ij}\forall_{i,j\leq n}))\dots(1-P(p \mid a_{ij}^{(n-i+1)}\forall_{i,j\leq i+1}))\beta_1(k)^{i^2} \\ &+ \sum_{i=2}^{l-1}(1-P(p \mid a_{ij}\forall_{i,j\leq n}))\dots(1-P(p \mid a_{ij}^{(n-i+1)}\forall_{i,j\leq i+1}))\beta_1(k)^{i^2}P(p^{l-i} \mid \det(M^{(n-i} \\ &+ (1-P(p \mid a_{ij}\forall_{i,j\leq n}))\dots(1-P(p \mid a_{ij}^{(n-1)}\forall_{i,j\leq 2}))\beta_l(k) \\ &\leq \sum_{i=l}^{n}\beta_1(k)^{i^2} + \sum_{i=2}^{l-1}\beta_1(k)^{i^2}P(p^{l-i} \mid \det(M^{(i)}/p)) + \beta_l(k)\end{aligned} \tag{4}$$

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

$$\begin{aligned}P(p^l \mid det(M)) &\leq \sum_{i=2}^{n}(1-P(p \mid a_{ij}\forall_{i,j\leq n}))\dots(1-P(p \mid a_{ij}^{n-i+1}\forall_{i,j\leq i+1}))\beta_1(k)^{i^2}P(p^{l-i} \mid \det(M^{(n-i)} \\ &+ (1-P(p \mid a_{ij}^{n-i+1}\forall_{i,j\leq n}))\dots(1-P(p \mid a_{ij}^{n-1}\forall_{i,j\leq 2}))\beta_l(k) \\ &\leq \sum_{i=2}^{n}\beta_1(k)^{i^2}P(p^{l-i} \mid \det(M^{(i)}/p)) + \beta_l(k).\end{aligned} \tag{5}$$

Again, we can use the induction to get the bound $P\left(p^{l-i} \mid \det(M^{(n-i)}/p)\right) \leq \frac{3}{p^{l-i}}$. Then, we can then bound both sums by

$$P(p^l \mid \det(M)) \leq \sum_{i=2}^{\infty}\beta_1(k)^{i^2}\frac{3}{p^{l-i}} + \beta_l(k) \leq \frac{3\beta_1(k)^4}{p^{l-2}}\frac{1}{(1-\beta_1(k)^5 p)} + \beta_l(k). \tag{6}$$

To prove the inequality $P(p^l \mid \det(M)) \leq \frac{3}{p^l}$, again, we have to consider several cases. For $p > 3$ we use the bound β_1 and β_l for $\beta_1(k)$ and $\beta_l(k)$ respectively. Then we have

$$\begin{aligned}&\frac{3\cdot 2^4 p^2(p+1)}{p^l((p+1)^5 - p2^5)} + \frac{2}{p^l+1} < \frac{2}{p^l} + \frac{1}{p^l}\frac{48p^2(p+1)}{(p+1)^5-(p+1)2^4} < \frac{2}{p^l} + \frac{1}{p^l}\frac{48p^2}{(p+1)^4-2^4} \\ &< \frac{2}{p^l} + \frac{1}{p^l}\frac{48p^2}{25p^2+4\cdot 5p^2+6\cdot p^2+4\cdot 5+1-16} < \frac{2}{p^l} + \frac{1}{p^l}\frac{48p^2}{51p^2} < \frac{3}{p^l}.\end{aligned}$$

For $p = 3$ it can be explicitly checked that $P(p^l \mid \det(M)) < \frac{3}{p^l}$ using the bound $\frac{p+1}{p^2+1}$ for $\beta_1(k)$ (notice, that $l > 1$). In this case we get $\frac{1}{3^l}\frac{3(\frac{2}{5})^4 3^2}{(1-(3\frac{2}{5})^5)} + \frac{2}{3^l} < \frac{1}{3^l}2.75$.

For $p = 2$ we have to consider $2^2, 2^3, 2^4$ and 2^l for $l > 4$ separately. Let us rewrite (4) and (5) in this cases.

- $l = 2$:

$$P(2^2 \mid \det(M)) \leq \sum_{i=2}^{n} \beta_1(k)^{i^2} + \beta_2(k) \leq \beta_1(k)^4 \frac{1}{1-\beta_1(k)^5} + \beta_2(k).$$

  As $\beta_1(k) \leq \frac{2+1}{4+1}$ we have $0.65 < 0.75$.

- $l = 3$:

$$\begin{aligned} P(2^3 \mid \det(M)) &\leq \sum_{i=3}^{n} \beta_1(k)^{i^2} + \beta_1(k)^4 P(2 \mid \det(M^{(n-2)}/p)) + \beta_3(k) \\ &\leq \beta_1(k)^9 \frac{1}{1-\beta_1(k)^7} + \beta_1(k)^4 P(2 \mid \det(M^{(n-2)}/p)) + \beta_3(k). \end{aligned}$$

  As $\beta_1(k) \leq \frac{4+1}{8+1}$ we have $0.33 < 0.375$.

- $l = 4$:

$$\begin{aligned} P(2^4 \mid \det(M)) &\leq \sum_{i=4}^{n} \beta_1(k)^{i^2} + \beta_1(k)^9 P(2 \mid \det(M^{(n-3)}/p)) + \beta_1(k)^4 P(2^2 \mid \det(M^{(n-2)}/p)) + \\ &\leq \beta_1(k)^{16} \frac{1}{1-\beta_1(k)^9} \beta_1(k)^9 P(2 \mid \det(M^{(n-3)}/p)) + \beta_1(k)^4 P(2^2 \mid \det(M^{(n-2)}/p)) + \beta_4(k) \end{aligned}$$

  As $\beta_1(k) \leq \frac{8+1}{16+1}$ we have $0.18 < 0.1875$.

- $l > 4$:We use inequality (6) with $\beta_1(k)$ bounded by $\frac{p^4+1}{p^5+1}$. Therefore we get $P(2^l \mid \det(M)) < \frac{1}{2^l}\left(\frac{3(2^4+1)^4 2^2}{(2^5+1)^4((2^5+1)^5)-2(2^4+1)} + 2\right) < 2.92\frac{1}{2^l} < \frac{3}{2^l}$.

□

We now discuss the impact of choosing only a few minors in the computation of $s_1 \dots s_{k+1}(N)$. Here, $\mathrm{ord}_p(x)$ denotes the highest power of p dividing x.

Lemma 3.9. *Let B,N and $\tilde{s}_n$ be as defined in lemma 3.6. Suppose that B is a random matrix with entries chosen uniformly from the set $\{0, 1, \dots S-1\}$ and $k = O(\log(n))$. Let $N = [N_1 | \dots N_\mu | N']^T$ where N_i are $(k+1) \times (k+1)$*

matrices, $\mu = \max\{\lfloor n/(k+1)\rfloor; 2\}$. *Then the size of the over-approximation due to the partial* gcd *computations is bounded as follows:*

$$\sum_{l=1}^{\infty}\sum_{p^l|\tilde{s}_n} P\left(\mathrm{ord}_p\left(\frac{\gcd_{i=1\ldots\mu}(\det(N_i))}{\gcd(minors(N))}\right)=l\right) l\log(p) \;\in\; O(1)+\log^2(H)\left(\frac{3}{\sqrt{S}}\right)^{\mu}, \tag{7}$$

where H *is a bound for* $\tilde{s}_n$.

Proof. First, we notice, that by the formula $N = \tilde{s}_n A^{-1} B$ the distribution of the entries of N is linked to that of a random matrix B. Therefore, we may consider that an entry of N is determined modulo p^l with the same probability as an entry of B i.e. $\frac{1}{S}\left\lceil\frac{S}{p^l}\right\rceil$ and that lemma 3.8 holds for the minors of N.

Now, in the sum (7) we notice that

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

Let us first consider primes p such that $p^l \leq S$. Therefore from Lemma 3.8 we get that

$$P(p^l|\det(N_i)) \leq \frac{3}{p^l}.$$

Then (7) can be bounded by

$$\sum_{l=1}^{\infty}\sum_{p^l|\tilde{s}_n, p^l\leq S} P\Big(\mathrm{ord}_p(\frac{\gcd_{i=1\ldots\mu}(\det(N_i))}{\gcd(minors(N))})=l\Big) l\log(p) \leq \sum_{l=1}^{\infty}\sum_{p^l|\tilde{s}_n, p^l\leq S} P\Big(p^l|\gcd_{i=1\ldots\mu}(\det(N_i))\Big)\log(p)$$

$$\leq \log 2(2+\sum_{l=3}^{\infty}(\frac{3}{2^l})^{\mu}) + \log 3(1+\sum_{l=2}^{\infty}(\frac{3}{3^l})^{\mu}) + \log 5(1+\sum_{l=2}^{\infty}(\frac{3}{5^l})^{\mu}) + \sum_{5<p^l<S}\sum_{l=1}^{\infty}\log(p)(\frac{3}{p^l})^{\mu}$$

$$\leq 2.19 + 1.78 + 2.36 + \sum_{5<p^l<S} 3^{\mu}\frac{\log(p)}{p^{\mu}-1} \leq 6.33 + 3^{\mu}\int_6^{\infty}\frac{\log(x)}{x^{\mu}-1}dx \leq 6.33 + 6.09$$

For $p > S$, following Eq. (2) we get $P(p|\det(M)) \leq \sum_{i=1}^{n}\beta_1^i \leq \frac{\beta_1}{1-\beta_1}$, where β_1 as defined in the proof of lemma 3.8 is $\frac{1}{S}\left\lceil\frac{S}{p}\right\rceil$. For $p \geq S$ this gives $\beta_1 = \frac{1}{S}$ and the probability can be bounded by $\frac{1}{S-1}$. Suppose now that $p < S$ but $p^l \geq S$. Then the probability $P(p^l|\det(M))$ is less than $P(p^{\lfloor\log_p(S)\rfloor}|\det(M))$ and consequently can be bounded by $3\min(\frac{1}{p}, \frac{p}{S})$ which is less than $\frac{3}{\sqrt{S}}$.

Thus, the sum over $p^l > S$ can be bounded by

$$\sum_{l=1}^{\infty} \sum_{p^l|\tilde{s}_n, p^l>S} P\Big(p^l | \gcd_{i=1\ldots\mu}(\det(N_i))\Big) \log(p) \leq \sum_{p|\tilde{s}_n} \sum_{l=\lceil \log_p(S)\rceil}^{\lfloor \log_p(H)\rfloor} \left(\frac{3}{\sqrt{S}}\right)^{\mu} \log(p)$$

$$\leq \sum_{p|\tilde{s}_n} \log_p(H)\log(p)\left(\frac{3}{\sqrt{S}}\right)^{\mu} \leq \sum_{p|\tilde{s}_n} \log(H)\left(\frac{3}{\sqrt{S}}\right)^{\mu} \leq \log^2(H)\left(\frac{3}{\sqrt{S}}\right)^{\mu},$$

where H is a bound for $\tilde{s}_n$ (the Hadamard's bound for A).

With μ being about $\frac{n}{O(\log(n))}$, the expected size of overestimation due to partial gcd calculation is $O(1) + \log^2(H)\left(\frac{3}{\sqrt{S}}\right)^{\mu}$.

□

3.2.3 Removing the undesired factors

In the preceding section, we saw that taking only a subset of the possible minors of N potentially introduces extra factors. In lemma 3.6 we have mentioned that the choice of B can also introduce some over-estimation. We now show that repeating the solving twice with two distinct random right-hand sides B_1 and B_2 is in general sufficient to remove those extra factors.

Lemma 3.10. *Let A be an $n \times n$ integer matrix and B_i, $i = 1, 2$ be $n \times k$ matrices with the entries uniformly and randomly chosen from the set $\{0, 1, \ldots S-1\}$. Then for $M = s_n(A)A^{-1}$*

$$\log\left(\frac{\pi_k(A)}{s_n(A)^k}\gcd\left(s_2\cdots s_k(MB_1), s_2\cdots s_k(MB_2), s_n(A)^k\right)\right) \in O(1) + \frac{k\log^2(H)}{S}.$$

Proof. First, notice that $\pi_k(A) = \frac{s_n(A)^k}{s_1\ldots s_k(M)}$. Therefore

$$\log\left(\frac{\pi_k(A)}{s_n(A)^k}\gcd(s_1\cdots s_k(MB_1), s_1\cdots s_k(MB_2))\right) = \log\left(\frac{\gcd(s_1\cdots s_k(MB_1), s_1\cdots s_k(MB_2), s_n(A}{s_1\cdots s_k(M)}\right.$$

The latter is less than

$$\sum_{p|s_n(A)} \log(p) l P(\mathrm{ord}_p\left(\frac{\gcd\left(s_1\cdots s_k(MB_1), s_1\cdots s_k(MB_2), s_n(A)^k\right)}{s_1\cdots s_k(M)} = l\right) =$$

$$\sum_{p|s_n(A)} \log(p) P(\mathrm{ord}_p\left(\frac{\gcd(s_1\cdots s_k(MB_1), s_1\cdots s_k(MB_2), s_n(A)^k}{s_1\cdots s_k(M)}) \geq l\right)$$

Now, [26, Lem. 5.17] states that for every matrix M there exist a full rank $k \times n$, $k \leq n$, matrix V, such that $\mathrm{ord}_p(\frac{s_1 \cdots s_k(MB)}{s_1 \cdots s_k(M)})$ is less or equal $\mathrm{ord}_p(\det(VB))$. Using this, we can then link the previous probability to the probability that p^l divides the determinant of $V_i B_i$. The entries of the latter can be treated as randomly distributed with the same distribution as B_i. Moreover, we only have to consider $p|s_n(A)$.

For $p^l < S$ Lemma 3.8 gives $P(B_i : \mathrm{ord}_p(\frac{s_1 \cdots s_k(MB_i)}{s_1 \cdots s_k(M)}) \geq l) = P(B_i : \mathrm{ord}_p(\det(V_i B_i)) \geq l) \leq \frac{3}{p^l}$,.

Now the expected size of the under-estimation is less than or equal to

$$\log(2)(1 + \sum_{l=2}^{\infty}(\frac{3}{2^l})^2) + \log(3)\sum_{l=1}^{\infty}(\frac{3}{3^l})^2 + \log(5)\sum_{l=1}^{\infty}(\frac{3}{5^l})^2 + \sum_{5<p\leq H}\sum_{l=1}^{\infty}\log(p)(\frac{3}{p^l})^2$$
$$\leq 1.75 + 1.79 + 0.88 + \int_6^{\infty}\log(x)\frac{9}{x^2-1}dx,$$

which is $O(1)$.

For $p^l \geq S$ the expected size of underestimation is

$$\sum_{p|s_n(A)}\sum_{l=\lceil \log_p(S)\rceil}^{\log_p(H^k)} l\log(p)(\frac{3}{\sqrt{S}})^2 \leq \frac{k16\log^2(H)}{S},$$

which gives the result. □

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 [15].

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 (here we use timings) while processing a given instance of the problem. This kind of auto-adaptation is called introspective in [12]. In the following, CRA loop refers to algorithm 2.1, slightly modified to compute $\det(A)/K$. If we re-run the CRA loop, we use the already computed modular determinants first whenever possible.

Informally, the general idea of the introspective scheme is:

1. Initialize the already computed factor K of the determinant to 1;
2. Run fast FFLAS LU routines (or the Wiedemann's algorithm in the case of a sparse matrix) in the background to get several modular determinants $d_i = det(A) \mod p_i$.
3. From time to time try to early terminate the Chinese remainder reconstruction of $det(A)/K$.
4. In parallel or in sequential, solve random systems to get the last invariant factors one after the other.
5. Update K using the extended bonus ideas.
6. Loop back to step (2) until an early termination occurs or until the overall timing shows that the expected complexity is exceeded.
7. In the latter exceptional case, switch to a better worst case complexity algorithm.

More precisely, the full algorithm in shown on page 23.

4.1 Introspectiveness: dynamic choice of the thresholds

The introspective behavior of algorithm 4.1 depends paramountly on the number of system solvings and on the size of the random entries.

The parameter i_{max} controls the maximal total number of system solvings authorized before switching to a best worst-case complexity algorithm. The choice of i_{max} has to be discussed in terms of the matrix type and expected number of invariant factors estimated for A.

First, depending on the size of the set from which we are sampling the random right-hand sides, a minimum number of solvings is required to get a good probability of correctness. We thus define this to be i_{min}.

In the dense case, the (ii) part of theorem 2.2 states that $i_{min} = 2$ is sufficient.

In the sparse case, the (iv) part of theorem 2.2 has to be used and therefore a number of repetitions $i_{min} = \lceil 2\log(\log(H)) \rceil$ is required.

Then this number i_{min} has also to be augmented if the expected number of non trivial invariant factors is high. We thus set

$$i_{max} = \max(i_{min}, \mathbf{E}(\#factors(A))).$$

In the dense case $\mathbf{E}(\#factors(A))$ is less than $\lceil \sqrt{2\log_\lambda(n)} \rceil + 2$ as shown in theorem 3.2 ; in the sparse case, theorem 3.3 ensures that the expected

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 $\{0, 1 \ldots, \beta\}$
Require: 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
$i_{max} = i_{max}(A), i_{min} = i_{min}(A)$; //see §4.1
$\tilde{\pi}_0 = 1; K = 1$;
$k_{done} = 0; k_{app} = 0; j = 0$;
while $k_{done} \leq i_{max}$ **do**
  $i = k_{done} + 1$;
  while $i \leq i_{max}$ **do**
    Generate $b_i^{(j)}$ a random vector of dimension n from the stream S;
    Compute $\tilde{s}_n$ by solving $Ax_i^{(j)} = b_i^{(j)}$; //see Section 2.2
    if $i = 1$ **then** $i = 2; \tilde{\pi}_1 = \tilde{s}_n$;
    else
      $N := \tilde{s}_n X$, where $X = [x_l^{(j)}]_{l=0,\ldots i}$; //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
    $K = \mathrm{lcm}(\tilde{\pi}_i, K)$; $\tilde{\pi}_i = K$;
    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 $i > i_{min}$ **then**
      if $\tilde{\pi}_i = \tilde{\pi}_{i-1}$ **then**
        if $i > k_{app}$ **then**
          $k_{done} = k_{app}; k_{app} = i; j = j + 1 \mod 2$; break;
        else
          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 if
    end if
  end while
end while
run an asymptotically better integer determinant algorithm;

number of non trivial invariant factors is less than $\lceil \log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 2) \rceil + 7$.

Now, random vectors are randomly sampled a set of size β. For a dense matrix A we can take $\beta = \lceil (n+1)H \rceil$ to get a good probability of success together with a fast execution, as shown in theorem 2.2(ii) and lemma 3.10. For a sparse matrix we should take $\beta > 9$, but we are restricted to $\log(\beta) \in O(n^{0.5})$. The choice $\beta = \max(9, 2^{\sqrt{n}})$ is sufficient to keep a small asymptotic cost of the system solvings while still remaining in the $O(1)$ estimations of lemmata 3.9 and 3.10. Then the good probability of success is preserved to the cost of a small increase (part (iv) of theorem 2.2) in the number of solvings.

Additionally, (see lemma 3.10) we should ensure that π_k is computed twice using different matrices B. We therefore introduce the variables k_{done} and k_{app} which store respectively the number of factors computed at least twice (up to $O(1)$)) or once (thus only approximated).

4.2 Correctness and complexity

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 [18, 25] ensures that a $1 - \epsilon$ probability is obtained. □

The following theorem gives the complexity of the algorithm.

Theorem 4.2. *The expected complexity of Algorithm 4.1 in the case of a dense matrix is*

$$O^{\approx}\left(n^\omega \log(1/\epsilon) + n^3(\log n + \log(\|A\|))^2 \log^{0.5}(n)\right)$$

where $O^{\approx}$ *hides some* $\log(\log(n))$ *factors. For a sparse matrix* A *we get*

$$O^{\approx}\left(n\Omega \log(1/\epsilon) + n^2(\log n + \log(\|A\|))(n^{0.5} + \log(\|A\|)) \log(n)\right).$$

The pessimistic complexity depends on the algorithm used in the last step.

In the case of a sparse matrix, the space complexity is $O(n^2 \log^2(n))$.

Proof. To analyze the complexity of the algorithm we would consider the complexity of each step for the dense and sparse case respectively.

For a dense matrix A, with k defined as in line 1, the complexity of the initial CRA iterations is $O\left(n^{\omega}\log(1/\epsilon)\right)$. The while loop is constructed in a way that we perform at most $2i_{max}$ iterations (see subsection 4.1 for the bound on i_{max}) with $\log(\|B\|) = O(n\log(n))$. Therefore the cost is $O\left(n^3(\log(n)+\log(\|A\|))^2\sqrt{\log(n)}\right)$. Considering the time limit, this is also the time of all CRA loop iterations. Now, for the computation of the $\lfloor\frac{n}{i}\rfloor$ determinants $det(N_l)$, an exact integer determinant algorithm for dense matrices has to be used. Here, as the matrices are small, a simple Chinese remaindering of LU's is sufficient. Its overall cost is $O\Big(\lfloor n/i\rfloor i^4(\log(i)+ n(\log(n)+\log(\|A\|)+\log(\|B\|))\Big)$ bit operations, which for $i=2,\ldots,i_{max}$ with i_{max} being $O(\log(n))$ is $O^{\sim}(n^2)$ and is thus negligible.

For a sparse matrix A, we use the Wiedemann's algorithm in the CRA iterations, thus, the complexity of the first step is $O(\Omega n\log(1/\epsilon))$. With a $O^{\approx}(\log(n))$ bound for i_{max}, the while loop will cost at most $n^{1.5}(\log(n)+\log(\|A\|)\log(n)\Omega + O(n^2(\log(n)+\log(\|A\|))(n^{0.5}+\log(\|A\|))\log(n))$. The cost of computing $\tilde{\pi}_i$ is again 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 and Lemma 3.10), thus it is possible to recover it by several CRA loop iterations.

The space complexity in the case of a sparse matrix A is connected with the cost of storing matrix N. It is a $n\times\log(n)$ matrix with entries of size $n\log(n)$. Thus, the space complexity is $n^2\log^2(n)$. □

In the last step, for a sparse matrix, a modified algorithm of Eberly, Saunders and Villard [15] can be used instead as an already large number of non-trivial invariant factors have been found. Thus, in practice their binary search has many chances to be the most efficient. For a dense matrix we propose e.g. the best worst case $O^{\sim}(n^{3.2}\log(\|A\|))$ algorithm of Kaltofen [19] and refer to [18] for a survey on the complexity of determinant algorithms.

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 1 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

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.

s_{n-i} decreases whilst the cost of its computation increases. This accounts for better performances of Abbott's algorithm, which computes only s_n. For bigger n calculating s_{n-1} starts to pay out. The same pattern repeats in further iterations.

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 based on CRA only 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 an even more appropriate moment. This can be done by changing the condition in line 27 ($\tilde{\pi}_i = \tilde{\pi}_{i-1}$) to

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

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 section 3.1 and [15].

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 a way that minimizes the number of system solvings 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.

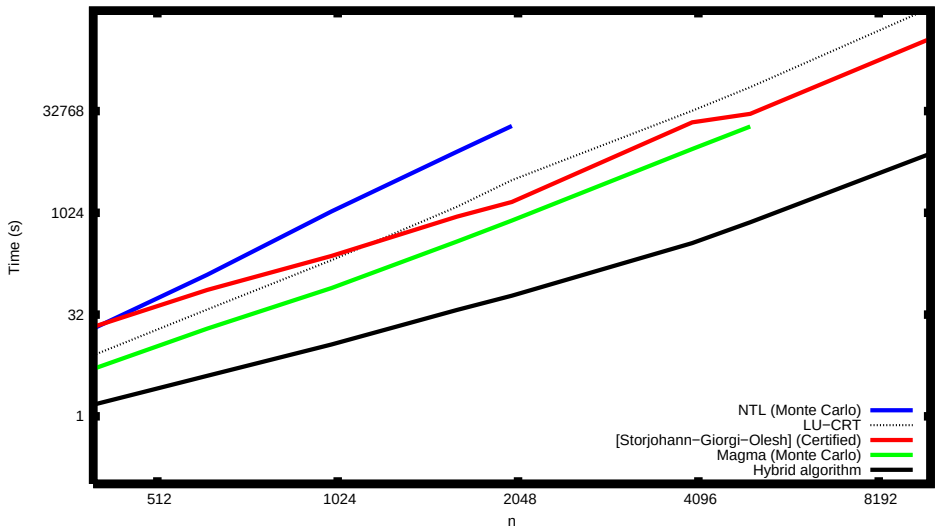

Figure 1: Comparison of our algorithm with other existing implementation. Tested on random dense matrices of the order 400 to 10000, with entries {-8,-7,. . . ,7,8} Using fast modular routines puts our algorithm several times ahead of the others. Scaling is logarithmic.

Indeed for random dense 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 1 and 2 where comparison of timings for different algorithms are presented.

Figure 3 gives a comparison of the performance of sparse and dense variants of our algorithm. For the sparse solver of [16] we used the best blocking on our machine (blocks size ranging from 25 for the 200×200 matrix to 500 for the 20000×20000 matrix were used). The figure shows that using the dense algorithm outperforms the sparse solver on our matrix set/machine architecture by a factor of 3.3 to 2.3. Still, Thanks to the space-

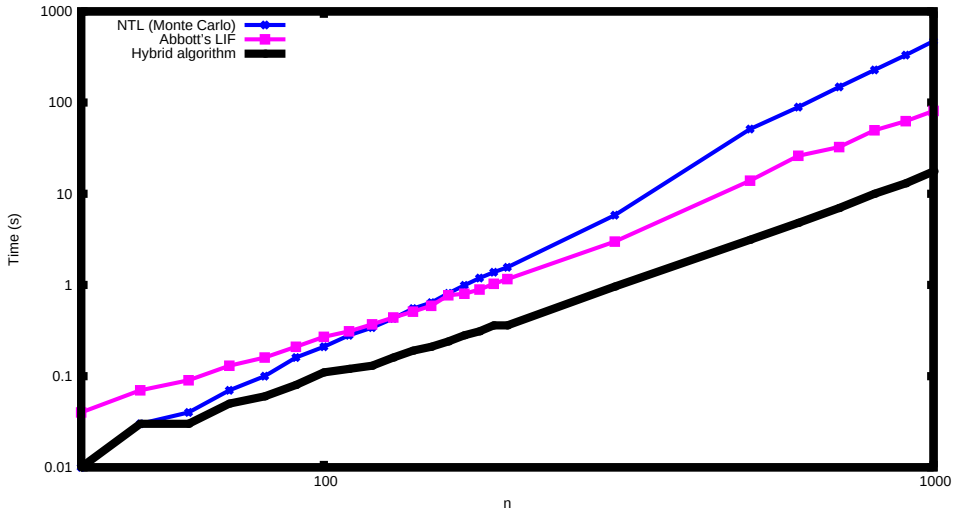

Figure 2: Comparison of our algorithm with early terminated Chinese remaindering algorithm (LU) and the algorithm of Abbott *et al.* [1] (LIF). Tested on random dense matrices of the order 40 to 1000, with entries {-100,-99,. . . ,99,100}. When matrix size exceeds 80 the adaptive algorithm wins. Scaling is logarithmic.

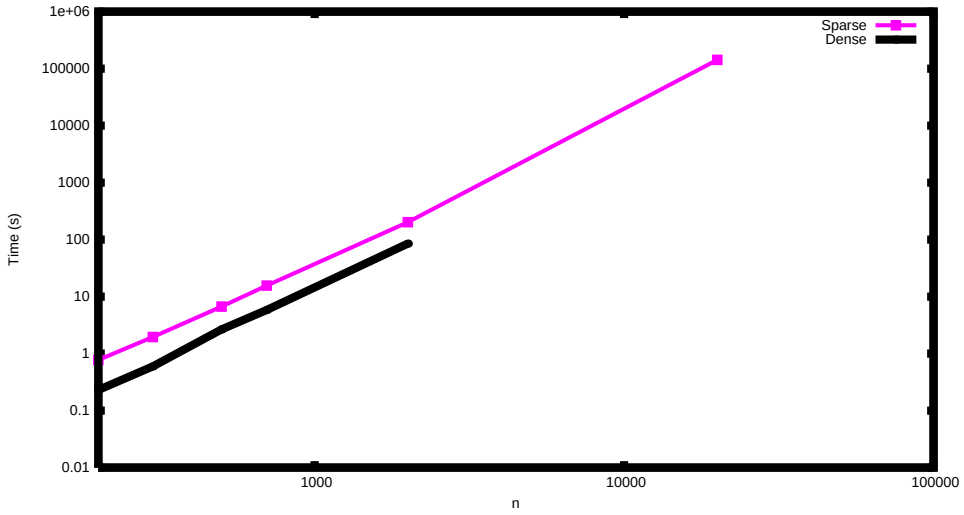

Figure 3: Comparison of sparse and dense variants of our determinant algorithm. Scaling is logarithmic.

efficiency of the sparse algorithm we are able to compute determinants of 20000×20000 matrix for which the dense solver thrashes.

6 Conclusions

In this paper we have presented an algorithm computing the determinant of an integer matrix which can have dense or sparse variants. The expected time complexity depends on the cost of the system solving procedure used and the expected number of invariant factors. This gives $O\left(n^3(\log(n) + \log(\|A\|))^2 \log^{0.5}(n)\right)$ for the dense case and $O\left(n^2(\log(n) + \log(\|A\|))(n^{0.5} + \log(\|A\|)) \log(n)\right)$ for the sparse case when the cost of matrix-vector product is $O(n)$. Our algorithm uses an introspective approach so that its actual expected complexity is only $O\left(n^\alpha(\log(n) + \log(\|A\|))^\beta k\right)$ if the number k of invariant factors is smaller than *a priori* expected but greater than i_{min}; $\alpha = 3$ or 2.5, $\beta = 2, 1$ depending on the matrix type. The actual running time can be even smaller, assuming that any under-estimation resulting from probabilistically correct procedures can be compensated sooner than 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 rou-

tines, 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 [3]. A modification to be tested, is to try to reconstruct s_n with only some entries of the solution vector $x = \mathbf{n}/d$. Parallelization is also to be considered. There, 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)*, pp. 197-204, ACM Press, 1999.

[2] L. Chen, W. Eberly, E. Kaltofen, B.D. Saunders, W.J. Turner, G. Villard. Efficient matrix preconditioners for black box linear algebra. In *Linear Algebra and Applications*, pp. 343–344. 2002.

[3] 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)*, pp. 92–99, ACM Press, 2005.

[4] J. Dixon. Exact Solution of Linear Equations Using P-Adic Expansions. In *Numer.Math.* 40(1), pp. 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), pp. 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, G. Villard. Computing the rank of large sparse matrices over finite fields. *CASC'2002 Computer Algebra in Scientific Computing*. 2002.

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

[11] J.G. Dumas, C. Pernet, Zhendong Wan. Efficient Computation of the Characteristic Polynomial. *ISSAC'2005*, pp. 181-188. 2005.

[12] 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*. 2006.

[13] J.G. Dumas, A. Urban'ska. An introspective algorithm for the integer determinant. In: *Proceedings of Transgressive Computing 2006, Granada, España*. 2006.

[14] W. Eberly, E. Kaltofen. On randomized Lanczos algorithms. *ISSAC'1997*. 1997

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

[16] W. Eberly, M.Giesbrecht, P. Giorgi, A. Storjohann, G. Villard. Solving Sparse Integer Linear Systems. *ISSAC'2006*. 2006.

[17] O.H. Ibarra, S. Moran, R.Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45ŏ201356, Mar.1982.

[18] 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), pp. 133-146. 2004.

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

[20] T. Mulders, A. Storjohann. Diophantine Linear System Solving. *ISAAC'1999*, pp. 181-188. 1999.

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

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

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

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

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

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

[27] D. Wiedemann. Solving sparse linear equations over Finite Fields. In *IEEE Trans. Inf. Theory*, pp. 54-62. 1986.

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}`